



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

**DOTTORATO DI RICERCA IN
DATA SCIENCE AND COMPUTATION**

CICLO XXXVI

Settore Concorsuale: 02/A1

Settore Scientifico Disciplinare: FIS/01

Deep Learning for New Physics Search

Presentata da:

Luca Anzalone

Coordinatore Dottorato

Prof. Daniele Bonacorsi

Supervisore

Prof. Daniele Bonacorsi

Esame finale anno 2025

Abstract

The ultimate goal of high-energy physics is searching for novel elementary particles and fundamental interactions, which may enable the formulation of a more comprehensive physics Beyond the Standard Model (BSM) able to explain a wider range of phenomena compared to the actual Standard Model (SM) of particle physics. Physics analyses are numerous and complex; so far, no theory tested against the SM has proven to be the right one. There could be various reasons for this: the detector apparatus of particle colliders, which contains tons of sensors that react to particles, is not able to reveal the new physics; the trigger system, which is responsible for the first selection of the huge volume of data produced by the collision of particles, discards the almost totality of “interesting”, i.e., potentially new physics, events; lastly, the space of plausible alternative hypotheses to the Standard Model is so vast that we may be looking in the wrong directions. Assuming the detector is sufficiently expressive to capture the potential new physics, this thesis describes and provides novel data-driven tools, powered by deep learning, to improve various aspects of particle physics analyses resulting in better sensitivity to the new physics. In particular, the signal sensitivity can be increased either *directly* (e.g., better classifiers), *indirectly* (e.g., improved physics pipelines), or *jointly* (i.e., enhancing the main task results in better downstream models.)

Parameter-conditional neural networks are deep learning models that incorporate domain knowledge at various levels, for example by conditioning the network with one or more physics parameters allowing a single model to handle the whole parameter range, also thanks to the interpolation capability. This kind of neural network is effective at enhancing the separation of the background from the signal in a model-dependent search scenario, also reducing the complexity of the analysis since only a single model is necessary to test for multiple values of the parameter, which can be the searched particle’s mass for example.

If we consider events from known physics the norm, unexpected, out-of-distribution, rare, and surprising events can be regarded as anomalies from which novel particles and interactions may originate. In this case, the problem of searching for new physics can be seen as an *anomaly detection* task. Auto-encoders are popular anomaly detection models, and this thesis presents two variations of them aiming at both improving the triggering system of the particle detector and, as well as, increasing the sensitivity towards a broad range of signals given the known physics targeting a model-independent search.

Problems like accelerator tuning, data quality monitoring, and particle reconstruction indirectly affect the sensitivity of particle physics analyses, meaning that either a sub-optimal configuration of the particle accelerator, an undetected malfunctioning during the data taking, or even an inaccurate reconstruction of the particle’s properties can negatively limit the upper-bound sensitivity to the new physics. Usually, these problems are optimized by hand-defined heuristics guided by domain knowledge, which could not be optimal due to difficulties or impossibility in specifying a well-defined objective function. To this

end, *reinforcement learning* can help overcome these limitations, potentially learning optimal decision-making policies from data, simulations, and even human feedback, indirectly improving future physics analyses.

Following the enormous success of large language models in these years, *foundation models* are proposed to be general-purpose rather than specialized models for a given domain. Thanks to both large-scale training and model architectures, these models can learn a general and transferable data representation, allowing them to solve multiple tasks without being specifically designed in advance to do so. This capability could unlock unprecedented data efficiency, enabling us to leverage large unlabeled datasets and solve tasks for which few samples are available.

All the models, methods, and ideas presented in this thesis are generally applicable. They may help particle physicists conduct better analyses by leveraging the increasing amount of big data that particle accelerators, like the Large Hadron Collider, can produce and collect when collisions between two beams occur.

*To the memory of Professor Flavio Savastano, my first mentor, who taught me more
than plain computer science. Thanks.*

Contents

1	Introduction	1
2	Parameter-conditional Neural Networks	4
2.1	Brief overview of a HEP analysis	4
2.2	Feed-forward neural networks	6
2.2.1	Definition	7
2.2.2	Training and validation	9
2.2.3	Regularization and the bias-variance trade-off	11
2.3	Conditioning on a physics parameter	16
2.3.1	Parameterized neural networks	17
2.3.2	Embedding a physics observable	18
2.3.3	Gating coefficients as feature importance	22
2.3.4	Exploiting domain knowledge	24
2.3.5	Interpolation and extrapolation	33
2.3.6	Handling continuous and multiple parameters	34
2.3.7	Relation to multi-task learning	35
2.4	Experimental results	37
2.4.1	The HEPMASS imbalanced dataset	37
2.4.2	Evaluation metrics	40
2.4.3	Model implementation and training	45
2.4.4	Classification results	46
2.4.5	Interpolation and extrapolation capability	48
2.4.6	Evaluation of conditioning mechanisms	51
2.5	Application of parameterized models in a real-world HEP analysis	52
2.5.1	Introduction	53
2.5.2	Analysis strategy	54
2.5.3	Improvement over the cut-based approach	57
2.5.4	Conclusions	59
2.6	Summary	59
2.7	Conclusions and future perspectives	59

3	Anomaly Detection for Model Independent Search	61
3.1	The promises of model-independence	61
3.2	Introduction to anomaly detection	62
3.2.1	Deep anomaly detection	62
3.2.2	Real-world challenges	65
3.2.3	Evaluation metrics	67
3.2.4	Convolutional neural networks	68
3.2.5	Auto-encoder models	72
3.2.6	Variational auto-encoders	74
3.3	State of the Art	78
3.3.1	Anomaly detection in deep learning	78
3.3.2	Anomaly detection in HEP	81
3.4	Joint variational representations for anomaly detection	87
3.4.1	The QCD vs Top benchmark	88
3.4.2	Learn discrete representations	89
3.4.3	Joint variational auto-encoders	92
3.4.4	The JointVAE architecture	93
3.4.5	FPGA implementation feasibility study with HLS4ML	95
3.4.6	Discussion and conclusions	101
3.5	Conditional dual auto-encoders for dark force search	102
3.5.1	Searching for Hidden Valley models	103
3.5.2	The CMS detector	104
3.5.3	Simulated particle collisions	105
3.5.4	Physics-inspired image augmentations	110
3.5.5	Conditional dual auto-encoders	111
3.5.6	Evaluation protocol and results	118
3.5.7	Conclusions and outlook	130
3.6	Summary	131
4	Deep Reinforcement Learning for Particle Physics	132
4.1	Introduction to reinforcement learning	132
4.1.1	Markov decision processes	134
4.1.2	The reinforcement learning objective	135
4.1.3	Learning the optimal policy	137
4.1.4	Model-based RL	142
4.1.5	Learning from demonstrations	146
4.1.6	Practice and challenges	149
4.1.7	Software libraries and tools	152
4.2	Applications in particle physics	153
4.2.1	Controlling complex equipment	153
4.2.2	Data quality and monitoring	156
4.2.3	Jet representation and reconstruction	158

4.2.4	Charged particle tracking	162
4.2.5	Emerging and future applications	164
4.3	Summary and outlook	166
5	Foundation Models for Particle Physics	167
5.1	The attention mechanisms	171
5.1.1	Additive, hard, soft, local and global attention	172
5.1.2	Multi-head self-attention	173
5.2	Generative pre-trained transformers	175
5.2.1	Decoder-only transformer architecture	175
5.2.2	Scaling laws	177
5.2.3	Aligning to human preferences	179
5.3	OmniJet, Masked Particle Modeling, and OmniLearn	181
5.3.1	Pre-training objectives	182
5.3.2	Leveraging knowledge transfer	185
5.3.3	Performance on downstream tasks	186
5.4	Leveraging real data in a foundation model	188
5.4.1	Generative pre-training	189
5.4.2	Classification fine-tuning	190
5.5	Future research directions	192
5.5.1	Limitations and challenges	193
6	Conclusions	196
	Appendices	198
A	Parameter-conditional Neural Networks	199
A.1	Code and data repositories	199
A.2	Discrete distribution for parameter assignment	199
A.3	Parameterized architectures	200
A.4	Comparison with standard ML approaches	202
A.4.1	Machine learning classifiers	202
A.4.2	Comparison results	203
A.4.3	Hyperparameters	206
B	Anomaly Detection for Model Independent Search	207
B.1	Data generation, simulation, and reconstruction	207
B.2	CoDAE, CoDVAE, and CAE architectures	208
B.3	AE architecture	210
	Bibliography	211

Chapter 1

Introduction

In the last decade, we have seen the rise of *Deep Learning* (DL) [1] over the more classic way of learning from data, i.e., what we commonly call *Machine Learning* (ML) [2]. Such revolution was possible thanks to the constant increase in *computational power*, the availability of *large datasets*, and that *domain knowledge* is often not a strict requirement since DL models can learn highly complex and non-linear relations from the (raw) input features, therefore not necessarily requiring a hand-made *feature engineering* phase which is often mandatory to achieve satisfactory performance with classical ML methods. Through an apparently easy recipe, another revolution is happening these days: scaling existing effective neural network models and architectures. By leveraging a *distributed training* paradigm over a sufficiently large computing infrastructure, it is possible to scale up (i.e., enlarge) a well-performing neural network, like a Transformer [3], even thousands of times obtaining models with *billions* (or even *trillions*) of learnable parameters. These so-called *large* or *foundation models* [4] can learn to solve a specific task at unprecedented accuracy, and even demonstrate a wide generalization capability, which enables them to solve problems (related to the main one) in a *zero-* or *few-shot regime* [5, 6], i.e., when provided with zero or a few additional data examples. Indeed, this is only possible by also scaling up the dataset(s) and, often, by employing both *pre-training* [7] and (multiple) post-training [8] phases.

Deep learning models can fit very complex and unknown functions thanks to the combination of *stochastic gradient descent* (SGD) [9] and *back-propagation* [10], which allows optimizing the model's parameters by following the gradient of the prediction error. This basic idea can be applied to various data modalities (or types), like table entries, images, videos, sequences (e.g., audio samples, time series, text), and even graphs (e.g., point clouds). For each of these kinds, a specific computational model (i.e., a *layer* in DL terminology) has been developed such that learning is more efficient: *convolutions* [11] for images and videos, *recurrences* [12] and *attention* [13] for sequences, and *graph networks* [14] for graphs. This allows applying DL models, i.e., deep neural networks, to a wide range of domains (e.g., chemistry, physics, engineering, etc.) and problems (e.g., protein folding [15], face recognition [16], weather forecasting [17] and much more) that admit a solution that is *differentiable* w.r.t. the model's parameters. In case of problems whose mathematical formulation allows

for a discontinuity at some point, it is possible to recur to various interesting techniques like either a *variational approximation* [18], a continuous relaxation of a mathematical operator, or even a *score estimator* [19] of the gradient. In this way, the non-differentiable parts of the model are either relaxed (e.g., subject to a continuous approximation because discrete) or transformed such that there is now a continuously differentiable path in the resulting computational graph, which represents both the model computation and parameters.

As briefly discussed, DL can be applied to (almost) whatever problem for which a sufficient amount of data is available¹. In general, the ease of gathering image and textual data, for example by scraping the web, allowed a fast-paced development of both *Computer Vision* (CV) [20] and *Natural Language Processing* (NLP) [21] paradigms, over the more classic *image processing*, and early *language models* and *grammars*. Today, CV and NLP mostly dominate the application landscape of DL models. The huge research effort behind them provided us excellent default and generalist model architectures like the *ResNet* [22, 23], *U-NET* [24], the *Transformer* [3], and the recently hyped *GPTs* [7, 25] to name a few. These models are so successful that are employed in a variety of fields like medicine and biology [26] (e.g., for lung cancer detection), engineering (e.g., autonomous vehicles and robots [27]), music and audio [28], astronomy, meteorology (e.g., weather forecast [29]), economics (e.g., automated stock trading [30]), physics, and much more.

It should be noted that each domain poses its unique challenges. For example, we would like to have *explainable* predictions [31] such that a human expert can later evaluate if these follow the underlying physics theory. Another example can be found in medicine, where predictions must be *trustworthy* [32], i.e., at least complying with strict confidence intervals. Next, when developing a robot (i.e., a physical agent), it is crucial to avoid hurting the human operator and preventing the agent from damaging itself [33] since the hardware and equipment are usually expensive. This is often solved by training in simulation and fine-tuning on the real world [34] or even by learning a *world model* [35], which is used to *plan* [36] an action before executing it. Then, in economics, the DL model can be subject to *distribution shifts* [37] due to trends developing and evolving as time passes. Other challenges are about the difficulty in gathering data, therefore the model must be efficient at learning from few samples [38, 5], and the robustness against *adversarial attacks* [39], which can exploit the model’s weaknesses for malicious intents: e.g., extract the training data violating privacy [40], confuse or alter predictions, damaging the equipment and so on. The impact of these challenges should be considered before facing a new problem with deep learning, and, often, the introduction of domain knowledge by an expert, either in form of a prior, feature engineering, pre-processing, or as an *inductive bias* [41] of some form (e.g., ad-hoc differentiable operator, loss function, auxiliary supervision, etc), can greatly mitigate such issues yielding an overall better solution. As mentioned, expert knowledge has not disappeared with the advent of deep learning since it can be a useful resource to improve the model’s performance: the hard part is to figure out how this knowledge can be

¹ML is usually more effective than DL when the data is scarce. Moreover, some classes of problems admit an exact solution that can be found by studying the nature of the problem. Then, even if we can collect some data for them, a search-based, algorithmic, or analytical solution is often preferred to solve them.

embedded into a DL workflow.

This thesis presents multiple examples of integrating knowledge from the high-energy physics (HEP) domain in a deep learning system. It discusses potential solutions, requirements, and difficulties. While the examples and guidelines discussed in this thesis are specific to HEP, they may hopefully inspire other scientific fields as well. In particular, the topics that will be presented in this thesis are all related to improving specific aspects of new physics searches:

- **Chapter 2** introduces the notion of parameter-conditional neural networks. These are mainly demonstrated for signal-background classification assuming a physics model of the signal.
- **Chapter 3** delves into anomaly detection and auto-encoders to implement model-independent analyses, which are complementary to the techniques of the previous chapter.
- **Chapter 4** first describes modern reinforcement learning, then discusses several tasks and applications strictly related to HEP analyses. Better solutions to these problems are expected to yield general improvements in model-dependent and -independent new physics searches.
- **Chapter 5** presents the recent practice of large-scale pre-training, resulting in domain-specific foundation models, which can be particularly beneficial for high-energy physics.
- **Chapter 6** concludes this dissertation with an overview of the discussed topics.

Chapter 2

Parameter-conditional Neural Networks

This thesis chapter introduces the research line about a class of neural networks that establish a dependency on one or more physics parameters (i.e., quantities relevant in particle physics). These novel neural networks are particularly important for the search for new physics when a model of the expected signal is assumed, helping to achieve a superior discrimination accuracy of physics events.

2.1 Brief overview of a HEP analysis

One of the fundamental problems in high-energy physics is the separation of the so-called *signal* from the *background* to achieve signal sensitivity. This task is usually called *signal-background classification* (or signal selection), which is about classifying (discriminating) already known physics events, i.e., the background, from an often small fraction of them that can potentially represent a new physics: the signal. These *events* are originated by colliding known particles (e.g., protons) in accelerators like the Large Hadron Collider (LHC) [42]; the collision of such particles generates a variety of products or *decays*, identified by the triggers. Through the observed decay products, the original event is reconstructed, and inspected by physicists looking for *resonances* that may represent novel particles. The various products of the collision are predicted by the underlying theory and validated experimentally through complex equipment or *detectors*, like the Compact Muon Solenoid (CMS) [43] at the LHC, which can detect (react to) specific kinds of particles (e.g., muons). Detectors usually comprise multiple layers or sub-detectors, composed of varied materials and technologies that are sensible to a particular set of particles. Then, if such a particle passes through the material it produces one or more *hits*, which can be combined across the different layers, forming a *track*. Historically, tracks were inspected by eye in *bubble chambers* since their peculiar shape may reveal the type of particle that hit the detector. Nowadays, things are much more complex and, therefore, the existence of a new particle (or not) is validated through strict statistical tests [44] based on the collection and processing of real-world

collisions of particles uncovered by a given detector.

In general, a conventional HEP analysis first involves the *generation* of simulated particle collisions, from which the objects of interest are *reconstructed*. Then, the resulting events are filtered according to various criteria, either defined by the underlying theory or as a result of the experimental equipment, to remove trivial background events; this *pre-selection* phase can discard even a large portion of the generated data. A successive *event selection algorithm* is applied, estimating its efficiency on selecting signal and rejecting background by measuring the count of events passing it. Follows a precise estimation of the Standard Model (SM) background and signal processes, through simulations or in a data-driven fashion, to understand the contribution of each. Lastly, a statistical test, which involves the usage of the real data collected by the collider, is performed to test for the presence of the signal through a simultaneous fit of the signal and background distributions of a chosen observable (i.e., the signal + background hypothesis) against the SM background-only hypothesis, hoping to extract a non-zero signal component whose magnitude is given by the *significance*, i.e., the probability that the signal exists. To certify a discovery this significance should exceed five standard deviations, representing an extremely low *p*-value, to be sure that the existence of the signal cannot be excluded with high confidence and cannot be mismatched for a background fluctuation.

Event selection algorithms can be implemented by hand or in a data-driven fashion. Earlier criteria used to be expert-designed based on single *cuts* (i.e., a threshold on a particularly relevant variable) driven by physical considerations. The role of the physics expert was to inspect the distribution of each measured variable characterizing the collected events. Then, the variable for which most of the background can be rejected while preserving the majority of the signal is chosen, and the cut (i.e., the threshold) is tuned to meet a target signal or background efficiency. After this point, the thresholded background events are used to estimate the background contribution, for example by fitting a known mathematical model with the *real-world data* (i.e., collision events collected by the detector), which is next employed to perform a statistical test that should reject or not the hypothesis for which the data also contains the signal. The test can have three outcomes: 1) the significance is too low, therefore the new particle does not exist with certainty; 2) the significance is larger than five standard deviations, thus the signal may exist with certainty (unless a piece of dedicated equipment does not reveal it); and 3) the significance is neither high nor low, hence nothing can be concluded since there is too much uncertainty, and the best attempt we can do is to improve the analysis workflow hoping for a higher significance. Alternatively, since there could still be a possible discovery, in this latter case, *upper limits* can be established to constrain the phase space for subsequent searches.

These are the steps that a typical particle physics analysis follows, and these days we are experiencing a shift from hand-designed to data-driven methods and criteria since machine learning can be introduced at various points of such pipeline:

- **Data simulation.** Theory-driven software packages like `Pythia8` [45] can generate several particle decays, following a specification of the desired physics phenomena. These Monte Carlo simulations are very accurate, trying to mimic data originating

from real-world particle collisions in accelerators, but computationally demanding; to achieve higher fidelity, **Geant4** [46], which models the detector’s geometry, simulates the interaction of the particles with the detector. The field of *fast simulation* exploits DL to learn to generate high-quality samples directly through a generative model, therefore without intermediate costly computational steps that simulate the underlying physics. Notable examples are [47, 48, 49, 50] that can generate high-quality events in a fraction of time achieving considerable speedups.

- **Event selection.** As briefly described above, physics experts often conduct event selection to determine human-readable selection criteria (or cut) to separate background from signal events. In principle, this operation can be entirely replaced by a variety of ML models, like Decision Trees (DTs) [51] or Neural Networks (NNs) [52, 53], adapting the selection criteria according to the given data. These models can learn to classify the simulated samples, outputting either the class or probability that an event is a signal or background. In this regard, the main challenge is to develop models that are both accurate and interpretable, such that, later, a human expert can reason about which features were most relevant for the classification: this is one of the requirements for trustworthy predictions since even well-performing models can put too much focus on either noisy variables or weak ones (from a particle physics theory perspective) to establish the predicted class. It should be noticed that simulated events are just approximations of Nature, therefore, patterns and spurious correlations due to *systematic uncertainties* (i.e., inaccuracies of the measurement due to, for instance, limitations of the experimental apparatus) can be involuntarily injected in the data samples which, in turn, could be captured by the learning algorithms resulting in unreliable discrimination performance.
- **Hypothesis testing.** Even the statistical test between the reference (or null) and alternative hypothesis can be learned by machine learning. A notable example is the *New Physics Learning Machine* (NPLM) [54, 55] that, through simulated or toy data, can fit a parametric form of the log-likelihood-ratio hypothesis test in a complete end-to-end manner, by parameterizing the data distribution with a neural network, which results in a p -value for the discovery. The NPLM can also account for uncertainties by introducing nuisance parameters in the test computation.

As just discussed, ML and DL can be introduced to either speed up or improve various parts of a physics analysis, therefore, being complementary to domain experts.

2.2 Feed-forward neural networks

The basics of neural network definition and training are introduced in this section, being a useful foundation for the rest of this thesis.

2.2.1 Definition

Neural networks are a mathematical (or, better said, statistical) tool to approximate a mapping, say $f : \mathbb{R}^d \rightarrow \mathbb{R}^k$ in which: $\mathbb{R}^d$ and $\mathbb{R}^k$ are, respectively, arbitrary input and output *shapes*. By shape we mean the rank and dimensionality of each axis, therefore by changing d and k , we can indicate: scalars ($d = 1$), vectors ($d = n$, where $n \in \mathbb{N}$ is an integer indicating the number of elements of the vector), or tensors ($d = n_1 \times n_2 \times \dots \times n_r$ where r is the rank of the tensor and each $n_i \in \mathbb{N}$ indicate the dimensionality of each axis). For example, a vector of shape (10,) has ten elements, while a 2×3 matrix has shape (2, 3) and rank two. Notice that although the shape is mostly a practical rather mathematical concept, introduced by numerical computation frameworks, it is useful for understanding how the input data interacts with the components of a neural network.

Given an input $x \in \mathbb{R}^d$, a neural network maps (or, transforms) it to an output $z \in \mathbb{R}^k$ by the following application: $f_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^k$, which can also be written as $z = f_\theta(x)$ or $z = f(x; \theta)$. The notation f_θ or $f(\cdot; \theta)$ indicates that the neural network is a *parametric function* whose set of parameters, θ , can be learned by an optimization algorithm, like gradient descent. In practice, thanks to both hardware and software optimization, we can *vectorize* the underlying operations that made up a neural network, therefore f_θ can be applied to a *batch* of data rather than a single example. Since this case is quite common, we indicate the *batch size* by B and modify both input and output shapes to account for an additional axis (usually the first or last one):

$$f_\theta : \mathbb{R}^{B \times d} \rightarrow \mathbb{R}^{B \times k}. \quad (2.2.1)$$

The resulting function now operates on a collection of inputs $X \in \mathbb{R}^{B \times d}$, and outputs $Z \in \mathbb{R}^{B \times k}$, where a single element in the batch is denoted by $X^{(i)}$ with $i \in 1, \dots, B$.

At this point, the question is: *what function f_θ is?* The answer is a composition of functions! Neural networks are built from basic building blocks called *layers*, which can be arranged in an arbitrary complex manner forming a structure called a *directed acyclic graph*, or DAG in short. The DAG (figure 2.1a) represents the computation and data flow: nodes depict layers, incoming edges their inputs, and outgoing edges their outputs. If the nodes are further expanded to show the basic operations that a layer implements, we obtain the *computational graph* of the neural network. This graph fully explains all the underlying mathematical operations, intermediate results, partial derivatives, and the network output: figure 2.1b. To avoid useless complexities, let's consider a common sequential design of a neural net: a *feed-forward* neural network (FFNN), also called *fully-connected*, or a *multi-layer perceptron* (MLP) [56]. A FFNN is made of n layers, arranged in sequence one after the other as follows:

$$f_\theta = f_n \circ f_{n-1} \circ \dots \circ f_2 \circ f_1. \quad (2.2.2)$$

Denoting a generic layer as f_i (with $i \in 1, \dots, n$), we use the symbol of function composition, $\circ$, to combine multiple layers sequentially. With this notation, f_1 is the first layer to be applied to the inputs X , which is also called the *input layer*, and f_n is the last layer, responsible for computing the outputs Z : in fact, it is called the *output layer*. All the

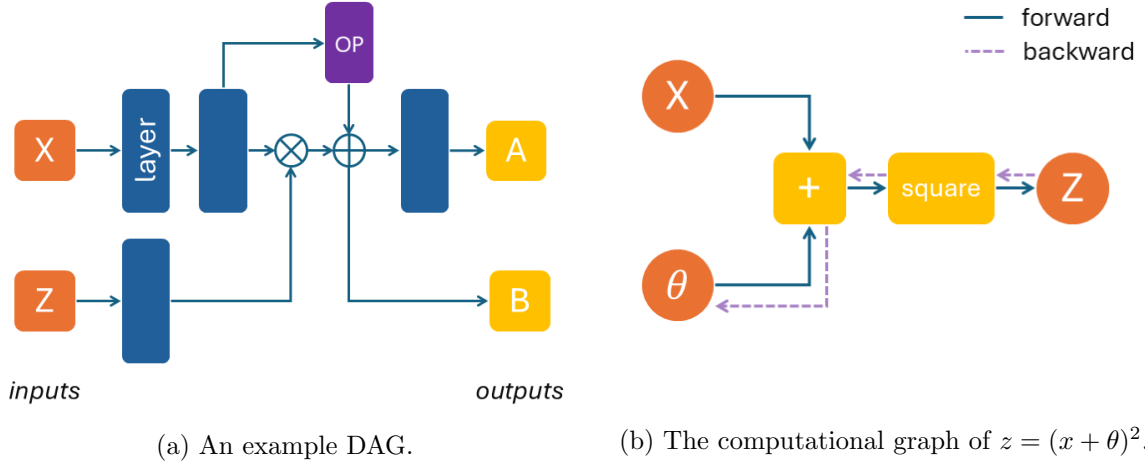


Figure 2.1: Left: the arbitrary arrangement of layers and operations can be pictured as a DAG, describing the architecture of a neural network; neural networks can have multiple inputs and outputs, consisting of layers with learnable parameters (blue rectangles) and fixed, custom operations (purple). Right: a simple computational graph, in which input (x, θ) and output (z) nodes are linked by basic mathematical operations; blue, solid arrows denote the forward computation or pass, while their back-propagated gradient path (from output, z , w.r.t. the parameter, θ) is shown by the purple, dotted line; refer to the next section for more details about the backward pass.

other intermediate layers (from f_2 to f_{n-1}) are named *hidden layers* because their training dynamics are somehow hidden from the user. Moreover, we call *deep* whatever neural network with more than one hidden layer¹, and *shallow* otherwise.

The power of deep neural networks (DNNs) not only resides in the composition of many layers, but it is also about two elements: *weights*, and *non-linear activation* functions. The weights of a layer are its learnable parameters, which are all collected in θ : some layers have no weights, for example, the ones that perform a normalization operation. Each layer applies the weights to its inputs, weighting its features. In some regards, we can think of the weights as multiple learnable *filters* able to capture a data-dependent set of (hidden) features from a given input. We distinguish two kinds of weights: the *biases*, b , and the *weight matrix*, W . The biases are vectors of size u , which indicates the number of *units* (also called neurons, or the number of filters). Whereas the weight matrix W has shape (d, u) , where d is the input dimensionality in the case of vectors. A one-layer, therefore shallow, FFNN can be denoted as:

$$f_{\theta}(X) = g(X^{\top}W + b), \quad (2.2.3)$$

where g is the so-called activation function, and $\theta = (W, b)$. If g is the identity function

¹Nowadays this distinction is a bit controversial, as networks with few layers (depending on what “few” means) are still considered shallow, since there exist extremely deep networks with hundreds of layers.

(i.e., $g(x) = x$), we obtain a *linear* neural network. These are simply linear models that can only learn first-order (i.e., linear) relations between the inputs and the desired output(s). To be useful the activation should be whatever non-linear transformation such as $\text{ReLU}(x) = \max(0, x)$ [57], or the *sigmoid* $\sigma(x) = \frac{1}{1+e^{-x}}$, which is also called the *logistic function*. The right side of equation 2.2.3 depicts what we call a *dense* or fully connected layer, which implements an affine transformation on the input vector, $\mathbf{x}$, changing its dimensionality from d to u . Values like u , which is layer-specific, are called *hyperparameters* because these are outside the scope of the optimization algorithm. Hyperparameters often govern design choices, being usually hand-picked. Selecting good hyperparameters is fundamental since these influence the goodness of the model; the field of *hyperparameter optimization* (HPO) [58] aims at finding the best set of hyperparameters, given a computational budget, through sophisticated search strategies. Therefore, we call an *architecture* the arrangement of layers and, sometimes, even the value of particularly relevant hyperparameters that define a neural network. Research brought us default architectures like the ResNet [22] or the Transformer [3], which are often employed to solve a variety of problems with minor modifications, simplifying how we approach a problem for the first time.

2.2.2 Training and validation

Finding a well-performing architecture is only half the work, the other half is about finding a good set of parameters too! Since neural networks are built around differentiable operations, gradient-based optimization techniques can help us learn (discover) the best weights.

To train (or fit) a neural network we need essentially three elements: *target* data, a way to find the direction of improvement, and an update rule. Target data are usually denoted by $\mathbf{y} \in \mathbb{R}^m$ (where m indicates the shape of $\mathbf{y}$) and, assuming a *supervised* setting, are collected along with the input features, $\mathbf{x}$. In this regard, the targets represent our desired predictions or the *ground-truth*; typically: classes or categories for *classification* problems, or continuous values for *regression* tasks. A supervised dataset, with N data samples, is often denoted as $\mathcal{D} = \{(\mathbf{x}, \mathbf{y})_i\}_{i=1}^N$, emphasizing the coupling between features, $\mathbf{x}$, and targets, $\mathbf{y}$ since we want to learn (i.e., approximate with a NN) the mapping $f : \mathbf{x} \rightarrow \mathbf{y}$. The notation $(\mathbf{x}, \mathbf{y})_i$ is a shorthand for $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$, which indicates the i -th data sample.

To find the direction of improvement we have to first *score* the predictions of our neural network model. The *loss*, *cost*, *error*, or *objective* function, say $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ where $\hat{\mathbf{y}} = f_{\theta}(\mathbf{x})$, which is continuously differentiable, tells us how large the discrepancy between ground-truths, $\mathbf{y}$, and our predictions, $\hat{\mathbf{y}}$, is. Since we train over mini-batches, $\mathcal{B} \subset \mathcal{D}$, we usually *average* the loss over them:

$$\mathcal{L}_{\mathcal{B}} = \frac{1}{B} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{B}} \mathcal{L}(\mathbf{y}, f_{\theta}(\mathbf{x})). \quad (2.2.4)$$

The loss, $\mathcal{L}_{\mathcal{B}}$, is now part of the *forward* computational graph (i.e., all the operations the neural net performs during a prediction, also called *inference* or *forward pass*) and, since differentiable, we can compute the derivative of each operation (i.e., each node in the graph w.r.t. its inputs, depicted by the incoming edges) to obtain the direction along which our

predictions, $\hat{y}$, are expected to improve. The way we get these derivatives is by the *back-propagation* algorithm [10], which exploits the *chain rule* of calculus to yield the *gradient* of the loss w.r.t. the set of learnable parameters $\theta = [\theta_1, \theta_2, \dots, \theta_n]$:

$$\nabla_{\theta} \mathcal{L}_{\mathcal{B}} = \frac{d\mathcal{L}_{\mathcal{B}}}{d\theta} = \left[\frac{\partial \mathcal{L}_{\mathcal{B}}}{\partial \theta_1}, \frac{\partial \mathcal{L}_{\mathcal{B}}}{\partial \theta_2}, \dots, \frac{\partial \mathcal{L}_{\mathcal{B}}}{\partial \theta_n} \right], \quad (2.2.5)$$

where n is the number of (hidden) layers, and each $\theta_i = (W^{(i)}, b^{(i)})$ represents the parameters (weight matrix and biases) of each layer. The gradient is a collection of partial derivatives, each telling in which direction the respective parameter (or weight) *increases* in value. Therefore, to improve the predictions the weights should be updated by taking a step in the direction pointed by the *negative* gradient: this is because we want to *minimize* the error, rather than maximizing it². During training this is iterated until convergence: at a generic iteration, t , the typical update rule of *mini-batch stochastic gradient descent* (SGD) [37] is the following:

$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \eta^{(t)} \nabla_{\theta} \mathcal{L}_{\mathcal{B}}, \quad (2.2.6)$$

where $\eta^{(t)}$ is the so-called *learning rate* (or step size), whose value can change according to a schedule [59] which depends on the iteration value. The role of the learning rate is to shrink the gradient to avoid taking a too large step, hence preventing instabilities and divergence during training.

As depicted in equation 2.2.5, the gradient of the loss is obtained by back-propagating the error from the output layer down to the input layer by exploiting the chain rule. All these operations to compute the gradient are indicated as the *backward pass*, in which starting from the loss, $\mathcal{L}_{\mathcal{B}}$, computes

$$\frac{\partial \mathcal{L}_{\mathcal{B}}}{\partial \theta} = \frac{\partial \mathcal{L}_{\mathcal{B}}}{\partial z} \frac{\partial z}{\partial \theta}, \quad (2.2.7)$$

until reaching the models' inputs, x , and where z denotes an intermediate result (i.e., yield by a hidden layer). The term $\frac{\partial z}{\partial \theta}$ in equation 2.2.7 is expanded, by applying the same rule again, until $z = x$.

At this point, we are ready to train our model, f_{θ} , on a training dataset $\mathcal{D}_{\text{tr}}$. The usual training loop is divided into *epochs*, in which each should process all the examples in $\mathcal{D}_{\text{tr}}$. Each epoch is divided into training steps, in which a mini-batch sampled from $\mathcal{D}_{\text{tr}}$ is fed to the model whose predictions are used to compute the loss and then update the model's parameters by following equations 2.2.7 and 2.2.6; an epoch contains as much training steps as the size of $\mathcal{D}_{\text{tr}}$ divided by the batch size, B . After the end of each epoch, it is a good practice to validate the hyperparameters by evaluating the model on a separated set of data, $\mathcal{D}_{\text{val}}$, called the *validation set*. By keeping track of the target metric(s) achieved on validation at each epoch, we can understand at which point in time we obtained the model that is expected to perform the best on the *test set*, $\mathcal{D}_{\text{test}}$: i.e., a

²In some cases, we want to perform a maximization. To achieve that we *negate* the loss, and update the model as usual: this is called *gradient ascent*.

completely separated set of data used for the final evaluation of the model, which occurs only after a complete training. The practice of saving the model's weights at the best epoch (according to validation metrics) is called *model selection*.

2.2.3 Regularization and the bias-variance trade-off

A successful training phase aims not to learn the model that attains the lowest possible error on the training data but to achieve the widest possible *generalization* on unseen data: i.e., the best performance on them. For a given task or problem T , say classifying images of dogs and cats, there exists an ideal dataset, $\mathcal{D}_T$, that contains all the possible data samples regarding the task T ; in our example, $\mathcal{D}_T$ holds all the images regarding the various breeds of dogs and cats, which may include photographs, digital images, sketches, and so on. As said, $\mathcal{D}_T$ is ideal (and potentially infinite): in practice, the best we can do is to collect a subset of it, i.e., $\mathcal{D}_{\text{data}} \subset \mathcal{D}_T$. As a rule of thumb, our dataset, $\mathcal{D}_{\text{data}}$, is divided (split) into three parts: training set, $\mathcal{D}_{\text{tr}}$, validation set, $\mathcal{D}_{\text{val}}$, and test set, $\mathcal{D}_{\text{test}}$, as seen in the previous section and depicted in figure 2.2.



Figure 2.2: How the collected data is split for model training and evaluation. In DL it is common to divide the data into training, validation, and test sets. Ideally, each split is randomly drawn from the same underlying distribution to introduce no shifts.

The split usually follows a 80/20 proportion, especially if we have plenty of data, meaning the 20% of $\mathcal{D}_{\text{data}}$ is allocated to test samples; the remaining 80% is further divided to account for the validation samples, which take a 20-25% of them.

Data collection usually happens few times (typically, one), and imagine, instead, we collect multiple datasets, i.e., $\mathcal{D}_{\text{data}}^i \sim \mathcal{D}_T$ and, then, fit the same model (without changing hyperparameters) on each training dataset $\mathcal{D}_{\text{tr}}^i \subset \mathcal{D}_{\text{data}}^i$. If we record all the performance achieved on the different training sets and draw them we may notice some variability, which, in some cases, can be quite large: this is a statistical phenomenon, called *variance*, which is influenced by the particular sampling of the (collected) data. Suppose we take the mean of all these models, which we call the *average model*; the difference in performance between the average model and a single one is the variance.

Now suppose we were given the full dataset, $\mathcal{D}_T$, and trained a model on it (let's also assume an optimal set of hyperparameters): the performance of this model is the *best* it could be achieved, and if the data were clean from flaws (e.g., mislabeling, noisy measurement, intrinsic ambiguities, etc) such best model would match the *perfect model*, i.e., an

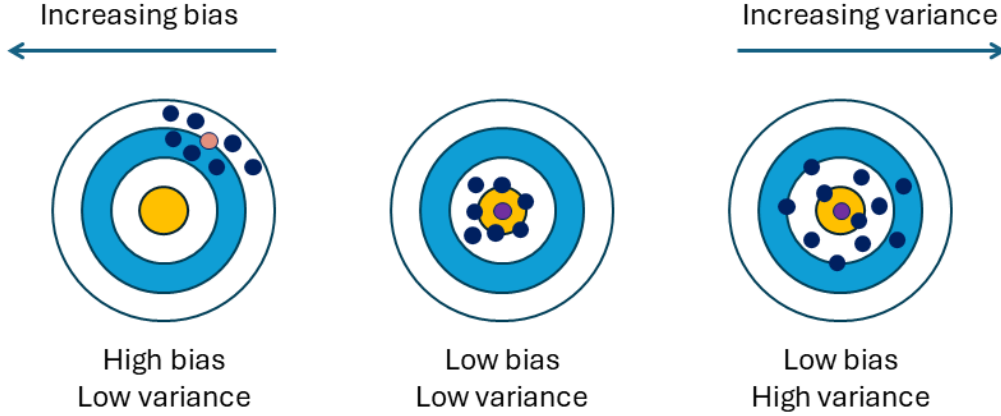


Figure 2.3: The bias-variance trade-off: three regimes are possible in practice. The ideal low-bias, low-variance regime, can be achieved by training bigger and better models for longer, with more data, and proper regularization. Colored circles (●, ●, and ●) denote the training runs, average model, and perfect model, respectively.

ideal model that achieves the best possible performance. In practice, data are ambiguous therefore there is a gap, called the *irreducible error*, between the best performer(s) and the ideal model. Hence, there are unattainable models since theoretical, because we do not have access to the full population of data samples for the task T .

With this in mind, we should collect data and then clean them to be representative of the population. Even so, the achieved model performance is only a mere approximation of the full picture: in this regard, we aim to get a low bias (thus, low error) and low variance (i.e., small variability) regime. In practice, this can be achieved by training a capable model family (i.e., a performant architecture with good hyperparameters) on a sufficient number of high-quality data, and applying the right regularization techniques: we seek a good compromise (figure 2.3) between bias and variance because, due to a trade-off between them, decreasing variance causes the increase of bias and vice-versa.

Regularization is usually employed when: after reaching a low-bias regime, meaning our model can fit the training data well, the generalization performance on test data is still too weak (i.e., there is too high variance); this situation is also regarded as *over-fitting* (figure 2.4), where the training error tends to zero while the validation error keeps increasing rather than decreasing. This happens because the model is too powerful for the amount and type of data we have, therefore we should reduce its complexity. Instead, a situation in which the model yields a high error (or high bias) is called *under-fitting* since the model is too simple for fitting a more complex function. Therefore, regularization techniques aim at increasing generalization (i.e., reducing variance) while keeping a good fit of the target function (low bias). In the context of neural networks, popular methods are:

- **Weight decay** [1]. Parameter norm penalties like weight decay (also known as l_2 ,



Figure 2.4: Over-fitting can be detected by inspecting the learning curves: if either the loss or the target metric (e.g., accuracy) computed on the validation set starts to diverge compared to the curve achieved on the training set, then after that epoch the model begins to over-fit the training data. Figure adapted from <https://neptune.ai/blog/early-stopping-with-neptune>.

ridge, or Tikhonov regularization) add a penalty term to the loss function such that it increases as the l_2 norm of θ (i.e., the parameters) increases. With the penalty, the overall loss function now is:

$$\mathcal{L}_{\text{tot}} = \mathcal{L}_{\mathcal{B}} + \frac{\lambda}{2} \|\theta\|_2^2, \quad (2.2.8)$$

where $\mathcal{L}_{\text{tot}}$ is the new loss function, and λ is a coefficient that controls the strength of the regularization term (i.e., the square of the Euclidean norm of the parameters.) Recent research pointed out that if we update the weights by following the equation 2.2.6, the weight decay term does depend on the learning rate, η , making it more difficult to tune during training. Therefore, optimization methods that implement the *decoupled weight decay update rule* [60] are preferred, allowing for better control of the regularization strength through λ .

- **Label smoothing** [61, 62]. To solve classification problems the model should learn class probabilities, which are often extreme values like being exactly zero or one. The *cross-entropy loss*, which is typically used in classification, encourages the model to match those probabilities therefore still penalizing the model if its predictions are not exactly zero or one: for example, if the neural net predicts 0.95 instead of 1.0, the loss is still not negligible; hence, the gradient of the error is still large. Label smoothing is a simple technique in which the class probabilities (or their one-hot encoding³) are softened by a small value, ϵ : typically 0.05 or 0.1. In binary classification, if a sample

³Considering three class labels (1, 2 and 3), these can be one-hot encoded as [1,0,0], [0,1,0], and [0,0,1], respectively. The one-hot encoding (OHE) generally works for categorical variables.

is assigned a probability of 1 this regularization technique smooths the value to 0.95 (with $\epsilon = 0.05$.) Instead, in multi-class classification, the one-hot encoding, y^{ohc} , of a label, y , is modified as follows:

$$y_i^{\text{ohc}} = \begin{cases} 1 - \epsilon \frac{C-1}{C} & \text{if } (i = C - 1) \\ \epsilon/C & \text{otherwise} \end{cases}, \quad (2.2.9)$$

where C is the number of classes, and $i = 1, \dots, C$ indexes the one-hot encoding which is a C -dimensional vector. Label smoothing is a handy method to both mitigate overfitting, caused by large gradients that try to exactly match the probability value, and mislabeled class labels as well, which may be (or considered as) corrupted by noise with magnitude ϵ .

- **Early stopping** [1, 37]. When training models with a sufficient capacity to overfit the learning task, we often observe the training error decreasing over time while the validation error increases. To fight this phenomenon we can think of constraining the number of epochs rather than the values of the weights, returning the parameters associated to the lowest validation error. This is known as early stopping, in which the validation error is monitored at regular intervals (e.g., after each epoch) during training to determine if the learning should be halted at some point. Early stopping can be interpreted as an efficient hyperparameter selection algorithm, allowing control of the effective capacity of the model by determining the number of training steps. Moreover, it can be combined with other regularization strategies. Since early stopping requires a validation set, which reduces the effective number of training samples, we can think of performing additional training after the initial one in which the validation data is also used for training. In this case, we train the model on all the training data, i.e., $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$, for the same number of steps the early stopping procedure determined. Lastly, the regularization effect of early stop can be studied in terms of l_2 -regularization, in which, compared to weight decay, early stopping has the advantage of automatically determining the correct amount of regularization [1]. In addition, an early stop can be particularly effective in the presence of label noise or intrinsic variability in the target variables [37].
- **Ensembles.** It turns out that if we aggregate (e.g., by averaging) the predictions of (many) models, even of different kinds, we get an *ensemble of models*, i.e., a new model, which is better than every single model in the ensemble. A fundamental requirement for the ensemble to work is that the individual models should make statistically *independent* mistakes. If models are uncorrelated, the average of the errors made by each model will cancel out as more models are added to the ensemble, reducing variance. Regarding neural networks, a way to make independence more likely is to change *weight initialization* method [63], or even to train the models on random subsets of the data such as in *bagging* [64]. To be effective, ensembles should be sufficiently large, hence the computational resources to both train and infer on these models can get quite costly. Knowledge distillation [65] is a simple technique

that can *distill* (compress) either a large model or ensemble into a smaller neural network, without losing its discriminative power while keeping inference lightweight.

- **Dropout [66].** In each forward pass, during training, some activations are randomly set to zero therefore *dropped out*. In practice, according to the dropout probability p (usually set between 0.2 and 0.5), a new binary mask is sampled at each mini-batch, which is then multiplied by the output of a given hidden layer (i.e., the one to which the regularization should be applied to). Dropout has two main interpretations. The first regards the model being forced to learn a *redundant representation* that prevents the co-adaptation of features, i.e., a phenomenon in which a neuron relies too much on the other ones. The second, instead, is as if dropout enables training a large ensemble of models that share parameters; these models are obtained each time a binary mask is sampled since the mask is as if changing the structure of the neural network removing and moving neurons randomly. Since sampling is involved, dropout (as described) makes the output of our neural network stochastic! To avoid averaging many predictions at test time (inference), which is costly, the *inverted dropout* scales the model parameters during training by the inverse of the dropout probability, i.e., $1/p$, to yield a model that at inference time behaves like the average of each sampled networks.
- **Data augmentation [67].** In general, the model complexity of a neural network can also be controlled by the amount of training data we use. For example, say f_θ has 1M parameters. If we keep the number of parameters fixed while varying the number of training examples, we may notice that: 1) with few samples (say less than a thousand), the model easily over-fits them since these can be memorized in θ ; 2) with a lot of varied samples (e.g., $100\times$ more than the parameters) the model cannot fit all the data equally well, therefore it may start to under-fit them due to limited capacity; and 3) with the right amount, the model can fit the training-set well also achieving good generalization. In case we witness over-fitting, by gathering more training data we can mitigate it but, in practice, this can be either difficult or prohibitive. Hence, data augmentation techniques aim to generate *synthetic* data by smartly altering the real ones, increasing both quantity and variety. Most augmentation techniques are developed for images, as shown in figure 2.5, and mostly involve: *geometry* (e.g., rotation, flip, shear, translation, zoom) and *color* (hue, contrast, brightness, saturation, channel shuffle or drop). Furthermore, advanced image augmentations were developed to improve the learning of better representations [68] such as *cutout* [69], or to ensure a wider generalization like in CutMix [70] and MixUp [71], in which both samples and their class probabilities are mixed to fill the “gaps” in the feature (pixel) space, thus ensuring a smoother fit in those regions where data samples are scarce. Notice that data augmentations are only applied to training data.

As mentioned, when training a neural network we should aim to reach a low-bias low-variance regime, meaning that the model achieves both low error and generalizes on unseen data samples. The general recipe to achieve this regime is to train a high-capacity model

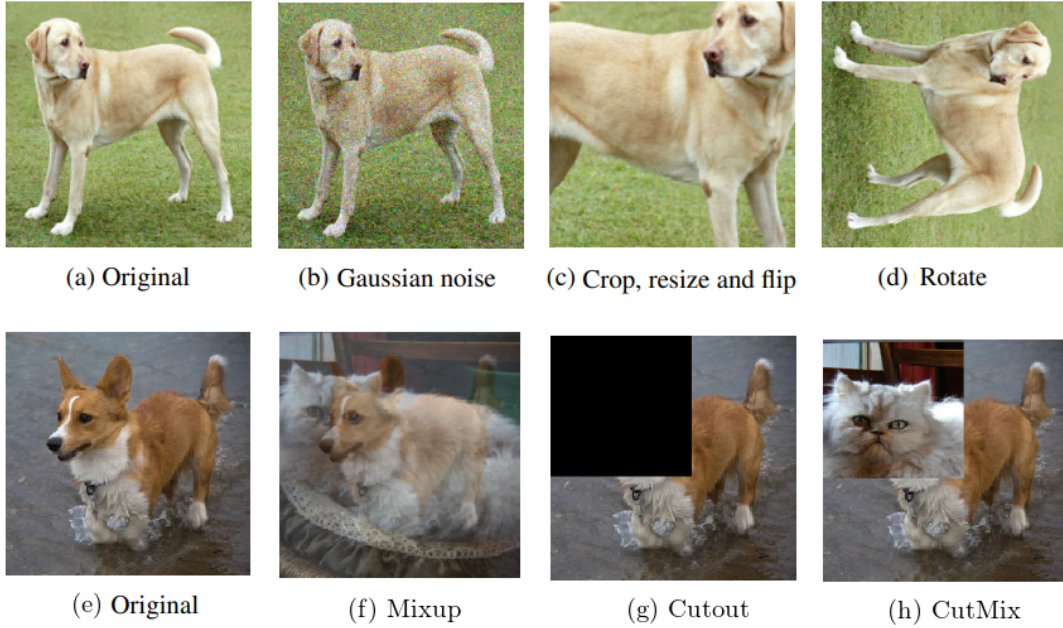


Figure 2.5: Examples of data augmentations for images. Figure adapted from [68] and [70].

first, then, regularize it until overfitting is mitigated. Once a good model is obtained, we can improve its performance further by searching for better hyperparameters or building an ensemble of models.

2.3 Conditioning on a physics parameter

So far we have seen a simple yet working formulation of neural networks, which can be directly employed to solve the signal-background classification part of a HEP analysis as described in section 2.1. For example, Baldi et al. [52] showed that a deep neural network trained on either low-level variables (e.g., energy, momentum, and transverse momenta of each observed particle) or high-level ones (e.g., the invariant mass of a particle) can surpass both decision trees and shallow neural nets, achieving a higher background rejection rate while preserving a larger amount of signal events. Nevertheless, their approach suffers for those benchmark scenarios in which the mass of the hypothetical particle or decay is not known in advance because the theoretical prediction is uncertain. In such cases, the new physics analysis is often carried over M different (signal) *mass hypotheses*, therefore many signal events are generated for each. Then, a further responsibility of the analysis is to understand which mass point is more plausible for the signal to exist. In this regard, the canonical approach was to train as many *individual and independent* classifiers as the mass hypotheses, i.e., M . It is easy to see that such an approach won't scale as M increases, since the physicists' effort increases proportionally. We need a kind of classifier that can handle

all the mass points jointly, thus avoiding the redundancy of training multiple models.

2.3.1 Parameterized neural networks

The Baldi's *parameterized neural network* (or pNN) [53] is a neural network architecture that leverages an additional input, e.g., a physics quantity, which can replace a set of individual classifiers (each specific for a mass hypothesis), therefore achieving smooth predictions over a mass range. Let's define the set of (discrete) mass hypotheses as $\mathcal{M} = \{m_1, m_2, \dots, m_M\}$, in which each m_i is a signal generating mass (with $i = 1, \dots, M$) and $|\mathcal{M}| = M$; the input feature vector as $\mathbf{x}$, which can represent either low- or high-level physics variables. The pNN formulation introduces an additional (auxiliary) input, called the *mass feature* $\mathbf{m}$, which is a vector of mass hypotheses $m^{(i)} \in \mathcal{M}$ each associated to an event: ideally the model learns from both $\mathbf{x}$ and $\mathbf{m}$, therefore

$$\hat{y} = f_{\theta}(\mathbf{x}, \mathbf{m}) \quad (2.3.1)$$

is our new classifier. To overcome the problem of how to learn from two input vectors, Baldi et al. simplify the above formulation to a standard FFNN by introducing the *extended* features $\bar{\mathbf{x}} = [\mathbf{x} \ \mathbf{m}] \in \mathbb{R}^{d+1}$, which are obtained by concatenating $\mathbf{x} \in \mathbb{R}^d$ with $\mathbf{m} \in \mathbb{R}$, hence: $f_{\theta}(\bar{\mathbf{x}})$ is a special case of equation 2.3.1 that can be implemented with dense layers as shown in figure 2.6.

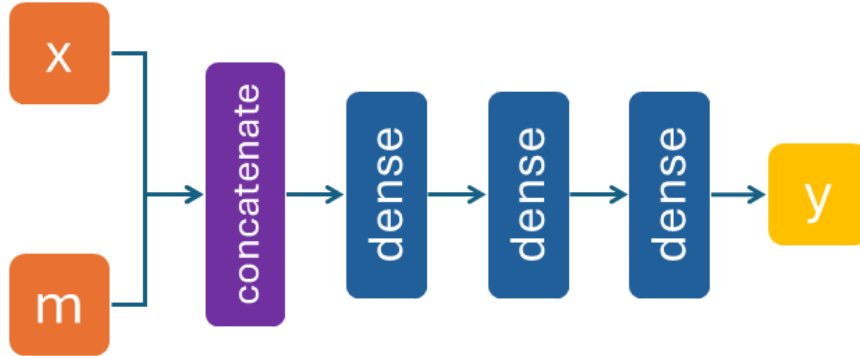


Figure 2.6: A schematics of Baldi's pNN architecture: the parameter is concatenated with the features, and then non-linearly transformed.

It should be noticed that our supervised dataset now has the form of $\mathcal{D} = \{(\mathbf{x}, y, \mathbf{m})_i\}_{i=1}^N$, in which the mass hypotheses should also be collected as features. Moreover, we can think of the mass feature as providing us with some form of domain knowledge, and, in principle, this can be exploited to improve the pNN model further: this will be discussed in detail in the next sections.

Parametric neural networks show an interesting property called *interpolation*. As anticipated at the beginning of this section, a pNN can smoothly replace a set of individual classifiers, each trained on a specific hypothesis m_i , but what does it mean at all? Assume

our signal events are generated at three, equally spaced, mass points like 200, 400 and 600 GeV⁴. A pNN capable of interpolation can predict the class of an unknown (i.e., novel) event even if this were generated at an *intermediate* mass point, like 300 or 500 GeV. This capability is crucial since it allows physicists to reduce both the number of signal hypotheses and the time required to generate the events from simulated collisions, which are tremendously expensive. Moreover, without retraining the model we can perform inference and, therefore, validate new mass hypotheses within the same analysis setup, saving time and effort. As we will see, a pNN does not always accurately interpolate the intermediate mass points. So it is important to evaluate such capability before employing the model in an HEP analysis.

2.3.2 Embedding a physics observable

As seen in equation 2.3.1, to implement a pNN we need a way to combine the features, x , with the mass, m : in the following, we assume both are shaped like vectors. To do so, we can follow three basic *feature-wise transformations* [72]:

- **Concatenation-based conditioning.** The physics parameter, m , is first concatenated (along the last dimension or axis) with the input features, x , and the result is then passed through a linear layer, which can shrink or expand the result according to the target dimensionality. The overall conditioning operation is

$$z = W[x \, m] + b, \quad (2.3.2)$$

where z is the *conditioned representation*, and (W, b) indicates the learnable weight matrix and bias vector of the linear combination. Considering a numerical computation framework, such as TensorFlow [73] or PyTorch [74], the concatenation, which is denoted by $[\cdot]$, can be implemented as `tf.concat([x, m], axis=-1)` or `torch.cat([x, m], dim=-1)`.

- **Conditional biasing.** The physics parameter is first expanded (through a linear layer) to match the dimensionality of x , which is then added in an element-wise fashion:

$$z = (Wm + b) \oplus x, \quad (2.3.3)$$

where the symbol $\oplus$ denotes element-wise addition. Here, the first part of the equation (i.e., $Wm + b$) yields a learnable *bias vector* which is then used to bias (offset) the individual input features: each element of this vector can be positive or negative, representing the relative amount of conditioning, resulting in how much the corresponding input variable should be increased or decreased such that to be best exploited by the subsequent layer(s).

⁴The GeV is a common unit measure in HEP, where $1\text{eV} = 1,609 \cdot 10^{-19}$ joule.

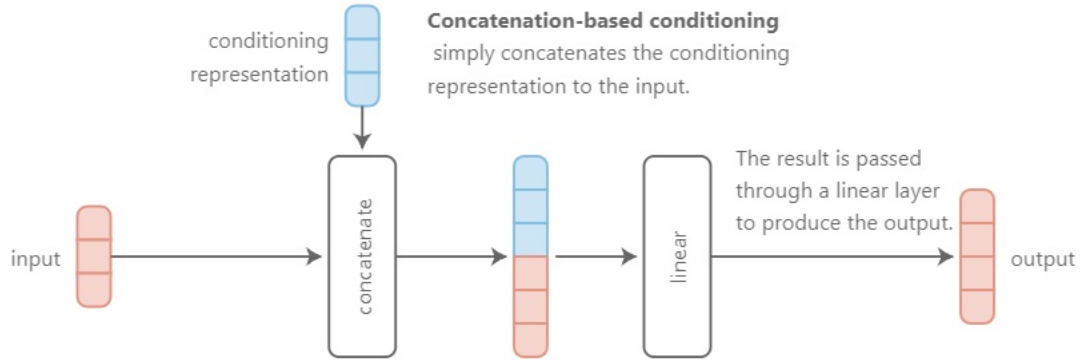


Figure 2.7: Concatenation-based conditioning: in the context of parameterized neural networks, the conditioning representation is the physics parameter like the mass feature. Figure from [72].

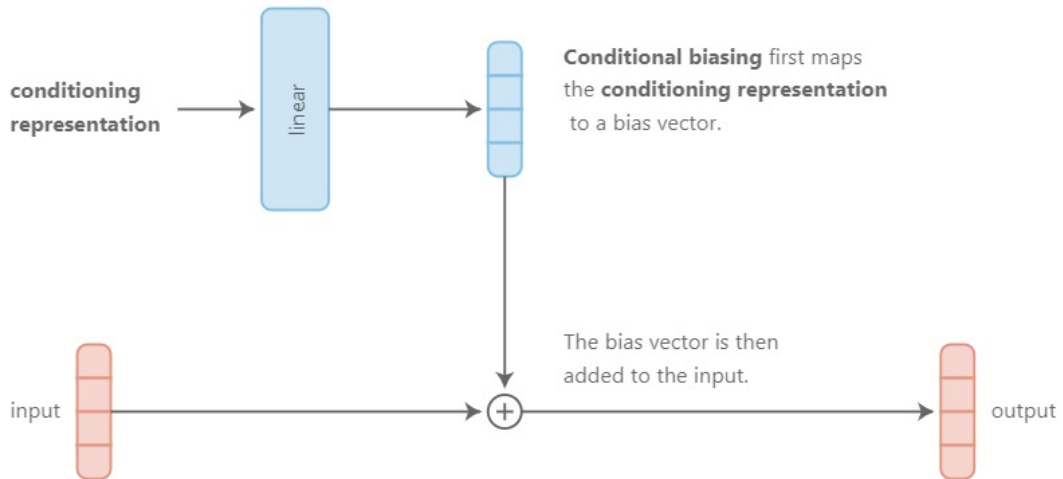


Figure 2.8: Conditional biasing. Figure from [72].

It turns out that conditional biasing is equivalent to concatenation-based conditioning, as pictured in figure 2.9. Considering the matrix-vector product in equation 2.3.2, i.e., $W[x \ m]$, this can be decomposed into two sub-products: $W_x x$ and $W_m m$, in which the concatenation of the new weight matrices, W_x and W_m , yields the original one, i.e., $W = [W_x \ W_m]$ since these are obtained by slicing W along its columns. Then, the resulting vectors are added together, in which the m -dependent one is the conditional bias that alters the representation of x .

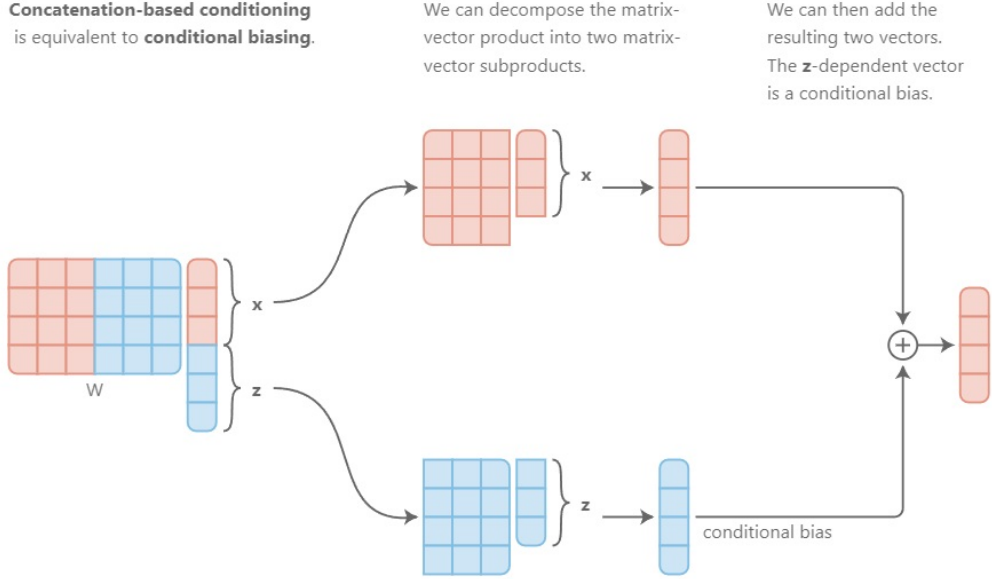


Figure 2.9: The equivalence between concatenation-based conditioning and conditional biasing. Figure from [72].

- **Conditional scaling.** The mass feature, m , is first expanded to a *scaling vector* and then multiplied element-wise to the input features:

$$z = x \odot (Wm + b), \quad (2.3.4)$$

where the symbol $\odot$ denotes the Hadamard product, which is an element-wise multiplication. In general, the scaling operation modulates (alters) the input features according to the value of the physics parameter. Intuitively, this operation can shrink the value of irrelevant features while enlarging important ones.

A special case of conditional scaling is called *sigmoidal gating*, likewise the input and forget gates [12], in which the scaling vector is squashed between 0 and 1 (by a sigmoid function) allowing to select which features are passed forward and which are ignored (i.e., forget because zeroed-out).

Both additive (concatenation: figure 2.7, biasing: figure 2.8) and multiplicative (scaling: figure 2.10) interactions are effective yet simple to implement and fast. So, which one is

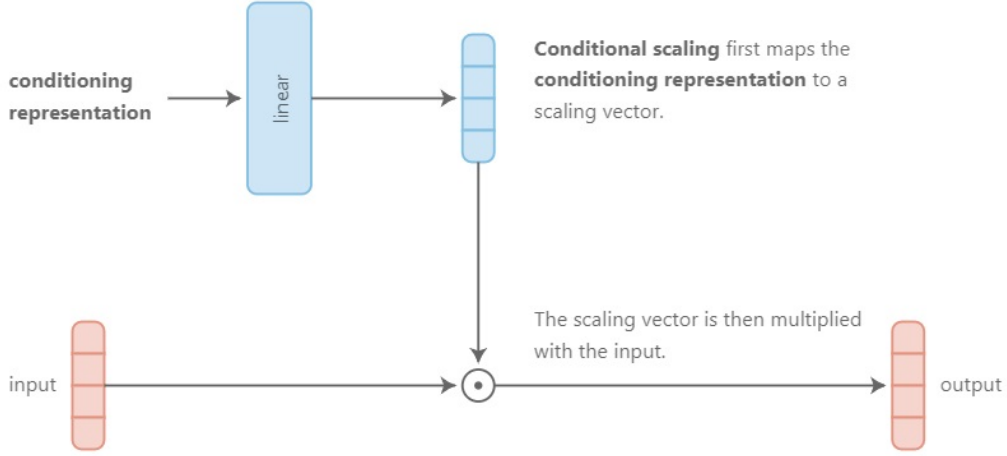


Figure 2.10: Conditional scaling. Figure from [72].

the best? Neither of these mechanisms is the best in all the situations: the scaling tends to be more effective at learning relations between the two inputs (since multiplication is the base for the dot-product, which is a measure of similarity), whereas additive interactions should be better suited for applications that involve feature aggregation (i.e., combining useful parts of each input into a lower-dimensional representation).

In this regard, we can combine both conditioning mechanisms yielding a more general one called *affine conditioning* [75], which is implemented by an affine transformation:

$$z = x \odot s_{\phi}(m) + b_{\psi}(m), \quad (2.3.5)$$

where the functions s_{ϕ} and b_{ψ} represent the conditional scaling and biasing mechanisms, respectively, along their set of learnable parameters, ϕ and ψ . The structure of the mechanism is explained in figure 2.11.

Another question regarding the conditioning is *where* it should occur. Should we condition the input layer, the output, or all the intermediate layers? Well, it depends. If our problem requires incorporating the conditioning information at the input layer, we can condition them obtaining also a parameter-efficient solution. In other cases, the conditioning information should be propagated across layers, since the hidden features may benefit from it. Lastly, it is beneficial for some tasks to only condition the model's output. In general, if we do not know this information in advance we can avoid any assumption and, therefore, condition the input and the hidden layers. For this purpose, section 2.4.6 evaluates the impact of the conditioning method and where it occurs on our signal-background classification problem.

The conditioning mechanisms described in this section act at the feature level, meaning they leverage feature-wise interactions between the conditioning representation and the conditioned model; if there is no relation between the two, the conditioning is expected to be meaningless. Although more complex interactions exist [76, 77] (e.g., ones that operate

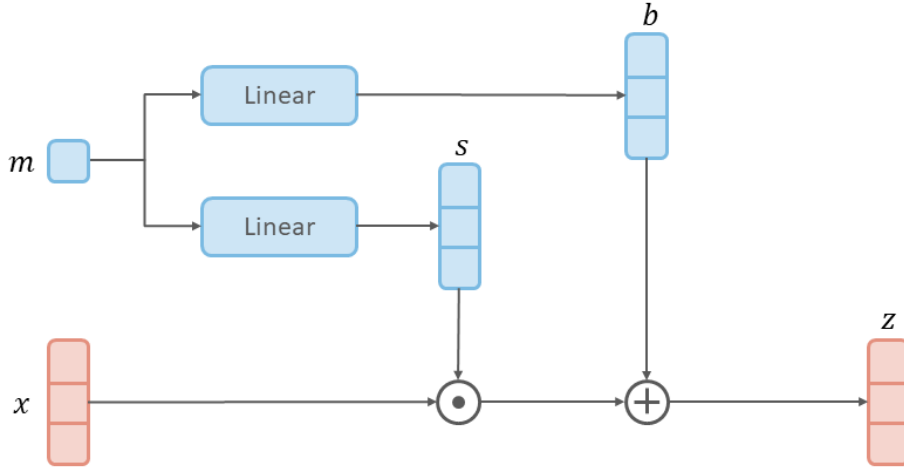


Figure 2.11: The affine conditioning mechanism first, independently, expands the physics parameter, m , twice, by two linear layers mapping it to scaling and biasing vectors, s and b , which are subsequently multiplied to the inputs, x , and finally added to the resulting vector.

at a global rather than local or feature level), feature-wise transformations are a compromise that is both effective and efficient: the number of scaling or shifting coefficients to predict scales linearly with the number of features in the model. It is worth noticing that these transformations only enforce a limited inductive bias and remain *domain-agnostic*, meaning that these mechanisms can be widely effective across problem domains.

2.3.3 Gating coefficients as feature importance

As discussed in the previous section, sigmoidal gating is a special case of conditional scaling in which the scaling coefficients can select which input feature should be propagated to the next layers according to

$$z = x \odot \sigma(Wm + b), \quad (2.3.6)$$

in which the sigmoid function, $\sigma(\cdot)$, yields gating coefficients in a $[0, 1]$ interval. If sigmoidal gating is applied to the input layer (as in the above equation), therefore on the input features, x , it turns out that these coefficients (i.e., $\sigma(Wm + b)$, which yields a $|x|$ -dimensional vector) can be interpreted as a form of *feature importance*: values close to one indicate important features, whereas coefficients close to zero depict features that are ignored because probably not relevant (e.g., redundant) for the task to solve. The learned coefficients are multiplied by the features, x , hence, “filtering” them. Moreover, it should be noted that the gating coefficients depend on the physics parameter, m , rather than the features.

In figure 2.12 we can see an example of how the gating coefficients can be interpreted as feature importance scores. In particular, we can observe that 1) for some features the coefficients are zero meaning that the respective features are completely ignored, whereas

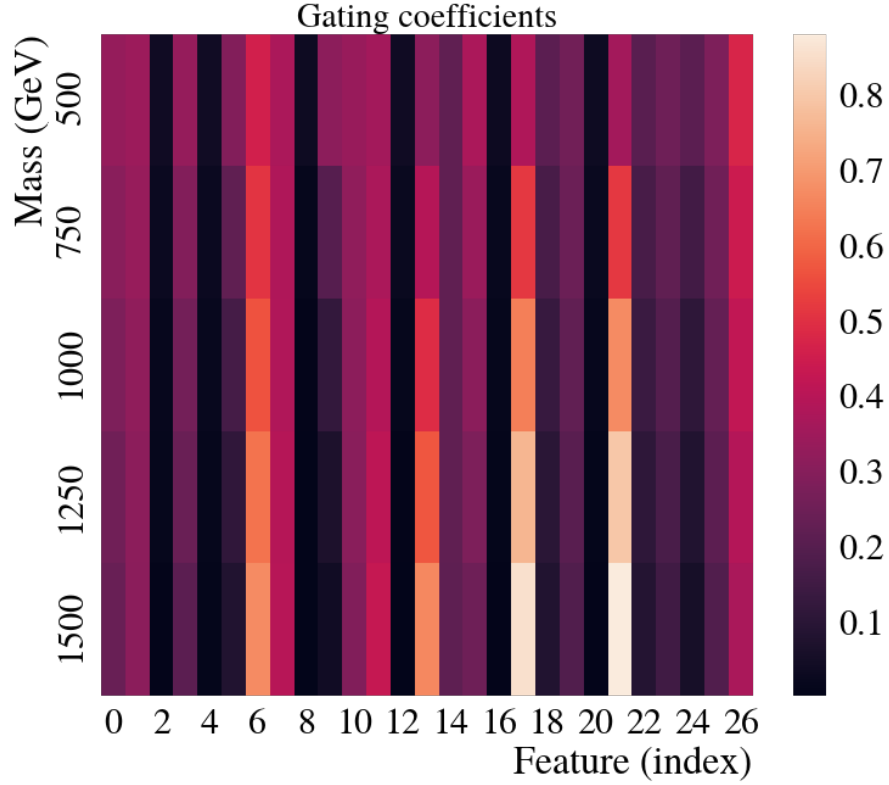


Figure 2.12: Feature importance through gating coefficients, computed by training a pNN [78] on the HEPMASS-IMB dataset [79], which is introduced in section 2.4.1. The importance is computed for 27 features aggregated over five mass hypotheses and class labels (since there were no relevant differences in the coefficients between the two classes.) Values range from zero to one, corresponding to darker colors and brighter ones.

for some others the importance quickly tends to zero as the mass increases; this can mean that such features became less and less discriminative for the classification of such events. 2) in some cases, the increase in mass yields an increase in the feature importance indicating a positive correlation whereas, for others, the correlation is negative: the importance decreases. And 3) for a few features their importance remains stable across mass points, meaning that these are somewhat independent of the mass value.

Feature importance can be a useful means to sanity check the features but also to perform *feature selection* when there are several of them and, in which, we only keep the most relevant (i.e., important) of them. As depicted by the figure 2.12 we may remove from the training data those features for which the importance is close to zero and retrain our pNN model: in such a case we should expect the same classification performance since the removed features were not relevant; in this regard, the importance of features may help us

at finding *redundant* variables that do not carry any additional information. This fact is confirmed experimentally, in which six features (**f2**, **f4**, **f8**, **f12**, **f16**, and **f20**) were removed according to their importance which was close to zero: after retraining, the model demonstrated the same classification performance without any loss in accuracy.

In conclusion, removing unused, noisy, or redundant features is also useful to achieve a more optimized model. This model is expected to be smaller (since the input dimensionality is lower), faster (because there are fewer parameters), and potentially even more robust, as both noisy and redundant features may hurt generalization.

2.3.4 Exploiting domain knowledge

Domain or expert knowledge can be integrated into a deep learning solution in various ways: at the *architecture*, *data*, or *training* level. For example, our knowledge of the problem or underlying theory may allow us to implement a novel differentiable operator, which can be included in a neural network architecture or even learned, or to design better features from the available data, or even guide the model training utilizing a better loss function or fitting procedure. These are examples of inductive biases [41] that can be embedded in a DL model, making it more efficient, hence, boosting its performance.

Learning the particle representation. The variables used to describe particles or events are usually manually engineered. The Lorentz Boost Networks (LBNs) [80] provide an effective way to autonomously learn high-level features from low-level ones like particle four-momenta vectors (also called *four-vectors*⁵), which describe the kinematics of the particle. After the learned preprocessing, the extracted features are fed to a task-specific neural network.

The LBN combines N input four-vectors into M particles, C^P , and M rest frames, C^R , through linear combinations, whose weighting coefficients are learned. Then, the Lorentz transformation

$$B = \Lambda^R \cdot C^P \quad (2.3.7)$$

applies the boost matrix Λ^R on the rest frames, boosts the particles, yielding B which represents variables like invariant masses, transverse momenta, pseudo-rapidities, and angular distances between them. Lastly, feature extraction transforms the M boosted particles (each with four components) into F one-dimensional features: the overall process is described in figure 2.13. In this example, the Lorentz transformation (from the special relativity theory) provides the inductive bias for automated feature engineering, which can be employed to solve multiple HEP problems.

⁵The four-vectors are quadruplets (E, p_x, p_y, p_z) consisting of energies, E , and momentum components, p_x , p_y and p_z , for the particle p .

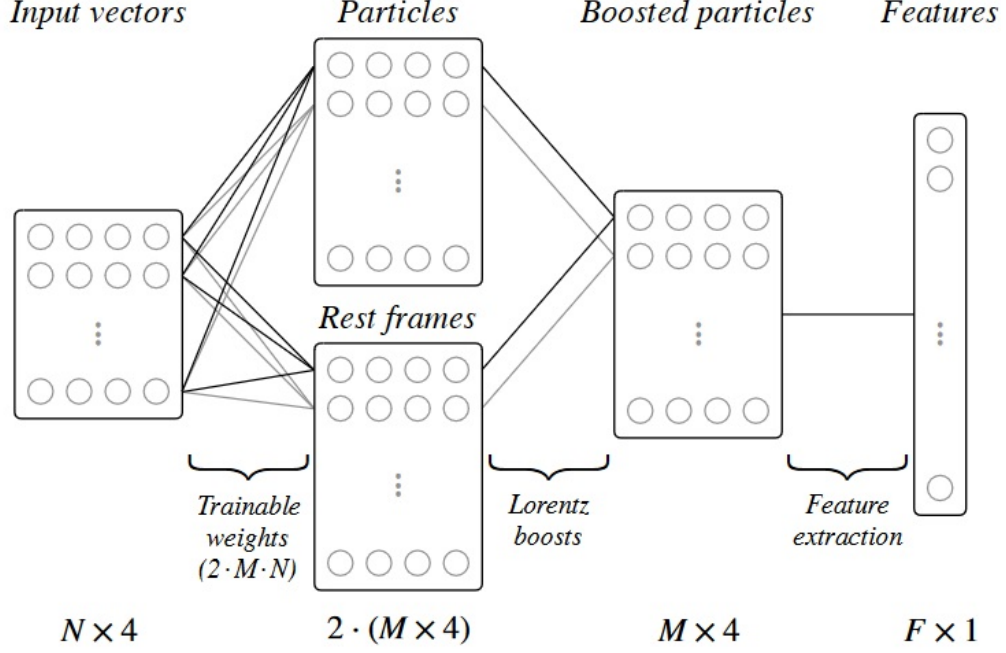


Figure 2.13: The LBN: input four-vectors are linearly combined to get particles and rest frames, which are then boosted and, lastly, a generic set of features is extracted. Figure adapted from [80].

In a similar spirit, since collision events consist of a variable-length, unordered list of particles their observables should be defined as functions of sets of particles. Energy flow networks (EFNs) and Particle flow networks (PFNs) [81] adopt a per-particle internal representation, ϕ , that is then summed over all particles, resulting in a permutation-invariant event-level representation:

$$\text{EFN} : F\left(\sum_{i=1}^M z_i \phi(\hat{p}_i)\right), \quad \text{PFN} : F\left(\sum_{i=1}^M \phi(p_i)\right). \quad (2.3.8)$$

The functions $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^\ell$ (for EFNs, $d = 2$) and $F : \mathbb{R}^\ell \rightarrow \mathbb{R}$ are both parameterized by neural networks, z_i is the energy and $\hat{p}_i$ the angular information of particle i . Moreover, p_i denotes d kinematic features, such as the particle's p_T , rapidity, and azimuthal angle. Instead, ℓ is the latent space dimensionality in which the particles are embedded. In addition, the function F can further generalize the representation obtained by summing the individual particles. The authors employ the above mappings to learn general observables to study *jets*, which are a spray of hadrons (made of quarks and gluons) produced in high-energy proton-proton collisions [82]. Since jets are composed of a varying number of particles, the above formulation can be used to obtain a fixed-length jet representation.

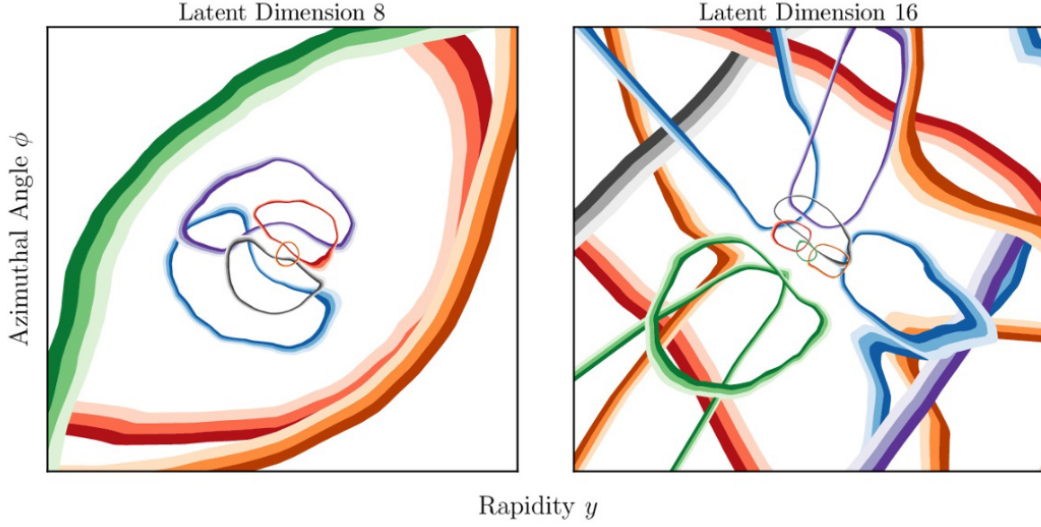


Figure 2.14: Learned filters, i.e., ϕ , for latent dimensions $\ell = 8$ and $\ell = 16$. The model appears to have learned a “pixelization” of the rapidity-azimuth plane: the pixels learned by the model are smaller near the core of the jet and larger at its edges. Figure adapted from [81].

Since the per-particle mapping, $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^\ell$, of an EFN, takes two-dimensional angular information as input, the resulting space can be easily visualized to get insights about what the EFN is learning. As shown in figure 2.14, the model projects each jet into a sort of dynamically-sized calorimetric image because each (colored) filter captures a different part of the jet kinematic in proportion to its size. EFNs and PFNs are notable examples of physics-informed automatic feature engineering, each achieving high performance and interpretability.

Assignment of the physics parameter. The mass feature (in general, the physics parameter) is only well defined for the signal samples since it represents the mass hypotheses, $\mathcal{M}$, at which the signal events are generated. This is because associating the signal-generating mass with the background events is usually not meaningful. Therefore, to train a pNN we need a safe way to assign the mass feature to the background without introducing any *label leakage*, i.e., an unwanted phenomenon in which the class label is indirectly provided to the model as input feature.

The general approach is to define a probability distribution, $p(\cdot)$, from the set of mass hypotheses and sample the mass features for the i -th background event from it: i.e., $m^{(i)} \sim p(\mathcal{M})$. The distribution can be discrete, uniform, or even Gaussian to name a few: the choice of the distribution can also be seen as a way to encode an assumption about the underlying physics. From my experiments, the discrete distribution is the best performer yet simple to define assignment strategy since the mass feature is directly sampled from $\mathcal{M}$:

i.e., $m^{(i)} \sim \mathcal{M}$, representing a Categorical distribution over a finite set of elements with equiprobable probabilities; refer to appendix A.2 for more details. Another conceivable approach is to uniformly assign the parameter in a continuous interval: $m^{(i)} \sim U(m_1, m_M)$, assuming $m_1 = \min \mathcal{M}$ and $m_M = \max \mathcal{M}$; but it is discouraged since it causes severe overfitting due to label leakage. The important point is that sampling the parameter only occurs during training for the background events. Whereas, at test time, the model is evaluated at each mass point considering the whole background and assigning it the current mass point as the mass feature.

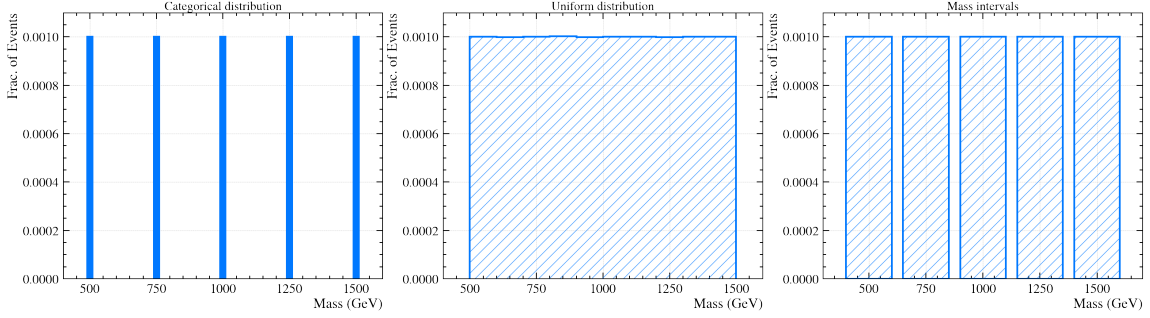


Figure 2.15: Comparison between parameter assignment strategies. The Categorical distribution (left) works only for discrete parameters, assigning a random value from its set to a background event. The Uniform distribution (middle) can cover a continuous range of values for discrete or continuous parameters. Lastly, by using intervals (right) we can discretize a continuous parameter but if these are not overlapping, some data samples will be discarded.

In principle, the mass feature can also be assigned according to the reconstructed mass of the event given the experimental signature being searched. In this case, all the background events whose reconstructed mass falls in a given mass interval are assigned the mean of that interval (or a predefined value) as the mass feature. For example, considering (150, 250) GeV to be a specific mass interval centered around 200 GeV, we can assign $m = 200$ as the mass feature for each background event whose reconstructed mass is within the interval. Figure 2.15 compares the three outlined approaches: discrete, uniform, and interval-based parameter assignment. In particular, for the last one, we can first see that the intervals are symmetric in size but also non-overlapping, meaning that some background events are discarded. Then, the width and symmetry of each interval are important design choices that should be set according to the distribution of the events' reconstructed mass, which is expected to be uneven in real scenarios.

The signal mass hypothesis is only one of the possible physics parameters that we can use to condition a neural network. Assuming a general parameter, $\rho \in \mathcal{P}$, the typical HEP workflow [83] keeps it fixed during detector simulations by construction to yield the data according to a probability, $p(x \mid \rho, y) \forall \rho \in \mathcal{P}, y \in \{0, 1\}$, which is conditional on both the parameter ρ and class label y . During the simulations, the parameter, ρ , is fixed beforehand

because it was determined by the hypothesis that should be validated with the analysis. Therefore, in the case of the parameter being the mass hypotheses, i.e., $\rho = m$, a signal event is generated according to $p(x | \rho = m, y = 1)$, where $m \in \mathcal{M}$ and $y = 1$ indicates the positive class. Instead, a background sample is yielded by $p(x | y = 0)$. Notice that in the latter probability there is no dependence on the parameter since, in the case of the mass hypotheses, it is meaningless for a background event to depend on a signal hypothesis. If we assume such probability to be a *marginal distribution* obtained by summing over the parameter, i.e.,

$$p(x | y = 0) = \mathbb{E}_{\rho \sim \mathcal{P}}[p(x | \rho, y = 0)], \quad (2.3.9)$$

it is reasonable in practice to consider a random assignment of the parameter for the background events during training, which should be sampled from a *class-specific prior* such as $p(\rho | y)$: for example a discrete distribution over $\mathcal{P}$. The prior depends on the class label and should be designed as discussed at the beginning of this paragraph. In general the prior is trivial for the signal, since the parameter is already specific to an event, but not for the background. Let's consider a background event, x_b . During training, it happens that the same x_b is associated with different values of the parameters (say $\rho', \rho'' \in \mathcal{P}$ with $\rho' \neq \rho''$), therefore the model yields predictions from the same event but conditioned on different values, which are expected to differ: i.e., $f_\theta(x_b, \rho') \neq f_\theta(x_b, \rho'')$. In the limit of the training iterations approaching a large number, the neural network would experience all possible parameter assignments for all the background events. Hence, the model would learn (converge to) the *best response on average* for a background example, x_b , which should be robust to the parameter value as if the model were marginalized: i.e.,

$$f_\theta(x_b, \rho') \approx \mathbb{E}_{\rho \sim p(\cdot | y=0)}[f_\theta(x_b, \rho)] \text{ with } \rho' \sim p(\cdot | y = 0). \quad (2.3.10)$$

If so the prediction on a random parameter, ρ' , i.e., $f_\theta(x_b, \rho')$, approximates the expected prediction averaged over the background's parameter prior, $p(\rho | y = 0)$; instead, predictions over signal samples should be fully dependent on the mass hypotheses. Since the parameter for the background is randomly assigned during training, *which value should we pick for a correct evaluation of these events?* At test time if we were sampling the parameters like the fitting phase we would end up in non-determinism, obtaining slightly different results at each sampling. To avoid that, in practice, we can replicate the background as many times as $|\mathcal{P}|$ (i.e., the size of $\mathcal{P}$) and weight the background contribution by $\frac{1}{|\mathcal{P}|}$. This is a feasible approach only if the parameter takes few discrete values, instead if continuous or large we can approximate the evaluation by just sampling the parameter one or few times for each background event. Notice that this is the typical scenario for pNNs, in which the parameter is the signal-generating mass, therefore the model should learn to average over it for the background since randomly assigned. Generally, both classes can establish an arbitrarily complex dependency on the parameter.

Balanced training. To further improve our parametric models we can leverage the domain knowledge about our data. In general, our background events can belong to P different processes⁶, and the physics parameter, if discrete⁷ or enumerable, can take at most M distinct values. This results in $P \times M \times C$ possible combinations of background processes, parameter value, and class labels (for binary classification, $C = 2$.) These possibilities allow us to organize (or *balance*) the samples within each training mini-batch in multiple ways, potentially allowing the network to model the underlying *sub-classes* (in this case, mostly the grouping by the signal generating mass and background processes), in which the data can be divided into, better: balancing aims at building mini-batches in which all sub-classes or frequencies are equally representative. We distinguish the following cases:

- **No balance.** The model is trained by experiencing the original frequency of the data sub-classes. For example, if there is a class imbalance mini-batches would reflect that imbalance: examples of the most (least) frequent class would be more (less) likely to appear in a mini-batch. Figure 2.16a.
- **Class balance.** The class labels, C , are balanced within each mini-batch, regardless of the signal and background processes. Considering two classes, we balance them by a *biased sampling* of events such that half the batch represents the positive class (i.e., the signal), and the other half the negative one (i.e., the background). A similar approach would be to weigh the loss function by sample- or class-weights. Notice that sampling with bias is as if class weights were provided. Figure 2.16b.
- **Signal balance.** Mini-batches are built such that there is an equal amount of signal events for each distinct mass hypothesis, implying that each $m \in \mathcal{M}$ is present in a batch: in real scenarios, it happens that some mass hypotheses are way rarer than others (i.e., fewer signal events pass the pre-selection phase), therefore such data samples could be missing in some mini-batches. In this case, the batch size $B = (M \times B_s) + B_b$ would contain a total of $M \times B_s$ signal samples (where B_s denotes the number of events to consider at each mass point), and B_b background events. Notice that this procedure balances only the positive class. Figure 2.16c.
- **Background balance.** Similarly to signal balance, an equal amount of background samples are allocated in a mini-batch per process, P . This results in a batch size of $B = B_s + (B_b \times P)$, where B_s is the proportion of the (unbalanced) signal and $B_b \times P$ is the total number of background events. In this case, all the background processes are always contained in a mini-batch even if some processes may be rarer than others, yielding less data, consequently having a lower chance of being sampled. Notice that with this procedure only the negative class is balanced. Figure 2.16d.
- **Full balance.** The mini-batches are built considering an equal proportion of samples given all the possible combinations of P , M , and C . For two classes the batch size,

⁶Following the HEP jargon, a process is a particle physics phenomena that results in a given decay of particles.

⁷If continuous, it can be binned to yield a finite number of disjoint intervals.

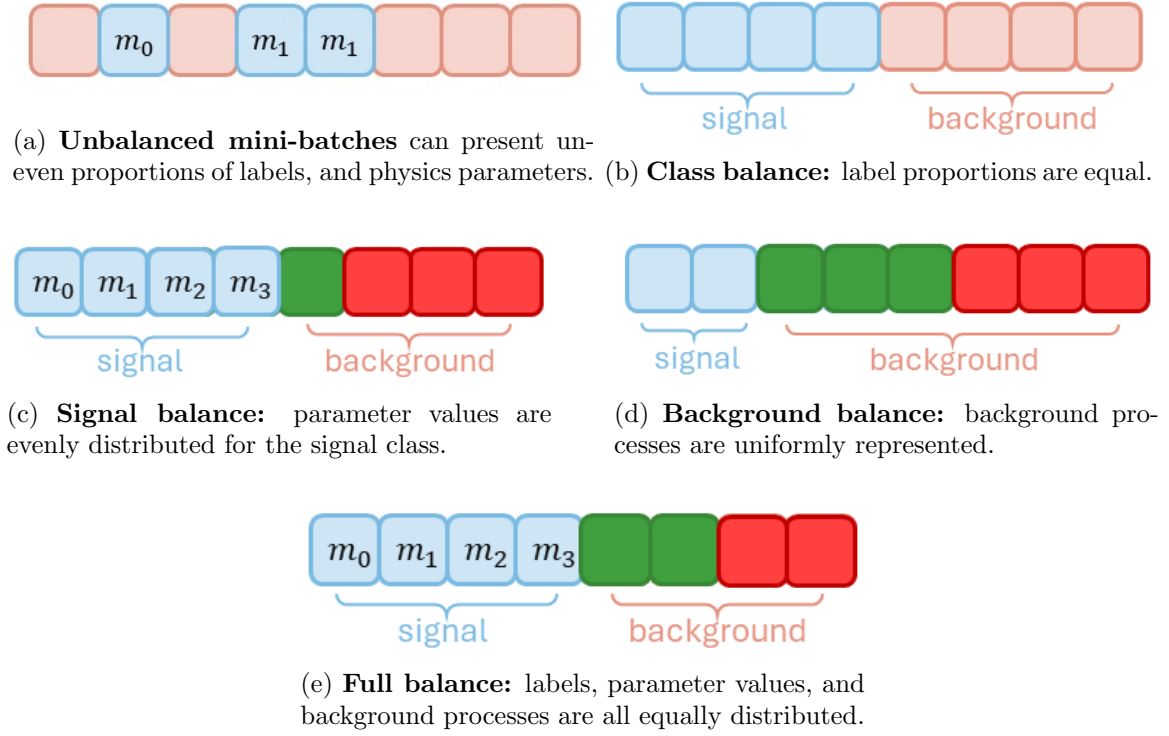


Figure 2.16: Visualizing mini-batch balancing strategies, assuming a discrete physics parameter: $\mathcal{M} = \{m_0, m_1, m_2, m_3\}$. Cyan squares (i.e., $\blacksquare$) denote signal events. Pink ones (i.e., $\blacksquare$) indicate background samples, which can be divided into different processes: here, as an example, green (i.e., $\blacksquare$) and red squares (i.e., $\blacksquare$) denote two distinct processes.

B , is divided in two balanced halves such that $B = (B_s \times M) + (B_b \times P)$, where B_s and B_b should be chosen such that $B_s \times M$ and $B_b \times P$ are approximately equal to avoid introducing class imbalance. Figure 2.16e.

Adopting the balanced training procedure can help mitigate the bias that arises from class frequencies, especially if some of them are largely more frequent than others. This can be particularly helpful if the sub-classes of the simulated collisions do not reflect the occurrence of them in real data, therefore the balanced training can help achieve a model that is robust (unbiased) against such discrepancy in events occurrence. Although the above balancing options are described by considering the mass hypotheses as a physics parameter, these can be extended to general parameters, $\mathcal{P}$.

Physics-inspired preprocessing. Raw collider data are rarely fed as input to a ML model, as these are often both high-dimensional and noisy making learning harder. Suppose an event is described at most by N constituents, i.e., a list of four vectors. These are unordered, and often we want to select only the top K of them because we need fixed-size

inputs, or transform them into images. How these are sorted would affect the model's performance [83], therefore the knowledge about the problem can be exploited to design a better sorting criterion, which can also be learned [84]. Particle constituents are often preferred to raw images of collisions, which are usually very sparse (i.e., most pixels are zero) and large in resolution. Nevertheless, if we want to exploit the power of computer vision we need to convert the four-vectors to an image of fixed size. The pixelization process can occur directly from the constituents but would yield quite non-homogeneous images. A popular preprocessing [85, 86] that normalizes the appearance of the images, is about first centering the constituents, then rotating them along a common axis, next flipping the image, and finally cropping the region of interest. Notice that this preprocessing kind assumes knowing the type of physics objects (e.g., a jet) the data describes. Often, this would make the model's performance depend on the specific preprocessing [87].

Data augmentation. In section 2.2.3 data augmentation was introduced to mitigate over-fitting by generating new samples. It can also be employed to indirectly make the model learn that its output should be *invariant* to a given transformation [83, 88]: this happens without enforcing any additional loss term. Given an operation g , a neural network is invariant to it if its output is the same regardless of the application of the operation on the inputs: i.e., if $f_\theta(x) = f_\theta(g(x))$ holds, then f_θ is invariant to g . For example, Dillon et al [89] employed both physical and anomaly augmentations to enrich the representation of events learned by a neural network by modifying the kinematics of the events. In particular, the model should be invariant to *physical augmentations*, which leaves the underlying physics unchanged by introducing symmetries in the physics system or mimicking the resolution effects of the detector. Instead, their *anomaly augmentations* (e.g., high-multiplicity, large missing energy, or transverse momentum) transform the original event to break physics invariance on purpose. Furthermore, in [88] physics-driven augmentations (e.g., rotations in the detector's coordinate system) are directly applied to raw collision images to encode detector symmetries. Note that a more comprehensive discussion about anomaly detection in HEP will be provided in the next part of this thesis.

Explicit parameterization. In some cases (e.g., Effective Field Theory's Wilson coefficients [90], and the parameterization of nuisance parameters in [91]), the dependence between the physics parameter and the function to learn is known or it can be approximated (e.g., with a polynomial). For example, let's consider the following function [90]:

$$f(x; \rho) = (1 + \rho\alpha(x))^2 + (\rho\beta(x))^2, \quad (2.3.11)$$

where $\alpha(x)$ and $\beta(x)$ are two real functions (for which we can collect data samples) of the input features, x . The overall function, $f(x; \rho)$, is parameterized by ρ on which it depends explicitly, in this case encoding a quadratic form on the parameter; this can be true for both classification and regression problems. If the target task admits an analytical (e.g., quadratic) form of the parameter or we want to ensure human interpretability, a regular pNN does not allow us to enforce an explicit dependency on the parameter since it would

learn the reliance between the features and the parameter automatically, in its hidden layers. Nonetheless, even if the pNN learned a quadratic relationship it would be hard for us to figure it out. Therefore, if the dependency is known in advance or it can be assumed, we can exploit this knowledge to simplify the learning task by directly encoding such assumption in the model. For example:

$$\hat{f}(\mathbf{x}, \rho) = \frac{1}{1 + (1 + \rho f_\alpha(\mathbf{x}))^2 + (\rho f_\beta(\mathbf{x}))^2}, \quad (2.3.12)$$

where $\hat{f}(\mathbf{x}, \rho) \in (0, 1)$ is a continuous-output classifier, and f_α and f_β are two standard neural networks that approximate $\alpha(\mathbf{x})$ and $\beta(\mathbf{x})$, respectively. As we can see from the equation, the parameter is not experienced by any neural network (hence, there is no learning) but only plugged in the functional form to agree with our domain knowledge. The model described in equation 2.3.12 is suitable for classification tasks, in which the dependency on the parameter is known (e.g., told by the underlying theory). Instead, for regression problems, we can opt for a polynomial approximation (e.g., similarly to [91]) in case the mathematical form of the dependency is not known:

$$\hat{f}(\mathbf{x}, \rho) = \exp \left[\rho \delta_1(\mathbf{x}) + \frac{1}{2} \rho^2 \delta_2(\mathbf{x}) + \dots \right], \quad (2.3.13)$$

where each function $\delta_i(\mathbf{x})$ should be suitably approximated by a neural network. To avoid incurring training to many models, one can, for example, truncate the expansion to the second order and only fit a pair of neural nets.

In a similar spirit, physics-informed neural networks (PINNs) [92] embed physics knowledge in terms of a known partial differential equation (PDE) to find an optimal numerical solution from a limited number of samples. A PINN is defined as

$$f(t, \mathbf{x}) = u_t + \mathcal{N}[u], \quad (2.3.14)$$

where the (unknown) PDE solution u_t is approximated by a neural net $u(t, \mathbf{x})$, and the non-linear differential operator, $\mathcal{N}[\cdot]$, is a problem-specific PDE providing the physics knowledge about the target task. Given initial and boundary measurement training data, $\{(t_u, \mathbf{x}_u, u)_i\}_{i=1}^{N_u}$, and collocation (or residual) points $\{(t_f, \mathbf{x}_f)_j\}_{j=1}^{N_f}$, the shared parameters between the networks $u(t, \mathbf{x})$ and $f(t, \mathbf{x})$ are learned by minimizing the following loss:

$$\mathcal{L} = \frac{1}{N_u} \sum_{i=1}^{N_u} \left(u(t_u^{(i)}, \mathbf{x}_u^{(i)}) - u^{(i)} \right)^2 + \frac{1}{N_f} \sum_{j=1}^{N_f} \left| f(t_f^{(j)}, \mathbf{x}_f^{(j)}) \right|^2. \quad (2.3.15)$$

In this way, PINNs can integrate information from measurements and PDEs, where PDEs provide a learning bias [93] as an additional loss term (i.e., the second term in the above equation). Hence, PINNs can overcome the limitations of numerical methods while satisfying physical invariants, and being data efficient. In addition, PINNs can be seen as pNNs that integrate the physics knowledge via a PDE-based loss term rather than an architectural conditioning mechanism, although limited to regressing PDE solutions.

Incorporating constraints. Neural networks are a capable class of data-driven models that often achieve high performance on the target learning task. Despite the attractive performance, NNs may violate fundamental constraints because these are unlikely to be learned while solving the underlying task. Constraints are usually driven by our domain knowledge or common sense, like constraints that ensure safety. This domain knowledge is quite improbable to be encoded in the problem we wish to solve, nor the provided dataset, therefore we should provide such information to the model separately. Considering a general constraint, $C(x, y, z) = 0$, that relies on input, x , output, y , and, eventually, auxiliary variables, z , there are two main approaches to enforce them [83]: *soft* constraints can be introduced by adding a penalty term, say C , to the loss function whose importance is controlled by a hyperparameter, λ ; and *hard* constraints can be enforced at the architecture-level by introducing custom layers encoding the knowledge about the constraint function [94]. In general, if the constraints are differentiable these can be easily introduced directly, although the target application must allow for some degree of violation. Instead, hard constraints, which are always fully satisfied so never violated, can be way more difficult to enforce especially if these cannot be written in closed form. For example, Beucier et al. [94] consider enforcing linear analytic constraints, described by a linear system of rank n . Their approach consists of augmenting a neural network architecture with *fixed-parameters* constraints layers, which take as input the input features along the output of the NN, able to solve the system of equations describing the constraints such that the final output respects them.

2.3.5 Interpolation and extrapolation

In principle, a parameterized model can smoothly *interpolate* intermediate values of the physics parameter that were not seen during training. Taking as an example the signal mass hypotheses $\mathcal{M}$, if we take any two adjacent pairs (m_i, m_{i+1}) such that $m_i \in \mathcal{M}, \forall i < |\mathcal{M}| - 1$, e.g., (750, 1000) GeV, the pNN is expected to interpolate, i.e., correctly predict, intermediate points of this mass interval like its mean, i.e., $(m_i + m_{i+1})/2$, for example. Such a mass point was never experienced during training, therefore its interpolation implies a generalization capability able to relate the novel parameter value to the input features. Thus, the model’s ability to generalize should be *twofold*: the pNN is requested to perform well on novel samples belonging to the known parameter values and to predict new events that belong to unseen parameter values correctly. This means that a pNN capable of good interpolation should provide more accurate outcomes compared to the ones that would be obtained by (linearly) interpolating the results of $|\mathcal{M}|$ individual models, instead. Furthermore, parameterized models can, to some degree, handle parameters whose values fall outside the range experienced during training: in such a case, we talk about *extrapolation* capability.

Interpolation is an important property for parameterized networks, allowing them to effectively replace a set of individual classifiers each trained on a specific parameter value. In principle, the pNNs can generalize on the entire continuous range of parameter values enabling this class of models to be more *data efficient*. This ability has another important

implication, especially for the HEP domain: generating signal events for the various mass hypotheses through accurate Monte Carlo simulations is costly, therefore the number of mass points to consider is often limited to a few hypotheses hence making the search of the new particle more coarse. Interpolation in pNNs can either help reduce the number of signal hypotheses, speeding up the event generation phase, since the model would interpolate them, or make the new physics search more fine-grained by generating fewer signal events at novel intermediate mass points. Indeed, the data should be simulated at suitable mass points to probe the potential new physics.

2.3.6 Handling continuous and multiple parameters

In section 2.3.4 we discussed multiple ways in which a single discrete physics parameter, ρ , can be assigned to our parameterized network, f_θ . For equation 2.3.10 to hold, we assumed an infinite number of training iterations to cover all possible combinations of the parameter value and background events. If we consider the simplest case of one discrete parameter, i.e., $\rho \in \mathcal{P}$, the equation can be rewritten as

$$\hat{y}_b = \mathbb{E}_{\rho \sim \delta(\mathcal{P})} [f_\theta(\mathbf{x}_b, \rho)] = \frac{1}{|\mathcal{P}|} \sum_{\rho \in \mathcal{P}} f_\theta(\mathbf{x}_b, \rho), \quad (2.3.16)$$

where the predicted class $\hat{y}_b$ for the background event $\mathbf{x}_b$ can be obtained by averaging the predictions of the model over all possible values, $\mathcal{P}$, of the parameter. Since discrete, we assumed a discrete distribution over the parameter, i.e., $p(\rho \mid y = 0) = \delta(\mathcal{P})$, but, in principle, this can be further generalized to arbitrary distributions which can also cover the case of a single *continuous* parameter:

$$\hat{y}_b = \mathbb{E}_{\rho \sim p(\cdot \mid y=0)} [f_\theta(\mathbf{x}_b, \rho)] = \int_{-\infty}^{\infty} f_\theta(\mathbf{x}_b, \rho) \cdot p(\rho \mid y = 0) \, d\rho, \quad (2.3.17)$$

which involves an intractable integration over the value range of the parameter. Hence, we incur a large computational cost if the discrete set, $\mathcal{P}$, is very large or if ρ is continuous. To avoid that we can resort to a Monte Carlo approximation of both equations, in which we sample only M values of the parameter such that

$$\hat{y}_b \approx \frac{1}{M} \sum_{\rho \sim p(\cdot \mid y=0)} f_\theta(\mathbf{x}_b, \rho). \quad (2.3.18)$$

In this way, during training, we incur a computational cost to predict a background sample that is independent of the size of the physics parameter space although it is still $O(M)$, where $M < |\mathcal{P}|$. Therefore, by tuning M we can trade accuracy for training speed. It is worth noticing that the assignment strategy discussed in section 2.3.4, corresponds to a special case of the above equation which can be recovered by setting $M = 1$. Moreover, this formulation also explains why those assignment strategies [78, 75] work at all. Lastly, these equations only tell us to modify the prediction step of background events during the

training phase, since the parameter is already assigned for the signal. A simple practical solution is replicating the background events of a mini-batch M times, associating them with different parameter values, and then computing the loss and gradient descent step as usual.

The above idea can be extended to a setting where we want to condition our model on multiple physics parameters. Let's start by considering the case of K discrete parameters, $\rho_1, \dots, \rho_K \in \mathcal{P}_1 \times \dots \times \mathcal{P}_K$, in which each parameter, $\rho_i \in \mathcal{P}_i$ (where $i \in 1, \dots, K$), has its own set of values. Hence, the equation 2.3.16 now writes as

$$\hat{y}_b = \frac{1}{|\mathcal{P}_1| \times \dots \times |\mathcal{P}_K|} \sum_{\rho_1 \in \mathcal{P}_1} \dots \sum_{\rho_K \in \mathcal{P}_K} f_\theta(\mathbf{x}_b, \rho_1, \dots, \rho_K), \quad (2.3.19)$$

where we need to average on each single parameter space with a computational cost of $O(|\mathcal{P}_1| \times \dots \times |\mathcal{P}_K|)$ for the enumeration, which can easily become infeasible. Similarly, for the continuous case, we would need to compute K integrals which is excessive. Hence, as done in equation 2.3.18 we can use a Monte Carlo average but, in this case, we need to at least sample a value for each parameter, thus: $M = \alpha K$, where $\alpha \in \mathbb{N}$ is a positive integer. Since a continuous parameter is governed by a probability distribution, we may consider *multivariate distributions* over the parameters that can also capture relations between them. In this way we are *not* saving up computation but rather making the estimate more efficient since by sampling the parameters in regions where these are more representative (compared to a uniform sampling), we can expect the training to converge to an equally good minimum but with a smaller value of M . In general, this idea can also be applied to the previous cases (single, multiple, discrete parameters) by changing the sampling strategy from uniform to an *informed* one: to make the sampling informed we can consider the density of the parameter(s) induced by our dataset and, also, the inclusion of domain knowledge from the underlying theory.

From the above perspective, we may consider the sampling (i.e., assignment) of the parameter(s) for the background events as a sort of *class-specific* data augmentation, since only one class is augmented, which may be interpreted as providing the network with the same events, multiple times, but associated with different parameter values as if to inform the model about the physical plausibility of the given feature-parameter pair. If so, there would be pairs $(\mathbf{x}_b, \rho)$ that are more physically relevant and others that are less. Therefore, these pairs may enable the pNN to learn the physics that relates both the features and class labels with a given parameter value.

2.3.7 Relation to multi-task learning

Recall that our parameterized model is trained on a dataset of the form: $\mathcal{D} = \{(\mathbf{x}, y, \rho)_i\}_{i=1}^N$, in which each pair $(\mathbf{x}^{(i)}, y^{(i)})$ is assigned a (discrete) value of the physics parameter, $\rho^{(i)} \in \mathcal{P}$. If we were “slicing” the dataset, $\mathcal{D}$, by selecting all the features and targets associated with each unique parameter value, we would get a set of $|\mathcal{P}|$ datasets:

$$\mathcal{T} = \left\{ \mathcal{D}_{\rho_i} = \{(\mathbf{x}, y, \rho)_j : \rho_j = \rho_i\}_{j=1}^N : \rho_i \in \mathcal{P} \right\}, \quad (2.3.20)$$

in which each dataset slice, $\mathcal{D}_{\rho_i}$, is associated to a parameter value, ρ_i . The above equation represents the set of all tasks, $\mathcal{T}$, in which each $\mathcal{T}_i = \mathcal{D}_{\rho_i}$ defines a single *task* we wish our model to solve: notice that these tasks are also related to each other. Since we want our pNN to solve all the tasks in $\mathcal{T}$ *jointly*, we refer to this learning scenario as *Multi-Task Learning* (MTL) [95].

MTL is a learning paradigm aiming at leveraging the knowledge contained in a task to learn from multiple related tasks simultaneously by a shared model [96, 97], which is our parameterized network: precisely, the pNN solves a multi-task supervised classification problem. Instead, learning a classifier on a single $\mathcal{D}_{\rho_i}$ is considered as single-task learning (STL). Compared to STL, MTL can leverage more data since learning from all tasks, $\mathcal{T}$, obtaining a more accurate learner for each task which also improves generalization: utilizing the data from all tasks can be seen as a form of data augmentation. In particular, we are in the case in which the different tasks lie in the same feature space, therefore the setting is called *homogeneous-feature* MTL [97]. Moreover, the parameterized neural network can be seen as an instance of *hard parameter sharing*, in which the model’s weights are shared among all the tasks minimizing a common loss function: there is a single model that learns from all the $\mathcal{T}_i$ at the same time. Instead, in *soft parameter sharing* different tasks have individual task-specific models with distinct weights, which can share a common backbone, plus a term added to the loss function that penalizes the distance between the model parameters of different tasks [96].

In the context of HEP, another conceivable MTL architecture is what we call the set of *individual networks*, in which each classifier is trained on a specific task, $\mathcal{T}_i$, hence this architecture can be seen and motivated as a soft parameter sharing alternative to the pNN, in which there is no shared part (i.e., the backbone) among the networks. Compared to pNNs, the individual classifiers may achieve similar classification performance but at the expense of using more training data and losing the interpolation capability, since there is no shared information among the tasks.

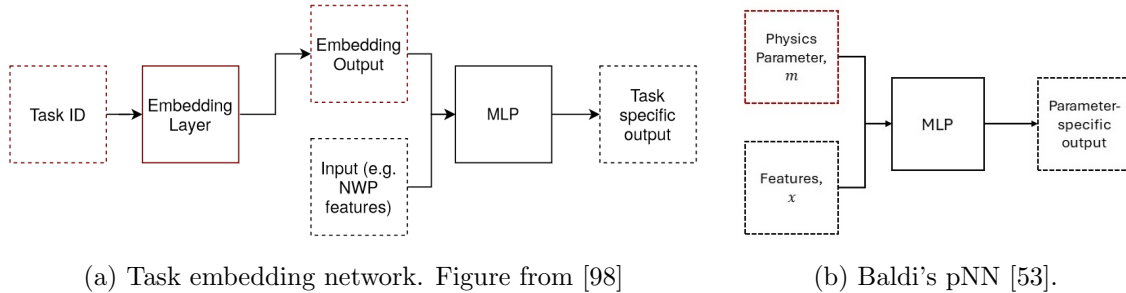


Figure 2.17: The parameterized neural network introduced by Baldi et al., which concatenates the parameter with the features, can be seen as an instance of the task embedding network, which is a MTL architecture.

Following the MTL paradigm, the idea of concatenating the *task embedding* (i.e., learning a dense vector that encodes an integer identifying the current task) with the input fea-

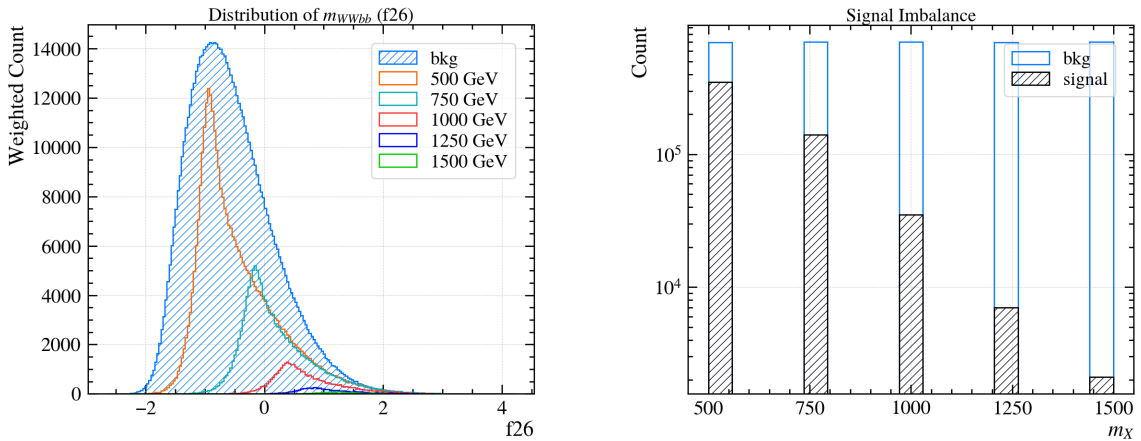
tures takes the name of *task embedding network* [98], which were mostly used for regression problems: we can notice how such network architecture resembles the Baldi’s pNN formulation [53], except for the employed embedding layer; the figure 2.17 compares the two architectures. In another example, Oeltz et al. study parameterized models for regression problems in finance [99], proving that as the number of tasks increases the complexity of a parameterized network decreases, resulting in lower overfitting compared to a STL model. Increasing the number of tasks can also be beneficial for interpolation, since it can induce the network to better fit the missing tasks (or physics parameters, in the case of HEP).

2.4 Experimental results

This section presents and discusses the results of our experiments [78, 75] about applying parameterized neural network variants on the **HEPMASS-IMB** dataset [79] (a more challenging version of **HEPMASS**), which is introduced in the next section.

2.4.1 The HEPMASS imbalanced dataset

The **HEPMASS-IMB** dataset is derived by *imbaling* (i.e., removing data samples to create a discrepancy between examples belonging to the same group or class) the **HEPMASS** dataset [100], which was used to design the original pNN [53]. In particular, we purposely make the learning problem associated with the dataset harder by creating a double imbalance, in which both the class label and mass feature (the physics parameter, m_X) are uneven as we can see in figure 2.18: the dataset is now less ideal, being closer to the real world.



(a) Resulting distribution of the reconstructed mass after having imbalanced **HEPMASS**. The background has been weighted by $1/5$, for visualization purposes.

(b) Imbalance of all the signal mass hypotheses: log-scale. We can see that the imbalance ratio grows as m_X increases, for a maximum difference of a few orders of magnitude.

Figure 2.18: Overall class and mass imbalance of **HEPMASS-IMB**.

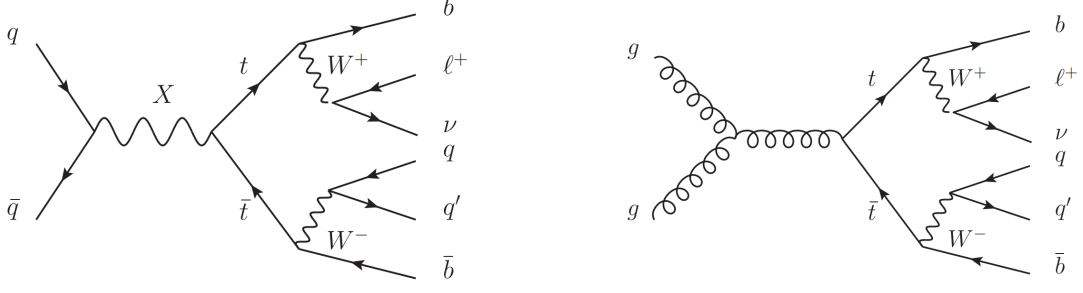


Figure 2.19: The Feynman diagrams depicting the production and decay of the hypothetical particle, $X \rightarrow t\bar{t}$, and SM background process of top quark pair production [53]. In both cases, the $t\bar{t}$ pair decays to a single lepton, ℓ , a neutrino, ν , and quarks, q and b .

The original HEPMASS dataset depicts the search of a hypothetical particle X with an unknown mass. Since the particle’s mass is not known, five signal mass hypotheses (in GeV) were assumed: $\mathcal{M} = m_X = \{500, 750, 1000, 1250, 1500\}$, meaning that the signal events were generated according to these five mass points. The particle X decays into a $t\bar{t}$ pair, and the final state consists in the most probable decay product: $X \rightarrow t\bar{t} \rightarrow W^+bW^-\bar{b} \rightarrow qq'bl\nu\bar{b}$. The dominant background considered for this specific signal is the Standard Model $t\bar{t}$ production, identical in the final state but different in kinematics due to the absence of the X resonance: i.e., $t\bar{t} \rightarrow W^+bW^-\bar{b} \rightarrow qq'bl\nu\bar{b}$. The Feynman diagrams showing the signal and background processes are presented in 2.19. Then, a comparison between HEPMASS and HEPMASS-IMB is provided in table 2.1, in which we can observe the ideal nature of the former one.

Characteristic	Dataset	
	HEPMASS [100]	HEPMASS-IMB [79]
Number of samples	7M (+ 3.5M test)	4M (+ 3.5M test)
Class imbalance (%)	None	13 (signal) / 87 (bkg)
Signal imbalance	None	up to $166\times$
Mass hypotheses	5 (equally spaced)	//
Number of variables	27	//
Background distribution	discrete	//

Table 2.1: Datasets comparison. We made the HEPMASS dataset more difficult by removing signal events from the training set to induce a double imbalance, mimicking more typical HEP scenarios where the signal is increasingly scarce.

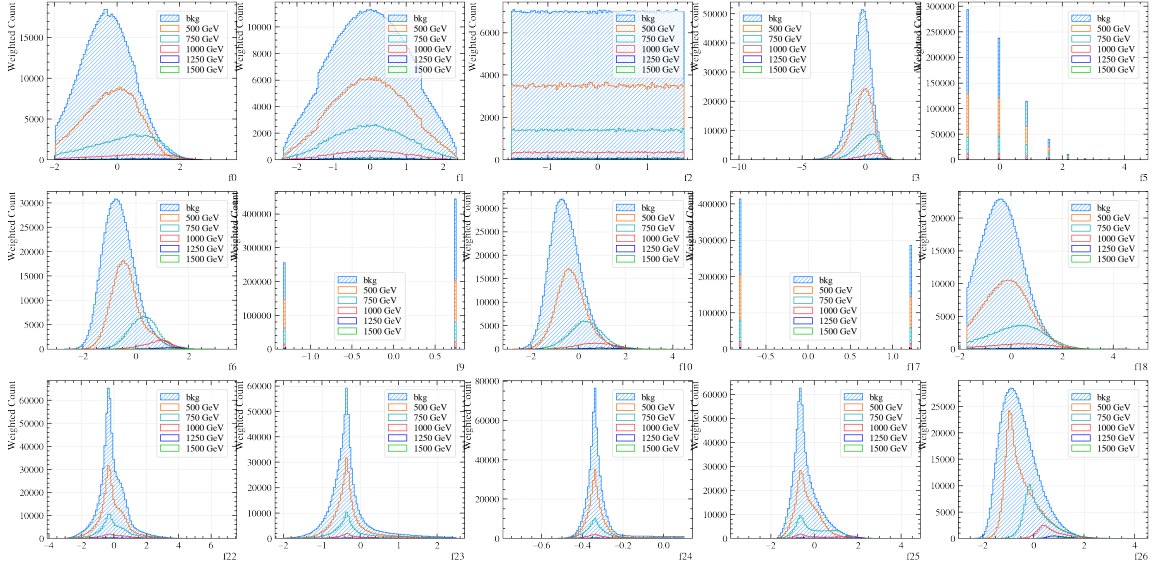


Figure 2.20: Distribution of selected training features of **HEPMASS-IMB**. For visualization purposes, the background contribution is weighted by $1/5$.

The dataset contains 27 features (excluding the mass feature, m), which are normalized to have zero-mean and unitary variance, depicting low-level kinematic variables (i.e., the four-vectors of the reconstructed events which include the magnitude and angle of the missing transverse momentum) along with high-level features obtained by combining the low-level ones to approximate the invariant masses of the intermediate physics objects of the decay. In figure 2.20 we can observe that the distribution of the signal at 500 GeV is the closest to the background, revealing that these events are the most difficult to classify which is confirmed by our evaluation of classification performance: see section 2.4.4. Moreover, as we can see in figure 2.21 there are some strong linear correlations among the features (from **f0** to **f25**), the reconstructed mass (**f26**), and the mass feature (m_X ; i.e., the **mass** column in **HEPMASS**). This can mean that the mass features can be (partially) determined from the reconstructed mass and that it can also provide some additional information during training. Finally, there are only weak correlations w.r.t. the class label.

The main reason for making the **HEPMASS** dataset harder is that it is more suited for demonstration than for benchmarking purposes since the classification performance already saturates to a high value even for simple, almost shallow, neural networks. Instead, real-world datasets are way more challenging: 1) having tens of mass hypotheses, which are often unevenly distributed; 2) multiple background processes, each contributing differently; and 3) the distribution of the events shows a long-tail shape, meaning that these can be quite numerous either at low or high mass points whereas, at the opposite side, there can be (very) few samples. In this regard, research about parameterized neural networks needs

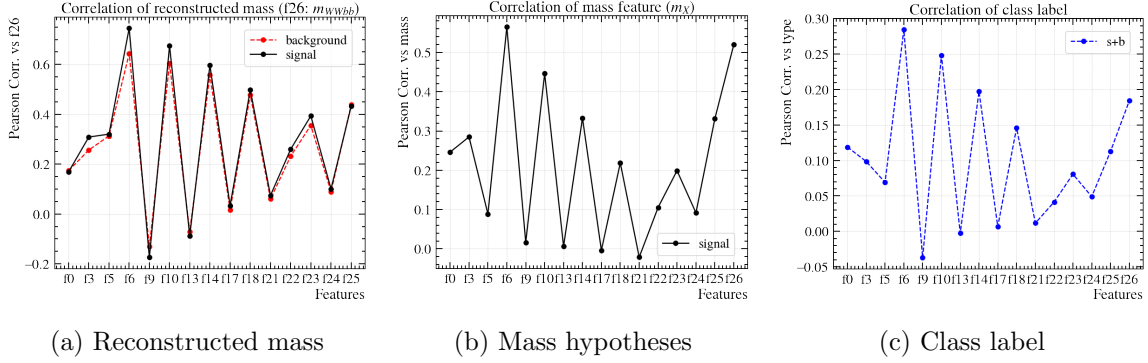


Figure 2.21: Linear (Pearson) correlations of some training variables of HEPMASS-IMB. Most features that do not correlate with the target variable are omitted for visualization purposes.

more complex and comprehensive datasets with various data modalities (e.g., images), multiple background processes and mass hypotheses, different physics parameters on which the conditioning can happen, but also describe real scenarios sufficiently either for classification, regression, and interpolation.

2.4.2 Evaluation metrics

The conducted experiments were evaluated through common classification metrics, such as the AUROC (Area Under the Receiving Operating Characteristic curve), and AUPR (Area Under the Precision-Recall curve). Moreover, to quantify the excess of classified signal over the background we designed a variant of the popular approximate median significance (AMS) formula [101, 44], called the *significance ratio*, which retains the same scale over the mass hypotheses.

AUROC. The Receiver Operating Characteristics (ROC) curve [102] is useful for visualizing and comparing the performance of classifiers. The ROC is a two-dimensional graph showing the trade-off between the *true positive rate* (TPR or *recall*) and the *false positive rate* (FPR), which can be interpreted as benefits against costs. The TPR is the ratio of true positives (TP , i.e., all the positive instances classified as positive, or as a signal in our case), i.e., $TPR = \frac{TP}{P}$. Whereas, the FPR is the ratio of the negative instances, FP , (e.g., background events) *incorrectly* classified as positive, i.e., $FPR = \frac{FP}{N}$: P and N denote, respectively, the total number of positive and negative samples. Regarding the HEP domain, the ROC can be interpreted as the *signal efficiency* (y -axis) against the *background efficiency* (x -axis) in terms of how much signal is retained when considering a certain fraction of the background. Otherwise, we can consider the *background rejection* (i.e., $= 1 - FPR$; also called *specificity*): how much signal is retained at a certain discard of background. An example ROC curve can be seen in figure 2.22a, where the following can be noticed.

- The point at the lower left corner (0, 0) represents the classification strategy of never

predicting a positive class: regardless of the input example, the class will always be negative;

- The opposite strategy of unconditionally issuing a positive class happens at the upper right coordinate $(1, 1)$, instead;
- The point at $(0, 1)$ represents perfect classification performance since all the positive samples would be correctly classified while making no mistakes about the negative ones.

Therefore, we aim to achieve ROC curves that approach the northwest corner, i.e., $(0, 1)$, as closely as possible. Moreover, the diagonal line (depicted in the figure by a dashed gray line) represents the strategy of randomly guessing a class label: even if we move along the line such that to increase the TPR, its FPR will increase as well because this classifier has no information about the class. Finally, any point of the curve that appears in the lower right triangle represents a classifier that performs worse than random since it mismatches the positive class for the negative; by negating it, the curve is flipped to the upper left triangle.

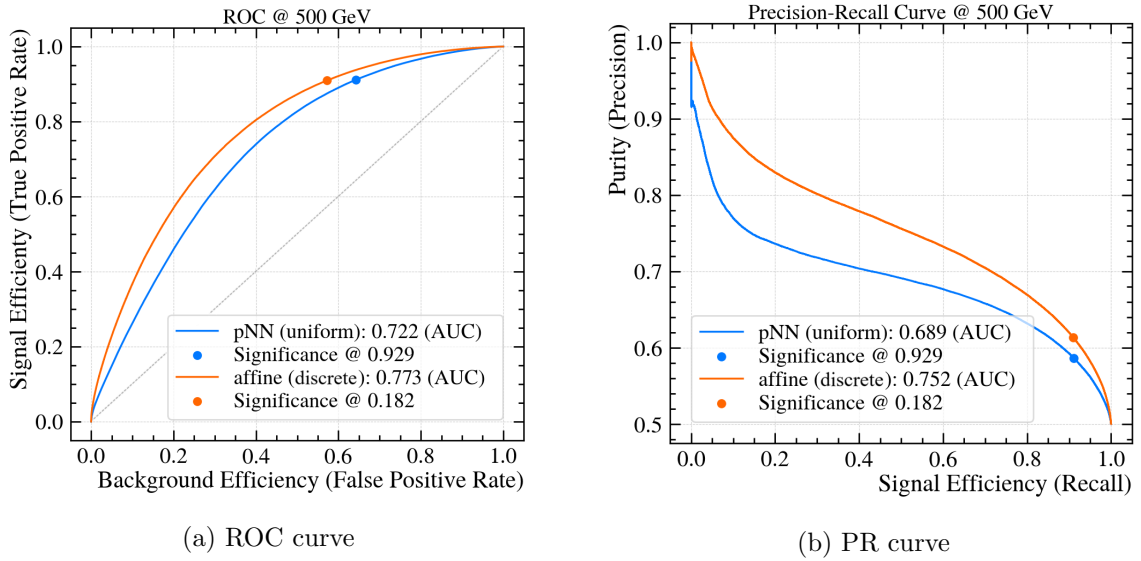


Figure 2.22: ROC and PR curves evaluated at $m_X = 500$ GeV on HEPMASS-IMB, in which a **pNN (uniform)** is compared to an **AffinePNN (discrete)**. The words **uniform** and **discrete** denote the parameter assignment strategy: refer to section 2.3.4; the two models are defined in the appendix tables A.1 and A.2. As we can see, the AffinePNN is always superior to the pNN for all classification thresholds. Moreover, the dots on the respective curves denote the threshold that maximizes the significance ratio: defined in the next paragraph. Hence, the thresholds (0.929 and 0.182) determine the working points of the respective models that best classify the signal according to the significance ratio metric.

A ROC can be built from either discrete or continuous classifiers. A discrete classifier has a binary output that represents the class label, instead a continuous one can either output probability or scores. In the latter case, the ROC is built by thresholding the continuous output varying it from $-\infty$ to ∞ , thus tracing a smooth line. In any case, the ROC is built from a finite number of data samples, therefore it is a step function that gets smoother and smoother as the number of examples approaches infinity.

Comparing two classifiers through their ROC curves can be difficult when these are interleaved (unlike figure 2.22a, in which the blue one is always smaller than the orange curve), i.e., neither of the two completely dominates the other. Hence, it would be quite handy to compare a single scalar representing the expected performance: such value is the *Area Under the Curve* (AUC) of the ROC, also abbreviated to AUROC. The AUC represents the portion of the area of a unit square, therefore its value is between zero and one, where an AUC of one is associated with perfect classification performance. Furthermore, the AUC can be interpreted as the expected probability that the classifier will score a randomly chosen positive instance higher than a random negative sample [102, 103].

Lastly, ROC curves are insensitive to changes in class distribution, meaning that if the proportion of positive to negative instances changes (e.g., in the test set), the ROC will not change. This is a useful property, especially in HEP, since the class distribution for a mass point is likely to be diverse from the others.

AUPR. The Precision-Recall curve (figure 2.22b) compares the signal efficiency (recall) with what we call the purity (or *precision*): the fraction of examples predicted as positive that truly belong to the positive class, which, in the HEP domain, can be interpreted as the number of the true signal divided by the number of events classified as signal (which also contains misclassification of the background). In PR space better curves tend to the upper-right corner. According to [104], the PR curve is preferred over the ROC for classification tasks characterized by a large skew in the class distribution. For example, if the negative samples greatly exceed the positive ones, a large change in false positives will occur, which can lead to a small change in FPR. Therefore, the ROC slightly changes, whereas the precision can still capture such difference: in HEP this can happen when a background process is significantly more numerous than a signal one. Hence, inspecting the PR curve can provide us with differences among models that were not obvious in the ROC curve. Moreover, the ROC and PR curves are related to each other. For example, a curve dominates in ROC space if and only if it also dominates in PR space. Moreover, models trained to optimize the AUROC are not guaranteed to maximize the area under the PR curve (called *AUPR*). Finally, a point on the ROC can be mapped on the PR curve as well, therefore it is possible to translate a curve in ROC space to PR space and vice-versa: this can be observed in figure 2.22, in which the classifiers obtained at the optimal threshold is mapped on both ROC and PR curves, allowing to understand the achieved TPR, FPR, and precision. As we can see, if we change the working point of the Affine pNN such that to have the same signal or background efficiency of the pNN (i.e., by moving it along the x -axis, or the y -axis), the affine model can always achieve a superior background rejection or signal efficiency,

meaning that the classified data sample will be more pure.

Significance ratio. Unlike a discrete binary classifier, which directly outputs a class label, a continuous-output one needs to be thresholded to determine the predicted class: usually, scores or probabilities larger than the threshold represent the positive class. According to both the domain and problem requirements, one way to determine the threshold is by fixing a target value of either TPR, FPR, or precision, and then tracing the point on the respective curve (ROC or PR) to determine the achieved rate on the other axis: taking the ROC in figure 2.22a as example, for our classifier we may set a target TPR of about 90% therefore we have fixed the y -axis, discovering that at that value the model achieves a FPR of about 60%. If this satisfies us, we can compute the score used when tracing the ROC to determine the classification threshold. This method is simple and may work well if we have a target value of either signal efficiency or background rejection to match.

In signal-background classification, though, we often care about maximizing the excess of the signal hypothesis over the background. This is quantified by computing the *approximate median significance* (AMS) [101, 44] as follows:

$$AMS(t) = \frac{s_t}{\sqrt{s_t + b_t}}, \quad (2.4.1)$$

in which s_t and b_t are the (weighted⁸) number of *true* signal and true background events, respectively, that passed the classification threshold t : i.e., $s_t = w_s \cdot |\{f_\theta(\mathbf{x}^{(i)}, \mathbf{m}^{(i)}) > t \wedge y^{(i)} = 1\}_{i=1}^N|$ and $b_t = w_b \cdot |\{f_\theta(\mathbf{x}^{(i)}, \mathbf{m}^{(i)}) < t \wedge y^{(i)} = 0\}_{i=1}^N|$, where w_s and w_b are class or per-sample weights. By running this formula over all possible threshold values, we can find the optimal classification threshold that determines the positive (signal) class, which in jargon is called the *best cut*: $t^* = \arg \max_t \{AMS(t)\}$. The AMS if correctly weighted can represent a statistically significant measure, whose value still depends on the total number of signal and background events present in our dataset. Hence, without any per-event weighting, the scale of the AMS will be different for each mass hypothesis (since each mass point is as if selecting a portion of the dataset with a different number of samples), making the per-mass AMS difficult to compare to each others preventing to fully understand the behavior of the parameterized model on the various mass points.

To avoid this issue, we designed a new metric called the significance ratio [78], σ_{ratio} , that is normalized in $[0, 1]$ thanks to a per-mass normalization constant, being easy to interpret and compare. The significance ratio for a given mass hypothesis is defined as the ratio between the best (maximum) AMS by the largest possible significance, thus:

$$\sigma_{\text{ratio}}(m) = \frac{AMS_m(t^*)}{s_{\text{max}}^m / \sqrt{s_{\text{max}}^m}}, \quad (2.4.2)$$

where AMS_m considers only events associated with a given mass hypothesis, by adding the condition $\mathbf{m}^{(i)} = m$ to determine s_t and b_t ; notice that this term is computed at the best

⁸To keep things general, the weighting refers to balancing the class distribution rather than introducing cross-section weights.

threshold, t^* . The ratio $s_{\max}^m / \sqrt{s_{\max}^m}$ represents the *ideal significance* for hypothesis m , in which perfect classification is assumed since all the background is rejected (i.e., $b_t = 0$), and all the signal at mass m is correctly predicted. Dividing by the constant enables each $\sigma_{\text{ratio}}(m)$ to have the same numerical scale for all mass points, $m \in \mathcal{M}$, allowing them to be comparable against each other. Moreover, the best models should approach a significance ratio of one.

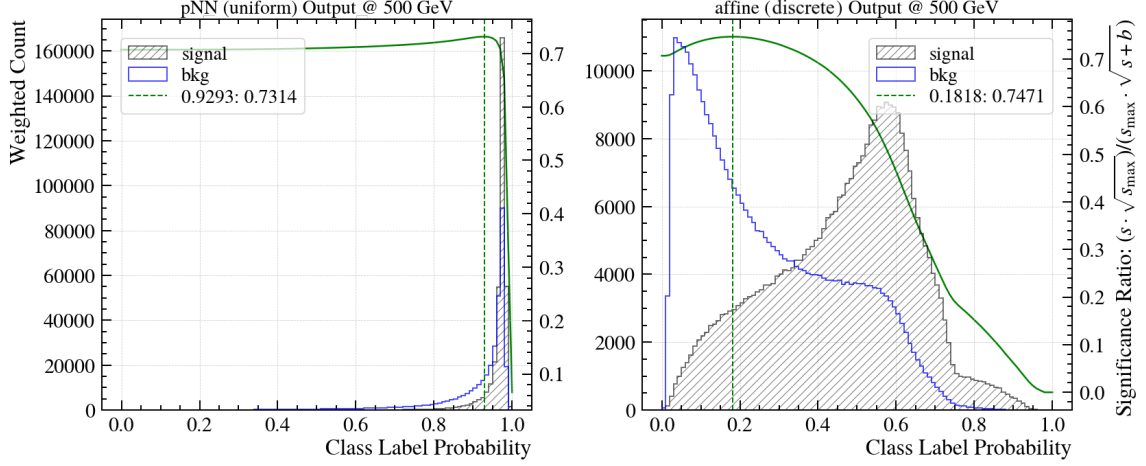


Figure 2.23: Comparison of the distribution of the predicted class label probability for two models: a pNN (**uniform**) against an affinePNN (**discrete**); the same models as in figure 2.22. Both models are evaluated at the mass point, $m_X = 500$ GeV on HEPMASS-IMB. The background contribution is weighted by $\frac{1}{|\mathcal{M}|} = \frac{1}{5}$. The histograms represent the predicted class distribution, the solid green curve is the significance ratio computed over various thresholds, and the vertical dashed line is the best cut (at 0.9293 and 0.1818), where σ_{ratio} is at its maximum: i.e., 0.7314 and 0.7471.

In figure 2.23 we can observe how the two models try to separate the signal from the background. The former struggles at identifying the signal, therefore the histograms are mostly overlapping, and, more importantly, the best cut is quite high (almost 0.93) approaching one. The latter model, instead, separates the two distributions almost well: we can see that the two peaks are at opposite sides and that the significance ratio decreases as we increase the threshold since more signal would be wasted, and that the classification threshold approaches the point in which the two distributions intersect; this model is closer to an ideal classifier.

It should be noted that the significance ratio is meant to be an additional metric to judge the classification performance of a discrete classifier, i.e., at a given threshold, and not a proper measure of statistical significance.

2.4.3 Model implementation and training

All the models considered for evaluation and comparison are built using TensorFlow 2 [73] along with the Keras [105] library for Python [106], having the same architecture consisting of four hidden layers each with 300, 150, 100 and 50 units, resulting in about 70k learnable parameters, with ReLU [57] activation and Dropout [66] regularization with a rate of 25%: the details of the two architectures (i.e., the pNN and affine pNN) employed for comparison can be found in the appendix A.3. The minimization of the binary cross-entropy (BCE) loss [37] was performed by the Adaptive moments (Adam) optimizer [107], with a learning rate of 5×10^{-4} , on a mini-batch size of 1024 data samples. The model's parameters are initialized following the `he_uniform` initialization [22], which are further regularized by an l_2 -penalty (i.e., equation 2.2.8) with strength of 10^{-5} and 10^{-6} for weights and biases, respectively.

Binary cross-entropy. The BCE loss is used as a cost function to learn neural network classifiers when the problem to solve involves discriminating two exclusive classes: a positive class, usually labeled with one, and a negative one associated with label zero. The convention about which class should be the positive or the negative is purely a design choice, in fact in classical ML these used to be associated with +1 and -1 labels, respectively. The BCE loss, for one data sample, is formulated as follows:

$$\mathcal{L}(y, \hat{y}) = -y \log \hat{y} + (1 - y) \log(1 - \hat{y}), \quad (2.4.3)$$

where y is the target class label and $\hat{y} = f_{\theta}(x)$ the predicted class. The loss is designed such that, according to the ground truth, either $\log \hat{y}$ or $\log(1 - \hat{y})$ is active: if the predicted class is completely wrong (e.g., $\hat{y} \approx 0$ but $y = 1$), the loss tends to plus infinity, which indicates the largest possible classification penalty; otherwise, if the label is completely right (i.e., $\hat{y} \approx 1$ and $y = 1$), the error would approach zero since there is no misclassification.

Adam optimizer. Compared to SGD, the Adam optimizer [107] enhances the update rule (i.e., equation 2.2.6) with estimated second-order information of the gradient:

$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \frac{\eta^{(t)} \hat{v}^{(t)}}{\sqrt{\hat{s}^{(t)} + \epsilon}}, \quad (2.4.4)$$

where $\hat{v}^{(t)}$ and $\hat{s}^{(t)}$ are the *de-biased* estimates of the first, $v^{(t)}$, and second moment, $s^{(t)}$, respectively, of the gradient $g^{(t)}$, which is computed by back-propagation as in equation 2.2.5. De-biasing is necessary since, initially, the estimates of the moments are initialized to zero: i.e., $v^{(t)} = s^{(t)} = 0$. Therefore, to avoid initial small values of the moments regardless of the gradient we need to perform the following bias correction:

$$\hat{v}^{(t)} = \frac{v^{(t)}}{1 - \beta_1^t} \quad \text{and} \quad \hat{s}^{(t)} = \frac{s^{(t)}}{1 - \beta_2^t}, \quad (2.4.5)$$

in which β_1 and β_2 are non-negative hyperparameters for the moving averages of the moments. Finally, the moments (mean and variance) are updated considering the history of the gradients:

$$\begin{aligned} \mathbf{v}^{(t)} &\leftarrow \beta_1 \mathbf{v}^{(t-1)} + (1 - \beta_1) \mathbf{g}^{(t)}, \\ \mathbf{s}^{(t)} &\leftarrow \beta_2 \mathbf{s}^{(t-1)} + (1 - \beta_2) (\mathbf{g}^{(t)})^2. \end{aligned} \tag{2.4.6}$$

The second-order information allows Adam to adapt (i.e., reduce or enlarge) the learning rate but on a per-parameter basis, effectively accelerating learning when the direction of the gradient is aligned with the direction of improvement⁹ and, at the same time, slowing down if the gradient’s direction is inconsistent also avoiding oscillations. These features make Adam a common default choice for optimizing neural networks, which also requires little tuning of the learning rate.

2.4.4 Classification results

To first assess the advantages brought by pNNs we should compare them to their “non-parameterized” (i.e., models that do not depend on a physics parameter) counterparts, namely *single* and *individual* neural networks: these two kind of models will represent our *baselines*. A single NN is just a neural network trained without the mass feature, m , as input but on all mass hypotheses simultaneously. Instead, the individual neural networks are a set of $|\mathcal{M}|$ single models, say f_{ϕ_i} , each of them trained to target a specific signal hypothesis: i.e., f_{ϕ_i} is trained on events associated to mass hypothesis m_i . Since each f_{ϕ_i} is optimized in isolation on one mass point against the whole background, the individual networks are expected to beat the single network, which should face a harder learning problem: there is no additional variable that provides information about the physics parameter, whereas for the individual networks knowledge about the parameter is implicit although there is no *parameter sharing*, i.e., there is no reuse of the learned weights to make learning more efficient since the networks are each learned separately.

As we can see in table 2.2, the single-NN is largely outperformed by all models whereas the individual networks is a strong baseline, as expected, capable of beating most of the pNNs trained with a uniform distribution of the mass parameter. For the parameterized models the best results are achieved by class-balanced training (see section 2.3.4) and discrete assignment of the parameter: class balancing helps to de-bias the model against disproportions of the class labels, whereas the discrete distribution of the mass feature allows for more robust learning. Furthermore, the affine architecture, which employs the affine conditioning at each hidden layer, can further boost the classification performance compared to a pNN that only concatenates the mass feature at the input level, hinting to us that stronger conditioning is effectively beneficial for the task to solve.

⁹Notice that here we consider the gradient estimated on a mini-batch of samples, which is noisy and potentially non-representative of the true gradient. Therefore, such a noisy gradient can point in a non-optimal direction.

Model		Mass (GeV)					Average (%)	
Kind	Mass Distribution	500	750	1000	1250	1500	AUC	σ_{ratio}
Single NN	None	72.62	86.56	92.68	93.83	94.06	87.95	
		73.52	79.51	85.59	86.85	86.86		82.47
Individual NNs	None	76.81	92.28	96.98	98.44	98.93	92.69	
		74.43	85.91	91.75	92.07	90.04		86.84
pNN	discrete	76.71	92.46	97.35	98.80	99.35	92.93	
		74.46	86.17	92.43	95.09	93.94		88.42
pNN (class)	discrete	76.93	92.56	97.41	98.85	99.39	93.03	
		74.67	86.34	92.55	95.32	96.64		89.11
pNN (full)	discrete	76.29	92.33	97.33	98.79	99.34	92.82	
		74.31	86.14	92.44	95.17	96.47		88.91
Affine	discrete	77.19	92.62	97.43	98.86	99.40	93.10	
		74.70	86.30	92.52	95.31	95.61		88.89
Affine (class)	discrete	77.19	92.64	97.46	98.88	99.42	93.12	
		74.76	86.38	92.62	95.37	96.77		89.18
Affine (full)	discrete	76.45	92.46	97.38	98.78	99.31	92.88	
		74.42	86.24	92.50	95.07	95.07		88.66
pNN	uniform	76.67	92.41	97.33	98.79	99.34	92.91	
		74.48	86.15	92.39	95.15	94.70		88.57
pNN (class)	uniform	71.00	91.56	97.14	98.75	99.32	91.55	
		72.71	85.64	92.22	95.09	96.45		88.42
pNN (full)	uniform	72.19	91.41	97.11	98.70	99.25	91.73	
		73.06	85.42	92.16	94.93	96.23		88.36
Affine	uniform	77.26	92.62	97.43	98.86	99.41	93.11	
		74.73	86.31	92.52	95.25	94.65		88.69
Affine (class)	uniform	77.29	92.68	97.47	98.89	99.43	93.15	
		74.82	86.42	92.64	95.40	96.80		89.22
Affine (full)	uniform	66.67	91.15	97.08	98.70	99.30	90.58	
		70.75	85.46	92.05	94.88	96.34		87.90

Table 2.2: Comparison of baselines, model architecture, parameter distribution for the background, and (balanced) training procedure on HEPMASS-IMB. Performances are evaluated in terms of AUROC and significance ratio. The top two best results are shown in boldface.

2.4.5 Interpolation and extrapolation capability

Ideally, to evaluate the ability of a parameterized model to interpolate and extrapolate intermediate and boundary parameters, respectively, we would need a specific test dataset, say $\mathcal{D}' = \{(x, y, \rho)_i\}_{i=1}^{N'}$ of size N' with $\rho^{(i)} \notin \mathcal{P}$, in which new events are associated to unseen parameter values. Since collecting or building such a dataset can be costly, especially in HEP, we can “edit” our training dataset such that to remove all the signal events associated with some parameter values: a new model will be trained considering the missing parameter values, meaning that some signal is missing, and the background events cannot be assigned the removed parameters. In this way, we can perform various interpolation and extrapolation tests by either removing single parameter values, a whole range of intermediate parameters, or even learning only from a single parameter forcing the model to extrapolate the entire parameter space. In the following paragraphs, these tests are demonstrated on the HEPMASS-IMB dataset considering the mass feature as the parameter to interpolate or extrapolate.

Single-mass interpolation and extrapolation. The simplest test we can perform is to remove the value of a single parameter, train a parameterized model, and evaluate it against the complete dataset. In this way, we can assess how difficult it is to interpolate or extrapolate that single parameter value, which, in this case, is about estimating the impact of a mass hypothesis on classification. By reusing the notation introduced in section 2.3.7, i.e., $\mathcal{D}_m = \{(x^{(i)}, y^{(i)}, m)\}_{i=1}^N$, we indicate a slice of the full dataset, $\mathcal{D}$, in which we select all the features and labels associated to a given parameter, i.e., $\rho = m$ with $m \in \mathcal{M}$: indeed, we need to choose the whole background. Doing so allows to either train on a single parameter value (like done in this section) or training on a subset of the parameters, e.g., by joining multiple dataset slices: $\mathcal{D}_{m_i, m_j} = \mathcal{D}_{m_i} \cup \mathcal{D}_{m_j}$. By inspecting the results in figure 2.24 we can say that each signal mass hypothesis has a different impact on generalization performance. In particular, we record a non-negligible loss in classification accuracy for low mass values as 500 and 750 GeV, especially for $m_X = 500$ GeV. This result confirms that low mass signal hypotheses are more overlapped with the background, as shown in figure 2.18a. Moreover, for high parameter values (e.g., 1000, 1250, and 1500 GeV) predicting the missing data points is almost trivial, entailing that the underlying learning task is easier than low mass parameters. Therefore, this kind of test can enable us to understand which parameter values (and potentially, their surroundings) are more difficult to predict, interpolate, or extrapolate.

Interpolating intermediate hypotheses. When targeting real applications, we wish to test our parameterized model on *intermediate* values of the physics parameter used to condition it. For simplicity, let’s consider the discrete case in which given two close training parameters, i.e., $m_i, m_{i+1} \in \mathcal{M}$ with $i \in \{1, |\mathcal{M}| - 1\}$, we wish our model to interpolate whatever novel parameter, m' , belongs to the (continuous) range $m' \in (m_i, m_{i+1})$. As seen for the previous single-mass test, removing a single parameter value can yield a too-simple interpolation task, which may not reflect real interpolation performance making it difficult

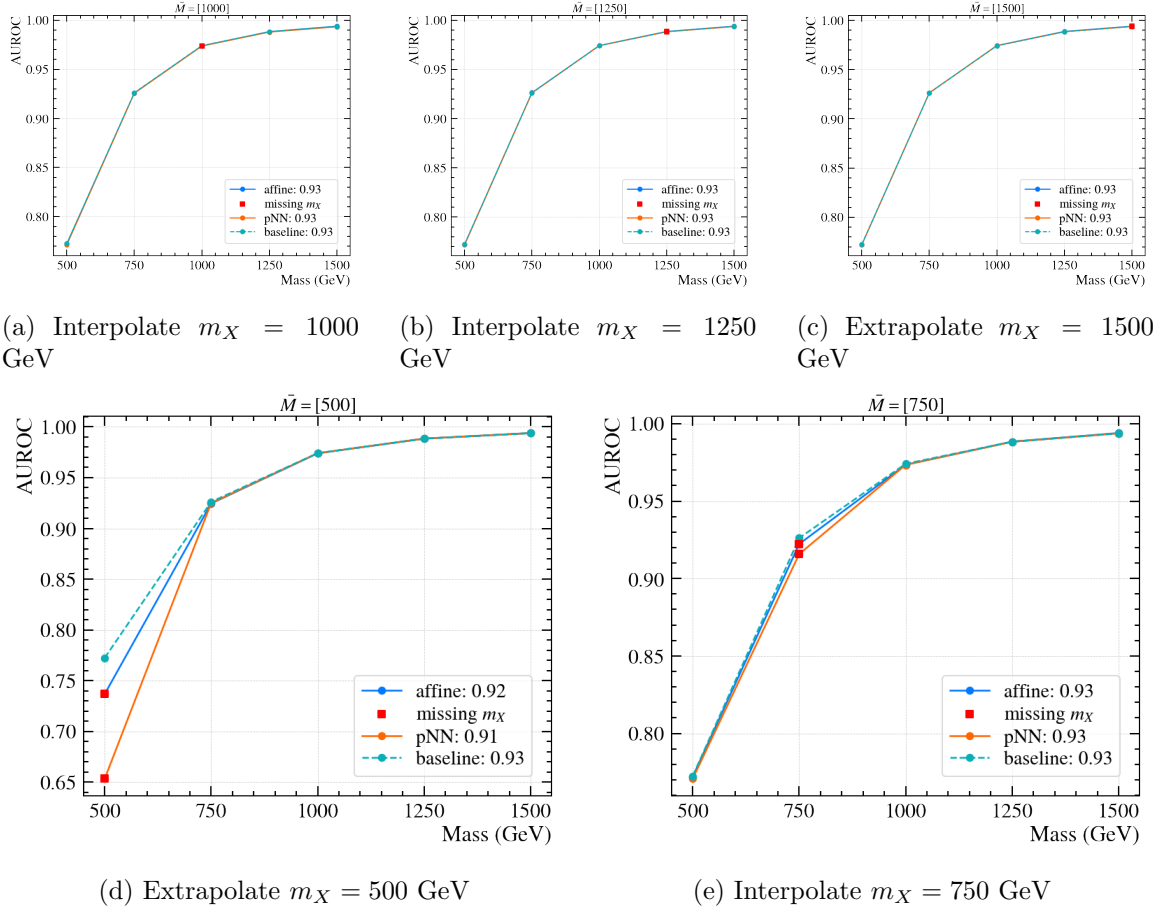


Figure 2.24: Single-parameter interpolation and extrapolation test on HEPMASS-IMB, comparing an affine pNN against a regular pNN. The baseline pNN is trained on all mass hypotheses. The red squares (i.e., $\blacksquare$) denote the mass hypotheses that were discarded during training, meaning that the respective signal events were unused. The notation $\bar{M}$ indicates the mass hypotheses (and corresponding events) not used during training.

to compare and rank models. Therefore, to better assess such capability, we removed all the intermediate mass hypotheses, i.e., 750, 1000, and 1250 GeV, to build a harder interpolation task. The results of this benchmark are shown in figure 2.25, in which we can see that the pNN model weakly generalizes on the 750 GeV hypothesis while the loss in performance for the affine architecture is way more limited but still large, hinting us that there is still room for improvement to design an even better architecture, training procedure, or parameter assignment strategy that could close this gap.

Extrapolating the entire mass range. To quantify the robustness of a parameterized architecture against novel parameter values, we can perform a sort of “extreme” test in

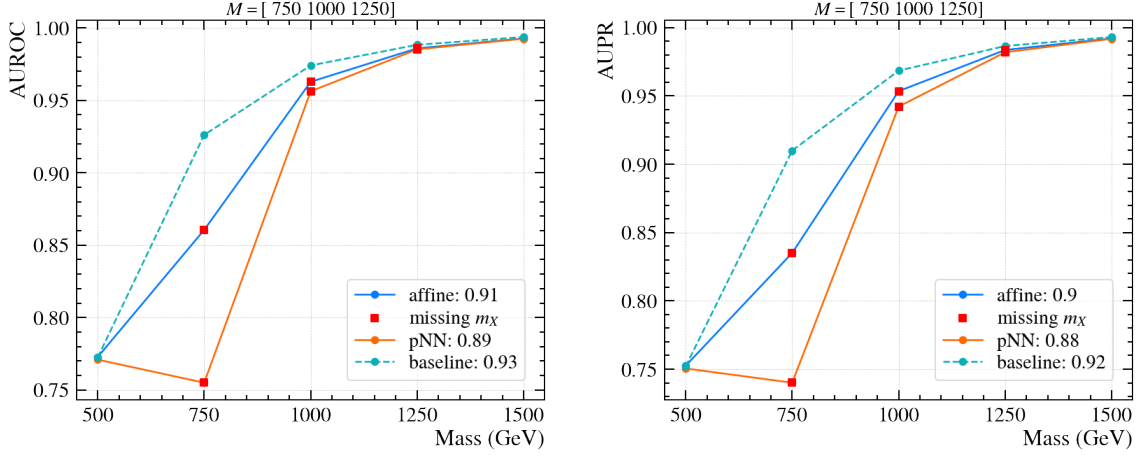


Figure 2.25: Interpolation on HEPMASS-IMB of the 750, 1000, and 1250 GeV mass hypotheses, showing the per-mass AUROC (left) and AUPR (right).

which we train our model on a single mass point and, then, evaluate how well the model can extrapolate both lower and higher mass values: in general, we can expect the performance of the model to degrade as the parameter values depart from the training ones. The result of such a test is summarized in figure 2.26, concluding that:

- The affine architecture is neatly more robust than a pNN that concatenates the parameter at the input layer, especially for low mass values (e.g., 500 and 750 GeV) implying that the affine conditioning better modulates the learned hidden features by the provided physics parameter, which results in wider generalization.
- Instead, for high mass values for which the HEPMASS-IMB dataset has more statistics the performance between the two architectures is comparable. This also entails that the affine pNN tends to be more data-efficient than a regular pNN.
- Lastly, this kind of test also allows us to understand the impact of each mass hypothesis. For our dataset, we can conclude that the 500 GeV hypothesis is the most difficult to learn as depicted in figure 2.18a. Therefore, we can expect lower interpolation and extrapolation performance if we remove the hardest hypotheses (or parameter values, in general) from the training and, instead, foresee good interpolation and extrapolation performance on the easiest hypotheses. This test can also be performed to get an initial idea about the complexity of our dataset, hence understanding which parameter values are better to interpolate (so, avoiding generating them) and which others should be simulated since expected to be difficult to learn.

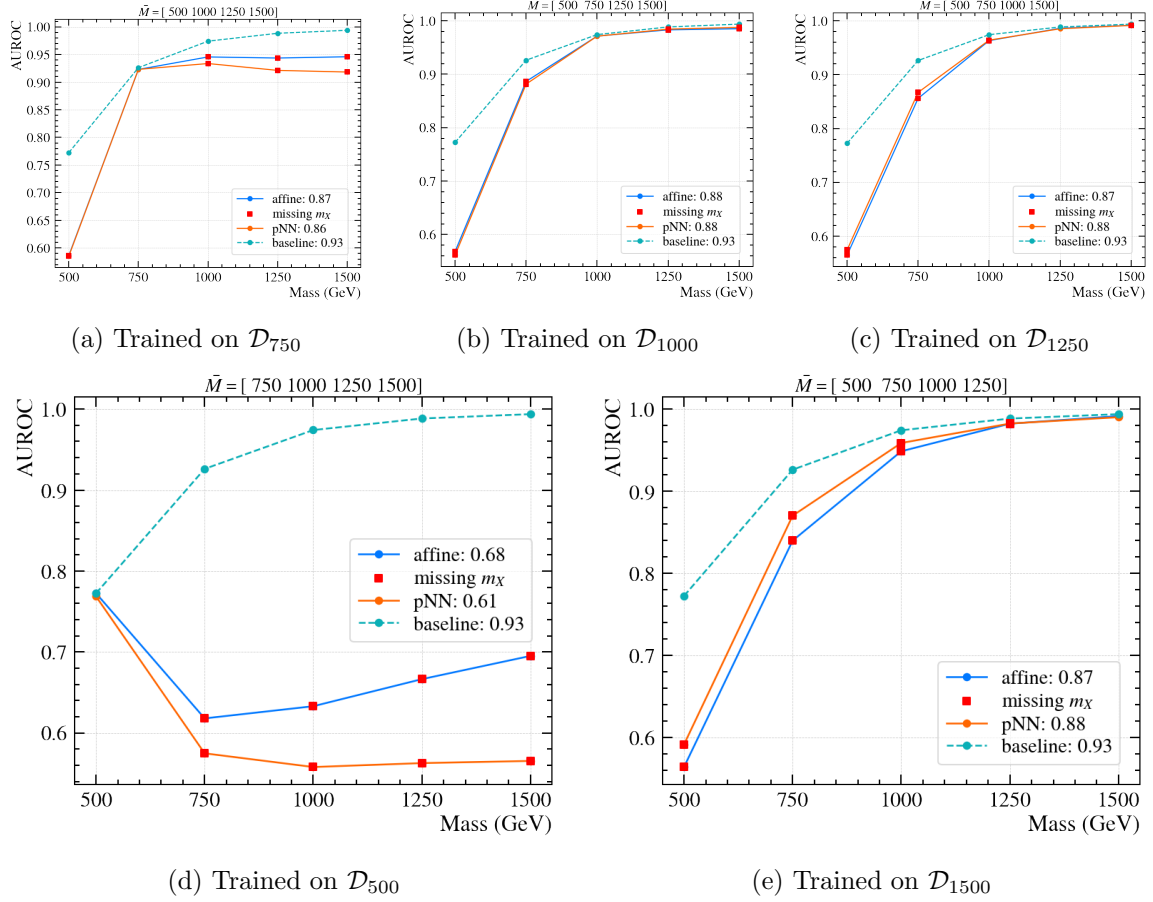


Figure 2.26: Extrapolation on HEPMASS-IMB, comparing an affine parameterized network against a regular pNN. The baseline pNN is trained on all mass hypotheses.

2.4.6 Evaluation of conditioning mechanisms

When designing pNNs the choice of the conditioning mechanism is an important one. In particular, we evaluated the *type* (concatenation, biasing, scaling, and affine conditioning) and the *position*, i.e., the place where the conditioning happens, such as: right after the input layer (**begin**), placed after every non-linearity (**all**), and just before the output layer (**end**). Table 2.3 shows the results for two tasks: classification, and interpolation [75]. The first thing to notice is that the conditioning method mostly affects the interpolation rather than the classification performance of the model: the difference between the best and worst model can be around 20% for interpolation, while the loss in performance is way less large for classification that is around 5% at most. This is an expected important fact since the conditioning mechanism enables the model to learn the dependency between the parameter and the features. Therefore, better conditioning implies better interpolation.

Conditioning		Classification (%)			Interpolation (%)		
Type	Position	AUROC	AUPR	σ_{ratio}	AUROC	AUPR	σ_{ratio}
Concat	begin	93.09	92.20	88.84	89.20	88.13	85.67
Concat	all	90.19	88.67	87.10	84.37	83.81	83.58
Concat	end	88.45	87.58	83.40	70.93	72.57	71.90
Biasing	begin	92.98	92.08	88.80	87.25	86.07	84.58
Biasing	all	93.12	92.23	88.95	91.44	90.36	86.99
Biasing	end	88.57	87.68	83.69	69.89	72.36	71.86
Scaling	begin	93.01	92.10	88.74	85.42	85.53	82.35
Scaling	all	92.93	91.99	88.73	87.95	86.80	84.81
Scaling	end	92.03	90.76	87.64	87.12	86.84	83.84
Affine	begin	92.41	91.54	88.14	90.34	88.98	86.21
Affine	all	93.05	92.14	88.92	91.50	90.33	87.61
Affine	end	91.98	90.79	87.45	86.78	86.40	83.97

Table 2.3: Impact of conditioning mechanisms on HEPMASS-IMB. For interpolation, the missing signal mass hypotheses are $\bar{\mathcal{M}} = \{750, 1000, 1250\}$. The best results are shown in boldface.

As depicted by table 2.3, conditioning at the **end** usually leads to the worst performance. In contrast, biasing and affine conditioning perform the best in both scenarios: the latter being more robust regardless of the position at which it occurs. Finally, concatenation (at **begin**) attains competitive results.

2.5 Application of parameterized models in a real-world HEP analysis

In this section, the advancements proposed and discussed in the previous sections are demonstrated by the application of a parameterized network: the affine pNN, which is employed to solve the signal-background classification part of a study¹⁰ of the decay of BSM neutral Higgs bosons to a $\mu^+\mu^-$ pair [108, 109].

¹⁰This analysis is still part of an ongoing work that uses data, simulations, and software from the CMS collaboration.

2.5.1 Introduction

In this analysis, the BSM Higgs can be described by two parameters: m_A , i.e., the mass of such particle, and $\tan\beta$, a parameter given by the theoretical model under consideration. The search for the BSM Higgs bosons is conducted considering proton-proton collisions at a center-of-mass energy of 13 TeV, with the 2016 Run 2 data.

The searched signal. The dominant production mechanism for the neutral BSM Higgs bosons is the *gluon fusion* (or, gg), which is relevant for small and medium values of $\tan\beta$. Instead, at large values, i.e., $\tan\beta > 30$, the b -associated production (bb) becomes increasingly dominant. The searched signal is hypothesized to exist within a mass range between 130 and 1500 GeV. Moreover, the presence of a signal, whose production is described by the diagrams in figure 2.27, would be characterized by an excess of events over the background in the dimuon invariant mass (i.e., the mass of the $\mu^+\mu^-$ final state), which would then correspond to the mass of the boson.

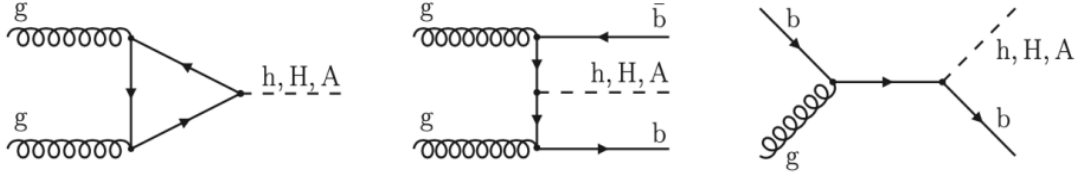


Figure 2.27: Feynman diagrams depicting the searched BSM production of the neutral Higgs bosons, h and H : gluon fusion, on the left, and b -associated production, on the center and right. Figure from [108].

Background sources. The main source of background for the Higgs bosons is the Drell-Yan (DY) process. Another relevant background source comes from opposite-sign dimuon pairs produced in the semi-leptonic decay of the top quark in $t\bar{t}$ and single top events. Lastly, less relevant contributions come from the diboson production processes, namely $W^\pm W^\mp$, $W^\pm Z$, and ZZ . In particular, these events contribute very little to the dimuon invariant mass for masses larger than 130 GeV, where the Higgs signal is searched.

Signal samples. In general, the analysis adopts a data-driven approach for the background estimation relying on the Monte Carlo simulation only for the signal modeling, in which the MC samples are generated for collisions at $\sqrt{s} = 13$ TeV. For the signal, the samples have been produced according to the values of $m_A = [130, 150, 170, 200, 250, 300, 350, 400, 450, 500, 600, 700, 800, 1000, 1200, 1500]$ and $\tan\beta = [2, 5, 10, 15, 20, 25, 30, 40, 50, 60]$ parameters. Consequently, for each $(m_A, \tan\beta)$ pair, the BSM Higgs bosons produced via gluon fusion (i.e., gg) and in association with b -quarks (i.e., bb) have been generated.

2.5.2 Analysis strategy

The events selected for the BSM Higgs to $\mu^+\mu^-$ analysis are pre-selected by the CMS trigger system, requiring a muon candidate within the pseudo-rapidity range of $|\eta| < 2.4$ and $p_T > 24$ GeV. Then, since events containing b -jet candidates provide the highest sensitivity for the b -associated production channel, and events that do not comprise b -tagged jets provide the best sensitivity for the gluon fusion production channel, these have been split into *two, exclusive* categories: the b -tag category (**cat1**), which includes events with at least one b -tagged¹¹ jet, and the no b -tag category (**cat2**), containing events without b -tagged jets. This theory-driven pre-categorization of the events yields a better sensitivity to the signal than discriminating on both categories simultaneously.

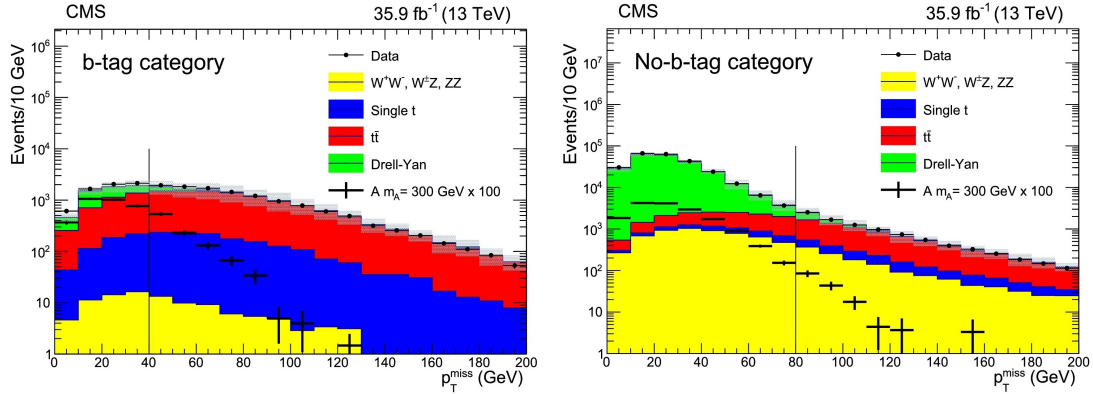


Figure 2.28: On the left, the transverse missing energy in the event for b -tagged events (**cat1**), while on the right the same quantity for non b -tagged events (**cat2**). The quantities refer to events with dimuon invariant mass larger than 130 GeV. The plots are stacked histograms with each color representing a background process. Figure from [109].

Moreover, due to the above categorization, the dominant background process changes according to the category as can be appreciated in figure 2.28. In particular, the background from $t\bar{t}$ events is now much more relevant for the b -tag category, whereas for the no b -tag category the dominant background is the DY production.

Physics parameter. As anticipated at the beginning of the section, the analysis is conducted by training a parameterized classifier, whose physics parameter is the mass hypothesis of the BSM Higgs boson, i.e., m_A . Since the value of the parameter is only well defined for the signal samples, we employ an assignment strategy (see section 2.3.4) for the background based on *mass intervals*. In particular, depending on the characteristic range in the dimuon invariant mass, $m_{\mu^+\mu^-}$, the respective m_A is assigned to the background events included in that interval. The mass intervals with the respective assigned parameter values

¹¹The tagging (or classification) of a jet from the b quark is performed by ML and, more recently, DL algorithms, like DeepJet [110]. This tagging phase provides additional variables for subsequent methods.

are listed below:

- $m_{\mu^+\mu^-} = (115, 180)$ GeV $\rightarrow m_A = 130$ GeV;
- $m_{\mu^+\mu^-} = (115, 200)$ GeV $\rightarrow m_A = 150$ GeV;
- $m_{\mu^+\mu^-} = (120, 220)$ GeV $\rightarrow m_A = 170$ GeV;
- $m_{\mu^+\mu^-} = (150, 250)$ GeV $\rightarrow m_A = 200$ GeV;
- $m_{\mu^+\mu^-} = (200, 300)$ GeV $\rightarrow m_A = 250$ GeV;
- $m_{\mu^+\mu^-} = (225, 375)$ GeV $\rightarrow m_A = 300$ GeV;
- $m_{\mu^+\mu^-} = (275, 425)$ GeV $\rightarrow m_A = 350$ GeV;
- $m_{\mu^+\mu^-} = (300, 500)$ GeV $\rightarrow m_A = 400$ GeV;
- $m_{\mu^+\mu^-} = (350, 650)$ GeV $\rightarrow m_A = 500$ GeV;
- $m_{\mu^+\mu^-} = (400, 800)$ GeV $\rightarrow m_A = 600$ GeV;
- $m_{\mu^+\mu^-} = (500, 900)$ GeV $\rightarrow m_A = 700$ GeV;
- $m_{\mu^+\mu^-} = (600, 1000)$ GeV $\rightarrow m_A = 800$ GeV;
- $m_{\mu^+\mu^-} = (700, 1800)$ GeV $\rightarrow m_A = 1000$ GeV;
- $m_{\mu^+\mu^-} = (700, 1800)$ GeV $\rightarrow m_A = 1200$ GeV;
- $m_{\mu^+\mu^-} = (700, 1800)$ GeV $\rightarrow m_A = 1500$ GeV.

There are a few things to notice. 1) the value of the intervals is determined by inspecting the $m_{\mu^+\mu^-}$ distribution of background events: as can be seen in figure 2.29, the local peaks of the signal subsume a distribution that gets wider and wider; 2) the size of the intervals is not constant but, instead, increases as m_A increases: this is to account for the limited statistics of the background samples available at high invariant mass; 3) lastly, for the (700, 1800) intervals multiple values for the m_A parameter are possible: therefore, we assign either 1000, 1200 or 1500 GeV randomly, with uniform probability.

Model training. Two different models, one for each category have been trained: their architecture and hyperparameters are the same, but the input features are not. This is because event categorization allows us to select a relevant subset of features for each category, making the classification problem easier. In fact, by exploiting our domain knowledge, we can make learning more efficient by discarding irrelevant features, enabling us to study the resulting classifiers better while improving signal sensitivity. Moreover, half the data are

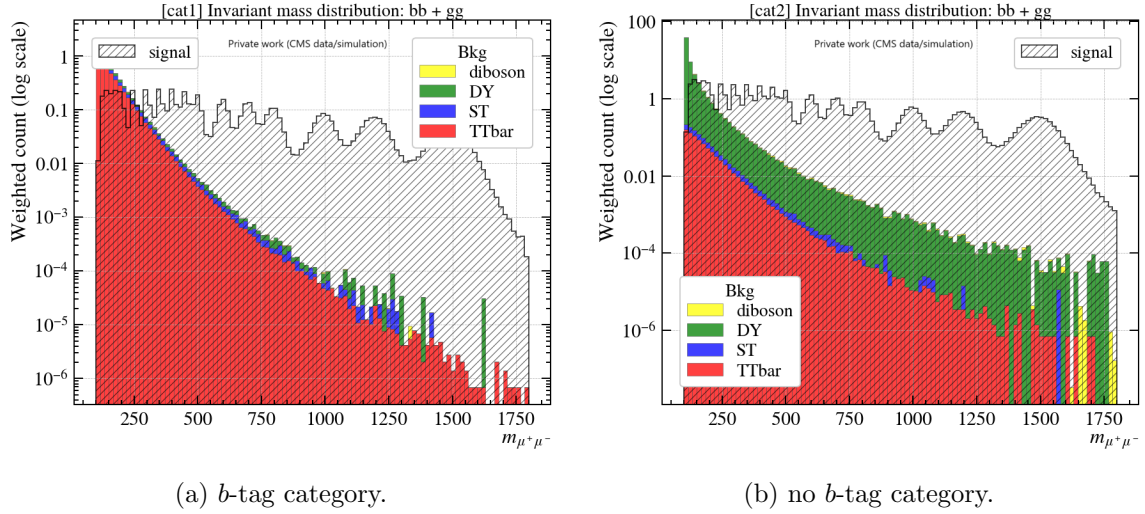


Figure 2.29: Comparison of the invariant mass distribution between categories. Stacked colored histograms represent the weighted contribution of each background process. As we can see, after about 350 GeV the imbalance between the two classes gets larger and larger.

used for training¹² and the other half for the final testing: 25% of the training is reserved for validation, which occurs by tracking the AUROC metric. Before training, the data is normalized to have approximately unitary variance and zero mean by subtracting the mean and dividing by the standard deviation of the features. Moreover, the input features were limited in value to a more suitable range due to severe outliers, which are expected to occur especially for high-mass events where the data is more scarce.

The chosen model for the analysis is the affine pNN (see appendix A.3), featuring four fully-connected layers each with 300, 150, 100 and 50 units: the activation is ReLU; Dropout [66] is also applied after each layer. To account for the discrepancy in statistics between the signal and background classes, also considering the multiple background processes, the full balanced training procedure (section 2.3.4) has been used to train the parameterized network: at the mini-batch level, both classes are balanced, all the m_A are represented for the signal, and all background processes (which are also imbalanced) have an equal number of events. Fully balanced mini-batches contain a mix of events whose chance is biased compared to the original occurrence frequency of events in the unbalanced case: balancing causes the model to experience less represented events more often, encouraging the network to fit them equally well. In this case, the balancing technique resulted in overall better performance.

¹²For the two categories the training split is mixed as follows: **cat1** has 8.8M background samples, and about 1.4M b-associated and 150k gluon-fusion signal samples; **cat2**, instead, presents 1.7M background samples and 600k b-associated and 540k gluon-fusion signal samples. The same applies to the test set.

2.5.3 Improvement over the cut-based approach

The previously described data-driven classifier is compared to the *cut-based discriminator* [109], i.e., a set of classification rules designed by physics experts through careful inspection of the available feature distributions. Since the data is divided into two categories, we have two cut-based rules. For the **cat1**, the criteria to select the signal is:

$$(N_{b\text{-jet}} = 1) \wedge (N_{l\text{-jet}} < 2) \wedge (E_{\text{miss}}^T < 40), \quad (2.5.1)$$

where $N_{b\text{-jet}}$ and $N_{l\text{-jet}}$ is, respectively, the number of b quark jets and light quark jets, and E_{miss}^T the missing energy in the transverse plane. Instead, for the **cat2** the selection is:

$$E_{\text{miss}}^T < 80. \quad (2.5.2)$$

The cut-based discriminator is fixed, meaning that, unlike a neural network, the classification threshold cannot be tuned to trade off the *type I* error (i.e., false positives) for the *type II* one (i.e., false negatives) or vice-versa, for example. To accomplish that, domain experts should adapt the quantities associated with the respective features in a cut.

Figure 2.30 shows the classification performance of the affine pNN compared to what a cut-based approach can achieve. As we can see, the pNN always outperforms by a large margin the traditional, not data-driven, cut-based approach for the evaluated signal mass hypotheses. Moreover, even if we change the neural network's working point to match the background rejection rate of the cut-based approach, we can notice (by moving along the y -axis) that larger signal retention is still achieved. This means that after applying the pNN as a selection cut, the resulting sample is more pure resulting in improved statistical significance or upper limits. Furthermore, for the no b -tag category both signals are way harder to separate from the background because, in this category, the DY process is dominant being distributed quite similarly to the signal. Therefore, to preserve a relevant amount of signal most of the background also passes the selection. Luckily, the pNN can still improve over the cut-based approach in such a difficult scenario. Lastly, compared to a more traditional neural network approach (e.g., [52]), in which for each value of m_A a distinct classifier would be trained, our single pNN can tackle all the numerous signal-generating hypotheses at once simplifying the whole workflow.

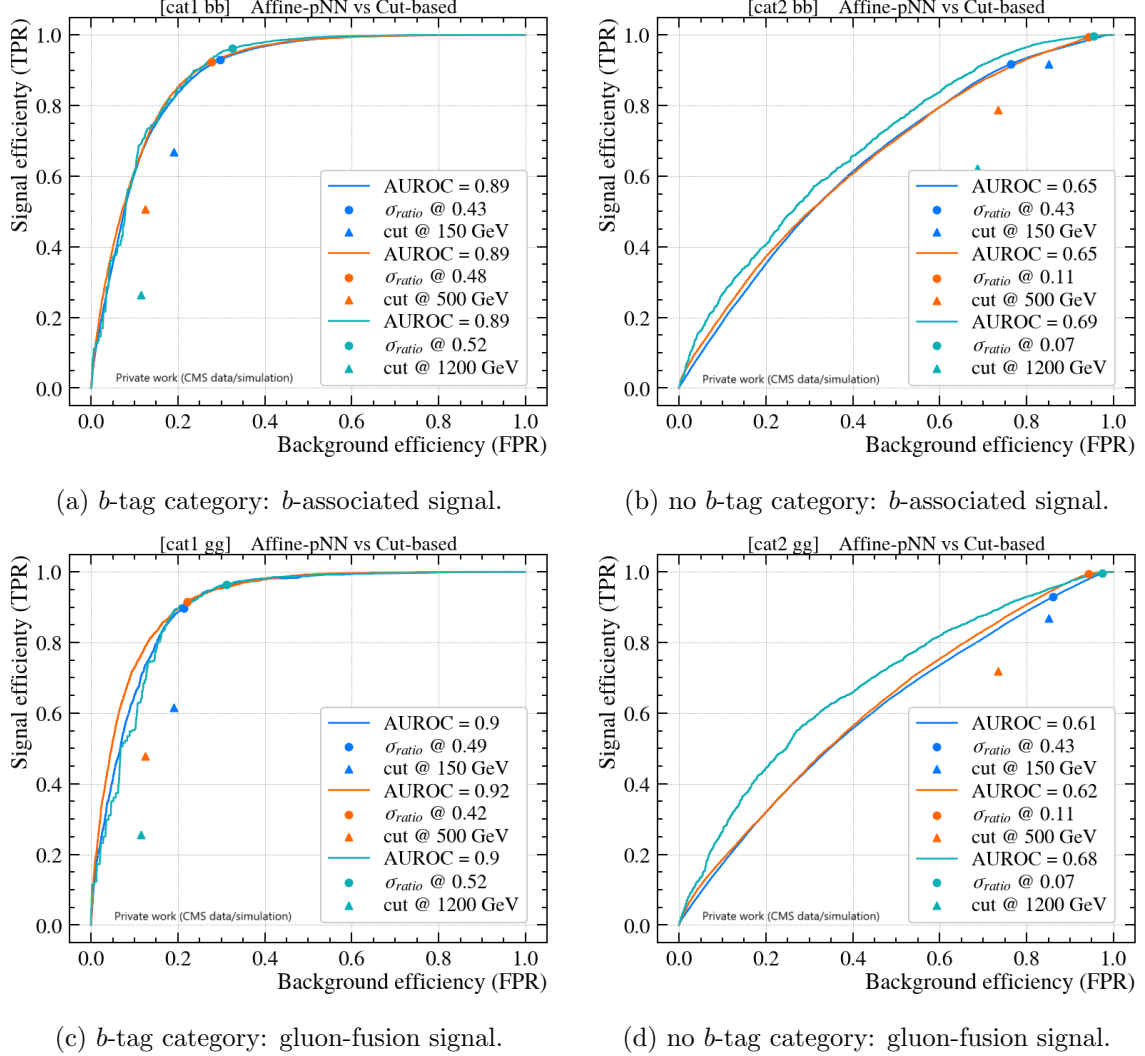


Figure 2.30: Performance of an affine pNN compared to the cut-based approach [109]: each plot confronts the results for three values of m_A : 150, 500 and 1200 GeV. The ROC curves can be computed only for the pNN. Conversely, the other approach yields a single point on the ROC space denoted by colored triangles ($\blacktriangle$, $\blacktriangle$, and $\blacktriangle$). Colored circles ($\bullet$, $\bullet$, and $\bullet$) depict the working point of the pNN, given by the threshold that maximizes the significance ratio (σ_{ratio} ; defined in section 2.4.2).

2.5.4 Conclusions

Designing, training, and evaluating a classifier for signal-background discrimination is only the first part of the analysis. The rest involves the data-driven estimation of the background contribution using the Run 2 data, which is then used to define the null, H_0 , and the alternative, H_1 , hypotheses. The hypothesis H_0 assumes the existence of the signal along the background, which is tested against the background-only hypothesis H_1 we wish to reject. Doing so allows us to determine a p -value converted into an equivalent significance. The discovery of a new signal process requires the rejection of the background-only hypothesis with a significance of at least 5σ , indicated as standard deviations of the Gaussian distribution. By employing a parameterized neural network approach we enriched the signal sensitivity of the discriminator compared to the classical, hand-designed cut-based approach. More details about the analysis setup, calculations, and results are available in [108].

2.6 Summary

In this first part of the thesis, we described what feed-forward neural networks are, and introduced the notion of *conditioning on a physics parameter* through parameterized neural networks. The pNN concatenates the features along the parameter to solve the signal-background classification problem, a central part of HEP analyses. We then presented multiple ways in which domain or expert knowledge can be embedded into a deep learning workflow to improve classification performance, data efficiency, and interpretability, or to make the model physics-aware. Next, we discussed various parameterized network properties, such as interpolation and extrapolation. Subsequently, we presented and discussed the results of our experiments, showing that a better architecture with informed training can lead to superior classification and interpolation performances. Lastly, the validity of the proposed method is demonstrated in a real-world HEP analysis, largely surpassing the signal separation performance attained by a domain-specific cut-based approach.

2.7 Conclusions and future perspectives

Parameter-conditional neural networks can be seen as a generalization of parameterized neural networks, in which we can handle multiple discrete or continuous physics parameters, and include domain knowledge at various levels: architecture, loss, and training. Potentially, this kind of model can be employed, following similar ideas, in many scientific fields beyond HEP (e.g., finance [99]) bringing improvements for real-world problems (e.g., modeling auditory models [111]). Notoriously, the challenges of the real-world, regarding deep learning, are various starting from: *data-efficiency*, since new (labeled) data can be either difficult or costly to collect; *fast inference*, as the model may be required to work in real-time complaining to (strict) latency requirements; *interpretability*, neural networks are very difficult to understand, hence, the introduction of domain-driven models may ease interpretation; *safety*, a model that controls some equipment (like a detector) must operate

it without introducing any damage or hazard; and *robustness*, as, ideally, the model should handle distribution shifts (e.g., caused by either novel, anomalous, rare, or erroneous data) and resist to adversarial attacks [39].

The introduction of expert knowledge can make training more efficient, thus requiring fewer data samples, it can make the model more interpretable if this, for example, realizes a theoretical formula, and even improves its generalization to handle rare and unexpected data. These are long-term goals that all we care to be addressed, and short-term research, in this regard, may focus on providing better *benchmark datasets* (e.g., more complex, with multiple parameters and background processes) and *scenarios* (e.g., interpolation and extrapolation tasks), improved *conditioning mechanisms*, *efficient sampling* in case of multiple parameters, HEP *equations and constraints* that can be embedded and studied, and also *novel problems* that go beyond signal-background classification as these ideas are general enough to be applied to other kinds of models too.

Chapter 3

Anomaly Detection for Model Independent Search

In the previous chapter of this thesis, we introduced the class of parameter-conditional neural networks, which are employed in HEP to solve the signal-background classification problem in the context of model-dependent analyses. In this chapter, we will introduce self-supervised methods expected to decrease the dependency on the assumed physics model, providing complementary benefits compared to previous approaches.

3.1 The promises of model-independence

HEP analyses can be either model-dependent or model-independent. *Model-dependent* analyses target a specific new physics (i.e., a signal) being conducted around the assumed physics model. In the second chapter of this thesis, we have already seen an example of model-dependence, in which the pNN was trained to separate the signal from the background: the knowledge about the signal events provides the dependence on the physics model that should describe the targeted new physics, to which the classifier should be robust to. In this regard, the neural network adapts its learnable parameters to be sensitive to the signal: this allows it to achieve the best classification performance on that specific signal but at the cost of either unwanted correlations, e.g., with the invariant mass, or even weaknesses caused to the erroneous modeling (or any modeling at all) of the *systematic uncertainties*, i.e., uncertainties related to the measurement, experiment, and assumed model. Since in these kinds of analyses, the DL models are trained on *simulated* data, these do not (fully) account for *detector effects*, which are yielded by the interaction of particles with the detector material that can alter the measurement of the experimental data. Therefore, a classifier when tested on real data (which presents these effects) can produce misleading and erroneous results due to not being designed to be robust against these forms of uncertainties. Hence, the appealing advantage of being *model-independent* or *model-agnostic* is that the learned model is intrinsically more robust against those signal-specific uncertainties since in a model-independent analysis we target specific backgrounds rather than a specific new

physics: this is the key difference in which the model now adapts to the background processes (rather than the signals) being potentially and generally applicable to a wide range of (related) signals. Being model-agnostic means that we assume no or minimal knowledge about the physics model that describes the new physics phenomena (either new particles or unexpected physical processes), which allows us to find deviations with respect to the background data therefore enabling us to search for multiple new physics scenarios at the same time [112].

For the above reasons, model-independent searches are becoming a valid alternative to model-dependent ones, aiming to discover new physics Beyond the Standard Model (BSM) governed by a vast parameter space. Hence, an enormous number of model-dependent analyses would be required to exhaustively test such a vast parameter space since each search targets a specific signal. In practice, the dependence on the physics model can be reduced by assuming to be in a weakly-supervised [113] or even unsupervised [114] setting in which only what it is related to the background events is known (allowing also the background sample to be contaminated by some degree with the signal.)

3.2 Introduction to anomaly detection

Anomaly detection (AD) is the task of determining which data samples violate some notion of normal or expected behavior: once identified, these samples will be flagged as *outliers* or *anomalies* representing either corrupted (e.g., malfunctioning sensors, erroneous or noisy measurement or labeling), rare (i.e., with low probability to occur), or interesting (i.e., unexpected) events [115]. In the context of HEP, anomalous samples can potentially represent new physics signals, in terms of new particles or physics processes. Beyond particle physics, AD is employed in many applications such as *intrusion detection* (i.e., identifying malicious activity in a computer system), *fraud detection* (e.g., in telecommunications, insurance, and banking), *spam and ads detection*, *log anomaly detection* (to detect the failure of a system), *malware detection*, *video surveillance*, and much more. Moreover, anomaly detection can be performed on a variety of data modalities, such as: *text* (e.g., log files), *images* (i.e., medical and industrial), *videos* (e.g., frames from security cameras), *graphs* (e.g., social networks), and even *time series* (e.g., stream of sensor data) [115].

3.2.1 Deep anomaly detection

When combined with deep learning [1], anomaly detection takes the name of *deep anomaly detection* (DAD) in which neural networks are responsible for learning a representation of the data (either normal or anomalous, or both), or even directly devise an anomaly score [115, 116].

Following the notation of [116], given a set of inputs (e.g., arbitrarily shaped tensors), $\mathcal{X} = \{\mathbf{x}^{(i)}\}_{i=1}^N$ where $\mathbf{x}^{(i)} \in \mathbb{R}^D$, we aim at either learning a feature representation mapping, $f_\phi : \mathcal{X} \rightarrow \mathcal{Z}$ where $\mathcal{Z} \in \mathbb{R}^K$ is a lower-dimensional representation space such that $K \ll D$, or an anomaly score (i.e., the criteria that allow to determine if a sample is anomalous or

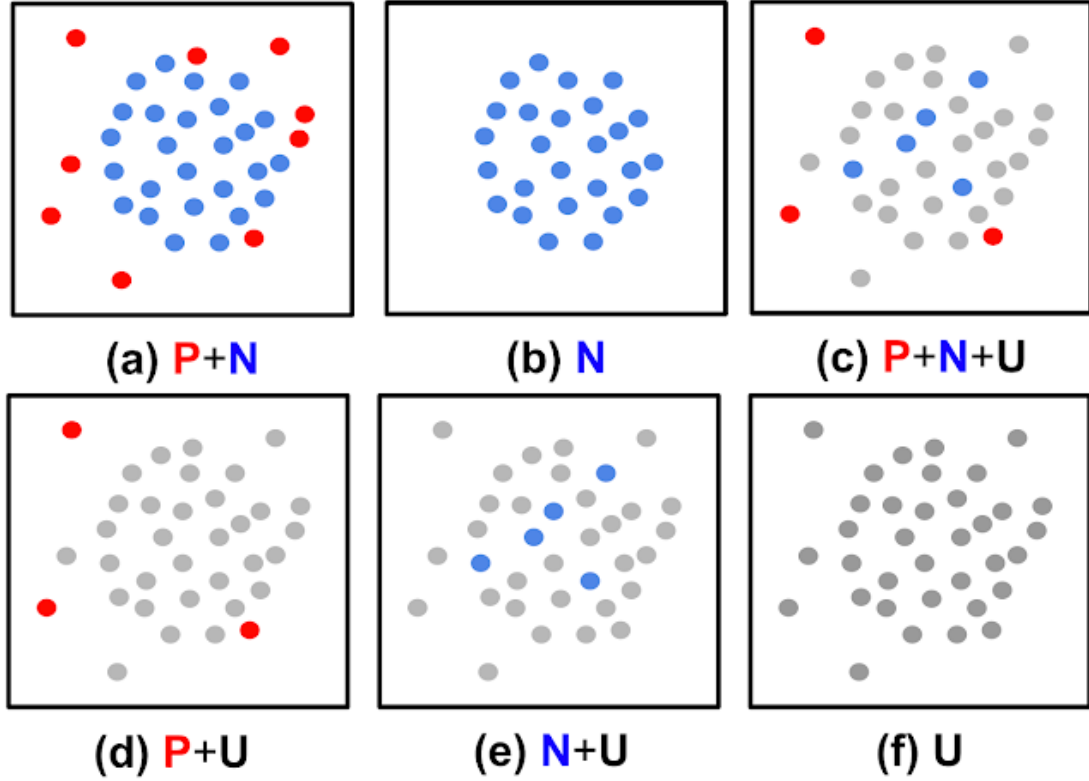


Figure 3.1: How the class label relates to the kind of anomaly detection: **positive (P)** samples are anomalous, **negative (N)** ones are normal, lastly the class of **unlabeled (U)** samples is unknown. (a) Supervised anomaly detection, (b) normal-only anomaly detection, (c, d, e) semi-supervised anomaly detection, (f) unsupervised anomaly detection. Figure adapted from <https://research.google/blog/unsupervised-and-semi-supervised-anomaly-detection-with-data-centric-ml/>.

not), $f_\tau : \mathcal{X} \rightarrow \mathbb{R}$, through neural networks, f_ϕ or f_τ . In the former case, learning a feature representation entails an additional step to compute the anomaly score for each data point. The latter case, instead, allows us to directly infer the anomaly score from the raw inputs, $\mathbf{x}$. Often, the anomaly score provides a distribution distinguishing normal from anomalous events. Therefore, we seek to determine an optimal threshold such that all instances with scores exceeding this threshold are classified as anomalies. These two AD approaches can be implemented using the supervised, semi-supervised, or unsupervised paradigms [115].

Supervised anomaly detection. Supervised AD can be regarded as (multi-class) classification since full knowledge about the class labels is available, therefore we know which samples are normal and which are anomalous: as can be seen in figure 3.1a, in which all data points have a label assigned and, so, no examples are unlabeled or unknown. Hence,

anomaly detection models belonging to this category can directly learn an anomaly score (e.g., an estimate of the class label probability) to separate the samples or act as feature extractor first by using pre-trained models (e.g., VGG [117] and ResNet [23]). Thanks to class label knowledge, fully supervised AD can achieve the highest performance compared to the other settings but at the cost of gathering plenty of data (since the anomalous samples may rarely occur) with accurate labels. Another advantage is that the testing phase is simple and fast, not requiring additional steps to determine the class label. Regarding HEP, supervised (or model-dependent) searches can be constructed as hypothesis test, in which the optimal test statistics is the *likelihood ratio* [118]: $p_{\text{SM}+\text{BSM}}(x)/p_{\text{SM}}(x)$ for an event x , where **SM** represents the background hypothesis and **BSM** the presence of the signal.

Semi-supervised anomaly detection. Semi-supervised or one-class techniques assume that all training instances belong to only one class. For example, in the context of new physics searches, we may consider our training set to contain only background events: this can also be regarded as *normal-only* anomaly detection, as shown in figure 3.1b. A more realistic situation would be to assume the normal data to be contaminated with some little (but unknown) fraction of the signal. Instead, in a weakly supervised setting we may want to exploit the few labeled data we have (either signal, background, or both) to learn from a large (but unknown) mixture of unlabeled data points: figures 3.1c, 3.1d, and 3.1e. Notice that in the context of HEP, weak supervision entails learning from *noisy* labels [118] (e.g., mixtures of signal- or background-like data). In contrast, the labels in a semi-supervised setting are noiseless but only available for a subset of the training data. Methods that aim at learning the notion of normality usually optimize a generic feature learning objective that is not mainly designed for anomaly detection. However, the learned representation is still useful since the objective enforces capturing major data regularities [116]. Common approaches are based on auto-encoders [119], generative adversarial networks [120], and self-supervised learning (SSL) [68]. In principle, semi-supervised methods can be trained directly on real detector data, which is expected to be mostly background with little signal contamination, thus avoiding the common step of first training on simulations instead. This thesis chapter focuses on normal-only anomaly detection applications powered by auto-encoders.

Unsupervised anomaly detection. In unsupervised AD knowledge of the class label is not assumed, as depicted in figure 3.1f, therefore, the algorithm has to find patterns in the data such that some of these can be associated with normal samples and some others to outliers: notice that the data should still present both anomalous and regular samples, but their proportion is unknown. Such a class of approaches could, in principle, detect new types of anomalies as new data are collected. In this regard, we distinguish between *distance-based methods*, such as the k -nearest neighbors [121] which assumes the anomalies to be far from their close neighbors while normal data group in dense neighborhoods; and *clustering-based methods*, which suppose that normal instances have greater coherence than anomalies, expected to deviate from the discovered clusters significantly. In both distance and clustering approaches it is common to have *pseudo* class labels (obtained by some

intermediate algorithm) that guide the representation learning or the clustering process. Distance-based methods tend to be more theoretically grounded but they suffer from high computational cost (since to score a sample the distance between other samples is required) and potential weaknesses inherent to the employed distance measure, conversely deep clustering methods can learn representations tailored for AD but the final performance can be sensitive to the clustering result and also be biased when there are anomalies in the training data [116].

In general, semi-supervised and unsupervised searches for new physics do not rely on a specific signal hypothesis, since it is not possible to construct an optimal test statistic for all potential signals because the background is targeted instead. Therefore, unsupervised searches target events with low p_{SM} probability as new physics signatures. Conversely, weakly and semi-supervised methods can use label information to construct an alternative likelihood ratio, e.g., $p_{\text{data}}(x)/p_{\text{SM}}(x)$, which is optimal for a data-versus-background hypothesis test. However, this test requires the estimation of the data probability, p_{data} , hence it is called the *idealized anomaly detector* [118].

3.2.2 Real-world challenges

Compared to classification or even clustering, which is related to anomaly detection since it aims at capturing regularities in the data, (deep) anomaly detection must handle minority, uncertain, unexpected, novel, and rare events, which can exacerbate the already known weaknesses of deep models. Therefore, to robustly employ anomaly detection methods in real-world applications, the following challenges have to be addressed or mitigated [116]:

- **Heterogeneity.** One class of anomalies may demonstrate a completely different anomalous behavior than another class. This makes it difficult to have a global knowledge of what is anomalous since this notion is often domain-specific. Moreover, anomalies can be divided into three categories: *point* anomalies are individual anomalous instances compared to the majority (e.g., detector malfunctions); *conditional* or *contextual* anomalies are only abnormal in a specific context, otherwise these are regarded as normal; lastly, *group* or *collective* anomalies are subsets (groups) of data that are only anomalous when the individual instances group together, otherwise taken individually are considered to be normal (e.g., resonant and transient signals): HEP often targets these kinds of anomalies. Since both conditional and group anomalies are very different from the point ones, most existing approaches cannot be used to identify them. Therefore, one big challenge is to extend point anomaly detection methods over conditional and group anomalies, potentially exploiting multiple (related) data sources and modalities.
- **Class imbalance.** Anomalies are typically associated with rarely occurring events (this is also a common assumption), therefore there is a natural class imbalance compared to the normal instances which are the majority. The number of anomalies can

even be an arbitrary small fraction of the data, therefore collecting them can be extremely difficult and expensive due to their rarity or unpredictability: this results in the unavailability of large-scale labeled data in most applications and domains. Moreover, the misclassification cost associated with anomalous data is usually much higher than in normal instances, hence in applications where this is fundamental the method should be tuned to account for the class imbalance while minimizing such cost.

- **High false positives.** Since anomalies are rare and heterogeneous, many normal instances are wrongly marked as anomalous while true anomalies are often missed. This introduces *false positives* which can be dangerous (e.g., consider a quality check that classifies as good broken products) or costly. In particular, unsupervised AD often achieves low recall rates since the unsupervised setting is the most difficult and uncertain. For this reason, monitoring and evaluating the FPR (described in section 2.4.2) of an anomaly detection model is important. Sometimes, the decision threshold on a given anomaly score can be adjusted to achieve a suitable recall rate for the intended application.
- **High-dimensionality and not i.i.d.** High-dimensional data can contain redundant or non-linearly correlated features, causing anomalous characteristics to become unnoticeable unless compressed in a proper low-dimensional feature space. This problem is less dangerous with deep anomaly detection models because these can automatically identify a subset of discriminative non-linear relations among the input features. Indeed, one prerequisite is to train on a sufficiently large dataset. A major challenge related to high dimensionality is identifying high-order, non-linear feature interactions, and couplings among the components, potentially representing highly discriminating and expressive features for anomaly detection. Moreover, deep models often learn a more compact intermediate (or latent) representation of the inputs, thus another challenge is to ensure this representation to preserve the necessary information for accurate downstream AD. Lastly, temporal, spatial, and graph data are usually not identically distributed and independent, so another interesting challenge is to exploit these relations and dependencies to find the conditional and group anomalies, which can depend on such not i.i.d. patterns.
- **Data efficiency.** Deep neural networks are notoriously data-hungry, especially when the number of parameters is large, therefore, collecting fully labeled samples is often too costly making supervised AD impractical or ineffective. Instead, unlabeled data can be gathered more easily making semi-, self-, and unsupervised AD potential solutions to this problem. Hence, two major challenges are learning normality and anomalous representations with as few labeled data as possible, and designing anomaly detection methods to handle novel anomalies. In this regard, interesting research directions may be found at the intersection of deep anomaly detection with zero- or few-shot learning [122, 123], to improve data efficiency, and *lifelong* or *continual learning* [124], to incrementally generalize on novel classes of anomalies.

- **Anomaly explanation.** In domains in which safety and thrust are of fundamental importance (e.g., finance, cybersecurity, healthcare), anomaly detection models should be interpretable in the sense that these should provide motivations for why a specific data sample is anomalous: e.g., approaches like [125, 126] can provide either heatmaps or segmentation masks that highlight the anomalous region, if any, in the input image. Since anomalies are rare data points these may lead to algorithmic bias against other minority groups presented in the data. Therefore, anomaly explanations can enable a human expert to inspect the potential anomaly and, eventually, correct the bias. HEP is an example domain in which explaining anomalies [86] is of great value, especially for *online* triggering stages, in which data is not stored for later inspection.

Apart from the above challenges, the HEP domain poses some additional ones. Since proton-proton collisions occur at a rate of the nanosecond order, such a huge amount of data cannot be stored in its entirety and so it is subject to multiple filtering, by the trigger, which reduces the data rate enough so that the events can be stored for later *offline* analysis. This can be seen as a major limiting factor for successful new physics searches, since the triggering system may discard most interesting (potentially signal) events as it implements simple decision rules. Therefore, *real-time anomaly detection* may enable an *online* discrimination of the anomalies which can be first identified and then stored for further analysis. For example, the CMS trigger system [127] has two stages: the *Level-1 trigger* (L1), with a latency of at most a microsecond, and the *High-Level Trigger* (HLT), whose latency requirements are less strict being up to 100 ms [112]. Targeting the Level-1 trigger is the most interesting (no previous filtering has occurred) yet difficult, since an efficient hardware implementation through field programmable arrays (FPGA), for example, is required. Another difficulty is related to being robust to *detector effects* and *glitches* [128]: these are effects introduced by the collider equipment and therefore should not be considered as anomalies but rather identified and discarded, or corrected. Often the anomaly score is correlated with some physical quantity like the particle’s mass: these correlations are unwanted and may bias the remaining analysis. Decorrelation techniques, like adversarial learning [129] and DisCo [130], which is used as a regularizer that penalizes the distance between the mass and predicted class label, can be employed but a major issue regards how to decorrelate the variable of interest while preserving the discrimination performance. Lastly, the searched signal might be less complex than the background [131, 132] when considering some physical quantity (e.g., having lower total energy). This can be another major issue since we often assume the signal (treated as anomalous) to score higher than the background. Otherwise, the signal can be easily mismatched for the background. Thus, it is important to devise anomaly detection models that learn proper, disentangled representations of the background and signal to avoid ambiguities.

3.2.3 Evaluation metrics

Anomaly detection performance can be evaluated by common classification metrics (e.g., the ones presented in section 2.4.2), like the (balanced) accuracy, F1-score, and the AU-

ROC. These metrics can be complemented with other ones developed for out-of-distribution detection [133], like the false positive rate at $N\%$ of the true positive rate (FPR N), to get a more comprehensive evaluation. The AUROC summarizes the performance of the discriminator across multiple thresholds. The FPR N , instead, evaluates the performance at one specific threshold value: such threshold is often specific for the application and domain requirement.

The AUROC can be considered as the probability that a signal (i.e., anomalous) sample is assigned a higher AD score than a background (i.e., normal) example. Thus, higher AUROC values are better, depicting a higher signal retention at a lower background efficiency (and so at a higher rejection rate of the background.) Moreover, the AUROC is robust against class imbalance which is typical in anomaly detection scenarios, in which the anomalies can be either rare, novel, or unexpected phenomena or frequent unwanted, noisy, or erroneous data points.

On the other side, the FPR N metric is more suited to compare strong models. These models achieve a comparable AUROC, for which it is necessary to study the ROC (or PR) curve around a predetermined area to decree the winning model. The FPR N is interpreted as the probability that a normal (background) sample is flagged as an anomaly (so as a signal) when the $N\%$ of signals are correctly detected. Since we want to decrease such false alarm probability, lower FPR N values indicate a better model. Compared to the AUROC, this metric requires establishing a suitable value of $N\%$, i.e., the TPR rate, which is often application-specific. This also allows us to study the behavior of the target model around a given efficiency or rejection value, rather than a fixed classification threshold.

A related metric to the FPR N is the TPR@FPR, i.e., the true positive rate at a given % of the false positive rate. By fixing a value for which the model classifies a signal event erroneously as background, this metric allows us to compare the number of truly positively classified samples knowing that the amount of type I error is fixed beforehand. Given the same FPR rate, higher metric values indicate a better-performing model.

3.2.4 Convolutional neural networks

Convolutional neural networks (CNNs) [37] are a specialized kind of neural network that unlike the MLP (section 2.2), which works on tabular data, can process data arranged in a grid-like manner, like *time-series*, *audio* samples, *images*, and *videos*. The core mathematical operation that powers CNNs is the *convolution* rather than the matrix multiplication of dense networks. Considering images as an example, convolutional networks can exploit the fact that given a pixel its neighboring ones are typically related to each other: this is a powerful inductive bias that MLPs cannot leverage, and applying those neural networks on image-like data is very inefficient either in terms of performance, resources, or training time.

CNNs are at the core of most breakthroughs in both deep learning and computer vision that happened in recent years, such as *image classification* (e.g., AlexNet [134]), *object detection* (e.g., Fast R-CNN [135], YOLO [136]), *semantic* and *instance segmentation* (e.g., U-NET [24], DeepLab [137] and Mask R-CNN [138]), *depth estimation* [139], *face recognition*

(e.g., FaceNet [16]), *autonomous driving* [140], *super resolution* (e.g., SRGAN [141]), and generative tasks [142, 143, 144] as well.

Discrete convolution. A CNN is made of multiple convolutional layers, each of which applies a convolution operation between the inputs, x , and the learned *kernel*, K , outputting what we call a *feature map*, F . Considering the time index, i.e., t , to take only integer values (e.g., the indices of pixels, or the index of a sequence), the *discrete* convolution operation (denoted with an asterisk) writes as follows:

$$F(t) = (x * K)(t) = \sum_{k=-\infty}^{\infty} x(k)K(t - k). \quad (3.2.1)$$

Since in ML, the input is usually a tensor (i.e., a multi-dimensional array), we can implement the infinite summation in the above equation as a sum over a finite number of array elements. Let's consider a monochrome (e.g., grayscale) image, I , as an example: I is two-dimensional because it has dimensions along its height and width. In this case, we want to apply a two-dimensional kernel too, therefore the overall *2D* convolution operation¹ is now:

$$F(i, j) = (I * K)(i, j) = \sum_{m=1}^{K_H} \sum_{n=1}^{K_W} I(i + m - 1, j + n - 1)K(m, n), \quad (3.2.2)$$

where K_H and K_W are the kernel sizes, usually set to odd values like three or five. In practice, the above equation is applied to each *spatial locations* allowed in I , thus not only to pixel coordinates (i, j) . For an image of size $H \times W$, the spatial locations are:

$$\{(i, j) : i = 1, \dots, H - K_H + 1, j = 1, \dots, W - K_W + 1\}. \quad (3.2.3)$$

Properties [1]. A convolutional layer leverages three important ideas: sparse interactions, parameter sharing, and equivariant representations. For example, matrix multiplication in FFNNs needs a separate learnable parameter to describe an interaction between each input and output unit because each output unit interacts with every input unit. CNNs, instead, have *sparse interactions* because the kernel only interacts with a small portion of the input. This fact is of practical importance since convolutions require fewer parameters than fully connected layers, consuming less memory and being more statistically efficient. For example, if there are m inputs and n outputs, matrix multiplication requires $m \times n$ parameters, instead a convolution requires only $k \times n$ of them, with $k \ll m$. Moreover, since the same kernel is applied to each spatial location, convolutions also *share parameters* meaning that they do not require learning a separate set of parameters for each input location. Furthermore, the particular form of parameter sharing causes the convolution to develop *equivariance against translation*. Specifically, a function $f(x)$ is equivariant to

¹This operation is the *cross-correlation* which corresponds to a convolution in which the kernel is not flipped. The reason for this is that it is easier to implement by ML libraries and frameworks.

a function g if $f(g(x)) = g(f(x))$ holds, meaning that if the input changes the output changes in the same way. In practice, for time-series the convolution is as if producing a timeline that shows when different features appear in the input; if we move an event later in time, the same representation will appear in the output but later. Similarly, for images the convolution creates a grid that localizes a certain feature over the input: if the object is moved, its representation will move accordingly. Nonetheless, convolutions are only translation equivariant by design: meaning that other transformations like changes in scale and rotation would affect the representation.

Stride and padding. If we want to reduce the computational cost or downsample the output, we can modify eq. 3.2.2 to allow the kernel to skip some positions as follows:

$$F(i, j) = (I * K; s)(i, j) = \sum_{m=1}^{K_H} \sum_{n=1}^{K_W} I((i-1) \times s + m, (j-1) \times s + n) K(m, n), \quad (3.2.4)$$

where s is the *stride* indicating we want to sum only every s pixels. By default $s = 1$; instead, if we set $s = 2$ the output feature map is halved in size. Generally, applying multiple convolutions results in feature maps that shrink progressively because the borders are not preserved. This fact limits the maximum number of convolutional layers that we can apply to a given input size. To avoid that we can *zero pad* the input (or last feature map), by adding pixels to the boundaries of the image making it temporarily larger: this is sometimes referred to as *same* convolution. If padding is employed, the size of the output feature map is $(H - K_H + 1 + 2P, W - K_W + 1 + 2P)$ according to the number of added pixels, P : if $P = \lceil \frac{K_H-1}{2} \rceil$, the height of the output will be the same as the input. Therefore, padding is useful to reduce the rate at which feature maps shrink and propagate the information at the border of the images.

Pooling. A common pattern in convolutional architectures is to put a *pooling layer* after a convolution. The pooling operation aggregates a certain spatial location with a single value by applying a fixed (i.e., not learned) kernel, considering the input channels independently. Common pooling functions are the *average* and *max*, from which the name average pooling and max pooling are derived. In particular, max pooling helps the model learn a representation that is approximately invariant to small input translations: a useful property for image classification. Moreover, it promotes the propagation of only the most important features which may allow a CNN to be a good pre-trained backbone for feature extraction. Pooling is also employed as an efficient down-sampling: a common setting is to have a 2×2 kernel with $s = 2$, whose output size is half of the input, thus improving the computational efficiency of the next convolution because it will act on a much smaller input.

Popular architectures. A convolutional architecture is usually organized in *blocks* (i.e., groups of layers), arranged sequentially. A common block includes a (same) 3×3 convolution, followed by a non-linearity (e.g., ReLU), ending with a 2×2 max pooling operation.

Multiple blocks are usually applied until the spatial resolution (i.e., the width and height) of the last feature map is sufficiently small (e.g., smaller than the kernel size). At this point, a common practice is to *flatten* the feature map, that is reshaping it to a 1D tensor: e.g. if the feature map is large $F_H \times F_W \times F_C$, the flatten operation yields a tensor of shape $(1, F_H \cdot F_W \cdot F_C)$. After flattening the last feature maps, dense layers can be applied instead of convolutions to the output. In this way, the last fully connected layer can, for example, output class logits for a classification task. This design resembles the old LeNet-5 architecture [11], which can be a good default choice for simple problems. Complex tasks, instead, require a network architecture that is easy to train, capable of extracting rich features, flexible, and parallelizable.

For example, the VGG architecture [117] “factorizes” (substitutes) a single convolution with a large kernel (e.g., with sizes greater than seven) with multiple (e.g., two or three) convolutional layers with a small kernel (typically of size three or five). In this way, the overall *receptive field* of each block, i.e., the number of *input* pixels affected by the convolution operation, is as large as the costly convolution, although requiring fewer parameters and computation thanks to the adoption of a much smaller kernel.

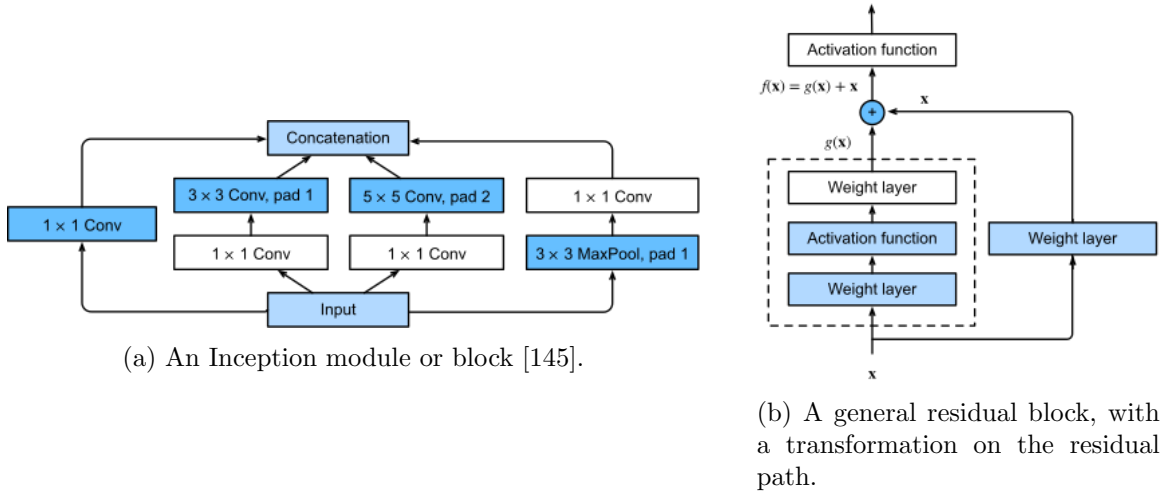


Figure 3.2: Building blocks of widely successful convolutional architectures. In principle, these ideas are not limited to convolutional networks. For example, the weight layers in (b) can also be dense or recurrent. Figures adapted from [37].

The GoogLeNet [145] focuses on efficiency, by proposing a block called the *inception module* (figure 3.2a) that combines multiple convolutions and max pooling in parallel, whose output channels are then concatenated for the subsequent block. This module also introduces *point-wise* or 1×1 convolutions [146], which act as a shared non-linear MLP applied on individual spatial locations that can reduce or enlarge the depth (channels) of the feature map: in this context, these are usually used to decrease the output channels of the single convolutions to prevent too rapid growth of them. To further diminish the number of pa-

rameters, the *global average pooling* is introduced as a replacement for the flatten operation since it removes the spatial dimensions by averaging on them; if the input of the global average pooling is a $H \times W \times C$ tensor, its output will be $1 \times C$. This allows the subsequent dense layers to have fewer connections, saving many weights.

Residual networks (or ResNets) [22, 23] introduce the idea of learning the *residual* function through *skip* or *residual connections*. Their residual block (figure 3.2b) has two convolutional layers without pooling, and the skip connection lets the input skip the two convolutions. Here, the intuition is that adding the input directly to the block's output allows it to recover the identity function, which enables the block's parameters to learn the residual instead. Moreover, skip connections allow the information (and thus the gradient) to propagate from the very end of the network, preventing the vanishing gradient problem [147] hence easing the optimization process (i.e., the loss landscape is smoothed, looking more convex [148]), allowing to train very deep networks with hundreds of layers, which can therefore fit very complex functions. Various ad-hoc modifications and improvements have been proposed for the residual blocks design [149], which can also be combined with the inception module [150]. Building on this latest idea, the ResNeXt [151] decomposes each residual block into G parallel branches that are then concatenated, in which by tuning G it is possible to balance the computational complexity of the block. This design enables better efficiency at the same complexity. Moreover, the parallel branches can be efficiently implemented by a *grouped convolution* [134, 152], in which the input channels are considered in non-overlapping groups of size G .

3.2.5 Auto-encoder models

Auto-encoders (AEs) [153, 119] are self-supervised neural networks, in which we can distinguish three major components: an *encoder* neural network, $f_\phi(\mathbf{x})$, the *latent space* (also called the bottleneck or coding layer), $\mathbf{z} \in \mathbb{R}^K$, and a *decoder* neural network, $p_\theta(\mathbf{z})$. The encoder generally compresses the (high-dimensional) input features, $\mathbf{x}$, to a K -dimensional latent representation², i.e., $\mathbf{z} = f_\phi(\mathbf{x})$, from which the decoder tries to *reconstruct* the original input, i.e., $\hat{\mathbf{x}} = p_\theta(\mathbf{z})$.

The AE aims to learn a latent representation that can describe the input data within a lower-dimensional manifold; this goal is called *compression* or *dimensionality reduction* [153]. To achieve this, the supervisory signal is the input data itself, therefore we use *self-supervision* to train the model. The reconstructions, $\hat{\mathbf{x}}$, usually exhibit simplified characteristics compared to the original input, because $\hat{\mathbf{x}} \approx \mathbf{x}$. This is often desired as AEs can also be used to *denoise* [154, 155] corrupted input samples as in figure 3.3. Other notable applications of AEs, since they embed the inputs to a dense vector, are classification, clustering, recommendation systems, and anomaly detection [119]. Moreover, AEs are commonly applied to tabular, image, and time series data.

²If K is equal to the data dimensionality, the AE is said to be *complete*; if larger it is *over-complete*, otherwise it is *under-complete*. Over-complete AEs are discouraged to use, since these may learn an exact copy of the inputs.

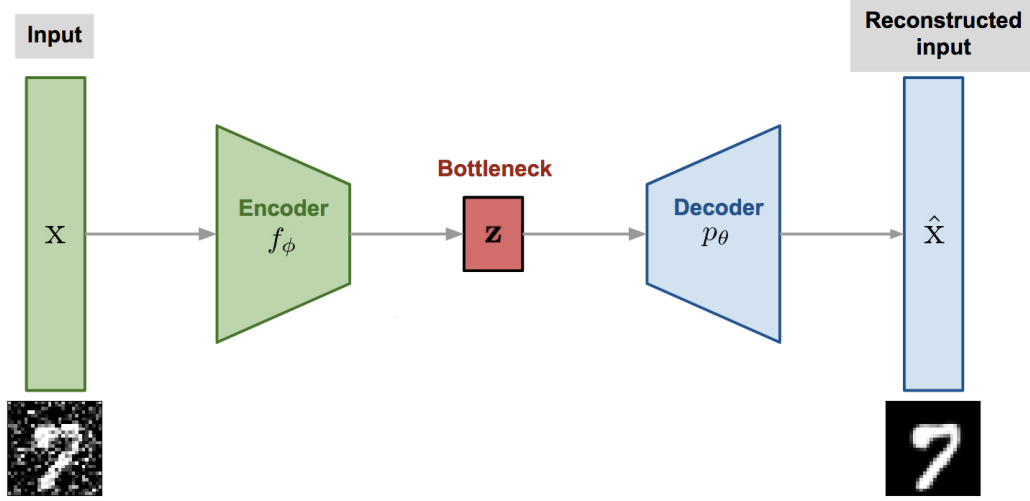


Figure 3.3: Illustration of an auto-encoder architecture, demonstrating a denoising task.

The model parameters (ϕ, θ) are jointly learned during training by minimizing a reconstruction objective or loss, such as the popular *mean squared error* (MSE):

$$\mathcal{L}_{\text{AE}}(\mathcal{B}) = \frac{1}{B} \sum_{\mathbf{x} \in \mathcal{B}} \left\| \mathbf{x} - p_{\theta}(f_{\phi}(\mathbf{x})) \right\|_2^2, \quad (3.2.5)$$

where $\mathcal{B} = \{\mathbf{x}^{(i)}\}_{i=1}^B$ is a mini-batch of size B . The MSE loss can be applied to arbitrary value ranges, instead if the input features were in $(0, 1)$ (as if subject to sigmoid activation) we can use the binary cross-entropy loss instead. Many variations of the above formalism exist, such as:

- **Denoising AEs [154].** The inputs are assumed to be *corrupted* by some form of noise, i.e., $\tilde{\mathbf{x}} \sim \mathcal{M}(\tilde{\mathbf{x}} | \mathbf{x})$, and so the model is asked to recover the original input, $\mathbf{x}$. The corruption process can be arbitrary and is controlled by a distribution, $\mathcal{M}(\tilde{\mathbf{x}} | \mathbf{x})$, which can be equipped with prior knowledge about the noise source and type. The noise has a regularization effect forcing the AE to learn *robust* latent features that represent the original input.
- **Sparse AEs [156].** A sparse constraint is applied to the latent space ensuring only the most important components are non-zero. A penalty term like the l_1 or Lasso regularization is added to the loss function: $\mathcal{L}_{\text{SAE}} = \mathcal{L}_{\text{AE}}(\mathcal{B}) + \lambda \|\mathbf{z}\|_1$, where λ is a hyperparameter that controls the strength of the sparsity constraint: if λ is too high, most of the latent components would be zero yielding poor reconstructions, otherwise if too low the constraint may be ignored. Sparsity is a way to mitigate overfitting and improve robustness, for example in cases where there are many (redundant) latent components.

- **Contractive AEs [157].** The aim is to learn a compact and robust representation by introducing a loss penalty based on the Frobenius norm of the Jacobian matrix of the encoder's activations w.r.t. the input, such that the model is less sensitive to input variations. The penalty term corresponds to the sum of squares of all partial derivatives of the learned encoding along the input dimensions.
- **Convolutional AEs [158].** Typically, the latent space is a vector (i.e., a rank-1 tensor). To better exploit the spatial correlations of image-like data (i.e., rank-3 tensors), e.g., along the width, height, and channel dimensions, we can learn a *convolutional bottleneck* that has the same rank of the inputs, i.e., three. Furthermore, when designing CAEs the shape of the convolutional latent space can have a non-trivial impact, in terms of faster training, better reconstructions, wider generalization, and improved data representation (e.g., useful for downstream tasks such as anomaly detection) [159].
- **Adversarial AEs [160].** It is a probabilistic auto-encoder, in which a third *adversarial network* attached on top of the latent space is responsible for regularizing the whole model by matching the aggregated posterior, $q_\phi(z)$, to an arbitrary prior, $p(z)$. Both the adversarial network and the auto-encoder are jointly trained in two phases: the reconstruction phase, in which the AE attempts to minimize the reconstruction error; and the regularization phase, in which the adversarial network first updates its discriminative network to tell apart the true samples (generated using the prior) from the generated samples (computed by the AE), to update the encoder such that to confuse the discriminative network. These AEs are expected to learn a more robust latent representation of the data.

AEs are widely applied in semi-supervised classification, unsupervised clustering, dimensionality reduction, and data visualization.

3.2.6 Variational auto-encoders

The variational auto-encoder (VAE) [161, 119] is a *deep generative model* yield by the combination of *probabilistic graphical models* [162] with methods of *variational Bayesian* [163], which allows to infer the value of random variables by solving an optimization problem. Compared to a standard AE, the VAE attempts to map the input into a *probability distribution* (e.g., a Gaussian), which now governs the latent space, rather than a *fixed* vector.

Given a dataset, $\mathcal{X}$, of i.i.d. samples, the generative process (illustrated in figure 3.4) we want to capture with a VAE is the following: the probability of a data point, $x^{(i)}$, to occur (i.e., to be generated) is conditioned on an unobserved latent variable, $z^{(i)}$, which encodes its generating factors, captured by a *probabilistic decoder* (the generative model), $p_\theta(x | z)$ with parameters θ ; the reverse process of inferring the latent variable given the sample, is called recognition or inference, which is implemented by a *probabilistic encoder*, $q_\phi(z | x)$ with parameters ϕ , that represents an approximate posterior distribution. Moreover, the latent variables are assumed to follow a *prior distribution*, $p(z)$, which is a way to encode our

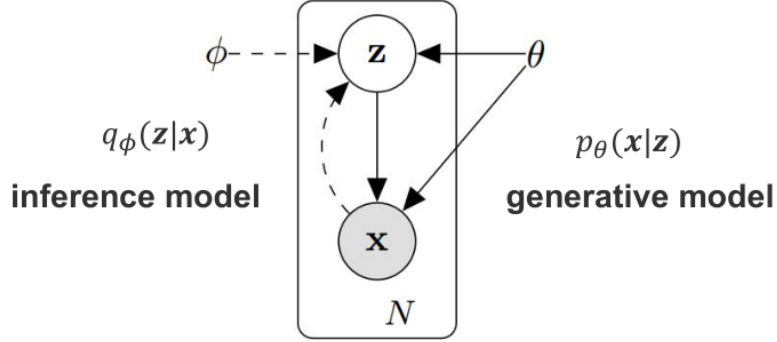


Figure 3.4: The VAE represented as a directed graphical model [161]. Solid lines denote the generative or decoder model, while dashed lines denote the inference or encoder model.

assumptions about the nature of the factors that could have generated the data. Compared to standard AEs, VAEs follow a probabilistic framework while introducing an additional component: the latent space prior.

Evidence lower bound [164, 163]. The goal of training a VAE is to find an approximation of the true data generating distribution, i.e., $p(x) = \int p_\theta(x | z)p(z) dz$, which is unfortunately unknown and it is also prohibitive to marginalize over all possible values of z . To facilitate the search, we learn an approximation of this probability through $q_\phi(z | x)$, i.e., the estimated posterior, using the samples provided by our dataset, $\mathcal{X}$. We want to ensure the learned posterior is close to the real one, i.e., $p_\theta(z | x)$. To do so, we use the *Kullback-Leibler (KL) divergence* (also called relative entropy) [165] that quantifies the distance³ between two probability distributions: $D_{\text{KL}}(p \parallel q)$ measures the amount of information (in nats) required to transform the distribution q into p . In particular, we want to minimize the *reverse KL*, i.e., $D_{\text{KL}}(q_\phi(z | x) \parallel p_\theta(z | x))$, with respect to parameters ϕ .

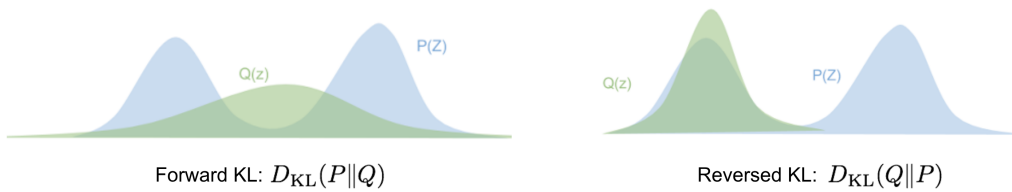


Figure 3.5: Forward and reverse KL divergences induce the approximating distribution, $Q(Z)$, to match the target distribution, $P(Z)$, in two complementary ways. Figure adapted from <https://lilianweng.github.io/posts/2018-08-12-vae>.

³The KL divergence is not a proper distance metric since, in general, $D_{\text{KL}}(p \parallel q) \neq D_{\text{KL}}(q \parallel p)$ but it is zero if $p = q$, and it is always non-negative.

From figure 3.5 we can see that: the forward KL ensures the approximating distribution to be positive, i.e., $Q(Z) > 0$, whenever $P(Z) > 0$, encouraging $Q(Z)$ to cover the entire probability mass of $P(Z)$; the reverse KL, instead, exhibits a mode-seeking behavior trying to cover an arbitrary mode of the target distribution by forcing $Q(Z) = 0$ whenever $P(Z) = 0$. In both cases we have advantages and disadvantages, especially when dealing with multi-modal distributions: in the former case a sort of average distribution is obtained, which covers the entire probability mass but misses the modes; in the latter case, though, the ignored mode can induce false negatives since there is probability mass where we think there is none.

Since $p_\theta(z | x)$ is unknown, it turns out that we can rewrite our reverse KL divergence in terms of the prior as follows:

$$D_{\text{KL}}(q_\phi(z | x) \parallel p_\theta(z | x)) = \log p(x) + D_{\text{KL}}(q_\phi(z | x) \parallel p(z)) - \mathbb{E}_{z \sim q_\phi(\cdot | x)} [\log p_\theta(x | z)].$$

The above equation can be rearranged such that

$$\underbrace{\log p(x) - D_{\text{KL}}(q_\phi(z | x) \parallel p_\theta(z | x))}_{\text{VAE's objective}} = \mathbb{E}_{z \sim q_\phi(\cdot | x)} [\log p_\theta(x | z)] - D_{\text{KL}}(q_\phi(z | x) \parallel p(z)), \quad (3.2.6)$$

the left-hand side is the objective we want to maximize, i.e., we want to optimize the log-likelihood of generating real data, $\log p(x)$, while minimizing the distance between the real and estimated posterior (the D_{KL} term). In practice, this is done by maximizing the reconstruction quality (or, equivalently, reducing the reconstruction error), i.e., the term $\log p_\theta(x | z)$, and regularizing the posterior to match the prior by minimizing the KL divergence term. Finally, by negating equation 3.2.6 we obtain the loss function to train our VAE model:

$$\mathcal{L}_{\text{VAE}}(\theta, \phi) = -\mathbb{E}_{z \sim q_\phi(\cdot | x)} [\log p_\theta(x | z)] + \beta D_{\text{KL}}(q_\phi(z | x) \parallel p(z)). \quad (3.2.7)$$

Notice the introduction of a hyperparameter, β , responsible for balancing the reconstruction error against the regularization effect of the KL divergence. This more general formulation is called the beta-VAE [166]. Because the two terms in the loss function have *contrasting* effects, tuning β to achieve high fidelity in both reconstructions and generated samples is important. In particular, the reconstruction loss (e.g., MSE) pushes to improve the reconstruction quality while ignoring the structure of the latent space; conversely, the KL divergence introduces a regularizing effect that normalizes and smooths the latent space towards the prior. Therefore, if not properly tuned, the KL term can induce a sub-optimal use of network capacity where only a limited set of latent variables are exploited for sample generation: this phenomenon is called *variable* or *posterior collapse* [167]. Moreover, lower values of β improve reconstruction quality but *not* of the generated samples, since the shape of the latent space is not exploited. On the contrary, higher values of β may impact reconstruction quality resulting in more blurred images. A practical way to balance the two terms is to let β be a running average of the reconstruction loss, estimated on mini-batches,

which acts as a normalizing factor [168]. In this way, the balancing is constant and adaptive during training.

In Variational Bayesian methods, from which the VAE is also built, the loss function denoted in equation 3.2.7 is also known as the *variational lower bound*, or the *evidence lower bound* (ELBO) since the loss is the lower bound of the true data generating probability, that is:

$$-\mathcal{L}_{\text{VAE}}(\theta, \phi) = \log p(\mathbf{x}) - D_{\text{KL}}(q_{\phi}(\mathbf{z} | \mathbf{x}) \parallel p_{\theta}(\mathbf{z} | \mathbf{x})) \leq \log p(\mathbf{x}), \quad (3.2.8)$$

because the KL divergence between the estimated and true posterior is always non-negative. Therefore, the minimization of the loss function, $\mathcal{L}_{\text{VAE}}$, corresponds to the maximization of the lower bound of the probability of generating realistic data samples.

Reparameterization trick [164]. In the previous paragraph the ELBO and the corresponding loss function were introduced. The loss, $\mathcal{L}_{\text{VAE}}$, contains an expectation term related to the reconstruction error, i.e., $\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}[\log p_{\theta}(\mathbf{x} | \mathbf{z})]$, in which we need to sample latent vectors, i.e., $\mathbf{z} \sim q_{\phi}(\cdot | \mathbf{x})$, to generate new samples. Since sampling is a stochastic process, we cannot back-propagate the gradient through it as the operation is not deterministic. To enable the model to be trainable, the sampling of the random variable must be reformulated as a deterministic operation, allowing the gradient to flow through it. This is called the *reparameterization trick*, in which the random variable is expressed as $\mathbf{z} = \mathcal{T}_{\phi}(\mathbf{x}, \epsilon)$, where ϵ is an auxiliary independent random variable and $\mathcal{T}_{\phi}$ a transformation parameterized by ϕ .

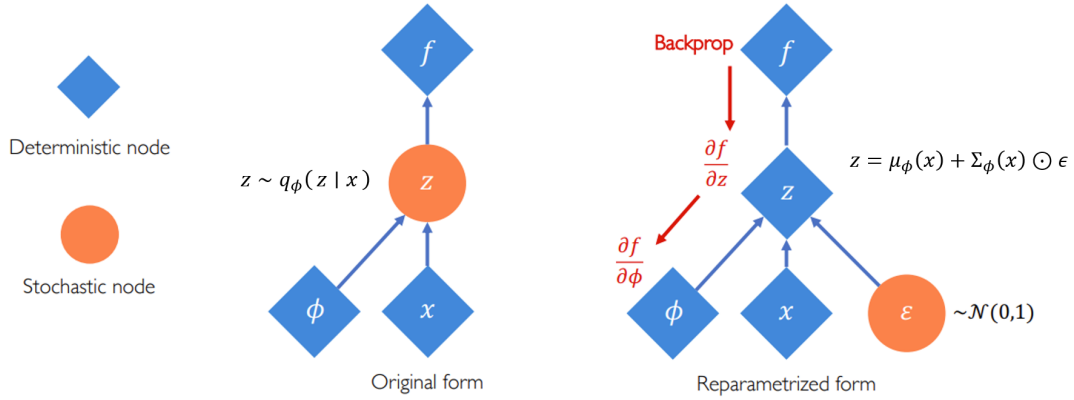


Figure 3.6: Graphical explanation of the reparameterization trick. In the original computation graph, the gradient cannot flow because it is blocked by the stochastic node located in the middle of the back-propagation path. In the reparameterized version, the stochastic operations (i.e., the sampling of ϵ) are rearranged to create a deterministic path to the parameters, ϕ , free of stochastic operations.

Assuming the variational posterior to be a multivariate Gaussian with diagonal covariance

(a common choice in VAEs), i.e., $\mathcal{N}(\mu, \sigma^2 I)$ where I is the identity matrix, we have that:

$$z \sim q_\phi(z | x^{(i)}) = \mathcal{N}(z | \mu^{(i)}, (\sigma^2)^{(i)} I), \quad (3.2.9)$$

for an input sample $x^{(i)}$, in which the encoder network outputs the parameters, μ and $\sigma^2 I$, as a non-linear function of the input, to parameterize a diagonal Gaussian⁴ from which we sample the latent codes. In practice, the mean, μ , is free to vary, whereas the standard deviation should be strictly positive, i.e., $\sigma > 0$, which is commonly ensured by applying a softplus⁵ or exponential activation, otherwise, it is considered as being $\log \sigma^2$. In the Gaussian case, the reparameterization trick is quite simple (as depicted in figure 3.6), in which the sampling of the latent vectors is rewritten as

$$z = \mu + \sigma \odot \epsilon, \text{ where } \epsilon \sim \mathcal{N}(0, I), \quad (3.2.10)$$

a deterministic operation (i.e., mean plus noisy variance), in which the stochastic part is represented by the element-wise product of the learned standard deviation with the *noise*, ϵ , sampled from a standard Normal distribution. In this way, back-propagation goes through only μ and σ since outputted by q_ϕ , for which there is a dependency on the inputs and, so, the sampling of the noise is ignored during the backward pass. Since the latent space is assumed to follow a multivariate Gaussian, we should select a suitable latent space prior such as an *isotropic Gaussian*, i.e., $p(z) = \mathcal{N}(0, I)$: in this case, the KL divergence simplifies to

$$D_{\text{KL}}(\mathcal{N}(\mu, \sigma^2) \parallel \mathcal{N}(0, I)) = \frac{1}{2} \sum_{j=1}^K \left(1 + \log \sigma_j^2 - \mu_j^2 - \sigma_j^2 \right), \quad (3.2.11)$$

where K is the dimension of the latent space, and $\mu, \sigma = q_\phi(x)$. Lastly, the reparameterization trick is not limited to the Gaussian case as it can also be designed for other distributions: in the next sections, we will discuss an example.

3.3 State of the Art

This section reviews the relevant literature about (deep) anomaly detection. We first provide an overview of popular deep learning and computer vision approaches, then conclude with methods and applications specifically developed for high-energy physics.

3.3.1 Anomaly detection in deep learning

Deep learning enabled to learn effective representations of high-dimensional data, allowing numerous applications of anomaly detection not limited to *intrusion detection*, *fraud detection*, *industrial fault and damage detection*, *medical diagnosis*, and *scientific discovery*

⁴For a more expressive latent space we could even learn a Gaussian with a full-covariance matrix, Σ , but at the cost of learning K^2 more parameters, where K is the size of the latent space. A lower triangular covariance matrix is a more parameter-efficient choice. Instead, the diagonal case (i.e., $\Sigma = \sigma^2 I$) is the most efficient with linear growth in K .

⁵ $1 + \text{softplus}(x) = 1 + \log(1 + e^x)$. Adding one ensures avoiding negative values for small x .

in chemistry, bioinformatics and physics, which leverage complex data modalities such as images, videos, audios, text, graphs, multivariate time series, and biological sequences. The main groups of approaches to anomaly detection are reconstruction models (e.g., AEs), probabilistic models (e.g., VAEs and GANs), and one-class classifiers [169].

To improve the clustering and anomaly detection performance based on the learned latent space representation of AEs, Aytekin et al. [170] propose to apply an l_2 -normalization during training claiming the obtained representations to be more separable and compact in Euclidean space. The l_2 -normalization is applied on the activations of the encoder’s representations, i.e., $z = f_\theta(x)$, such that

$$z' = \frac{z}{\|z\|_2}, \quad (3.3.1)$$

which imposes a unit norm constraint on them. This is expected to improve downstream methods based on the Euclidean distance (e.g., k -means [171]) because the distance between latent vectors is now independent of their length, depending only on the angle between them. Building on this idea, the authors propose a *normality score* for anomaly detection in which the dot product between z' and the cluster centroids (yield by running k -means on z') is treated as a measure of similarity, marking samples as anomalous if the score is below a threshold.

Zimmerer et al. [125] are the first to combine reconstruction- with density-based anomaly scores in medical images. They augment a VAE with a *context encoder* (CE), i.e., a specialized denoising AE that masks out local patches of the input image instead of applying noise. Masking aims to induce the learning of semantically richer and more robust representations, in the spirit of SSL. Their ceVAE model has two branches: the first is a regular VAE, while the second is a CE branch trained to reconstruct the masked inputs by using the same encoder and decoder networks. The CE branch introduces a reconstruction loss term added to the VAE objective, i.e., eq. 3.2.7. The ELBO (eq. 3.2.8), which combines the KL and reconstruction terms, is used for anomaly detection as an estimate of the probability of a data sample, i.e., $p(x)$, to occur under the unknown data distribution. Lastly, the authors propose a pixel-wise anomaly score, scoring each reconstructed pixel to localize and interpret anomalies.

Abati et al. [77] tackle the problem of novelty detection with a deep auto-encoder equipped with a parametric density estimator that learns the probability distribution underlying its latent space. The density, $p(z)$, is estimated via an auto-regressive process, thus avoiding adopting a specific family of distributions. The authors argue that modeling normal data should converge to a model that exhibits low differential entropy, leading to the extraction of easily predictable features. To this end, they propose a *novelty score* that represents the sum of the reconstruction term and the estimated log-likelihood term, i.e., $\log p(z)$, each of them independently min-max normalized⁶ using a reference set of samples. The proposed score improves over the individual terms.

⁶Given scores L and a set of indices S , a single score L_i is normalized as follows: $L'_i = \frac{L_i - \max_{j \in S} L_j}{\max_{j \in S} L_j - \min_{j \in S} L_j}$.

Hendrycks et al. [133] utilizes an auxiliary dataset of outliers to improve the performance of deep anomaly detection. Their approach, called Outlier Exposure (OE) involves training a model, f , to learn heuristics to detect whether a sample is from $\mathcal{D}_{\text{in}}$ (i.e., in-distribution dataset) or $\mathcal{D}_{\text{OE}}$ (i.e., out-of-distribution dataset) by changing the original learning objective, $\mathcal{L}$, as follows

$$\mathcal{L}'(\mathcal{D}_{\text{in}}, \mathcal{D}_{\text{OE}}) = \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}_{\text{in}}} \left[\mathcal{L}(f(\mathbf{x}), \mathbf{y}) + \lambda \mathbb{E}_{\mathbf{x}' \sim \mathcal{D}_{\text{OE}}} [\mathcal{L}_{\text{OE}}(f(\mathbf{x}'), f(\mathbf{x}), \mathbf{y})] \right]. \quad (3.3.2)$$

In cases where labels are unavailable, the $\mathbf{y}$ are ignored in the above loss. In practice, for classification tasks, the predictions on the OE-set, i.e., $f(\mathbf{x}')$, are encouraged to match a uniform distribution, e.g., via an additional cross-entropy loss. Otherwise, for density estimation the $\mathcal{L}_{\text{OE}}$ is set to a margin ranking loss on the log-probabilities between $f(\mathbf{x}')$ and $f(\mathbf{x})$. OE can also be applied as a fine-tuning step on off-the-shelf pre-trained models. In a similar spirit, DeVries and Taylor [172] propose to add a confidence branch to estimate how confident the network is at predicting a given sample. Assuming multi-class classification, they interpolate the predicted classes according to the estimated confidence, $c \in (0, 1)$, and then augment the objective with a confidence loss as follows:

$$\mathcal{L} = - \underbrace{\sum_{i=1}^C \log(c \cdot p_i + (1 - c)y_i)}_{\text{classification loss}} y_i + \underbrace{-\lambda \log(c)}_{\text{confidence loss}}, \quad (3.3.3)$$

where C is the number of classes, and p_i the predicted logits for the i -th class. Once the model is trained, the confidence estimates, c , can be directly used to detect out-of-distribution samples if the confidence is below a given threshold.

Schlegl et al. [126] propose AnoGAN for the medical domain, i.e., a deep convolutional GAN [120, 173] that learns a manifold of normal data, and a novel anomaly score based on the mapping from image space to latent space. The proposed anomaly score

$$A(\mathbf{x}) = (1 - \lambda)R(\mathbf{x}) + \lambda D(\mathbf{x}), \quad (3.3.4)$$

combines a *residual score*, $R(\mathbf{x})$, which measures the dissimilarity (in image space) between the query and generated images, with a *discrimination score*, $D(\mathbf{x})$, which quantifies the degree to which the features of the query and generated images match. Given a query image, $\mathbf{x}$, to determine whether this is anomalous or not the AnoGAN searches for a latent vector, $\mathbf{z}$, such that the generated image, i.e., $G(\mathbf{z})$ by a generator network G , is as similar as possible to the query. The search starts by sampling a random $\mathbf{z}^{(i)}$, generating an image, $G(\mathbf{z}^{(i)})$, and then based on the generated image the latents are updated, yielding a new vector, $\mathbf{z}^{(i+1)}$, which is used to repeat the process until convergence. The latent vector obtained at the last iteration, say $\mathbf{z}^{(\gamma)}$, is plugged in eq. 3.3.4 yielding the anomaly score for the query image, $\mathbf{x}$. Furthermore, the residual image, i.e., $|\mathbf{x} - G(\mathbf{z}^{(\gamma)})|$, can be used to identify abnormal regions within an image. The main disadvantage of AnoGAN is its computational inefficiency in the iterative search of $\mathbf{z}^{(\gamma)}$, therefore f-AnoGAN [174] introduces a deep encoder network that learns the inverse mapping, i.e., $E(\mathbf{x}) : \mathbf{x} \rightarrow \mathbf{z}$, such that to avoid the costly iterations.

Another interesting idea introduced by Kwon et al. [175] is to use the *back-propagated gradients* as a representation of the input samples, which is based on the assumption that anomalous data requires more model updates to characterize them fully compared to normal instances. Therefore, the gradient-based anomaly score measures how many updates the input sample requires compared to the normal data to score a new data point. Compared to the usual activation-based representation, the gradient-based one can be extracted at various levels of data abstraction, i.e., at different levels of the model’s hierarchy of layers. This implies that, in principle, it is possible to capture abnormalities in low-level learned features and high-level class semantics. Since the gradient-based representation also provides directional information it can be combined with the activation-based representation to improve the overall anomaly detection.

Lastly, methods based on one-class classification avoid learning a full density, aiming instead at directly fitting a decision boundary that yields low error when applied to unseen data. For these methods, the objective is to learn a one-class decision boundary that minimizes both the false alarm for normal instances (type I error) and the miss ratio of the true anomalies (type II error). Since it is possible to achieve a zero false alarm rate by drawing a boundary that encloses all data points, in practice a target false alarm rate, $\alpha \in [0, 1]$, is minimized instead. A notable method is the one-class Deep SVDD [176], in which a feature representation of the data, i.e., $\phi_w(x) \in \mathbb{Z}$ is learned such that to minimize the squared distance over all data points towards a center, $c \in \mathbb{Z}$. The main issue in deep one-class classification is how to regularize the model to avoid a feature map collapse, i.e., $\phi_w \equiv c$, resulting in a trivial solution [169].

3.3.2 Anomaly detection in HEP

In the context of HEP, anomaly detection is mainly used to power new physics searches, but also for monitoring the detectors [177], which involves the measurement of physical properties of proton collisions attempting to detect irregular behavior that is outside of the expected set of statistics: the aim is to identify and localize detector anomalies that can affect both the (sub-)detector(s) performance and the quality of the collected data. In this thesis, the focus is on searching new physics through AD, which broadly categorizes in *outlier detection* [85, 131, 178] and *finding over-densities* [179, 180, 181, 182]. Outlier detection identifies rare and unexpected events that stand out from the norm, usually found in the tails of distributions. These outliers can hold valuable insights into the fundamental nature of particle interactions. Instead, finding over-densities (or resonances) involves looking for localized excesses in the data compared to the expected distribution, which may indicate the presence of new particles or unexpected physical processes [112].

Weakly supervised methods. Weak supervision entails noisy labels [118]: the aim is to classify two sets of mixed data samples, A and B , that are composed of the same two classes (one for signal, and zero for background) but with different proportions. The class-label

probability for the mixtures, A and B , are respectively given by

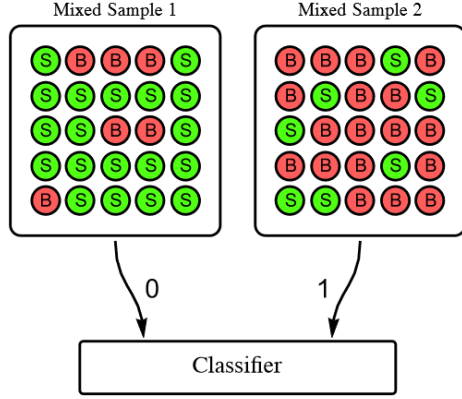
$$\begin{aligned} p_A &= s_A p_{\text{sig}} + (1 - s_A) p_{\text{bkg}} \\ p_B &= s_B p_{\text{sig}} + (1 - s_B) p_{\text{bkg}}, \end{aligned} \quad (3.3.5)$$

where s_A and s_B are unknown signal fractions, and p_{bkg} and p_{sig} the true (unknown) class label probability of background and signal, respectively. If $s_A > s_B$ the mixture A is given the noisy label of one (i.e., being signal-like), and the mixture B is assigned the noisy label of zero (background-like). Then, a classifier trained on these noisy labels will learn a function monotonically related to the likelihood ratio p_A/p_B which, in turn, is monotonically related to $p_{\text{sig}}/p_{\text{bkg}}$, therefore optimal. The Classification Without Labels (CWoLa) [183] is a framework that allows learning from partial, or imperfect labels, which can be applied to a variety of classifiers: the only requirement is that the classifier's output can be thresholded to vary the working point, as it is the case for continuous-output models like neural networks. The CWoLa approach fits a model to classify the mixed samples A and B from one another, rather than directly classify the signal from the background: figure 3.7a. This procedure results in an optimal classifier, $h(\mathbf{x}) = p_{\text{sig}}(\mathbf{x})/p_{\text{bkg}}(\mathbf{x})$, although *asymptotically* (i.e., in the limit of an infinite number of training events), able to discriminate the signal from the background without knowing in advance the class-label proportions: s_A and s_B . One disadvantage of this approach is that the mixed datasets require a careful design, especially for a model-independent search [112]. However, a main advantage is that this method is insensitive to coordinate transformations [87], as shown in figure 3.7b, since a likelihood ratio is learned.

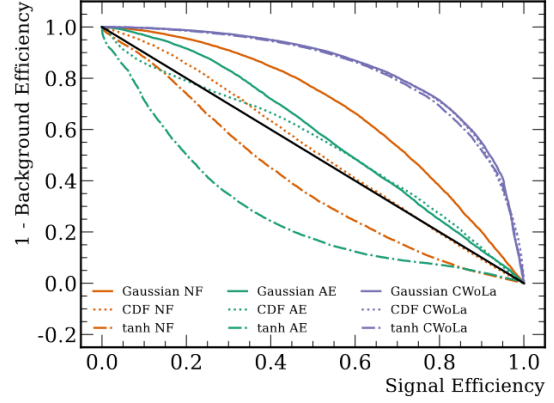
Another approach called ANODE (anomaly detection with density estimation) [181] leverages *masked auto-regressive flows* (MAF) [184] to estimate the conditional probability density of the data in a signal region and its sidebands. The MAF estimates the densities by using a succession of neural networks to gradually transform the samples of a simple distribution (e.g., a Normal) to the target data distribution. In this way, it is possible to estimate the background contribution from the learned density and thus construct a likelihood ratio of data versus background, which can be used for anomaly detection. The ANODE method assumes a feature, say m (e.g., the invariant mass), in which the hypothetical signal can be localized around a point, m_0 . Given some offset, δ , the *signal region* (SR) is defined as the interval $[m_0 - \delta, m_0 + \delta]$, and the *sideband region* (SB) as $m \notin [m_0 - \delta, m_0 + \delta]$. To perform a model-agnostic bump hunt in m , ANODE attempts to learn two densities, $p_{\text{data}}(\mathbf{x} | m)$ and $p_{\text{bkg}}(\mathbf{x} | m)$, for $m \in \text{SR}$ and given input features, $\mathbf{x}$. The classification is then performed by using the likelihood ratio:

$$R(\mathbf{x} | m) = \frac{p_{\text{data}}(\mathbf{x} | m)}{p_{\text{bkg}}(\mathbf{x} | m)}. \quad (3.3.6)$$

In the absence of the signal (assuming perfect estimations), $R(\mathbf{x} | m) = 1$, otherwise the presence of the signal can be enhanced in SR by applying a threshold: i.e., $R(\mathbf{x} | m) > c$ for $c > 1$. Compared to CWoLa, ANODE does not require independence between $\mathbf{x}$ and m to achieve signal sensitivity, working even when strong correlations are present.



(a) The CWoLa framework.



(b) CWoLa is robust under coordinate transforms.

Figure 3.7: Classification without labels. On the left, the main idea behind CWoLa is pictured in which the classifier learns from samples mixed with different proportions of labels. On the right, CWoLa is compared to both AE and normalizing flows (NF): the original coordinate space is Gaussian, and only the CWoLa method (purple curves) is robust to both cumulative density function (CDF) and tanh coordinate transformations as the ROC curves demonstrate.

CATHODE (i.e., classifying anomalies through outer density estimation) [182] improves over ANODE by training a MAF outside the signal region (i.e., in sidebands), thus learning $p_{\text{data}}(x \mid m \notin \text{SR}) \approx p_{\text{bkg}}(x \mid m \notin \text{SR})$ which approximates the background density. At this point, the learned density on SB is then interpolated to the SR, from which background-like events are sampled using the MAF inverse map. Next, similarly to CWoLa, a classifier is trained to distinguish the generated events (expected to follow the interpolated background distribution in SR) by the simulations, which contain background plus signal in the SR. Lastly, the learned classifier is applied to data in SR for anomaly detection: hence the events classified as data-like should be signal enriched, regardless of the signal. Compared to CWoLa and ANODE, CATHODE usually outperforms them because: 1) it avoids learning the likelihood of the data in the signal region; 2) it can oversample the events from the learned background density, enhancing classification performance; 3) since the background-like events in SR are the result of interpolation by the MAF (because not trained on that region), CATHODE is expected to be more robust against correlations between x and m .

In the next two paragraphs self-supervised anomaly detection methods built on auto-encoders are described and discussed. Since AEs embed the input in a latent space that is successively reconstructed, it is possible to derive anomaly scores that are, respectively, obtained through reconstructions and from the latent space.

Reconstruction-based methods. Reconstruction-based anomaly scores are obtained by comparing the reconstructions $\hat{x} = p_\theta(f_\phi(x))$ in case of AEs, or $\hat{x} = p_\theta(q_\phi(z | x))$ for VAEs, with the original inputs, x . Different scores can be determined according to the distance or similarity function used to compare images, and each of them will capture a different notion of normality or abnormality resulting in different sensitivity to the anomalies. For example, popular anomaly scores are the:

- **MSE** scores, defined in terms of the mean squared error:

$$\text{MSE}(x, \hat{x}) = \frac{1}{D} \sum_{i=1}^D (x_i - \hat{x}_i)^2, \quad (3.3.7)$$

where the summation runs over the output dimensions, D . A disadvantage of this score is that outliers can easily influence the metric value due to the squared differences, especially if the compared inputs are not normalized in a reasonably small interval.

- **MAE** scores, defined in terms of the mean absolute error, which is less sensitive to large outlier values compared to the MSE:

$$\text{MAE}(x, \hat{x}) = \frac{1}{D} \sum_{i=1}^D |x_i - \hat{x}_i|. \quad (3.3.8)$$

Both MSE and MAE scores are generally applicable to arbitrary value ranges.

- **BCE** scores, defined in terms of the binary cross-entropy loss, assuming the individual values to be the output of a sigmoid activation (i.e., constrained within zero and one.) For VAEs, instead, the metric can be interpreted as if assuming an independent Bernoulli distribution on each output dimension, i.e., $\hat{x}_i \sim \text{Bern}(k; p) = p^k(1 - p)^{1-k}$.

$$\text{BCE}(x, \hat{x}) = - \sum_{i=1}^D x_i \log \hat{x}_i + (1 - x_i) \log(1 - \hat{x}_i). \quad (3.3.9)$$

The following popular approaches employ the above scores and their modifications. Heinel et al. [85] introduce the benchmark dataset of QCD vs top jets, in which they studied the problem from a particle-based point of view, training auto-encoders to reconstruct either jet images or 4-momenta vectors. Their approach heavily relies on a specific pre-processing that transforms raw collisions into particles (high-level physics objects), greatly simplifying the problem: the jets are first centered, then rotated and mirrored, before yielding 40×40 cropped images, which are further normalized to sum to one. They demonstrate that their LoLa auto-encoder based on 51 extended four-vectors, which leverages knowledge about jet-level kinematics, can beat the image-based AE by a large margin even with a smaller latent space, although at the cost of introducing an even larger dependency on the jet invariant mass. The correlation with a physics observable, like the mass, is unwanted.

Hence, the authors train an additional network (the adversary) to decorrelate the jet mass; the adversary predicts the jet mass, minimizing an adversarial loss. In this way, decorrelation is achieved but the loss-balancing coefficient should be carefully tuned to avoid a too large drop in discrimination performance, which was still observed.

Finke et al. [131] discuss some more limitations of using AEs on the dataset. The authors claim that “due to the sparsity and the specific structure of the jet images, the auto-encoder fails to tag QCD jets if it is trained on top jets”. This problem is known as *complexity bias*, in which given a certain representation or observable the signal appears to be less complex than the background; this issue is partially motivated by a correlation between the reconstruction loss and the number of non-zero pixels in the images, as the loss tends to increase when the number of non-zero pixels is higher. For this reason, the authors introduce the kernel-MSE loss function, which is less sensitive than MSE while encouraging the network to learn the structure of the inputs. The authors also compare the performance of their AEs against various intensity remapping functions, e.g., $R(x) = \sqrt{x}$, used to alter the pixel values of the input images. Both improvements enabled the authors’ AE to learn even dim pixels.

Recently, Dillon et al. [178] propose to use a *normalized auto-encoder* (NAE) [185] to overcome the complexity bias, thus identifying anomalous jets symmetrically⁷. The NAE is trained to maximize the likelihood of the data by minimizing an energy function (the reconstruction error), used to define a Gibbs distribution representing its probability density. Under this probabilistic formulation, the NAE is forced to inhibit the reconstruction of an outlier since it has to maximize the likelihood of the training data, guaranteeing low reconstruction error only for such data. In practice, the authors first pre-train the model with a MSE loss, then switch to the NAE loss, reconstructing images on which a Gaussian filter is applied to reduce sparsity. Although, NAEs are well-suited for anomaly detection, avoiding their training instabilities is still a practical challenge.

Kasieczka et al. [87] point out that since auto-encoders indirectly learn a probability density (where samples with high probability are reconstructed well, and the ones with low probability poorly), these class of models is *not invariant under coordinate transformations* meaning that the sensitivity of the signal can depend on the initial choice of the coordinates, thus impacting anomaly detection performance: this is also true for anomaly detection methods based on density estimation, like the ones powered by *normalizing flows* [186]. Instead, weakly supervised methods that learn a likelihood ratio, as they demonstrate, are independent of invertible coordinate transformations. Being not invariant under coordinate transformations implies that the choice of the transformation can be optimal for one set of signals but sub-optimal for others, which is unwanted in signal-agnostic searches. Therefore, this fact causes some transformed signal data to overlap with the background as if those were not anomalous. Moreover, we get different anomaly selections (i.e., given the same cut, diverse sets of events are selected) depending on the coordinates used to represent

⁷A model can identify two classes (say A and B) of data symmetrically if there is no substantial drop in performance by discriminating A from B or B from A. This test checks whether switching the classes during training affects classification performance.

the events. For those reasons, further research is needed to understand whether or not auto-encoders can be robust under coordinate transforms.

Latent-based methods. Latent-based anomaly scores are defined from the latent space captured by the encoder network. There are two general approaches: using a latent component as an anomaly score, or the KL divergence between the learned posterior and the prior instead. In the former approach, directly using the learned latent representation to tag anomalies can be difficult due to its high dimensionality. Some components may be redundant, carrying very little discriminative information. Therefore, effectively combining the information brought by each latent component may require explicit supervision. In the case of Gaussian VAEs, we can use the KL divergence, as defined in equation 3.2.11, as an anomaly score otherwise, for general priors and posterior distributions, we can approximate it with a sample-based Monte Carlo estimate as follows:

$$D_{\text{KL}}(q_\phi(z | x) \parallel p(z)) \approx \frac{1}{M} \sum_{z \sim q_\phi(\cdot | x)}^M \left(\log q_\phi(z | x) - \log p(z) \right), \quad (3.3.10)$$

in which latent vectors z are sampled each time from the posterior distribution q_ϕ , and where the approximation quality improves as $M \rightarrow \infty$.

Dillon et al. [86] proposed to use a Dirichlet VAE (DVAE) [187] to learn a bi-modal, one-dimensional latent space that naturally encodes the two classes (signal and background), also achieving symmetric anomaly detection performance: since a VAE encodes structure in the latent space rather than compressing the inputs, it is expected to mitigate the complexity bias problem. The authors show that the Dirichlet prior on the latent space naturally leads to mode separation, something that was not observed for both the regular VAE [161] and the Gaussian-mixture VAE (GM-VAE) [188], without enforcing any additional loss term. To make training practical, the authors opt for a *softmax-Gaussian* approximation of the Dirichlet distribution to enable the reparameterization trick. Compared to the regular VAE and GM-VAE, with two effective latent dimensions the proposed DVAE reaches the highest class separation performance. Despite the promising results, the DVAE performance degrades as the imbalance between the signal and background increases. This implies that weak supervision is always required and that the Dirichlet prior should be carefully handpicked to achieve consistent results.

Bortolato et al. [189] propose to use the per-event Kullback-Leibler divergence between the learned and prior Gaussian distributions as an anomaly score to detect anomalous LHC jets. In their setting, in which $S/B = 0.1$, they measured a consistent improvement in significance discussing that imposing a cut on the KL divergence can be effective at pre-selecting events for a subsequent bump hunt.

Dillon et al. [190] compared the effectiveness of using low-dimensional latent space representations instead of the event space features of jets to perform model-agnostic anomaly detection. They trained a permutation-invariant transformer encoder [3] to optimize the JetCLR’s contrastive objective [191], where symmetry augmentations (e.g., rotation, distortion, collinear split, and shift) were employed to define positive and negative pairs for the

contrastive learning, which should preserve physical symmetries to mitigate the complexity bias: the contrastive loss optimizes the learned jet representation for *alignment* (i.e., map original and augmented jets to similar latent space regions) and *uniformity* (i.e., yield jet representations that make full use of the latent space.) Through a binary classification test, the authors discovered that a sufficiently large latent space (e.g., of size 512) is required to encode the physical symmetries of jets. Finally, the CWoLa [183] method was used to perform model-agnostic anomaly detection, showing that still a significant fraction of signal events is required to achieve meaningful class separation.

Govorkova et al. [192] demonstrate a real-world deployment of a simple, dense VAE on FPGA hardware for real-time AD at the LHC [42], targeting the L1 trigger. To achieve so, model *pruning* [193] and *quantization* [194] were employed using the `hls4ml` library [195, 196]. The authors compared the performance of both reconstruction- and KL-based anomaly scores, for both AE and VAE models. They concluded that with a minor loss in performance, the scores based on the KL divergence allowed them to deploy only the VAE’s encoder on the FPGA, thus saving both hardware resources and latency costs. For more information about model compression and acceleration, refer to subsection 3.4.5.

Recently, Cheng et al. [180] enhanced a (dense, Gaussian) VAE with the outlier exposure technique [133], which makes use of an auxiliary set of out-of-distribution (OOD) data (in their case, mass-rescaled boosted W jets) to improve the sensitivity to anomalies. An auxiliary loss term is computed from OOD predictions, whose analytical form (e.g., sigmoid or margin loss) depends on whether OE is performed on the input or latent space, to ensure a good compromise between high separation of anomalies and jet mass decorrelation (achieved through mass distribution matching) thus facilitating search approaches such as bump hunt; moreover, the injection of OOD samples allowed for symmetric anomaly detection. The authors also compared their approach with DisCo (distance correlation) [130] in terms of mass decorrelation performance, showing that their OE-VAE achieves consistent mass-decorrelation without the performance drop that occurs with DisCo regularization. Despite the promising results, it is not yet clear if data from the same physics domain is enough to be considered OOD.

3.4 Joint variational representations for anomaly detection

If new physics does exist at the scales investigated by the LHC, it is more elusive than expected. Therefore, finding interesting results may be challenging using conventional methods unless substantially increasing the (already large) number of analyses. Standard signal-driven search strategies could fail to achieve new results, and unsupervised ML techniques could close this gap by complementing them. Such methods, running in the trigger system of the LHC experiments, could spot anomalous events that would otherwise go unnoticed, enhancing the LHC’s scientific capabilities. In this regard, an ideal approach would have weak or no dependence on the signal being able to flag anomalies that may represent either new signals, particles, or processes, taking into account all the known physics so far. Moreover, such an approach should allow fast inference to perform online (or real-time) anomaly

detection.

For this reason, we studied the *Joint variational auto-encoders* (JointVAEs) [197] proposing them for anomaly detection on the trigger system. Regular AEs may suffer from the complexity bias [85], in which signals structurally simpler than the background are reconstructed even better than it because simpler features can be more easily compressed and, thus, learned. Conversely, the JointVAE can learn a richer representation by combining information from its multiple latent spaces, potentially mitigating the complexity bias. This fact is advantageous over a standard VAE, which can only leverage a single (often Gaussian) latent space.

3.4.1 The QCD vs Top benchmark

We demonstrate our JointVAE on the benchmark dataset of QCD vs Top jets [85], in which the considered task is to discriminate the QCD jets (the background) from the top jets, which represents the signal. To do so, we assume a normal-only setting for anomaly detection in which the JointVAE is trained solely on images of the QCD background. The jets are generated using `Pythia8` [45] with a center-of-mass energy of 14 TeV, without considering pile-up⁸ and multi-parton interactions. A fast detector simulation is then performed using `Delphes` [198]. Next, the jets are derived by running the anti- k_T algorithm [199] with a radius of $R = 0.8$. For each event, only the leading jet with $500 < p_T < 650$ GeV and $|\eta| < 2$ is considered: where p_T is the transverse momentum, and η is the pseudo-rapidity. Instead, the top jets are matched to a parton-level top within the jet radius, and all parton-level decay products are within the jet radius. Every jet is defined by its top 200 constituents according to the p_T , and the ones that have less of them are zero-padded.

Preprocessing. We employ the widely used [131, 86] preprocessing steps proposed by the dataset authors [85]. In particular, the preprocessing initially works at the constituent level by first centering the jets using the k_T -weighted centroid, such that the major principal axis points upwards. Next, the jets are rotated, so that the principal axis is vertical, and then mirrored along both axes such that most of the p_T is located in the first quadrant. After that, the jets are converted into two-dimensional jet images with 40×40 pixels corresponding to a two-dimensional transverse momentum histogram of the jet constituents, covering $\eta \in [-0.57, 0.57]$ and $\phi \in [-0.69, 0.69]$ (i.e., the azimuth) with pixel sizes of $[\delta\eta, \delta\phi] = [0.029, 0.035]$ in detector space. Finally, each jet image is normalized by dividing by its total p_T so that the pixel intensities now sum to one. The result of this preprocessing can be visualized in figure 3.8.

⁸“Pile-up” indicates the phenomenon in which unwanted particles yield by extra collisions overlap with the events of interest. These pile-up interactions should be removed to not interfere with the measurement.

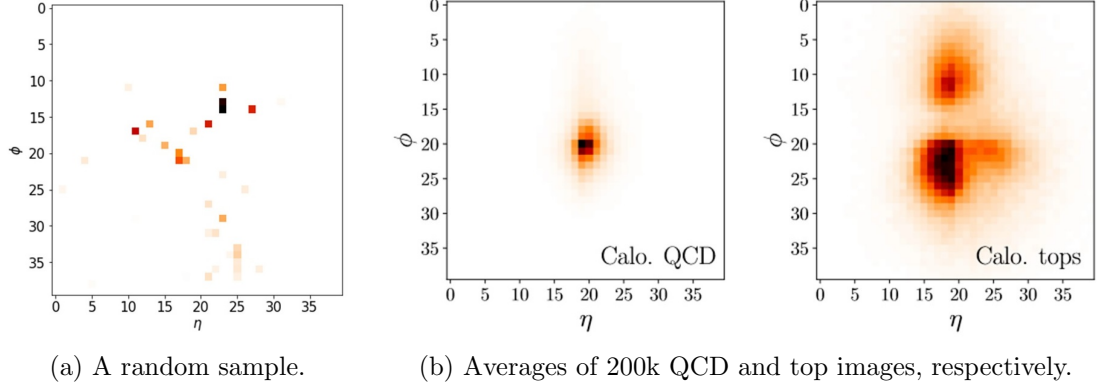


Figure 3.8: QCD vs top dataset: jet constituents are pre-processed to yield images.

3.4.2 Learn discrete representations

Since the JointVAE can learn both continuous (e.g., Gaussian) and *discrete* features in its latent space, we start by introducing a variation of the VAE framework that allows learning a latent space that is governed by a discrete distribution, like the Categorical⁹.

Categorical distribution. Let's say we want to model the repeated tossing of a K -sided die, with $K \geq 2$. The outcome of this experiment can be encoded in a K -dimensional random vector, $\mathbf{x} = (x_1, \dots, x_K)$, where x_j is the number of times side j of the die occurs. The probability of a given vector, $\mathbf{x}$, to occur is governed by a *multinomial* distribution [171]:

$$\text{Mu}(\mathbf{x} \mid n, \theta) \triangleq \frac{n!}{x_1! x_2! \dots x_K!} \prod_{j=1}^K \theta_j^{x_j}, \quad (3.4.1)$$

where θ_j is the probability of side j to occur, and $n \geq 1$ is the number of dice rolling (i.e., the number of repetitions). If $n = 1$, the dice is rolled only once obtaining a vector $\mathbf{x}$ of 0s and 1s, in which only one component can be one. In this case, we can think the vector $\mathbf{x} = [\mathbf{1}(x = 1), \dots, \mathbf{1}(x = K)]$ to represent the *one-hot encoding* of a scalar categorical random variable, x , with K values (or states). In this case, the probability mass function simplifies to [171]:

$$\text{Cat}(\mathbf{x} \mid \theta) \triangleq \text{Mu}(\mathbf{x} \mid 1, \theta) = \prod_{j=1}^K \theta_j^{\mathbf{1}(x_j=1)}. \quad (3.4.2)$$

This special case is called the categorical or discrete distribution, denoted as $\text{Cat}(\mathbf{x} \mid \theta)$. In the context of VAEs, learning a Categorical latent space allows modeling discrete variables like counts (e.g., the counting of objects like jets) and (pseudo) class labels, which cannot be captured with a Gaussian latent space that embeds, instead, continuous features.

⁹Also called the generalized Bernoulli or multinoulli distribution.

Gumbel-Softmax trick. Unfortunately, for the same reasons described in section 3.2.6, we cannot directly equip our VAE with a Categorical distribution without experiencing gradient explosions due to its non-differentiable behavior. To allow the model learn a Categorical latent space, we need to resort to an ad-hoc reparameterization trick called the *Gumbel-Softmax trick* [200, 201], which introduces a continuous relaxation over the Categorical (i.e., eq. 3.4.2) controlled by a *temperature* hyperparameter, τ , that is called the Gumbel-Softmax [201] or the Concrete [200] distribution. Considering c as a one-hot encoded categorical variable with class probabilities, $\alpha_1, \dots, \alpha_C$, we can sample from a continuous approximation of the Categorical by first sampling an i.i.d. set of $g_k \sim \text{Gumbel}(0, 1)$, and then applying a softmax transformation to them as follows [197]:

$$y_k = \frac{\exp((\log \alpha_k + g_k)/\tau)}{\sum_{i=1}^C \exp((\log \alpha_i + g_i)/\tau)} \quad \forall k = 1, \dots, C, \quad (3.4.3)$$

where each g_k is sampled from a Gumbel distribution¹⁰, and the vector $y = [y_1, \dots, y_C]$ is a continuous approximation of the one-hot representation of c . We denote the above process as $y \sim \text{Cat}(\alpha, \tau)$, in which the degree of relaxation is controlled by τ allowing us to interpolate (in the limit) between a Uniform ($\tau \rightarrow \infty$) and a Categorical ($\tau \rightarrow 0$) distribution as pictured in figure 3.9.

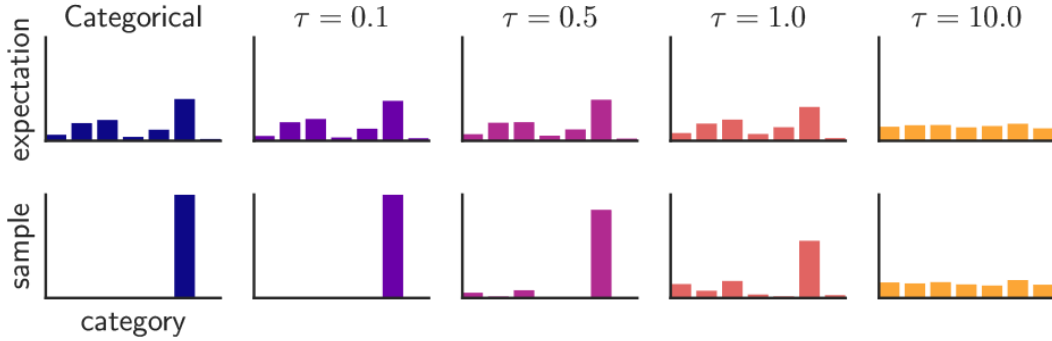


Figure 3.9: By tuning τ we can let the Gumbel-Softmax interpolate between a discrete one-hot categorical and a continuous categorical density. As the temperature increases the expected value converges to a uniform distribution over categories, α . Figure adapted from [201].

¹⁰The Gumbel noise is obtained by first sampling from a Uniform distribution, i.e., $u_k \sim U(0, 1)$, and then transforming the uniform samples by $g_k = -\log(-\log u_k)$. This is called inverse transform sampling [201].

Notice that the Gumbel-Softmax distribution is smooth for $\tau > 0$, having a well-defined gradient $\partial y / \partial \alpha$ w.r.t. the distribution parameters α . While Gumbel-Softmax samples are differentiable, they are not identical to samples from the corresponding Categorical distribution for non-zero temperature. During learning, there is a trade-off between small temperatures, where samples are close to one-hot vectors but the variance of the gradients is large, and large temperatures, where samples are smooth but the variance of the gradients is small. Therefore, the temperature can be tuned to account for such variance, e.g., by either following an annealing schedule or learning it along the distribution’s parameters. The former case can be interpreted as *entropy regularization* [201], where the Gumbel-Softmax distribution can adaptively adjust the “confidence” of proposed samples as training proceeds.

Categorical VAE. The last modifications to the Gaussian VAE are the following. The model now has to learn the parameter of a Gumbel-Softmax distribution, therefore the encoder network should predict class probabilities, i.e., $q_\phi(c \mid \mathbf{x}^{(i)}) = \text{Cat}(c \mid \alpha^{(i)}, \tau)$ where $\alpha^{(i)} = q_\phi(\mathbf{x}^{(i)})$, instead of the means and variance of a diagonal Gaussian. Moreover, to provide extra modeling power, e.g., enabling the learning of diverse discrete features, it is possible to approximate more than one Categorical distribution by letting the encoder parameterize the product of multiple independent Gumbel-Softmax distributions. Once the form of the distribution that the encoder should parameterize is established, we need to define a suitable latent prior, $p(c)$, which can be a (product of) uniform Gumbel-Softmax distribution(s) in which each class probability is uniform, i.e., $\alpha_k = 1/C$. At this point, we can reparameterize the latent variables, c , minimizing the loss in eq. 3.2.7 in which the KL-divergence term can be calculated as follows:

$$D_{\text{KL}}(\text{Cat}(\alpha, \tau) \parallel \text{Cat}(1/C)) = \sum_{j=1}^C \left(\alpha_j \log \alpha_j - \alpha_j \log(1/C) \right), \quad (3.4.4)$$

where $p(c) = \text{Cat}(1/C)$ is the uniform categorical prior, and C the number of categorical classes.

3.4.3 Joint variational auto-encoders

Building on the previous section, a joint VAE [197] modifies the original VAE framework by letting the encoder network parameterize both continuous (e.g., Gaussian) and discrete (e.g., relaxed categorical) distributions. In our case, the encoder outputs the parameters of a diagonal Gaussian, i.e., μ and σ^2 , along with the parameters of a single Gumbel-Softmax, i.e., α , that approximates a Categorical distribution according to the chosen temperature, τ . We can then write: $\mu^{(i)}, (\sigma^{(i)})^2, \alpha^{(i)} = q_\phi(\mathbf{x}^{(i)})$. Next, we sample the two latent vectors, $\mathbf{z} \sim \mathcal{N}(\mu^{(i)}, (\sigma^{(i)})^2 I)$ and $\mathbf{c} \sim \text{Cat}(\alpha^{(i)}, \tau)$, from the respective distributions through a reparameterized sampling, to successively concatenate them into one latent vector which is passed as input to the decoder network, p_θ , for reconstructing the input sample, $\mathbf{x}^{(i)}$.

By embedding the input to both continuous and discrete latent spaces, the joint VAE can learn a richer variety of features compared to a standard VAE: the Gaussian latent space can encode *continuous features* like angles, widths, color, energies, coordinates and so on; instead, the Categorical latent space can capture *discrete variables* like types and classes, style information, and counts (of objects) to name a few. Furthermore, as both kinds of information are provided to the decoder, which is complementary in meaning, these can enable the network to potentially reconstruct (and generate) the input data sample better.

In addition, since the factors of variations describing the data-generating distribution, which are unknown, are (by construction) factored into two distributions, this design can enable the model to recover factors that are better disentangled, resulting in improved interpretability of the single latent components and, also, enhanced editing and manipulation of the latent vectors.

Learning objective. Since two latent spaces are learned, we have to modify the KL term in the loss function (eq. 3.2.7) accordingly. In particular, the continuous and discrete latent variables are assumed to be *conditionally independent* (this is both reasonable and wanted, as we want to capture disentangled factors of variations), i.e., $q_\phi(\mathbf{z}, \mathbf{c} | \mathbf{x}) = q_\phi(\mathbf{z} | \mathbf{x})q_\phi(\mathbf{c} | \mathbf{x})$, and similarly for the joint prior, i.e., $p(\mathbf{z}, \mathbf{c}) = p(\mathbf{z})p(\mathbf{c})$. We can then rewrite the joint KL

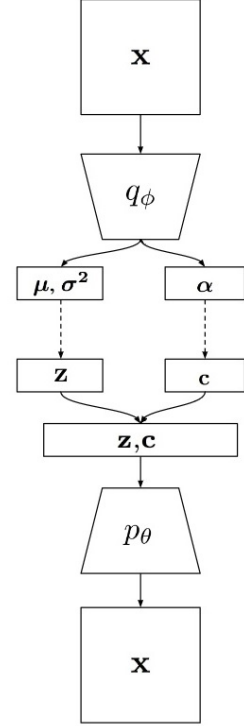


Figure 3.10: JointVAE architecture. Figure adapted from [197].

divergence as follows:

$$\begin{aligned}
 D_{\text{KL}}(q_\phi(z, c | x) \parallel p(z, c)) &= \mathbb{E}_{z, c \sim q_\phi(\cdot | x)} \left[\log \frac{q_\phi(z, c | x)}{p(z, c)} \right] \\
 &= \mathbb{E}_{z \sim q_\phi(\cdot | x)} \mathbb{E}_{c \sim q_\phi(\cdot | x)} \left[\log \frac{q_\phi(z | x) q_\phi(c | x)}{p(z) p(c)} \right] \\
 &= \mathbb{E}_{z \sim q_\phi(\cdot | x)} \left[\log \frac{q_\phi(z | x)}{p(z)} \right] + \mathbb{E}_{c \sim q_\phi(\cdot | x)} \left[\log \frac{q_\phi(c | x)}{p(c)} \right] \\
 &= D_{\text{KL}}(q_\phi(z | x) \parallel p(z)) + D_{\text{KL}}(q_\phi(c | x) \parallel p(c)),
 \end{aligned} \tag{3.4.5}$$

which results in the decomposition of the individual contributions of the KL divergences of the respective distributions. This entails that the total loss function is now

$$\mathcal{L}_{\text{JVAE}} = \underbrace{-\mathbb{E}_{z, c \sim q_\phi(\cdot | x)} [\log p_\theta(x | z, c)]}_{\text{reconstruction loss}} + \underbrace{\beta D_{\text{KL}}(q_\phi(z | x) \parallel p(z))}_{\text{continuous KL}} + \underbrace{\beta D_{\text{KL}}(q_\phi(c | x) \parallel p(c))}_{\text{discrete KL}}. \tag{3.4.6}$$

3.4.4 The JointVAE architecture

The proposed JointVAE [202] implements both the encoder and decoder as residual networks. In particular, each encoder’s residual block down-samples the intermediate feature map by a factor of two, whereas the decoder’s residual block doubles the intermediate representation. Both kinds of blocks have two 3×3 grouped convolutions (initialized by following [22]) in between the skip connections, whose individual output is first subject to instance normalization (IN) [203] to which is then applied the leaky ReLU activation [204]: both operations are explained in the next paragraphs. Specifically, the encoder comprises four residual blocks in which the convolutions in each block apply 32, 64, 128, and 128 filters respectively, resulting in about 330k parameters. Lastly, the encoder outputs the parameters of the latent space distributions: a 20d-vector to parameterize the Gumbel-Softmax, and two 32d-vectors being the mean and variance of a diagonal Gaussian distribution.

Similarly, the decoder includes four residual blocks applying 32, 128, 128, and 64 filters each. Moreover, we apply an upsampling layer in each block to avoid the reconstructed samples to show the *checkerboard artifacts*, caused by transposed convolutions [205]. In particular, a $2 \times$ bilinear upsampling operation doubles the spatial resolution of the current feature map, which is then smoothed by a convolution (with the “same” padding rule) normalized by IN. To contain the growth of parameters, the decoder uses eight groups for convolutions instead of four, resulting in about 550k learnable weights. The last peculiarity of the decoder network is the usage of the spatial broadcast operation [206]. In particular, after the encoder prediction, continuous and discrete latent vectors are concatenated and replicated along each spatial location of a 3×3 square. The resulting tensor of size $3 \times 3 \times 54$ is then fed to the decoder’s residual blocks to yield the reconstructed samples.

In the following paragraphs, further details of the JointVAE model are discussed.

Instance normalization. IN was initially proposed for generative tasks like style transfer [207], in which the content of a reference image provides the “artistic style” to be applied to an input image, as an alternative to batch normalization [208] but later discovered to be beneficial also for auto-encoders [159]. Considering the case of applying IN on convolutional layers, i.e., on a $H \times W \times C$ input sample, $\mathbf{x}$, the normalization happens along the channel dimension (figure 3.11a) as follows:

$$\bar{\mathbf{x}} = \gamma \frac{\mathbf{x} - \mathbb{E}[\mathbf{x}]}{\sqrt{\mathbb{V}[\mathbf{x}] + \epsilon}} + \beta, \quad (3.4.7)$$

where γ and β are learnable vectors of size C , ϵ a small positive constant that prevents the square root at the denominator to be zero, and $\mathbb{E}[\mathbf{x}]$ and $\mathbb{V}[\mathbf{x}]$ compute, respectively, the mean and variance of the input but only along its spatial dimensions, i.e., by considering only the height and width axes. To normalize the channels of $\mathbf{x}$, the mean and variance are $1 \times 1 \times C$ tensors required to be broadcasted during subtraction and division to match the shape of $\mathbf{x}$.

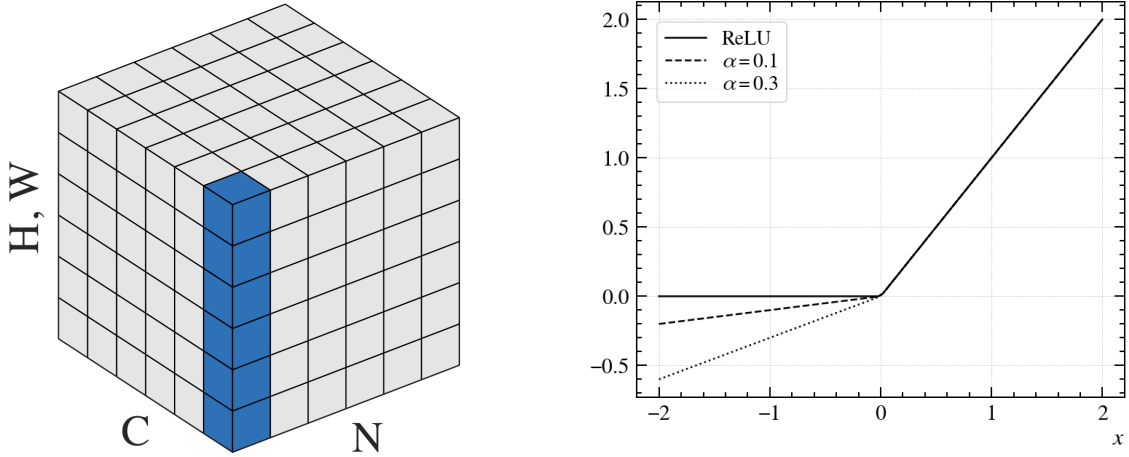


Figure 3.11: (a) Instance normalization: N indicates instances in a mini-batch. (b) The ReLU activation with and without the negative slope.

Leaky ReLU. The ReLU activation function is well known to suffer from the *dying neurons* problem [204, 209], in which negative values (being the result of the product of the learned weight by the input feature) provided as input to the activation became zero (by design). If so, the gradient of this operation is zero too, effectively halting learning. To fix this issue, we can introduce a *negative slope* hyperparameter, α , that accounts for a small contribution when a negative value is subject to the activation (as shown in figure 3.11b),

yielding a non-zero gradient. Hence, the activation is now

$$\text{LeakyReLU}(x, \alpha) = \underbrace{\max(0, x)}_{\text{ReLU}} + \underbrace{\alpha \min(0, x)}_{\text{negative slope}}. \quad (3.4.8)$$

Common values for α are 0.1, 0.2, or even 0.3.

Spatial broadcast. Motivated by the learning of disentangled latent factors, we can provide a generic *spatial bias* to the decoder by associating the latent vectors, which are yielded by concatenating categorical and continuous latent spaces, to Cartesian (x, y) coordinates¹¹. According to the outcomes of [206], the resulting *spatial decoder* better disentangles the latent factors (therefore resulting in improved control and interpretation of the latent space), also achieving more robust reconstructions. Considering a latent vector, z , and a set of coordinates $X = \{-1, \dots, 1\}$ and $Y = \{-1, \dots, 1\}$, of size w and h respectively. The spatial broadcast operation first replicates the latent vectors such that to create the width and height spatial dimensions (yielding a volume, say b , of size $h \times w \times |z|$), to then concatenate the X and Y coordinates (but arranged in a grid-like manner such that to match b) along the last axis of b , yielding the broadcasted latents, z_b , of size $h \times w \times (|z| + 2)$. Hence, at each pixel location of z_b , we have a vector corresponding to the latent vector z but concatenated with the xy coordinates of the corresponding location. Such operation provides the decoder with spatial (e.g., coordinate) information allowing it to relate the latent vectors in space when reconstructing the input.

3.4.5 FPGA implementation feasibility study with HLS4ML

Deep neural networks are usually oversized compared to what is necessary to solve a problem with adequate accuracy. The general recipe is to train a large enough model (i.e., a network with more capacity than needed) first and then regularize it until sufficient generalization is attained. Through regularization, we effectively control and shrink only the complexity of the model but the number of learnable parameters, multiplications and accumulations (MAC, in short), the memory footprint, and the FLOPS (floating point operations per second) are left untouched. The problem is that finding a network architecture that is both fast and accurate is hard, requiring solving a joint problem between hyperparameter tuning and architecture search [210].

Compression and acceleration. Model compression [211] is a branch of DL specialized in making already trained NNs more computationally efficient without scarifying performance, though various techniques like *compact model*, *tensor decomposition*, *data quantization* and *network sparsification*.

¹¹In the context of HEP, an interesting attempt would be to associate polar or detector coordinates, e.g., (η, ϕ) , to pixels. Thus, introducing a knowledge bias about the detector geometry.

- **Compact model.** This line of work aims to design small and accurate models, mainly focusing on CNNs. For example, stacking two 3×3 convolutions can approximate a 5×5 one but with 28% fewer parameters and operations. More drastically, a $n \times n$ convolution can be factorized into a $n \times 1$ convolution followed by a $1 \times n$ one, reducing the computational cost from $O(n^2)$ to $O(2n)$ [61]. Furthermore, expensive convolutions can be replaced with group convolutions to increase efficiency or when the training is distributed on multiple devices (e.g., AlexNet [134]). A side effect of stacking multiple group convolutions is that the output from a certain channel is obtained only from a certain group, introducing a regularization effect that lowers the network power. In ShuffleNet [152] this is mitigated by introducing the *channel shuffle* operation that rearranges the feature map's output channels. In addition, depth-wise convolutions are a special case in which the number of groups equals the number of channels. However, these do not aggregate the input channels to form the new ones. To avoid that, a point-wise convolution is applied on top of the depth-wise one, combining information across the individual channels. This pair of layers is called a *depth-wise separable convolution* [212], used in many lightweight architectures like the Mobile-Net v2 [213]. Moreover, neural architecture search (NAS) [214] can either optimize an existing architecture or find a new one from a pre-defined set of blocks, also taking into consideration the underlying hardware and resource constraints [215]. Lastly, knowledge distillation [65] can be regarded as a model compression technique that distills (i.e., compresses) the knowledge of either a large model or ensemble of models, called the teacher, to a smaller, more efficient model: the student. KD works by forcing the student to mimic the *softened* output of the teacher, being often limited to models that output softmax probabilities.
- **Tensor decomposition.** Decomposition methods divide large parameter tensors into smaller matrices or tensors to reduce the memory requirement and operation number. The basic idea is to leverage *matrix decomposition* to represent a full-rank (weight) matrix with lower-rank matrices, which are smaller to store and way faster to compute, reducing the inference time. Singular value decomposition (SVD) [216] can achieve good performance with a low loss in accuracy, and can even be used to compress the input data speeding up the network training. Despite the good performance, SVD is quite expensive to compute. Moreover, low-rank decomposition works best on fully connected layers which are more redundant than convolutional filters. In addition, the decomposition should be performed layer by layer, and often, retraining to achieve convergence is necessary [217].
- **Quantization [194].** Often, representing the weights, activations, gradients, or model updates with full precision is more than sufficient. Quantization reduces the number of bits (i.e., the *bitwidth*) to represent one or more of these numerical quantities, attaining model compression and acceleration. Discretization levels can be uniform or non-uniform. Uniform levels imply an even step length between levels, whereas non-uniform (e.g., logarithmic) levels have variable step lengths that can dy-

namically allocate more bits to important data regions (achieving a higher dynamic range) or low bits to less relevant data values. For example, quantizing the weights, which are often represented as single or double-precision floating point numbers (denoted as `fp32` and `fp64`), to fixed-precision integers (like the `int8` and `int4`, which respectively occupy eight or four bits instead of thirty-two or sixty-four) allows to: 1) greatly reduce the space required to store the model; 2) diminish the network bandwidth to send the model updates in a distributed setting; 3) decrease the download size of the model for over the air updates; 4) speed-up inference because integer arithmetic is way faster than floating-point calculations, also allowing to execute large models on commodity hardware; and 5) reduce the electricity consumed by the system. Usually, quantization is applied to the weights and, or activations. In particular, we quantize the hidden layers of a model and not its input or output ones, as the model accuracy can drastically worsen. In addition, quantization can be performed after or during training. If we quantize during training, *mixed precision* is often used to account for the fact that the model updates are quite sensitive to quantization, requiring higher numerical precision than the weights or activations.

- **Sparsification.** Layers with many parameters often contain redundant weights or neurons. Sparsification or *pruning* [193] aims to remove these redundant connections to yield lighter models, resulting in fewer memory accesses and computational operations. Compared to quantization, pruning cannot simplify the arithmetic of the operations. Sparsification can reduce the model size since many weights and neurons can be zeroed. However, unless the underlying hardware allows for some form of *sparse computation*, the additional index overhead for addressing the nonzero element can even slow down execution. For example, FPGAs can reconfigure themselves to account for the pruned model therefore not allocating resources for the zero elements. Weight pruning can be magnitude-aware by zeroing absolute weight values below a given threshold, or we can remove the top-% smallest weights. Neuron pruning, instead, can follow the magnitude rule or neurons can even be removed on a random or probabilistic base. Usually, the lost performance can be recovered by subsequent fine-tuning after pruning. In addition, iterative pruning with retraining allows us to attain higher network sparsity at the same performance level. Generally, pruning in dense layers can reach even the 90-95% while the sparsity in convolutional filters, since less redundant, is more limited at most about 40-50%. Furthermore, modern hardware can benefit from pruning if performed on a structural rather than element-wise level.

The compression techniques described above can be combined, obtaining what it is called *joint-way compression* [211], to achieve an even higher compression ratio. For example, the design of compact models can be paired with quantization or pruning. Moreover, the tensors resulting from low-rank decomposition can benefit from further compression with sparsification. Lastly, combining quantization with sparsity is often a popular approach.

Compressing a model can either enable its deployment on less capable hardware (e.g., mobile and edge devices, single-board computers), improve inference performance (i.e., pro-

cessing more samples per unit of time), or even save resources (e.g., memory and storage) and costs (e.g., energy bills, cloud computing subscriptions) on existing computing systems. Yet, inference performance can be further boosted by employing specialized or optimized hardware, like FPGAs and ASICs (application-specific integrated circuits). Compared to general-purpose CPUs and GPUs, FPGAs allow for a re-programmable logic whereas this is fixed (i.e., it cannot be modified unless a new circuit is designed and implemented) for ASICs, which are faster but extremely costly.

FPGAs. Field Programmable Gate Arrays [218] are specialized computational devices capable of implementing circuits with an efficiency comparable to the hardware. This is advantageous for power consumption, size, and performance compared to software and general-purpose hardware implementations, making FPGAs particularly suitable for real-time data processing. Furthermore, FPGAs can be conveniently reconfigured for various tasks (e.g., image and signal processing) without incurring significant cost: the expense of designing a new silicon is completely avoided, unlike ASICs. Designing an FPGA-based circuit involves configuring the memory bits that determine routing decisions properly. This is accomplished by generating a bitstream that is loaded onto the device. FPGAs consist of replicated units of digital electronic circuits known as *logic blocks*, embedded in a general routing structure. These logic blocks contain processing elements that carry out simple combinational logic and flip-flops for implementing sequential logic. The general routing structure permits arbitrary wiring, allowing the logical elements to be connected as desired. Modern FPGAs can compute functions on the scale of millions of basic gates, operating at clock frequencies in the hundreds of Megahertz. More powerful FPGAs have additional ad-hoc elements embedded into the array, such as large memories, multipliers, and microprocessors, to further boost speed and capacity. With these predefined, fixed-logic units, FPGAs can implement complete systems in a single programmable device by adjusting the interconnections between these components to perform complex algorithms. From a practical perspective, FPGAs are between computer CPUs and custom chips (e.g., ASICs) being more complicated to program than a CPU but faster and way more flexible than an ASIC.

The main components to care about (in terms of utilized resources) when designing a processing workflow to be deployed on an FPGA are the following:

- *Configurable logic blocks* (CLBs): these are the fundamental computational units, which can be programmed to perform logic functions. A single CLB includes look-up tables (LUTs) and flip-flops (FFs). Logical circuits are implemented by connecting many LUTs, whose results may be stored in flip-flops.
- *Digital signal processing* (DSP) *slices*: these are optimized circuits that perform specialized signal processing operations like filtering or multiplying.
- *Block random access memory* (BRAM): This is the dedicated memory of the device; the CLBs can be further divided into virtual blocks, each of which is assigned a given amount of BRAM.

FPGAs are used in the context of HEP (e.g., [192]) to process the large amounts of data generated by the detectors and increase the speed of the data processing, filtering, and compression, reducing the number of bits that needs to be transferred over the network and stored on disks.

HLS4ML [195, 219]. Traditionally, FPGAs are designed using hardware description languages (HDLs) such as VHDL or Verilog, requiring profound expertise in circuit design and machine learning since the mathematical operations underlying deep networks need to be translated into a series of low-level hardware-specific commands. However, recent synthesis tools enable a much easier workflow through a *behavioral description* of the DL model, which can be written in C++ or Python, for a rapid deployment on the FPGA. Tools like the HLS4ML first encode the various neural network layers and custom operations with a high-level synthesis (HLT) language, then compile and optimize it yielding C++ code. Next, this code is embedded into the hardware through a vendor-specific compilation phase that strictly depends on the specifics of the FPGA manufacturer. This is a common synthesis workflow, in which it is advisable to use model compression techniques (as described in the previous section) to optimize a pre-trained deep learning model before the actual deployment on the FPGA accelerator. In particular, weight pruning eliminates unimportant neural network connections enabling the accelerator to allocate fewer resources, in terms of LUTs and DSPs, to implement the model. In addition, quantization can help reduce the consumption of BRAM while increasing the inference speed to process a sample, thanks to the smaller numerical precision.

Feasibility study. Before the FPGA deployment, we first train our JointVAE model and then quantize its layers (using QKeras¹²) to both shrink its size and lower the memory consumption. At this point the HLS4ML tool can be used to compile the pre-trained model: in particular, we only target the encoder network for synthesis, thus discarding the more costly decoder, following a latent-based anomaly detection paradigm (section 3.3.2) which is particularly suitable to perform inference on FPGA devices thanks to the intrinsic reduced cost (usually more than halved) by only employing the encoder network.

Unfortunately, even if we quantize and reduce the number of layers, the initial convolutional layers that process the full image resolution of 40×40 pixels remain too large to fit on the FPGA. In this case, we cannot deploy the model on the accelerator because its resources (e.g., LUTs, DSPs, etc) are exhausted. However, the layers that act on feature maps with lower spatial resolution can be compiled on the device, therefore it is possible to exploit this fact to estimate the amount of resources that a given convolutional layer would require to be synthesized on the device. Starting from a layer for which its resource consumption is known, say b , the cost of a new layer, a , can be estimated in terms of the

¹²<https://github.com/google/qkeras>

resource factor [202], ρ_b^a , as follows:

$$\rho_b^a = \frac{(h_{\text{in}}^a/s^a) \cdot (w_{\text{in}}^a/s^a) \cdot d_{\text{in}}^a \cdot d_{\text{out}}^a}{(h_{\text{in}}^b/s^b) \cdot (w_{\text{in}}^b/s^b) \cdot d_{\text{in}}^b \cdot d_{\text{out}}^b}, \quad (3.4.9)$$

where $h_{\text{in}}, w_{\text{in}}, d_{\text{in}}$, and d_{out} are, respectively, the height, width, and depth of the input feature map, plus the output depth (i.e., equal to the number of convolutional filters) of the convolution operation: if the strides of the two layers are equal, i.e., $s^a = s^b$, then they cancel out in the equation. The formula is derived by considering the computational complexity of the convolution operation [213]: i.e., $O(k^2(hw)/s \cdot d_{\text{in}} \cdot d_{\text{out}})$ where k is the kernel size. Thus, taking the ratio between the complexity of two convolutions it is possible to understand which layer is more expensive. If $\rho_b^a > 1$, the new layer, a , is more expensive than layer b ; otherwise, it is cheaper to implement. For example, say we want to estimate the number of LUTs for layer a provided we already know the number of lookup tables for convolutional layer b . To do so, we compute the resource factor by following eq. 3.4.9, multiplying the already known value of LUTs by it to get the estimated LUTs necessary to synthesize layer, a .

Layer name	$h_{\text{in}}, w_{\text{in}}, d_{\text{in}}, d_{\text{out}}$	DSPs (%)	LUTs (%)	FFs (%)	BRAM (%)
conv2_b0_0*	20, 20, 16, 16	120 (1)	990351 (57)	36725 (1)	46 (2)
conv2_b1_1	10, 10, 20, 20	47 (~ 0)	386856 (22)	14346 (~ 0)	18 (1)
dconv_b2	5, 5, 20, 4	15 (~ 0)	309544 (17)	9352 (~ 0)	4 (~ 0)
final block	5, 5, 4, 2	851 (7)	91763 (5)	28074 (~ 1)	377 (19)

Table 3.1: Resource estimation for some layers of the quantized JointVAE encoder, computed by Vivado HLS 2020.1 for a target clock period of 5ns on the Alveo U250 [220, 221] FPGA¹³: remaining layers are omitted since having the same consumption of the listed ones. The cost of the layer denoted with an asterisk (i.e., conv2_b0_0) has been estimated by multiplying the resource factor, $\rho_b^a = 2.56$, by the entry corresponding to layer conv2_b1_1.

Utilizing the resource factor, we can estimate the costs associated with convolutional layers that are too large for the FPGA, as shown in table 3.1. Indeed, the resource estimation is approximated: if the estimated quantities follow a linear relationship w.r.t. the reference layer then the approximation is accurate. Nevertheless, this can be a useful means to guide the choice of architecture-related hyperparameters towards a working FPGA implementation. Moreover, inspecting the number of DSPs, LUTs, FFs, and BRAM required by each layer or block can provide insights about potential bottlenecks and waste of resources, suggesting which layers should be further compressed.

¹³The Alveo U250 board has 1.728K LUTs, 3.456K FFs, 12288 DSPs, and 2000 36Kb block RAMs.

3.4.6 Discussion and conclusions

As anticipated in the previous sections, we perform latent-based anomaly detection as a cheaper alternative to reconstruction-based anomaly scores. Moreover, since we employ a joint VAE model, which learns two latent spaces (a continuous and discrete one), it is possible to devise a new anomaly score by combining both discrete (eq. 3.4.4) and continuous KL (eq. 3.2.11) divergences: we call it the **KL total** score. The result of our evaluation is summarized in table 3.2.

Model	AD score	TPR@FPR 10^{-1} [↑]	AUROC [↑]
Complete encoder	KL discrete	52.74%	73.63%
	KL continuous	58.10%	86.46%
	KL total	58.34%	86.63%
Reduced encoder	KL discrete	31.46%	68.29%
	KL continuous	63.48%	84.52%
	KL total	63.63%	84.64%
Quantized encoder	KL discrete	51.67%	77.38%
	KL continuous	59.79%	80.88%
	KL total	60.03%	81.17%

Table 3.2: Anomaly detection performance comparison among the *complete* (i.e., full-precision), *reduced* (i.e., with fewer filters and layers), and *quantized* (i.e., fixed-precision reduced model) encoders. The symbol $\uparrow$ denotes that higher values are better.

As expected, the complete model with full floating-point precision outperforms the other two. Although the reduced encoder is much smaller¹⁴ compared to the full model, the drop in performance is limited regarding the AUROC, which summarizes the whole ROC curve. Interestingly, the reduced model achieves a higher true positive rate at the given error rate (i.e., the TPR@FPR, considering a type I error rate of 10%) meaning that the shapes of the ROC curve favor lower classification threshold values than the complete model. Lastly, the quantized model¹⁵ achieves the highest performance regarding the discrete KL meaning that fixed-precision quantization may help the learning of discrete features. More importantly, in all cases combining the two KL divergences yields an improvement in anomaly

¹⁴The reason for this is to avoid a constraint on the per-layer number of parameters introduced by Vivado HLS, which prevents synthesis on the FPGA.

¹⁵Quantization is applied during training to the weights and activations of both convolutional and dense layers. We impose a fixed precision of 16 bit, six of which are for the integer part, and the remaining ten bits for the floating part.

detection performance compared to the single scores, confirming the fact that: continuous and discrete features can complement themselves, learning different aspects of the input sample which, in turn, improves the sensitivity towards the anomalies.

In conclusion, the size, memory requirements, and latency of deep learning models can be effectively optimized by model compression techniques. By using hardware accelerators, like the flexible FPGAs, it is possible to increase the neural network inference speed to comply with strict latency requirements, like the ones found at the L1 trigger. In practice, achieving an accurate, optimized, and fast model on the accelerator can be quite hard: compression usually decreases performance, which in some scenarios is already hard to get satisfactory predictions, and acceleration often requires extra effort and proper hardware and software support. Accelerating a model is not trivial because the synthesis software can introduce additional constraints (e.g., on the number of parameters, and kinds of layers that can be compiled) apart from the ones that are intrinsic to the hardware. This introduces a feedback loop that forces an iterative design and subsequent training, making it difficult to deploy a pre-trained model. However, even if employing large and powerful FPGAs like the Alveo U250, deploying convolutional models is still quite difficult since the high dimensionality of the input (e.g., even a $40 \times 40 \times 1$ image can be considered too large) incur a high cost (in terms of resource consumption on the accelerator device) at the initial layers or blocks, with consequent impossibility to synthesize them on the device. These issues could be mitigated or solved by introducing compression techniques to reach higher compression ratios and improved software support for FPGA deployment.

3.5 Conditional dual auto-encoders for dark force search

This application focuses on two important and highly challenging manifestations of Hidden Valley (HV) models [222], more specifically, of dark quantum chromodynamics (QCD), namely Soft Unclustered Energy Patterns (SUEPs) [223] and Semi-Visible Jets (SVJs) [224, 225, 226]. We propose a data-driven model-independent search strategy powered by Conditional Dual (Variational) Auto-Encoders (CoDAEs) and normal-only anomaly detection [85], demonstrating generalization over both signals despite not being trained on them. The proposed models can learn from highly sparse, unprocessed images of particle collisions, and their design allows for easy acceleration enabling us to meet the latency of the high-level trigger¹⁶. Moreover, we make minimal assumptions on the signal aiming to implement a new physics search with a high degree of model independence. Thus, we only assume: 1) access to not anomalous data samples represented by the QCD background; and 2) that the two signals can be revealed through tracking information.

¹⁶The CMS trigger system is a two-tiered event selection system. The electronics-based first level (L1) uses information from the calorimeters and muon detectors, reducing the event rate from 40 MHz to around 100 kHz within a time interval of 4 μ s. The second level, the High-Level Trigger (HLT), runs a version of the full event reconstruction software optimized for fast processing on a farm of processors. The HLT reduces the event rate to about 1 kHz within $O(10^2)$ ms, and the selected events are transferred to storage.

3.5.1 Searching for Hidden Valley models

The hidden valley models are a type of BSM physics describing the existence of a new sector of particles and forces. The HV models have been mainly developed to address the origin of dark matter [227], whose experimental signatures often feature non-isolated objects with high-multiplicity or low-energy final states, representing a challenging target for existing analyses at the LHC [228].

The hidden valley models can produce dark quarks in proton-proton collisions at the LHC, leading to a dark shower¹⁷ and the production of a large number of dark hadrons, analogous to QCD jets [222, 223]. Depending on the details of the theory, the dark showers can follow large-angle emission and dark hadrons do not form narrow QCD-like jets. The decay of dark hadrons results in dark photons, which further decay to low-energy SM particles with transverse energy, E_T , of $\mathcal{O}(10^2)$ MeV, whose final experimental signature being high-multiplicity spherically-symmetric soft unclustered energy patterns [223]. Through their decay to SM particles, these processes become visible and in principle detectable in detectors at the LHC such as ATLAS [229] and CMS [43]. We focus on a well-motivated scenario where SUEP is produced via gluon fusion and all dark hadrons decay promptly and exclusively to pions and leptons, an experimental nightmare scenario because of an overwhelming multi-jet QCD background.

Another manifestation of hidden valleys can be semi-visible jets [226, 224], a phenomenon in which energetic particles are emitted in a spray of stable invisible dark matter. These showers are partially detectable, with the visible components looking like QCD showers [224]. This partial visibility makes it challenging to identify and study these particles thoroughly, having a low acceptance rate with current methods.

Related BSM analyses. The studies conducted by Barron et al. [230] and Canelli et al. [231] are related to our search scenario, since a similar signal setting is assumed. In particular, Barron et al. [230] target the same SUEPs scenario in which the signal decays to exotic Higgs, and all the dark hadrons to SM hadrons. The authors identify three observables: charged particle multiplicity, event isotropy, and interparticle distance. These are used to build the input features for their unsupervised fully connected auto-encoder. Canelli et al. [231], instead, study the SVJs signature by training a fully connected auto-encoder on jet-level and jet substructure variables, minimizing the mean absolute error. Compared to these two studies, we neither rely on high-level (e.g., substructure information) nor engineered particle- or jet-based features but, instead, learn from raw detector images. Moreover, our models are evaluated against both signals, demonstrating anomaly scores that can identify both.

¹⁷A “shower” is a cascade effect of secondary particles (e.g., electrons, photons) initiated by the interaction of a high-energy particle with matter.

3.5.2 The CMS detector

The CMS experiment [43] is a general-purpose detector at the LHC, developed to explore the physics of proton-proton collisions through a system of different sub-detectors, each designed to measure various aspects of the particles produced in a collision. Given its *cylindrical* design, as we can see in figure 3.12, it is often convenient to adopt a polar coordinate system (θ, ϕ) where: $0 \leq \theta \leq \pi$ is the polar angle, and $0 \leq \phi \leq 2\pi$ is the azimuthal angle. From these coordinates, it is possible to explain the particle's kinematic as (p_T, y, ϕ, m) : where m is the invariant mass, p_T the transverse momentum, and y the rapidity. A quantity related to the rapidity is the *pseudo-rapidity* η , which measures the angle of the particle's motion relative to the beam line. The generated events that support our search for hidden valley models are represented as images in the η - ϕ plane, therefore considering the amount of energy that distributes along the pseudo-rapidity and azimuth of the particle's kinematic.

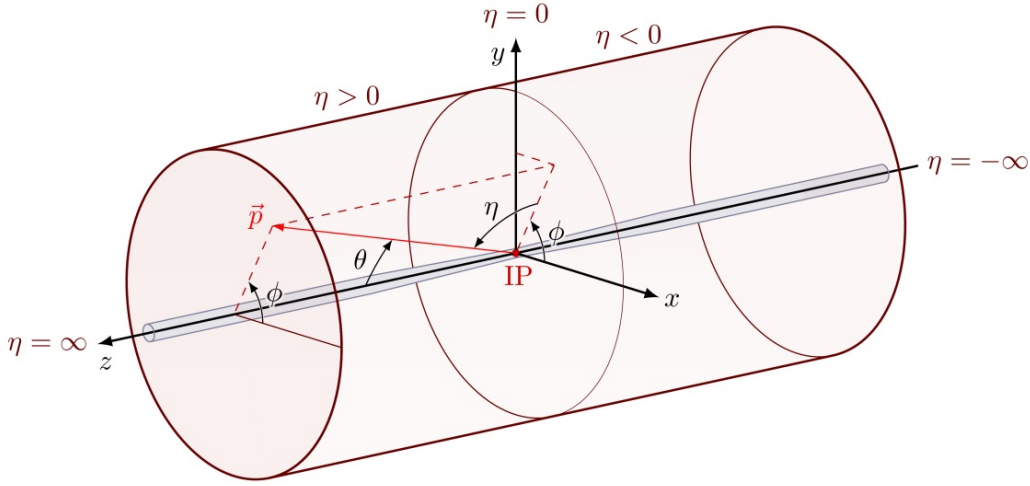


Figure 3.12: The CMS coordinate system, which explains the particle's motion within the cylindrical detector. Figure generated from <https://tikz.net/axis3d.cms/>.

The CMS detector consists of several layers that measure various properties of particles produced in high-energy collisions. The ones [232] relevant to our work are the:

- **Inner tracking system** [233], which measures the momentum of particles by their curvature radius through the magnetic field. The tracker can monitor the paths of charged particles (e.g., muons, electrons, and hadrons) by measuring their positions at several points, and it is built in a way to disturb the particle as little as possible. This sub-detector is the innermost part of the CMS detector, receiving the highest volume of particles: it can cover a pseudo-rapidity region of up to $|\eta| < 2.5$, being made of 66M silicon pixel detectors ($100 \times 150 \mu m^2$ in size) for accurate measurement of the particle's trajectory.

- **Calorimeters**, consisting of an Electromagnetic calorimeter (ECAL), and a Hadron calorimeter (HCAL) [234]. The calorimeters can measure the direction and energy of both charged and neutral particles. In particular, the ECAL measures the energies of emerging particles like electrons and photons, whereas the HCAL estimates the energy of hadrons. In addition, the HCAL can indirectly detect the presence of non-interacting, uncharged particles such as neutrinos: measuring these particles is important as they can tell us if new particles have been formed. The HCAL is designed to capture every particle emerging from the collision. e.g., by observing particles that are shot out on one side of the detector but not the other (with an imbalance in the momentum and energy), it is, in principle, possible to deduce the production of “invisible” particles, which are particles that do not leave records of their presence in any part of the CMS detector. The two sub-detectors have different granularity: for the ECAL, the granularity of $0.0174 \times 0.0174 \text{ rad}^2$ results in $286\eta \times 360\phi$ bins for the size of the images, whereas the HCAL is 25 times less granular, i.e., $0.087 \times 0.087 \text{ rad}^2$. Therefore, each HCAL image is upsampled by a factor of 25 in the preprocessing step, assigning each pixel a $1/25$ th of the energy.

Detectors are vital equipment for the discovery of new particles. Within these apparatuses several collisions happen, producing an even larger volume of data. The collection, filtering, and lastly processing of these events can (hopefully) shed new light on new fundamental particles, interactions, and physics processes.

3.5.3 Simulated particle collisions

The dataset employed in our study contains simulated images of size $360 \times 286 \times 3^{18}$, for a total of about 615k samples, divided in 442k QCD, 67k SUEPs, and 106k SVJs. The image channels represent two-dimensional E_T (energy) deposits in the η - ϕ plane, measured by the Inner tracker (Trk), ECAL, and HCAL sub-detectors of CMS, respectively. In practice, each image is annotated with a:

- **Class label.** The label 0 indicates the QCD background, the label 1 is associated to SUEP signal samples, and the label 2 refers to the second SVJ signal. Considering an anomaly detection setting, events labeled with one or two are depicted as anomalous.
- **Mass label.** Signal samples only are identified by the mediator particle masses m_H [223] and $m_{Z'}$ [231] at which these were generated. In particular, SUEPs were generated at $m_H = \{125, 200, 300, 400, 700, 1000\} \text{ GeV}$ and the SVJ signal at $m_{Z'} = \{2.1, 3.1, 4.1\} \text{ TeV}$. For simplicity, we denote a particular signal sample by its mediator mass, such as $\text{SUEP}(m_H \text{ GeV})$ and $\text{SVJ}(m_{Z'} \text{ TeV})$.
- **Number of tracks.** This is a model-independent quantity that best approximates the number of decay products, obtained by the particle-flow reconstruction algorithm

¹⁸The images are transposed for convenience, otherwise the size (considering the detector geometry in η - ϕ) would be 286×360 , instead.

[232]. We refer to this variable as **nTracks**. It should be noted that computing this quantity is expensive, although not feasible for real-time inference at the HLT. Therefore, we will utilize it as a baseline discriminator to determine the improved detection performance of the proposed models.

In particular, SM multi-jet QCD, as well as SVJ and SUEP events, are generated using **PYTHIA 8.244** [45], simulated and reconstructed with **DELPHES 3.5** [198] fast detector simulation using the CMS detector configuration, to obtain the tracker, ECAL, and HCAL images. More details are available in the appendix B.1.

Event preselection. A preselection criterion is applied to background and signal events based on the detector’s geometry and typical characteristics of QCD events. In particular, the $|\eta|$ of E_T deposits is restricted to be less than 2.5, as defined by the inner tracker coverage. Then, the sum of E_T deposits of reconstructed electrons, muons, photons, and jets in an event is required to be > 500 GeV, which is a typical requirement at the HLT after L1 trigger selection during Run 3 [235]. The event preselection efficiency is about 85% for QCD; ranging from 2 – 22% for SUEPs; and almost 10% for SVJs.

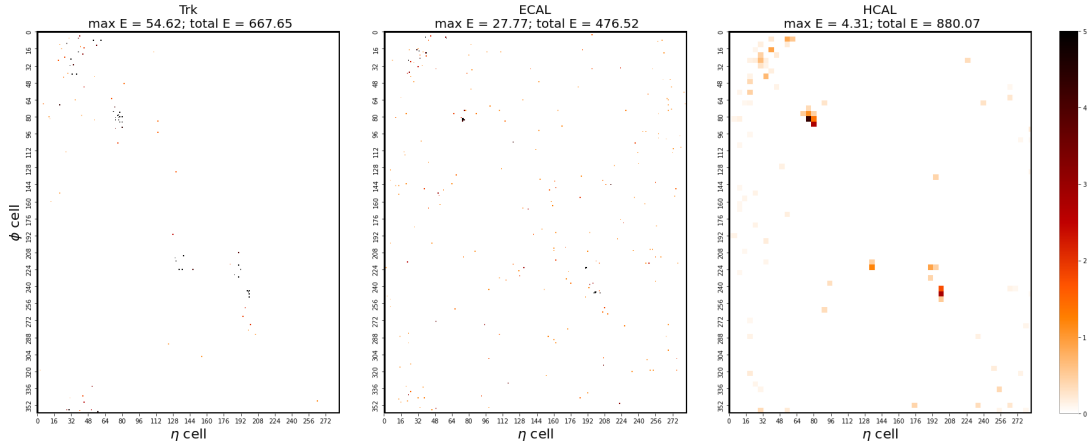


Figure 3.13: E_T deposits for a QCD event in the inner tracker (left), ECAL (middle), and HCAL (right). To improve the visualization of the energy deposits, the intensity of each pixel has been limited to a maximum value of five as shown by the colored bar on the right. The same is applied to the next two figures. “max E ” denotes the intensity of the brightest pixel, while “total E ” is the sum of all pixel values.

The plots showing the E_T deposits for a QCD event, a SUEP(400 GeV) event, and a SVJ(3.1 TeV) event, which satisfies the preselection criterion, in the inner tracker, ECAL, and HCAL are given in figures 3.13, 3.14, and 3.15 respectively. For the particular QCD event shown in figure 3.13, we can appreciate the typical signature of two jets in the η - ϕ plane: one mostly localized in the upper-left corner, and the other around the lower-right one. Also, we acknowledge the sparse nature of data: only about 15k pixels out of a total

of 308k image pixels have non-zero values. This observation is valid in general for all QCD events. On the other hand, for the SUEP(400 GeV) event (figure 3.14), there is a clear spherically-symmetric signature of a large number of low- E_T particles in ϕ .

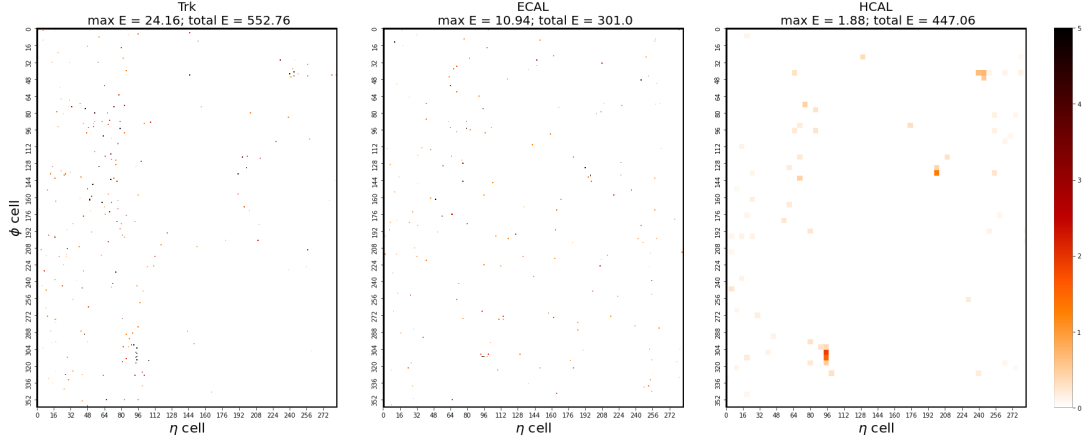


Figure 3.14: E_T deposits for a random SUEP(400 GeV) event in the inner tracker (left), ECAL (middle), and HCAL (right).

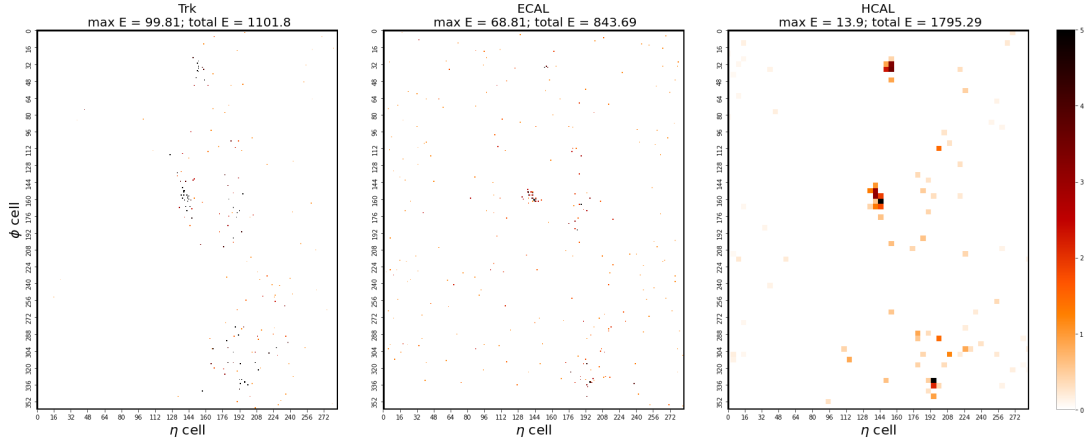


Figure 3.15: E_T deposits for a random SVJ(3.1 TeV) event in the inner tracker (left), ECAL (middle), and HCAL (right).

Image preprocessing. Since the images are very sparse, having about 99.4% of zero pixels, and also moderately large, we employ a simple preprocessing (as described by algorithm 1) that down-scales the images, thus reducing sparsity while also preserving their total energy. The down-scaling is performed by convolving a 5×5 kernel with all ones on the input images along the channel dimension (i.e., in a depth-wise fashion), in a non-overlapping manner with a stride equal to the kernel size, yielding a $25\times$ reduction in spatial resolution while preserving the sum of the energy deposits: the sparsity is also reduced to 96%; zero-padding is also applied (according to the “same” rule) to let the output size be divisible by the kernel size. The last step of the preprocessing is to discard the HCAL and ECAL channels, considering only the tracker one, resulting in images of size $72 \times 58 \times 1$: smaller images are faster to process by the network and require less storage, allowing to save up parameters, memory, computation and time.

Algorithm 1: Image Preprocessing

Input: a batch of images $I \in \mathbb{R}^{B \times H \times W \times C}$, down-sampling factor K
Output: preprocessed images $I_{\text{mask}} \in \mathbb{R}^{B \times \lceil H/K \rceil \times \lceil W/K \rceil \times 1}$

```

/* Depthwise convolution to down-sample each channel by a factor of K */
1 kernel = tf.ones((K, K, C, 1))
2 I' = tf.nn.depthwise_conv2d(I, filter = kernel, strides = (1, K, K, 1),
3                               padding = "SAME")
/* Consider only the tracker channel, discarding the other two */
4 Itrk = I'[:, :, :, 0, tf.newaxis]
/* Compute the mask image */
5 Imask = tf.cast(Itrk > 0, dtype = float)
6 return Imask

```

Discarding the calorimeter information seems an unusual choice at first. This is motivated by the fact that, for our signals, using the tracking information rather than the energy allows our models to be more sensitive to the new physics. Notice that this is a signal-dependent choice that introduces a weak form of model dependence in our workflow: this means that *our exact setup is not expected to work for signals in which tracks information is a weak discriminator*. Therefore, other signal types may benefit from tracking and calorimeter information, or even from only calorimeter data. Knapen et al. [223] provide a possible explanation of this fact, in which at the calorimeter level the SUEP mostly resembles the pile-up interactions since lacking hard and isolated objects, therefore, being more noisy than informative: this was also confirmed by my experiments.

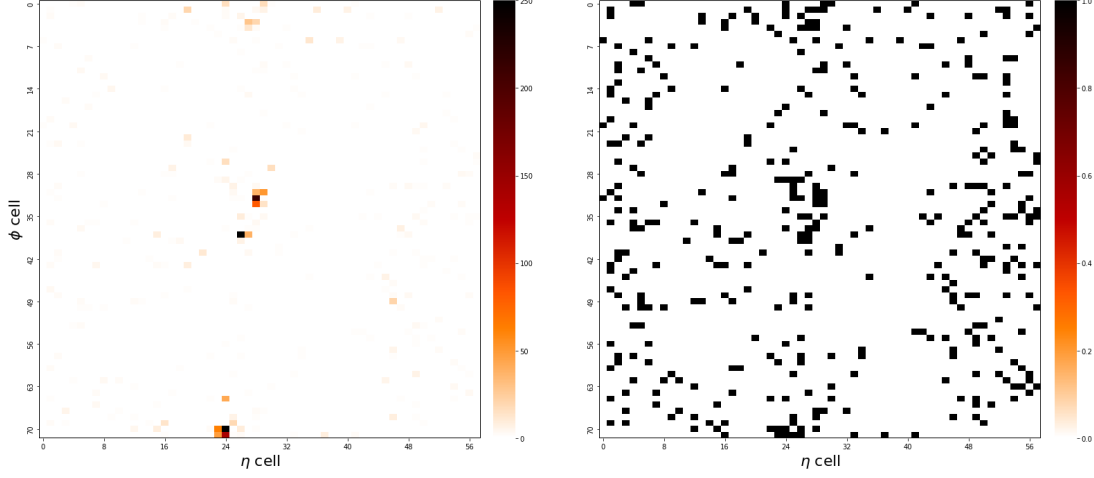


Figure 3.16: Energy, I , (left) and mask, I_m , (right) images of a single sample. Tracks information is more discriminative than energy deposits, so we train on I_m instead of I . The colored bars denote the pixel intensities.

The above preprocessing starts from an energy image to obtain what we call the *mask* image (figure 3.16) of the tracker channel. Since both the energy deposits and the **nTracks** variable can be seen as physics-motivated discriminators, the latter being much more sensitive to the searched signals than the energy, ideally representing a better input for our models. However, the number of tracks is unavailable for novel samples because running the particle-flow reconstruction algorithm would exceed the HLT latency constraint, preventing us from targeting this trigger. Therefore, by using the mask images we are effectively approximating the **nTracks** without incurring the extra cost of running track reconstruction. The mask image, denoted by I_m , can be easily computed from the image of energy deposits, I , by determining whether or not a pixel depicts a non-zero energy value: i.e., $I_m = \mathbf{1}[I > 0]$, where the indicator function is applied pixel-wise. Hence, since each pixel of I_m denotes whether a track occurred or not, summing a mask image yields the number of non-zero energy deposits associated with sensors in the detector that measured some energy. This quantity is similar but not equivalent to the **nTracks** variable since, depending on the granularity used to yield the images, two or more tracks can fall in the same bin (pixel) thus being not distinguished when counting non-zero deposits in images rather than counting the reconstructed tracks in the detector. Additionally, pixels in I_n have the benefit of being either zero or one, avoiding the need to normalize the E_T deposits that are often skewed towards small values inducing the presence of several outliers. In this context, normalization in the presence of outliers is always risky because it can cause a shift in distribution on test images or, even worse, a spurious correlation with the signals making them easier to identify. Instead, having identical value distributions between the background and signal avoids this issue, making the proposed method more robust and trustworthy since leveraging only the learned features to implement the anomaly scores.

3.5.4 Physics-inspired image augmentations

Since our data have physical properties like total energy, and the E_T deposits are arranged according to the design of the detector (other than being the result of a physics phenomenon), we cannot simply apply the usual off-the-shelf image augmentations like random crop, cutout, rotation, and jittering [236] that would reorganize the image' pixels without following the underlying physics and also without preserving both the individual and overall value of energy deposits. For this reason, we design novel data augmentations that retain the physical meaning of the images by working on the η - ϕ plane, hence, also respecting the geometry of the detector.

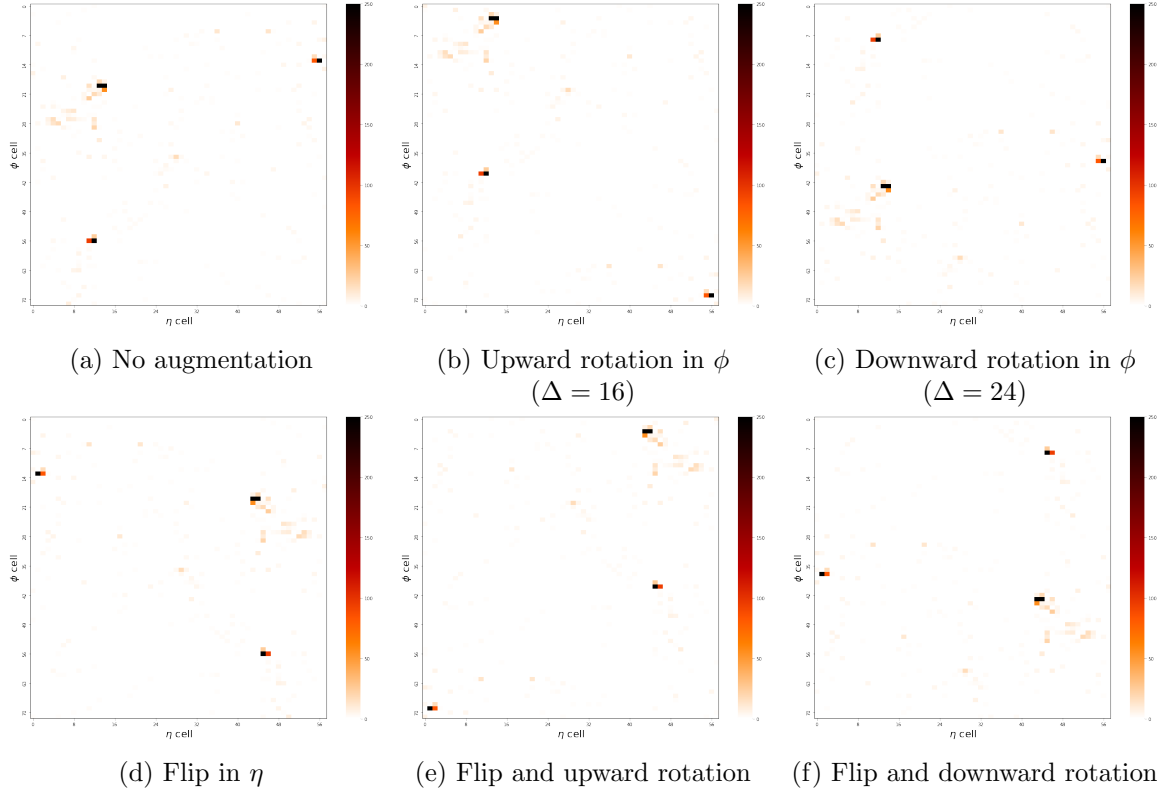


Figure 3.17: Example of five kinds of data augmentations, demonstrated on a random QCD sample considering energy deposits, whose value scale is depicted by the colored bar.

In particular, one kind of image augmentation involves a flipping in η , while the other is a rotation in ϕ : considering the figure 3.12 as a visual reference, the former can be interpreted as considering particles moving in the opposite direction concerning the beam-line (i.e., along the detector's z -axis), whereas the latter as rotating clockwise or anticlockwise the whole collision event on the detector's x -axis. The η -flip augmentation is simply implemented by mirroring the x -axis from left to right, where the ϕ -rotation is a little more

complex. Rotation in ϕ (i.e., along the image’s y -axis¹⁹) can occur both upward (i.e., anticlockwise) and downward (clockwise), in which a portion of the image moves up (or down). Consequently, the part in excess (the one that would fall off vertically from the image boundaries) is then attached to the bottom (or top). From a practical perspective, the ϕ -rotation is done in chunks of Δ rows, in which the chunk size is uniformly sampled for each image that should be rotated from the set, $\Delta \in \{8, 16, \dots, 56\}$, whose values are only multiples of eight: a hyperparameter value found to work well experimentally²⁰. By combining these two kinds of image augmentations (flipping and rotation), it is possible to yield a total of five combinations of augmentations: 1) upward rotation, 2) downward rotation, 3) flipping, 4) flipping and upward rotation, and lastly 5) flipping with downward rotation.

The image augmentations presented in figure 3.17 can be applied to raw images of particle collisions, regardless of the specific kind of signal and background processes. Moreover, these are also designed to encourage the model to be invariant to the detector geometry: learning the properties of the detector’s coordinate space is beneficial for anomaly detection, classification, and regression problems. Through data augmentations, it is, in principle, possible to introduce specific physics knowledge inducing the model to learn or be invariant and robust to it.

3.5.5 Conditional dual auto-encoders

Capturing a latent space that is both discriminative for anomaly detection, and high-capacity for accurate pixel-level reconstructions can be challenging due to a trade-off between the size of the latent space and the reconstruction quality. Large latent spaces can yield accurate reconstructions, especially in the presence of challenging (e.g., structured, sparse) data. However, these cannot be used directly as AD scores due to their high dimensionality unless summarized somehow. On the other hand, small latent spaces can encode discriminative features, but often at the cost of poor reconstructions due to the low-dimensionality that does not retain pixel-level details: *this fact can stop the training prematurely, resulting in sub-optimal anomaly scores that may not even discriminate the anomalies*. Since the model is trained to maximize the reconstruction fidelity, ensuring a good convergence of the loss can indirectly improve the anomaly scores too since these are, even if not optimized directly, defined from the reconstructions: reaching a better optimum entails a lower reconstruction loss on the normal examples, which, in the context of AEs, means obtaining a better (i.e., more structured, rich) latent space and a more accurate decoder.

¹⁹We refer to Cartesian x and y axes in the context of images, rather than x and y (i.e., rapidity) as detector coordinates.

²⁰In principle this can be generalized, for example, by uniformly sampling a positive integer in a given interval: i.e., $\Delta \sim \{n_0, n_0 + 1, \dots, n_1 - 1, n_1\}$ with $n_0, n_1 \in \mathbb{N}$ and $n_0 < n_1$.

Given the trade-off between the size of the latent space and the corresponding reconstruction quality, the intuition is to design an auto-encoder model with *two encoding paths* (i.e., with two encoders): one devoted to capture the rich detail of the input images, and the other one responsible for extracting discriminative features from the inputs. This is the idea behind the family of conditional dual auto-encoders (or CoDAE in short), pictured in figure 3.18, in which the aim is to learn two disentangled encoder networks without any additional supervision but through inductive biases instead.

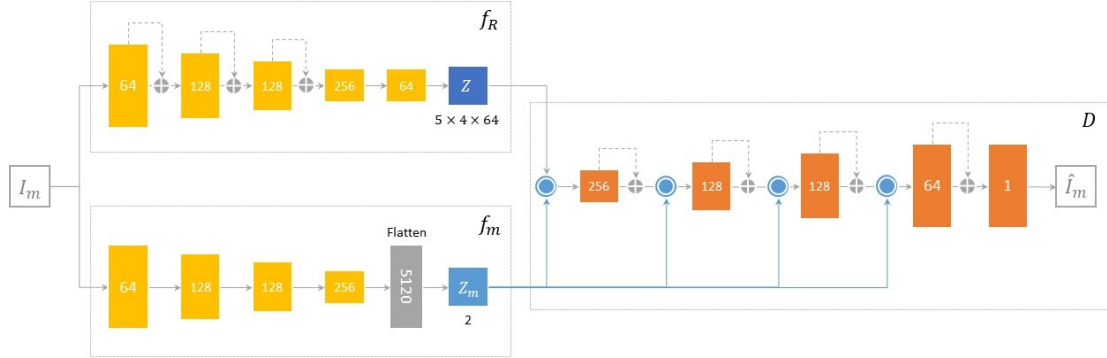


Figure 3.18: The CoDAE architecture: convolutional and up-sampling blocks in yellow and orange, respectively, gray dashed arrows and a $\oplus$ symbol depict a skip (or residual) connection [23], lastly blue solid arrows and the $\otimes$ symbol denote spatial conditioning of Z_m on Z . Each block uses 3×3 convolutions followed by instance normalization [203] and Leaky-ReLU [204]. Down-sampling in the encoder is performed by strided convolutions instead of pooling. The details are available in the appendix B.2.

The first encoder, f_R , is a residual network [23] that embeds the input mask images in a large convolutional latent space, $Z = f_R(I_m)$, of size $|Z| = 5 \times 4 \times 64$: given its large capacity, the latent components are expected to retain enough information to let the decoder yield high-quality reconstructions. The *auxiliary* encoder f_m , instead, is a shallower convolutional network aimed at learning a compact and discriminative auxiliary latent space, $Z_m = f_m(I_m)$ where $|Z_m| = 2$, such that its single components can be directly used as anomaly scores. Furthermore, the two networks have different architectures²¹ to induce a bias during training:

²¹In the general case, the two encoders can receive diverse inputs, have distinct architectures, and even diverse latent space sizes. In this case, the input is the same but the architecture is not. Another conceivable special case occurs when the input is diverse in content and the latent spaces are equally sized. If so, it is possible to combine the two spaces into a single one utilizing a learned linear combination, for example.

- The residual encoder, f_R , is high capacity and its skip connections can propagate information deeply in the layers' hierarchy helping to retain pixel-level details, which are then exploited by the decoder network to produce high-fidelity reconstructed samples instead of artifacts. These *artifacts* appear when the dimensionality of the latent space is too small to represent the complex and sparse input images, even if employing a very powerful encoder. For example, the artifact in figure 3.19 shows an elongated shape whose

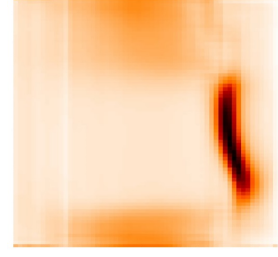


Figure 3.19: A reconstruction artifact.

- ally, in the corresponding truth image, there are only very few grouped pixels with a discrete amount of energy. Therefore we may conclude that the position along the x -axis of these artifacts is weakly correlated with the x -coordinate of the most energetic pixel in the original input. Indeed, when inspecting these reconstructions it is complicated to understand the parts in which the model is the most uncertain, which may correspond to abnormal values. Moreover, it would also be difficult to derive anomaly explanations or maps (as in [126]) from those artifacts.
- The max-pooling layers of the shallow encoder, f_m , are meant to consider only the most important activations to enhance the discrimination power of Z_m . Compared to other AE architectures like the one in [85], employing average pooling in the encoder may aid the decoder in reconstructing the inputs since the information about the content is still propagated. However, it can cancel out (since averaging) discriminative features that are expected to be preserved by a maximum operation, in which the most important (i.e., large in value) activations are retained instead. Therefore, even a more complex model is not said to be better at learning discriminative features in its latent space, since it can be more suited to encode detail rather than the peculiarities of the inputs.

For clarity, we refer to Z as the *large* or *convolutional latents*, and Z_m as the *auxiliary* or *compact latents* for the rest of this thesis chapter.

Bringing information together. The second part of the CoDAE architecture (figure 3.18) is the conditional decoder, D , which is a residual network [23] whose main input is the convolutional latent space, Z (i.e., the output of the residual encoder), from which it tries to reconstruct the input mask images, I_m . The latent space, Z , is sufficiently large to provide enough information to the decoder to enable high-quality reconstructions, but, the main question is about how to allow the auxiliary encoder to learn a compact latent space, Z_m , without introducing any additional supervision. One possible answer is provided by *conditioning* [72, 237], which establishes a dependency between the decoder and the auxiliary latent space such that to allow the gradients of the loss to flow through f_m without any explicit supervision, enabling the model to train its weights.

During training, the conditioning mechanism propagates the reconstruction error also to the shallow encoder providing feedback to learn Z_m to it, such as to maximize the reconstruction quality. Intuitively, we can write $\hat{I}_m = D(Z | Z_m)$ to highlight the fact that the compact latents, Z_m , must influence the reconstructions, $\hat{I}_m$, to represent meaningful and not just randomly encoded features. For this reason, the conditioning should be strong enough to prevent D from solely relying on the convolutional latents. If this happens, the contribution of the gradient w.r.t. the parameters of the auxiliary encoder will be negligible, hence preventing it from learning an effective latent space since the contribution of this model in the overall architecture would be neglected because the decoder could reconstruct with low error only from Z .

Algorithm 2: Spatial Multiplicative Conditioning

Input: latents $Z_m \in \mathbb{R}^2$, tensor $h_i \in \mathbb{R}^{H_i \times W_i \times C_i}$, kernel size K
Output: spatially conditioned representation $r_i \in \mathbb{R}^{H_i \times W_i \times C_i}$

```

/* See algorithm 1 in [238] */
1  $z = \text{SpatialBroadcast}(Z_m, W_i, H_i)$ 
/* Expand channels of  $z$  to match  $h_i$ , through a linear convolution */
2  $z = \text{Conv2D}(\text{filters} = C_i, \text{kernel\_size} = K, \text{padding} = \text{'same'})(z)$ 
/* Multiplicative conditioning: Hadamard product  $z \odot h_i$  */
3  $r_i = \text{tf.multiply}(z, h_i)$ 
4 return  $r_i$ 
```

In particular, our form of conditioning combines *spatial broadcast* [238] with a feature-wise transformation [72]: element-wise multiplication, also known as scaling. We will refer to this operation as *spatial conditioning*: described by algorithm 2 and equation 3.5.1; some feature-wise transformations were discussed in section 2.3.2, whereas the spatial broadcast in section 3.4.4. The spatial broadcast (SB) operation provides an inductive bias to the convolutional encoder, f_m , for learning *disentangled* latent factors in Z_m (which should encourage to capture independent features over the latent components) while, at the same time, modulating Z and the subsequent hidden feature maps. Taking the figure 3.18 as a reference, the spatial conditioning is performed on multiple spatial resolutions of the decoder’s hierarchy of layers, applying it before each up-sampling block. Initially, at stage $i = 0$, the conditioning is executed on the convolutional latents (i.e., $h_0 = Z$) which are then fed to the first layers of the decoder. Subsequently, for $i > 0$, the hidden feature maps (i.e., output of the previous residual block in stage $i - 1$), h_i , are conditioned on the same Z_m . In this way, the auxiliary latent space effectively modulates the decoder at different spatial resolutions forcing the intermediate feature maps to be conditioned by the auxiliary latents, this also implies each subsequent block to adapt on the previously conditioned representation.

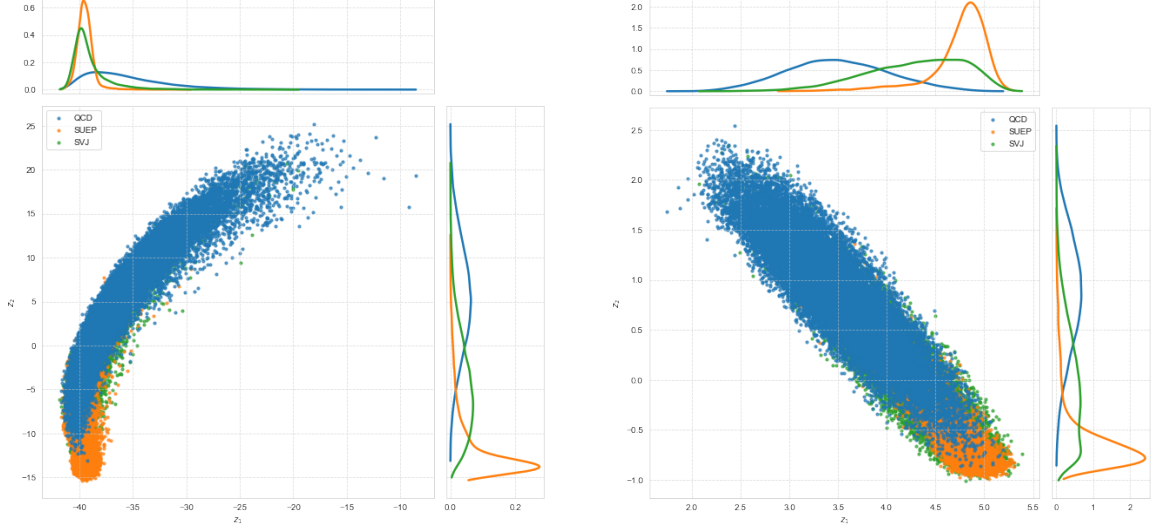


Figure 3.20: The CoDAE’s (left) and CoDVAE’s (right) auxiliary latent space, Z_m , along with kernel density estimates of its components: the best components are, respectively, Z_2 and Z_1 . As we can see in the density plots, the QCD (blue) and SUEPs (orange) look well separated, instead the SVJs (green) span in between the two classes.

Mathematically, the conditioning operation performed on a generic tensor h_i , of size $H_i \times W_i \times C_i$, can be written as:

$$r_i = \text{Conv}(\text{SB}(Z_m)) \odot h_i, \quad (3.5.1)$$

where SB (see algorithm 1 in [238]) replicates and expands Z_m to match the shape of $H_i \times W_i$, and the subsequent convolution (Conv) linearly expands the channels of the intermediate result to C_i , to finally perform the Hadamard product (denoted by $\odot$) with h_i , yielding the conditioned representation r_i at stage i . The result of the conditioning can be appreciated in figure 3.20, in which we can observe how the propagation of the gradient using the multiple spatial conditioning has structured the auxiliary latent space.

Generally, the spatial conditioning operation is not limited to only multiplicative interactions. Other conditioning mechanisms applied on feature maps, like biasing, concatenation, or even an affine-like operation that combines scaling with biasing, are feasible. In principle, it is also possible to exploit the domain knowledge of the problem and data to devise an application-specific conditioning mechanism that utilizes such knowledge: for example, we may think of conditioning on a relevant physics observable like the particle’s mass, energy, or variables that explain the detector’s geometry.

Categorical CoDVAE. The described CoDAE architecture can be easily extended to variational auto-encoders [161]. In particular, we explored the Categorical VAE [200, 163] in which the convolutional latents, Z , are sampled from a Gumbel-Softmax (or Concrete) distribution (section 3.4.2): $Z \sim \text{Cat}(\alpha_R, \tau_R)$, where $\alpha_R, \tau_R = f_R(I_m)$ are the learned logits

and temperature. The motivation is that the Categorical VAE can learn discrete features, like counts. Instead, the regular (i.e., Gaussian) VAE captures continuous quantities that are not completely related to the **nTracks** variable, which provides the number of tracks: the aim is to learn a tracks-related quantity but by using the mask images as input, instead. We call this model the *Categorical Conditional Dual Variational Auto-Encoder*, or CoDVAE in short. Likewise, in previous approaches [239, 240], we let the residual encoder also output a temperature τ_R , which can be difficult to tune properly, defining the degree of relaxation of the Categorical distribution (approximated by the Gumbel-Softmax) parameterized by α_R . We add a second point-wise convolution with a softplus activation, to ensure positive values, on the last residual block of figure 3.18 (diagram section about f_R). Then, a base temperature, τ_0 , which is a hyperparameter set to 1, is added to τ_R to avoid gradient instabilities due to small numbers as $\tau_R \rightarrow 0$ recovers the true Categorical distribution: conversely, if $\tau_R \rightarrow \infty$ the distribution approaches a Uniform one. Since we learn the temperature, we define a more complex prior $p(Z)$ as a uniform mixture of N Gumbel-Softmax with different temperatures sampled uniformly in $[0.1, 1]$, to provide more flexibility to the latent space:

$$p(Z) = \frac{1}{N} \sum_i^N \text{Cat}(\log 1/C, \tau_i), \quad \tau_i \sim U(0.1, 1). \quad (3.5.2)$$

We then approximate the KL divergence (D_{KL}) between the prior and the learned distribution, $z \sim q_\phi(z | x^{(i)}) = \text{Cat}(\alpha_R^{(i)}, \tau_R^{(i)})$, with a Monte Carlo estimate ($M = 10$), as follows:

$$D_{KL}(q_\phi(z | x) \parallel p(z)) \approx \frac{1}{M} \sum_{z \sim q_\phi(\cdot | x)}^M \log q_\phi(z | x) - \log p(z). \quad (3.5.3)$$

To enable sharp reconstructions of mask images, we opt for learning a probabilistic decoder in which each output pixel is independently governed by a Bernoulli distribution, which can represent pixel values that are either zero or one, as in I_m . A single Bernoulli distribution can be denoted as $\text{Bern}(p_\theta)$ where p_θ is learned and specific for a single pixel. For sharp predictions we take the *mode* of such distribution since it is 1 if $p_\theta > \frac{1}{2}$ and 0 otherwise: its mean, instead, is suitable to represent smooth values in $[0, 1]$ like normalized energy deposits.

Compared to the CoDAE, the CoDVAE can, in principle, also generate new samples that look like the training ones. In particular, the dual encoder architecture allows for *conditional generation* in which the features encoded in Z_m can be used to guide the generation of samples by the CoDVAE model. Ideally, this capability may allow the CoDVAE to be employed for fast simulations of events.

Training and acceleration. During training, we make use of the set of augmentation functions, $\mathcal{T}$, defined in section 3.5.4. At each mini-batch of mask images I_m a random augmentation is sampled $t \sim \mathcal{T}$ and applied to it. The augmented images, $\tilde{I} = t(I_m)$, are

then fed to both residual and auxiliary encoders; the decoder is then trained to reconstruct $\tilde{I}$. The data augmentations enable the CoDAE models to learn invariances related to the coordinates η and ϕ of the detector, which can also be seen as an implicit way to impart some physics notions to the model. In addition, we perform model selection according to the value of the *structural similarity index* (SSIM) metric [241] between the true and reconstructed images, which provides a more human-aligned measure of image quality, computed on a validation set of only background samples. In particular, the SSIM is built by assuming the human visual perception is highly adapted for extracting structural information from a scene, therefore the quality assessment is based on the degradation of structural details. Given a pair of images, x and y , the SSIM compares local patterns of pixel intensities that have been normalized for luminance and contrast such that:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c)(2\sigma_{xy} + c)}{(\mu_x^2 + \mu_y^2 + c)(\sigma_x^2 + \sigma_y^2 + c)}, \quad (3.5.4)$$

where μ_x, μ_y is an estimate of the luminance of each signal as their mean intensity, σ_x, σ_y their standard deviation as an estimate of the contrast, $\sigma_{xy} = \mathbb{E}[(x - \mu_x)(y - \mu_y)]$ the correlation coefficient between the two images, and $c \propto L^2$ a constant proportional to the image dynamic range, L . The above formulation can capture the perceptual quality of visual degradation (e.g., change in contrast, compression, blurriness, noise, etc), whose MSE is nearly identical since it only considers the average of pixel intensities as demonstrated in [241].

The whole CoDAE models are learned end-to-end using the **AdamW** optimizer [107], whose learning rule decouples the weight decay regularization term from the main objective [60] making it easier to tune, minimizing the binary cross-entropy loss²². It turns out that the strength of the l_2 -regularization, λ , is tightly coupled with the learning rate, η : e.g., in the case of SGD, the weight decay regularization coefficient should be set to $\lambda' = \frac{\lambda}{\eta}$, instead, to be equal to l_2 -regularization. For the Adam optimizer, the discrepancy is worse since the weights associated with large historic gradient amplitudes are regularized less when using weight decay, causing regularization to be less effective for generalization. Therefore, the fix in this case, is to rescale the gradients by a factor of $\eta v^{(t)} / \sqrt{s^{(t)}}$, where v and s are the estimates of the first and second moments. For training, the **AdamW** optimizer is left with default parameters and learning rate, except for the weight decay coefficient set to 10^{-4} . Furthermore, to improve training stability, we limit the l_2 -norm of each gradient to be at most one. Lastly, all the weights are initialized by following the **he_uniform** [22] scheme with zero biases except for the decoder, whose biases are initialized to -1 to provide a better starting point for the initial reconstructions. The CoDAE is trained for 30 epochs using a batch size of 128. Instead, the CoDVAE is trained for 100 epochs due to its slower convergence.

We use *TensorFlow lite* [73] for both model compression and acceleration of our CoDAE models. In particular, we employ a very light **float16** (i.e., half floating-point precision)

²²The loss is summed over spatial dimensions (height, width, and channels), and averaged over the batch size.

weight and activation quantization that resulted in a $5.7\times$ reduction in model size and lower inference time, without any decrease in anomaly detection and reconstruction performance. In this way, we can achieve inference latency on a single consumer CPU hardware of about: 3 ms (i.e., $6 - 8\times$ faster) for the auxiliary encoder, and 40 ms (i.e., $3.2 - 4.1\times$ faster) for a full forward pass on the whole model; therefore well under the 100 ms time-limit of the HLT. This also demonstrates that our model architecture is prone to be optimized for deployment. In principle, the models can be further compressed with an integer quantization with successive retraining to limit the degradation of predictions.

3.5.6 Evaluation protocol and results

This section first presents several anomaly detection scores, comparing them to supervised approaches, related methods, and physics-inspired baseline discriminators. These are all evaluated using metrics commonly used in classification, anomaly detection, and out-of-distribution detection. Moreover, we present the reconstruction capability of our models compared to competing approaches.

Anomaly scores. As discussed in section 3.3.2, AEs provide the opportunity to define a variety of scores to perform anomaly detection. In addition to those scores, our Co-DAE architecture introduces additional latent-based scores by learning an auxiliary, low-dimensional latent space, which allows each component to be an anomaly score: in our particular case $|Z_m| = 2$, so we have two extra discriminators²³. Furthermore, with our CoDVAE model, we can exploit the fact that the KL divergence between the learned posterior and prior distributions is not *symmetrical*: $D_{KL}(q \parallel p)$, i.e., the reverse KL, and $D_{KL}(p \parallel q)$, which is the forward KL, have two different meanings as shown in figure 3.5. The former promotes mode-seeking behavior while the latter encourages coverage of probability mass. We exploit these two additional latent-based scores (denoted as KL-R and KL-F, respectively) with our Categorical CoDVAE model.

Let be x an input image and $\hat{x}$ its reconstruction. To ease the notation, we denote the set of pixel indices as $\mathcal{P} = \{i, j, k \mid i = 0, \dots, H - 1, j = 0, \dots, W - 1, k = 0, \dots, C - 1\}$ for images of size $H \times W \times C$, and define our reconstruction-based scores on that as follows:

- **BCE scores**, defined as the sum of the *binary-cross entropy* between the true and predicted pixels.

$$\text{BCE}(x, \hat{x}) = - \sum_{p \in \mathcal{P}} x_p \log \hat{x}_p + (1 - x_p) \log(1 - \hat{x}_p). \quad (3.5.5)$$

Since the pixel values in each mask image are either zero or one, this measure is well-defined, without any normalization. Considering the individual pixels being governed by independent Bernoulli distributions and summing their cross-entropy, we can

²³In principle, the latent space Z_m could have an arbitrary number of dimensions but there is a trade-off: learning either too numerous or too few (e.g., one) latent components may result in individual scores with weak discriminatory power.

interpret the above score as quantifying the likelihood of a reconstructed sample conditional on the provided ground truth. Reconstructed samples from the two signals, which should be considered anomalous, are expected to yield lower likelihoods than the reconstructed background.

- **SSE scores**, which represent the *sum of squared errors* between the true and reconstructed pixels.

$$\text{SSE}(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{p \in \mathcal{P}} (\mathbf{x}_p - \hat{\mathbf{x}}_p)^2. \quad (3.5.6)$$

This anomaly score can be seen as a modification of the canonical MSE score (section 3.3.2), in which a summation occurs instead of averaging over spatial dimensions.

- **SAE scores**, depicting the *sum of absolute errors*, which is the absolute difference between true and predicted pixels.

$$\text{SAE}(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{p \in \mathcal{P}} |\mathbf{x}_p - \hat{\mathbf{x}}_p|. \quad (3.5.7)$$

It is worth noticing that if the Categorical CoDVAE model is evaluated on mask images, this metric is equal to the SSE since both squared and absolute differences can be either zero or one (according to whether the pixel is correctly predicted or not): this is a direct consequence of taking the mode of the learned Bernoulli decoder, whose output pixels are binary values. Thus, in such cases, we omit the results of the SAE scores. In general, the two metrics are expected to differ. Moreover, this score is related to the MAE in the same way as the SSE is related to the MSE.

- **Dice scores**, defined as the inverse of the Dice coefficient [242], to measure dissimilarity between two sets of pixels: $\mathbf{x}$ and $\hat{\mathbf{x}}$.

$$\text{Dice}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{\sum_{p \in \mathcal{P}} \mathbf{x}_p^2 + \sum_{p \in \mathcal{P}} \hat{\mathbf{x}}_p^2}{2 \sum_{p \in \mathcal{P}} \mathbf{x}_p \cdot \hat{\mathbf{x}}_p}. \quad (3.5.8)$$

Compared to the MSE and SSE, the Dice score tries to capture the global appearance of the image rather than focusing on its pixels. For this reason, the Dice coefficient is a popular metric in semantic segmentation to evaluate the coverage of the segmentation masks compared to the truth mask.

- **PixelSum score**, defined as the sum of each predicted pixel value.

$$\text{PixelSum}(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{p \in \mathcal{P}} \hat{\mathbf{x}}_p. \quad (3.5.9)$$

This score can be interpreted as *predicted total energy* if the model is trained to reconstruct I (i.e., the energy images), or as an approximation of the **nTracks** if reconstructing the mask images, I_m , instead.

The above reconstruction-based scores complement the already cited latent-based ones. Both categories of anomaly scores have their advantages and disadvantages. Reconstruction scores are generally easier to define, for example from common loss functions or metrics. However, these are slower to compute since a forward pass of the full model (encoder and decoder) is always required to reconstruct the samples. Instead, latent-based scores involve only the encoder predictions which are more suited for fast inference and model acceleration (e.g., FPGA deployment), although possibly more difficult to define (e.g., analytical or empirical KL), visualize and interpret (e.g., dimensionality reduction on high-dimensional latent space).

Baseline discriminators. For a fair comparison and assessment of our method, we determined various baseline discriminators: two physics-motivated ones, a fully supervised classifier, a convolutional auto-encoder (CAE), and two related anomaly detection models. The two physics discriminators are based on the *total energy* (i.e., the sum of energy deposits, E_T , in each image channel), and the **nTracks** variable, respectively. Specifically, the **nTracks** is a model-independent classical variable corresponding to the total number of tracks per event [229, 43]. Such quantity is related to the number of decay products providing the best approximation of such to discern the signal particles, being also independent of the binning used to discretize the detector resolution, unlike images. In particular, we define the total energy baseline as follows:

$$s^{(k)}(c) = \sum_i^H \sum_j^W x_{i,j,c}^{(k)}, \quad c \in \{0, 1, 2\} \quad (3.5.10)$$

where $s^{(k)}(c)$ is the score value for channel c (which denotes, respectively, the Trk, ECAL, and HCAL) of the k -th image $\mathbf{x}^{(k)}$ with height H and width W . Then, the discrimination will be performed according to the obtained scores, s , per channel²⁴.

Furthermore, we also consider the performance of a *supervised* classifier, therefore assuming an ideal setting where we would have perfect knowledge of the data: this is the typical scenario of signal-dependent new physics searches. Such a supervised baseline will provide a good approximation regarding upper-bound discrimination performance that our self-supervised model may achieve at its best. In particular, we consider a state-of-the-art model: a *Compact Convolutional Transformer* (CCT) [243], in which

- The image tokenizer is a two-layer CNN. Each convolutional layer has a 3×3 kernel, ReLU activation, and **he_normal** initialization of weights, which is then followed by overlapping max-pooling (i.e., pooling with a 3×3 kernel, instead of a 2×2 one, with stride 2).

²⁴In principle, it is also possible to sum over channels to calculate a single *total energy score* but, in practice, the discrimination power of each channel is partially incoherent, meaning that, when summed together, the beneficial effect of some channels is lost yielding lower performance than some of the individual energy scores.

- The CCT is made of two transformer blocks [3]. Each block has two heads in the multi-head attention layers, with 128 projection units, layer normalization [244], and a 10% dropout rate. A multi-layer perceptron with skip connections follows the attention layers. Each MLP has two layers with 128 units, 20% dropout rate, and GELU [245] activation. The skip connections are regularized by stochastic depth [246], in which a residual path is dropped with a rate of 20%.

The CCT model outputs the logits for our three classes (QCD, SUEPs, and SVJs), resulting in about 406k parameters. The CCT is trained for 30 epochs with a batch size of 128, and the physics-inspired image augmentations. The model is optimized by AdamW with our default hyperparameters. Model selection is performed according to the best AUROC.

The next baseline is a convolutional auto-encoder derived from our CoDAE: it has the same architecture (see appendix B.2) and hyperparameters, lacking only the second (shallower) encoder network and the spatial conditioning mechanism in the decoder since there is no additional input (i.e., CoDAE’s the auxiliary latent space) to perform conditioning on. This CAE model is trained in the same way the CoDAE models are.

The last baselines are two popular anomaly detection models: an unsupervised AE inspired by [85], and the Dirichlet VAE from [86]. In particular, the AE model has a total of 600k parameters, a latent space of size 32, and was trained to minimize a mean squared loss. Compared to our CoDAE, the AE lacks the second encoder and residual connections, embedding its inputs into dense vectors, and employing transposed convolutions to up-sample the feature maps during the decoding process. Instead, the Dirichlet VAE is a weakly supervised approach that, therefore, also requires a fraction of the signals for training: we assume a realistic setting in which the background is contaminated with 0.01% of the signals. This model has a three-dimensional latent space, whose prior distribution is the Dirichlet [187] instead of the Gaussian distribution, resulting in 2M parameters: during training, we normalize the images to sum to one, and use the same hyperparameters as in [86]. We apply the data augmentations defined in section 3.5.4 to both AD models. We trained the AE and Dirichlet VAE for 50 epochs (the latter converged earlier in training), with a batch size of 128, employing the AdamW optimizer.

The ASO test. Motivated by the fact that the performance of deep neural networks is dependent on many stochastic factors (e.g., sampling of the data, random weight initialization, dropout, etc) and specific hyperparameters (e.g., architectural and optimization-related), making it difficult to conclude which model is better than another one according to few (average) scores, especially when these show counter-intuitive results: e.g., one score is the best on a signal, and the worst on the other. Hence, to complement the AUROC and FPR40 metrics (see section 3.2.3), we employ the *Almost Stochastic Order* (ASO) test [247, 248] as implemented by [249] that provides a statistically significant result from which it can be decided the best performing algorithm.

The ASO test is specifically designed for neural networks, building on the concept of *stochastic order* in which one distribution of scores (e.g., AUROC) is said to stochastically dominate another one if the cumulative distribution of the former is lower than the latter

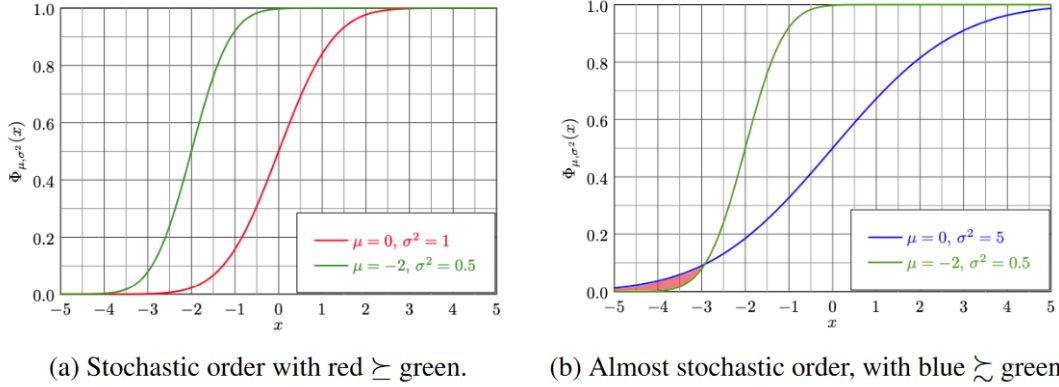


Figure 3.21: Examples of stochastic order (a) and almost stochastic order (b), illustrated using the cumulative density function of two normal random variables. Often the stochastic order is too strict to be practical since no violations are admitted. Instead, the ASO allows for some degree of violation (i.e., the red area in (b)). Figure from [249].

for every point. Since the stochastic dominance is too strict to be practical, the *almost stochastic dominance* is used instead, quantifying the extent to which stochastic order is violated: this is useful in practice since the real score distributions are not known and must be estimated empirically. Figure 3.21 visually compares these two measures. Therefore, the ASO test returns an upper bound, called $\epsilon_{\min}$, expressing the amount of violation: if $\epsilon_{\min} < \tau$ (where τ is the *rejection threshold* usually set to 0.5 or less, like 0.2 for a more confident result), then the former algorithm is superior than the latter. The value $\epsilon_{\min}$ can be interpreted as a confidence score: the lower it is, the more sure we can be about the dominance of one model over another. Experimentally, we follow the best practice suggested in [249] by fixing one set of hyperparameters and comparing multiple runs of the same model (where possible) by using 42, 51 and 73 as seeds for the pseudo-random number generators. Moreover, we build an empirical distribution of the AUROC by computing this metric on a thousand random, class-balanced subsets (with $2k$ samples each) of the data, for each benchmark mass point.

Reconstruction quality. To assess the reconstruction quality of the compared models, we evaluate two metrics: the mean squared error, and the SSIM index [241]. In particular, the popular MSE metric is useful for determining the texture quality of the reconstructions, since it penalizes pixel-level differences. Instead, the SSIM is a perceptual quality metric designed to match the perceived visual quality of humans more accurately. These two metrics are complementary: the MSE looks at fine details, while the SSIM considers the global appearance of the images. Using both provides a more comprehensive assessment of the quality of the reconstructed samples.

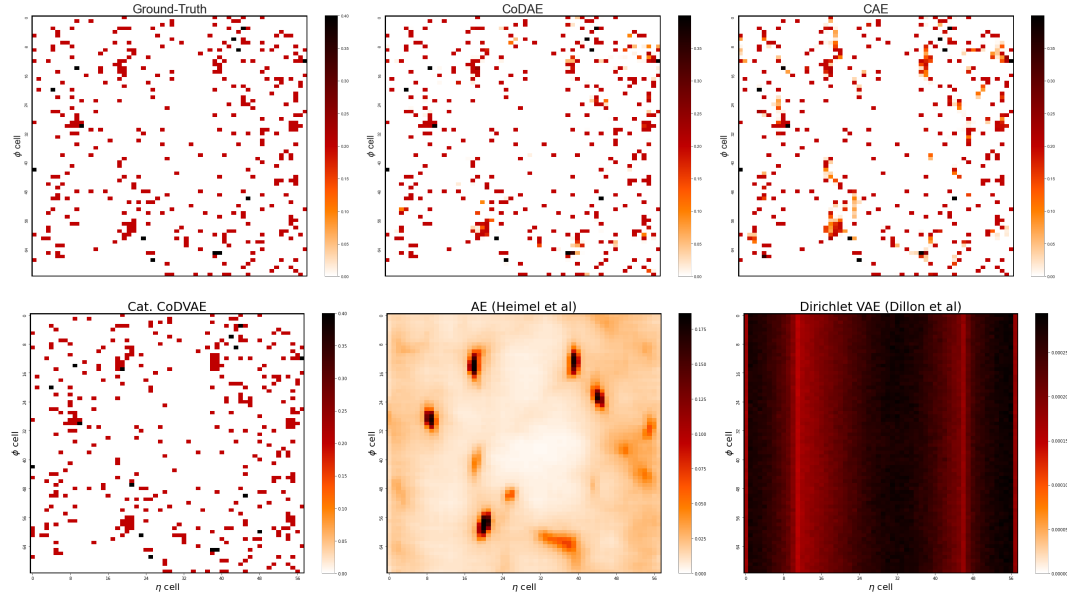
Metric	CoDAE	CAE	CoDVAE	AE [85]	Dirichlet VAE [86]
MSE ($\downarrow$)	5.5 ($\pm$ 5.6)	9.4 ($\pm$ 7.4)	17.7 ($\pm$ 13)	70.9 ($\pm$ 24.7)	79.3 ($\pm$ 26)
SSIM ($\uparrow$)	0.99 ($\pm$ 0.01)	0.98 ($\pm$ 0.02)	0.95 ($\pm$ 0.04)	0.36 ($\pm$ 0.12)	0.17 ($\pm$ 0.11)

Table 3.3: Evaluation of reconstruction quality: QCD test images. Lower values of MSE as well as higher SSIM ones are better. Each entry denotes the average metric value with its standard deviation (in parenthesis). By design, the Dirichlet VAE outputs images that sum to one, so we undo the normalization before computing the metrics to have pixel values on the same scale. The best results are shown in boldface.

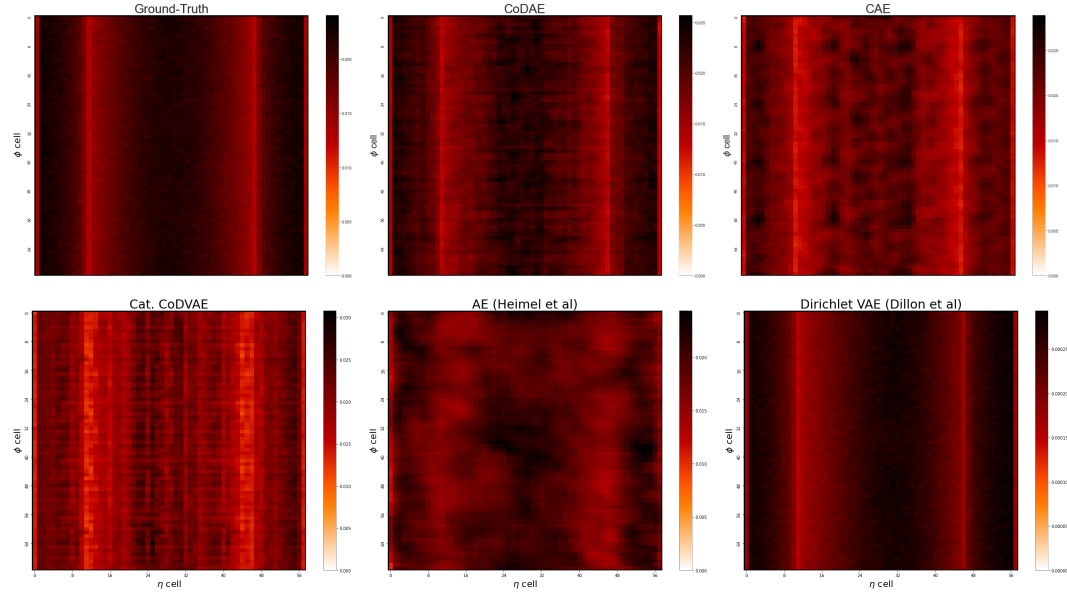
Reconstruction performance is summarized in table 3.3 as well visually in figure 3.22. From these results, we can deduce that the CoDAE, CAE, and CoDVAE models achieve the lowest MSE and highest structural similarity, attaining accurate single-sample reconstructions, which results in good predictions on average. As we can notice from table 3.3, all models show some variance in the reconstructed background, indicating that some QCD samples deviate from the ideal background event²⁵. Moreover, AE-based models show softer and smoother pixel predictions. Instead, the Categorical CoDVAE, thanks to its Bernoulli decoder, can sharply reconstruct by taking the mode of each learned distribution, one per output pixel. Furthermore, in figure 3.22 we can observe how the Dirichlet VAE, which is designed to capture a multi-modal latent space distribution, only identifies the “QCD mode” since each sample, regardless of being background or signal, is predicted as a sort of average of QCD images: this is confirmed by the low reconstruction metrics. We noticed similar behavior when training regular AEs with a too-small latent space (e.g., of size 2), even with a high-capacity residual encoder.

We want to highlight the importance of accurate (or at least coherent) reconstructions. Since anomaly detection scores can be defined from such predictions, it is necessary to avoid the model learning to predict some spurious pattern or artifact instead of the inputs, otherwise, it would be difficult to understand for a human expert why a new sample deviates from the training data: this is particularly useful when a given event is not stored for later processing. Moreover, since an auto-encoder is trained to maximize the reconstruction quality which, in turn, mostly affects the reconstruction-based anomaly scores, it is equally important to avoid premature convergence of the model since this can lead to sub-optimal anomaly detection performance: employing a large latent space paired with powerful encoder and decoder networks can mitigate such issue, as seen for our CoDAE and CoDVAE. This is essential to obtain coherent anomaly detection predictions, high discrimination performance, to have a trustworthy model, and even to debug the model itself, once deployed.

²⁵This fact can be used in principle to filter the QCD training data by removing the events that are less “QCD-like”. A successive training from scratch on the filtered data may improve the sensitivity to the anomalies.



(a) Reconstructions of five random QCD samples.



(b) Reconstructions of the QCD test-set.

Figure 3.22: Comparison of per-sample (a) and average (b) reconstructions. In order: ground-truth, CoDAE, CAE, Categorical CoDAE, AE [85], and Dirichlet VAE [86]. As we can see our models are capable of pixel-accurate single-sample reconstructions. On the contrary, the AE only captures the prominent pixels, and the Dirichlet VAE learns to predict an average image instead. As expected, the CAE is accurate since it shares the same encoder and decoder architecture as the CoDAE.

Anomaly detection results. In this paragraph the discrimination performance of our two models (the CoDAE and Categorical CoDVAE) are compared against the baselines (energy, number of tracks, supervised CCT, CAE, AE, Dirichlet VAE) defined previously. For all the models and baselines we computed the anomaly scores defined in section 3.5.6, where possible (e.g., it is not possible to calculate the KL for the AE), and evaluated their respective AUROC and FPR40: in the tables to follow, we also show the value of the metric averaged over all mediator masses for a given signal (i.e., averaged over m_H for the SUEPs, and over $m_{Z'}$ for the SVJs), denoting it as mAUC for the AUROC and mFPR for the FPR40. The evaluation results are summarized by tables 3.4 and 3.5.

Discussing the physics baselines, for the SUEPs we can see that the sum of E_T deposits of the signal is lower than the one of the QCD background, resulting in shifted distributions of scores that yield an AUROC below 50%. This total energy baseline indicates that one signal (the SUEPs) is less complex than the background: the complexity bias affects this score. Instead, such an issue does not occur for the **nTracks** baseline, which already provides high separation performance for both signals on both evaluation metrics, especially for the SUEPs at high mass; citing [223]: counting the number of tracks is particularly sensitive to high multiplicity soft particles like SUEPs, making them easier to identify. Compared to the AUROC, the FPR40 metric helps us better understand the shape of the ROC curve around our target signal efficiency of 40%, showing a very high (almost ideal) background rejection rate for the supervised classifier in all the benchmark scenarios.

Compared to the **nTracks** baseline, which requires counting the track multiplicity possible only if fully reconstructing the event as in an off-line analysis, our models are already able to get competitive performance in the latent space (showed in figure 3.20) and even improve by a neat margin when using reconstruction-based scores such as the BCE or SSE. We can notice that for some benchmark points, like SUEP(400 GeV), SUEP(700 GeV) and SUEP(1000 GeV), the AUROC easily saturates attaining an almost perfect background rejection, therefore the improvement brought by a data-driven approach is negligible. Indeed, these are easier to detect since they are expected to deviate significantly from the QCD background. Instead, in the most challenging scenarios our best model achieves an AUC improvement of at most +10.1% for SUEPs and +3.9% for SVJs, as well as a reduction of FPR of at most 6% for SUEPs and 2.6% for SVJs, at the predefined signal efficiency compared to the **nTracks** baseline. Our models can reduce the gap with the supervised classifier despite being trained only on the background class. Next, the AE underperforms the CAE and CoDAE, and the Dirichlet VAE is only competitive against the total energy baseline attaining a too-low background rejection rate. The CAE, then, performs slightly better than both the CoDAE and CoDVAE. This is expected since optimizing the model is easier: it can be seen as a simplified CoDAE lacking the spatial conditioning mechanism, although losing its advantages.

Discriminator	SUEP (GeV)							SVJ (TeV)			
	125	200	300	400	700	1000	mAUC	2.1	3.1	4.1	mAUC
Energy (Trk)	45.2	48.4	54.1	58.5	67.4	70.9	57.4	55.9	72.1	81.5	69.8
nTracks	78.7	92.4	98.3	99.6	99.9	99.9	94.8	82.1	89.3	92.9	88.1
CCT [243]	89.7	96.6	98.9	99.4	99.9	99.9	97.4	96.1	98.2	98.8	97.7
*CoDAE (Z_1)	69.8	81	83.8	82	77.9	77.6	78.7	73.8	77.6	79.3	76.9
Z_2	79.8	93	98.3	99.5	99.5	99.2	94.9	83.2	88.3	90.8	87.4
Dice	89.9	97.8	99.7	99.9	99.9	99.9	97.9	84.8	89.1	91.5	88.5
BCE	88.7	97.5	99.6	99.9	99.9	99.9	97.6	86	90.4	92.6	89.7
SSE	88.9	97.6	99.6	99.9	99.9	99.9	97.7	85.7	90.1	92.4	89.4
PixelSum	79.1	92.9	98.4	99.5	98.9	97.3	94.3	83.1	88.3	90.9	87.5
SAE	88.7	97.5	99.6	99.9	99.9	99.9	97.6	86	90.4	92.6	89.7
CoDVAE (Z_1)	77.5	91.5	97.7	99.2	99.4	99.2	94.1	81.4	86.4	88.9	85.6
Z_2	22.8	8.2	1.9	0.5	0.2	0.3	5.6	17.3	12.2	9.7	13.1
KL-R	65.9	80	90.6	95	97.8	98.1	87.9	76.8	81.3	83.4	80.5
KL-F	69.3	83.9	93.2	96.4	98	98.2	89.8	79.7	84.1	86.1	83.3
Dice	87.3	96.9	99.5	99.9	99.9	99.9	97.3	84.1	88.3	90.6	87.6
BCE	86.9	97	99.5	99.9	99.9	99.9	97.2	85.3	89.8	92	89
SSE	86.9	97	99.5	99.9	99.9	99.9	97.2	85.3	89.8	92	89
PixelSum	78.1	90.5	96.4	98	96.59	93.5	92.2	82.2	87.7	90.4	86.7
SAE	63.3	87.6	98.4	99.8	99.9	99.9	91.5	70.3	76.3	80.8	75.8
*CAE (Dice)	90.4	98	99.7	99.9	99.9	99.9	98	84.4	88.9	91.3	88.2
BCE	89.4	97.8	99.7	99.9	99.9	99.9	97.8	86	90.5	92.7	89.7
SSE	89.5	97.8	99.7	99.9	99.9	99.9	97.8	85.6	90.1	92.5	89.4
PixelSum	79.1	92.9	98.4	99.6	99.9	99.9	95	83.2	88.3	90.9	87.5
SAE	89.4	97.8	99.7	99.9	99.9	99.9	97.8	86	90.5	92.7	89.7
AE [85]-like (Dice)	33	37.3	40.6	42.5	40.5	36.1	38.4	49.4	39.4	32.2	40.4
BCE	77.9	92.2	98.2	99.6	99.9	99.9	94.6	82.7	87.2	89.4	86.5
SSE	75.3	91.1	98.1	99.6	99.9	99.9	94	82.1	86.6	88.9	85.9
PixelSum	83.9	94.6	98.7	99.7	99.9	99.9	96.1	83.3	88.7	91.4	87.8
SAE	77.9	92.2	98.2	99.6	99.9	99.9	94.6	82.7	87.2	89.4	86.5
DVAE [86] (Z_1)	52.9	53.1	52.7	51.8	49.8	49.1	51.6	45.5	44.8	44.7	45
Z_2	51.9	55	59.6	63.2	71.2	74.5	62.6	63.5	67.4	69.3	66.7
Z_3	51.7	51	49.7	49	47.1	46.6	49.2	46.6	46.1	45.3	46

Table 3.4: Comparison of baselines and models on test-set. AUROC metric, higher is better. Entries associated with an (*) were averaged over three random seeds. Top three best results in boldface.

Discriminator	SUEP (GeV)							SVJ (TeV)			
	125	200	300	400	700	1000	mFPR	2.1	3.1	4.1	mFPR
Energy (Trk)	50.63	48.11	41.81	37.71	29.3	25.11	38.78	33.84	17.46	9.16	20.15
nTracks	11.84	2.85	0.28	0.03	~ 0	~ 0	2.5	5.45	1.6	0.54	2.53
CCT [243]	0.24	0.02	~ 0	~ 0	~ 0	~ 0	0.04	0.12	0.02	0.01	0.05
*CoDAE (Z_1)	20.38	13.43	13.57	16.57	20.73	20.51	17.53	15.49	13.43	13	14
Z_2	11.44	2.68	0.35	0.11	0.24	0.43	2.54	4.65	2.01	1.17	2.61
Dice	4.15	0.47	0.04	0.01	~ 0	~ 0	0.78	3.11	1.27	0.66	1.68
BCE	5.83	0.81	0.06	~ 0	~ 0	~ 0	1.12	2.83	1.03	0.52	1.46
SSE	5.67	0.79	0.06	0.01	~ 0	~ 0	1.09	2.9	1.07	0.54	1.5
PixelSum	10.97	2.37	0.26	0.08	0.3	0.79	2.46	4.39	1.77	0.94	2.37
SAE	5.83	0.81	0.06	~ 0	~ 0	~ 0	1.12	2.83	1.03	0.52	1.46
CoDVAE (Z_1)	12.91	3.33	0.58	0.22	0.23	0.34	2.93	5.9	2.87	1.85	3.54
Z_2	76.85	93.59	99.27	99.87	99.89	99.77	94.87	86.57	92.91	95.39	91.62
KL-R	21.76	9.61	3.02	1.22	0.18	0.1	5.98	9.7	6.6	5.54	7.28
KL-F	19.08	7.5	2.64	1.45	0.74	0.68	5.35	7.43	4.91	4.09	5.47
Dice	5.25	0.6	0.03	~ 0	~ 0	~ 0	0.98	3.46	1.5	0.81	1.92
BCE	6.66	0.95	0.07	0.01	~ 0	~ 0	1.28	3.07	1.18	0.59	1.61
SSE	6.65	0.94	0.07	0.01	~ 0	~ 0	1.28	3.07	1.18	0.59	1.61
PixelSum	12.01	3.85	1.15	0.77	1.69	3.19	3.77	5.53	2.46	1.45	3.15
SAE	13.5	1.06	0.05	~ 0	~ 0	~ 0	2.44	6.01	1.87	0.76	2.88
*CAE (Dice)	3.75	0.38	0.03	~ 0	~ 0	~ 0	0.69	3.21	1.27	0.64	1.7
BCE	5.18	0.65	0.04	~ 0	~ 0	~ 0	0.98	2.694	0.94	0.46	1.37
SSE	5.11	0.68	0.05	0.01	~ 0	~ 0	0.97	2.81	1	0.5	1.44
PixelSum	10.94	2.25	0.15	0.01	~ 0	~ 0	2.22	4.27	1.63	0.81	2.24
SAE	5.174	0.655	0.04	0.002	~ 0	~ 0	0.98	2.693	0.944	0.463	1.37
AE (Dice)	63.79	57.04	52.05	49.48	52.99	59.42	55.8	41.47	56.06	66.91	54.82
BCE	11.99	2.55	0.21	0.02	~ 0	~ 0	2.46	4.61	2.085	1.23	2.64
SSE	13.48	2.86	0.23	0.01	~ 0	~ 0	2.76	5.01	2.28	1.29	2.86
PixelSum	8.22	1.9	0.21	0.04	~ 0	~ 0	1.73	4.35	1.74	0.86	2.32
SAE	11.99	2.55	0.21	0.02	~ 0	~ 0	2.46	4.61	2.09	1.23	2.64
DVAE (Z_1)	25.5	21.03	15.29	13.72	11.2	11.12	16.31	44.39	44.72	44.9	44.67
Z_2	26.06	17.38	10.15	6.66	2.572	1.42	10.71	21.42	14.89	10.92	15.74
Z_3	28.3	24.23	19.91	17.12	14.87	14.23	19.78	41.6	42.3	43.12	42.34

Table 3.5: Anomaly detection results on test-set. FPR40 metric, lower is better. Entries associated with an (*) were averaged over three random seeds. Top three best results in boldface.

Lastly, in table 3.6 we summarize the results of the ASO test between our models against the baselines, considering the AUROC metric as score distribution for the statistical test, which is performed between two distributions at a time. As we can see, our models (in particular, when using the **BCE** and **SSE** scores) can beat most of the baselines and for the SUEP signal also the supervised classifier, except for the SUEP(125 GeV) benchmark point. Furthermore, this test confirms the superiority of the CAE model over the CoDVAE but not against the CoDAE: the violation ratio fluctuates around 0.5, making it difficult to decide which model is the best. This also implies that the CAE is a valid alternative to our dual-encoder models, especially in settings where the auxiliary latent space has low discriminatory power.

*CoDAE (Z_2)									
Baseline	SUEP (GeV)						SVJ (TeV)		
	125	200	300	400	700	1000	2.1	3.1	4.1
*CAE (BCE)	0.99	0.99	0.99	0.99	0.99	0.98	0.99	0.99	0.99
AE (PixelSum)	0.99	0.99	0.99	0.34	0.99	0.57	0.7	0.99	0.99
nTracks	0.66	0.65	0.99	0.99	0.99	0.99	0.33	0.99	0.99
Supervised CCT	0.99	0.99	0.99	0.45	0.69	1	0.99	0.99	0.99
*CoDAE (BCE)									
*CAE (BCE)	0.55	0.55	0.55	0.55	0.53	0.65	0.43	0.54	0.39
AE (PixelSum)	0	0	0	0	0	0	0	0	~ 0
nTracks	0	0	0	0	0	0	0	~ 0	0.99
Supervised CCT	0	0	0	0	0	0	0.99	0.99	0.99
CoDVAE (Z_1)									
*CAE (BCE)	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
AE (PixelSum)	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
nTracks	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
Supervised CCT	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
CoDVAE (SSE)									
*CAE (BCE)	0.99	0.99	0.99	1	0.99	0.99	1	1	1
AE (PixelSum)	0	0	0	0	0	0	0	0	0
nTracks	0	0	0	0	0	0	0	0	0.99
Supervised CCT	0.99	0	0	0	0	0	0.99	0.99	0.99

Table 3.6: Pairwise comparisons of models and baselines, performing the ASO test based on the AUROC with a 95% confidence level: the headings denote the model and the anomaly score, which are compared to the baselines (the entries). Models and baselines denoted with a (*) were run on three random seeds. The smaller the value, the better the model compares against the respective baseline. Note: the listed values are the result of an approximate calculation, therefore there may be either false positives or negatives.

3.5.7 Conclusions and outlook

Compared to related analyses [231, 230], we have demonstrated the first successful application of (variational) auto-encoders to be deployed in the real-time event-triggering stages of experiments like ATLAS [229] and CMS [43], to search for two dark showers models: SUEPs and SVJs. Their discovery can potentially shed new light on the existence of dark matter and novel hidden sectors, which are currently uncovered and undercover at the LHC.

We have followed a model-independent setup in which the CoDAE models are trained to learn the QCD background, thus, without any explicit knowledge about the searched signals. Compared to physics-inspired baselines (like the number of tracks) and related approaches (e.g., [85], [86]), the proposed models achieve higher background rejection rates meaning that the classified signal is less contaminated with false positives, which, in turn, boosts the statistical significance of actually finding a newly theorized physics signal when analyzing the real data collected during the LHC's runs. For the SUEP signals, we can even equal the model-dependent classifier even if this model still achieves largely superior rejection rates at the target signal efficiency.

Unlike the common trend in many related works [85, 180, 86, 131, 178], we do not employ a specific particle-based preprocessing of our data, avoiding preprocessing-specific performance [87], nor hand-made low- or high-level features, but instead learn directly from raw images of particle signatures obtained by discretizing the detector response, thus reducing the dependency on the physics model by only assuming tracking information to be relevant for the considered signals: although the calorimeter information is discarded, the ECAL and HCAL channels are still expected to be generally useful, e.g., for searching long-lived particles. Learning from raw images can potentially simplify the whole analysis setup.

In addition, our dual encoder architecture, which also learns a smaller, auxiliary encoder, f_m , is particularly suitable for fast AD: the f_m model can score novel events in just 3ms, enabling more than real-time applications at the HLT, however at the cost of sacrificing accuracy; in principle, such a model can be further optimized to match even tighter latency requirements, for example deploying it on FPGA hardware like done in [192, 202]. Moreover, even the whole model can be easily optimized, for example, through quantization, to reduce the inference latency and resource usage.

Our models are evaluated against both signals, demonstrating anomaly scores that can identify both. The CoDAE models aim to adapt to the background samples, potentially allowing us to capture whatever novel signal that is diverse from what the model learned about the QCD background. Ideally, training a single model to reject one or more background processes would be possible, filtering only the events that resemble potential new physics signals. The approach described in this thesis chapter can potentially enable generic physics searches for unknown, new signals from raw images only and with little or no assumptions about the physics model. Further research, improved models, better anomaly scores, and larger benchmark datasets would be required to accomplish such an important goal. In principle, a single auto-encoder model can be trained on a big enough dataset comprising a sufficiently large variety of background processes to enable it to understand SM

physics. Such a model can be trained on unprocessed detector images, eventually equipping it with knowledge about the detector’s geometry and relevant SM physics variables through inductive biases as seen in the previous chapter. Then, a large test dataset of simulated signals should be employed to assess the model’s capability of generalizing toward the new physics without targeting any specific BSM model. The anomaly scores are often hand-designed from the latent space vectors or the reconstructed samples. This implies that the model is trained to minimize a different objective, rather than maximizing the sensitivity w.r.t. the anomalous features: this is an important point to address.

Furthermore, we may think of directly training an AE-based model on the real data, which can be seen as an unknown mixture of background and potentially new signals. In this regard, this may represent the ultimate, most complex learning problem as the model has to solve a truly *unsupervised* anomaly detection task, in which it has to autonomously find a set of features that is sensitive to the anomalies but not to the normal, expected data. Such a challenge is exacerbated by various factors: a huge *class imbalance* between signal and background since almost the totality of the data is expected to be background; *data scarcity* since the data that can be collected is way limited compared to the simulations that can be endlessly performed; and *robustness* to noise, sensor faults, and systematic uncertainties, which may alter the way predictions are interpreted in the presence of either noise, imprecise measurement or object reconstruction, or an erroneous physics model.

3.6 Summary

In this thesis chapter, we discussed model-independent new physics searches implemented by anomaly detection, as an alternative to signal-dependent ones. The main advantage of these kinds of searches is the minimal or absent reliance on the signal model, allowing them, in principle, to be sensitive to a wider variety of BSM signals. This enables them to test multiple new physics hypotheses in a single run. Then, we have seen two anomaly detection applications: the JointVAE, able to learn a joint latent space of continuous and discrete features to enhance its sensitivity to anomalies, and the CoDAE, which enables the learning from raw, sparse images of particle collisions. Combining the KL divergences term of the two latent spaces, the JointVAE can improve its discrimination performance, enabling its encoder to target a fast inference scenario on a FPGA accelerator. Instead, the CoDAE, thanks to its two encoding paths, can capture both the detail and discriminative features, allowing its auxiliary encoder to be deployed for real-time anomaly detection. Moreover, this model, among the related and compared ones, achieves the best discrimination performance on the two considered hidden valley models by only assuming the signals to be sensitive to tracking information, which is implicitly learned from the mask images.

Chapter 4

Deep Reinforcement Learning for Particle Physics

Deep Reinforcement Learning (DRL) [250], i.e., the combination of deep learning with Reinforcement Learning (RL) [251], is becoming increasingly popular since the world-renowned match of the game of Go between the world champion Lee Sedol and the AlphaGo agent, which sensationally won over the human player [252]. Since then, both RL and DRL are employed to solve complex control [253] and robotic tasks [254], autonomous driving [255], high-dimensional strategic games [256], prohibitive combinatorial problems [257], the discovery of new algorithms [258], and much more: refer to [259] for a more comprehensive listing of RL applications.

Reinforcement learning is a powerful learning paradigm that complements supervised and unsupervised learning. It allows us to solve sequential decision-making problems for which there is no or weak supervision. In this chapter, modern reinforcement learning is first introduced. Then, some notable applications in the context of HEP and particle physics are described, which are expected to indirectly improve the sensitivity to the new physics of downstream methods. Lastly, the chapter concludes by discussing the potentialities and limitations of RL in this field.

4.1 Introduction to reinforcement learning

Reinforcement learning is a learning paradigm at the intersection of multiple fields like computer science, psychology, neuroscience, mathematics, economics, and engineering, devoted to the solution of *decision-making* problems. RL provides a formalism for learning behavior in which a software or physical *agent* (e.g., a bot, a robot, or self-driving car) learns to take optimal *actions* within an *environment* (i.e., a real or simulated world) by trial and error, guided only by positives or negatives scalar *reward*¹ signals. Board games, video games, driving simulators, the stock market, a facility, and so on are all examples of environments

¹The rewards the agent receives are also called *reinforcements* [260], thus the name reinforcement learning.

in the context of RL; the respective actions may be: the new position of the piece on the board, an integer telling which button to press, the amount of steering, brake and acceleration (provided as a 3-dimensional vector), whether to sell or not (i.e., a boolean action), or even a high-dimensional tensor describing how the components of a complex system should interact together. After applying the provided action, the environment updates its (typically hidden) internal state to compute the reward, which can be: either one, zero, or minus one (only provided to the agent when the match is won, there is a tie, or it is lost; this kind of reward is said to be *delayed*), the real-time score of the game (an example of a *dense* reward), the result of a formula that penalizes collisions, fuel consumption and travel time (can be seen as the satisfaction of *multiple objectives*), the monetary gain or loss (this can be evaluated on the long term too), and the result of a complex formula relating the safety of the operation and the yield of the production for example.

The above examples describe classes of problems that are difficult to solve with supervised or unsupervised learning. In particular, a decision-making problem usually exhibits the following features:

- *Implicit supervision*: since the optimal actions (i.e., the supervised targets) are usually not known in advance, the agent can only learn them by trial and error, guided by a reward signal which provides *indirect* supervision. Moreover, even if we collect good enough actions for a given situation and train an agent by supervised learning, this will likely fail when facing unknown scenarios because the learned behavior is just a weak mapping lacking robust generalization.
- *Delayed feedback*: the reward, which only evaluates the instantaneous goodness of a past decision made by the agent, regarding the predicted action at time t , is usually not enough to understand if that decision was right or not. It can happen to receive many positive rewards that end with a catastrophic one, concluding that the decision at time t was poor: so, actions can have long-term consequences. If so the feedback is said to be delayed. This implies that *greedy policies*, i.e., a policy that maximizes the instantaneous reward, is not advisable.
- *Correlated data*: the mechanisms that power both machine and deep learning are based on the statistical assumption for which the data is identically and independently distributed (i.i.d.) but in *sequential* decision-making problems, the data (e.g., image frames, time-series) are naturally correlated in time: a decision made at time $t + 1$ is highly correlated with the decision made in the past at time t . Therefore, the i.i.d. assumption is broken and both ML and DL methods are not ensured to work properly.
- *Non stationarity*: during learning, the agent's actions usually affect the state of the environment which, in turn, can affect the future distribution of the data the agent would experience. Such a distribution can be non-stationary, introducing a distribution shift over time.
- *Reward hypothesis*: in RL, a learning task is solvable if it can be expressed as the

maximization of rewards. If the task cannot be solved by learning to maximize the rewards, then RL won't work. This is a critical assumption.

As a rule of thumb: if your problem exhibits one or more of the above characteristics (including the reward hypothesis), then reinforcement learning can solve it.

4.1.1 Markov decision processes

A Markov Decision Process (MDP) [261], is a five-element tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$ describing a learning problem associated to the underlying environment, in which:

- $\mathcal{S}$ is the *state space*, defining the input data (e.g., features, images) the agent could experience while interacting with the environment. States can be discrete or continuous, scalar or multi-dimensional. In general, even if the state space is enumerable, the agent does not know all possible states in advance, which incrementally discovers (a portion of) them. When considering a time index, t , also called a *timestep*, the states are usually denoted as s_t being the state yield by the environment at time t : otherwise, s denotes a generic state. In general, MDPs define *fully observable* environments in which the current state, s_t , completely characterizes the internal environment state. Conversely, if the environment's internal state is partially hidden from the agent, it receives an observation, $o \in \mathcal{O}$, rather than a state. From a given observation, the agent may extract the minimal, non-redundant, state representation². Moreover, the state space only contains *Markov* states. A state, s_t , is Markov if holds

$$p(s_{t+1} | s_t) = p(s_{t+1} | \underbrace{s_1, \dots, s_t}_{\text{history}}). \quad (4.1.1)$$

This is known as the *Markov property*, stating that the future state, s_{t+1} , is *conditionally independent* of the past (i.e., the history: $s_1, \dots, s_{t-1}$) given the present state, s_t . This property is important for algorithmic efficiency since, to predict the action for the next state, we only need to feed the present state to the agent and not the entire history of past states. Learning is more efficient if it happens from states rather than observations since these are not Markovian.

- $\mathcal{A}$ denotes the *action space*, which specifies the actions, a_t , the agent is allowed to draw. Actions can be binary, discrete, or continuous.
- $\mathcal{P}$ indicates the *transition model* (or *environment dynamics*), a structure (e.g., a 3d array) that specifies the probability of a state transition, i.e., $p(s_{t+1} | s_t, a_t)$, for each possible state ($s \in \mathcal{S}$) and action ($a \in \mathcal{A}$). If the transition model is not known nor provided, we are in the *model-free* setting which is the most common. Instead, if $\mathcal{P}$ is either given or learned, the *model-based* agent can use the environment's model to plan future actions without actually interacting with it.

²Full observability implies that $o_t = s_t$, which equals the environment's internal state. This is the default assumption in MDPs. On the other hand, if $o_t \neq s_t$, the agent only indirectly observes the environment: if so, we have a Partially Observable MDP (POMDP) [262].

- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the *reward function*, which provides the agent an *immediate reward*, $r_t = r(s_t, a_t, s_{t+1})$ at timestep t . This is the most general formulation in which also the next state is taken into account, otherwise, it can be simplified to $r(s_t, a_t)$. In general, rewards can be *sparse* (i.e., either zero or one), *noisy* (e.g., imprecise), and even *stochastic*.
- $\gamma \in [0, 1]$ is the *discount factor* that “discounts” the value of rewards as the time progresses by a factor of γ^t . Discounting is also mandatory for *continuing* environments, i.e., environments where the interaction does not normally terminate, to prevent the sum of rewards from diverging. Moreover, rewards obtained after $t \approx 1/(1 - \gamma)$ timesteps, called the *discounted horizon*, tend to be approximately zero. So, the contribution of rewards received after that time is negligible.

In terms of the MDP language, an environment can be described as $\langle \mathcal{S}, \mathcal{A}, \mathcal{P} \rangle$ and so by changing one of its components (e.g., the state or action space, or even the probability of the next state to occur) we are presenting the agent with a different environment. Moreover, even if we keep the environment fixed, a change in the reward function results in a different learning problem.

4.1.2 The reinforcement learning objective

Given an MDP, the agent interacts with it. One step of interaction results in a *transition* (s_t, a_t, s_{t+1}, r_t) , i.e., a change of state due to the application of the predicted action, a_t , which results in a new state, s_{t+1} , according to the transition model $\mathcal{P}$. Consequently, the new state is evaluated by the reward function providing the immediate reward, r_t . A collection of transitions, resulting from multiple sequential interactions, is called a *trajectory*: $\tau = \{(s_t, a_t, s_{t+1}, r_t)\}_{t=1}^T$. The size of the trajectory, T , is called the *horizon* (or episode length) representing the maximum number of timesteps after which the episode terminates. For continuing environments, $T = \infty$, therefore the interaction only terminates when a *terminal state* is reached³: e.g., the match is over, or the vehicle has reached the target destination.

In general, a RL agent may include one or more of these components:

- *Policy*: it represents the agent’s behavior function, allowing it to decide (predict) which action is best to take in a given state. The policy maps states to actions, i.e., $\pi : \mathcal{S} \rightarrow \mathcal{A}$, being either stochastic or deterministic. A *stochastic* policy, denoted as $\pi(a \mid s)$, is a probability distribution over actions conditioned on states, representing the likelihood of a given s . According to the action space, we can either opt for a *Bernoulli* distribution when there are two binary actions, a *Categorical* distribution over discrete, enumerable, and finite actions, a *Gaussian* for continuous and unbounded action spaces even with multiple actions, and a *Beta* [263] or *Squashed*

³In practice, we can set a maximum number of timesteps or learn by considering “chunks” of T trajectories at a time. Nevertheless, early episode termination or truncation should be properly handled.

*Gaussian*⁴ for bounded, continuous actions. According to the specific application, by taking the mean or mode of such distributions we can obtain a *deterministic* policy, often indicated as $a = \mu(s)$, which always outputs the same action given a state. The deterministic mapping, $\mu(s)$, can also be directly learned.

- *Value function(s)*: there are two kinds of them. The *state-value function*, $V(s)$, evaluates how good transitioning to a given state will be. Instead, the *action-value function* (or Q-function), $Q(s, a)$, provides how valuable taking action, a , given the state, s , is. The former evaluates the goodness of states, while the latter quantifies the goodness of actions. An agent can have one or more value functions to improve or derive a policy. These value functions are represented by tables or linear functions, in classical RL [251], or by neural networks, in deep RL.
- *Model*: it depicts the agent's representation of the environment. The environment model can be given or learned. If learned it represents an approximation of the true transition model, so $p(s_{t+1}, r_t | s_t, a_t) \approx \mathcal{P}$: the learned model can eventually include a *reward model* that predicts the next reward. The agent can use the environment model to simulate new experiences without interacting with the environment, plan the next action, and improve its components.
- *Buffer*: the agent may keep a temporary memory, in which relevant or recent transitions are stored. The memory or buffer, can be later used to improve the state-value or action-value function estimates or the environment model.

The agent aims to learn a policy that maximizes the sum of rewards over timesteps. Formally, the goal of RL is to maximize the *return*, G :

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \quad (4.1.2)$$

where G_t is the total discounted reward from timestep, t . In practice, the summation rather than being infinite is limited to a maximum of T steps: in this way, we can handle both non-terminating (i.e., continuing) and episodic (i.e., terminating) environments. The return can capture the long-term effects of the reward, a thing that the instantaneous reward, r_t , cannot capture. The discount factor, then, can alter such notion: $\gamma \rightarrow 0$ leads to *myopic evaluation*, in which short-term rewards are preferred, instead, if $\gamma \rightarrow 1$ long-term rewards are favored by the *far-sighted evaluation*. Furthermore, discounting the rewards is an elegant way to: reduce the uncertainty of future interactions, prevent infinite returns, and even express a preference over immediate rewards.

To maximize the return, the agent has to interact with the environment to learn the optimal policy that solves the MDP. In DRL, the policy is a neural network therefore the

⁴The action is first sampled from a Gaussian, and then constrained to $[-1, 1]$ by a tanh operation: the likelihood is adjusted accordingly. This was proposed in [264].

*reinforcement learning objective*⁵ is to find the optimal parameters of the policy such that to maximize the *expected* return over all possible trajectories, τ :

$$\theta^* = \arg \max_{\theta \in \Theta} \mathbb{E}_{s_t, a_t \sim p_\theta} \left[\underbrace{\sum_{t=1}^T \gamma^{t-1} r(s_t, a_t)}_{= G_1} \right], \quad (4.1.3)$$

where Θ is the space of all possible neural network parameters, which is usually searched by gradient ascent, and p_θ is the *induced trajectory probability* that provides the likelihood of a trajectory

$$p_\theta(\tau) = p(s_1) \prod_{t=1}^T p(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t \mid s_t) \quad (4.1.4)$$

according to the *initial state distribution*, $p(s_1)$, which provides the probability of sampling a given initial state from $\mathcal{S}$, the environment dynamics, and agent policy: the length of trajectories, τ , is assumed to be at most T . As we can see, eq. 4.1.3 aims to search for a policy whose actions lead to the maximization of the return but over all possible trajectories, i.e., by averaging over all possible sequences of interaction with the MDP, such that to ensure optimal decisions in any situation. From the RL objective, it can be derived a *performance measure*, $J(\theta) = \mathbb{E}_{s_t, a_t \sim p_\theta} [\sum_t r(s_t, a_t)]$, which is used to evaluate the performance of an agent with a given set of parameters, θ .

4.1.3 Learning the optimal policy

The RL objective (eq. 4.1.3) cannot be maximized directly, therefore we cannot learn the parameters, θ , of the optimal policy in a differentiable way since there is no dependence between the rewards and θ . Alternative approaches exist that either maximize a related (or surrogate) objective or the same objective but indirectly. These methods can be divided into two main categories: Q-learning, and policy gradient.

Deep Q-Learning. The aim of Q-learning [265] is to learn the optimal action-value function, $Q^*(s, a)$, and then derive a consequently optimal policy from it: this is possible because the Q-function allows us to understand which action achieves the highest return in a given state. Given a Q-function, $Q(s, a)$, it is possible to derive either deterministic or stochastic policies as follows:

$$\mu(s) = \arg \max_{a \in \mathcal{A}} Q(s, a), \quad (4.1.5)$$

where $\mu(s)$ is the *greedy* policy, which deterministically outputs the action associated with the largest Q-value. This can also be seen as a stochastic policy that assigns probability one to the optimal action, say a^* , and zero otherwise. The greedy policy is said to always

⁵For finite horizon problems, the objective in eq. 4.1.3 works even without discounting. Instead, for infinite horizon tasks it is possible to truncate the trajectory at a predefined value.

exploit: it selects only the best actions, discarding sub-optimal ones. A policy that always exploits is not favorable while learning since it can trap the agent into a local minimum as the *exploration* of diverse actions is not ensured, which causes an imprecise estimation of the action value. In this context, a way to force exploration is to “sometimes” act randomly by selecting a random action. Such a policy is called *epsilon-greedy*, since the ratio of exploration is controlled by setting the $\epsilon \in [0, 1]$ hyperparameter:

$$\pi_\epsilon(a | s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a = \arg \max_{a' \in \mathcal{A}} Q(s, a') \\ \epsilon/m & \text{otherwise} \end{cases}. \quad (4.1.6)$$

The ϵ -greedy policy will select a random action with probability ϵ and, since actions are sampled uniformly, each action, a , has a chance of ϵ/m (where $m = |\mathcal{A}|$ is the number of discrete actions) to be drawn. Furthermore, the random action can still be the greedy (i.e., optimal) one: this implies that the best action is selected with probability $\epsilon/m + 1 - \epsilon$ rather than $1 - \epsilon$. Alternatively, it is possible to bias the sampling of actions in proportion to their value:

$$\pi(a | s) \propto \exp(Q(s, a)). \quad (4.1.7)$$

This is the *Boltzmann policy* in which the probability of sampling an action is proportional to its action value: the resulting policy may be sensitive to imprecise estimation of the Q-values, due to the exponential function that enlarges even small differences in value.

To estimate the optimal Q-function, it is possible to set up a *supervised* learning problem to regress the action-value targets, y , minimizing a squared loss. In practice, during learning, the agent interacts with the environment multiple times each called an *episode*⁶. Each episode (or *rollout*) results in a trajectory. Since the transitions within a trajectory are all correlated, being not i.i.d., each is stored in a *replay memory* (i.e., a buffer) to stabilize training. Once the memory is full, old transitions are evicted. A batch of transitions, $\mathcal{B}$, is then sampled from the buffer when computing the regression targets:

$$y_i = r_i + \gamma \max_{a' \in \mathcal{A}} Q_{\phi'}(s'_i, a') \quad \forall (s_i, a_i, s'_i, r_i) \in \mathcal{B}. \quad (4.1.8)$$

Since the Q-network⁷, Q_ϕ , would be involved in the computation of y_i , resulting in *moving targets* as soon as the parameters ϕ got updated, therefore making y_i quite noisy, a copy of the Q-function, $Q_{\phi'}$ with parameters ϕ' , called a *target network*, is used instead. The role of the target network is to stabilize the value of the targets since the parameters ϕ' are updating less often than ϕ , hence the target remains stable across multiple updates of the main parameters, ϕ . The overall algorithm is described in listing 3, taking the name of Deep Q-Network (DQN) [266]: notice we assume the environment to have `reset()` and `step(a)` methods, in which the former restore its internal state returning an initial state sampled from the initial state distribution, and the latter updates its internal state by applying the given action.

⁶In DL terminology, an episode is the rough equivalent of an epoch.

⁷A Q-function approximated by a neural network, i.e., $Q_\phi \approx Q$, is usually called a Q-network.

Algorithm 3: DQN

```

Allocate replay memory  $\mathcal{D}$  with capacity  $N$ , and batch size  $B$ 
Initialize  $Q_\phi$  with random weights, and make a copy  $\phi' = \phi$ 
Assume discount factor  $\gamma$ , learning rate  $\eta$ , epsilon  $\epsilon$ , and target network update
frequency  $K$ 
Instantiate the environment (MDP),  $env$ 
1  $k = 0$  // init counter for target network
2 for episode,  $e = 1$  to  $E$  do
3    $s_1 = env.reset()$  // get initial state or observation
4   for timestep,  $t = 1$  to  $T$  do
5      $k \leftarrow k + 1$ 
6      $a_t \sim \pi_\epsilon(\cdot | s_t)$  // sample action from  $\epsilon$ -greedy policy
7      $s_{t+1}, r_t, d_t = env.step(a_t)$  // environment step, which applies  $a_t$ 
8     Store transition  $(s_t, a_t, r_t, s_{t+1}, d_t)$  in  $\mathcal{D}$  //  $d_t$  tells whether  $s_{t+1}$  is terminal
9     Sample mini-batch  $\mathcal{B} = \{(s_i, a_i, r_i, s'_i, d_i)\}_{i=1}^B$  from  $\mathcal{D}$ 
10    Compute targets:  $y_i = r_i + d_i \cdot \gamma \max_{a' \in \mathcal{A}} Q_{\phi'}(s'_i, a')$ ,  $\forall (s_i, a_i, r_i, s'_i, d_i) \in \mathcal{B}$ 
11    Update step:  $\phi \leftarrow \phi - \eta \cdot \nabla_{\phi} \frac{1}{B} \sum_{i \in \mathcal{B}} (Q_\phi(s_i, a_i) - y_i)^2$ 
12    if  $k \bmod K == 0$  then
13      Update target network:  $\phi' \leftarrow \phi$  // copy current parameters
14    if  $d_t$  is True then
15      Terminate episode

```

DQN is a simple yet powerful agent, capable of solving various games in the Atari learning environment (ALE) [267]. DQN can be applied to only discrete action spaces, even if a variation for continuous actions has been proposed [268]. DQN learns to take optimal actions in an *off-policy* way: this means that the transitions used to regress the Q-values are not required to be recent, moreover, it is also possible to use transitions obtained from a different policy, called the *behavioral policy* μ_b . Many improvements have been proposed to the DQN agent, such as reducing the action-value overestimation error [269], a tailored architecture for more efficient learning [270], improved experience replay mechanism [271, 272], better exploration strategies [273], and even learning from the empirical distribution of returns [274].

Policy gradient. Compared to Q-learning, policy gradient (PG) agents learn the policy explicitly by directly following the gradient of $J(\theta)$. To derive the policy gradient, we first need to write the expectation in the performance objective as an integral over trajectories:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta} [G(\tau)] = \int p_\theta(\tau) G(\tau) d\tau,$$

having defined $G(\tau) = \sum_t^T \gamma^{t-1} r(s_t, a_t)$ as the discounted return over a given trajectory. Next, we take the derivative of both sides:

$$\begin{aligned} \nabla_\theta J(\theta) &= \int \nabla_\theta p_\theta(\tau) G(\tau) d\tau && \text{(derivative)} \\ &= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) G(\tau) d\tau && \text{(likelihood ratio trick)} \\ &= \mathbb{E}_{\tau \sim p_\theta} [\nabla_\theta \log p_\theta(\tau) G(\tau)]. && \text{(integral as expectation)} \end{aligned}$$

At this point we obtain an expectation over the *score function*, $\nabla_\theta \log p_\theta(\tau)$, in which the derivative of the $\log p_\theta(\tau)$ term is:

$$\begin{aligned} \nabla_\theta \log p_\theta(\tau) &= \nabla_\theta \left[\log p(s_1) + \sum_{t=1}^T \log \pi_\theta(a_t | s_t) + \log p(s_{t+1} | s_t, a_t) \right] && \text{(log of eq. 4.1.4)} \\ &= \nabla_\theta \left[\cancel{\log p(s_1)} + \sum_{t=1}^T \log \pi_\theta(a_t | s_t) + \cancel{\log p(s_{t+1} | s_t, a_t)} \right] && \text{(no dependence on } \theta \text{)} \\ &= \nabla_\theta \sum_{t=1}^T \log \pi_\theta(a_t | s_t). \end{aligned}$$

Therefore, by plugging the above term in the previous equation we obtain the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta} \left[\sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_t | s_t) G_t \right], \quad (4.1.9)$$

also called the REINFORCE [275] gradient estimator, which is the derivative of the policy's log-likelihood weighted by the return⁸. The PG formalizes the notion of *trial and error learning* because the gradient tries to increase the likelihood of good actions whose return is high while decreasing the likelihood of bad actions achieving low rewards: in practice, this can be implemented as shown in listing 4.

PG agents are more theoretically grounded than Q-learning, and the learned policy can directly handle continuous or discrete actions without modifications. Since the learning is *on-policy*, i.e., the latest experience must be used to update the policy parameters, PG is expected to be less sample-efficient than Q-learning because the old transitions are discarded rather than reused. Another well-known issue of PG is its high-variance gradients, arising when the gradient is estimated from a few samples. This variance can be reduced by subtracting a *baseline function*, $b(s)$, such that the gradient estimation is still unbiased in expectation. A good baseline is the state-value function, $b(s) = V^{\pi_\theta}(s)$, computed considering the policy's predicted actions, which allows us to rewrite the policy gradient

⁸Equation 4.1.9 refers to the non-discounted case, i.e., $\gamma = 1$. For the general setting, the unbiased estimator of the policy gradient is instead: $\mathbb{E}_{\tau \sim p_\theta} \left[\sum_{t=1}^T \gamma^t \nabla_\theta \log \pi_\theta(a_t | s_t) G_t \right]$ [276].

Algorithm 4: REINFORCE

```

Initialize  $\pi_\theta$  with random weights
Assume discount factor  $\gamma$ , learning rate  $\eta$ , number of trajectories  $N$ , and horizon  $T$ 
Instantiate the environment (MDP),  $env$ 
1 for update step,  $k = 1$  to  $K$  do
2   Initialize buffer:  $\mathcal{D} = \{\}$ 
   /* Collect  $N$  trajectories */
3   for  $i = 1$  to  $N$  do
4      $\tau_i = \text{Rollout}(\pi_\theta)$  // Rollout yields a trajectory of maximum length  $T$ 
5     Compute returns:  $G_i = \{ \sum_{t'=t}^T \gamma^{t'} r(s_{t'}, a_{t'}) \mid \forall t = 1, \dots, T \mid \tau_i \}$ 
6     Store  $\tau_i$  and  $G_i$  in  $\mathcal{D}$ 
   /* Update the policy parameters by gradient ascent: Eq. 4.1.9 */
7    $\theta \leftarrow \theta + \alpha \cdot \frac{1}{|\mathcal{D}|} \sum_{\tau_i, G_i \in \mathcal{D}} \sum_{t=1}^{|\tau_i|} \gamma^t \nabla_\theta \log \pi_\theta(a_{i,t} \mid s_{i,t}) G_{i,t}$ 

```

using the *advantage function*, $A^{\pi_\theta}(s, a) = Q(s, a) - V^{\pi_\theta}(s)$, such that

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta} \left[\sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_t \mid s_t) A^{\pi_\theta}(s_t, a_t) \right]. \quad (4.1.10)$$

The above formulation of the PG achieves significantly lower variance compared to REINFORCE: this is thanks to the advantage function, which provides the difference between the current action value, $Q(s, a)$, and the value of the average action by $V^{\pi_\theta}(s)$. If this difference is positive, i.e., $Q(s, a) > V^{\pi_\theta}(s)$, the action a is better than average, otherwise if negative the action is worse than average: this formulation strengthens the interpretation of the PG. In this case, the value function is interpreted as the “value of the average action” because the following relationship with the Q-function holds: $V^{\pi_\theta}(s) = \mathbb{E}_{a \sim \pi_\theta(\cdot \mid s)} [Q(s, a)]$, being the average of action-values weighted by the policy’s likelihood of sampling that action.

Usually, the advantage function is estimated by introducing an additional network, the *critic* which learns either or both value functions by supervised regression. If we learn two critics, one to approximate $V(s)$ and another for estimating $Q(s, a)$, then the advantage can be calculated directly. Otherwise, if learning only the state-value function, i.e., $V_\phi \approx V$, the advantage can be computed as

$$A(s, a) = \underbrace{r(s, a) + V_\phi(s')}_{\approx Q(s, a)} - V_\phi(s), \quad (4.1.11)$$

although requiring the knowledge of the next state, s' , for which we compute its value: notice that the term, $r(s, a) + V_\phi(s')$, is an estimate of the action-value function; the value function can be learned by fitting the empirical returns, $\hat{G}_t$, computed over one or more trajectories. More powerful estimators for the advantage function are the *n-step returns* [277], and the *Generalized Advantage Estimation* (GAE) [278] which allows to trade-off

bias for variance. The PG formulation in eq. 4.1.10 that makes use of a critic is called the *actor-critic* (AC), which somehow unifies both Q-learning and REINFORCE: in this context, the policy is also named the *actor*.

In both PG and AC methods many improvements have been proposed, such as a *thrust region* [279, 280] to ensure monotonic improvements, gathering experience in parallel to either decorrelate the samples [277] or increase the sample efficiency [280], learning deterministic policies [281], and better exploration through entropy regularization [264].

4.1.4 Model-based RL

So far we have seen examples of model-free agents, i.e., learning algorithms that discover an optimal policy despite having no knowledge about the environment and its dynamics. This knowledge about the MDP is built implicitly by interacting with it, experiencing the states, and discovering the value of actions. Instead, in model-based RL (MBRL) [36] the agent can query the model of the environment without actually interacting with the MDP to gather more information about it, which can be used either to improve value estimates or build a policy. Having a model of the environment’s dynamics can be useful in multiple scenarios, for example: when the interaction with the real world is expensive or slow, to increase the data efficiency, to safely explore the state and action spaces, to transfer between similar dynamics, and to ensure safety in advance.

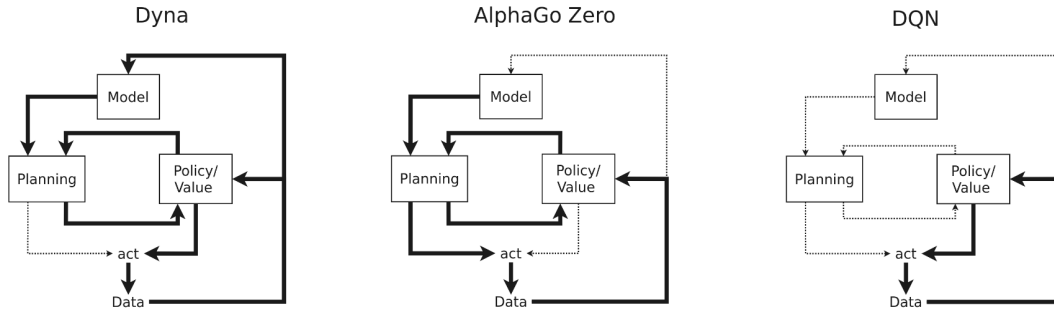


Figure 4.1: In model-based RL (first two figures), the learned or given model can be used to either plan or improve the value function(s). Instead, in model-free RL (last figure) we are forced to interact with the environment to get learning data. Figure adapted from [36].

Following [36], MBRL is a class of algorithms to solve MDPs that use either a given or learned model of the MDP’s dynamics to find a global solution. In MBRL, the learning can estimate the dynamics model, or find a global value or policy function. Moreover, the model provides a *reversible* access to the MDP dynamics, as shown in figure 4.1, allowing the agent to reason about the future without changing the current state: a capability that model-free RL lacks since a state transition occurring at each interaction step is *irreversible*.

Dynamics model learning. The environment’s dynamics model is usually learned from observed data, yielded by the actual interaction between the agent and environment. The

problem is supervised since we have both the next states and reward targets. Given a batch of one-step transitions, $\{(s_t, a_t, r_t, s_{t+1})_i\}_{i=1}^B$, there are three types of dynamics functions we might learn:

- *Forward dynamics*: $(s_t, a_t) \rightarrow s_{t+1}$ or $p(s_{t+1} | s_t, a_t)$. It predicts the next state given the current state and the devised action. Knowing the forward dynamics enables look-ahead planning.
- *Backward dynamics*: $s_{t+1} \rightarrow (s_t, a_t)$ or $p(s_t, a_t | s_{t+1})$. This model predicts which state is the possible precursor of a given state, allowing the agent to plan backward.
- *Inverse dynamics*: $(s_t, s_{t+1}) \rightarrow a_t$ or $p(a_t | s_t, s_{t+1})$. An inverse model predicts which action is needed to get from one state to another. This can be useful in representation learning, for example.

These models can be learned either exactly or approximately. If both states and actions are discrete, or their dimensionality is sufficiently low we can learn the forward dynamics, for example, by counting the number of times a transition (s, a, s') is observed:

$$P(s' | s, a) = \frac{n(s, a, s')}{\sum_{s' \in \mathcal{S}} n(s, a, s')}, \quad (4.1.12)$$

in which $P = \mathcal{P}$ in the limit of $n \rightarrow \infty$. Counting is implemented with a *table* indexed by the state, s , action, a , and next state, s' . Indeed, this approach suffers from the curse of dimensionality that prevents scaling it to more complex problems, therefore function approximation, e.g., neural networks, can be used as a more flexible alternative to tables. The basic approach is to learn a model, $p_\theta(s', r | s, a)$, by regressing both the next state and reward by using a dataset of trajectories, which can be collected by using the current, exploratory, or expert policy. In this regard, a popular approach is Dyna [282] in which the dynamics model is learned from real experience yield by the interaction of the policy with the environment. Then, the learned model generates simulated experiences that, together with the real data, are used to improve a value or policy.

World models [35], instead, make use of LSTMs [12] and VAEs [161] to deal with *partial observability* that can occur when some information is hidden to the agent. The VAE (defined in section 3.2.6) is responsible for extracting a useful, non-redundant representation from the observation, which can be regarded as a *state abstraction*, that is then fed to the recurrent network which, since sharing weights along the time dimension, can effectively approximate the history of past interaction potentially recovering the hidden information. The world model is used to generate the most likely next state, hence it can also deal with stochastic environments by predicting a distribution over states rather than a point estimate, which is fed to the controller, i.e., the policy that predicts the next action. A peculiar characteristic of world models is that a policy can be trained entirely from simulated experience, i.e., trajectories generated by the dynamics model. Nonetheless, this can be risky since the agent may learn to exploit approximation errors in the generative model

which does not generalize to the real world, yielding a policy that achieves lower return on the real environment.

Lastly, dynamics models can be either *local* or *global*. Local models are valid only on a specific subset of the state space. These are discarded after planning (because after the real step, the region of the state space will probably change) requiring training them multiple times. Instead, global models approximate the environment dynamics over the entire state space. In general, the reliability of dynamics models tends to decrease as we predict far in the future, therefore uncertainty estimation is an important topic in MBRL research [36]. Being able to estimate the *epistemic* uncertainty, i.e., uncertainty due to limited availability of training data is important to assess after how many steps the planning should stop because, afterward, the plan is not reliable enough [283]. Moreover, dynamics models can be designed to make long-term predictions, which are expected to be more useful when planning. One-step dynamics models, e.g., the one described in eq. 4.1.12, can only predict the next state: this can be generalized by predicting a sequence of transitions of size $n > 1$. Indeed, one-step models can handle multi-step predictions by feeding the previously predicted next state back to the learned model. However, doing so will result in a rapid accumulation of errors, causing the estimated states to deviate from the true dynamics.

Combining planning and learning. Having access to a given or learned environment’s model, allows the agent to predict the future after one or multiple steps. If we collect the experience generated by the dynamics model, real and simulated data can be used to train and improve the policy and value functions. But before doing so, there are multiple things to consider when planning with a dynamics model, such as:

1. *At which state the planning should start.* The most expensive choice is to start planning on every state of the state space, selecting them uniformly to cover the entire space. Otherwise, we can only consider states that can be reached by starting planning on previously visited states. Another option would be to order the states according to some criteria, starting to plan from the one with the highest priority. Probably the most common choice, which is also the least expensive, is to plan from the current state. After we get the plan, the real environment step would induce a sort of pruning of the state space because the resulting next state would exclude the other possible ones. Consequently, the new state should be considered the starting state for the next planning phase.
2. *How much planning budget to allocate.* Both the number of planning steps and how frequently we plan, determine the planning budget. Approaches like Dyna [282] and AlphaGo [252] plan for a hundred steps after every real environment step. Others, like PILCO [284], first collect data for a whole episode, then replan the entire trajectory. To determine the number of planning steps at each iteration we need to consider the cost of each planning step, which usually involves forwarding the dynamics model and, eventually, a Monte Carlo Tree Search (MCTS) [285] iteration. The planning budget introduces a trade-off between planning and real data collection, in which the

two extremes are model-free RL (no planning at all) and *exhaustive search* (infinite planning budget, required to cover all the possible trajectories): practical MBRL methods strive to find a good intermediate compromise.

3. *How to plan.* We distinguish between discrete and differentiable planning methods. Discrete planning, e.g., MCTS, is about making discrete back-ups (i.e., a search over enumerable states and actions) stored in a tree, or table, or used as training targets to improve a value or policy function. Differentiable planning (e.g., PILCO and Dreamer [286]), instead, exploits the fact that if both the dynamics model and the policy are differentiable, then it is possible to directly take the gradient of the cumulative reward objective w.r.t. the policy parameters. For example, Dreamer solves long-horizon control tasks [287] by latent imagination through a *recurrent latent dynamics model*: it uses representation learning [288, 289] to encode a dataset of experience (observations and actions) into compact latent codes, which are used to predict environment rewards and state-values. The actions are learned by propagating gradients back through imagined trajectories (i.e., sampled from the latent dynamics model), which is possible through stochastic back-propagation [161]. Moreover, the kind of dynamics model determines the plan strategy. In particular, the *direction* of planning can be forward or backward as the dynamics model: forward planning is a look-ahead search, whereas backward planning can help propagate information more quickly over the state space. In addition, planning algorithms can even balance *breath* and *depth* of the plan: by breath, we mean exploring the actions available from a state, in which from the same start state can yield many different trajectories; and by depth, the length of the considered trajectories. For example, a plan with a breath of one only considers a single action (i.e., an individual trajectory is considered). If the planning algorithms do a full-depth search, the trajectories are expanded until a terminal state is reached. For example, exhaustive search performs full-depth and full-breath searches together, which becomes computationally infeasible quickly, whereas dynamic programming looks for depth one, full-breath plans.

Planning and learning can both benefit from each other in multiple ways. For example, when planning we can use the estimated value of a state or state-action pair, or even the probability that the policy will select an action, to create confidence bounds that can guide the planning in deciding when it is best to stop. Moreover, the transitions gathered during planning can be used as training targets (or additional training data) to learn value functions. For example, the regression target can be the state or state-action value at the root of the planning search tree. It is also possible to learn a policy from planning: AlphaGo Zero [290] uses the visitation count as pseudo-class-probabilities to train a policy; Guided Policy Search [291] minimizes the KL divergence between the planned and predicted trajectories; lastly, Policy Gradient Search [292] uses values estimated from a search tree to update a policy. Furthermore, Model Predictive Control (MPC) [293] uses only planning to determine the best action: although expensive, MPC is robust to changing state and action spaces thanks to the adaptation brought by continuous planning.

In conclusion, model-based RL is preferable over model-free RL if we have access to the underlying dynamics of the MDP, e.g., a simulator (like CARLA [294]), an exact specification (like in Chess, or Go), or a dataset from which a dynamics model can be learned with sufficient fidelity. If so we can bypass the real environment and only query the model, potentially speeding up training, especially when a transition in the real world is either expensive or slow. For example, imagine a robot that has to perform some task: doing so in simulation is both safe (we do not risk damaging the robot itself and harming others) and fast (e.g., the software can be either accelerated or parallelized with multiple instances), instead we often lose these properties in the real world. Indeed, agents that have been trained within simulated environments should be adapted to act in the real world, otherwise, they may underperform due to either exploiting simulator defects or due to distribution shift. The techniques from the sub-field of *sim-to-real transfer* [295] aim at transferring the agent’s policy from simulation to the real-world domain while maintaining as much performance as possible.

4.1.5 Learning from demonstrations

In some specific cases, we cannot specify or build an environment, e.g., because it could be just too difficult, expensive, or infeasible, but, instead, have access to an *expert* either in the form of an already collected, fixed dataset or as a capable policy (sometimes even human-powered) with which we can collect a dataset of expert trajectories by often incurring in a non-negligible cost. In general, the expert could complete our target task with high performance but, for some reason, it is not widely applicable or costly⁹. Therefore, we want to train an agent to match or surpass the expert policy, generalizing across the various scenarios in our task. The main approaches to achieve this are: *imitation learning* (IL) [296, 297], *inverse reinforcement learning* (IRL) [298], and *offline reinforcement learning* (ORL) [299].

Imitation learning. Given a dataset of expert trajectories, i.e., $\mathcal{D} = \{(o_i, a_i)\}_{i=1}^N$, made of observations, o , and corresponding actions, a , taken by the expert, we can train an agent by plain supervised learning. If the actions are discrete, we minimize the cross-entropy between them and the predictions, otherwise, if continuous we minimize the MSE instead. This basic approach is called *behavioral cloning* (BC) [300], and its main limitation is the lack of robustness against accumulating errors due to shifts in the distribution of the experienced observations. Say the agent made a mistake (even small) at time t , so from that timestep and onward the agent will continue making mistakes since it will experience data that were not seen during training (but also due to noise and uncertainty in the observations), causing the predicted trajectory to diverge against the training one. For BC, the error accumulates quadratically with time. Therefore, approaches to mitigate this issue

⁹For example, human drivers are very competent so we can employ them to collect driving experience to train an autonomous driver. Ideally, this procedure can be applied to whatever number of vehicles and driving scenarios.

involve: applying data augmentation to learn a more robust representation [301], collecting more training data, and even using sub-optimal trajectories that show the agent how to recover from erroneous situations. Moreover, in *Dataset Aggregation* [302], a human is asked to label the actions predicted by the agent’s policy such that, if repeating this enough times, the policy can be adjusted to match the data distribution without actually collecting any new data. Indeed, the main limitation of this approach is that human feedback is involved in the training loop, which can also be imprecise: the whole process can be regarded as a form of *active learning* [303].

The main takeaway about IL is that it allows the agent to learn a policy by supervised learning: although problem-specific “tricks” must be introduced to deal with the robustness and stability issues. In some cases, IL is preferable over RL because it can achieve comparable performance with less data and effort.

Inverse RL. In many real-world problems (e.g., autonomous driving, accelerator control) it is difficult to define a reward function that fully captures the true complex behavior, constraints, and rules that underlie a task. IL avoids this issue by learning to “copy” the expert’s actions, without reasoning about their effect as RL does. Instead, IRL exploits expert demonstrations to recover the reward function the expert maximized. After learning that, an optimal policy can be discovered by subsequent model-free or model-based RL, or even directly through the IRL process.

Formally, given a dataset of states (or observations) and actions, we want to infer a parameterized reward function, $r_\psi(s, a)$, which can be a neural network or a (linear) combination of features for example, from which a policy, $\pi(a | s)$ is then learned. One issue with this formulation is that multiple reward functions can explain the same behavior, so the search space must be restricted to avoid ambiguity. One way to deal with this is to use the *maximum causal entropy* formulation [304], which assigns low cost to the expert policy and high cost to other policies such that the optimal policy can be found by standard reinforcement learning:

$$\text{RL}(c) = \arg \min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_\pi [c(s, a)], \quad (4.1.13)$$

where $H(\pi) = \mathbb{E}[-\log \pi(a | s)]$. The above formula maps a cost function, c , to high-entropy policies that minimize the expected cumulative cost. The cost function (i.e., the reward) is searched over a family of functions $\mathcal{C}$ (e.g., neural networks) by solving the following optimization problem:

$$\max_{c \in \mathcal{C}} \left(\min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_\pi [c(s, a)] \right) - \mathbb{E}_{\pi_E} [c(s, a)], \quad (4.1.14)$$

where π_E is a provided expert policy represented by a set of trajectories; it turns out the above problem can be seen as a two-player game like a GAN [120], in which the generator represents the policy we are learning, and the discriminator a binary classifier defined in terms of the reward or cost function whose role is to distinguish between expert and generated behavior. Generative adversarial imitation learning (GAIL) [305] implements this idea,

in which the discriminator, $D(s, a)$, is first trained to separate the samples of the current policy, π_θ , against the expert trajectories, to then take a policy improvement step by using $\log D(s, a)$ as cost function. The optimal policy is found when the discriminator cannot distinguish between expert and predicted trajectories. Hence, GAIL can simultaneously infer a reward function while training a policy, which can even surpass the expert.

Offline RL. Given a previously collected dataset of transitions, $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})^{(i)}\}_{i=1}^N$, which is kept fixed during the whole training since the agent has no longer the ability to both collect new data and interact with the environment, the goal of ORL is to extract a policy, $\pi(a | s)$, that demonstrates the largest possible cumulative reward when deployed for interaction with the real MDP. The dataset, $\mathcal{D}$, can be collected by any policy (e.g., exploratory, expert, or even sub-optimal) called the *behavioral policy* denoted by $\pi_\beta(a | s)$. ORL tries to learn a policy that *attains* higher cumulative reward than π_β , thereby allowing the potential automation of a wide range of decision-making domains, ranging from healthcare and education to robotics, and eventually physics, especially for those problems for which plenty of data is available but implementing an MDP is hard or infeasible. In principle, off-policy algorithms, like DQN [266], could be used to solve the ORL problem by setting $\mathcal{D}$ to be the replay buffer, and then estimating the Q-function as usual. Another more principled approach is to introduce *importance weights*, w , defined as the ratio between the actual and behavioral policy, to the RL objective

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \mathcal{D}} \left[\sum_{t=1}^T \left(\underbrace{\prod_{t'=1}^t \frac{\pi_\theta(a_{t'} | s_{t'})}{\pi_\beta(a_{t'} | s_{t'})}}_{=w} \right) \gamma^{t-1} r(s_t, a_t) \right], \quad (4.1.15)$$

from which a Q-learning rule or off-policy policy gradient can be derived. In both cases, the variance of this estimator is too large to be practical, although unbiased.

The offline setting introduces additional challenges about the impossibility of exploring and, more importantly, *distributional shift*. Exploration is about collecting on-policy interaction data, therefore it is often assumed that $\mathcal{D}$ adequately covers the space of high-reward transitions such that the offline agent can learn to reach them. Moreover, in ORL, distributional shift happens because the agent tries to answer *counterfactual* questions regarding what might happen if actions different from the ones in the dataset were predicted: this makes the learning problem not i.i.d., anymore. The distributional shift can occur:

- At test time, when the trained agent is asked to act in the MDP. Therefore, the agent is trained on one distribution of states and evaluated on a likely different one. If the test distribution shifts, i.e., the state-visitation frequency is different, even with optimal training actions the agent may incur a reward loss that grows quadratically in the time horizon.
- At training time. Since the targets of the Bellman equations [251], e.g., the Q-function regression targets in eq. 4.1.8, depend on an action sampled by the current policy,

i.e., $a' \sim \pi(a' | s')$, it causes the regression of the Q-function to depend on estimated Q-values for actions that are outside of the distribution of the actions that the Q-function was ever trained on. If so, the policy can produce OOD actions for which the learned Q-function yields too large values, which, in turn, fake the policy’s value. This issue is called *action distribution shift*.

Distribution shifts can be mitigated by either adding *constraints* as additional loss term (e.g., KL divergence), or *penalties* either to the reward or Q-targets, that enforce the policy, $\pi(a' | s')$, to be sufficiently close to $\pi_\beta(a' | s')$: in general, π should not match π_β exactly otherwise its behavior cannot be improved. Another approach to mitigate the effect of OOD actions is to estimate the epistemic uncertainty of the Q-function: this can calibrate the action values to get conservative estimates, which are less optimistic.

In conclusion, the offline setting poses additional challenges that may require extending the current statistical and ML techniques to non-i.i.d. data to solve them effectively, therefore enabling the potential to turn large datasets of collected behavior into powerful decision-making agents.

4.1.6 Practice and challenges

Too frequently agent implementations in reinforcement learning differ from their theoretical or formal specification. There can be different motivations for this discrepancy: most implementations assume the *average reward setting*, in which the discount factor in front of the gradient (see eq. 4.1.3) is dropped for simplicity; generally, continuing environments are treated as episodic ones through *time-limits* [306], which explicitly truncates the episode length: this usually introduces a source of error because the time-limit truncation must be not confused with the real episode termination, which happens when a terminal state is reached; furthermore, comparisons even among the same algorithm are often unfair due to (frequently undocumented) “implementation tricks”, also called *code-level optimizations* [307], e.g., state/reward/returns normalization, weight/gradient/reward clipping, etc, which are auxiliary details to the core algorithm that can actually boost the performance by a significant margin: Andrychowicz et al. [308] performed a large scale study, also considering network architecture, data collection scheme, optimizers, action distributions, and even regularization, providing plenty of empirical guidance about the impact of code-level optimizations and common hyperparameters values; lastly, reproducibility issues [309] along with non-standard evaluation protocols [310] may make difficult to fairly compare the performance of different agents. In particular, Agarwal et al. [310] propose to replace point estimates (e.g., mean and median) of the different runs with robust aggregated statistics like the inter-quartile mean (IQM), and also to embed uncertainty information to build confidence intervals along the reported performance. The proposed evaluation protocol should be robust against the variability of the runs (or at least quantify such amount) while allowing to compute the probability for which algorithm A is better than algorithm B, which is more robust and general compared to deciding that A is better than B based only on the median cumulative reward, for example.

Apart from the above practical discrepancies, reinforcement learning is still difficult to apply to real-world problems or in production due to assumptions made by research work that rarely can be satisfied in practice. The following nine challenges identified and empirically studied by Dulac-Arnold et al. [311] characterize common requirements of real-world systems which, as opposed to fully-observable simulators, may introduce latencies, noise, high-dimensional states and actions, constraints, operational costs, and more importantly actions have non-negligible consequences:

1. *Learning on the real system from limited samples.* Online learning on the real system is often slow due to high response time, costly to operate, and applied actions are rarely without consequences. Therefore the agent must be data-efficient, limit exploration to avoid dangerous actions, and quickly learn from mistakes. In this regard, the authors identified two desirable properties of the learning algorithm: 1) it should minimize the *regret* (i.e., the amount of return lost compared to the best-known policy that solves the task) until convergence, and 2) be *stable* after convergence, which means that the agent should always achieve a return that is no less than a sufficient amount.
2. *System delays.* Real systems can have delays in either sensing (e.g., low-frequency), actuation because of safety checks or transformations applied to actions before their actual implementation, and even reward feedback which can be delayed because of the time for an action's effect to manifest. In RL terms, an action delay can be simulated by delaying the agent's action execution for n timesteps. Whereas, an observation/reward delay is defined as withholding an agent's observation/reward for n timesteps: for action delays the agent receives the reward $r(s_t, a_{t-n})$, whereas for observation delays the reward is $r(s_{t-n}, a_t)$. As the delay increases, the performance of conventional algorithms is expected to degrade at a rate that depends on the difficulty of the credit assignment (i.e., understanding which past actions led to the immediate reward).
3. *High-dimensional continuous state and action spaces.* Large state and action spaces can be found in systems characterized by numerous sensors and actuators, for example. These high-dimensional observations may challenge table-based RL methods due to the curse of dimensionality (i.e., an exponential explosion of the number of table entries), which neural networks can mitigate. Nonetheless, even in deep RL, most actions may pose exploration challenges. For example, when the cardinality of a discrete action space is large (e.g., $|\mathcal{A}| > 10^5$) conventional methods, e.g., DQN, may not work. If so, a viable solution can be to decompose the action selection process into two steps: action candidate generation and action ranking. High-dimensional states and actions are expected to slow the learning process because of increased complexity and instability.
4. *Satisfying environmental constraints.* Improper control of a physical or software system can cause damage, performance degradation, failure, and provide incorrect results. System and hardware constraints are important for the safe and efficient operation

of a system, and also to avoid self-harm of the agent. Examples of constraints are limits on temperature and forces, maintaining minimum battery levels, avoiding obstacles, and meeting latency and throughput requirements. To this end, Constrained MDPs (CMDPs) [312] were introduced in which the agent is asked to optimize multiple reward functions: one of them optimizes the main task to solve; while the other auxiliary cost functions restrict the actions to be predicted by the agent's policy.

5. *Partial observability and non-stationarity.* Unlike simulators, real systems often do not provide the agent with observations that exhaustively describe the full state of the system (e.g., the state of sensors, motors, joints, mental state of a user, etc.) These partial observations may appear as noise, stochasticity, or non-stationarity challenging the learning. Common techniques to deal with partial observability either incorporate history into the observation, use recurrent neural networks, or even transfer the policy from simulation to the real world.
6. *Multi-objective reward functions.* Most systems have multi-dimensional costs to be minimized: for example, an autonomous vehicle should at least minimize collisions and travel time, while respecting driving rules [313]. In these cases, it is not obvious which metric to optimize. Also, how to best combine the various objectives into a single one is non-trivial, since there usually are trade-offs among them. There are many ways to handle multi-objective rewards, like predicting a value function for each objective or finding a policy that optimizes each sub-problem. Moreover, distributional RL methods [274], which model the distribution over returns, allow to directly optimize a *Conditional Value at Risk* (CVaR) [314] objective that is particularly relevant when balancing a task reward with negative outcomes. In addition, as briefly surveyed in the previous section, the reward function can even be learned from demonstrations when it is difficult to specify. Because the global reward function is generally a balance of multiple sub-goals, a proper evaluation should separate their contributions to understand the impact of each sub-goal given the current policy.
7. *Real-time inference.* To deploy RL to a production system, policy inference must be performed in real-time, matching the control frequency of the system. The constraint is violated when the agent is either slower or faster than real-time: if slower, we cannot trade accuracy for computation (e.g., deeper searches); if faster, we cannot generate data more rapidly than the control frequency. The impact of real-time inference can be measured in terms of *latency* and *throughput*: the latency is the time the agent needs to output an action given an observation, while the throughput is the frequency of input observations the agent can process.
8. *Training from offline logs.* Learning from scratch through online interaction with the real system is often too expensive, time-consuming, or dangerous. For these reasons, algorithms that learn satisfactory policies from offline system logs should be preferred. These system logs often come from an existing rule-based, or heuristic policy, documenting its behavior. We refer to this setting as offline reinforcement

learning, whose characteristics and intrinsic challenges were reviewed in the previous section.

9. *Policy explainability.* Real systems are owned and operated by humans, who must be reassured about the controllers' intentions and require insights regarding failure cases. Policy explainability is important, especially in cases where the policy might find an alternative and unexpected approach to controlling a system: understanding the longer-term intent of the agent is also important for obtaining stakeholder buy-in. In the event of policy errors, understanding the error's origins is essential. For example, Coppens et al. [315] distills a trained neural network policy into a Soft Decision Tree (SDT) [316], which is a hybrid model between a DT and a NN that learns a hierarchy of filters each of which is a simple perceptron. In this way, by examining the filters along the traversed path (from root to leaf), it is possible to rank the input features by importance (since scored by the SDT) and then map them to predicted actions given a particular state.

As a real-world problem may introduce one or more of the above challenges it is important to develop RL methods that can address all (or, at least, most) of them.

4.1.7 Software libraries and tools

Thanks to the increasing popularity and research efforts in reinforcement learning, there is a large number of open-source software libraries [317], frameworks, and collections of algorithms, ready to be applied to the target task. Environments, or Markov decision processes, are commonly implemented through the OpenAI Gym [318] (now Gymnasium [319]) Python [106] package, which provides APIs (Application Programming Interfaces) to implement an MDP, $\mathcal{M}$:

- `obs = env.reset()`: provides the first observation or initial state, s_1 , which can be regarded as sampling from the initial state distribution, $p(s_1)$. The method is called before the interaction with the environment, `env`, begins.
- `obs, r, = env.step(a)`: updates the environment's state by applying the provided action, `a`, yielding the reward, `r`, for taking that action along the next observation or state, `obs`. This method call corresponds to querying the environment dynamics, i.e., $p(s', r \mid s, a)$.
- `env.action_space` and `env.observation_space` represent, respectively, the observation or state space, $\mathcal{O}$ or $\mathcal{S}$, and the action space, $\mathcal{A}$, of the MDP, specified by implementing the `spaces.Space` class interface. The library offers predefined *discrete* (i.e., `Discrete(n)` for n discrete actions from 0 to $n - 1$), *enumerable* (multi-dimensional discrete actions by `MultiBinary` and `MultiDiscrete`), *continuous* (i.e., `Box` represents multi-dimensional continuous, bounded or unbounded actions), and *strings* (i.e., character sequences by `Text`.) These basic spaces can be combined in arbitrarily shaped ones through the `Tuple`, `Dict`, `Graph`, and `Sequence` classes.

Environments can be then “wrapped” by `Wrapper` classes to implement, for example: action repetition, reward shaping, observation normalization, time limits, etc.

After defining a suitable environment, software libraries are used to specify, train, monitor, and evaluate RL agents. Popular ones are (categorized according to the class of implemented agents) but not limited to:

- Model-free: `stable-baselines 3` [320], `Tensorforce` [321], `TorchRL` [322], and `Cleanrl` [323].
- Model-based: `Baconian` [324] and `Bellman` [325].
- Offline RL: `d3rlpy` [326].
- `Avalanche RL` [327] for continual reinforcement learning, `Marllib` [328] for multi-agent RL applications and research, and `Arena` [329], which is a platform to evaluate RL agents.

4.2 Applications in particle physics

This section describes prominent and emerging applications of (deep) reinforcement learning in particle and high-energy physics, concisely surveying the relevant literature.

4.2.1 Controlling complex equipment

The control of complex equipment like a particle accelerator, which can have hundreds or thousands of tunable parameters, is usually achieved by either hand-tuning by experts or numerical methods that given an objective function find the optimal solution in terms of a parameter configuration: therefore, even a marginal improvement translates into improved scientific yield for the many applications (e.g., fundamental physics research, cancer treatment, vaccines and drugs development, production of novel materials) that find use in a particle accelerator. To do so, a modular control system is usually built. This control system combines low-level hardware parameters with higher-level accelerator physics parameters, whose physics-to-hardware translation is stored in databases and software rules. For correction and tuning, low-level feedback systems and high-level physics algorithms adjust the beam parameters based on observables from the instrumentation. However, a control module is designed and tuned for a specific accelerator, therefore newly commissioned accelerators require brand-new tools for tuning and performance optimization, which is costly and time-consuming. At each deployment, numerical methods optimize for the best solution, which can be a long process that RL algorithms can reduce in time while providing a better solution to the control problem. In principle, a trained RL agent can compute the optimal solution (i.e., the action) given the current state of the accelerator in one inference step: figure 4.2. In this regard, adequate instrumentation is required to acquire meaningful state information, which can restrict the applicability of RL methods. Another limitation arises from sample efficiency, i.e., the number of interactions with the accelerator required

until the problem is solved with sufficient performance. Notice how the accelerator control task embeds all the challenges listed in the previous section: data efficiency or offline learning is required to minimize the queries to the real system; the state and action spaces can be large and partially observable; the operation must respect safety constraints and potentially optimize multiple objectives; and the inference must match the control frequency of the equipment while being also explainable.

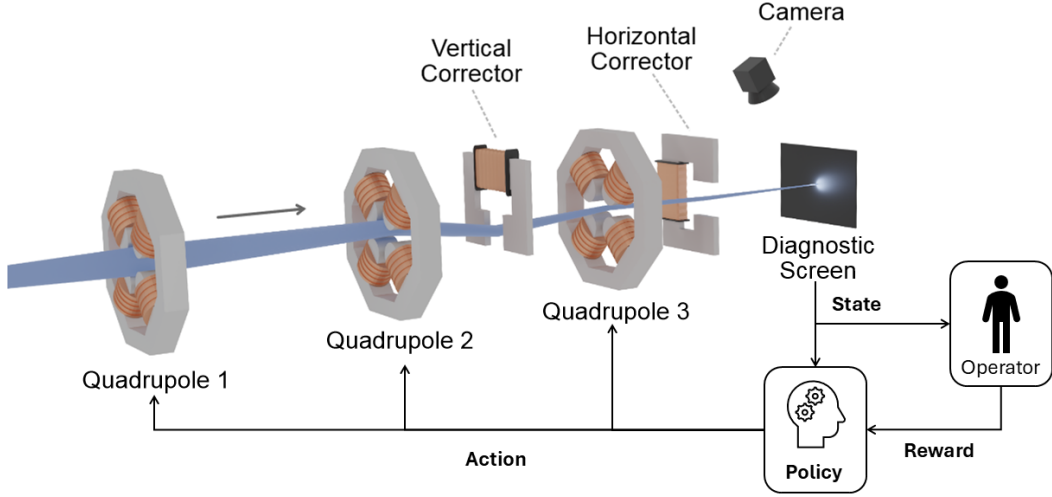


Figure 4.2: An example experimental apparatus where the quadrupoles and correctors concentrate and steer the particle beam (depicted in blue) as the control policy predicts, which can be rule-based or learned. The human operator controls the operation, acting if necessary (e.g., on faults or poor beam control). Figure adapted from [330].

Kain et al. [331] were the first to successfully deploy a model-free RL agent for online trajectory correction at the AWAKE and LINAC4 facilities at the European Organization for Nuclear Research (CERN). The authors employ a variant of the DQN agent that learns a normalized advantage function (NAF) [268] able to estimate Q-values for continuous actions, which factorizes the action-value in terms of the state-value plus advantage, i.e., $Q(s, a) = A(s, a) + V(s)$ for which it is assumed a quadratic form:

$$A(s, a) = -\frac{1}{2}(a - \mu(s))^\top P(s)(a - \mu(s)), \quad (4.2.1)$$

where $P(s)$ and $\mu(s)$ are state-dependent neural network features, for which occurs that $a^* = \mu(s)$ yields the optimal action. To improve the sample-efficiency of NAF, a prioritized experience replay (PER) [271] buffer is utilized, which biases the sampling of experience such that transitions devised as more important are sampled with higher probability: this allows the agent to utilize more relevant experience to estimate the Q-function, hence converging in less iterations. The overall agent, called PER-NAF, is trained in a simulated version of the accelerator, whose reward function is defined as the negative root mean squared value

of the difference between the measured and reference trajectories. The agent was first pre-trained in simulation and then fine-tuned online on the real system to account for effects lacking in the simulated environment. Moreover, the learning episode is reset when the agent's actions are no longer safe: this implicitly provides the agent with knowledge about system constraints.

Since building a simulator of the target equipment can be hard, a *surrogate model* can be developed from the data collected by running the real system. This is the approach employed by [332], whose task is to regulate the gradient magnet power supply at the Fermilab Booster accelerator complex. The authors first train a LSTM network [12] on the real data to approximate the environment model such that to provide a safe environment for training and evaluation, to then train a DQN agent (with a discretized, one-dimensional continuous action space) within the surrogate model, and finally deploy the trained agent on a FPGA accelerator to satisfy the latency constraints of the Fermilab Booster. The agent, trained to maximize the negative absolute value of the regulation error, achieves a twofold improvement compared to the existent proportional-integral-derivative (PID) [333] controller¹⁰. Lastly, surrogate models can leverage large datasets and supervised learning to approximate a dynamics model although attention should be paid to avoid the agent exploit model inaccuracies.

Hirlander and Bruchon [334] studied a model-based method for intensity optimization at the FERMI FEL system, finding that learning the dynamics of the problem provides higher representational power, improved sample efficiency, and increased robustness to noise. Although, the asymptotic performance of model-free agents is still superior. The authors used model ensembling techniques to estimate the uncertainty of the environment model, which is useful to understand when stopping using the model due to large inaccuracies towards the true dynamics. An approximate dynamics model is also useful for generating synthetic experiences that can be used to train a model-free agent, thus reducing the interactions with the real environment.

Kaiser et al. [330] demonstrate a successfully deployed model-free RL agent to a continuous, partially observable problem on a real linear particle accelerator. To achieve this, the authors first trained a policy with a five-dimensional action space in simulation, without using any data from the machine, to then transfer the policy while including model errors to achieve generalization in the real world. To enable a better policy transfer, the authors defined the action as a continuous delta of adjustment values to the actuators instead of using the magnet settings: this allowed them to account for real-world effects such as magnet hysteresis and environmental factors. Moreover, their problem is inherently partially observable because true information regarding the incoming beam and the magnet misalignment cannot neither be measurable nor constant: this poses an additional learning challenge, which was mitigated by injecting uniformly sampled values for the unobserved quantities to enable the agent to generalize over these sources of uncertainty. Lastly, the re-

¹⁰The PID controller is capable of continuous control by applying a correction according to the difference between the desired target value with the measurement. The controller attempts to minimize the operation error over time.

ward function is the difference between the optimization objective computed on the current and last timestep. The resulting agent is faster and comparably accurate than a competitive optimization algorithm.

To conclude, particle accelerator control is probably the perfect problem for RL. However, a satisfactory solution is non-trivial because of multiple real-world challenges. The most critical of them regards the safety of the operation, which can be mitigated by training a simulated model of the accelerator or via offline training. Moreover, the need for real-time inference could be accomplished by deploying the agent on a FPGA accelerator. Solving these challenges efficiently may enable truly general RL agents to optimize the whole workflow of a particle accelerator, potentially generalizing to different equipment types (e.g., linear, circular) and tasks (e.g., magnet, beam regulation).

4.2.2 Data quality and monitoring

Physics detectors like CMS [43], which are used to conduct high energy physics collider experiments, are complex systems that can exhibit malfunctioning or data acquisition bugs with an unknown frequency, variably and unpredictably: to run a trustworthy experiment that does avoid any bias, it is of paramount importance to detect and correct these mistakes. The interaction between the collision products and the detector material is read as an electrical signal, amplified, and transformed until the selection filters and the final storage. Errors might occur at each stage of the data flow and invalidate the whole process: considering the equipment to have hundreds of millions of sensors read every 25 ns, the chance of malfunctioning is non-negligible. Therefore, monitoring systems are deployed to assess the data quality and keep the acquisition system under control [335].

Data quality monitoring (DQM) procedures that periodically inspect the data are performed by human shifters, which is costly and results in limited accuracy. DQM is divided into two subsequent stages: online and offline. The online stage consists of continuous data monitoring that occurs synchronously with the data collection process, which is done at fixed time intervals. When anomalies are detected, these need to be fixed before continuing the data collection. After the data has been recorded, it is inspected again in the offline stage. The lack of intrinsic time constraints allows evaluating high-level distributions that require larger data sets and more time to reconstruct, to spot anomalies that have not been found in the online step and consequently discard the corresponding data: therefore high accuracy is favored instead of inference time. To this end, Parra et al. [336] provide a proof of concept regarding the application of model-free RL with human feedback to automate the DQM process, hence aimed at substituting a pool of rotating non-expert shifters that have to examine a large set of histograms representing different data features. Shifters can have diverse criteria to judge the data quality and, often, cannot thoroughly inspect all the histograms which can lower accuracy. Therefore, the authors trained two Proximal policy optimization (PPO) [280] agents: a *predictor*, whose task is to classify each histogram; and a *checker*, whose task is to decide, based on the histogram and the decision of the predictor, whether to call a human shifter who will themselves provide a label, which will be used to train the predictor. Compared to the REINFORCE estimator (eq. 4.1.9), PPO introduces

several advancements such as a refined advantage estimation method [278] which can trade bias for variance, a *trust region* that if well-tuned can guarantee monotonic improvements of the cumulative reward and better sample-efficiency. A PPO policy network is optimized towards the following objective:

$$\mathcal{L}_{\text{PPO}}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{s_t, a_t \in \mathcal{D}} \min \left(\rho_t(\theta) A(s_t, a_t), \text{clip}(\rho_t(\theta), 1 - \epsilon, 1 + \epsilon) A(s_t, a_t) \right), \quad (4.2.2)$$

where $\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\bar{\theta}}(a_t|s_t)}$ denotes the probability ratio of taking action a_t given state s_t between the current, π_θ , and old, i.e., $\pi_{\bar{\theta}}$, policies. To increase the data efficiency, $\mathcal{D}$, is reused for $K > 1$ epochs. Therefore, as the parameters of the policy, θ , got updated the ratio $\rho_t(\theta)$ towards the old policy will increase. For this reason, a clipping function, i.e., $\text{clip}(\cdot, 1 - \epsilon, 1 + \epsilon)$, is introduced to bound the growth of the ratio: after K epochs, the old weights are updated towards the newest ones, i.e., $\bar{\theta} \leftarrow \theta$. To improve the sample efficiency further, the authors augmented the training data by inserting some artificially generated histograms. Moreover, the current data point, the operation mode, and eventually a human label if the shifter has been called, define the state space of the DQM environment. During training, the predictor is asked to match the human decisions while the checker learns to ask for human feedback when the predictor is uncertain. The authors show that the whole system can learn from noisy labels improving over the human-only baseline, although the main limitation is to achieve robustness against the shifter's bias.

Bower proposes the *Adversarial Thresholding Semi-Bandits* (ATSB) formalism [337] to efficiently monitor the current flow from multiple electronic device channels on the ALICE [338] experiment at the LHC. The operation challenge is to prevent the detector from automatically switching off a device channel when the current flow exceeds a given limit because, doing so, negatively impacts the experiment's capacity to collect data. Therefore, ATSBs only observe channels when their associated current exceeds (or deviates by more than) a given threshold. The adversarial thresholding semi-bandit problem is an extension of the non-stochastic multi-armed bandit (MAB) model [339], where a player wants to only pull arms (i.e., actions) with feedback (i.e., reward) meeting some threshold condition: the term *adversarial*, here indicates that no statistical assumption can be made and that rewards can be generated by some potentially competitive environment. The classical MAB problem¹¹ is defined over a set of K arms, $i \in \{1, \dots, K\}$ and a sequence of rounds $t \in \{1, \dots, T\}$. Pulling arm i in round t results in reward $R_{t,i}$, which is a random variable with an unknown distribution: in the adversarial setting the rewards are not i.i.d., which is a better assumption for real-world applications. Therefore, the goal of the player is to maximize the number of times an arm returns a value exceeding some threshold after being pulled and over a total of T pulls. The goodness of a bandit algorithm is evaluated through

¹¹Multi-armed bandits can be seen as a simplification of the reinforcement learning problem, in which actions (or arms) are *not* state-dependent. Although, extensions to MAB such as the contextual bandits [339] associate a *context* to actions and rewards, which is closer to RL.

the *cumulative regret*:

$$\mathbb{E}[\mathcal{R}_T] = \mu^* T - \sum_{i=1}^K \mu_i \mathbb{E}[N_{T,i}], \quad (4.2.3)$$

where $\mu_i = \mathbb{E}[R_{t,i}]$ is the average reward for arm i , $N_{T,i}$ is the number of times arm i has been pulled, and μ^* is the reward gained by playing optimally. The lower the regret, the more optimal the agent plays. Compared to MAB, the decision process of an ATSB player is decomposed into *selection*, in which the player decides which arms to pull, and *querying*, which is about deciding whether or not to observe none, some, or all selected arms for feedback. Moreover, the player knows a threshold value, $\tau \in (0, 1)$, which can be static or adaptive. In ATSB, the player's objective is to observe arms only for the rounds whose feedback exceeds the threshold, whose rewards amount to $R_{t,i} 1[R_{t,i} > \tau]$, i.e., the feedback is zero if the reward is below the threshold for a given arm i and round t . To address the challenges introduced by the online monitoring of the anode current in ALICE, the author proposes an ATSB algorithm called **AliceBandit**, which comprises three modules: the first updates a list containing arms that exceeded the threshold in round t , i.e., $i \in \mathcal{T}_t$, which are all selected (pulled); the second module is responsible for querying the pulled arms according to Bernoulli random variables, $B_{t,i}$, that are updated during the play; the final module pulls any arm that has not been pulled for at least $t_{\max}$ rounds, ensuring that even if the likelihood of querying an arm becomes very small it is not forgotten about completely. **AliceBandit** allows solving the monitoring task for both fixed threshold and threshold interval approaches, neatly improving over the baselines: the former is useful to implement *overcurrent detection*, while the latter enables *current deviation detection*. In this latter setting, **AliceBandit** identified deviations in anode current channels hours before a trip event, enabling the human operator to take preventative actions.

4.2.3 Jet representation and reconstruction

Reinforcement learning can also be utilized as an improved “preprocessing step” of jet and particle observables, enabling the optimization of their properties [340], the reconstruction of decay patterns [341], and even to optimize the order of the input variables [84]. These RL applications allow enhancing the jet or particle representation to boost the accuracy of deep learning models on target downstream tasks.

Carrazza and Dreyer developed a DQN [266] agent to find an optimal jet grooming strategy, which is a post-processing of jets that removes soft wide-angle radiation (i.e., low-energy particles emitted at wide angles relative to the jet direction) that is not associated with the underlying jet substructure. Therefore, a grooming that does optimize the jet properties can boost the efficiency of particle taggers. The authors represent a jet as a binary tree (figure 4.3), where each node contains information about the kinematic properties of the two parent particles. For each node of the tree, the children nodes a and b are ordered according to the transverse momentum such that: $p_{t,a} > p_{t,b}$. In this way it is possible to define the state space, in which the individual states depend on the two children as follows:

$$\mathcal{S} = \{z, \Delta_{ab}, \psi, m, k_t \mid a, b\}, \quad (4.2.4)$$

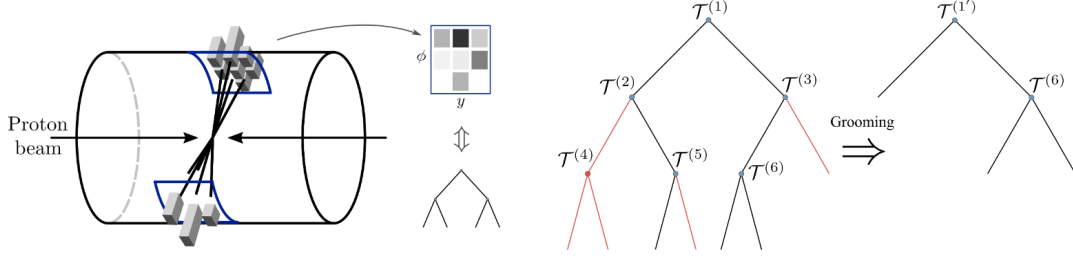


Figure 4.3: Jets originating from proton-proton collision are represented as binary clustering trees (left), which are successively pruned by the grooming process (right), yielding a representation that is more suited for downstream tasks. Figure adapted from [340].

where $z = p_{t,b}/(p_{t,a} + p_{t,b})$ is the momentum fraction of the softer (i.e., or lower momentum) child b , Δ_{ab} the momenta of the two particles, ψ is the azimuthal angle, m is the mass, and $k_t = p_{t,a}\Delta_{ab}$ is the transverse momentum of b relative to a . Since the jets are binary trees, a grooming algorithm recursively applies a policy, $\pi(s_t)$, on each branch of the tree to decide based on the current state, s_t , whether to remove the softer of the two branches. The reward function that guides the agent's learning comprises two terms: the first component compares the mass of the current jet to a target mass, called R_M , and the second term, R_{SD} , rewards either the removal of wide-angle soft radiation or the preservation of hard-collinear (i.e., low-angle) emissions. The overall reward function is defined as follows:

$$R(m, a_t, \Delta, z) = \frac{\Gamma^2}{\pi(|m - m_{\text{target}}|^2 + \Gamma^2)} + \quad (= R_M)$$

$$+ \frac{a_t \min\left(1, e^{g(\alpha_1, \beta_1, z_1)}\right) + (1 - a_t) \max\left(0, 1 - e^{g(\alpha_2, \beta_2, z_2)}\right)}{N_{SD}}, \quad (= R_{SD})$$

where Γ , N_{SD} , α_i, β_i, z_i with $i \in 1, 2$ are hyperparameters, and $g(\alpha_i, \beta_i, z_i) = -\alpha_i \ln(1/\Delta) + \beta_i \ln(z_i/z)$. Since the grooming agent is trained on both signal and background samples, the reward term R_M is changed as follows

$$R_M^{\text{bkg}} = \frac{m}{\Gamma_{\text{bkg}}} \exp\left(-\frac{m}{\Gamma_{\text{bkg}}}\right), \quad (4.2.5)$$

to account for the missing reference mass in case of a background event: Γ and Γ_{bkg} are also changed accordingly. The resulting grooming agent optimally removes soft wide-angle radiation achieving improved mass resolution, background reduction, and resilience to hadronization, matching state-of-the-art techniques used at the LHC.

Brehmer et al. train a model-based agent to solve the jet clustering problem starting from observed final-state particles [341]. The agent is responsible for constructing the most plausible binary tree of particle splittings that best explains the parton shower process that leads to a jet. This can be framed as an optimization problem, whose search space is too

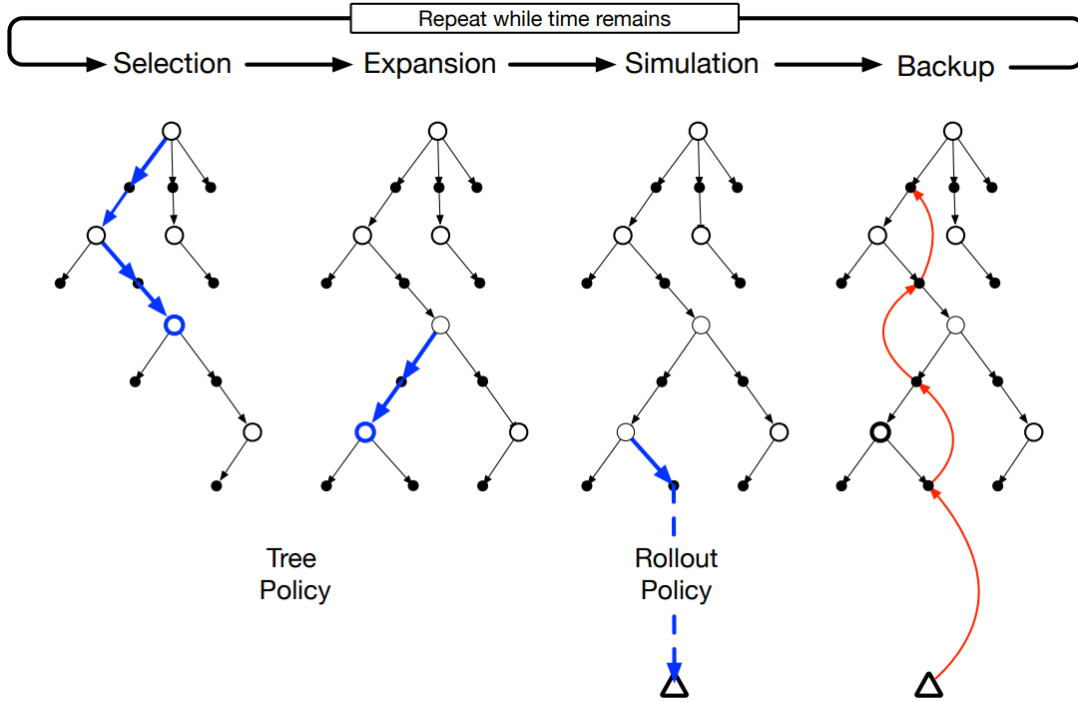


Figure 4.4: The Monte Carlo Tree-Search algorithm comprises four generic steps, in order: selection, expansion, simulation, and backup. Figure from [251].

large to find the true maximum-likelihood tree. Therefore, MCTS [285] guided by a neural network policy is employed to search over such a large space of solutions efficiently. The problem of clustering is formulated as a MDP, where:

- The *state space* is given by all possible particle sets at any given point during the clustering process, i.e., $s_t = \{p_{t,1}, \dots, p_{t,n_t}\}$ where n_t is the number of particles $p_{t,i}$ at time t . The initial state is the final-state momenta, $s_N = \{p_{N,1}, p_{N,2}, \dots, p_{N,n_N}\}$, where n_N is the number of particles and N the time index¹².
- The *actions* are the choice of the two particles i or j to be merged, with $1 \leq i \leq j \leq n_t$.
- The *transition dynamics* updates s_t to s_{t-1} by replacing the particles $p_{t,i}$ and $p_{t,j}$ with a parent $p_{t-1,i} = p_{t,i} + p_{t,j}$. Each state transition reduces the number of particles by one, hence the episode terminates after N splittings when only a single particle is left.
- The *rewards* are the splitting probabilities, i.e., $r(s_t, a_t, s_{t-1}) = \log p_s(s_t | s_{t-1}, a)$.

¹²Notice that for this problem the time step *decreases* instead of increasing, since the agent starts from the leaf nodes (i.e., the final-state momenta) to build the clustering tree until reaching the root node.

Rolling out an episode leads to a proposed clustering tree, $s = \{s_1, \dots, s_N\}$, with the cumulative received reward being the log-likelihood of this tree. Since the transition model is known, model-based planning such as MCTS can be applied to search the solution space. MCTS incrementally builds a search tree by adding nodes each representing a promising state based on the rollouts' cumulative reward, as pictured in figure 4.4. Outside the tree and at the leaf nodes the *rollout policy* is used for action selections, which can be as simple as uniform selection. Then, for states inside the tree a *tree policy*, which is informed by value estimates, can balance exploration and exploitation. For example, the authors select the action that maximizes the upper confidence bound on the action values (PUCT):

$$U(s, a) = Q(s, a) + c\pi(a | s) \frac{\sqrt{N(s)}}{1 + N(s, a)}, \quad (4.2.6)$$

where $Q(s, a)$ is the mean normalized reward used as an estimate of the action value, $N(s)$ and $N(s, a)$ are the state and action visitation count, and c a hyperparameter that balances exploration and exploitation. MCTS is executed after encountering each new state, to select the agent's action for that state. Each MCTS iteration is made of four steps [251]:

1. *Selection.* Starting at the root node, a tree policy based on estimated action values traverses the tree to select a leaf node.
2. *Expansion.* The tree is expanded from the selected leaf node by adding one or more child nodes reached from the selected node via unexplored actions.
3. *Simulation.* From the selected node, or one of its newly added child nodes, a simulation of a complete episode is run with actions devised by the rollout policy. The result is a Monte Carlo trial with actions chosen first by the tree policy, and beyond the tree (i.e., after the leaf node) by the rollout policy.
4. *Backup.* The return generated by the simulated episode is propagated back to update the action values attached to the edges of the tree traversed by the tree policy in the current iteration. No values are saved for the states and actions visited by the rollout policy beyond the tree.

MCTS continues executing the above steps, starting each time at the tree's root node, until the computational budget is over. Then, after applying the selected action the environment consequently transitions to a new state, from which MCTS is re-executed. In the limit of infinite rollouts (i.e., episodes), for each state, MCTS will converge to the true visitation count and value, which are used to plan the next best action, a^* . Following AlphaGo Zero [290], a neural network policy is then trained to mimic the output of the MCTS planning allowing to predict the best action given a state without planning at each time step. To improve performance, the authors provide the splitting probabilities as input along with the four-momenta of the particles, and initialize the MCTS search tree by running beam search. The resulting agent can generate a clustering tree of higher likelihood than both

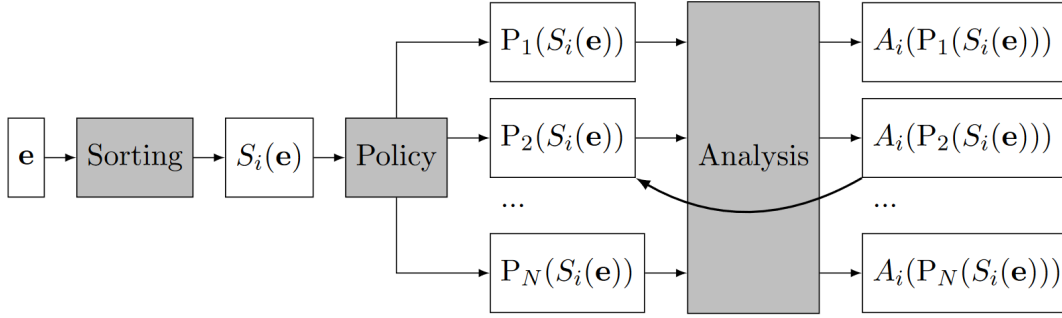


Figure 4.5: Reinforced sorting network architecture. Sorting and Analysis are two distinct networks, while the Policy performs fixed permutations, P_j , of the sorted events, $S_i(e)$. Figure from [84].

greedy and beam search, which is expected to improve the results of downstream physics analyses.

Often, the performance of deep learning models on particle physics tasks strongly depends on the order of their input variables. To exploit this, Erdmann et al. developed a reinforcement learning sorting agent to rearrange the input variables to maximize the performance on the downstream task [84]. They designed a two-stage system (figure 4.5), in which the sorting network (the agent) learns using feedback from the analysis network (the environment), which is trained on a target task like classification. The sorting network receives p_T -sorted events, producing an internal representation for the arrangement of the jets in terms of position indices. These indices specify how the input should be rearranged for a given jet. Since the analysis network is asked to separate the signal from the background, optimal classification performance can be reached if the jets are sorted according to generator information, which provides the upper bound performance. Initially, the sorting network is pre-trained to mimic the sorting according to the transverse momenta, which helps to avoid early convergence. Then, the subsequent training alternates between the sorting and analysis network. Given a current ordering of the events, i.e., $S_i(e)$, these are first permuted according to a fixed schema (e.g., all permutations of size two) and, then, the classification (or analysis) network is evaluated on each permutation. The top ranking permutation, $S_{i+1}(e)$, is then used as a learning target for the sorting network and as input to improve the analysis network. Therefore, the performance of the two networks depends on each other. With this learning scheme, the sorting agent can predict an ordering that outperforms the classification performance reached on the p_T -ordered sequence, approaching the optimal sorting, which is usually unknown in advance.

4.2.4 Charged particle tracking

The problem of track reconstruction, or *tracking*, delves into the identification of charged particle trajectories that traverse particle detectors, which estimate the position, direction,

and momentum of the particles produced in a collision event like muons, electrons, and hadrons to name a few. To this end, a tracker device is positioned close to the collision region, receiving the highest volume of particles. The tracker consists of multiple layers and detectors, which can be seen as cells (or pixels) of a grid. When the charged particle crosses a pixel it deposits charge, known as a *hit*. According to the particle's incident angle, its energy can deposit in multiple adjacent cells which need to be clustered to localize the true hit before tracking can begin. Traditionally, the tracking task involves four steps: track candidates (or seeds) generation, track finding, track fitting, and track selection [342]. *Seed generation* provides an initial estimate of the five parameters¹³ that describe the helical path of a track. This is done by connecting hits in adjacent layers. Then *track finding*, builds a track from each candidate. Based on track direction and position, the next compatible detector layers are identified, and if there is more than one of them the track is split into different candidates. The process stops when the track reaches the end of the tracker, the p_T is too low, or it contains too many missing hits (i.e., unrecorded hits due to detector inefficiencies). Next *track fitting*, takes the set of measurements in a track and uses a Kalman filter [343] to estimate the parameters describing the state of a particle in the tracking device. Since a track can be split into multiple candidates, some of which are not associated with a charged particle, *track selection* removes such fake tracks.

The major computational bottleneck of these algorithms is the combinatorial explosion of track candidates arising from the increased density of particle hits, especially in high-luminosity environments. Therefore, deep learning methods, particularly graph neural networks [344], are introduced to mitigate the increasing complexity of the combinatorial solution space, especially for future applications. Training these methods requires computationally intensive Monte Carlo simulations containing ground-truth information, possibly introducing discrepancies between simulation and reality. For this reason, Kortus et al. formulate the problem of particle tracking as a Markov decision process solving it with reinforcement learning without simulated or labeled data [345].

The authors focus on reconstructing particle trajectories of protons captured in a high-granularity digital tracking calorimeter (DTC). The aim is to find a policy that maximizes the physical plausibility of the found trajectories. To do so, the policy learns from a MDP defined on a graph over the layered point cloud of detector readouts: the states define partial tracks; the action set depicts possible transitions between hit centroids; and the reward function encodes the underlying physics of particle interaction in matter. This graph has vertices defined by hit centroids, and edges connecting existing neighboring readouts over subsequent layers but in reverse direction. Hence, the graph establishes the possible states and valid actions for the MDP, where a full track candidate is represented by the trajectory $\tau = (s_0, a_0, \dots, s_T, a_T)$ where T is the episode length. The policy and state-value branches are implemented using a shared encoder-decoder Pointer Network architecture [346], receiving distinct action- and observation features. The network is equipped with

¹³A helix can be described by $(\frac{q}{p}, \lambda, \phi, x_\perp, y_\perp)$, where $\frac{q}{p}$ is the charge of the particle divided by the momentum, λ the dip angle, ϕ the azimuthal angle, and $x_\perp$ and $y_\perp$ are defined in a coordinate system local to the track [342].

multi-head attention (explained in detail in section 5.1) calculating attention weights α_i^π and α_i^V for each possible action $a_t^{(i)} \in \mathcal{A}_t$, where $\mathcal{A}_t \subset \mathcal{A}$ is the set of valid actions at time t . These weights are employed to estimate the policy and state-value function as follows:

$$\pi_\theta(a_t^{(i)} | s_t) = \frac{\exp(\alpha_i^\pi)}{\sum_{j=0}^{|\mathcal{A}_t|} \exp(\alpha_j^\pi)}, \quad V_{\pi_\theta}(s_t) = \frac{1}{|\mathcal{S}|} \sum_{i=0}^{|\mathcal{S}|} \alpha_i^V.$$

Lastly, the immediate reward, r_t , is defined by exploiting the physics knowledge of particle interactions. In particular, when the particle traverses the detector layers its trajectory is mainly affected by multiple Coulomb scattering. Therefore

$$r_t = \log p(\Delta\theta(s_t : s_{t+1}) | X_0(s_t : s_{t+1}), R_0(\tau)),$$

where $p(\Delta\theta(s_t : s_{t+1}) | \cdot)$ is the probability of observing the angular deflection $\Delta\theta$ given the state transition $s_t \rightarrow s_{t+1}$ and the full trajectory τ ; $X_0(s_t : s_{t+1})$ the radiation length of the detector section enclosed by $s_t : s_{t+1}$; and $R_0(\tau)$ the estimated range of the track candidate τ . The authors show that maximizing the above reward function corresponds to maximizing the log probability of observing a track. In practice, the agent's parameters are found by optimizing a PPO-CLIP objective [280]. The resulting agent is evaluated to various unseen phantom geometries, particle densities, and beam spot positions, showing robust generalization performance rivaling an equivalent supervised trained model and surpassing manually tuned heuristic search algorithms.

4.2.5 Emerging and future applications

In this section, we discuss frontier applications of reinforcement learning in particle physics and possible future uses given the current capabilities of modern RL.

Physics model building. Reinforcement learning can substitute conventional optimization methods to automate the exploration of new BSM physics theories, especially when systematic scanning is impossible, proposing plausible physics models under the desired characteristics. Initial successful approaches have been demonstrated to construct quark mass models [347] and explore the flavor structure of quarks and leptons [348]. Both studies consider the space of Froggatt-Nielsen (FN) models, discovering physics models consistent with the experimentally measured quark and lepton masses.

In particular, the environment consists of the set $\{\mathcal{Q}\}$ of all FN charge matrices, and an action a adds or decreases a single charge, $\mathcal{Q}_a$, by one while keeping the other charges unchanged. Moreover, Harvey and Lukas [347] leverage domain knowledge to design a state-value function, $V(\mathcal{Q})$, that checks order of magnitude agreement rather than measuring the goodness of a state $\mathcal{Q}$. Through $V(\mathcal{Q})$, it is possible to identify phenomenologically promising terminal states, which give rise to approximately correct masses and mixings. Given this, the agent can learn the optimal action for a state by maximizing the reward

function:

$$r(\mathcal{Q}, a, \mathcal{Q}') = \begin{cases} V(\mathcal{Q}') - V(\mathcal{Q}) & \text{if } V(\mathcal{Q}') - V(\mathcal{Q}) > 0 \\ r_{\text{offset}} & \text{otherwise} \end{cases},$$

where $\mathcal{Q}'$ is the next state, and r_{offset} is a fixed negative value penalizing a decrease of the state value. Lastly, the authors performed a Monte Carlo search to optimize the value of the terminal states found during training or inference further, achieving models explaining the quark mass well while exploring only a fraction of the search space. However, a major research effort is required to develop a general search strategy of plausible BSM models.

Physics instrument design. Physics detectors are essential instruments to test fundamental theories, as they detect and measure the particles produced during an experiment. These apparatuses require a specialized design in which every component is optimized for maximum efficiency, sensitivity, and precision. This implies that their design, building, and maintenance costs are extremely high. Reinforcement learning can help discover instrument designs that maximize overall performance while reducing expenses, as it can explore non-convex, multi-dimensional, and even combinatorial solution spaces efficiently. Moreover, it can deal with discrete choices (e.g., the number of detector layers, which material to use, etc) unlike gradient-based optimization, without assuming a model of the instrument.

As a proof of concept, Qasim et al. applied RL to optimize the longitudinal segmentation of a calorimeter, and the placement of trackers in a spectrometer [349]. In both cases, the PPO [280] agent places layers sequentially until the design is complete or the resource budget is exhausted. The agent predicts a continuous action to determine the position of the next layer and a discrete action to select the layer's physical properties (e.g., sensor type, and granularity). The environment is designed to be partially observable, sending the agent observations containing the current longitudinal location and the resource budget (or number of placed layers). The reward is the design score of the instrument quantifying its resolution, awarded to the agent only at the final interaction step. Nonetheless, the agent can receive a penalty for design violations (e.g., two layers are too close) at intermediate steps. The authors show that the designs found by the RL agent improve over the baselines. However, significantly more work is needed to balance multiple objectives and to design a full instrument (e.g., account for the geometry, joint optimization of sub-detectors, etc.)

Real-time event selection. Since particle collisions yield huge amounts of data, event-selection (or *trigger*) criteria are necessary to understand which events to discard to keep the data volume manageable for the storage system. This selection must be performed in real-time (matching the frequency of event production, i.e., 40 MHz at the LHC [42]), therefore event selection is implemented through simple rules (e.g., based on object type, and p_T) or with specialized hardware such as FPGAs or ASICs to be fast enough for online event analysis. Considering general-purpose experiments, like ATLAS [229] and CMS [43], an effective trigger strategy should achieve a reduction rate of about 10^7 while being sensitive to a wide range of physics processes and their occurrence probability [108]. Hence, the

trigger should reject most background events while preserving interesting events, which might represent new physics signals.

Applying either ML or DL to this setting is challenging because the prediction should be fast; the objective function is not obvious as the model should maximize the yield of physics analyses; and the environment can introduce distribution shifts as operational conditions may vary during time and across different runs. Reinforcement learning can ease these challenges as it can train well-performing agents even with few learnable parameters, and the interactive nature of the training makes them robust against small distribution shifts. Nonetheless, specifying a suitable reward function may still be difficult: the likelihood of events w.r.t. known background processes would be one option; another way could be first to learn the reward through inverse [298] or offline [299] RL (briefly surveyed in section 4.1.5) using currently selected data, and then optimize it with model-free RL. Therefore, a successful RL implementation is expected to retain events with greater physical interest.

4.3 Summary and outlook

Reinforcement learning is a data-driven paradigm that through repeated interactions with a real or simulated environment can discover sequential decision-making policies that act optimally within the environment. If the environment dynamics is known or can be approximated, model-based RL methods should be preferred over model-free ones. Furthermore, variants of RL known as inverse and offline RL can be applied to fixed datasets of interaction experience, which can be sub-optimal, to learn agents without deploying them in the environment, which can be dangerous or costly. To be applicable, RL requires the target problem to be defined in terms of a Markov decision process with a suitable reward function. For this reason, obvious applications of RL are mostly found in games, robotics, and control problems.

In recent years, promising applications of RL in particle physics are emerging challenging the classical way of solving the respective problems. These applications are related to physics analyses but only *indirectly*. For example, DQM tries to minimize the negative impact that sensor malfunctioning has on data collection since low-quality data reduces the efficiency of data-driven (i.e., ML and DL) classifiers. Hence, the improvement of DQM (as well as accelerator control, and so on) alone is expected to enhance the signal sensitivity in both novel and old physics analyses. Therefore, through data and interactions, RL agents can learn decision strategies expected to surpass human-designed, domain knowledge-based, heuristics and rules. However, RL is in general still far from being easily and widely applicable to real-world problems due to concerns about the data efficiency, safety of the policy, compliance to system, hardware and human-specified constraints, interpretability of the decision process, and the definition of a suitable yet informative reward function. These are among the most critical issues that future research in RL for particle physics should focus on.

Chapter 5

Foundation Models for Particle Physics

Introduction. A *foundation model* [4] can be defined as a typically large neural network¹ (pre-)trained on a vast amount of data, which results in learned weights that can be easily adapted to the target downstream task(s) achieving high performance with either low data or few gradient updates. Foundation models (FMs) leverage *transfer learning* [350, 351] implemented via *pre-training* to learn transferable and generally useful knowledge about a given domain. The FM is first pre-trained on a surrogate task (i.e., a task that is not your target task, but somehow related to it) and then *fine-tuned* on the downstream task of interest. After pre-training, the FM is expected to have learned a representation of the domain and given data modalities, such that successfully solving the downstream task is achieved at a fraction of both training data and compute compared to directly learning to solve the task *from scratch*, i.e., starting from a completely untrained model: often, the performance at convergence is even higher than training from scratch thanks to the better initialization point.

In general, when fine-tuning on a target downstream task a *head*, i.e., a typically small(er) neural network, is added on top of the *backbone*, which is the part of the pre-trained model that can be reused for multiple tasks: figure 5.1. During pre-training, a head is still necessary to train the backbone on the pre-training (or *pretext*) task, which usually involves optimizing a contrastive [68, 352] (i.e., predict one or more properties or parts related to the input) or generative [352] (i.e., synthesize an output related to the input according to the data distribution) objective. Nowadays, especially for Large Language Models (LLMs) [7, 353, 25, 354, 355], the generative pre-training is the favored approach to learn the backbone mainly because, objectives like *predicting part of a masked input* (BERT [353] style) or the auto-regressive *next-token prediction* (GPT [7] style) are general enough to enable an efficient fine-tuning on a vast variety of tasks, even across data modalities (e.g.,

¹Although the notion of “large” can be subjective, what is nowadays considered to be a large model has a parameter count that spans between at least a few hundred million (e.g., $\sim 500\text{M}$) and up to a trillion (e.g., $< 1000\text{B}$). Common parameter counts are about 1B, 3B, 7B, 8B, 70B, and 450B.

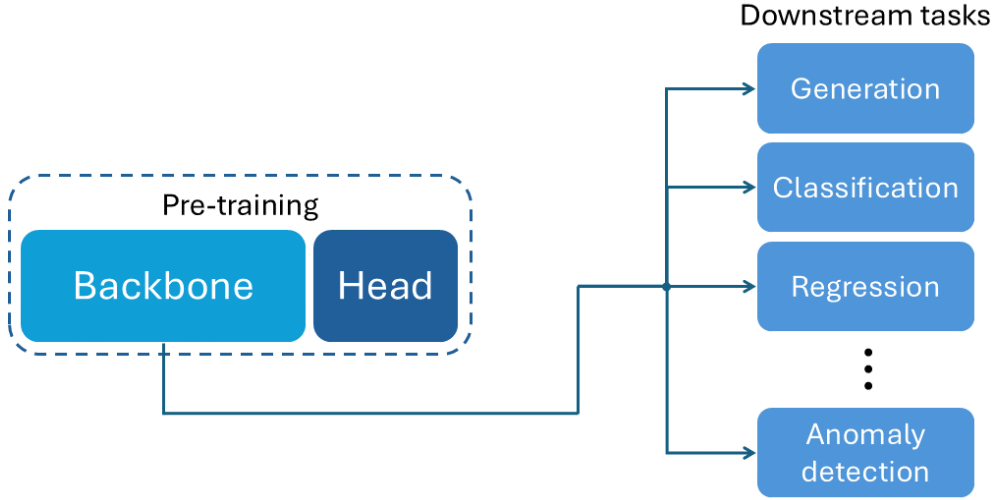


Figure 5.1: The pre-trained backbone can be reused for multiple downstream tasks when fine-tuned. Nonetheless, each task requires a specific head.

images and videos paired with text).

Essentially, foundation models leverage an old recipe, i.e., transfer learning via pre-training, with the addition of *scale*. By increasing scale, we mean enlarging the model size (i.e., its width and height, which results in more learnable parameters), increasing the dataset size, and consequently increasing the computational budget. When the scale is not trivial (e.g., billions of parameters and training samples), the pre-trained models start to exhibit interesting properties, like efficient fine-tuning, zero-shot capabilities [356], i.e., being able to solve a task without any further training, and even *in-context learning* [25], in which the model can be adapted to a new task just by editing its textual *prompt*² to describe (even partially) the task. Indeed, training on such a vast amount of data is only possible by *not* leveraging labels, otherwise, the cost, time, and accuracy of data labeling would be the major bottleneck to increase the dataset size. Unlabeled data can be effectively leveraged by self-supervised learning [289], which provides the pre-training task. Previously, models were pre-trained mostly on large annotated datasets, like ImageNet [358], and then fine-tuned on a much smaller (usually, specifically built) dataset for the target task or domain. In this way, the pre-training accounts for the loss in performance that would be achieved if directly training on the target dataset. Moreover, FMs are also related to multi-task learning [95]: in MTL, the backbone is simultaneously trained to solve multiple tasks (e.g., classification and regression), learning a common representation of them. Conversely, the focus of FMs is to find a general yet rich representation via pre-training that can enable wide transfer, rather than learning tasks-specific representation which may be less transferable.

²In the context of LLMs, the prompt is the query (expressed in natural language) the user submits to the language model. For a broader discussion on prompting techniques, refer to Liu et al. [357].

Capabilities. Modern LLMs (e.g., GPT-4 [354], Llama 3 [355], etc) are becoming more and more attractive for the users (and companies) thanks to their improved accuracy and wider capabilities. The most powerful LLMs are *multi-modal*, meaning they can relate multiple data modalities (e.g., images, videos, and speech) to the text prompts, allowing them to expand their capabilities beyond pure language tasks (e.g., translation, rephrasing, summarizing text, etc) to also vision tasks (e.g., image captioning, object recognition, image and video generation, etc.) The recent introduction of *tool use* [355] in LLMs, allowing them to utilize APIs, search engines, and code compilers to answer the user query more effectively and with up-to-date information, has the potential to transform the actual dialogue systems (e.g., ChatGPT [359]) into more general assistants in the (near) future. An important characteristic of LLMs, which makes them useful in practice, is their ability to follow instructions. A user can specify a prompt of the kind

Summarize this scientific article in five points, sorting them by importance.,

to instruct the model to first “execute” text summarization on the provided document (or, look it up on the web via tool use), then format and sort the result as desired. This *instruction following* behavior is ensured via a subsequent training phase originally called *Reinforcement Learning from Human Feedback* (RLHF) [360], in which the answers of the model are aligned to satisfy human preferences. Models trained at a scale comparable or even larger than GPT-3 [25], start to exhibit a capability called in-context few-shot learning in which the LLM can be adapted to a downstream task just by providing it a prompt “tuned” for the specific task, without the need of collecting additional data for fine-tuning; the prompt can include task instructions or even one or more demonstrations (e.g., input-output examples) such that to guide the LLM towards the desired use case, for example:

Format the date as <day>/<month>/<year>.

Here is an example: 9th of May, 1967 → 09/05/1967.

In this case, the model is adapted to perform a date formatting task, in which a single example is provided: the subsequent interactions only require the user to provide a date in textual form, which is then formatted by the LLM as instructed. This practice enables us to get task-specialized LLMs just by *prompt engineering* [357]³, i.e., the manual or automated process of creating a prompting function that results in the most effective performance on the task of interest.

Reasoning tasks, being arithmetic, commonsense, or even symbolic, are notoriously difficult for LLMs. One way to enhance their reasoning skills is via *chain-of-thought prompting* [361]: a chain-of-thought (CoT) is a series of intermediate reasoning steps, described in

³Notice that this can be regarded as few-shot learning [122] since a demonstration is provided. However, no gradient updates occur to fine-tune the model!

natural language, leading to the final output. A language model can be provided with CoT prompts, consisting of triples (**input**, **chain-of-thought**, **output**), showing how to decompose complicated reasoning tasks into intermediate steps, ideally solving each before outputting the final answer. An example commonsense CoT prompt (from [361]) is the following (the CoT is highlighted in bold):

Q: Yes or no: Would a pear sink in water?

A: **The density of a pear is about 0.6 g/cm^3 , which is less than water.**

Thus, a pear would float. So the answer is no.

CoT prompting enables the model to allocate more computation to problems that require additional reasoning steps. Furthermore, each step is potentially human-interpretable because the model describes it in natural language.

Lastly, the common feature of FMs (including LLMs) is the ability to specialize them on a given task, but also to adapt them to a given domain, by fine-tuning on a specific dataset: since FMs are usually considerably large models, lightweight adaptation techniques (such as Low-Rank Adaptation, or LoRA [362]) have been developed to lower the computational requirements for training them. In particular, LoRA significantly decreases the number of trainable parameters compared to full fine-tuning, by injecting low-rank decomposition matrices into each transformer layer while keeping the pre-trained weights fixed.

Limitations. Since FMs are trained on a vast corpus of data, usually collected by automated tools such as web scrapers, these can contain harmful content, and inherent biases, posing security and privacy concerns. A major human effort is required to clean, filter, quality check, and balance [355] the data used for training to mitigate these issues. The danger of defects in the FM is exacerbated by the fact that these kinds of models are generally deployed to be fine-tuned on a wide variety of tasks, which can inherit biases, failure modes, and unsafe behavior of the backbone, compromising all of them at the same time: ad hoc prompts, may even reveal the training data posing major privacy concerns [40]. Therefore, a foundation model may become a *single point of failure* being the main target for attacks against applications powered by it. For example, the *data poisoning* attack [363] can inject malicious training data that might impact the applications adapted from the FM. Moreover, both FMs and LLMs are little understood so unknown emergent capabilities (for which the model was not purposely designed and trained) may exist, which can be exploited for malicious intents: listing, studying, and evaluating them is a major research challenge.

FMs are often embedded in commercial products or offered via APIs, exposing them to a vast audience of users, which can be impacted by either misuse or inappropriate answers by the model: the societal and ethical implications of foundation models should also be estimated to drive responsible future research. In practice, *safety classifiers* like Llama Guard [364] can be used to detect whether input prompts and generated output responses violate safety policies on specific categories of harm. This allows us to study and quantify the amount of “danger” the model has indirectly learned, and to evaluate the effectiveness of data cleaning and filtering along with other implemented safety measures.

Another source of danger regarding LLMs is their tendency to *hallucinate* incorrect statements pretending to be correct: notable examples are prompts answered with invented facts or citing nonexistent but plausible sources. This phenomenon is attributed to knowledge gaps in the model itself. However, it can also occur when the LLM tries to justify an erroneous answer by making incorrect assertions, which is called hallucination snowballing [365]. Since current LLMs cannot revise their answer (unless explicitly told) by acknowledging their initial mistake, conditioning on a false context may induce them to behave erroneously. Evidence provided by [365], shows that using prompts that generate a reasoning chain (i.e., a CoT) can significantly reduce the error rate. One example prompt is to append the phrase **Let’s think step-by-step** at the end of the original question.

Foundation models represent an attractive data-driven paradigm for designing general-purpose models for the target domain, enabling effective and efficient adaptation to multiple related tasks. In section 5.1, we will learn about the attention mechanisms, which are presumed to be responsible for the capabilities learned by LLMs. Section 5.2, introduces the transformer architecture that powers popular models like GPT-4 [354] and Llama 3 [355]. Subsequent, section 5.3 outlines the potential of large models in particle physics being investigated by current research. Next, section 5.4 demonstrates an application of a HEP FM on real collision data. Lastly, section 5.5 concludes this thesis chapter by outlining HEP-specific open problems.

5.1 The attention mechanisms

Attention is among humans’ most fundamental cognitive functions, allowing us to process only a subset of the incoming information selectively. This is particularly relevant for vision⁴ since our visual system cannot process the huge amount of information it constantly receives. Therefore, the *selective attention* capacity allows us to focus only on the portion of the input that can control (or influence) our behavior, discarding the rest. Attention introduces a bias toward salient or novel information, which gets prioritized over less relevant and distracting ones. In some cases, for example, when searching for a particular object, attention creates a “template” holding task-relevant features that effectively boost our performance on the task [366].

In a similar spirit, the attention mechanism in deep learning provides the model the ability to dynamically focus on a relevant portion of the input, allowing it to pair and relate information even across time, significantly enhancing performance in applications where this capability is valuable like neural machine translation [367, 368] and image captioning [369].

⁴Vision is our predominant sense to which most neuronal “resources” are allocated, making up the visual cortex whose organization is hierarchical processing increasingly complex information.

5.1.1 Additive, hard, soft, local and global attention

Although the attention mechanism has been developed mainly for natural language tasks [367, 13, 25], its formulation is general enough to be applied to data modalities beyond text, like images [369, 26], videos [370], sequence of proteins [15], and even sets of particles [371, 372]. Attention enables neural networks to be effective on sequence-like data, equipping them with a mechanism that allows the model to leverage the information contained in sequences of non-trivial length⁵.

Formally, an attention mechanism can be generally described as mapping a *query* and a set of *key-value* pairs to an output, where the query, Q , keys, K , values, V , and output, Z , are all matrices⁶. In general, an attention mechanism involves the following steps:

$$\begin{aligned} S &= \varphi(Q, K) && (1. \text{ attention scores}) \\ A &= \rho(S) && (2. \text{ attention weights}) \\ Z &= \omega(A, V). && (3. \text{ compute output}) \end{aligned}$$

Here, the functions $\varphi(\cdot)$, $\rho(\cdot)$ and $\omega(\cdot)$ are generic to illustrate the concept: a precise choice of them defines various attention mechanisms as we will see shortly. Hence, an attention mechanism first computes *attention scores*, S_{ij} , each of them quantifying the importance of the query element at time index i in relation to the key element at time index j : the scores, S , is thus a matrix of shape $T \times T$, where T is the sequence length of queries and keys, respectively. The scoring function, φ , is responsible for computing the similarity of the time indices for the current query input w.r.t. the provided keys. According to the specific implementation, we can achieve different notions of similarity, for example:

$$\begin{aligned} \varphi(\cdot) &= \tanh(Q + K) && (\text{additive}) \\ \varphi(\cdot) &= QK^\top, && (\text{multiplicative}) \end{aligned}$$

implement two different mechanisms, the former called *additive* (or Bahdanau-style [367]) attention and the latter one is referred to *multiplicative* (Luong-style [368]) or dot-product attention.

Secondly, the application of the ρ function normalizes the scores (along the key-axis) into probabilities, A_{ij} , called *attention weights* or attention probabilities, which can be interpreted as, given a time index, at which timestep(s) the model is currently attending to. The attention scores are commonly transformed into probabilities by applying a softmax function over the rows of S such that:

$$A_{ij} = \frac{\exp(S_{ij})}{\sum_{t=1}^T \exp(S_{it})}, \quad (5.1.1)$$

⁵Long sequences, say with hundreds of steps and higher, were historically problematic in DL due to either exploding or vanishing gradients [147], which limits the model to extract long-term relationships.

⁶Apart from having a sequence length, queries, keys, and values are usually multi-dimensional; if not, these can be considered as being vectors.

which provides a “soft” weighting over keys, in the sense that all the keys contribute to the output but each is weighted differently according to A_{ij} . An alternative approach would be to restrict the model to only attend to the most important time step. This “hard” weighting can be achieved by placing zero probability everywhere except for the element that yields the highest score:

$$A_{ij} = \begin{cases} 1 & \text{if } S_{ij} = \arg \max_t S_{it}, \forall t = 1, \dots, T \\ 0 & \text{otherwise} \end{cases}. \quad (5.1.2)$$

This *hard* attention mechanism can be useful for applications in which the model should focus only on the prominent element rather than the entire sequence, as occurs in *soft* attention. Xu et al. [369] introduced the notion of stochastic hard attention, in which the soft attention weights (i.e., the ones computed by eq. 5.1.1) are used to parameterize a categorical distribution (see eq. 3.4.2) from which the index of the position to attend is sampled.

Lastly, the attention probabilities are usually used to weight the contribution of the values, resulting in outputs, $Z = AV$, obtained by matrix multiplication between A and V : intuitively the output at index i , i.e., the row vector $Z_{i\cdot}$, is computed by attending (i.e., performing a weighted summation) over all the keys’ time indices. If $\omega(A, V) = AV$, then *global* attention occurs because each output element is yielded by attending over the entire sequence of queries, in particular:

$$Z_{i\cdot} = \sum_{j=1}^T A_{ij} \cdot V_{j\cdot}, \quad (5.1.3)$$

resulting in $Z \in \mathbb{R}^{T \times D_V}$, where D_V is the dimensionality of each value element since $V \in \mathbb{R}^{T \times D_V}$. Alternatively, if we consider only a subset of the sequence, i.e., $\Delta < T$ to compute each output element, this results in

$$Z_{i\cdot} = \sum_{j=i-\Delta}^{i+\Delta} A_{ij} \cdot V_{j\cdot}, \quad (5.1.4)$$

in which each output vector, $Z_{i\cdot}$, is yielded by the weighted summation⁷ of only Δ value vectors. This is called *local* attention [368], which we may denote as $\omega(A, V; \Delta)$, allowing to trade computation cost for accuracy of the output representation.

5.1.2 Multi-head self-attention

At this point we discussed different aspects of the attention mechanism without explaining these queries, keys, and values. The values can correspond to word embeddings, image

⁷The computation of the attention weights, A , should be restricted to only Δ elements to get proper normalized probabilities over the time indices.

patches, or your input sequence in general. The keys can be like values but also represent the (current or previous) hidden state of the model. Lastly, likewise the keys, the queries can also depict the output at the previous step. Usually, the values, queries, and keys are all lower-dimensional projections (by learned matrices W_V , W_Q , and W_K) of the respective inputs.

Say $X \in \mathbb{R}^{T \times D}$ is an input embedding sequence (it can represent word tokens, image patches, and so on), an interesting question is to compare this sequence against its own, i.e., $Q = X$ and $K = X$, and use the resulting attention scores, S , to compute the output sequence which would weight against X , as $V = X$. This special case is called *self-attention* [373] because the attention matrix, A , is computed by measuring the similarity between locations in the sequence with the content of the sequence itself, which involves all the queries, keys, and values to be *different* linear projections of the input sequence, X : i.e., $Q = XW_Q$, $K = XW_K$, and $V = XW_V$; resulting in $Q, K \in \mathbb{R}^{T \times D_K}$ and $V \in \mathbb{R}^{T \times D_V}$ since $W_Q, W_K \in \mathbb{R}^{D \times D_K}$ and $W_V \in \mathbb{R}^{D \times D_V}$. In this way, projecting the inputs decouples the attention computation from the content length to the projection size. In addition, when computing the attention scores, asymmetries between queries and keys are introduced by using diverse projection matrices.

The self-attention mechanism described above utilizes only one attention matrix to capture the similarity (according to $\varphi(\cdot)$) of two locations within the sequence. This can limit the expressiveness of the model architecture since, according to the specific transformation learned by W_Q, W_K , and W_V , only one notion of similarity of projected inputs can be learned. By introducing H projections matrices, e.g., $W_Q^{(1)}, \dots, W_Q^{(H)}$, for all the values, keys, and queries, and then applying self-attention in parallel we can learn H attention matrices each of which captures a different notion of similarity within the input sequence. This generalized mechanism is called *multi-head* self-attention (MHSA) [373], since the application of self-attention given a set of matrices $\{W_Q^{(h)}, W_K^{(h)}, W_V^{(h)}\}$ is termed a *head*. In MHSA the output function, ω , now concatenates over the attention heads [13] which are then projected back to the output dimension of the input sequence X , i.e., D :

$$\omega(\cdot) = \left[V^{(h)} A^{(h)} \mid h = 1, \dots, H \right] W_O, \quad (5.1.5)$$

where the attention weights for head h are yield by $A^{(h)} = \rho(\varphi(Q^{(h)}, K^{(h)}))$ ⁸. In particular, the output projection matrix, W_O , has shape $HD_V \times D$ since the heads are concatenated along the feature axis, i.e., $[V^{(h)} A^{(h)}] \in \mathbb{R}^{T \times HD_V}$. The above operation results in a new output sequence $Z \in \mathbb{R}^{T \times D}$.

⁸To keep the notation light, we can define the projected queries, keys and values for head h as: $Q^{(h)} = W_Q^{(h)} Q$, $K^{(h)} = W_K^{(h)} K$, and $V^{(h)} = W_V^{(h)} V$.

5.2 Generative pre-trained transformers

The generative pre-trained transformer (GPT) [7] is a family of models based on the Transformer [13] architecture, which was originally designed to solve language tasks by providing it with input and output sequences of word tokens. A *token* is just an integer index, i , corresponding to a word, w_i . The largest index defines the vocabulary or code-book size, representing the number of unique words we intend to use to solve a given language task. Since indices alone are not useful for learning a suitable word representation, the tokens are usually embedded in high-dimensional spaces through (an often learned) *embedding table* in which each index is mapped to a unique high-dimensional vector. Therefore, a piece of text is just a sequence of tokens, which are individually embedded and then processed by the model to solve the target task: e.g., classification, translation, and generation, to name a few.

Apart from its architecture, the power of GPT as a language model is provided by the joint combination of scale, pre-training, and fine-tuning as we will see in the following sections.

5.2.1 Decoder-only transformer architecture

Transformer block. The basic component of the transformer architecture is the *transformer block* [373], which comprises two stages: the attention over the sequence, with a successive refinement of the features representation. The first stage is represented by *scaled dot-product multi-head self-attention*: a particular form of global self-attention that utilizes multiple heads each of which employs multiplicative attention to compute the attention scores. For a generic head, h , the scaled dot-product self-attention can be written as

$$\text{softmax}\left(\frac{Q^{(h)}(K^{(h)})^\top}{\sqrt{D_K}}\right)V^{(h)}, \quad (5.2.1)$$

where $\sqrt{D_K}$ is a scale factor intended to improve the numerical stability of the softmax operation, especially when the dot product between queries and keys grows large in magnitude due to high values of D_K [13, 373].

The MHSA is then followed by a second stage of processing that operates across features by applying a MLP (usually a two-layer feed-forward NN) to each time index separately and identically. Before each stage, layer normalization [374] is applied to each token separately (i.e., along the feature dimensions) removing the mean and dividing by the standard deviation. The two stages are also connected via residual connections [23], which induce the model towards learning simpler functions stabilizing training as well [150]: the overall design of the transformer block is presented in figure 5.2.

Positional encoding. Since the attention mechanism is performed on each time index of the sequence, considering the sequence as a whole each time to score and weight the values, the resulting output, Z , can be seen as a *set*: if we permute the input sequence (or the

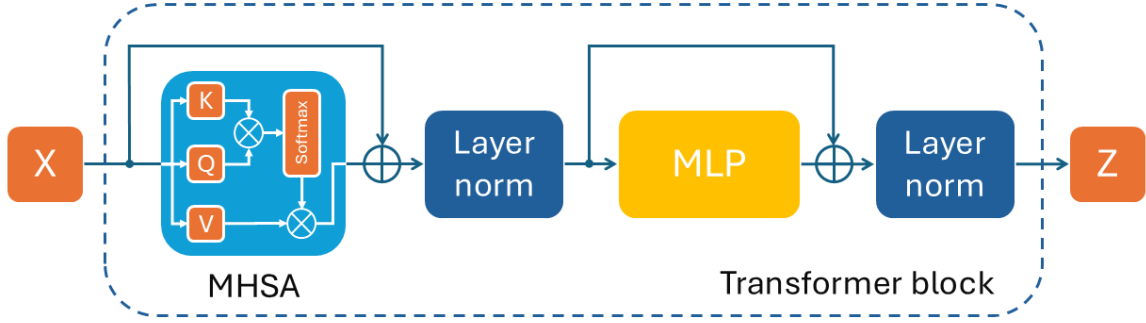


Figure 5.2: The Transformer block with simplified schematics of the multi-head self-attention. The inputs, X , can be the sequence embedding or the output, Z , of the previous transformer block. The symbol $\oplus$ denotes a skip connection and $\otimes$ matrix multiplication.

queries, keys, and values, identically), the resulting representation would be permuted in the same way. This *permutation invariance* property can be leveraged for applications in which the data has no natural ordering (e.g., a set of particles) but is detrimental, instead, for problems in which the order of the sequence matters such as language modeling⁹. In this case, positional information is explicitly paired with the tokens, forcing the sequence to be ordered. Therefore, relative or absolute token positions are added to the input embeddings with a *positional encoding* that can be either fixed (e.g., sine wave [13]) or learned (as in BERT [353]).

Decoder-only transformers. The transformer architecture or Transformer [13] in short, is just a stack of multiple attention blocks, where, at the input level, the token embeddings and their positional encoding are summed together and then processed by the various blocks. Former language translation models [367, 368] used to have an *encoder-decoder architecture*, in which: the encoder is responsible for first mapping the input sequence to a continuous representation, letting the decoder generate output tokens, one at a time, given the encoder’s output. The generation of the next token is *auto-regressive* [375], i.e., the prediction of the token at time t is based on the previous representations $Z^{(t-1)}, \dots, Z^{(1)}$. Therefore, to prevent the decoder from attending future positions a *causal mask*¹⁰ is multiplied to the attention matrix, A , such that the masked positions do not contribute to the results.

It turns out the encoder-decoder architecture can be simplified to a decoder-only transformer [376], with a notable saving in parameter count, just by combining the input and output sequences of tokens separated by a special token. In this way, by training the model to predict the next token (or word) the error signals are propagated from both input and

⁹In language, the meaning of a phrase can be greatly altered just by swapping two words. Therefore, we want our model to capture the phrase meaning as a function of the tokens ordering.

¹⁰A causal mask is an upper-triangular matrix made of zeros and ones, designed to prevent future information from influencing the past: a prediction at time t can depend only on previous outputs from time $t - 1$ to 1.

output timesteps avoiding the need for an explicit encoder network. As just mentioned, the training objective is to predict the next word given the previous ones (since the model is still auto-regressive), therefore maximizing the likelihood of the combined input:

$$p(w_1, \dots, w_{T+T_o}) = \prod_{t=1}^{T+T_o} p(w_t \mid w_1, \dots, w_{t-1}), \quad (5.2.2)$$

where T is the sequence length for the inputs (embeddings or tokens), x , and T_o the size of the output sequence, y , which are merged into a single one, e.g., $(w_1, \dots, w_{T+T_o}) = (x_1, \dots, x_T, y_1, \dots, y_{T_o})$; the notation should be interpreted for a single pair of input-output sequences, not for a batch of them.

5.2.2 Scaling laws

The transformer architecture captures long-range interactions and dependencies among tokens in the same sequence. It is already so effective that improving the whole model can be achieved simply by increasing its *scale*, avoiding the need to devise sophisticated architectures or attention variations. Scaling up the model size, dataset, and compute, can be achieved rigorously and predictably by following the so-called *scaling laws* [377]. A scaling law is an equation explaining a specific scenario, e.g., test loss given compute budget or accuracy as a function of dataset size, discovered by fitting a low-degree polynomial function to the empirical data collected by training multiple models.

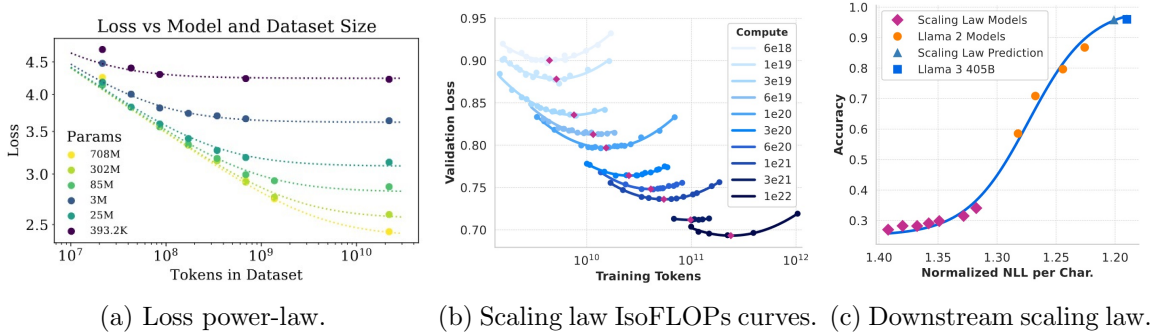


Figure 5.3: Example scaling laws. Left: the test loss varies as a function of model and dataset size, according to equation 5.2.3. Middle: the measured loss fit with a second-degree polynomial, whose minimum represents a compute-optimal model; the resulting law is represented by equation 5.2.4. Right: compute-optimal models can be used to build a scaling law to forecast the performance of the downstream task. Figures from [377, 355].

Usually, the scaling law is first built by considering hyperparameters in a restricted range (which are not of direct interest) to be applied to the values of interests, forecasting their impact thanks to the polynomial fit. The test loss is often the quantity of interest

to be predicted either as a function of model parameters¹¹, N , dataset size (or number of tokens), D , or the compute budget, C . When one of the parameters is studied in isolation, the other two are assumed to be infinite (or reasonably large). Moreover, N , D , and C can be jointly studied to understand their impact when scaled together: figure 5.3a. An example power-law that predicts the test loss, L , as a function of both N and D is the one discovered by Kaplan et al. [377]:

$$L(N, D) = \left[\left(\frac{N_c}{N} \right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D} \right]^{\alpha_D}, \quad (5.2.3)$$

in which N_c , D_c , α_N and α_D are empirical fitted values for the above trend. By discovering scaling laws many insights can be gathered, for example, [377]:

- When jointly scaling N and D , an eight-times increase in model size requires enlarging the dataset about five times to avoid detrimental performance.
- Large models are more sample-efficient than small models, reaching the same level of performance with fewer optimization steps and using fewer data points.
- When the compute budget, C , is fixed, optimal compute-efficient training can be attained by stopping large models significantly short of convergence.
- Given a fixed compute budget, if the model size is between around $0.5\times$ and $2\times$ the optimal size, the gap in performance compared to the optimal model can be met by a 20% increase in computation.

Scaling laws that only predict the pre-training loss are limiting because large language models are supposed to be used and evaluated on several downstream tasks. Recent work [355] built scaling laws for downstream tasks by correlating the performance of the compute-optimal model and the training FLOPs. The first step is to construct scaling laws for the validation loss. To do so, a set of models was trained assuming a given compute budget. Then, the measured loss values were fit using a second-order polynomial to identify the minimum point of each training scenario, called a *compute-optimal* model: figure 5.3b. The points corresponding to compute-optimal models, were subject to a second fit to identify the coefficients, (α, A) , of the following power law:

$$N^*(C) = AC^\alpha, \quad (5.2.4)$$

which predicts the optimal number of training tokens, N^* , as a function of the computational resources, C . Next, the models found using the above law were used to establish a sigmoidal relation among the log-likelihood, accuracy on the task, and training FLOPs: figure 5.3c. In principle, doing so allows us to predict model performance on the desired downstream task before pre-training. This result is important as it avoids wasting resources in case of unsatisfactory predicted performance.

¹¹Usually, it is considered the non-embedding parameter count, i.e., the number of model parameters excluding the ones for the embedding table.

5.2.3 Aligning to human preferences

Language models trained to predict the next word (or similar) are not expected to be *aligned* to our common sense. The reason is that these models are just trained to predict the most likely word, so when a query (or prompt) is provided to them these will generate the most probable text given the user query. Indeed, the likelihood of the predicted words is directly influenced by the quality of the training data. By this, I mean that if our training dataset contains harmful content then the LLM will generate content that looks novel but still dangerous. Of course, the need to build larger and larger datasets makes the role of data collection, text extraction, filtering, cleaning, and mixing [355], more and more valuable but also more challenging. Datasets of increasing size are likely to contain harmful, unsafe, biased, privacy-sensitive, low-quality, and duplicated content, therefore when the model is asked to learn the likelihood of the training words, the predictions will inevitably reflect those. However, even if we could clean the data perfectly, the generated content would still not follow our expectations (e.g., being untruthful, unhelpful) because the unsupervised nature of the training objective makes the model behavior uncontrolled.

In recent years, methods to *steer* (or control) the behavior of language models to be aligned with our preferences have been developed. The most successful of them [8, 378] is based on *reinforcement learning from human feedback* [360] (RLHF), in which a human-in-the-loop component is added to the standard RL paradigm (introduced in section 4.1). Human feedback quantifies the goodness and appropriateness of the responses generated by the model: these are, then, ranked and grouped by downstream task, to build datasets of human preferences employed for the alignment *post-training*. Preference queries are expected to be less expensive to collect compared to optimal, high-quality demonstrations of desired behavior. Moreover, human feedback also overcomes the difficulties of designing and eventually shaping a reward function.

Reinforcement learning from human feedback. The RLHF approach is general [360], applying to whatever problem in which designing a reward function or obtaining expert demonstrations is difficult, compared to collecting human feedback. Furthermore, human annotations can prove useful in avoiding unintended behaviors, and the exploitation of incorrect reward specifications or modeling by the agent. In the context of language models, the RLHF uses human preferences as a reward signal to fine-tune the LLM. The general pipeline is the following [8]:

1. Given a set of prompts, the human annotators provide high-quality demonstrations of the desired behavior on the input distribution. These demonstration data are then used to fine-tune a pre-trained language model via supervised learning: this phase is also called *supervised fine-tuning* (SFT), and is usually the first post-training step after model pre-training; the SFT LLM will provide baseline alignment performance.
2. A dataset of human-labeled comparisons between several outputs sampled from the same SFT model (i.e., the fine-tuned model obtained from the previous step) is collected. Since the language model learns a distribution over words or tokens, the

model is expected to generate slightly different content given the same prompt¹². These comparisons are then ranked from best to worst, and a *reward model* (RM) is next trained on this dataset to predict which model output would be preferred by the human annotators.

Given a pair of preferences, (y_a, y_b) , and a prompt, x , the Bradley-Terry (BT) model [379] models the human preference distribution, p^* , as follows:

$$p^*(y_a \succ y_b \mid x) = \frac{\exp(r^*(x, y_a))}{\exp(r^*(x, y_a)) + \exp(r^*(x, y_b))}, \quad (5.2.5)$$

where $r^*(x, y)$ is an unknown latent reward model, and the notation “ $y_a \succ y_b$ ” should be read as y_a is preferred over y_b . The above distribution can be approximated by solving a binary classification problem, minimizing the negative log-likelihood loss:

$$\mathcal{L}_{\text{RM}}(r_\phi, \mathcal{D}) = -\mathbb{E}_{x, y_a, y_b \sim \mathcal{D}} [\log \sigma(r_\phi(x, y_a) - r_\phi(x, y_b))], \quad (5.2.6)$$

where $r_\phi(x, y)$ is a parameterized reward model, σ the logistic or sigmoid function, and $\mathcal{D} = \{x^{(i)}, y_a^{(i)}, y_b^{(i)}\}_{i=1}^N$ a dataset of N comparisons.

3. Lastly, the RM provides the reward signal for the LLM alignment (or *instruction-following*) fine-tuning, often using the PPO [280] reinforcement learning algorithm. The RLHF fine-tuning represents a second post-training phase that solves the following optimization problem:

$$\max_{\pi_\theta} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_\theta(\cdot \mid x)} [r_\phi(x, y)] - \beta D_{\text{KL}}[\pi_\theta(y \mid x) \parallel \pi_{\text{ref}}(y \mid x)]. \quad (5.2.7)$$

The hyperparameter, β , controls the deviation from the reference policy, π_{ref} , corresponding to the SFT LLM obtained at step 1. The added constraint prevents the model from deviating too far from the distribution on which the reward model is accurate. In addition, it ensures generation diversity also preventing mode-collapse to single high-reward answers. From the above objective, a reward function $r(x, y) = r_\phi(x, y) - \beta(\log \pi_\theta(x \mid y) - \log \pi_{\text{ref}}(x \mid y))$ can be constructed and optimized via model-free RL, as PPO.

Direct preference optimization. RLHF has proven effective for aligning large language models such as GPT-3 [8] to user intents and instructions, also improving the response quality judged by humans, resulting in generally safer, more helpful and engaging LLMs. Despite the practical effectiveness, RLHF is a complex, computationally expensive, and often unstable procedure that requires careful hyperparameter tuning. Direct preference optimization [378] (DPO) is a valid alternative to RLHF that is stable, performant, and computationally lightweight, eliminating the need for fitting a reward model, sampling from the LLM during

¹²When sampling the next most likely word, a temperature parameter, τ , is often introduced to bias the generation between creativity ($\tau \rightarrow \infty$) or preciseness ($\tau \rightarrow 0$).

fine-tuning, or performing significant hyperparameter tuning. DPO leverages an analytical mapping from reward functions to optimal policies to directly optimize the language model on preference data through supervised learning instead of RL. DPO introduces a preference model defined in terms of the optimal and reference policies (respectively, π^* and π_{ref}), such that to directly seek an optimal solution to the KL-constrained objective in eq. 5.2.7:

$$p^*(y_a \succ y_b | x) = \left(1 + \exp \left(\beta \log \frac{\pi^*(y_a | x)}{\pi_{\text{ref}}(y_a | x)} - \beta \log \frac{\pi^*(y_b | x)}{\pi_{\text{ref}}(y_b | x)} \right) \right)^{-1}. \quad (5.2.8)$$

In this way, the optimal policy is designed to satisfy the above preference model and can be learned by solving a simple classification problem:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{x, y_a, y_b \sim \mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi^*(y_a | x)}{\pi_{\text{ref}}(y_a | x)} - \beta \log \frac{\pi^*(y_b | x)}{\pi_{\text{ref}}(y_b | x)} \right) \right], \quad (5.2.9)$$

which corresponds to minimizing a cross-entropy loss over K ranked completions, without any intermediate reward modeling. In practice, DPO is as effective as RLHF requiring less computation to align the language model properly. Given these characteristics, DPO is employed to fine-tune flagship LLMs such as the recent Llama 3 [355].

5.3 OmniJet, Masked Particle Modeling, and OmniLearn

In both high energy and particle physics, phenomena are usually studied with the aid of either or both real and simulated data: the former is usually limited in statistics also being unlabeled, and the latter ones can be endlessly produced with detailed information and labels as long as computational resources are available. Foundation models designed to handle particle physics data aim to mimic the success and effectiveness of large language models in the physics domain rather than natural language.

FMs in particle physics are expected to solve multiple related tasks (e.g., classification, anomaly detection, and generation) through learning a general representation that can be reused and shared across problems, datasets, and even experiments. For this end, a general and rich data representation is required: *point clouds* of particles are dubbed as a sufficiently general and comprehensive representation since collisions are naturally arranged as graphs, being unordered with varying-length elements. Then, a second desideratum of FMs is to be as sample-efficient as possible on downstream tasks for which the data statistics is scarce: being able to reach high performance on a task with limited samples is useful at both reducing the computing requirements (e.g., training time, storage) but also to compensate for the limited statistics of rare processes, in which only a small fraction of them might survive the pre-selection rules.

FMs for particle physics tasks are still in their early infancy, being a recent effort (e.g., [371, 372, 380]), whose complexity is significantly far from cutting-edge LLMs (e.g., GPT-4 and Llama 3). The following sections, provide an outline of these early FMs to the reader.

5.3.1 Pre-training objectives

As for language models, the pre-training phase aims to induce the physics foundation model to learn a general, unsupervised representation of the domain. As we will see, the approaches followed by [371, 372] complement each other, trying to employ the same pre-training objective utilized by effective LLMs.

Tokenization. Both works propose a transformer-based FM that learns from a fixed-length sequence of particles which, taken as a whole, constitutes a single jet. In both models, the first step is to discretize the continuous input features to the analogous of a token representation: i.e., an integer index representing a “word” in the language of jet physics rather than text. This process called *tokenization* (illustrated in figure 5.4), occurs before pre-training by training a *Vector-Quantized Variational Auto-Encoder* [381] (VQ-VAE) to discretize and then reconstruct the continuous features that describe a particle. The encoder first embeds the input constituents into a latent space, whose latent vectors are then rounded to the nearest discrete embedding, to be finally reconstructed by the decoder. We should notice that this additional step is not required when modeling natural language since the sequences are already composed of discrete units (the words), which only need to be associated with a unique index (the token).

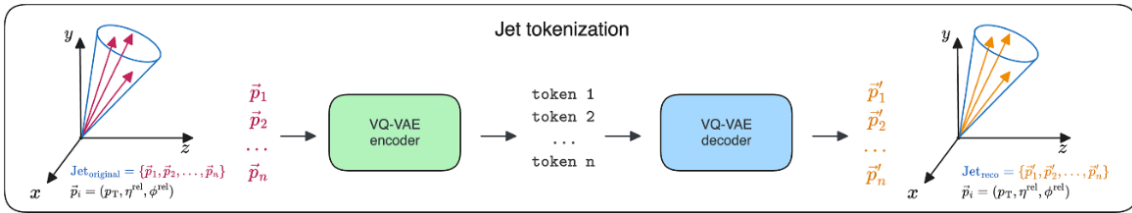


Figure 5.4: Jet tokenization: a jet represented as a set of 3-dimensional vectors is transformed to a sequence of discrete indices (tokens) by the VQ-VAE encoder, through a nearest neighbor search finding the closest discrete embedding. Conversely, the VQ-VAE decoder reconstructs a sequence of tokens to the physical space of jet features. Figure from [371].

In particular, Birk et al. [371] use only per-particle kinematic information, (p_T, ϕ, η) , pre-processed to be relative to the jet axis:

$$\begin{aligned}\eta^{\text{rel}} &= \eta^{\text{particle}} - \eta^{\text{jet}} \\ \phi^{\text{rel}} &= \phi^{\text{particle}} - \phi^{\text{jet}},\end{aligned}$$

where p_T is the transverse momentum, ϕ the azimuth angle, and η the pseudo-rapidity. These three features describe a jet constituent and, by design, a jet can be composed by up to 128 of them, which are also ordered by p_T in descending order. Hence, the VQ-VAE is first trained to quantize each $(p_T, \phi^{\text{rel}}, \eta^{\text{rel}})$ to a discrete token ID, then reconstruct the token ID back to physics space. In this way, a jet is mapped by the VQ-VAE encoder to a sequence of tokens, each of them depicting a single constituting particle: the number of

unique token IDs gives the *codebook* size, which is 512 in [372] and 8192 in [371]. Next, sequences of tokens (i.e., a batch of tokenized jets) are provided as input to the FM to (pre-)train its backbone.

The VQ-VAE represents a learned alternative to simple *binning tokenization*, where each input dimension is divided into bins. In this case, the codebook size is given by the product of the bins over each dimension, which grows exponentially, hence, suffering from the curse of dimensionality because, as the number of bins increases, less and less data will populate them. Conversely, the VQ-VAE can efficiently use its discrete codes, whose size is less sensitive to adding dimensions. Moreover, VQ-VAE tokenization can be either unconditional or conditional: *unconditional* tokens have a one-to-one correspondence since each particle is embedded individually, instead, *conditional* tokens are embedded from sequences, therefore one token can cover multiple parts of the feature space since being determined by the input sequence of constituents. Experiments by Birk et al. [371], demonstrate that conditional tokenization shows a better agreement with data at a lower codebook size.

After tokenization, the token IDs are individually embedded into high-dimensional vectors. In this context, the role of pre-training is to learn a general, transferable, and rich representation of the tokens. After pre-training, the learned token representation is extracted (usually kept frozen: i.e., there is no further adaptation) and employed for fine-tuning the downstream tasks of interest.

Next-token prediction. Auto-regressive sequence models usually optimize a next-token prediction (NTP) objective; often referred to as GPT-style pre-training. In a similar spirit, Omnijet- α [371] is pre-trained on jet generation, in which the FM predicts one token at a time, in a sequential manner, starting solely from a symbolic `start_token`, to generate a p_T -ordered list of particles making a jet. The predicted sequence of tokens is then reconstructed by the VQ-VAE decoder, to recover the physical space corresponding to token IDs. In practice, the NTP objective (see equation 5.2.2) is a multi-class classification task over token IDs, representing the class labels. During this generative pre-training, the model leverages causal masking to prevent attention from occurring on future token positions. Causal masking allows auto-regressive transformers to be computationally efficient during training since sequential predictions are replaced by a single forward pass. However, at the cost of introducing positional information [373]. This may be detrimental for domains where the sequence order is not defined, as in particle physics where the jet constituents are naturally unordered. Nevertheless, in this specific case, an ordering is still implicitly enforced by sorting the constituents according to their p_T . Therefore, the model will learn to generate particles with decreasing p_T until a special `stop_token` is predicted or the maximum sequence length is reached.

Masked input prediction. An alternative to the NTP objective is provided by the BERT-style pre-training, in which some parts of the input sequence of tokens are first randomly masked, i.e., zeroed, then fed to the model whose role is to predict the missing tokens. Masked particle modeling (MPM) [372], pictured in figure 5.5, represents jets as

sets of unordered particles, where some of them are masked during pre-training: similar to the masked language modeling objective [353]. Partitioning the particles sequence, x , into masked, $\mathcal{M}^x$, and unmasked, $\mathcal{U}^x$, sets of token ids, the backbone is trained to minimize the following per-particle loss

$$\mathcal{L}_{\text{MPM}}(\mathcal{D}; \theta) = \mathbb{E}_{x \sim \mathcal{D}} \left[\frac{1}{|\mathcal{M}^x|} \sum_{p \in \mathcal{M}^x} \mathcal{L}(x_p, f_\theta(\mathcal{M}_p^x, \mathcal{U}_p^x)) \right], \quad (5.3.1)$$

where p denotes the particle index within the input sequence, such that the sets $\mathcal{M}_p^x$ and $\mathcal{U}_p^x$ yield the indices of the masked (and corresponding original) features for particle p ¹³. Moreover, the term $f_\theta(\mathcal{M}_p^x, \mathcal{U}_p^x)$ denotes the model's predictions of the missing elements, which is subject to a categorical cross-entropy loss.

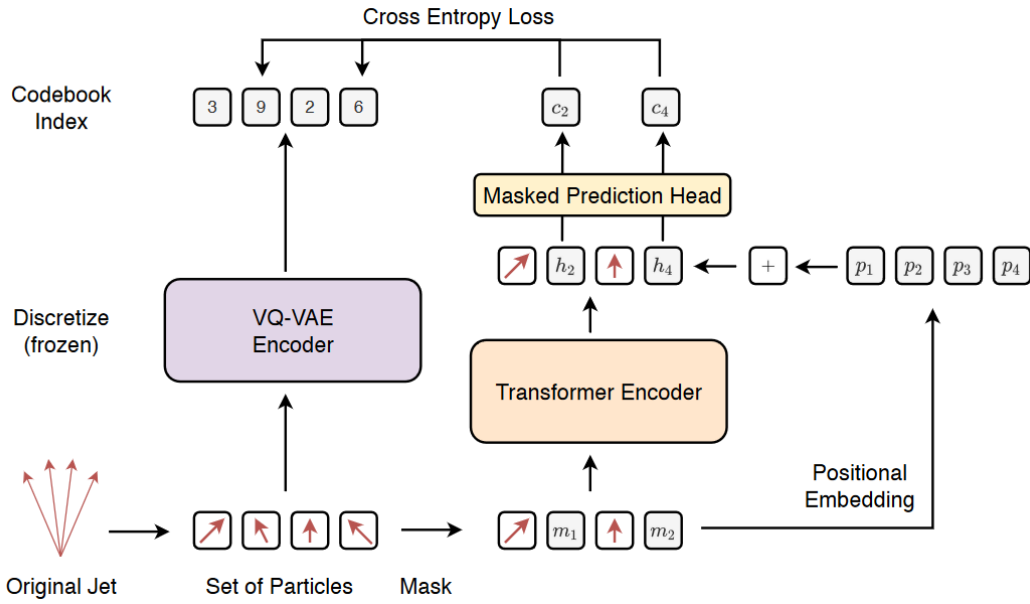


Figure 5.5: Masked particle modeling: the backbone predicts the missing token IDs from a masked set of particles, whose ground truth for learning is provided by the VQ-VAE encoder applied on the original jet. Figure from [372].

Compared to Omnijet- α , MPM pairs positional information to the tokens to account for a limitation of their pre-training objective: since the particles that form a jet are permutation invariant, and the transformer-encoder backbone is permutation equivariant by design¹⁴, then [372]:

if the same masked token is used to replace all of the particles in the masked set, then the output of the model will be equivalent at every masked position.

¹³For example, masked particles can be represented by a special `mask_token`.

¹⁴In general, equivariance provides a way to encode symmetries in a neural network model.

This is because permutation equivariant functions cannot distinguish between two input particles with the same values. Therefore, this redundancy can be removed by inducing an ordering, which can be direct (e.g., by sorting particles) or indirect (e.g., by providing a feature that relates two or more particles). Since inducing an ordering breaks the permutation invariance of the backbone for all downstream tasks, this may degrade predictive performance.

Multi-task objective. Differently from both OmniJet- α and MPM, the authors of OmniLearn [380] follow a MTL [95] approach (see section 2.3.7) in which the model is pre-trained to classify and generate jets jointly. In this way, assuming the model is flexible enough and that the two objectives are not interfering with each other, the model is expected to learn a representation that is suitable for both pre-training tasks. In particular, OmniLearn employs a point-edge transformer as the backbone, which learns a proper representation from the kinematic information of each particle along with an additional particle identification (PID) feature. Moreover, jet features (e.g., mass, η , p_T , and particle multiplicity) are introduced to the classifier head. Lastly, the generator head is a diffusion generative model [382] that simulates jet types from input particle embeddings and jet kinematic information, conditioned on class labels. In a few words, the diffusion process turns a structured signal (e.g., an image, video, etc) into noise, in multiple steps. By simulating this process, it is possible to generate noisy samples from the training set, which can be later denoised by a neural network trained to revert the diffusion process. Hence, we can generate new, realistic samples from just random noise.

5.3.2 Leveraging knowledge transfer

As mentioned in the introductory section, the most appealing capability of foundation models is the transfer of the representation learned during pre-training to multiple, different but still related downstream tasks, such that solving the task of interest can be done efficiently (i.e., be data-efficient) and effectively (i.e., achieving high performance.) In principle, we want to transfer the knowledge acquired from bigger datasets to smaller ones, since this would allow us to reduce the data collection, labeling, or generation/simulation effort by at least a few orders of magnitude (ideally, at least $100\times$ less) while achieving state-of-the-art performance on them, compared to either having access to a much larger dataset or simply training the model on the limited dataset size from scratch.

A prerequisite for transfer learning is that there should be a sufficient relation between the source and target domain or task, otherwise knowledge transfer could be unsuccessful [351]. In some cases, the transfer learning method may worsen the performance on the target task: if so, *negative transfer* [383] occurred. Factors like the degree of relevance between domains/tasks, the amount of labeled data, and the capacity of the model to find the transferable and beneficial part of the knowledge across domains, influence whether the transfer would be positive (successful) or negative. Moreover, transfer learning is said to relax the assumption for which the pre-training data (source domain) and the test data

(target domain) are i.i.d. [350]. This is especially relevant when the model is pre-trained on one dataset and then fine-tuned on a different one.

Following [351], a *domain*, $\mathcal{D} = \{\mathcal{X}, P(X)\}$, can be defined in terms of a feature space, $\mathcal{X}$, and the marginal distribution, $P(X)$, on an instance set (or dataset), $X = \{x \mid x_i \in \mathcal{X}, \forall i = 1, \dots, n\}$. Then, a *task*, $\mathcal{T} = \{\mathcal{Y}, f\}$, consists of a label space, $\mathcal{Y}$, and an implicit (usually unknown) decision function, f , which is expected to be learned from the sample data. Consequently, given observations from both the source, i.e., $\{(\mathcal{D}_S, \mathcal{T}_S)_{i=1}^N\}$, and the target domain(s) and task(s), i.e., $\{(\mathcal{D}_T, \mathcal{T}_T)_{j=1}^M\}$, transfer learning utilizes the knowledge implied in the source domain(s) to improve the performance of the learned decision functions, f_{T_j} , for each target task, T_j . In the context of FMs, each function f_{T_j} can be seen as a backbone fine-tuned on task T_j .

According to [350], transfer learning approaches can be categorized into four groups: *instance-based*, in which a portion of the source instances are inserted into the target domain, being appropriately weighted; *mapping-based*, in which the feature space of both source and target domains are transformed to a new data space; *network-based*, where a portion of the pre-trained network (including its structure and weights) is reused in the target domain; lastly, *adversarial-based* methods try to learn a representation that unifies the source and target domains, i.e., from which it is not possible to discriminate whether the domain is source or target.

Given the previous categorization, we can view foundation models as a unification of transfer learning approaches: FMs are inherently network-based, in which the pre-training objective (i.e., the source task) tries to learn a unified (similar to mapping-based) representation that well generalizes to the target domain(s) (as in adversarial-based methods, which can also be interpreted as regularization), with little or no instance reweighing.

5.3.3 Performance on downstream tasks

Recall that foundation models are often pre-trained on a task of no direct interest, for example, a self-supervised surrogate task (e.g., predict the next token or masked parts of the input) on a large (potentially unlabeled) dataset, whose only purpose is to learn a representation of the input data (jets in this case) that enables positive transfer on a variety of downstream tasks (i.e., the tasks of interest). In the context of HEP, the prominent downstream tasks are: *classification*, *generation*, *anomaly detection*, and *unfolding*. Indeed, other tasks are conceivable such as event classification, particle tagging, clustering, and regression, to name a few.

According to the backbone architecture, solving the mentioned problems requires training task-specific heads while fine-tuning the backbone. Since FMs are often large requiring non-trivial computing and memory resources, to maximize the practical usefulness of the backbone, it can be kept fixed (or frozen) during fine-tuning on the downstream task. In this way, both memory and computation required by back-propagation are drastically reduced (notice the inference cost is unaltered), being only a function of the complexity of the task-specific head since the backbone’s parameters are not learned. This strategy is often referred to as “fixed backbone” and, compared to the full fine-tuning in which both the back-

bone and the head are learned, it is expected to yield lower transfer learning performance. Therefore, reducing the gap between full and fixed fine-tuning may enable easier sharing and broader utilization of the pre-trained weights within the particle physics community.

Classification. The task of separating the signal(s) from the background is called classification or *tagging*. Chapter 2 provided a broad introduction to this supervised learning task, focusing on the special case in which the classifier is conditioned on one or more physics parameters. In principle, such a parameter-conditional model can be employed as a classification head for the FMs. Solving this task with classical transfer learning first involves pre-training on a large and varied classification dataset (e.g., ImageNet [358]), then fine-tuning the pre-trained classifier on the target dataset. On the other hand, the foundation model approach relies on the generalization induced by the unsupervised pre-training task, often avoiding this double-classification training.

In this case, we focus on jet tagging since the previously introduced FMs receive a jet (represented as a sequence of tokens) as input. In particular, all the three FMs considered in this thesis are pre-trained on the same dataset: JetClass [384], which contains 100M jets divided into ten categories. According to [372], classification experiments can be categorized as *in context*, when using the same dataset and same classes as in pre-training; *out-of-context*, when the dataset is the same as in pre-training but the fine-tuning occurs on classes not seen before; and *out-of-domain* when the pre-training and fine-tuning datasets are different potentially including a distribution shift: this scenario is closer to real-world usage of FMs, since we usually want to fine-tune our models on datasets that are diverse from the pre-training one(s).

Given so, OmniJet- α demonstrates very effective in-context performance for the task of $t \rightarrow bqq'$ vs q/g jet classification¹⁵, which are two classes of JetClass, achieving a considerable gain in accuracy even when fine-tuned on only 100 jets. Then, MPM demonstrates improved performance for most data regimes (i.e., number of training jets) across the various in-context, out-of-context, and out-of-domain tasks. Lastly, OmniLearn showcases generalization across jet types, detector responses (e.g., ATLAS [229] and CMS [43]), and collision systems as well. In the latter case, the model was pre-trained on proton-proton collisions and fine-tuned on electron-proton ones.

Generation. Both OmniJet- α and OmniLearn are pre-trained with a generative objective, therefore both models can naturally generate jets. In the former case, the generation of jets is achieved by first predicting the next token and, then, the predicted sequence is reconstructed to jet constituents by the VQ-VAE decoder: since auto-regressive, the model initially receives a special `start_token`, generating until the `stop_token` is predicted. Conversely, the latter model generates jets through a diffusion process controlled by a time parameter.

¹⁵JetClass includes jets initiated by quarks, q , gluons, g , and the W or Z vector bosons. The different processes are also distinguished by whether the decay chain contains a lepton ℓ , bottom quark b , or charm quark c [372].

In this context, generation-to-generation experiments are conceivable, in which the fine-tuning task requires generating novel kinds of jets that differ from the pre-training ones. In particular, OmniLearn is fine-tuned for conditional jet generation in which the model learns to generate jets based on particle information while being conditioned on four-momentum information $(p_T^{\text{jet}}, \eta^{\text{jet}}, \phi^{\text{jet}}, m^{\text{jet}})$.

Anomaly detection. The task of whether a sample is anomalous or not has been extensively discussed in chapter 3. Since FMs are expected to be trained on large and comprehensive datasets, therefore encoding the knowledge of “known physics”, it is reasonable to evaluate their anomaly detection performance to monitor how this capability improves as progress evolves. In particular, both MPM and OmniLearn focus on weakly supervised downstream tasks, in which the class labels are assumed to be noisy [118]. Moreover, both approaches employ the *significance improvement characteristic curve* (SIC), defined as the signal efficiency divided by the square root of the background efficiency, to evaluate how sensible their models are to potential new physics. The SIC represents a multiplicative factor by which the initial significance of a signal would increase when a particular threshold of the classifier output is chosen [380]. Therefore, the large pre-training phase is expected to amplify this significance in FMs compared to conventional approaches. Results from both authors show interesting performance, even when the number of injected signal events is limited.

Unfolding. Measured observables are subject to *detector effects*, in terms of limited resolution and efficiency, resulting in measured event properties, say y , that are different from what would be expected for an ideal detector: we want to measure the quantity x but, due to these effects, a related quantity, y , is obtained instead. The process of extracting the truth information from the observed measurements is called *unfolding* [128], which is mathematically defined as:

$$g(y) = \int A(y, x) f(x) \, dx, \quad (5.3.2)$$

where $g(y)$ is the measured distribution, $A(y, x)$ is the resolution function that describes the response of the detector, and $f(x)$ is the distribution we wish to recover. The resolution function or matrix, A , is not known analytically but is often represented by a dataset of MC simulations, which may include further statistical uncertainties.

In this regard, OmniLearn is compared to OmniFold [385] achieving similar or even superior performance when fine-tuned: it was trained with a similar iterative setup, in which the data reweighting function is obtained through a classifier.

5.4 Leveraging real data in a foundation model

This section presents an early application of the OmniJet- α foundation model, in which the backbone is pre-trained on real, open data collected by the CMS experiment during two

periods in 2016 rather than simulated jets, and fine-tuned on JetClass¹⁶.

5.4.1 Generative pre-training

The pre-training dataset consists of more than 170M high p_T jets originated by proton-proton collisions collected by CMS during two runs in 2016 [387, 388]. The dataset is *unlabeled*, consisting of mostly real-world QCD jets with other, unknown minor contamination, e.g., top jets. OmniJet- α can leverage large unlabeled datasets for the pre-training phase since the objective is auto-regressive, which does not require class labels to train the backbone, unlike the supervised classification task in OmniLearn. This fact can be valuable for foundation models since allowing us to leverage existing datasets of real-world collisions directly, rather than simulating events for both signal and background processes.

Following the setup and architecture of OmniJet- α , the backbone was pre-trained optimizing the next-token prediction objective, a generative task yielding novel jets, for two data regimes (i.e., number of jets): 10M and 170M; the former regime serves as a fair comparison for the fine-tuning presented in the next sub-section. Figure 5.6 shows the jet generation capability of the pre-trained backbone. In particular, we can see a good agreement between the data reconstructed and generated distributions, in which the predicted jets only show an overall slightly smaller p_T .

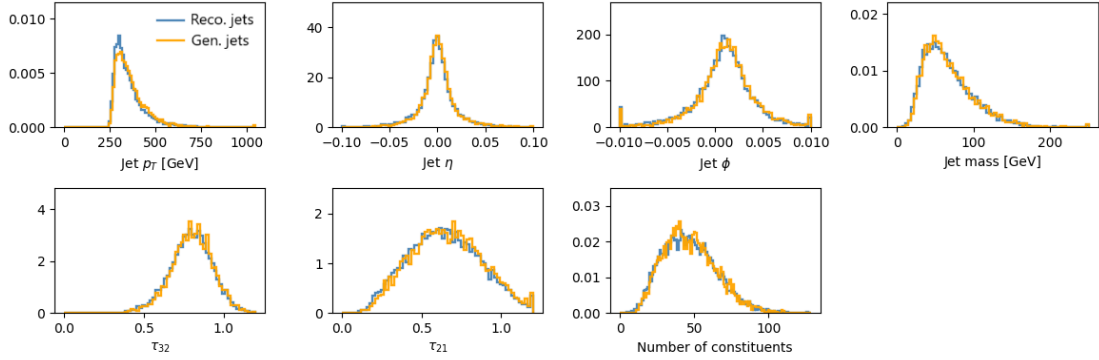


Figure 5.6: Jet-level feature distributions of the generated jets after pre-training on 170M jets: τ_{32} and τ_{21} represent substructure information; results for the 10M training are omitted because similar. The blue distributions (**Reco. jets**) are the ground-truth tokens reconstructed by the VQ-VAE decoder, subject to some resolution approximations compared to the original data distribution. Instead, the orange histograms (**Gen. jets**) are the predicted token sequences reconstructed by the VQ-VAE decoder.

¹⁶Further developments of the exploratory work presented in this section are available in [386].

5.4.2 Classification fine-tuning

The representation learned during pre-training is extracted for subsequent fine-tuning: the task is now to train a classification head to distinguish QCD (i.e., q/g) from top-quark (i.e., $t \rightarrow bqq'$) jets, utilizing the JetClass dataset. Before undertaking the fine-tuning, we need to assess how much the target task relates to the pre-training one. This is important because transfer learning on unrelated tasks will result in negative transfer. Therefore, to leverage the pre-trained weights effectively we always need the fine-tuning task to be relevant to some degree. An empirical way to do so is to quantify the domain shift between the two tasks, e.g., by inspecting the feature distributions as shown in figure 5.7, where we can notice that: the AOJ samples are closer to the QCD ones; the jet p_T is similarly shaped although translated; lastly, the $t \rightarrow bqq'$ jets have a distinct mass peak. These differences in distribution are non-trivial, constituting an interesting learning task.

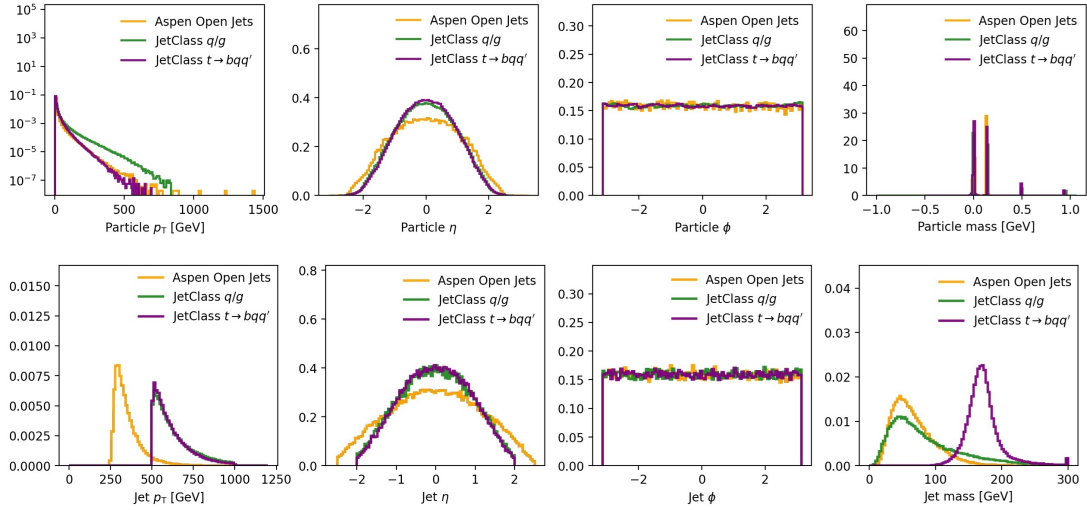


Figure 5.7: Comparison of particle (first row) and jet (second row) features distributions of the AOJ and JetClass datasets (but restricted to the two considered classes).

To quantify the improvement in data efficiency brought by the pre-training phase, compared to a *from-scratch* training in which the backbone is initially random, we fine-tune according to seven data regimes: 100, 1k, 10k, 100k, 1M, 2M, and 20M training jets. During training the backbone’s pre-trained weights are also adjusted, providing an additional performance improvement: this scheme is known as *full* fine-tuning, in which the backbone is not held fixed. Figure 5.8 presents the obtained results comparing the pre-trainings on open jets (green) with JetClass (blue), for two classification metrics: the accuracy and the R30. The accuracy is defined as $\frac{TP+TN}{P+N}$, and the R30 can be seen as the TPR at the 30% of FPR: recall sections 2.4.2 and 3.2.3. In particular, the 170M open jets pre-training is confronted to the original OmniJet- α setup [371] (i.e., 20M JetClass QCD + $tbqq$), and the performance obtained on the 10M dataset size is compared to an equivalent JetClass

size (i.e., 10M JetClass QCD). In general, we can observe that:

- All pre-trained models can learn much more efficiently than the from-scratch training. In particular, considering the red curves, we can appreciate the reduction in the required number of training jets to achieve the same performance, reaching even two orders of magnitude (by moving horizontally along the plots.)
- Pre-training on either 10M QCD or 10M open jets (dotted blue and green lines) results in similar fine-tuning classification performance.
- The 20M QCD+Tbqq pre-training is always more accurate than the 170M pre-trained backbone. This is reasonable since during pre-training the representation can also generate top jets, which are the classification targets. Instead, the 170M jets, although more numerous, are expected to be mostly QCD. Moreover, when fine-tuning using the full JetClass size (i.e., 20M jets), the performance of the two models is close.
- The 170M AOJ backbone is superior to the 10M AOJ one, suggesting that a larger pre-training dataset helps to learn a representation that is more efficient and performant, at least, for classification tasks.
- All the models (excluding the from scratch one) achieve very similar classification performance when trained on the largest amount of jets (i.e., the 10^7 point in the plots.)

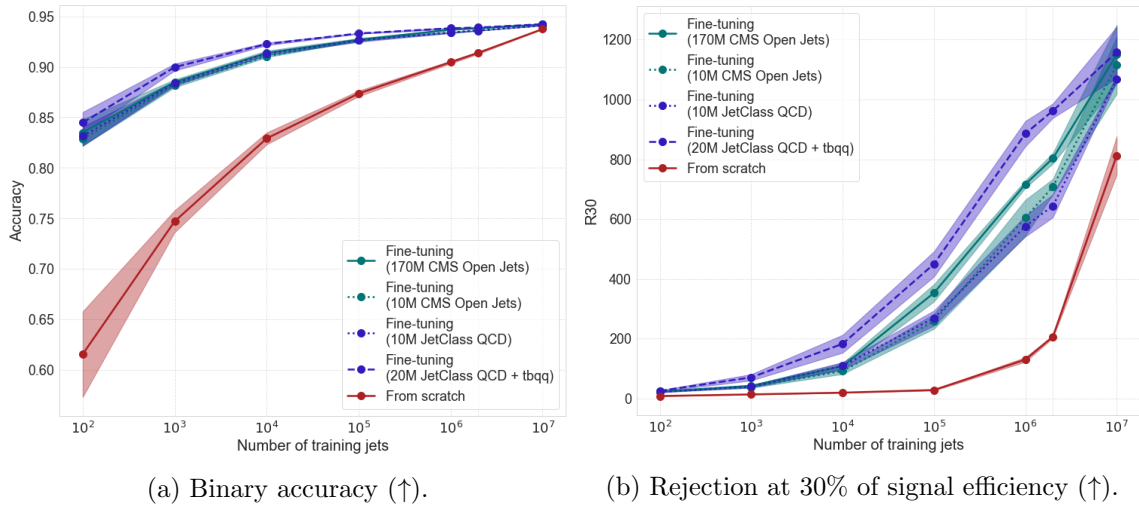


Figure 5.8: Fine-tuning evaluation of the pre-trained backbone on the QCD vs top-quark classification task. The name in parentheses within the legend indicates the pre-training dataset. In addition, the from-scratch training is provided as a baseline. The uncertainty bands include the results within one standard deviation, computed from five training runs.

Leveraging large unlabeled datasets of real-world collisions can further improve the FMs' capabilities through pre-training. Real data is different from its simulated counterpart, being more accurate, mixed (in the sense that the dominant background is contaminated by other processes, in an unknown proportion), and showing effects and uncertainties typical of the detector which are usually not simulated. Hence, learning directly from the real data could boost signal sensitivity, especially for downstream tasks, like classification, anomaly detection, and density estimation, which may benefit from increased robustness towards real data.

5.5 Future research directions

Further developments of foundation models for HEP may lead to a breakthrough in the field since a single backbone would at the same time: 1) improve performance on all the downstream tasks of interest; 2) save human and compute resources: superior data-efficiency enables the fine-tuning on small datasets while reducing the need of large simulations; 3) allow the sharing of models across communities and collaborations instead of data, which may be private and protected by intellectual properties; and 4) democratize the access to latest research: ideally, the backbone would be only trained once, which is expensive requiring a computing cluster, whereas the task-specific heads can be fine-tuned with orders of magnitude less compute, being trainable alone even on commodity hardware.

To achieve such an important goal, it is crucial that research efforts regarding FMs address the following open questions:

- **HEP-specific scaling laws.** Similarly for LLMs (section 5.2.2), scaling laws should be devised for HEP such that the amount of data, model parameters, and the pre-training loss can be accurately modeled and predicted. Doing so would enable allocating specific training budgets to achieve the desired performance, given a training regime (e.g., number of data samples or tokens, and model parameters). Moreover, as in recent work [355], scaling laws could be devised to predict the performance on prominent downstream tasks, unlocking even more predictability of the FM in development. A first step towards this goal is the power-law scaling observed during generative fine-tuning [386], where a power-law $D^{-\gamma}$, expressed in terms of the training sample size, D , accurately predicts the downstream performance.
- **Fixed backbone fine-tuning.** As demonstrated by [371, 372, 380], fine-tuning both the backbone and the task-specific head leads to the best performance on the target task. However, as the backbone is scaled up, fine-tuning can become prohibitively expensive for users and researchers since the computational requirements would grow accordingly. Developing lightweight fine-tuning strategies, in which most or even all the backbone's parameters are fixed, could significantly lower the computational and memory requirements to fine-tune the FM. Ideally, the computational resources should be lower than full fine-tuning, while achieving comparable performance. In this way,

the backbone could be trained once and shared across the research community, making it accessible to small and large institutions.

- **Data representation and multi-modality.** HEP data can describe various physics objects (e.g., particles, jets, and showers), include multiple levels of information from the detector apparatus (e.g., calorimeter, tracking, or full-event information), and high-level observables as well. Moreover, it can appear in multiple modalities such as vectors, sets, images, and graphs. Each modality is advantageous in specific scenarios, therefore to leverage existing workflows, datasets, and architectures, FMs could adopt a multi-modal approach similar to recent LLMs [354, 355] that combine text with vision capabilities. It would be interesting to explore the *encoder* and *adapter* approach employed by Llama 3 [355] that add multi-modal capabilities to LLMs without impacting the knowledge learned on the main modality, which is text.
- **Impact on new physics searches.** FMs aim at improving a wide range of downstream tasks by leveraging the data representation learned during pre-training. In principle, the same backbone can be fine-tuned to perform simulation, density estimation, anomaly detection, classification, regression, unfolding, and so on: tasks that can all influence (directly or indirectly) the quality of subsequent new physics analyses. Therefore, quantifying the effect of FM usage in either toy or real-world searches could help us better understand the impact and usefulness of this new paradigm for large model development.
- **Standardized benchmarks.** For fair evaluation across HEP foundation models, and to track their progress, it is necessary to agree on a given set of downstream tasks and pre-training datasets. In this way, it would be easier to understand the impact of the pre-training scheme, model architecture, hyperparameters, and scale, on the fine-tuning tasks. Although the JetClass dataset [384] is employed by prior work [371, 372, 380] for pre-training their FMs, wide disagreement is present among downstream tasks regarding both their typology and dataset.
- **Safety.** Defects, memorization, and biases arising in the backbone, due to low-quality, redundant, duplicated data or sub-optimal pre-training, can easily propagate to all the downstream tasks, introducing a single point of failure, which is undesirable. Therefore, specific data-cleaning pipelines and model checks should be developed to understand, measure, and mitigate those failures.

5.5.1 Limitations and challenges

Even if the above research questions were addressed, foundation models for particle physics might still face the following limitations and challenges:

- **Data format.** Particle collisions are stored for subsequent offline analysis in a format designed to preserve as much relevant information as possible. According to the specific data format, this information may include low-level and high-level physics objects

(e.g., charged particles, jets, high-level observables) to characterize an event. Therefore, events may be described by a large variety of information that can be generally useful to analyze different aspects of the underlying physics, despite being overwhelming for a machine or deep learning workflow that expects the data to be represented as tensors. This makes it difficult to utilize the available open data to people outside a physics collaboration, who have the knowledge and tools to exploit these data formats fully. As the usage of both ML and DL within the HEP community constantly increases, providing ML-friendly data formats along with their physics-ready ones could further incentivize and speed up the development and improvement of data-driven solutions. This is especially true for foundational models which are domain-generalists, requiring access to extensive knowledge to model an entire domain.

- **Data collection, labeling, and cleaning.** FMs are large neural networks requiring huge amounts of data to reach unprecedented levels of generalization. This fact exacerbates the challenges related to data collection or generation, labeling, and cleaning. In particular, the dataset(s) should be large and variegated, populated with high-quality non-redundant data, reflecting the main purpose of the foundation model. The experimental equipment yields the data to be collected, otherwise, the samples can be simulated through sophisticated software. Given enough computation, data samples of different physics processes can be endlessly generated with related class labels and annotations in the latter case. Moreover, even the detector response can be simulated improving the simulation fidelity. One issue with the simulations is that these are only approximations of reality, only modeling the known physics with sufficient precision. On the other hand, experimental data can be noisy and erroneous due to measurement effects and apparatus defects (e.g., dead sensors). In addition, these are often an unlabeled mixture of physics processes with unknown proportions that can only be roughly estimated. Hence, simulated and empirical data offer complementary advantages to each other. Nevertheless, in both cases, it should be established which part of the event is relevant to keep, and which filtering criteria should be applied to preserve enough representative and interesting samples.
- **Physics alignment.** The pre-training objective that FMs optimize is mostly designed to encourage the model to learn a general, rich, and reusable representation of the data. These learning objectives do not ensure that what is learned does not violate the underlying physics. In the previous pages, we have seen examples of early FMs designed to model jet physics, in which a jet is a sequence of particles. Physically, this sequence is unordered meaning that a permutation of the jet's constituents should not alter the FM's output. In this sense, these FMs are required to be permutation invariant to comply with the underlying jet physics. For example, this constraint can be ensured at the architecture level, utilizing a permutation invariant backbone like a Transformer [13] without applying the positional encoding or causal masking. Other constraints can be enforced through regularization or additional loss terms. In general, these physics properties may be difficult to specify. A “physics alignment”

post-training phase might be, thus, required to ensure the pre-trained FM is properly aligned with the desired notion of physics. In a similar spirit to RLHF [360], a *physics preference model* can be specified (e.g., through selected samples that satisfy the physics) or learned from data to tune the output of a pre-trained physics FM such that to agree with the preference model. This post-training phase would be useful for generative rather than predictive outputs, guiding the sample generation toward more physically plausible results.

- **Negative transfer.** A useful feature of FMs is that the pre-trained knowledge can be efficiently transferred to the target task within the same domain. Therefore, pre-training objectives should be general enough to prevent or minimize negative transfer [383] but also induce learning a suitably rich representation that can be easily transferred, yielding high-quality predictions. The main issue is that negative transfer can manifest only when fine-tuning for the target task, leaving little possibility of anticipating it during the pre-training phase.
- **Computing and deployment challenges.** By definition a FM is built from a high-capacity backbone. Training large models may require a distributed infrastructure with capable computing nodes. Even if an organization can access such infrastructure, as the model and data are scaled up to build more advanced FMs, the computing requirement to keep the training time reasonable can quickly become infeasible. In addition, as the number of nodes increases, hardware faults and communication costs can become a major training bottleneck. For the same reasons, these models since large are subject to high inference times. Hence, the role of model optimization and acceleration is equally important in this context. Despite the current techniques (e.g., pruning, quantization, distillation), it would be challenging to deploy fine-tuned FMs on latency-strict environments like the L1 trigger where these models could be valuable for online data analysis.
- **Model updates.** Once a FM is completely trained, its knowledge will not remain up-to-date forever. As new insights and data are gathered, the data cleaning pipeline improves, and so on, the foundation model should be periodically updated. However, periodic full re-training can be too expensive, as the time and computational resources may increase. Therefore, fine-tuning or adapting the FM to the new data or continually learning the backbone [389] may constitute an affordable alternative. However, performance drops (or, even worse, catastrophic forgetting) on the target downstream tasks should be critically avoided.

Although promising, FMs cannot be universally deployed due to computational and data requirements that may be limiting in some applicative scenarios: small, easier-to-optimize models may be preferable to be deployed on latency-strict hardware; then, in the absence of a large and variegated pre-training dataset, training from scratch task-specific, regularly sized neural networks is favorable. Therefore, FMs are a valuable additional tool when these limitations are absent.

Chapter 6

Conclusions

In this dissertation, we explored the major learning paradigms, i.e., supervised, unsupervised, and reinforcement learning, to directly or indirectly tackle the problem of searching for novel elementary particles. Moreover, a new paradigm arises when computation, model size, and training data are jointly and sufficiently scaled up: models develop widely applicable knowledge and competence for the training domain, resulting in a foundation model. In particular, the ideas, guidelines, and methods proposed in this thesis are meant to be generally applicable to various background and signal processes, eventually extending them to multiple physics objects, detectors, and even collision systems.

The focus of the second chapter of the thesis was model-dependent searches, in which a model of the signal is assumed. These kinds of searches are mainly carried out as supervised signal-background classification tasks. Parameter-conditional neural networks (PCNNs) can incorporate knowledge about the particle physics domain at various levels, improving the sensitivity to the signal and, thus, to the potential new physics compared to traditional neural networks and machine learning classifiers. Achieving better classification performance directly translates to achieving an improved estimation of the signal and background hypotheses when considering the data collected by the detector equipment, hence giving the physicists more opportunities to study the hypothetical signal better and, more importantly, to formulate more effective statistical tests which are employed to validate if the signal hypothesis exists at all. PCNNs were demonstrated on a tabular dataset with low-level and high-level features, considering the particle's mass as a physics parameter for the conditioning. The method illustrated in the thesis can be extended to other data modalities such as (raw) detector images, and potentially to sequential data, hence broadening the applicability of such classifiers. In addition, diverse and multiple physics parameters can be embedded in the model architecture and training to boost the model performance further.

Since signal dependence requires precise data simulation and estimation of the uncertainties related to the unknown physics model, the third chapter describes model-independent searches via the anomaly detection paradigm showing advantages complementary to the techniques discussed in the previous chapter. Model-independent searches aim to be more robust to uncertainties and to have broader applicability, potentially allowing them to tar-

get multiple signals given the same background processes. Therefore, signal-independence could unlock the full utilization of the collected data to train broad and general anomaly detection models sensitive to novel physics. Auto-encoders and VAEs are simple yet effective DL models with which it is possible to implement background-only anomaly detection, which leverages events that are considered to be the norm (i.e., not anomalous, belonging to the background) to learn one or more anomaly scores that are, then, used to score the novel (potentially abnormal, i.e., not due to detector or data taking issues) data samples. Typically, novel events whose score exceeds a pre-defined threshold are considered anomalous. In this way, it is possible to train anomaly taggers from the known physics, which are expected to generalize over multiple backgrounds, thus enabling sensitivity to various new physics signatures. In this regard, the major challenges are building comprehensive benchmark datasets spanning various data modalities, which should comprise multiple background and signal processes, and designing or learning better anomaly scores.

In the fourth chapter of this thesis, diverse flavors of modern reinforcement learning (RL) were introduced: starting from model-free RL to offline RL, each of which enables solving a specific class of decision-making problems. Given the current state of the environment, model-free RL directly predicts the optimal action to be executed, which is the one that maximizes the reward function. If we are given a dynamics model, then, the agent can “reason” in advance by planning the next best action without querying the actual environment, which is expected to be more expensive but at the same time operationally safer. Furthermore, there exist applications for which specifying a well-defined reward function is challenging, because either: 1) the objective function is not fully captured by mathematics; 2) there are multiple objectives and trade-offs involved; or even 3) the desired metric is a complicated function of the state. In such cases, inverse RL can infer a parametric form of the reward function from a dataset of trajectories. Then, given the learned reward model, a policy can be found by either model-free or model-based RL. Lastly, offline RL tries to leverage large datasets of experience to learn a suitable policy without interacting with the target environment. Ideally, offline RL is the most suited RL paradigm to solve real-world problems (e.g., accelerator control, which requires safe operation and data-efficient learning), although still not as effective as model-free RL in terms of performance.

The fifth chapter concludes by introducing the recent paradigm of foundation models (FMs), which are multi-purpose large-scale models. FMs are trained on a given domain, being able to improve a potentially large set of downstream tasks simultaneously. Another appealing advantage of FMs is the opportunity to share the backbone, i.e., where the domain knowledge resides, to solve a novel task. The above reasons make FMs particularly promising for HEP: OmniJet- α , MPM, and OmniLearn are early examples of FMs that all focus on jet physics. Furthermore, we can hypothesize the development of *instruction-following* HEP FMs, i.e., models whose output is directly influenced by a given user query or prompt. We can imagine asking a FM to generate jets according to a user specification, or to test the feasibility of a conjectured particle decay. Such a model would be an assistant rather than a physicist replacement, potentially helping to design and conduct physics analyses in less time while achieving an overall better result.

Appendices

Appendix A

Parameter-conditional Neural Networks

A.1 Code and data repositories

The original studies presented in this thesis can be reproduced by following the provided open-source code and data. In particular, the results showed in sections 2.4 and 2.5, about parameter-conditional neural networks, are both based on the following code and data:

- Code repository: <https://github.com/Luca96/affine-parametric-networks>;
- Data repository (HEPMASS-IMB): <https://zenodo.org/records/6453048>.

A.2 Discrete distribution for parameter assignment

Considering a discrete physics parameter like the signal mass hypotheses, $\mathcal{M} = \{m_1, \dots, m_M\}$ where $|\mathcal{M}| = M$ and each $m_i \in N^+$ is a positive integer (or a discrete value to be more general.) The assignment strategy regarded as “discrete distribution”, denoted as $m^{(i)} \sim \mathcal{M}$, is implemented in practice by a Categorical distribution over M categories with a uniform probability vector, $\mathbf{p}$. This distribution can be indicated as $\text{Cat}(k = M, \mathbf{p} = \mathbf{p}_M)$, where $\mathbf{p}_M = [1/M, \dots, 1/M]$ is a vector of size $k = M$. Therefore, the sampling process is

$$\begin{aligned} i &\sim \text{Cat}(M, \mathbf{p}_M) \\ m^{(i)} &= \mathcal{M}[i]. \end{aligned}$$

This process first involves sampling a discrete index, i , that is then used to look up the corresponding parameter value, i.e., m_i (equivalently denoted by vector indexing as $\mathcal{M}[i]$). The sampled value, $m^{(i)} = m_i$, is lastly assigned to a background sample.

Let’s consider $\mathcal{M} = \{m_1 = 500, m_2 = 750, m_3 = 1000\}$ as example parameter values. The above sampling process would sample an index (say 2), resulting in parameter $m_2 = 750$

GeV. Moreover, the m_1 , m_2 and m_3 values have the same occurrence probability since uniform, i.e., $p(m_1) = p(m_2) = p(m_3) = 1/3$.

In principle, we can design the probability vector, p , to make some parameter values more probable than others. This can be useful to encode our prior assumptions or domain knowledge about the physics parameter.

A.3 Parameterized architectures

All the parameterized models employed for comparison, interpolation, and evaluation are feed-forward neural networks in which conditioning occurs on a single, discrete parameter: the mass of the hypothetical particle, i.e., m_X . Moreover, unless stated otherwise the physics parameter is assigned to the background by following the discrete (Categorical) distribution in which a parameter value is sampled from the set $\mathcal{M}$.

Input(s)	Layer	Units	Input(s)	Layer	Units
x, m	concatenation	-	x, m	concatenation	-
h	dense	500	h	dense	300
h	dense	500	h	dropout	-
h	dense	500	h	dense	150
h	dense	500	h	dropout	-
h	dense	500	h	dense	100
h	dense	1	h	dropout	-
			h	dense	50
			h	dropout	-
			h	dense	1
Parameters	2M		Parameters	70k	

Table A.1: PNN architectures. On the left, the original Baldi’s architecture [53], and our reinterpretation on the right: x denote the input features, m, the mass feature or physics parameter, and h represents the output of the previous hidden layer. Lastly, **concatenation** depicts the concatenation-based conditioning operation in which the features are concatenated with the physics parameter.

As outlined in table A.1, we revised the original pNN architecture [53] to overcome its limitations. Since Baldi’s pNN architecture has a large parameter count, compared to the number of samples and complexity of the HEPMASS dataset, it suffers from overfitting which, in turn, also impacts its interpolation and extrapolation capabilities. Therefore, we found that reducing the parameter count along with **dropout** regularization yields better generalization performance, fast and stable convergence during training, and an overall lightweight model.

Input(s)	Layer	Units
x	dense	300
x, m	affine	300
h	dropout	-
h	dense	150
h, m	affine	150
h	dropout	-
h	dense	100
h, m	affine	100
h	dropout	-
h	dense	50
h, m	affine	50
h	dropout	-
h	dense	1
Parameters	72k	

Table A.2: The architecture of the affine pNN. The affine conditioning (**affine**) takes two tensors as input. Initially, the conditioning is performed at the input level, i.e., on x and m, then, the hidden representation, h, is conditioned.

A.4 Comparison with standard ML approaches

Machine learning methods such as (boosted) decision trees [390] had a long history in HEP before the recent advent of deep learning, which is allowing us to leverage increasingly large datasets of particle collisions. Unlike DL, ML methods tend to be more effective on small tabular datasets, having lower training and inference times and better interpretability. On the other hand, deep neural networks can automatically learn highly non-linear relations from high-dimensional inputs without recurring to feature engineering while learning incrementally thanks to gradient descent, enabling us to leverage huge datasets effectively although at a considerable computational cost. This section explains why neural networks, particularly parametrized ones, are the best models for attacking the signal-background classification problem.

A.4.1 Machine learning classifiers

To better illustrate and clarify the above motivations, popular machine learning methods such as logistic regression [391], Support Vector Machines (SVMs) [392], Random Forests (RFs) [393], and XGBoost (eXtreme Gradient Boosting) [394] are compared¹ to an affine parameterized neural networks on classification, interpolation, and extrapolation tasks targeting the HEPMASS dataset. A brief introduction of the compared ML models follows.

Logistic regression. It is a linear model for classification tasks, formulated as follows:

$$y(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + w_0), \quad (\text{A.4.1})$$

where $\sigma(\cdot)$ is the sigmoid or logistic function, and $\mathbf{w} = (\mathbf{w}, w_0)$ its learnable parameters (i.e., the weight vector and bias) that yield the linear, multi-dimensional decision surfaces [391]. Logistic regression models predict a probability, which can be used as a classifier by applying a threshold. Moreover, l_2 -regularization penalizing $\frac{1}{C}\|\mathbf{w}\|_2^2 = \frac{1}{C}\mathbf{w}^\top \mathbf{w}$ is applied during training, whose strength is inversely controlled by a C hyperparameter. Considering the above formulation, a logistic regression model can be seen as a simple one-layer neural network.

SVM. A support vector machine is a linear classifier that learns a high-dimensional hyperplane between two classes of the kind $\mathbf{x}^\top \beta + \beta_0$. If the two classes are linearly separable, the optimal separating hyperplane is the one with maximal distance, M , from the *support vectors*, i.e., the minimal set of data samples closer to it: that is $y_i(\mathbf{x}^\top \beta + \beta_0) > M$, holding for each dataset point where $y_i \in \{-1, 1\}$ is the class label assigned to the i -th sample. In the general case, the two classes are not linearly separable, therefore a tolerance parameter, ξ , is introduced allowing us to find the classification margin that minimizes the number of

¹For a fair comparison, all the ML methods were tuned by grid search on a validation set and the physics parameter (i.e., the mass feature) was provided as additional input. Moreover, since HEPMASS contains various millions of samples, a linear SVM approximation with lower computational complexity was used.

violations allowed on the support vectors. If the optimal separating hyperplane does not exist in the original feature space, a kernel, $K(\mathbf{x}, \mathbf{x}')$, can transform the feature space into another high-dimensional space, in which separating the two classes is easier.

Random Forest. A RF is an ensemble of decision trees (DTs) [395], each of which is learned on a randomly sampled subset of the training data. In this way, by learning each tree on a random set of samples and features, they are more likely to make independent mistakes, which, when aggregated through averaging, will result in a model with lower variance, generalizing better. This is possible because DTs are powerful enough to reach a low-bias regime but are unstable, suffering from high variance, causing them to generalize poorly. If the trees make uncorrelated errors, the average of those errors will approach zero as more trees are added to the ensemble, thus reducing variance. A DT is a non-parametric model that applies a hierarchical sequence of binary choices on a selection of variables from the training dataset. DTs can generate complex separation surfaces, which can be visualized and easily interpreted. Unfortunately, this feature is lost in RFs since these aggregate many DTs. Nevertheless, a RF can quantify the importance of each input feature, e.g., by counting how many times a variable appears in all trees.

XGBoost. It is an efficient and scalable implementation of the gradient boosting framework [396]. XGBoost trains an ensemble of K trees combined through summation:

$$\hat{y}^{(i)} = \sum_{k=1}^K f_k(\mathbf{x}^{(i)}), \quad (\text{A.4.2})$$

where $\hat{y}^{(i)}$ is the predicted class label for the i -th sample $\mathbf{x}^{(i)}$, and $f_k(\cdot)$ a decision tree. XGBoost minimizes a differentiable loss function while penalizing the model complexity to prevent overfitting. Gradient boosting sequentially adds predictors (DT classifiers in our case) to an ensemble. Each new predictor, say f_i , is trained on the residual error, r_{i-1} made by the previous one, f_{i-1} , i.e., $r_i = r_{i-1} - \hat{r}_{i-1}$, where $\hat{r}_{i-1} = f_{i-1}(\mathbf{x}^{(i)})$ and $r_1 = y^{(i)} - f_0(\mathbf{x}^{(i)})$. Compared to RFs, which aim to reduce variance, boosting trains correlated models such that the final *strong learner* achieves low bias. Given the powerful formulation of this model, increasing the number of estimators in the ensemble may lead to overfitting since the residual error is fitted each time. Therefore adequate regularization is necessary to guarantee sufficient generalization on novel data samples.

A.4.2 Comparison results

The evaluation results are shown in figures A.1 and A.2, where we can conclude that:

- Conditioning on the physics parameter can happen only via feature concatenation for ML models, because, unlike NNs, there is no direct control of the model architecture.
- For classification, XGBoost performs sensibly better than the other ML approaches but still underperforms the affine pNN by a large margin when comparing the AUROC.

Instead, all the evaluated models achieve competitive classification performance at the threshold that maximizes the significance ratio, σ_{ratio} , even if the pNN is still more consistent across the mass hypotheses.

- Regarding interpolation, linear models (i.e., the SVM and logistic regression) can retain the same classification performance despite being trained on fewer mass points. This behavior is expected because these models can linearly interpolate by construction. However, the pNN is better on average because it can learn a non-linear way to interpolate the intermediate values. Lastly, the XGBoost and RF generalize poorly on the unseen values, achieving the lowest performance.
- The affine pNN surpasses all the competing models by a neat margin during extrapolation, confirming that this class of models is capable of wide generalization while better relating the parameter along the input features.

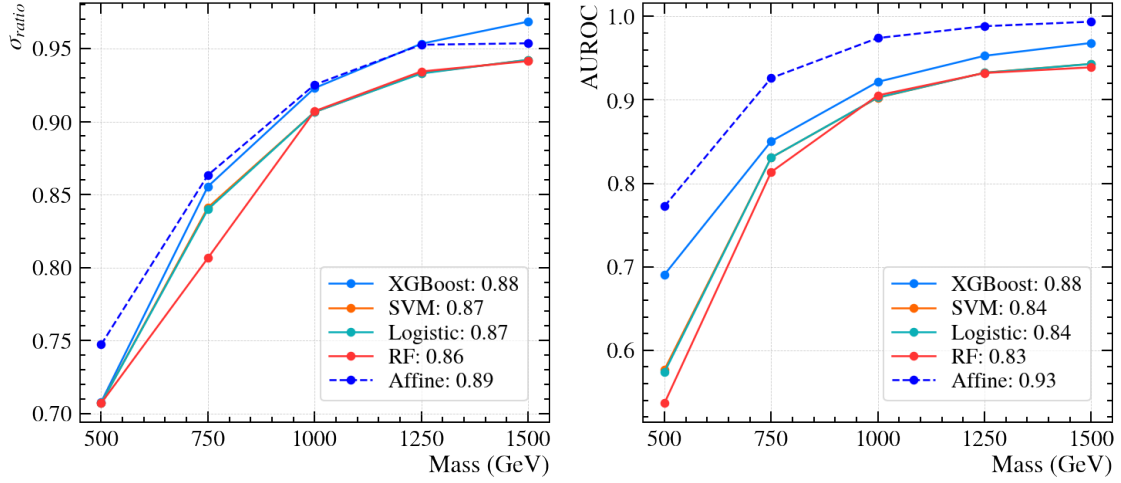


Figure A.1: Classification comparison on HEPMASS.

In conclusion, parameterized neural networks show a neat advantage over competing ML models at the evaluated tasks. This makes pNNs the optimal choice for the problem of signal-background classification with large and high-dimensional datasets: in principle, pNNs can go beyond tabular data, for example, learning from image datasets. Moreover, unlike other ML models, the XGBoost classifier needs different hyperparameters (as listed in the next section) to solve each task optimally.

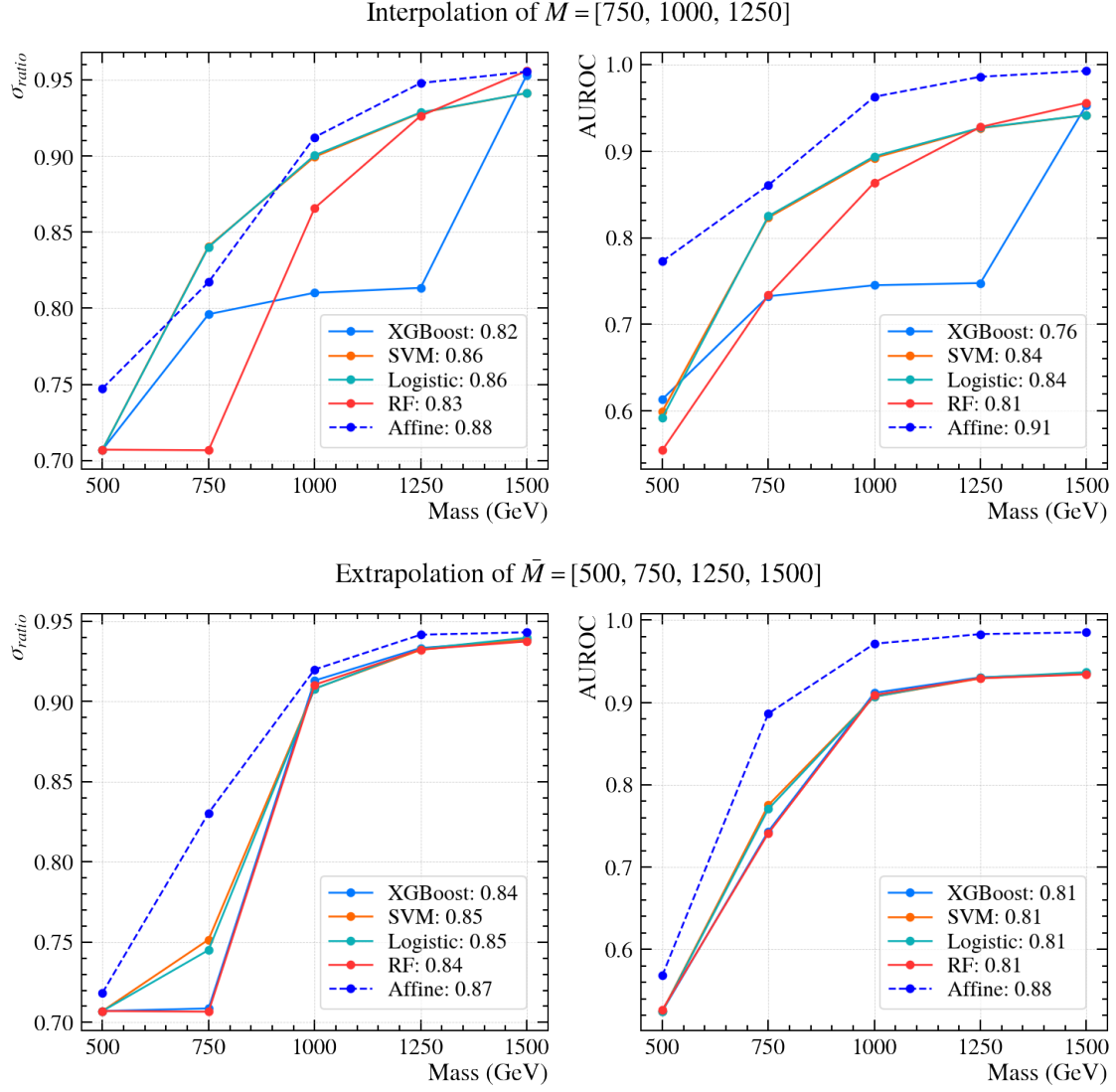


Figure A.2: Interpolation and extrapolation tasks on the HEPMASS dataset: $\bar{M}$ denotes the set of missing mass hypotheses whose corresponding signal samples were not used for training the respective models.

A.4.3 Hyperparameters

The best hyperparameters discovered by grid search are listed in this section (tables A.3, A.4, and A.5), for each of the above ML models; the random seed was fixed to 42 for all training runs. The other hyperparameters were left to default values as implemented by the `sklearn` [397] and `xgboost`² Python libraries.

C	Penalty	Loss
0.2	12	hinge

Table A.3: SVM hyperparameters. A linear kernel was used to lower the training time; the same hyperparameters were optimal for the three tasks.

Criterion	Max depth	Num. estimators
gini	6	200

Table A.4: RF hyperparameters; the same hyperparameters were optimal for the three tasks.

Task	Learning rate	Lambda	Max depth	Num. estimators
Classification	0.3	3	8	300
Interpolation	0.1	4	2	50
Extrapolation	0.1	3	6	25

Table A.5: XGBoost hyperparameters. The model minimizes a logistic loss; the log-loss was used for validation with ten early stopping rounds. Here we notice that the model should be complex to solve classification but way more regularized, with lower capacity, to avoid overfitting when interpolating and extrapolating.

²<https://xgboost.readthedocs.io/en/stable/index.html>.

Appendix B

Anomaly Detection for Model Independent Search

The work on conditional dual auto-encoders for anomaly detection, presented in section 3.5, is available at the code repository: <https://github.com/Luca96/dark-autoencoders>.

B.1 Data generation, simulation, and reconstruction

SM multi-jet QCD, as well as SVJ and SUEP events, are generated using `PYTHIA 8.244` [45], simulated and reconstructed with `DELPHES 3.5` [198] fast detector simulation using the CMS detector configuration, to obtain the tracker, ECAL, and HCAL images. Proton-proton collisions at the LHC Run-3 center-of-mass energy of 13.6 TeV are considered, with approximately 50 collisions per bunch crossing (pileup). A pileup reduction is applied to the reconstructed tracks: the E_T deposits in the inner tracker by the pileup tracks are removed. No pileup reduction occurs for calorimeters, following the HLT system's configuration during Run 3 [235].

The model parameters for the QCD events generation are identical as in [398]. For SUEP event generation, we used the `SUEP_Generator` plugin [399], and the following parameter settings have been used [223]:

- Mediator masses (m_H) = {125, 200, 300, 400, 700, 1000} GeV
- Dark meson mass (m_{ϕ_D}) = 2 GeV
- Branching ratio of $\phi_D \rightarrow 2Z_D = 100\%$
- Dark photon mass (m_{Z_D}) = 0.7 GeV
- Branching ratios of $Z_D \rightarrow \pi^+\pi^-, e^+e^-, \mu^+\mu^- = 70\%, 15\%, 15\%$
- Dark temperature (T_D) = 2 GeV.

The model parameters for the SVJ signal are identical to those in [231]. The mediator mass, $m_{Z'}$, values of 2.1 TeV, 3.1 TeV, and 4.1 TeV are considered.

A preselection criterion is applied to both background and signal events. The event preselection efficiency is found to be about 85% for QCD and approximately of 2.1%, 4.0%, 7.1%, 10.4%, 18.4%, and 22.7% for SUEP(125 GeV), SUEP(200 GeV), SUEP(300 GeV), SUEP(400 GeV), SUEP(700 GeV), and SUEP(1000 GeV), respectively. The preselection efficiencies for SVJ(2.1 TeV), SVJ(3.1 TeV), and SVJ(4.1 TeV) are about 9.3%, 9.8%, and 9.9%, respectively.

B.2 CoDAE, CoDVAE, and CAE architectures

In this section we describe the architectures of the three networks that build the CoDAE and CoDVAE models, namely the residual encoder (612k parameters), the auxiliary encoder (270k parameters), and the conditional decoder (763k parameters), in tables B.1, B.2 and B.3, respectively. Moreover, the CAE baseline is built by implementing the architectures depicted in table B.1 and B.3.

Input	Operator	Filters	Strides	Groups	# Conv
$72 \times 58 \times 1$	<code>conv_blk</code>	64	2	1	1
$36 \times 29 \times 64$	<code>residual</code>	64	1	4	2
$36 \times 29 \times 64$	<code>conv_blk</code>	128	2	4	1
$18 \times 15 \times 128$	<code>residual</code>	128	1	4	2
$18 \times 15 \times 128$	<code>conv_blk</code>	128	2	4	1
$9 \times 8 \times 128$	<code>residual</code>	128	1	4	2
$9 \times 8 \times 128$	<code>conv_blk</code>	256	2	4	1
$5 \times 4 \times 256$	<code>residual</code>	256	1	4	2
$5 \times 4 \times 64$	<code>conv1x1</code>	64	1	1	1

Table B.1: Architecture of the residual encoder, f_R . The operator `conv_blk` denotes a convolutional block made of a 2d convolution followed by instance normalization [203], with the Leaky-ReLU [204] activation ($\alpha = 0.2$) at the end. Instead, the `residual` operator comprises two `conv_blk` blocks, plus a skip connection. All the convolutions have a 3×3 kernel size. Down-sampling is performed by strided convolutions (with $s = 2$), instead of (max-)pooling. The last point-wise convolution [146] (`conv1x1`) linearly combines the feature maps to yield the $5 \times 4 \times 64$ latent space.

Input	Operator	Filters	Strides	Groups
$72 \times 58 \times 1$	conv_blk	64	1	1
$72 \times 58 \times 64$	max_pool	-	2	-
$36 \times 29 \times 64$	conv_blk	128	1	2
$36 \times 29 \times 128$	max_pool	-	2	-
$18 \times 15 \times 128$	conv_blk	128	1	2
$18 \times 15 \times 128$	max_pool	-	2	-
$9 \times 8 \times 128$	conv_blk	256	1	2
$9 \times 8 \times 256$	max_pool	-	2	-
$5 \times 4 \times 256$	flatten	-	-	-
5120	dense	2	-	-

Table B.2: Architecture of the auxiliary encoder, f_m . Overlapping Max-Pooling (**max_pool**) is employed to reduce the spatial dimensions by a factor of two, having a 3×3 kernel. When the features maps are small enough, a **flatten** operation reshapes them to a rank-1 tensor, and a final fully-connected (**dense**) layer yields the latent space of size two.

Input	Operator	Filters	Strides	Groups	# Conv
$5 \times 4 \times 64$	conditioning	-	-	-	-
$5 \times 4 \times 256$	up-conv_blk	256	1	4	1
$10 \times 8 \times 256$	residual	256	1	4	2
$10 \times 8 \times 256$	conditioning	-	-	-	-
$10 \times 8 \times 256$	up-conv_blk	128	1	4	1
$20 \times 16 \times 128$	residual	128	1	4	2
$20 \times 16 \times 128$	conditioning	-	-	-	-
$20 \times 16 \times 128$	up-conv_blk	128	1	4	1
$40 \times 32 \times 128$	residual	128	1	4	2
$40 \times 32 \times 128$	conditioning	-	-	-	-
$40 \times 32 \times 128$	up-conv_blk	64	1	4	1
$80 \times 64 \times 64$	residual	64	1	4	2
$80 \times 64 \times 64$	crop	-	-	-	-
$72 \times 58 \times 64$	conv1x1	1	1	1	1

Table B.3: Architecture of the conditional decoder D . Conditioning occurs at the input level as well as at multiple spatial resolutions. The **up-conv_blk** denotes a block that comprises a 2×2 bilinear up-sampling operation, followed by a grouped convolution and instance normalization, similarly to **conv_blk**. Since the up-sampled feature maps get larger than the input images, we employ a center crop operation (**crop**) to obtain matching sizes. The final **conv1x1** outputs $72 \times 58 \times 1$ images by a **sigmoid** activation.

B.3 AE architecture

Table B.4 describes the architecture of the AE model inspired¹ by [85] that we used to compare against the CoDAE models. The AE has 600k parameters, a latent space of size 32, and was trained to minimize the MSE loss on augmented mask images for 50 epochs by the AdamW optimizer.

Encoder	Layer	Filters	Decoder	Layer	Filters
$72 \times 58 \times 1$	conv2d	10	32	dense	100
$72 \times 58 \times 10$	avg_pool	-	100	dense	400
$36 \times 29 \times 10$	conv2d	5	400	reshape	-
$36 \times 29 \times 5$	conv2d	5	$20 \times 20 \times 1$	conv_t	5
$36 \times 29 \times 5$	avg_pool	-	$40 \times 40 \times 5$	conv2d	5
$18 \times 14 \times 5$	flatten	-	$40 \times 40 \times 5$	conv_t	5
1260	dense	400	$80 \times 80 \times 5$	conv2d	5
400	dense	100	$80 \times 80 \times 5$	crop	-
100	dense	32	$72 \times 58 \times 5$	conv2d	1

Table B.4: Encoder and decoder architectures of the AE model, inspired by figure 2 in [85]. In particular: `conv2d` denotes a convolutional layer with a 3×3 kernel, `avg_pool` a 2×2 average pooling operation, and `conv_t` a 3×3 transposed convolution [400]. After each convolutional, dense, and transposed convolutional layer the PReLU [22] activation, which learns per-dimension slope parameters, is applied.

¹To the best of my knowledge, the authors provide only a figure outlining their model architecture and not a comprehensive description. This figure was used as a reference to guess their model architecture.

Bibliography

- [1] I. J. Goodfellow, Y. Bengio, and A. C. Courville, *Deep Learning*. Adaptive computation and machine learning, MIT Press, 2016.
- [2] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” *Science*, vol. 349, no. 6245, pp. 255–260, 2015.
- [3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.
- [4] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, *et al.*, “On the opportunities and risks of foundation models,” *arXiv preprint arXiv:2108.07258*, 2021.
- [5] W. J. Scheirer, A. de Rezende Rocha, A. Sapkota, and T. E. Boult, “Toward open set recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 7, pp. 1757–1772, 2012.
- [6] H. Larochelle, D. Erhan, and Y. Bengio, “Zero-data learning of new tasks,” in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pp. 646–651, AAAI Press, 2008.
- [7] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, “Improving language understanding by generative pre-training,” *preprint*, 2018.
- [8] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, *et al.*, “Training language models to follow instructions with human feedback,” *Advances in neural information processing systems*, vol. 35, pp. 27730–27744, 2022.
- [9] H. Robbins and S. Monro, “A stochastic approximation method,” *The annals of mathematical statistics*, pp. 400–407, 1951.
- [10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [12] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [14] D. Bacciu, F. Errica, A. Micheli, and M. Podda, “A gentle introduction to deep

-
- learning for graphs,” *Neural Networks*, vol. 129, pp. 203–221, 2020.
- [15] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Židek, A. Potapenko, *et al.*, “Highly accurate protein structure prediction with alphafold,” *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.
- [16] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815–823, 2015.
- [17] R. Lam, A. Sanchez-Gonzalez, M. Willson, P. Wirsberger, M. Fortunato, F. Alet, S. Ravuri, T. Ewalds, Z. Eaton-Rosen, W. Hu, *et al.*, “Learning skillful medium-range global weather forecasting,” *Science*, vol. 382, no. 6677, pp. 1416–1421, 2023.
- [18] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, “Stochastic variational inference,” *Journal of Machine Learning Research*, 2013.
- [19] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, “Monte carlo gradient estimation in machine learning,” *Journal of Machine Learning Research*, vol. 21, no. 132, pp. 1–62, 2020.
- [20] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, “Deep learning for computer vision: A brief review,” *Computational intelligence and neuroscience*, vol. 2018, 2018.
- [21] D. W. Otter, J. R. Medina, and J. K. Kalita, “A survey of the usages of deep learning for natural language processing,” *IEEE transactions on neural networks and learning systems*, vol. 32, no. 2, pp. 604–624, 2020.
- [22] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *IEEE International Conference on Computer Vision, ICCV*, pp. 1026–1034, IEEE Computer Society, 2015.
- [23] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pp. 770–778, IEEE Computer Society, 2016.
- [24] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *Medical Image Computing and Computer-Assisted Intervention - MICCAI*, vol. 9351 of *Lecture Notes in Computer Science*, pp. 234–241, Springer, 2015.
- [25] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, “Language models are few-shot learners,” *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- [26] F. Shamshad, S. Khan, S. W. Zamir, M. H. Khan, M. Hayat, F. S. Khan, and H. Fu, “Transformers in medical imaging: A survey,” *Medical Image Analysis*, vol. 88, p. 102802, 2023.
- [27] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, “Learning by cheating,” in *Conference on Robot Learning*, pp. 66–75, PMLR, 2020.
- [28] H. Purwins, B. Li, T. Virtanen, J. Schlüter, S.-Y. Chang, and T. Sainath, “Deep learning for audio signal processing,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 2, pp. 206–219, 2019.
- [29] M. Andrychowicz, L. Espeholt, D. Li, S. Merchant, A. Meroze, F. Zyda, S. Agrawal, and N. Kalchbrenner, “Deep learning for day forecasts from sparse observations,” *arXiv preprint arXiv:2306.06079*, 2023.

-
- [30] N. Lee and J. Moon, “Offline reinforcement learning for automated stock trading,” *IEEE Access*, 2023.
 - [31] A. Khot, M. S. Neubauer, and A. Roy, “A detailed study of interpretability of deep neural network based top taggers,” *Machine Learning: Science and Technology*, vol. 4, no. 3, p. 035003, 2023.
 - [32] B. Lambert, F. Forbes, S. Doyle, H. Dehaene, and M. Dojat, “Trustworthy clinical ai solutions: a unified review of uncertainty quantification in deep learning models for medical image analysis,” *Artificial Intelligence in Medicine*, p. 102830, 2024.
 - [33] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, “Safe model-based reinforcement learning with stability guarantees,” *Advances in neural information processing systems*, vol. 30, 2017.
 - [34] A. Farahani, S. Voghoei, K. Rasheed, and H. R. Arabnia, “A brief review of domain adaptation,” *Advances in data science and information engineering*, pp. 877–894, 2021.
 - [35] D. Ha and J. Schmidhuber, “Recurrent world models facilitate policy evolution,” *Advances in neural information processing systems*, vol. 31, 2018.
 - [36] T. M. Moerland, J. Broekens, A. Plaat, C. M. Jonker, *et al.*, “Model-based reinforcement learning: A survey,” *Foundations and Trends® in Machine Learning*, vol. 16, no. 1, pp. 1–118, 2023.
 - [37] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into deep learning*. Cambridge University Press, 2023.
 - [38] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” in *International conference on machine learning*, pp. 1126–1135, PMLR, 2017.
 - [39] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in *3rd International Conference on Learning Representations, ICLR*, 2015.
 - [40] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlings-son, *et al.*, “Extracting training data from large language models,” in *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2633–2650, 2021.
 - [41] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, *et al.*, “Relational inductive biases, deep learning, and graph networks,” *arXiv preprint arXiv:1806.01261*, 2018.
 - [42] L. Evans and P. Bryant, “LHC Machine,” *JINST*, vol. 3, p. S08001, 2008.
 - [43] S. Chatrchyan *et al.*, “The CMS Experiment at the CERN LHC,” *JINST*, vol. 3, p. S08004, 2008.
 - [44] G. Cowan, K. Cranmer, E. Gross, and O. Vitells, “Asymptotic formulae for likelihood-based tests of new physics,” *The European Physical Journal C*, vol. 71, pp. 1–19, 2011.
 - [45] T. Sjöstrand, S. Ask, J. R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C. O. Rasmussen, and P. Z. Skands, “An introduction to PYTHIA 8.2,” *Comput. Phys. Commun.*, vol. 191, pp. 159–177, 2015.
 - [46] S. Agostinelli, J. Allison, K. a. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand, *et al.*, “Geant4—a simulation toolkit,” *Nuclear instruments and methods in physics research section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 506, no. 3, pp. 250–303, 2003.

-
- [47] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol, and K. Krüger, “Decoding photons: Physics in the latent space of a bib-ae generative network,” in *EPJ Web of Conferences*, vol. 251, p. 03003, EDP Sciences, 2021.
- [48] G. Aad, B. Abbott, D. C. Abbott, A. A. Abud, K. Abeling, D. K. Abhayasinghe, S. H. Abidi, A. Aboulhorma, H. Abramowicz, H. Abreu, *et al.*, “Atlfast3: the next generation of fast simulation in atlas,” *Computing and software for big science*, vol. 6, no. 1, p. 7, 2022.
- [49] C. Krause and D. Shih, “Caloflow: fast and accurate generation of calorimeter showers with normalizing flows,” *arXiv preprint arXiv:2106.05285*, 2021.
- [50] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol, W. Krcari, K. Krüger, and P. McKeown, “Caloclouds: Fast geometry-independent highly-granular calorimeter simulation,” *Journal of Instrumentation*, vol. 18, no. 11, p. P11025, 2023.
- [51] B. De Ville, “Decision trees,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 5, no. 6, pp. 448–455, 2013.
- [52] P. Baldi, P. Sadowski, and D. Whiteson, “Searching for Exotic Particles in High-Energy Physics with Deep Learning,” *Nature Commun.*, vol. 5, p. 4308, 2014.
- [53] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski, and D. Whiteson, “Parameterized Neural Networks for High-Energy Physics,” *The European Physical Journal C*, vol. 76, no. 5, pp. 1–7, 2016.
- [54] R. T. D’agnolo and A. Wulzer, “Learning new physics from a machine,” *Physical Review D*, vol. 99, no. 1, p. 015014, 2019.
- [55] R. T. d’Agnolo, G. Grosso, M. Pierini, A. Wulzer, and M. Zanetti, “Learning multivariate new physics,” *The European Physical Journal C*, vol. 81, no. 1, pp. 1–21, 2021.
- [56] A. K. Jain, J. Mao, and K. M. Mohiuddin, “Artificial neural networks: A tutorial,” *Computer*, vol. 29, no. 3, pp. 31–44, 1996.
- [57] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th International Conference on Machine Learning*, pp. 807–814, Omnipress, 2010.
- [58] L. Yang and A. Shami, “On hyperparameter optimization of machine learning algorithms: Theory and practice,” *Neurocomputing*, vol. 415, pp. 295–316, 2020.
- [59] L. N. Smith, “Cyclical learning rates for training neural networks,” in *IEEE winter conference on applications of computer vision (WACV)*, pp. 464–472, IEEE, 2017.
- [60] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in *7th International Conference on Learning Representations, ICLR*, OpenReview.net, 2019.
- [61] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.
- [62] R. Müller, S. Kornblith, and G. E. Hinton, “When does label smoothing help?,” *Advances in neural information processing systems*, vol. 32, 2019.
- [63] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, JMLR Workshop and Conference Proceedings, 2010.
- [64] L. Breiman, “Bagging predictors,” *Machine learning*, vol. 24, pp. 123–140, 1996.

- [65] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, 2015.
- [66] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [67] C. Shorten and T. M. Khoshgoftaar, “A survey on image data augmentation for deep learning,” *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.
- [68] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, “A simple framework for contrastive learning of visual representations,” in *International conference on machine learning*, pp. 1597–1607, PMLR, 2020.
- [69] T. DeVries, “Improved regularization of convolutional neural networks with cutout,” *arXiv preprint arXiv:1708.04552*, 2017.
- [70] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, “Cutmix: Regularization strategy to train strong classifiers with localizable features,” in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6023–6032, 2019.
- [71] Z. Zhang, T. He, H. Zhang, Z. Zhang, J. Xie, and M. Li, “Bag of freebies for training object detection neural networks,” *arXiv preprint arXiv:1902.04103*, 2019.
- [72] V. Dumoulin, E. Perez, N. Schucher, F. Strub, H. d. Vries, A. Courville, and Y. Bengio, “Feature-wise transformations,” *Distill*, 2018. <https://distill.pub/2018/feature-wise-transformations>.
- [73] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, “TensorFlow: a system for Large-Scale machine learning,” in *12th USENIX symposium on operating systems design and implementation (OSDI)*, pp. 265–283, 2016.
- [74] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” *Advances in neural information processing systems*, vol. 32, 2019.
- [75] L. Anzalone, T. Diotallevi, and D. Bonaccorsi, “Affine parametric neural networks for high-energy physics,” in *21st International Workshop on Advanced Computing and Analysis Techniques in Physics Research, ACAT*, 2022.
- [76] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, “Cross-stitch networks for multi-task learning,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3994–4003, 2016.
- [77] D. Abati, A. Porrello, S. Calderara, and R. Cucchiara, “Latent space autoregression for novelty detection,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 481–490, 2019.
- [78] L. Anzalone, T. Diotallevi, and D. Bonaccorsi, “Improving parametric neural networks for high-energy physics (and beyond),” *Machine Learning: Science and Technology*, vol. 3, p. 035017, oct 2022.
- [79] L. Anzalone, T. Diotallevi, and D. Bonaccorsi, “Hepmass-imb,” Apr. 2022.
- [80] M. Erdmann, E. Geiser, Y. Rath, and M. Rieger, “Lorentz boost networks: autonomous physics-inspired feature engineering,” *Journal of Instrumentation*, vol. 14, no. 06, p. P06006, 2019.
- [81] P. T. Komiske, E. M. Metodiev, and J. Thaler, “Energy flow networks: deep

-
- sets for particle jets,” *Journal of High Energy Physics*, vol. 2019, no. 1, pp. 1–46, 2019.
- [82] CMS, “Jets at CMS and the determination of their energy scale.” <https://cms.cern/news/jets-cms-and-determination-their-energy-scale>, 2012. [online; accessed 26-September-2024].
- [83] P. Baldi, P. Sadowski, and D. Whiteson, *Deep Learning from Four Vectors*, ch. Chapter 3, pp. 59–83. World Scientific Publishing, 2022.
- [84] M. Erdmann, B. Fischer, and D. Noll, “Reinforced sorting networks for particle physics analyses,” in *Journal of Physics: Conference Series*, vol. 1525, p. 012098, IOP Publishing, 2020.
- [85] T. Heimel, G. Kasieczka, T. Plehn, and J. M. Thompson, “QCD or What?,” *SciPost Phys.*, vol. 6, no. 3, p. 030, 2019.
- [86] B. M. Dillon, T. Plehn, C. Sauer, and P. Sorrenson, “Better Latent Spaces for Better Autoencoders,” *SciPost Phys.*, vol. 11, p. 061, 2021.
- [87] G. Kasieczka, R. Mastandrea, V. Mikuni, B. Nachman, M. Pettee, and D. Shih, “Anomaly detection under coordinate transformations,” *Physical Review D*, vol. 107, no. 1, p. 015009, 2023.
- [88] L. Anzalone, S. S. Chhibra, B. Maier, N. Chernyavskaya, and M. Pierini, “Triggering dark showers with conditional dual auto-encoders,” *Machine Learning: Science and Technology*, vol. 5, p. 035064, sep 2024.
- [89] B. M. Dillon, L. Favaro, F. Feiden, T. Modak, and T. Plehn, “Anomalies, representations, and self-supervision,” *arXiv preprint arXiv:2301.04660*, 2023.
- [90] S. Chen, A. Glioti, G. Panico, and A. Wulzer, “Parametrized classifiers for optimal eft sensitivity,” *Journal of High Energy Physics*, vol. 2021, no. 5, pp. 1–39, 2021.
- [91] R. T. d’Agnolo, G. Grosso, M. Pierini, A. Wulzer, and M. Zanetti, “Learning new physics from an imperfect machine,” *The European Physical Journal C*, vol. 82, no. 3, p. 275, 2022.
- [92] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,” *Journal of Computational physics*, vol. 378, pp. 686–707, 2019.
- [93] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, “Physics-informed machine learning,” *Nature Reviews Physics*, vol. 3, no. 6, pp. 422–440, 2021.
- [94] T. Beucler, M. Pritchard, S. Rasp, J. Ott, P. Baldi, and P. Gentine, “Enforcing analytic constraints in neural networks emulating physical systems,” *Physical Review Letters*, vol. 126, no. 9, p. 098302, 2021.
- [95] R. Caruana, “Multitask learning,” *Machine learning*, vol. 28, pp. 41–75, 1997.
- [96] M. Crawshaw, “Multi-task learning with deep neural networks: A survey,” *arXiv preprint arXiv:2009.09796*, 2020.
- [97] Y. Zhang and Q. Yang, “A survey on multi-task learning,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 12, pp. 5586–5609, 2021.
- [98] J. Schreiber and B. Sick, “Emerging relation network and task embedding for multi-task regression problems,” in *25th International Conference on Pattern Recognition (ICPR)*, pp. 2663–2670, IEEE, 2021.

-
- [99] D. Oeltz, J. Hamaekers, and K. F. Pilz, “Parameterized neural networks for finance,” *arXiv preprint arXiv:2304.08883*, 2023.
 - [100] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski, and D. Whiteson, “HEPMass Dataset.” <http://archive.ics.uci.edu/ml/datasets/HEPMass>, 2015.
 - [101] C. Adam-Bourdarios, G. Cowan, C. Germain, I. Guyon, B. Kégl, and D. Rousseau, “The Higgs Boson Machine Learning Challenge,” in *NIPS Workshop on High-energy Physics and Machine Learning*, pp. 19–55, PMLR, 2015.
 - [102] T. Fawcett, “An introduction to roc analysis,” *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.
 - [103] P. A. Flach, “Roc analysis,” in *Encyclopedia of machine learning and data mining*, pp. 1–8, Springer, 2016.
 - [104] J. Davis and M. Goadrich, “The relationship between precision-recall and roc curves,” in *Proceedings of the 23rd international conference on Machine learning*, pp. 233–240, 2006.
 - [105] A. Gulli and S. Pal, *Deep learning with Keras*. Packt Publishing Ltd, 2017.
 - [106] G. Van Rossum and F. L. Drake, “Python,” 2009.
 - [107] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR* (Y. Bengio and Y. LeCun, eds.), 2015.
 - [108] T. Diotallevi, *Application of Deep Learning techniques in the search for BSM Higgs bosons in the $\mu^+\mu^-$ final state in CMS*. PhD thesis, alma, Giugno 2022.
 - [109] CMS Collaboration, “Search for MSSM higgs bosons decaying to $\mu^+\mu^-$ in proton-proton collisions at $\sqrt{s} = 13$ tev,” *Physics Letters B*, vol. 798, p. 134992, 2019.
 - [110] E. Bols, J. Kieseler, M. Verzetti, M. Stoye, and A. Stakia, “Jet flavour classification using deepjet,” *Journal of Instrumentation*, vol. 15, no. 12, p. P12012, 2020.
 - [111] P. A. L. Bysted, J. Jensen, Z.-H. Tan, J. Østergaard, and L. Bramsløw, “A parameter-conditional neural network framework for modelling parameterized auditory models,” in *Proc. Euroregio/BNAM Joint Acoust. Conf*, pp. 307–316, 2022.
 - [112] V. Belis, P. Odagiu, and T. K. Aarrestad, “Machine learning for anomaly detection in particle physics,” *Reviews in Physics*, p. 100091, 2024.
 - [113] Z.-H. Zhou, “A brief introduction to weakly supervised learning,” *National science review*, vol. 5, no. 1, pp. 44–53, 2018.
 - [114] H. U. Dike, Y. Zhou, K. K. Deveerasetty, and Q. Wu, “Unsupervised learning based on artificial neural network: A review,” in *IEEE International Conference on Cyborg and Bionic Systems (CBS)*, pp. 322–327, IEEE, 2018.
 - [115] R. Chalapathy and S. Chawla, “Deep learning for anomaly detection: A survey,” *arXiv preprint arXiv:1901.03407*, 2019.
 - [116] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, “Deep learning for anomaly detection: A review,” *ACM computing surveys (CSUR)*, vol. 54, no. 2, pp. 1–38, 2021.
 - [117] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *3rd International Conference on Learning Representations, ICLR*, 2015.
 - [118] G. Karagiorgi, G. Kasieczka, S. Kravitz, B. Nachman, and D. Shih, “Machine learning in the search for new fundamental physics,” *Nature Reviews Physics*, vol. 4, no. 6, pp. 399–412, 2022.

-
- [119] D. Bank, N. Koenigstein, and R. Giryes, “Autoencoders,” *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pp. 353–374, 2023.
- [120] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.
- [121] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.
- [122] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, “Generalizing from a few examples: A survey on few-shot learning,” *ACM computing surveys (csur)*, vol. 53, no. 3, pp. 1–34, 2020.
- [123] F. Pourpanah, M. Abdar, Y. Luo, X. Zhou, R. Wang, C. P. Lim, X.-Z. Wang, and Q. J. Wu, “A review of generalized zero-shot learning methods,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 4, pp. 4051–4070, 2022.
- [124] Z. Chen, B. Liu, R. Brachman, P. Stone, and F. Rossi, *Lifelong Machine Learning*. Morgan & Claypool Publishers, 2nd ed., 2018.
- [125] D. Zimmerer, S. A. Kohl, J. Petersen, F. Isensee, and K. H. Maier-Hein, “Context-encoding variational autoencoder for unsupervised anomaly detection,” *arXiv preprint arXiv:1812.05941*, 2018.
- [126] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, “Unsupervised anomaly detection with generative adversarial networks to guide marker discovery,” in *International conference on information processing in medical imaging*, pp. 146–157, Springer, 2017.
- [127] V. Khachatryan *et al.*, “The CMS trigger system,” *JINST*, vol. 12, no. 01, p. P01020, 2017.
- [128] V. Blobel, “An Unfolding method for high-energy physics experiments,” in *Conference on Advanced Statistical Techniques in Particle Physics*, pp. 258–267, 8 2002.
- [129] C. Shimmmin, P. Sadowski, P. Baldi, E. Weik, D. Whiteson, E. Goul, and A. Søgaard, “Decorrelated jet substructure tagging using adversarial neural networks,” *Physical Review D*, vol. 96, no. 7, p. 074034, 2017.
- [130] G. Kasieczka and D. Shih, “Robust jet classifiers through distance correlation,” *Physical review letters*, vol. 125, no. 12, p. 122001, 2020.
- [131] T. Finke, M. Krämer, A. Morandini, A. Mück, and I. Oleksiyuk, “Autoencoders for unsupervised anomaly detection in high energy physics,” *JHEP*, vol. 06, p. 161, 2021.
- [132] K. Fraser, S. Homiller, R. K. Mishra, B. Ostdiek, and M. D. Schwartz, “Challenges for unsupervised anomaly detection in particle physics,” *Journal of High Energy Physics*, vol. 2022, no. 3, pp. 1–31, 2022.
- [133] D. Hendrycks, M. Mazeika, and T. G. Dietterich, “Deep anomaly detection with outlier exposure,” in *7th International Conference on Learning Representations, ICLR, OpenReview.net*, 2019.
- [134] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.
- [135] S. Ren, K. He, R. B. Girshick, and J. Sun, “Faster R-CNN: towards real-time object detection with region proposal networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017.

-
- [136] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.
 - [137] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.
 - [138] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, “Mask R-CNN,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 386–397, 2020.
 - [139] C. Godard, O. Mac Aodha, and G. J. Brostow, “Unsupervised monocular depth estimation with left-right consistency,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 270–279, 2017.
 - [140] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun, “End-to-end interpretable neural motion planner,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8660–8669, 2019.
 - [141] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, “Photo-realistic single image super-resolution using a generative adversarial network,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4681–4690, 2017.
 - [142] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.
 - [143] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” in *The 9th ISCA Speech Synthesis Workshop*, p. 125, ISCA, 2016.
 - [144] J. Ho, A. Jain, and P. Abbeel, “Denoising diffusion probabilistic models,” *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.
 - [145] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.
 - [146] M. Lin, Q. Chen, and S. Yan, “Network in network,” in *2nd International Conference on Learning Representations, ICLR*, 2014.
 - [147] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.
 - [148] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, “Visualizing the loss landscape of neural nets,” *Advances in neural information processing systems*, vol. 31, 2018.
 - [149] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, “Bag of tricks for image classification with convolutional neural networks,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 558–567, 2019.
 - [150] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, 2017.

-
- [151] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.
 - [152] X. Zhang, X. Zhou, M. Lin, and J. Sun, “Shufflenet: An extremely efficient convolutional neural network for mobile devices,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856, 2018.
 - [153] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *science*, vol. 313, no. 5786, pp. 504–507, 2006.
 - [154] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, 2008.
 - [155] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, “Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion,” *Journal of machine learning research*, vol. 11, no. 12, 2010.
 - [156] A. Ng *et al.*, “Sparse autoencoder,” *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19, 2011.
 - [157] S. Rifai, G. Mesnil, P. Vincent, X. Muller, Y. Bengio, Y. Dauphin, and X. Glorot, “Higher order contractive auto-encoder,” in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD*, pp. 645–660, Springer, 2011.
 - [158] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, “Stacked convolutional auto-encoders for hierarchical feature extraction,” in *21st International Conference on Artificial Neural Networks*, pp. 52–59, Springer, 2011.
 - [159] I. Manakov, M. Rohm, and V. Tresp, “Walking the tightrope: An investigation of the convolutional autoencoder bottleneck,” *arXiv preprint arXiv:1911.07460*, 2019.
 - [160] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, “Adversarial autoencoders,” *arXiv preprint arXiv:1511.05644*, 2015.
 - [161] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” in *2nd International Conference on Learning Representations, ICLR*, 2014.
 - [162] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.
 - [163] E. Jang, “A beginner’s guide to variational methods: mean-field approximation,” *blog.evjang.com*, 2016.
 - [164] L. Weng, “From autoencoder to beta-vae,” *lilianweng.github.io*, 2018.
 - [165] T. M. Cover, *Elements of information theory*. John Wiley & Sons, 1999.
 - [166] I. Higgins, L. Matthey, A. Pal, C. P. Burgess, X. Glorot, M. M. Botvinick, S. Mohamed, and A. Lerchner, “beta-vae: Learning basic visual concepts with a constrained variational framework,” in *5th International Conference on Learning Representations, ICLR*, OpenReview.net, 2017.
 - [167] A. Razavi, A. van den Oord, B. Poole, and O. Vinyals, “Preventing posterior collapse with delta-vaes,” in *7th International Conference on Learning Representations, ICLR*, OpenReview.net, 2019.
 - [168] A. Asperti and M. Trentin, “Balancing reconstruction error and kullback-leibler divergence in variational autoencoders,” *IEEE Access*, vol. 8, pp. 199440–199448, 2020.

-
- [169] L. Ruff, J. R. Kauffmann, R. A. Vandermeulen, G. Montavon, W. Samek, M. Kloft, T. G. Dietterich, and K.-R. Müller, “A unifying review of deep and shallow anomaly detection,” *Proceedings of the IEEE*, vol. 109, no. 5, pp. 756–795, 2021.
 - [170] C. Aytekin, X. Ni, F. Cricri, and E. Aksu, “Clustering and unsupervised anomaly detection with l_2 -normalized deep auto-encoder representations,” in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6, IEEE, 2018.
 - [171] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.
 - [172] T. DeVries and G. W. Taylor, “Learning confidence for out-of-distribution detection in neural networks,” *arXiv preprint arXiv:1802.04865*, 2018.
 - [173] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” in *4th International Conference on Learning Representations, ICLR*, 2016.
 - [174] T. Schlegl, P. Seeböck, S. M. Waldstein, G. Langs, and U. Schmidt-Erfurth, “fanogan: Fast unsupervised anomaly detection with generative adversarial networks,” *Medical image analysis*, vol. 54, pp. 30–44, 2019.
 - [175] G. Kwon, M. Prabhushankar, D. Temel, and G. AlRegib, “Backpropagated gradient representations for anomaly detection,” in *16th European Conference on Computer Vision, ECCV*, pp. 206–226, Springer, 2020.
 - [176] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, “Deep one-class classification,” in *International conference on machine learning*, pp. 4393–4402, PMLR, 2018.
 - [177] A. Harilal, K. Park, M. Andrews, and M. Paulini, “Autoencoder-based online data quality monitoring for the cms electromagnetic calorimeter,” in *21st International Workshop on Advanced Computing and Analysis Techniques in Physics Research, ACAT*, 2022.
 - [178] B. M. Dillon, L. Favaro, T. Plehn, P. Sorenson, and M. Krämer, “A normalized autoencoder for lhc triggers,” *SciPost Physics Core*, vol. 6, no. 4, p. 074, 2023.
 - [179] B. Bortolato, A. Smolkovič, B. M. Dillon, and J. F. Kamenik, “Bump hunting in latent space,” *Physical Review D*, vol. 105, no. 11, p. 115009, 2022.
 - [180] T. Cheng, J.-F. Arguin, J. Leissner-Martin, J. Pilette, and T. Golling, “Variational autoencoders for anomalous jet tagging,” *Physical Review D*, vol. 107, no. 1, p. 016002, 2023.
 - [181] B. Nachman and D. Shih, “Anomaly detection with density estimation,” *Physical Review D*, vol. 101, no. 7, p. 075042, 2020.
 - [182] A. Hallin, J. Isaacson, G. Kasieczka, C. Krause, B. Nachman, T. Quadfasel, M. Schlaffer, D. Shih, and M. Sommerhalder, “Classifying anomalies through outer density estimation,” *Physical Review D*, vol. 106, no. 5, p. 055006, 2022.
 - [183] E. M. Metodiev, B. Nachman, and J. Thaler, “Classification without labels: Learning from mixed samples in high energy physics,” *Journal of High Energy Physics*, vol. 2017, no. 10, pp. 1–18, 2017.
 - [184] G. Papamakarios, T. Pavlakou, and I. Murray, “Masked autoregressive flow for density estimation,” *Advances in neural information processing systems*, vol. 30, 2017.
 - [185] S. Yoon, Y. Noh, and F. C. Park, “Autoencoding under normalization constraints,” in *Proceedings of the 38th International Conference on Machine Learning, ICML*,

-
- vol. 139 of *Proceedings of Machine Learning Research*, pp. 12087–12097, PMLR, 2021.
- [186] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, “Normalizing flows for probabilistic modeling and inference,” *Journal of Machine Learning Research*, vol. 22, no. 57, pp. 1–64, 2021.
- [187] W. Joo, W. Lee, S. Park, and I. Moon, “Dirichlet variational autoencoder,” *Pattern Recognit.*, vol. 107, p. 107514, 2020.
- [188] N. Dilokthanakul, P. A. M. Mediano, M. Garnelo, M. C. H. Lee, H. Salimbeni, K. Arulkumaran, and M. Shanahan, “Deep unsupervised clustering with gaussian mixture variational autoencoders,” *CoRR*, vol. abs/1611.02648, 2016.
- [189] B. Bortolato, A. Smolkovič, B. M. Dillon, and J. F. Kamenik, “Bump hunting in latent space,” *Phys. Rev. D*, vol. 105, no. 11, p. 115009, 2022.
- [190] B. M. Dillon, R. Mastandrea, and B. Nachman, “Self-supervised anomaly detection for new physics,” *Phys. Rev. D*, vol. 106, no. 5, p. 056005, 2022.
- [191] B. M. Dillon, G. Kasieczka, H. Olschlager, T. Plehn, P. Sorrenson, and L. Vogel, “Symmetries, safety, and self-supervision,” *SciPost Phys.*, vol. 12, no. 6, p. 188, 2022.
- [192] E. Govorkova *et al.*, “Autoencoders on field-programmable gate arrays for real-time, unsupervised new physics detection at 40 MHz at the Large Hadron Collider,” *Nature Mach. Intell.*, vol. 4, pp. 154–161, 2022.
- [193] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” *Advances in neural information processing systems*, vol. 28, 2015.
- [194] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding,” in *4th International Conference on Learning Representations, ICLR*, 2016.
- [195] FastML Team, “fastmachinelearning/hls4ml.” <https://github.com/fastmachinelearning/hls4ml>, 2023.
- [196] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran, *et al.*, “Fast inference of deep neural networks in fpgas for particle physics,” *Journal of instrumentation*, vol. 13, no. 07, p. P07027, 2018.
- [197] E. Dupont, “Learning disentangled joint continuous and discrete representations,” *Advances in neural information processing systems*, vol. 31, 2018.
- [198] J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens, and M. Selvaggi, “DELPHES 3, A modular framework for fast simulation of a generic collider experiment,” *JHEP*, vol. 02, p. 057, 2014.
- [199] M. Cacciari, G. P. Salam, and G. Soyez, “The anti-kt jet clustering algorithm,” *Journal of High Energy Physics*, vol. 2008, no. 04, p. 063, 2008.
- [200] C. J. Maddison, A. Mnih, and Y. W. Teh, “The concrete distribution: A continuous relaxation of discrete random variables,” in *5th International Conference on Learning Representations, ICLR*, OpenReview.net, 2017.
- [201] E. Jang, S. Gu, and B. Poole, “Categorical reparameterization with gumbel-softmax,” in *5th International Conference on Learning Representations, ICLR*, OpenReview.net, 2017.
- [202] L. Valente, L. Anzalone, M. Lorusso, and D. Bonacorsi, “Joint variational auto-

- encoder for anomaly detection in high energy physics,” in *International Symposium on Grids and Clouds (ISGC)*, vol. 19, 2023.
- [203] D. Ulyanov, A. Vedaldi, and V. Lempit-sky, “Instance normalization: The missing ingredient for fast stylization,” *arXiv preprint arXiv:1607.08022*, 2016.
- [204] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, “Rectifier nonlinearities improve neural network acoustic models,” in *Proc. icml*, vol. 30, p. 3, Atlanta, GA, 2013.
- [205] A. Odena, V. Dumoulin, and C. Olah, “Deconvolution and checkerboard artifacts,” *Distill*, 2016.
- [206] N. Watters, L. Matthey, C. P. Burgess, and A. Lerchner, “Spatial broadcast decoder: A simple architecture for learning disentangled representations in vaes,” *arXiv preprint arXiv:1901.07017*, 2019.
- [207] L. A. Gatys, A. S. Ecker, and M. Bethge, “Image style transfer using convolutional neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2414–2423, 2016.
- [208] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*, pp. 448–456, pmlr, 2015.
- [209] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, “Dying relu and initialization: Theory and numerical examples,” *arXiv preprint arXiv:1903.06733*, 2019.
- [210] W. Jiang, L. Yang, E. H.-M. Sha, Q. Zhuge, S. Gu, S. Dasgupta, Y. Shi, and J. Hu, “Hardware/software co-exploration of neural architectures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4805–4815, 2020.
- [211] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, “Model compression and hardware acceleration for neural networks: A comprehensive survey,” *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.
- [212] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.
- [213] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.
- [214] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” in *5th International Conference on Learning Representations, ICLR, OpenReview.net*, 2017.
- [215] H. Benmeziane, K. E. Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, “A comprehensive survey on hardware-aware neural architecture search,” *arXiv preprint arXiv:2101.09336*, 2021.
- [216] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for machine learning*. Cambridge University Press, 2020.
- [217] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, “A survey of model compression and acceleration for deep neural networks,” *arXiv preprint arXiv:1710.09282*, 2017.
- [218] S. Hauck and A. DeHon, *Reconfigurable computing: the theory and practice of FPGA-based computation*. Elsevier, 2010.
- [219] T. Aarrestad, V. Loncar, N. Ghielmetti, M. Pierini, S. Summers, J. Ngadiuba, C. Petersson, H. Linander, Y. Iiyama, G. Di Guglielmo, *et al.*, “Fast convolutional neural networks on fpgas with hls4ml,”

-
- Machine Learning: Science and Technology*, vol. 2, no. 4, p. 045015, 2021.
- [220] “AMD Alveo U250 datasheet.” <https://www.avnet.com/opasdata/d120001/medias/docus/190/XX-A-U200-P64G-PQ-G-Datasheet.pdf>. Accessed on May 15, 2024.
- [221] “AMD Alveo U250 website.” <https://docs.amd.com/r/en-US/ds962-u200-u250/Alveo-Product-Details>. Accessed on May 15, 2024.
- [222] M. J. Strassler and K. M. Zurek, “Echoes of a hidden valley at hadron colliders,” *Phys. Lett. B*, vol. 651, pp. 374–379, 2007.
- [223] S. Knapen, S. Pagan Griso, M. Papucci, and D. J. Robinson, “Triggering Soft Bombs at the LHC,” *JHEP*, vol. 08, p. 076, 2017.
- [224] T. Cohen, M. Lisanti, and H. K. Lou, “Semivisible Jets: Dark Matter Undercover at the LHC,” *Phys. Rev. Lett.*, vol. 115, no. 17, p. 171804, 2015.
- [225] T. Cohen, M. Lisanti, H. K. Lou, and S. Mishra-Sharma, “LHC Searches for Dark Sector Showers,” *JHEP*, vol. 11, p. 196, 2017.
- [226] D. Kar and S. Sinha, “Exploring jet substructure in semi-visible jets,” *SciPost Phys.*, vol. 10, no. 4, p. 084, 2021.
- [227] T. R. A. Denk, “Quirky composite dark matter,” bachelor thesis, Tech. U., Munich, 2011.
- [228] S. Born, R. Karur, S. Knapen, and J. Shelton, “Scouting for dark showers at cms and lhcb,” *Physical Review D*, vol. 108, no. 3, p. 035034, 2023.
- [229] T. A. Collaboration, G. Aad, *et al.*, “The atlas experiment at the cern large hadron collider,” *Journal of Instrumentation*, vol. 3, p. S08003, aug 2008.
- [230] J. Barron, D. Curtin, G. Kasieczka, T. Plehn, and A. Spourdalakis, “Unsupervised hadronic suep at the lhc,” *Journal of High Energy Physics*, vol. 2021, p. 129, Dec 2021.
- [231] F. Canelli, A. de Cosa, L. Le Pottier, J. Niedziela, K. Pedro, and M. Pierini, “Autoencoders for semivisible jet detection,” *Journal of High Energy Physics*, vol. 2022, p. 74, Feb 2022.
- [232] A. M. Sirunyan *et al.*, “Particle-flow reconstruction and global event description with the cms detector,” *Journal of Instrumentation*, vol. 12, p. P10003, Oct 2017.
- [233] CMS, “Tracking.” <https://cms.cern/detector/identifying-tracks>. Accessed on May 17, 2024.
- [234] CMS, “Energy of Hadrons (HCAL).” <https://cms.cern/detector/measuring-energy/energy-hadrons-hcal>. Accessed on May 17, 2024.
- [235] V. Khachatryan *et al.*, “The CMS trigger system,” *JINST*, vol. 12, no. 01, p. P01020, 2017.
- [236] C. Shorten and T. M. Khoshgoftaar, “A survey on image data augmentation for deep learning,” *J. Big Data*, vol. 6, p. 60, 2019.
- [237] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” *CoRR*, vol. abs/1411.1784, 2014.
- [238] N. Watters, L. Matthey, C. P. Burgess, and A. Lerchner, “Spatial broadcast decoder: A simple architecture for learning disentangled representations in vaes,” *CoRR*, vol. abs/1901.07017, 2019.
- [239] S. Havrylov and I. Titov, “Emergence of language with multi-agent games: Learning to communicate with sequences of symbols,” in *Advances in Neural Information Processing Systems*, pp. 2149–2159, 2017.

- [240] S. Yan, J. S. Smith, W. Lu, and B. Zhang, “Hierarchical multi-scale attention networks for action recognition,” *Signal Process. Image Commun.*, vol. 61, pp. 73–84, 2018.
- [241] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, 2004.
- [242] R. Deng, C. Shen, S. Liu, H. Wang, and X. Liu, “Learning to predict crisp boundaries,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 562–578, 2018.
- [243] A. Hassani, S. Walton, N. Shah, A. Abuduweili, J. Li, and H. Shi, “Escaping the big data paradigm with compact transformers,” *CoRR*, vol. abs/2104.05704, 2021.
- [244] L. J. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *CoRR*, vol. abs/1607.06450, 2016.
- [245] D. Hendrycks and K. Gimpel, “Bridging nonlinearities and stochastic regularizers with gaussian error linear units,” *CoRR*, vol. abs/1606.08415, 2016.
- [246] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, “Deep networks with stochastic depth,” in *Computer Vision - ECCV* (B. Leibe, J. Matas, N. Sebe, and M. Welling, eds.), vol. 9908 of *Lecture Notes in Computer Science*, pp. 646–661, Springer, 2016.
- [247] E. Del Barrio, J. A. Cuesta-Albertos, and C. Matrán, “An optimal transportation approach for assessing almost stochastic order,” in *The Mathematics of the Uncertain*, pp. 33–44, Springer, 2018.
- [248] R. Dror, S. Shlomov, and R. Reichart, “Deep dominance - how to properly compare deep neural models,” in *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL*, pp. 2773–2785, Association for Computational Linguistics, 2019.
- [249] D. Ulmer, C. Hardmeier, and J. Frellsen, “deep-significance-easy and meaningful statistical significance testing in the age of neural networks,” *arXiv preprint arXiv:2204.06815*, 2022.
- [250] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep reinforcement learning: A brief survey,” *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.
- [251] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [252] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [253] J. Degraeve, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas, *et al.*, “Magnetic control of tokamak plasmas through deep reinforcement learning,” *Nature*, vol. 602, no. 7897, pp. 414–419, 2022.
- [254] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, *et al.*, “Solving rubik’s cube with a robot hand,” *arXiv preprint arXiv:1910.07113*, 2019.
- [255] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, “Learning to drive in a day,” in *International conference on robotics and automation (ICRA)*, pp. 8248–8254, IEEE, 2019.

-
- [256] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, “Grandmaster level in starcraft ii using multi-agent reinforcement learning,” *nature*, vol. 575, no. 7782, pp. 350–354, 2019.
- [257] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, “Reinforcement learning for combinatorial optimization: A survey,” *Computers & Operations Research*, vol. 134, p. 105400, 2021.
- [258] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatin, A. Novikov, F. J. R Ruiz, J. Schrittwieser, G. Swirszcz, *et al.*, “Discovering faster matrix multiplication algorithms with reinforcement learning,” *Nature*, vol. 610, no. 7930, pp. 47–53, 2022.
- [259] Y. Li, “Deep reinforcement learning: An overview,” *arXiv preprint arXiv:1701.07274*, 2017.
- [260] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Pearson, 2016.
- [261] M. L. Puterman, “Markov decision processes,” *Handbooks in operations research and management science*, vol. 2, pp. 331–434, 1990.
- [262] M. L. Littman, “A tutorial on partially observable markov decision processes,” *Journal of Mathematical Psychology*, vol. 53, no. 3, pp. 119–125, 2009.
- [263] P.-W. Chou, D. Maturana, and S. Scherer, “Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution,” in *International conference on machine learning*, pp. 834–843, PMLR, 2017.
- [264] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.
- [265] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, pp. 279–292, 1992.
- [266] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [267] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The arcade learning environment: An evaluation platform for general agents,” *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.
- [268] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, “Continuous deep q-learning with model-based acceleration,” in *International conference on machine learning*, pp. 2829–2838, PMLR, 2016.
- [269] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.
- [270] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, “Dueling network architectures for deep reinforcement learning,” in *International conference on machine learning*, pp. 1995–2003, PMLR, 2016.
- [271] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” in *4th International Conference on Learning Representations, ICLR*, 2016.
- [272] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, “Hindsight experience replay,” *Advances in neural information processing systems*, vol. 30, 2017.

- [273] M. Plappert, R. Houthoofd, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," in *6th International Conference on Learning Representations, ICLR*, OpenReview.net, 2018.
- [274] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *International conference on machine learning*, pp. 449–458, PMLR, 2017.
- [275] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, pp. 229–256, 1992.
- [276] C. Nota and P. S. Thomas, "Is the policy gradient a gradient?," in *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, pp. 939–947, International Foundation for Autonomous Agents and Multiagent Systems, 2020.
- [277] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, PMLR, 2016.
- [278] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *4th International Conference on Learning Representations, ICLR*, 2016.
- [279] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.
- [280] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
- [281] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR*, 2016.
- [282] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM Sigart Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.
- [283] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," *Advances in neural information processing systems*, vol. 32, 2019.
- [284] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML)*, pp. 465–472, 2011.
- [285] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*, pp. 282–293, Springer, 2006.
- [286] D. Hafner, T. P. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," in *8th International Conference on Learning Representations, ICLR*, OpenReview.net, 2020.
- [287] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, *et al.*, "Deepmind control suite," *arXiv preprint arXiv:1801.00690*, 2018.
- [288] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.
- [289] J. Geiping, Q. Garrido, P. Fernandez, A. Bar, H. Pirsiavash, Y. LeCun, and M. Goldblum, "A cookbook of

-
- self-supervised learning,” *arXiv preprint arXiv:2304.12210*, 2023.
- [290] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, “Mastering the game of go without human knowledge,” *nature*, vol. 550, no. 7676, pp. 354–359, 2017.
- [291] S. Levine and V. Koltun, “Guided policy search,” in *International conference on machine learning*, pp. 1–9, PMLR, 2013.
- [292] T. Anthony, R. Nishihara, P. Moritz, T. Salimans, and J. Schulman, “Policy gradient search: Online planning and expert iteration without search trees,” *arXiv preprint arXiv:1904.03646*, 2019.
- [293] Y. Tassa, T. Erez, and E. Todorov, “Synthesis and stabilization of complex behaviors through online trajectory optimization,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4906–4913, IEEE, 2012.
- [294] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “Carla: An open urban driving simulator,” in *Conference on robot learning*, pp. 1–16, PMLR, 2017.
- [295] W. Zhao, J. P. Queralta, and T. Westerlund, “Sim-to-real transfer in deep reinforcement learning for robotics: a survey,” in *IEEE symposium series on computational intelligence (SSCI)*, pp. 737–744, IEEE, 2020.
- [296] S. Schaal, “Is imitation learning the route to humanoid robots?,” *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.
- [297] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, “Imitation learning: A survey of learning methods,” *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–35, 2017.
- [298] A. Y. Ng and S. Russell, “Algorithms for inverse reinforcement learning,” in *Proceedings of the Seventeenth International Conference on Machine Learning ICML*, pp. 663–670, Morgan Kaufmann, 2000.
- [299] S. Levine, A. Kumar, G. Tucker, and J. Fu, “Offline reinforcement learning: Tutorial, review, and perspectives on open problems,” *arXiv preprint arXiv:2005.01643*, 2020.
- [300] M. Bain and C. Sammut, “A framework for behavioural cloning,” in *Machine Intelligence 15*, pp. 103–129, 1995.
- [301] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, “End to end learning for self-driving cars,” *arXiv preprint arXiv:1604.07316*, 2016.
- [302] S. Ross, G. Gordon, and D. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635, JMLR Workshop and Conference Proceedings, 2011.
- [303] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, B. B. Gupta, X. Chen, and X. Wang, “A survey of deep active learning,” *ACM computing surveys (CSUR)*, vol. 54, no. 9, pp. 1–40, 2021.
- [304] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, “Maximum entropy inverse reinforcement learning,” in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pp. 1433–1438, AAAI Press, 2008.
- [305] J. Ho and S. Ermon, “Generative adversarial imitation learning,” in *Advances in Neural Information Processing Systems*, pp. 4565–4573, 2016.
- [306] F. Pardo, A. Tavakoli, V. Levдик, and P. Kormushev, “Time limits in reinforcement learning,” in *International Confer-*

- ence on Machine Learning, pp. 4045–4054, PMLR, 2018.
- [307] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, “Implementation matters in deep rl: A case study on ppo and trpo,” in *International conference on learning representations*, 2019.
- [308] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, *et al.*, “What matters in on-policy reinforcement learning? a large-scale empirical study,” *arXiv preprint arXiv:2006.05990*, 2020.
- [309] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, “Deep reinforcement learning that matters,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.
- [310] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare, “Deep reinforcement learning at the edge of the statistical precipice,” *Advances in neural information processing systems*, vol. 34, pp. 29304–29320, 2021.
- [311] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester, “Challenges of real-world reinforcement learning: definitions, benchmarks and analysis,” *Machine Learning*, vol. 110, no. 9, pp. 2419–2468, 2021.
- [312] E. Altman, *Constrained Markov decision processes*. Routledge, 2021.
- [313] L. Anzalone, P. Barra, S. Barra, A. Castiglione, and M. Nappi, “An end-to-end curriculum learning approach for autonomous driving scenarios,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 10, pp. 19817–19826, 2022.
- [314] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, “Implicit quantile networks for distributional reinforcement learning,” in *International conference on machine learning*, pp. 1096–1105, PMLR, 2018.
- [315] Y. Coppens, K. Efthymiadis, T. Lenaerts, A. Nowé, T. Miller, R. Weber, and D. Magazzeni, “Distilling deep reinforcement learning policies in soft decision trees,” in *Proceedings of the IJCAI workshop on explainable artificial intelligence*, pp. 1–6, 2019.
- [316] N. Frosst and G. E. Hinton, “Distilling a neural network into a soft decision tree,” in *Proceedings of the First International Workshop on Comprehensibility and Explanation in AI and ML co-located with 16th International Conference of the Italian Association for Artificial Intelligence (AI*IA)*, vol. 2071 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2017.
- [317] L. Anzalone, D. Bonacorsi, *et al.*, “Reinforce-lib: A reinforcement learning library for scientific research,” *POS PROCEEDINGS OF SCIENCE*, vol. 415, pp. 1–17, 2022.
- [318] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.
- [319] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, *et al.*, “Gymnasium: A standard interface for reinforcement learning environments,” *arXiv preprint arXiv:2407.17032*, 2024.
- [320] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, “Stable-baselines3: Reliable reinforcement learning implementations,” *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

-
- [321] A. Kuhnle, M. Schaarschmidt, and K. Fricke, “Tensorforce: a tensorflow library for applied reinforcement learning.” Web page, 2017.
- [322] A. Bou, M. Bettini, S. Dittert, V. Kumar, S. Sodhani, X. Yang, G. De Fabritiis, and V. Moens, “Torchrl: A data-driven decision-making library for pytorch,” *arXiv preprint arXiv:2306.00577*, 2023.
- [323] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. AraÅsjo, “Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms,” *Journal of Machine Learning Research*, vol. 23, no. 274, pp. 1–18, 2022.
- [324] L. Dong, G. Gao, X. Zhang, L. Chen, and Y. Wen, “Baconian: A unified open-source framework for model-based reinforcement learning,” *arXiv preprint arXiv:1904.10762*, 2019.
- [325] J. McLeod, H. Stojic, V. Adam, D. Kim, J. Grau-Moya, P. Vrancx, and F. Leibfried, “Bellman: A toolbox for model-based reinforcement learning in tensorflow,” *arXiv preprint arXiv:2103.14407*, 2021.
- [326] T. Seno and M. Imai, “d3rlpy: An offline deep reinforcement learning library,” *Journal of Machine Learning Research*, vol. 23, no. 315, pp. 1–20, 2022.
- [327] N. Lucchesi, A. Carta, V. Lomonaco, and D. Bacciu, “Avalanche rl: A continual reinforcement learning library,” in *International Conference on Image Analysis and Processing*, pp. 524–535, Springer, 2022.
- [328] S. Hu, Y. Zhong, M. Gao, W. Wang, H. Dong, X. Liang, Z. Li, X. Chang, and Y. Yang, “Marllib: A scalable and efficient multi-agent reinforcement learning library,” *Journal of Machine Learning Research*, vol. 24, no. 315, pp. 1–23, 2023.
- [329] Y. Song, A. Wojcicki, T. Lukasiewicz, J. Wang, A. Aryan, Z. Xu, M. Xu, Z. Ding, and L. Wu, “Arena: A general evaluation platform and building toolkit for multi-agent intelligence,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, pp. 7253–7260, 2020.
- [330] J. Kaiser, O. Stein, and A. Eichler, “Learning-based optimisation of particle accelerators under partial observability without real-world training,” in *International Conference on Machine Learning*, pp. 10575–10585, PMLR, 2022.
- [331] V. Kain, S. Hirlander, B. Goddard, F. M. Velotti, G. Z. Della Porta, N. Bruchon, and G. Valentino, “Sample-efficient reinforcement learning for cern accelerator control,” *Physical Review Accelerators and Beams*, vol. 23, no. 12, p. 124801, 2020.
- [332] J. St. John, C. Herwig, D. Kafkes, J. Mitrevski, W. A. Pellico, G. N. Perdue, A. Quintero-Parra, B. A. Schupbach, K. Seiya, N. Tran, *et al.*, “Real-time artificial intelligence for accelerator control: A study at the fermilab booster,” *Physical Review Accelerators and Beams*, vol. 24, no. 10, p. 104601, 2021.
- [333] R. C. Panda, *Introduction to PID controllers: theory, tuning and application to frontier areas*. BoD–Books on Demand, 2012.
- [334] S. Hirlander and N. Bruchon, “Model-free and bayesian ensembling model-based deep reinforcement learning for particle accelerator control demonstrated on the fermi fel,” *arXiv preprint arXiv:2012.09737*, 2020.
- [335] G. Grosso, *Searching for Unexpected New Physics at the LHC with Machine Learning*. PhD thesis, Padua U., 2023.
- [336] O. J. Parra, J. G. Pardiñas, L. D. P. Pérez, M. Janisch, S. Klaver, T. Lehericy,

- and N. Serra, “Human-in-the-loop reinforcement learning for data quality monitoring in particle physics experiments,” *arXiv preprint arXiv:2405.15508*, 2024.
- [337] C. Bower, *Adversarial thresholding semi-bandits*. PhD thesis, University of Derby, UK, 2020.
- [338] S. Acharya *et al.*, “The ALICE Transition Radiation Detector: construction, operation, and performance,” *Nucl. Instrum. Meth. A*, vol. 881, pp. 88–127, 2018.
- [339] T. Lattimore and C. Szepesvári, *Bandit algorithms*. Cambridge University Press, 2020.
- [340] S. Carrazza and F. A. Dreyer, “Jet grooming through reinforcement learning,” *Physical Review D*, vol. 100, no. 1, p. 014014, 2019.
- [341] J. Brehmer, S. Macaluso, D. Pappadopulo, and K. Cranmer, “Hierarchical clustering in particle physics through reinforcement learning,” *34th Conference on Neural Information Processing Systems, 2011.08191*, 2020.
- [342] L. H. Vage, *New directions for particle tracking at the High-Luminosity LHC*. PhD thesis, Princeton University (US), 2024.
- [343] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Journal of Basic Engineering*, vol. 82, pp. 35–45, 03 1960.
- [344] J. Duarte and J.-R. Vlimant, “Graph neural networks for particle tracking and reconstruction,” in *Artificial intelligence for high energy physics*, pp. 387–436, World Scientific, 2022.
- [345] T. Kortus, R. Keidel, N. R. Gauger, B. pCT Collaboration, *et al.*, “Towards neural charged particle tracking in digital tracking calorimeters with reinforcement learning,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.
- [346] O. Vinyals, M. Fortunato, and N. Jaitly, “Pointer networks,” *Advances in neural information processing systems*, vol. 28, 2015.
- [347] T. Harvey and A. Lukas, “Quark mass models and reinforcement learning,” *Journal of High Energy Physics*, vol. 2021, no. 8, pp. 1–21, 2021.
- [348] S. Nishimura, C. Miyao, and H. Otsuka, “Exploring the flavor structure of quarks and leptons with reinforcement learning,” *Journal of High Energy Physics*, vol. 2023, no. 12, pp. 1–40, 2023.
- [349] S. R. Qasim, P. Owen, and N. Serra, “Physics instrument design with reinforcement learning,” *arXiv preprint arXiv:2412.10237*, 2024.
- [350] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, “A survey on deep transfer learning,” in *27th International Conference on Artificial Neural Networks, ICANN*, pp. 270–279, Springer, 2018.
- [351] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, “A comprehensive survey on transfer learning,” *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.
- [352] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang, “Self-supervised learning: Generative or contrastive,” *IEEE transactions on knowledge and data engineering*, vol. 35, no. 1, pp. 857–876, 2021.
- [353] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: pre-training of deep bidirectional transformers for language understanding,” in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*, pp. 4171–4186, Association for Computational Linguistics, 2019.

-
- [354] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, “Gpt-4 technical report,” *arXiv preprint arXiv:2303.08774*, 2023.
- [355] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, *et al.*, “The llama 3 herd of models,” *arXiv preprint arXiv:2407.21783*, 2024.
- [356] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, “Language models are unsupervised multitask learners,” *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.
- [357] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, “Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing,” *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.
- [358] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.
- [359] OpenAI, “ChatGPT.” <https://openai.com/index/chatgpt/>, 2022. [online; accessed 26-September-2024].
- [360] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, “Deep reinforcement learning from human preferences,” *Advances in neural information processing systems*, vol. 30, 2017.
- [361] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, *et al.*, “Chain-of-thought prompting elicits reasoning in large language models,” *Advances in neural information processing systems*, vol. 35, pp. 24824–24837, 2022.
- [362] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “Lora: Low-rank adaptation of large language models,” in *The Tenth International Conference on Learning Representations, ICLR*, OpenReview.net, 2022.
- [363] J. Steinhardt, P. W. W. Koh, and P. S. Liang, “Certified defenses for data poisoning attacks,” *Advances in neural information processing systems*, vol. 30, 2017.
- [364] H. Inan, K. Upasani, J. Chi, R. Rungta, K. Iyer, Y. Mao, M. Tontchev, Q. Hu, B. Fuller, D. Testuggine, *et al.*, “Llama guard: Llm-based input-output safeguard for human-ai conversations,” *arXiv preprint arXiv:2312.06674*, 2023.
- [365] M. Zhang, O. Press, W. Merrill, A. Liu, and N. A. Smith, “How language model hallucinations can snowball,” in *Forty-first International Conference on Machine Learning, ICML*, OpenReview.net, 2024.
- [366] V. Romei and G. Di Pellegrino, “Lecture 9: Neural mechanisms of selective visual attention,” in *Cognition and Neuroscience*, Università di Bologna, 2020.
- [367] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” in *3rd International Conference on Learning Representations, ICLR*, 2015.
- [368] T. Luong, H. Pham, and C. D. Manning, “Effective approaches to attention-based neural machine translation,” in *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP*, pp. 1412–1421, The Association for Computational Linguistics, 2015.
- [369] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, “Show, attend and tell: Neural image caption generation with visual attention,” in *International conference on machine learning*, pp. 2048–2057, PMLR, 2015.

-
- [370] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, “Transformers in vision: A survey,” *ACM computing surveys (CSUR)*, vol. 54, no. 10s, pp. 1–41, 2022.
 - [371] J. Birk, A. Hallin, and G. Kasieczka, “Omnijet- α : the first cross-task foundation model for particle physics,” *Machine Learning: Science and Technology*, vol. 5, p. 035031, aug 2024.
 - [372] L. Heinrich, T. Golling, M. Kagan, S. Klein, M. Leigh, M. Osadchy, and J. A. Raine, “Masked particle modeling on sets: Towards self-supervised high energy physics foundation models,” *Machine Learning: Science and Technology*, 2024.
 - [373] R. E. Turner, “An introduction to transformers,” *arXiv preprint arXiv:2304.10557*, 2023.
 - [374] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *arXiv preprint arXiv:1607.06450*, 2016.
 - [375] A. Graves, “Generating sequences with recurrent neural networks,” *arXiv preprint arXiv:1308.0850*, 2013.
 - [376] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer, “Generating wikipedia by summarizing long sequences,” in *6th International Conference on Learning Representations, ICLR, OpenReview.net*, 2018.
 - [377] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, “Scaling laws for neural language models,” *arXiv preprint arXiv:2001.08361*, 2020.
 - [378] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, “Direct preference optimization: Your language model is secretly a reward model,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.
 - [379] R. A. Bradley and M. E. Terry, “Rank analysis of incomplete block designs: I. the method of paired comparisons,” *Biometrika*, vol. 39, no. 3/4, pp. 324–345, 1952.
 - [380] V. Mikuni and B. Nachman, “Omnilearn: A method to simultaneously facilitate all jet physics tasks,” *arXiv preprint arXiv:2404.16091*, 2024.
 - [381] A. Van Den Oord, O. Vinyals, *et al.*, “Neural discrete representation learning,” *Advances in neural information processing systems*, vol. 30, 2017.
 - [382] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, “Deep unsupervised learning using nonequilibrium thermodynamics,” in *International conference on machine learning*, pp. 2256–2265, PMLR, 2015.
 - [383] Z. Wang, Z. Dai, B. Póczos, and J. Carbonell, “Characterizing and avoiding negative transfer,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11293–11302, 2019.
 - [384] H. Qu, C. Li, and S. Qian, “Particle transformer for jet tagging,” in *International Conference on Machine Learning*, pp. 18281–18292, PMLR, 2022.
 - [385] A. Andreassen, P. T. Komiske, E. M. Metodiev, B. Nachman, and J. Thaler, “Omnifold: a method to simultaneously unfold all observables,” *Physical review letters*, vol. 124, no. 18, p. 182001, 2020.
 - [386] O. Amram, L. Anzalone, J. Birk, D. A. Faroughy, A. Hallin, G. Kasieczka, M. Krämer, I. Pang, H. Reyes-Gonzalez, and D. Shih, “Aspen open jets: Unlocking lhc data for foundation models in particle physics,” *arXiv preprint arXiv:2412.10504*, 2024.

-
- [387] C. Collaboration, “JetHT primary dataset in miniaod format from RunG of 2016 (/JetHT/Run2016G-UL2016_MiniAODv2-v2/MINIAOD),” 2024.
- [388] C. Collaboration, “JetHT primary dataset in miniaod format from RunH of 2016 (/JetHT/Run2016H-UL2016_MiniAODv2-v2/MINIAOD),” 2024.
- [389] A. Razdaibiedina, Y. Mao, R. Hou, M. Khabsa, M. Lewis, and A. Almahairi, “Progressive prompts: Continual learning for language models,” in *The Eleventh International Conference on Learning Representations, ICLR*, OpenReview.net, 2023.
- [390] S. Chatrchyan *et al.*, “Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC,” *Phys. Lett. B*, vol. 716, pp. 30–61, 2012.
- [391] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, vol. 4. Springer, 2006.
- [392] C. Cortes and V. Vapnik, “Support-vector networks,” *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.
- [393] L. Breiman, “Random forests,” *Machine learning*, vol. 45, pp. 5–32, 2001.
- [394] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 785–794, 2016.
- [395] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth, 1984.
- [396] J. H. Friedman, “Greedy function approximation: a gradient boosting machine,” *Annals of statistics*, pp. 1189–1232, 2001.
- [397] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, “Scikit-learn: Machine learning in python,” *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.
- [398] A. Tumasyan *et al.*, “Search for resonant production of strongly coupled dark matter in proton-proton collisions at 13 TeV,” *JHEP*, vol. 06, p. 156, 2022.
- [399] S. Knapen, J. Shelton, and D. Xu, “Perturbative benchmark models for a dark shower search program,” *Phys. Rev. D*, vol. 103, no. 11, p. 115013, 2021.
- [400] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” *CoRR*, vol. abs/1603.07285, 2016.