Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA

Meccanica e Scienze Avanzate dell'Ingegneria

Prog.3 Meccanica Applicata

Ciclo XXIV

Settore Concorsuale di afferenza: 09/A2

Settore Scientifico disciplinare: ING-IND/13

INVERSE STATIC ANALYSIS OF MASSIVE PARALLEL ARRAYS OF THREE-STATE ACTUATORS VIA ARTIFICIAL INTELLIGENCE

Presentata da:

Ing. Felix Pasila

Coordinatore Dottorato:	**Relatore:**
Prof. Vincenzo Parenti Castelli	**Prof. Vincenzo Parenti Castelli**
	Dr. Rocco Vertechy
	Dr. Giovanni Berselli

Esame finale anno 2013

INVERSE STATIC ANALYSIS OF MASSIVE PARALLEL ARRAYS OF THREE-STATE ACTUATORS VIA ARTIFICIAL INTELLIGENCE

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy

University of Bologna

Presented by

Felix Pasila

Advisors:

Prof. Vincenzo Parenti Castelli

Dr. Rocco Vertechy

Dr. Giovanni Berselli

2013

Abstract

Massive parallel robots (MPRs) driven by discrete actuators are force regulated robots that undergo continuous motions despite being commanded through a finite number of states only. Designing a real-time control of such systems requires fast and efficient methods for solving their inverse static analysis (ISA), which is a challenging problem and the subject of this thesis. In particular, five Artificial intelligence methods are proposed to investigate the on-line computation and the generalization error of ISA problem of a class of MPRs featuring three-state force actuators and one degree of revolute motion.

Keywords: Massive parallel robots, inverse static analysis, artificial intelligences, real-time control

Acknowledgement

I would like to sincerely thank my advisers, Professor Parenti-Castelli, Dr. Vertechy and Dr. Berselli, for their immense help, encouragement and support through my research work.

I would also like to thank Dr. Carricato, Dr. Conconi, Dr. Carminelli, Dr. Qiaoling Meng, Dr. Benedetta Baldisserri, Dr. M. Mozaffari and all members of GRAB, for their constant feedbacks on this research.

Lastly, I would like to thank my wife, Hestiasari Rante and our sons, Amadeo Constantine Pasila and Egidio Geovanney Pasila, for the inspiration and supports they provided during my study. Without their companies and encouragement, I will not be able to go this far. I am also very thankful to my mother and parents-in-law, brothers and sisters, for their endless prayers.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Rationale of the thesis

This thesis addresses the efficient way to design the real-time control procedure of massive discrete-state manipulators (DSM). The DSM is a very special kind of mechanisms whose actuators can only be made switching among a finite number of states. The manipulators are introduced by Pieper (1968) and Roth (1973), in an effort to conceive sensor-less robots as well as to reduce the complexity of computer interfacing and control procedure. Currently DSM can be classified into two different groups depending on whether their actuators act as discrete displacement generators or discrete force generators. Examples of DSM of the first type are the binary snake-like robots (SLRs), proposed by Chirikjian et al (1994a, 1994b, 1995, 1996a, 1996b, 1997, 2001, 2004) and Dubowsky et al (2001, 2002), which are kinematically constrained mechanisms employing a large number of bi-stable actuators whose configuration either fully contracted (inactive state) or fully extended (active state) without consideration of the arbitrary external forces acting on them. Examples of DSM of the second type are the binary Massively Parallel Robots (MPRs) [Waldron et.al., 2001a, 2001b; Mukherjee, 2002], which are dynamically constrained robots employing a large number of on-off actuators that employ either a constant force (active state) or no force (inactive state) irrespective of their arbitrary kinematically unconstrained configuration.

Major advantages of SLRs and MPRs over conventional manipulators are the possibilities of:

1. Increasing system robustness against external disturbances, information and power signal noise, actuator and electronics aging, as well as actuator failure;
2. Employing simpler and cheaper actuators, sensors and electronics.

To achieve high position/force capabilities (both in terms of variation range and accuracy), the architecture of SLRs/MPRs practically requires a large number of actuators (usually 4-10 times larger than the number of degrees of freedom desired for the robot) that can be arranged in a hybrid series-parallel configuration (prevalently in-series for SLRs [Chirikjian et al., 1994a, 1994b, 1995, 1996a, 1996b, 1997, 2001; Dubowsky et al., 2001, 2002] whereas in-parallel for MPRs [Waldron et al., 2001a, 2001b; Mukherjee, 2002]. Owing to the large number and the discrete nature of the actuator variables (positions for SLRs and forces for MPRs), the inverse kinematic analysis (IKA) of SLRs and the inverse static analysis (ISA) of MPRs are usually very difficult problems whose solution practically requires quite complicated procedures. In the past, significant research efforts have been devoted to address these inverse problems, in particular by resorting to: exhaustive brute-force search approaches [Waldron et al., 2001a, 2001b; Mukherjee, 2002]; methods of classical differential geometry and variation of calculus [Chirikjian, 1995, 1997]; combinatorial heuristics algorithms [Lees 1996; Dubowsky et al., 2001, 2002]; genetic algorithms [Dubowsky et al., 2001, 2002]; probability theory [Chirikjian et al., 1996a, 2001]; high-gain Hopfield networks and Boltzmann machines [Waldron et al., 2001a, 2001b]. Even though most of the proposed solution schemes are formally very elegant and quite effective in reducing problem complexity from exponential time to polynomial time, the resulting algorithms still involve too many calculations for real-time manipulator control.

This thesis deals with the ISA problem of planar MPRs, that is to find the states of the actuator variables for a given external (force or moment) acting on the MPRs output link.

1.2 Hypothesis: ISA solution using Artificial Intelligence

In this context, we investigate the potentialities of using artificial intelligence (AI) methods for the real-time solution of the ISA of planar ternary MPRs that feature one revolute degree of freedom actuated by a number of in-parallel-placed three-state force generators. In particular, the thesis considers five different MPRs mechanisms that are actuated by *m* three state force generators (with the three states being -1, 0 and 1 and with *m* = 2, 4, 6, 8 and 10, hereafter referred to as *m*-ternary MPRs).

The proposed AI methods are based on the Neuro-Fuzzy (NF) and Neural Network (NN). The first method is a hybrid intelligent system which combines the human-like reasoning style of fuzzy systems with the learning ability of neural networks. The main advantages of a neuro-fuzzy system are: it interprets IF-THEN rules from input-output relations and focuses on accuracy of the output network and an efficient time consumption for on-line computation [Jang 1993; Palit 2001, 2002b, 2005]. The second method is instead inspired by the biological nervous system. This is because the NN consists of highly interconnected networks with a large number of processing elements (called artificial neurons), which resemble the human brain system. The advantages of using NN are: it is an efficient pattern recognition system and acts like parallel distributed processing (parallel computing) which accelerate the computational process [Elman 1990; Juang 2002; Palit 2002a, 2005; Toha 2008].

In general, the practical uses of AI methods have been recognized mainly because of such distinguished features:

- Pattern recognition capability: enables to capture patterns or essential relationship among the data (especially when the relationship is not known or very difficult to describe mathematically, and/or when the observation data is corrupted with noise).
- Universal approximation capability (universal function): enables modeling of highly nonlinear functions with good accuracy.
- Adaptive learning capability: enables to learn from examples using data-driven approach by updating the related parameters.

The first two features above are used here in the ISA solution for constructing the relation between the vector of actuator activation states $\boldsymbol{u} = [u_1,\dots, u_m]$, which is a m-ternary number, and the input couple of position and moment $\boldsymbol{X} =[\alpha, M]$, which are continuum real numbers. The last feature is used instead for tuning the parameters that are related to the NF and NN architectures.

1.3 Contributions: Six ISA solutions

In this thesis, we proposed six ISA solutions which are: 1. Look-Up Table (LUT); 2. Neuro-Fuzzy type Takagi-Sugeno (NFTS); 3. NFTS with Look-Up Table (NFLUT); 4. Neural Network type Multilayer Perceptron (MLP); 5. Recurrent Neural Network type Elman (ERNN); and 6. High-Gain Hopfield Network (HN). One of the proposed solutions should overcome the problem of ISA of the related MPRs mechanisms (2, 4, 6, 8 and 10- three-state force actuator), that is to find the states of the actuator in fast and efficient way (concerning the real-time computing and the generalization error terms). More detail of all methods including their training algorithms and the real-time computing performances will be compared in Chapters 3 and 4 respectively.

In brief, the main contributions of proposing AI methods for the ISA solution of *m*-ternary MPRs are:

- They can approximate the output *m*-ternary with good accuracy.
- They compute the output *m*-ternary in real-time.
- They are adaptive systems and have capability to update their parameters via the learning algorithm.

The proposed *m*-ternary mechanism as well as the six ISA solutions will be explained in Chapters 2 and 3 respectively.

1.4 The limitation of the proposed thesis

It is worth to mention that the results of the ISA solutions based on Artificial Intelligence (AI) which are shown in this thesis are not the global optima. This means that there could be other AI architectures that give better results in the terms of generalization error. This limitation is however tolerable because we require to derive efficient ISA solutions that have real-time computation response (i.e. less than 5ms) and also acceptable full generalization error (i.e. less than 10%). Suggestion to avoid the limitation in this thesis can be done by using multi-objective optimization strategies, instead of Random Hill Climbing (HC). The HC is a local search algorithm for finding the best parameters in the learning procedure (see Appendix A for detailed procedure).

1.5 Thesis outline

- **Chapter 1** introduces the motivation, the problem of ISA of ternary MPRs and the six ISA solutions as the main contribution of this thesis, such as: one brute-force algorithm and five AI methods.

- **Chapter 2** introduces five ternary MPRs mechanisms which respectively employ 2, 4, 6, 8 and 10- ternary actuators as force generators. The considered MPRs are planar mechanisms (i.e. with three states being -1, 0 and 1) with one degree of revolute motion. The five MPRs mechanisms have similar range of generated moment so the validation (comparison of the performances) can be made using the same data testing.
- **Chapter 3** explains the detailed description of ISA methods. They are Look-Up Table (LUT) method, two NF methods based on Takagi-Sugeno, two NN methods, including their training algorithms which are based on Levenberg-Marquardt Algorithm (LMA) and Backpropagation Algorithm (BPA), and one HN method. Each of these methods is a computational machine of the five ternary MPRs that associates output ternary number *u* with *m* ternary digits (2, 4, 6, 8 and 10) to the input couple of continuum real numbers $X(\alpha, M)$.
- **Chapter 4** outlines the training and testing results from six ISA solutions in the table and figures.

- **Chapter 5** introduces some methods that suitable also for ISA solution, i.e.: Bolzmann Machine (BM), Radial Basis Function Network (RBFN), Jordan Recurrent Neural Network (JRNN), Wavelet Neural Network (WNN) and Support Vector Machine (SVM).

- **Chapter 6** summarizes the main contributions to the ISA solution. Suggestion to use more complicated m-ternary MPRs as well as to apply Multi-objective Optimization strategies for reducing the generalization error performance will be mentioned also in this chapter.

Chapter 2

Five Ternary MPRs Mechanisms

In this chapter, we discuss the five ternary MPRs that are considered in this study, as depicted in Figs. 1(a) to 1(e). They feature 2, 4, 6, 8 and 10 identical Crank and Slotted-Lever (CSL) respectively with 3RP planar mechanisms. The terms R and P are for revolute and prismatic joint respectively, sharing the same crank at its moving revolute joint, centered at point $A(\alpha)$. The common crank is hinged at the frame at point O, the m links with variable length $A(\alpha)B_i$, where i =1, 2, …, m; here m = 2, 4, 6, 8 and 10, are hinged at the common point $A(\alpha)$ and at points B_i respectively, symmetrically located with respect to the Y axis along the circular arc with radius $r = OB_i$ and with spread angle 2β, where $\beta = \frac{\pi}{2*m}$.

So the arc has angular span equal to $\pi - 2\beta$ for all mechanisms and the different angular positions between points B_i for all models are always equal to 2β. For more information about the m-ternary mechanism, Table 1 shows the complete variables such as angle β, the different angular position between points B_i, and the force amplitude F, which is determined from the equivalent force system of five proposed mechanisms.

TABLE 1. PARAMETERS FOR M-TERNARY MECHANISM

Description of ***m*-ternary mechanism**	***m*= 2**	***m*= 4**	***m*= 6**	***m*= 8**	***m*= 10**
Angle β (deg)	45	22.5	15	11.25	9
Position between points B_i(deg)	90	45	30	22.5	18
Force Amplitude *F* (N)	34.24	19.44	13.24	10	8
Possible Crank Moment each α	3^2	3^4	3^6	3^8	3^{10}
Radius r = OB_i (m)	0.38				
Crank $l = \| A(\alpha) - O \|$ (m)	0.1				

For more information, some information (geometry, mass, inertia) about the actuator arrays for 8-ternary MPRs are given by Di Canio thesis (2011) and can be seen in Table 2 and Table 3.

TABLE 2. MASS AND INERTIA OF THE 8-TERNARY MPRS SYSTEMS

Variable	Rigid Body	Cylinder	Piston
Mass (kg)	0.0203	0.0148	0.0320
Inertia (kg*m^2)	2.1938e-004	9.6981e-005	1.3892e-004

TABLE 3. POSITION OF POINT B_i OF THE 8-TERNARY MPRS SYSTEMS

Axis	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
X (m)	0.37	0.32	0.21	0.07	-0.07	-0.21	-0.32	-0.37
Y (m)	0.07	0.21	0.31	0.37	0.37	0.31	0.21	0.07
α (deg)	11.25	33.75	56.25	78.75	101.25	123.75	146.25	168.75

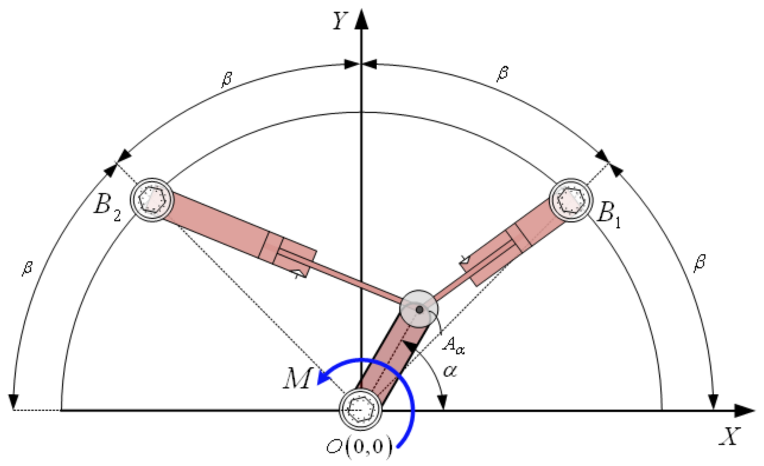
Furthermore, we also define the error intrinsic of the proposed MPRs mechanisms due to the discretization. The intrinsic error can be defined as the maximum error and the average error in the generation of a given desired moment. This error shows the accuracy of the m-ternary MPRs mechanisms and it can be compared with the error of proposed methods. To determine the intrinsic error, we define first the vector of the ascending moment that generated in the proposed MPRs and calculate the average error and the maximum error of moment between two closed data, as well as normalized of average error and normalized of maximum error with the maximum value of generated moment. The results of some information mentioned above can be seen in the Table 4.

TABLE 4. INTRINSIC ERROR OF M-TERNARY MPRS

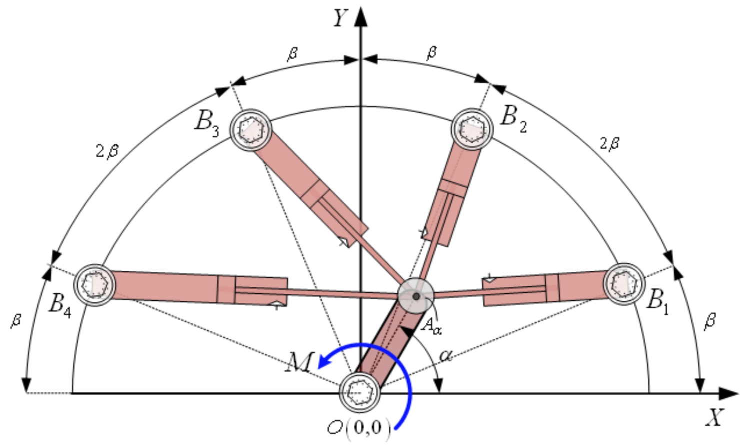
Ternary MPRs	**2-Ternary**	**4-Ternary**	**6-Ternary**	**8-Ternary**	**10-Ternary**
*E_avg =0.5*abs[m(i+1)-m(i)]/N*	0,074	0,0078	0,0017	0,0005	0,0002
*Max_err= 0.5*abs[max(m(i+1)-m(i))]*	0,2098	0,0890	0,0629	0,0443	0,0284
E_average/GM	0,0148	0,0016	0,0004	0,0001	0,00004
Max_err/GM	0,04196	0,01780	0,0126	0,0089	0,0057

Note: GM=max Torque of MPRs = 5Nm, N=length of vector moment m(i+1)

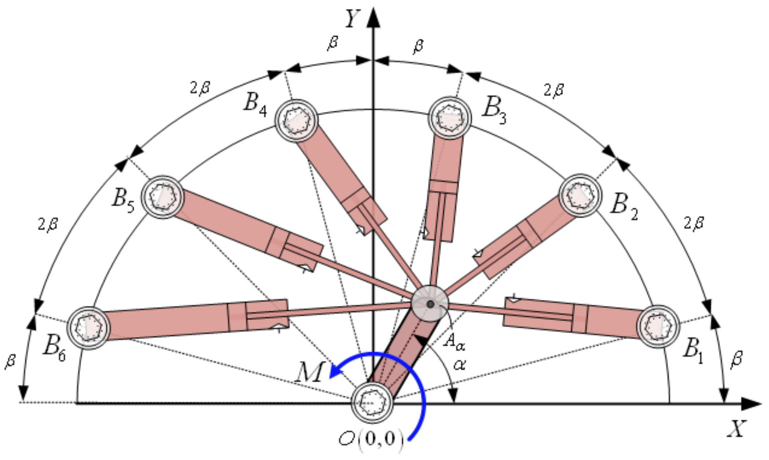
It can be inferred from Table 4 that increasing the number of ternary MPRs will reduce also the intrinsic error and increase the accuracy of the MPRs.



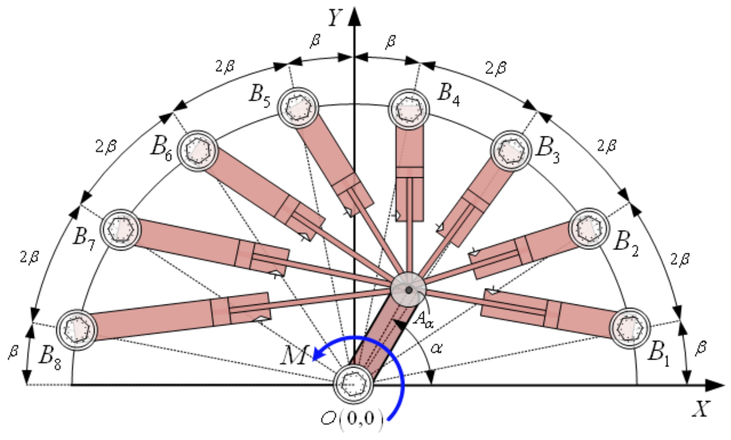
(a) 2-ternary MPRs



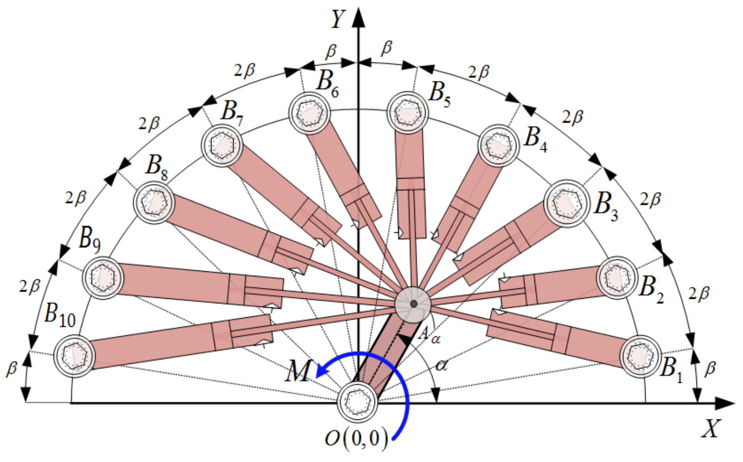
(b) 4-ternary MPRs



(c) 6-ternary MPRs



(d) 8-ternary MPRs



(e) 10-ternary MPRs

FIGURE 1. TERNARY MASSIVELY PARALLEL ROBOTS (MPRS) ACTUATED BY M-TERNARY FORCE GENERATORS: (A) 2-TERNARY, (B) 4-TERNARY, (C) 6-TERNARY, (D) 8-TERNARY, (E) 10-TERNARY

Furthermore, the output link of the considered MPRs is the common crank (similar to the all MPRs mechanisms). Differently from SLRs, crank motion is continuous and here it is limited in the range 0 ≤ α ≤ 180°. Discrete actuation is provided at the level of the *m*-P joints through identical three-state force generators which, irrespective of the relative position of slider and slotted lever, supply the forces (for *i* = 1, .., *m* and *m* = 2, 4, 6, 8, 10).

$$F_i = F\,u_i\left[A(\alpha)-B_i\right]/\left\|A(\alpha)-B_i\right\| \tag{2.1}$$

with *F* being a constant force magnitude that is shown in Table 1, and each mechanism has different values of *F* for creating the equivalent total force acting on them. Here u_i is the related activation state (either +1, 0 or –1) of the *i*-th actuator. Practical implementation of the mechanisms, as depicted in Figs. 1(a) to 1(e), could be obtained by employing two, four, six, eight and ten double-effect pneumatic cylinders with directional control valves in place of both slider and slotted-lever links.

By considering all force contributions for each mechanism, the resulting torque *M,* generated by the actuators on the output crank, can be written as follows:

$$M(\alpha, u_i) = F\sum_{i=1}^{m} u_i \left[k \cdot \left[A(\alpha) - O \right] \times \left[A(\alpha) - B_i \right] / \left\| A(\alpha) - B_i \right\| \right], \tag{2.2}$$

where k is the unit normal to the plane of motion of the mechanism.

Equation (2.2) represents the static equilibrium condition of the considered ternary MPRs (in this case we ignore the link weight and friction). Therefore, for any desired continuous value α^D (that is for any desired MPRs configurations), the Direct Static Analysis (DSA) problem amounts to find the torque M^*, within a range M of discrete values of M^*, which corresponds to a known combination of the activation states u_i^D. Conversely, for any desired continuous value α^D, the Inverse Static Analysis (ISA) problem amounts to find the best combination of the activation states

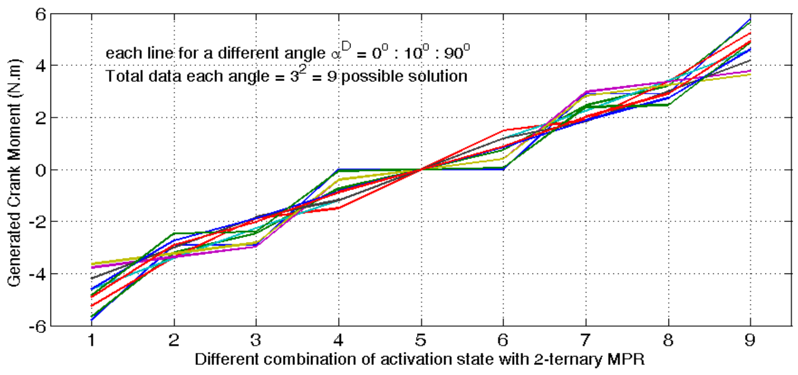
u_i^* (among a total of 3^m possibilities for any α^D) which enables the generation of the moment M^* (namely $M^* = M(\alpha^D, u_i^*)$) that more closely matches a desired torque M^D; that is, to find the state combination u_i^*, *i* = 1, …, *m*, and *m* = 2, 4, 6, 8 for which the error $e^* = \left\| M^D - M^* \right\|$ is

$$e^* = min_{u_i \in \{1,0,-1\}} \left\| M^D - M\left(\alpha^D, u_i\right) \right\| \qquad (2.3)$$

Notice that since the desired M^D can be any real value, whereas the range M is only a discrete subset, in general the minimum error e^* is different from zero. Moreover, owing to the discrete nature of the m variables u_i, the ISA described by Eq. (2.3) cannot be solved via standard pseudo-inverse methods.

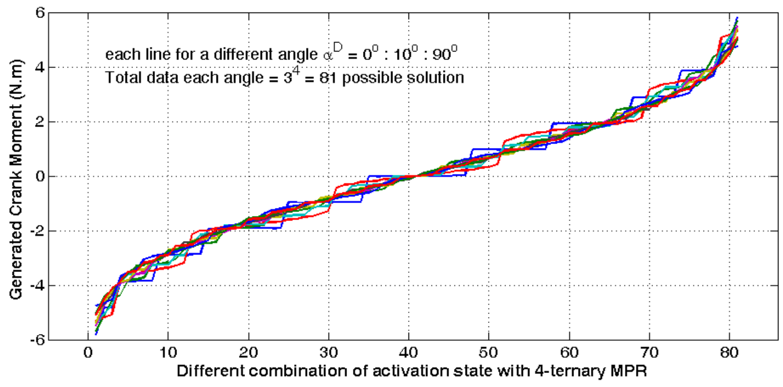
To give an idea of the range of available torques M that can be generated at the output crank of the MPRs described in Figs. 1(a) to 1(e), Eq. (2.2), with equal values of crank *l* and radius *r* and different force amplitude F like shown in Table 1, is plotted in Fig. 2(a) to 2(e) for 2, 4, 6, 8, 10-ternary MPRs respectively. Here we used ten different angular positions α^D ranging from 0 to 90° with 10° step. As presented in Figs. 1(a) to 1(e), even though with discrete activation, the MPRs are capable to generate torques in a range M having similar amplitude, but with different resolution. The 2 and 4-ternary MPRs produce 3^2 and 3^4 generated torques for each discrete alpha with less resolution while the 6, 8 and 10-ternary MPRs generate 3^6, 3^8 and 3^{10} torques with better resolution respectively.

In particular, each line in the Figs. 2(a) to 2(e) corresponds to a given crank position and represents all the torques that can be generated for all possible 3^m combinations of actuator activation states related to the *m*-ternary MPRs. It can be noted that, in all figures, the lines are drawn with the available moments sorted in ascending order. As shown in Fig. 2(a), despite the discrete behavior of the three-state actuators, the 2-ternary MPRs only provide 3^2 discrete solutions of generated Crank moment (in N.m) for every known angle. Unfortunately, these discrete solutions have a big gap between the two closed values and they cannot be compared to the generated Crank moment produced by the continuous model (the continuous model is a standard crank and slotted-lever mechanism with a single continuously regulated actuator attached in the common crank). The other MPRs mechanisms, as shown in Figs. 2(b) to 2(e), have 3^4, 3^6, 3^8, and 3^{10} discrete solutions of generated Crank moment for each respectively and these solutions might be sufficiently comparable to generated Crank moment of the continuous one.



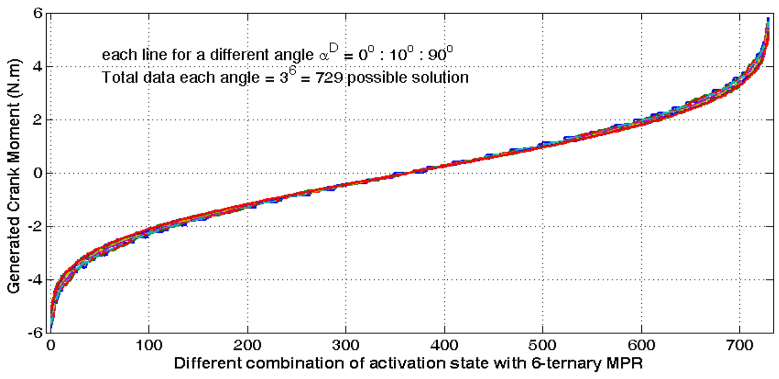
each line for a different angle $\alpha^D = 0^0 : 10^0 : 90^0$
Total data each angle = 3^2 = 9 possible solution
Generated Crank Moment (N.m)
Different combination of activation state with 2-ternary MPR

(a)



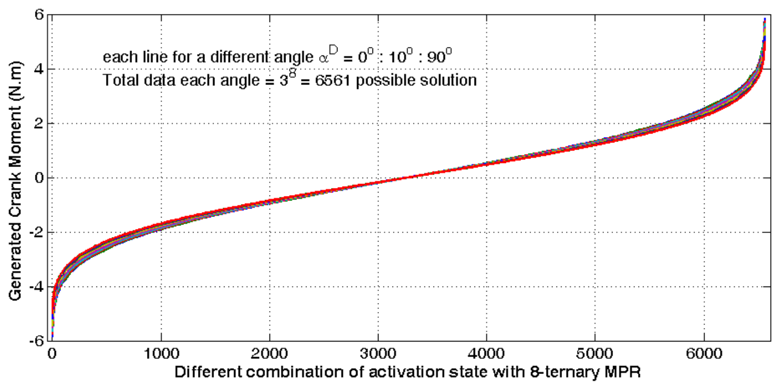
each line for a different angle $\alpha^D = 0^0 : 10^0 : 90^0$
Total data each angle = 3^4 = 81 possible solution
Generated Crank Moment (N.m)
Different combination of activation state with 4-ternary MPR

(b)

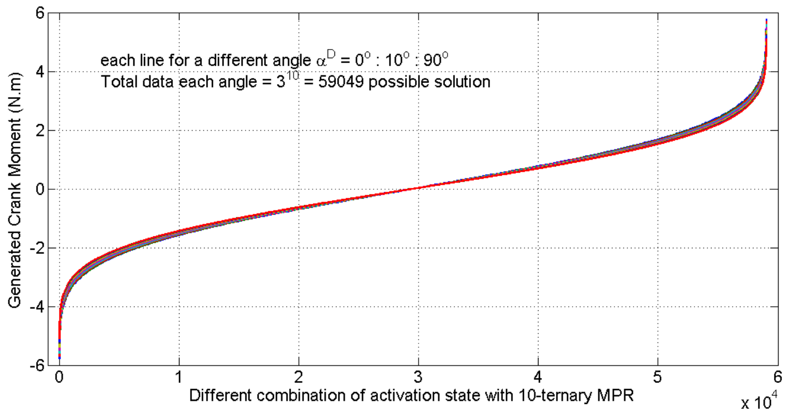


each line for a different angle $\alpha^D = 0^0 : 10^0 : 90^0$
Total data each angle = 3^6 = 729 possible solution
Generated Crank Moment (N.m)
Different combination of activation state with 6-ternary MPR

(c)



(d)



(e)

FIGURE 2. GENERABLE CRANK TORQUES AT 10-DIFFERENT ANGLES: A) 2-TERNARY MPRS; B) 4-TERNARY MPRS; C) 6-TERNARY MPRS; D) 8-TERNARY MPRS; E) 10-TERNARY MPRS

Additionally, due to the possibility of spatially distributing the partitioned actuation system, the considered MPRs also exhibit a rather identical torque generation capability within their full range of motion ($0 \leq \alpha \leq 180°$). Notice that this latter feature cannot be achieved by a standard

CSL mechanism actuated by a single continuously regulated force generator.

Further step is to use the generated moments from Figs. 2(a) to 2(d) and their corresponding angle α^D as input to the AI methods that will be explained in Chapter 3.

Chapter 3

Inverse Static Analysis Models

3.1 Introduction to the ISA models

This chapter presents six different methods for the solution of the ISA problem described in the previous chapter. Namely: one Look-Up Table model; two Neuro-Fuzzy models; and three Neural Network models. Essentially, each of these models is a computational machine of the five ternary MPRs, shown in Figs. 1(a) to 1(e), that associates an output ternary number $\boldsymbol{u} = [u_1, \ldots, u_m]$ with m = 2, 4, 6, 8 and 10 to an input couple of continuum real numbers $\boldsymbol{X} = [X_1, X_2] = [\alpha, M]$.

Initial set-up of all these methods requires the knowledge of an appropriate input-output ($\boldsymbol{X}$-$\boldsymbol{u}$) dataset Δ with finite dimensions. Here, Δ consists of $10 \cdot 3^m$ $\boldsymbol{X}$-$\boldsymbol{u}$ correspondences that are generated via Eq. (2.2) for ten different values of α, ranging from 0 to 90° with 10° step, and for all possible (3^m) combinations of $\boldsymbol{u}$ of each m-ternary MPRs (note that all the $\boldsymbol{X}$-$\boldsymbol{u}$ correspondences contained in Δ satisfy Eq. (2.3) with $e^* = 0$). Given the continuity of α, Δ is not an exhaustive enumeration of all the possible solutions of the ISA problem. Thus the considered methods are required to provide some generalization ability (that is the ability to find ***X***-***u*** correspondences for arbitrary values of α which are not contained within Δ).

To discuss about their suitability in real-time control applications, the six ISA solutions are compared in the terms of time of off-line preparation t_p, time of on-line calculation t_c, modeling error e_m (i.e. the error calculated via Eq. (2.3) in predicting ***X***-***u*** correspondences for input pairs $\boldsymbol{X}^{\boldsymbol{D}}$ = [α^D, M^D] contained in Δ), and generalization error e_g (i.e. the error calculated via Eq. (2.3) in predicting ***X***-***u*** correspondences for input pairs $\boldsymbol{X}^{\boldsymbol{D}}$ not contained in Δ).

3.2 Look-Up Table model (LUT)

3.2.1 Introduction

The Look-Up Table (LUT) model is a brute-force search approach and it is the simplest method considered here. LUT model uses a stored data structure as a pattern collection of the entire dataset Δ as described above. As such, LUT model does not require any learning algorithm. During model preparation, the input values ***X*** of Δ are first normalized between 0 and 1, then the modified dataset Δ is sorted and stored row by-row in an array.

During model usage, the desired inputs $\boldsymbol{X}^{\boldsymbol{D}}$ are first normalized, then they are compared to the corresponding entries of the LUT using a row-by-row similarity procedure, and finally the suitable outputs u^{*}_{i} (for i = 1, .., m, here m = 2, 4, 6, 8) are chosen from the LUT row which provides the minimum error between desired and stored inputs.

3.2.2 Implementation of LUT into the *m*-ternary MPRs

Implementation of brute force using LUT is simple concerning the time preparation, but it is not a recommended ISA solution regarding the on-line computation and generalization error. The LUT model performances of the ISA solution for *m*-ternary MPRs can be compared in the Table 5 below:

TABLE 5. LUT PERFORMANCE OF THE *M*-TERNARY MPRS

Description	**2-ternary**	**4-ternary**	**6-ternary**	**8-ternary**	**10-ternary**
Δ used (%)	100	100	33	12	4
off-line preparation t_p (s)	0.1	0.3	3.2	48	2.1e3
on-line calculation t_c (s)	0.008	0.009	0.013	0.35	2.8
modeling error e_m (N)	0	0	0	0	0
generalization error e_g (N)	2.594	1.664	0.891	0.805	0.551

We can see from the Table 5 that on-line calculation of LUT exponentially increase from t_c =0.008s (m=2) to t_c =2.8s (m=10). Moreover, the comparison of t_p, t_c e_m and e_g of LUT model with other ISA methods will be provided later in Chapter 4.

3.3 Neuro-Fuzzy (NF) models

3.3.1 Introduction

In the field of artificial intelligence, NF refers to combinations of artificial neural networks and fuzzy logic. This idea was proposed first by J. S. R. Jang [1993] and later was improved Palit et.al. [2001, 2002a, 2002b]. NF is a hybrid intelligent system, which combines the human-like reasoning style of fuzzy systems with the learning ability of neural networks. Moreover, a number of publications reported on the applications using NF network such as: adaptive control of inverted pendulum using NF inference by Kumar V. et al [2010]; recurrent NF hybrid-learning approach to accurate system modeling by Cheng K.H. et al [2007]; and electrical load forecasting using a neural-fuzzy approach by Popovich D. et al [2009].

In the following section, we proposed two NF models which are based on the Neuro-Fuzzy Takagi-Sugeno (NFTS) inference scheme with Gaussian membership functions. They are NFTS and the Look-Up Table version of NFTS, which is called as NFLUT. Concerning the ISA problem, both proposed models can be applied as solutions because they provide a strong connection between input values ***X*** of Δ with their output variables ternary number ***u*** = [u_1, …, u_m]. Moreover, they also have advantages such as: it interprets IF-THEN rules from input-output relations; and it focuses on accuracy of the output network and efficient time consumption for on-line computation.

3.3.2 Neuro-Fuzzy architecture

In this section, the architecture of two considered models is presented. Both models are based on the same overall architecture and only differ in the defuzzification operation, like depicted in Fig. 3. The architecture is called as feedforward Neuro-Fuzzy type Takagi-Sugeno multi-input multi output. It uses Gaussian membership function in the fuzzyfication phase.

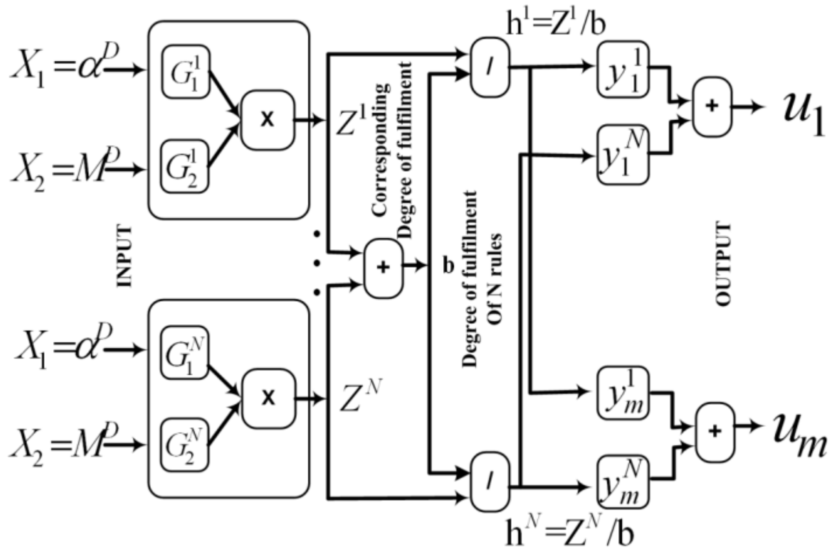


FIGURE 3. TAKAGI-SUGENO-TYPE MIMO (WITH INPUT REAL AND OUTPUT TERNARY) FEEDFORWARD NEURO-FUZZY NETWORK, NO. INPUT = 2, NO. OUTPUT = M, M= 2, 4, 6, 8,10; NO. MEMBERSHIP FUNCTION N, TRAINING METHOD: LMA

In particular, introducing the Gaussian membership functions to both NF methods G_j^n ($j = 1, 2$; $n = 1, \ldots, N$), as a fuzzyfication procedure for input pairs $\boldsymbol{X}^D = [\alpha^D, M^D]$.

$$G_j^n(X_j) = exp\left[-\left(\left(X_j - c_j^n\right)/\sigma_j^n\right)^2\right] \tag{3.1}$$

with characteristic means c_j^n and variance σ_j^n together with the corresponding fuzzy rules R^n can be written as:

$$\begin{aligned} &R^n : IF\ X_1 is\ G_1^n\ AND\ X_{21} is\ G_2^n \\ &THEN\ y_i^n = w_{0i}^n + w_{1i}^n X_1 + w_{2i}^n X_2 \end{aligned} \tag{3.2}$$

with w_{0i}^{n}, w_{1i}^{n} and w_{2i}^{n} (for *i* = 1, …, *m*, and *n* = 1, …, *N, N* is the number of optimized rules for each *m*-ternary model, here N = 3, 8, 10, 11 and 17) being the Takagi-Sugeno weights [Takagi 1985], the common part of the two considered Neuro-Fuzzy models calculates the continuous variables.

$$\bar{u}_i = \sum_{n=1}^{N} y_i^n \left[\prod_{j=1}^{2} G_j^n \left(X_j \right) \Big/ \sum_{n=1}^{N} \prod_{j=1}^{2} G_j^n \left(X_j \right) \right] \tag{3.3}$$

From Eq. (3.3), the two different models, hereafter briefly referred to as NFTS and NFLUT, are derived by alternatively estimating the actuator activation states u_i through one of the following threshold operations:

$$u_i = round(\bar{u}_i) \text{ or } u_i = RLUT(\bar{u}_i) \tag{3.4}$$

where *RLUT* indicates a properly Reduced Look-Up Table involving $\bar{u}_i$ as only input of the table. Additionally, the NFLUT requires the generation of the RLUT, which is here constructed by storing the most significant $\boldsymbol{u}$-$\bar{\boldsymbol{u}}$ correspondences that occurred during training with the known dataset **Δ**.

Prior to their use, NFTS and NFLUT models require the tuning of the parameters c_j^n, σ_j^n, w_{0i}^{n}, w_{ji}^{n} (for *j* = 1, 2; *i* = 1, …, *m*; *n* = 1, …, *N*; in the following *N* = 3, 8, 10, 11, 17). Here, the number of parameters for the considered MPRs are 30, 128, 220, 308 and 578 parameters for m = 2, 4, 6, 8 and 10 correspondingly. The values of these parameters are found by an optimized learning procedure. The learning procedure employs 100%, 100%, 33%, 12% and 4% of the ***X***-***u*** correspondences known from Δ for the 2, 4, 6, 8 and 10-ternary MPRs respectively. Here, Δ consists of $10 \cdot 3^m$ ***X***-***u*** correspondences that generated from Eq. (2.2). In particular, the learning procedure is performed via the Levenberg-Marquardt Algorithm (LMA), explained in Section 3.3.3, which is a fast second order training routine for NFTS network [Palit 2001, 2005].

3.3.3 LMA Training

The fuzzy logic system, once represented as the equivalent Multi-Input Multi-Output feed forward network, can generally be trained using any suitable training algorithm, such as standard Backpropagation Algorithm (BPA) that is generally used for training of the NN (Palit 2002b). Because of its slow speed of convergence, BPA needs to be further improved. Alternatively, a second order training algorithm, such as the Levenberg-Marquardt Algorithm (LMA), can also be used. It is noted that LMA is actually a second order training algorithm that is based on the modification of Newton's method and uses Jacobian matrix in order to approximate the second-order partial derivatives (called as Hessian Matrix).

Recently, LMA has some additional features such as momentum, modified error index (MEI) and oscillation control [proposed by Palit et al 2001, 2002a, 2005; Xiaosong et al 1995]. Briefly, the features mentioned above can be put in the updating procedures of LMA and they can be described as follows:

- Momentum version of LMA is proposed by adding a small adaptive factor of momentum (*mo*) into the updated equation, so the learning process can be accelerated in the iteration.
- MEI version of LMA is done by calculating the different between the error performance and the average error, multiplying with the constant factor that should be chosen properly.
- By adding both momentum and MEI in the training procedure, the speed convergence of the error performance is much faster than standard algorithm. To guarantee the every iteration will reduce the error performance, oscillation control procedure must be applied by a given limit Wildness Factor (WF). By doing this way in the NF training, the value of the error performance is either decreased steadily or at least stayed within the given limit of WF.

In this thesis, the error performances are calculated in the terms of Root Mean Squared Error (RMSE) and integrated with early stopping procedure to stop the iteration when the error requirement is achieved. The simple search procedure, called as Hill Climbing (HC), is used for finding the best parameters, in which the parameters give a minimum error. The HC and the early stopping procedures will be explained later in Appendix A and B respectively.

Moreover, to be applied in the NF methods, we need to overcome the complexity of the calculation of second order equations in the LMA. To avoid this, Hagan and Menhaj [26] implemented the LMA without direct computation of second order term. This is achieved in the following way. If a function $S(\boldsymbol{v})$ is minimized with respect to the parameter vector $\boldsymbol{v}$ (for the NF method as depicted in Figure 3, these parameters are the total network parameters, i.e.: $c_j^n, \sigma_j^n, w_{0i}^n, w_{ji}^n$), the next parameter and the updated parameter vector $\boldsymbol{v}(k+1)$ and $\Delta\boldsymbol{v}$ can be defined as:

$$\Delta\boldsymbol{v} = -[\nabla^2 S(\boldsymbol{v})]^{-1} \cdot \nabla S(\boldsymbol{v}) \tag{3.5a}$$

$$\boldsymbol{v}(k+1) = \boldsymbol{v}(k) + \Delta\boldsymbol{v} \tag{3.5b}$$

where $\nabla S(\boldsymbol{v})$ is the gradient of $S(\boldsymbol{v})$ (i.e. the vector containing the derivative of S with respect to vector $\boldsymbol{v}$) and $\nabla^2 S(\boldsymbol{v})$ is the Hessian matrix (i.e. the matrix containing the derivative of $\nabla S(\boldsymbol{v})$ with respect to vector $\boldsymbol{v}$). Considering a training set composed by N_T samples, for the NF method depicted in Figure 3 the function $S(\boldsymbol{v})$ is taken to be the following Sum Squared Error (SSE) function:

$$S(\boldsymbol{v}) = 0.5 \cdot \sum_{p=1}^{N_T} \sum_{i=1}^{M} \left(e_i^p(\boldsymbol{v})\right)^2, \tag{3.6a}$$

$$e_i^p(\boldsymbol{v}) = u_i^p - u_i^{D,p}, \tag{3.6b}$$

where the term e_i^p is the error between the predicted and the desired *i*-th output of the network for the *p*-th training sample. Then, the *a*-th component of the gradient $\nabla S(\boldsymbol{v})$ and the *ab*-th component of the Hessian matrix $\nabla^2 S(\boldsymbol{v})$ (*a* being the row index and *b* being the column index) give result as:

$$\partial S(\boldsymbol{v})/\partial v_a = \sum_{p=1}^{N_T} \sum_{i=1}^{M} \left(e_i^p(\boldsymbol{v}) \cdot \partial e_i^p(\boldsymbol{v})/\partial v_a\right) \tag{3.7a}$$

$$\partial^2 S(\boldsymbol{v})/\partial v_a \partial v_b = \\ \sum_{p=1}^{N_T} \sum_{i=1}^{M} \left(e_i^p(\boldsymbol{v}) \cdot \partial^2 e_i^p(\boldsymbol{v})/\partial v_a \partial v_b\right) + \\ + \sum_{p=1}^{N_T} \sum_{i=1}^{M} \left(\partial e_i^p(\boldsymbol{v})/\partial v_a \cdot \partial e_i^p(\boldsymbol{v})/\partial v_b\right) \tag{3.7b}$$

For the Gauss-Newton method, the first term in (3.7b) is assumed to be zero. Then, by condensing all the components of the error $e_i^p(\boldsymbol{v})$ in a single vector $\boldsymbol{e}(\boldsymbol{v})$ (with dimension $(N_T \cdot M)$) and all the components

$\partial e_i^p(\boldsymbol{v})/\partial v_a$ in a single Jacobian matrix $\mathbf{J}(\boldsymbol{v})$ (with dimension $(N_T \cdot M) \times N_P$, with N_P being the number of network parameters), then Equations (3.7a) and (3.7b) can be rewritten as

$$\nabla S(\boldsymbol{v}) = \mathbf{J}^T(\boldsymbol{v}) \cdot \boldsymbol{e}(\boldsymbol{v}) \tag{3.8a}$$

$$\nabla^2 S(\boldsymbol{v}) = \mathbf{J}^T(\boldsymbol{v}) \cdot \mathbf{J}(\boldsymbol{v}) \tag{3.8b}$$

Therefore, the updated equations according to (3.5a) will be:

$$\Delta \boldsymbol{v} = -[\mathbf{J}^T(\boldsymbol{v}) \cdot \mathbf{J}(\boldsymbol{v})]^{-1} \cdot \mathbf{J}^T(\boldsymbol{v}) \cdot \boldsymbol{e}(\boldsymbol{v}) \tag{3.9a}$$

Now let us see the Levenberg-Marquardt’s modifications of the Gauss-Newton method, based on Hagan and Menhaj [26]:

$$\Delta \boldsymbol{v} = -[\mathbf{J}^T(\boldsymbol{v}) \cdot \mathbf{J}(\boldsymbol{v}) + \mu \cdot \mathbf{I}]^{-1} \cdot \mathbf{J}^T(\boldsymbol{v}) \cdot \boldsymbol{e}(\boldsymbol{v}) \tag{3.9b}$$

where, **I** is the $N_P \times N_P$ identity matrix, and the damping term μ is multiplied or divided by some factor whenever the iteration step increases or decreases the value of $S(\boldsymbol{v})$. This thesis uses adaptive μ when training algorithm is processed as follows: for large μ, the algorithm becomes the steepest descent algorithm with step size $1/\mu$ (similar to BPA), and for small μ, it becomes the Gauss-Newton method.

Here, the updated equation according to (3.5b):

$$\boldsymbol{v}(k+1) = \boldsymbol{v}(k) - [\mathbf{J}^T(\boldsymbol{v}) \cdot \mathbf{J}(\boldsymbol{v}) + \mu \cdot \mathbf{I}]^{-1} \cdot \mathbf{J}^T(\boldsymbol{v}) \cdot \boldsymbol{e}(\boldsymbol{v}) \tag{3.9c}$$

Now, the computation of Jacobian matrix can be performed as follows. If the adjustable parameters of neuro-fuzzy networks v are defined as: c_j^n, σ_j^n, w_{0i}^n, w_{ji}^n, the gradient ∇S can be substituted by corresponding chain rule differentiations (see Palit, 2005 for details). Differentiating with respect to w_{0i}^n yields:

$$\partial S / \partial w_{0i}^n = \sum\nolimits_{p=1}^{N_T} \left(Z_p^n / b_p \right) \cdot e_i^p \tag{3.10}$$

With a similar procedure, the component of the gradient with respect to the parameter w_{ji}^n of the NF network can be written as:

$$\partial S / \partial w_{ji}^n = \sum\nolimits_{p=1}^{N_T} \left(Z_p^n / b_p \right) \cdot e_i^p \cdot X_j^p \ . \tag{3.11}$$

Now, the computation of the gradient for the remaining parameters c_j^n and σ_j^n can be written as follows (Palit, 2005):

$$\partial S / \partial c_j^n = \sum\nolimits_{p=1}^{N_T} \left(\sum_{i=1}^{m} \left(y_i^{np} - u_i^p \right) \cdot e_i^p \right) \cdot \left[2 \cdot \frac{Z_p^n}{b_p} \cdot \left(X_j^p - c_j^n \right) / \left(\sigma_j^n \right)^2 \right] \tag{3.12}$$

and

$$\partial S / \partial \sigma_j^n = \sum\nolimits_{p=1}^{N_T} \left(\sum_{i=1}^{m} \left(y_i^{np} - u_i^p \right) \cdot e_i^p \right) \cdot \left[2 \cdot \frac{Z_p^n}{b_p} \cdot \left(X_j^p - c_j^n \right)^2 / \left(\sigma_j^n \right)^3 \right]. \tag{3.13}$$

To solve (3.12) and (3.13), we need to define the new term

$$B_n = \sum_{p=1}^{N_T} \sum_{i=1}^{m} \left(y_i^{np} - u_i^p \right) \cdot e_i^p \; . \tag{3.14}$$

Let us denote

$$D_i^{np} = \left(y_i^{np} - u_i^p \right) . \tag{3.15}$$

So that, equation (3.14) reduces as

$$B_n = \sum_{p=1}^{N_T} \sum_{i=1}^{m} D_i^{np} \cdot e_i^p \tag{3.16}$$

The objective of doing (3.16) is to find the terms D_{eqv}^{np} such that

$$B_n \equiv D_{eqv}^{np} \cdot e_{eqv}^p , \tag{3.17}$$

where

$$e_{eqv}^p = \sqrt{\left(e_1^p\right)^2 + \left(e_2^p\right)^2 + \cdots + \left(e_m^p\right)^2} , \tag{3.18}$$

In the equations reported above (3.15), y_i^{np} is the output Takagi-Sugeno from the consequent part of Eq. (3.2), with $y_i^{np} = w_{0i}^n + w_{1i}^n X_1^p + \ldots + w_{ji}^n X_2^p$; and the vector $\boldsymbol{e}_{eqv} = \left[e_{eqv}^1 , .., e_{eqv}^p , ..., e_{eqv}^{N_T} \right]^T$ contributes the same amount of sum squared error that can be obtained jointly by all the errors e_i^p from the MIMO network.

From (3.17), the components of matrix $\left[D_{eqv}^{np} \right]$ can be determined as:

$$D_{eqv}^{np} = B_n e_{eqv}^{q} \left[e_{eqv}^{p} e_{eqv}^{q} \right]^{-1} \tag{3.19}$$

After defining (3.19), we can rewrite (3.12) and (3.13) as follows:

$$\partial S / \partial c_j^n = \sum_{p=1}^{N_T} D_{eqv}^{np} \cdot e_{eqv}^{p} \cdot \left[2 \cdot \frac{Z_p^n}{b_p} \cdot \left(X_j^p - c_j^n \right) / \left(\sigma_j^n \right)^2 \right] \tag{3.20}$$

and

$$\partial S / \partial \sigma_j^n = \sum_{p=1}^{N_T} D_{eqv}^{np} \cdot e_{eqv}^{p} \cdot \left[2 \cdot \frac{Z_p^n}{b_p} \cdot \left(X_j^p - c_j^n \right)^2 / \left(\sigma_j^n \right)^3 \right]. \tag{3.21}$$

The above procedure describes actually layer by layer computation of Jacobian matrices for all parameters of neuro-fuzzy network [Palit 2005]. Again, after finishing all computation, then back to the Eq. (3.9c) for updating the parameters. This updating procedure stops after achieving the maximum iteration or the minimum error function.

3.3.4 The optimized results of NF architectures

In order to find the best initial parameter vector w in Eq. (3.5) - these parameters are total parameters, i.e.: c_j^n, σ_j^n, w_{0i}^n, w_{ji}^n - and to be updated in the training algorithm, we proposed randomized Hill Climbing (HC) procedure in order to find the optimized number of rules *N* for each *m*-ternary models. This procedure is a local search algorithm that tries to find the best local minimum from the large number of iteration procedures by permitting the best training parameters that minimize the error model (e_m) and neglecting the others. The optimized N membership function after three weeks searching time as the results of HC procedure that explained in Appendix **A** are: N=3, 8, 10, 11 and 17 for the 2, 4, 6, 8 and 10-ternary respectively.

Regarding model performances for the ISA solution for 2-ternary: NFTS exhibits t_p = 2.2s, t_c = 8e-5s, e_m = 0.871N and e_g = 1.051N; NFLUT exhibits t_p= 2.3*s*, t_c=5e-4*s*, e_m = 0 and e_g = 1.283N; for the ISA solution for 4-ternary: NFTS shows t_p = 43s, t_c = 1e-4s, e_m = 0.909N and e_g = 0.909N; while the NFLUT shows t_p = 43.5*s*, t_c = 3e-3*s*, e_m = 0 and e_g = 0.611N; for the ISA solution for 6-ternary: NFTS exhibits t_p = 185s, t_c = 2e-4s, e_m = 0.917N and e_g = 0.932N; NFLUT exhibits t_p = 186*s*, t_c = 5e-3*s*, e_m = 0 and e_g = 0.498N; for the ISA solution for 8-ternary: NFTS shows t_p = 965s, t_c=1.4e-3s, e_m = 0.658N and e_g = 0.985N; while the NFLUT shows t_p = 983*s*, t_c = 1.4e-2*s*, e_m = 0 and e_g = 0.528N; and for the ISA solution for 10-ternary: NFTS shows t_p = 7.1e3s, t_c=1.9e-3s, e_m = 0.629N and e_g = 0.998N; while the NFLUT shows t_p = 7.3e3*s*, t_c = 3.8e-2*s*, e_m = 0 and e_g = 0.397N;

Additionally, the comparisons of NF methods with other ISA solutions are provided also in Chapter 4.

3.4 Neural Network models

3.4.1 Introduction

In general, the Neural Network (NN) is inspired by the human biological nervous system whereas the NN consists of highly interconnected networks with a large number of processing elements (called artificial neurons), which resemble the human brain system The advantages of using NN are: a) it is an efficient pattern recognition tool; and b) it acts like a parallel distributed processing (parallel computing), which makes it possible to accelerate the computational process.

We introduce the next extended network that improves the process of pattern recognition of NN as recurrent network, or mostly called as Recurrent Neural Network (RNN) and its similar architecture, a RNN without feedback, also known as Multi-Layer Perceptron (MLP). The RNN has short-term memory features that enable NN to achieve time-dependent mappings [Elman 1990, Juang 2002, Palit 2005, Toha 2008]. Besides two proposed methods above, some architectures that based on NN, such as Boltzmann machine, Radial Basis Function Network (RBFN) and Jordan RNN architectures, that well known in engineering practice will be discussed on

Chapter 5 briefly as challenged competitors for ISA solutions of ternary MPRs.

3.4.2 RNN architecture

Two Neural Network methods considered here are based on Recurrent Neural Networks (RNN) with hyperbolic-tangent activation functions. Additionally the third NN method, which is a probabilistic weight Hopfield Network (HN), will be explained in Section 3.5 with its performance results. Moreover, both RNN that described in this section are dynamic models that feature short-term memory and have capability to represent time-dependent mappings [Palit 2005]. Both models are based on the same overall architecture and only differ in the presence or absence of the context layer.

In particular, for a given input $\boldsymbol{X}(t) = [X_1(t), X_2(t)] = [\alpha(t), M(t)]$ at the time step t, both models calculate the actuator states $u_i(t)$ (for $i = 1, \ldots, m$) as:

$$u_i(t) = \mathrm{round}\left[G\left(b_{2i} + \textstyle\sum_{l=1}^{L} w_{il}^{HO} G\left(\alpha_l(t)\right)\right)\right] \tag{3.22}$$

with $$G(y) = y/\sqrt{1+y^2} \tag{3.23}$$

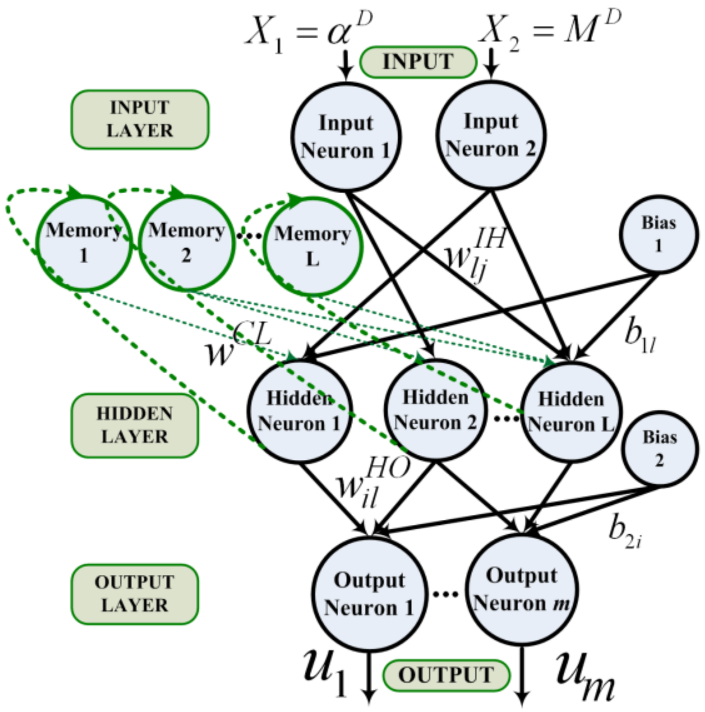


FIGURE 4. ELMANN-RNN, NO. INPUT = 2, NO. OUTPUT = M, M = 2, 4, 6, 8,10; NO. CONTEXT LAYER L = 21, 22, 25, 27, 30, TRAINING METHOD: BPA

$$\alpha_l(t) = \sum_{j=1}^{2} w_{lj}^{IH} X_j(t) + w^{CL}\alpha_l(t-1) + b_{1l} \tag{3.24}$$

with $\alpha_l(0) = 0$.

In the above mentioned equations, $\alpha_l(t)$ is the total input of *L*-hidden neuron and b_{1l} , b_{2i} , w_{il}^{HO}, w_{lj}^{IH} , and w^{CL} (for *i* = 1, …, *m*; *j* = 1, 2; l = 1, …, *L*) are model parameters need to be determined. From Eq. (3.22), the two different models, hereafter briefly referred to Multi-Layer Perceptron Neural Network (MLP) and RNN type Elman (ERNN), are derived by respectively selecting w^{CL} = 0 and w^{CL} = 1/*L*.

In addition, we need to establish the number of network parameter (L) of the m-ternary MPRs which is L = 107, 158, 231, 305 and 420 parameters for 2, 4, 6, 8 and 10-ternary respectively. These are found by the optimized three weeks learning procedure which employs 100% for the 2 and 4-ternary MPRs, 33% , 12% and 4% for the 6, 8 and 10-ternary MPRs of the $\boldsymbol{X}$-$\boldsymbol{u}$ correspondences known from the dataset Δ.

In particular, the learning is performed here via an accelerated version of the Back-Propagation Algorithm (BPA) [Palit 2005, Pasila 2006], which is explained more detailed in Section 3.4.3.

3.4.3 Backpropagation Algorithm (BPA) on NN

The Multi-Input Multi-Output RNN that is represented in Fig. 4 can generally be trained using suitable training algorithms. Some standard training algorithms are Backpropagation Algorithm (BPA) and Levenberg-Marquardt Algorithm (LMA). BPA, the standard algorithm for NN training, is a supervised learning technique based on delta rule procedure (a gradient descent method). It was first described by P. Werbos in 1974, and further developed by D.E. Rumelhart, G.E. Hinton and R.J. Williams in 1986. This algorithm is a learning rule for multi-layered Neural Networks. It is not only useful for feed-forward networks (networks without feedback, or simply, that have no connections that loop), but also for networks with feedback, such as recurrent network. The term BPA is an abbreviation for "backwards propagation of errors". The BPA is used to calculate gradient of error of the network with respect to the network's modifiable weights. This gradient descent method is almost used in a NN learning algorithm because it is a simple procedure for finding the suitable parameters that minimized the error training. To increase the accelerate learning in BPA, we introduce additional momentum and oscillation into the standard version of BPA.

In order to start the BPA procedure, we assume that data pairs input-output of five ternary models is the ***X***-***u*** correspondences and already known from dataset $\Delta\,(X_j^p, u_i^p)$. Here, the term j, i and p denote as the number of input-output in the networks related to the number of correspondence data training from data set Δ. The goal is to find network output, so that the performance, Sum Squared Error (SSE) is defined as:

$$SSE_i = 0.5 \cdot \sum_{p=1}^{N_T} \left(e_i^p \right)^2 \tag{3.25}$$

with the total SSE is minimized.

$$SSE = \sum_{i=1}^{m} SSE_i = \left(SSE_1 + SSE_2 + \ldots + SSE_m \right) \tag{3.26}$$

Here $e_i^p = u_i^p - u_i^{D,p}$ is the error of each output networks with *i* =1,…,*m* is the number of output networks, *p* = 1, …, N_T is the number of data training from dataset Δ, and u_i^p and $u_i^{D,p}$ are the predicted and desired outputs. The problem of learning is how BPA works to adjust parameters of the RNN (b_{1l}, b_{2i}, w_{il}^{HO} and w_{lj}^{IH}), so that SSE can be minimized. To answer this question, we introduce the gradient steepest descent rule for training of feed-forward neural network, which is based on the recursive equations (note: recursive equation is an equation that is used to determine the next term of a sequence using one or more of the preceding terms). By using recursive mode, we could determine the updated parameters without using the differential calculation [Palit, 2005]:

$$w_{lj}^{IH}(k+1) = w_{lj}^{IH}(k) - \eta \cdot \left(\frac{\partial SSE}{\partial w_{lj}^{IH}} \right) \tag{3.27a}$$

$$w_{il}^{HO}(k+1) = w_{il}^{HO}(k) - \eta \cdot \left(\frac{\partial SSE}{\partial w_{il}^{HO}} \right) \tag{3.27b}$$

$$b_{1l}(k+1) = b_{1l}(k) - \eta \cdot \left(\frac{\partial SSE}{\partial b_{1l}} \right) \tag{3.27c}$$

$$b_{2i}(k+1) = b_{2i}(k) - \eta \cdot \left(\frac{\partial SSE}{\partial b_{2i}} \right) \tag{3.27d}$$

where SSE is the performance function at the k_{th} iteration step and $w_{lj}^{IH}(k+1)$, $w_{il}^{HO}(k+1)$, $b_{1l}(k+1)$ and $b_{2i}(k+1)$ are the updated parameters on $(k+1)$ step. The starting values of those parameters are selected randomly in the range of 0 and 1. Moreover, constant learning rate η should be chosen properly. For convergence reason, practically a very small learning rate $\eta << 1$ is chosen.

Furthermore, we need to solve the corresponding chain rules of the last part in the right side Eqs. (3.27a) to (3.27d) so those equations have no differential part:

$$\left(\frac{\partial SSE}{\partial w_{lj}^{IH}}\right) = \sum_{i=1}^{m}\left(\frac{\partial SSE_i}{\partial a_{lp}}\right)\cdot\left(\frac{\partial a_{lp}}{\partial w_{lj}^{IH}}\right) \tag{3.28a}$$

$$\left(\frac{\partial SSE}{\partial w_{il}^{HO}}\right) = \left(\frac{\partial SSE_i}{\partial u_{ip}}\right)\cdot\left(\frac{\partial u_{ip}}{\partial w_{il}^{HO}}\right) \tag{3.28b}$$

$$\left(\frac{\partial SSE}{\partial b_{1l}}\right) = \sum_{i=1}^{m}\left(\frac{\partial SSE_i}{\partial a_{lp}}\right)\cdot\left(\frac{\partial a_{lp}}{\partial b_{1l}}\right) \tag{3.28c}$$

$$\left(\frac{\partial SSE}{\partial b_{2i}}\right) = \left(\frac{\partial SSE_i}{\partial u_{ip}}\right)\cdot\left(\frac{\partial u_{ip}}{\partial b_{2i}}\right) \tag{3.28d}$$

Here, $a_{lp} = G(\alpha_{lp})$ is the output *L*-hidden layer after the calculation of activation function in (3.23). Moreover, by using chain rule procedure as explained by Palit [2005, p.97-98], eqs. (3.28a) to (3.28d) can be written as:

$$\left(\frac{\partial SSE}{\partial w_{lj}^{IH}}\right) = \sum_{i=1}^{m}\sum_{p=1}^{N_T}\left(\left[e_i^p \cdot w_{il}^{HO}\right]\cdot\left[1-\left(a_l^p\right)^2\right]\right)\cdot X_j^p \tag{3.29a}$$

$$\left(\frac{\partial SSE}{\partial w_{il}^{HO}}\right) = \sum_{p=1}^{N_T} e_i^p \cdot a_l^p \tag{3.29b}$$

$$\left(\frac{\partial SSE}{\partial b_{1l}}\right) = \sum_{i=1}^{m}\sum_{p=1}^{N_T}\left[e_i^p \cdot w_{il}^{HO}\right]\cdot\left[1-\left(a_l^p\right)^2\right] \tag{3.29c}$$

$$\left(\frac{\partial SSE}{\partial b_{2i}}\right) = \sum_{p=1}^{N_T} e_i^p \tag{3.29d}$$

After substituting Eqs. (3.29a) to (3.29d) into Eqs.(3.27a) to (3.27d), we finally build the updated parameters for the ERNN as:

$$w_{lj}^{IH}(k+1) = w_{lj}^{IH}(k) - \eta \cdot \sum_{i=1}^{m}\sum_{p=1}^{N_T}\left(\left[e_i^p \cdot w_{il}^{HO}\right]\cdot\left[1-\left(a_l^p\right)^2\right]\right)\cdot X_j^p \tag{3.30a}$$

$$w_{il}^{HO}(k+1) = w_{il}^{HO}(k) - \eta \cdot \sum_{p=1}^{N_T} e_i^p \cdot a_l^p \tag{3.30b}$$

$$b_{1l}(k+1) = b_{1l}(k) - \eta \cdot \sum_{i=1}^{m}\sum_{p=1}^{N_T}\left[e_i^p \cdot w_{il}^{HO}\right]\cdot\left[1-\left(a_l^p\right)^2\right] \tag{3.30c}$$

$$b_{2i}(k+1) = b_{2i}(k) - \eta \cdot \sum_{p=1}^{N_T} e_i^p \tag{3.30d}$$

In general, the BPA training needs a large number of training epochs. To accelerate the learning algorithm and to avoid the possible oscillation in the training phase, a very small learning rate *η* as well as the momentum version of BPA are presented. Momentum version of BPA is applied by adding a small adaptive factor of momentum mo so the updated parameters can be faster than the standard BPA. As seen in equations (3.31), the update version of the parameters has a momentum constant which is usually less than one ($mo<1$). In practice, we can propose adaptive momentum and adaptive learning rate as well, if we find that learning process is too slow

$$v(k+1)=v(k)+\Delta v+mo\cdot(v(k)-v(k-1)) \tag{3.31}$$

where $\boldsymbol{v}$ denotes the parameter vector containing the parameters of ERNN, namely $w_{lj}^{IH}, w_{il}^{HO}, b_{1l}, b_{2i}$.

Other issue of accelerating the training algorithm is that the training can proceed in the opposite direction and usually produce oscillation. Like in LMA, the oscillation must be control by using oscillation control routine [16-17] as follows:

- Two sets of adjustable parameters are stored.
- If the following iteration reduces the error, then the next iteration proceeds with the new parameters must be updated and then replacing the old parameters set.

- On the other hand, if the next iteration increases the error beyond the given limit, say as Wildness Factor (WF) of oscillation (WF multiplies with the current error is bigger), then the new set of parameters must be discarded and the next iteration proceeds with the old parameters set.

3.4.4 Results of the optimized Neural Network models

Regarding the performances of the proposed Neural Network models for the ISA solution, the optimized number of context layer L = 21, 22, 25, 27, 30 applied to the five ternary models, gives the results: *a)* 2-ternary: MLP features t_p = 40.1*s*, t_c = 1.3e-3*s*, e_m = 0.588N and e_g = 2.493N; ERNN features t_p = 9.3*s*, t_c = 1.3e-3*s*, e_m = 0.476N and e_g = 2.504N; *b)* 4-ternary: MLP features t_p = 318*s*, t_c = 1.5e-3*s*, e_m = 0.447N and e_g = 1.681N; ERNN features t_p = 75*s*, t_c = 1.5e-3*s*, e_m = 0.422N and e_g = 1.697N. *c)* for 6-ternary: MLP features t_p = 2622*s*, t_c = 1.8e-3*s*, e_m = 0.448N and e_g = 0.963N; ERNN features t_p = 229*s*, t_c = 1.8e-3*s*, e_m = 0.377N and e_g = 0.962N. *d)* for 8-ternary: MLP features t_p = 11358*s*, t_c = 2.6e-3*s*, e_m = 0.464N and e_g = 0.515N; ERNN features t_p = 639*s*, t_c = 2.6e-3*s*, e_m = 0.377N and e_g = 0.389N. *e)* for 10-ternary: MLP features t_p = 3.1e+4*s*, t_c = 3.3e-3*s*, e_m = 0.346N and e_g = 0.379N; ERNN features t_p = 7.8e+3*s*, t_c =3.3e-3*s*, e_m = 0.300N and e_g =

0.335N. Comparison of MLP and ERNN model performances with the other ISA methods are provided also in Chapter 4.

3.5 Hopfield Network model

3.5.1 Introduction

HN is a deterministic local search model which is based on NN and proposed by Hopfield and Tank [28-29]. It has a single-layer fully interconnected recurrent network with symmetric weight parameters without self-connection. The output of each neuron is fed back through a delay unit to the inputs of all neurons. This condition gives the network auto-associative capabilities, which means the network can store number patterns in the weight matrix. By request, the patterns can be recalled to the network until it reaches the stable condition (practically, some or even all number of patterns of the ternary state can be saved in the updated matrix). In the HN method, weight and bias of the network can be updated using training algorithm such as BPA. But for on-line implementation, updating the learning parameter is not suggested because it is very time-consuming. Besides updating the weight matrix via BPA, HN can be also applied on-line without learning phase. The examples of the proposed HN without learning, which is based on the amount of gain amplifier in the network, are High-Gain HN and Low-Gain HN. For example, Waldron et al [2001a, 2001b] suggested the Hopfield Network architecture with High Gain algorithm for finding the binary output of parallel array. Other case, Low-Gain HN is proposed for detecting a peak in a neural A/D converter application, by Dempsey et al [1995].

In this thesis, a probabilistic HN with High-Gain closed loop network is applied together with a deterministic HN with High-Gain method according to the Yang's thesis(2001b).

3.5.2 Hopfield Network Architecture

The proposed model of HN is adapted from a continuous High-Gain Hopfield Network that is offered by Waldron et al (2001a, 2001b). The architecture of such HN can be seen in the Fig 5.

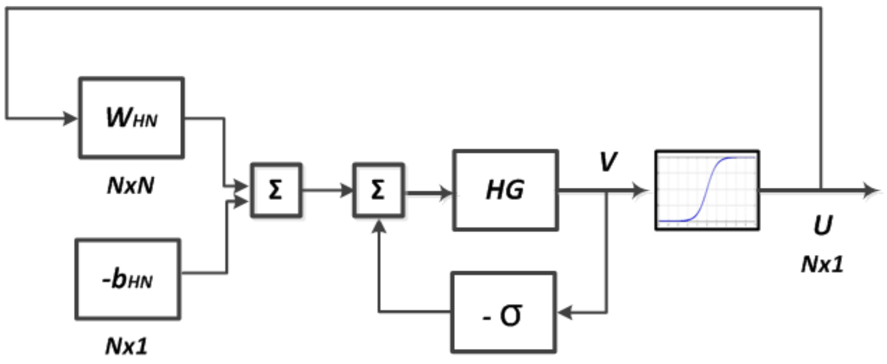


FIGURE 5. ARCHITECTURE OF CLOSE-LOOP HOPFIELD NETWORK WITH HIGH-GAIN

In this scheme, the High-Gain close-loop block diagram is connected to the random weight and bias and ternary activation function. The output U is the results of HN and is actually clamped with the desired output ternary of MPRs. The updated out can be written as:

$$U(t+1) = tansig(V) \tag{3.32}$$

where the term V is the output before activation function (using *tansig*). After close-loop of High-Gain is changed to open-loop using [27], the term V can be written as follows:

$$V = \left(\frac{HG}{1+HG\cdot\sigma}\right)\left(-b_{HN} + \sum w_{HN} \cdot U(t)\right) \tag{3.33}$$

Here HG is High Gain that can be selected experimentally (i.e. HG = 1000); σ is the feedback gain, can be set as 1; and the initial $U(t)$ is starting with a null vector. The weight $w_{HN}$and bias b_{HN} are fixed. They are chosen using two procedures, as follows:

1) Probabilistic weight of HN using random procedure

$$w_{HNp} = 1 - 2 * rand(N, N) \tag{3.34}$$

and $$b_{HNp} = round\big(1 - 2 * rand(N, 1)\big) \tag{3.35}$$

where, N is the number of ternary actuators. By using above procedures, we can find the weight of w_{HN} and the bias b_{HN} values between -1 to +1.

2) Deterministic weight of HN using non-random procedure

The weight and the bias for the deterministic version are found from the performance index equation (see Yang, 2001a for details), and briefly can be seen in (3.35) and (3.36) as follows:

$$w_{HNd} = \begin{bmatrix} 0 & f_1 f_2 & \cdots & f_1 f_N \\ f_2 f_1 & 0 & & f_2 f_N \\ \vdots & \vdots & & \vdots \\ f_N f_1 & f_N f_2 & & 0 \end{bmatrix} \tag{3.36}$$

and $$b_{HNd} = \begin{bmatrix} f_1^2/2 & - & f_1 * M_p \\ \vdots & - & \vdots \\ f_N^2/2 & - & f_N * M_p \end{bmatrix} \tag{3.37}$$

here, f_N is calculated from $M_p(\alpha, u_i) = \sum_{i=1}^{N} f_i u_i$, with *N* is the number of output mechanisms; and M_p is the vector moment with the *p* samples of component.

Both procedures follow mathematic model from (3.32) and (3.33). We can see that the HN as depicted in Fig. 15 is constructed by the High-Gain closed loop amplifier in on-line procedure and it has no learning step. We choose the closed loop gain procedure in order to saturate the summing output of weight and bias into ternary state. Additionally, the results of probabilistic HN are given in the Chapter 4 and shown that the proposed HN is a promising model for ISA solution, because it demonstrates the best generalization error compared to the deterministic HN. Moreover, the main problem of deterministic version is that the output U from (3.32) is easy to trap to the local minima. This is happened because the closed loop equation is too simple and the two gains (HG and T) are constant. To reduce the on-line error, we proposed adaptive gain (HG and T) in the on-line process.

3.5.3 Results of the Hopfield Network model

Regarding the performances of the proposed HN model for the ISA solution, the high gain number (HG) and the feedback gain σ are 1000 and 1 respectively. The closed loop architecture of HN, connected to the random-symmetric weight matrix as well as bias, gives the results: *a)* 2-ternary: t_p = 0.2*s*, t_c = 1.1e-3*s*, e_m = 0 and e_g = 0.833N; *b)* 4-ternary: t_p = 0.3*s*, t_c = 5.1e-3*s*, e_m = 0N and e_g = 0.585N; *c)* for 6-ternary: t_p = 1.4*s*, t_c = 0.022*s*, e_m = 0N and e_g = 0.484N; *d)* for 8-ternary: t_p = 24*s*, t_c = 0.121*s*, e_m = 0N and e_g = 0.329N; *e)* for 10-ternary: t_p = 892*s*, t_c = 0.202*s*, e_m = 0N and e_g = 0.301N.

The results of deterministic HN are shown in Table 6. These performances are established by using high number of iteration (maximum 5000 iterations) and combined with adaptive gain. The results show that the procedure is not optimal and need to be improved.

TABLE 6. COMPARISON OF THE M-TERNARY MPRS OF DETERMINISTIC HN

Method/Description	**2-ternary**	**4-ternary**	**6-ternary**	**8-ternary**	**10-ternary**
on-line calculation t_c (s)	0,0034	0,007	0,015	0,218	0.570
generalization error e_g (N)	1,4	1,17	1,01	0,854	0,766
Full-scale generalization error FGE (%)	28,3	23,5	20,3	17,1	15,3

Because of the limitation performance of deterministic HN, we compare the performances of probabilistic HN with other ISA methods.

Chapter 4

Comparison of the Six Inverse Static Analysis Models

4.1 Introduction

This chapter outlines the training and testing results from six ISA models in the tables and figures. Performance of the *m*-ternary MPRs will be compared to the proposed ISA models with the following descriptions:

t_p	: time for preparing the model, including learning procedure (s)
t_c	: time for computing online (s)
e_m	: error modeling (N)
e_g	: generalization error(N)
STD	: generalization standard deviation (N)
FGE	: full scale generalization error (%)

Additional description of STD and FGE are explained in the following equations:

$$STD = \sqrt{\frac{\sum_{r=1}^{N_{Test}} \left(e_r - e_{avg}\right)^2}{N_{Test}}}\ , \tag{4.1}$$

$$FGE = \frac{e_g}{max(GM)} \times 100\%\ , \tag{4.2}$$

where $e_g = \sqrt{\frac{\sum_{r=1}^{N_{Test}} e_r^2}{N_{Test}}}$ (4.3)

Here N_{Test} is the number of data test equal to 1701 rows of the input couple of continuum real numbers *X (α,M)* and those data are the same for all *m*-ternary models. Other term *GM* is a generated moment with the maximum value between ±5Nm as depicted in Fig. 2. The results of each *m*-ternary model can be seen in Sections 4.2 to 4.5.

4.2 Results of 2-ternary MPRs

The comparison results of the 2-ternary models can be considered in Table 7 as well as in the Figs. 6 and 7. The results show that HN is the best method concerning online computing time t_c and the generalization error e_g. As additional facts, LUT has the best preparation time t_p and the five methods have FGE more than 10%;

TABLE 7. PERFORMANCE COMPARISON OF THE 2-TERNARY MPRS OF THE CONSIDERED METHODS

Method/Description	LUT	NFTS	NFLUT	MLP	ERNN	HN
t_p (s)	0.1	2.2	2.3	40.1	9.3	0.2
t_c (s)	0.008	8e-5	5e-4	1.3e-3	1.3e-3	1.1e-3
e_m (N)	0	0.871	0	0.588	0.476	0
e_g (N)	2.594	1.051	1.283	2.493	2.504	0.833
STD of e_g (N)	1.528	0.579	0.903	1.495	1.483	0.480
FGE (%)	51.8	21	25.7	49.9	50	16.7

Note: the CPU has 32 bit operating system, dual core, 2.6 GHz, RAM 4 GB.

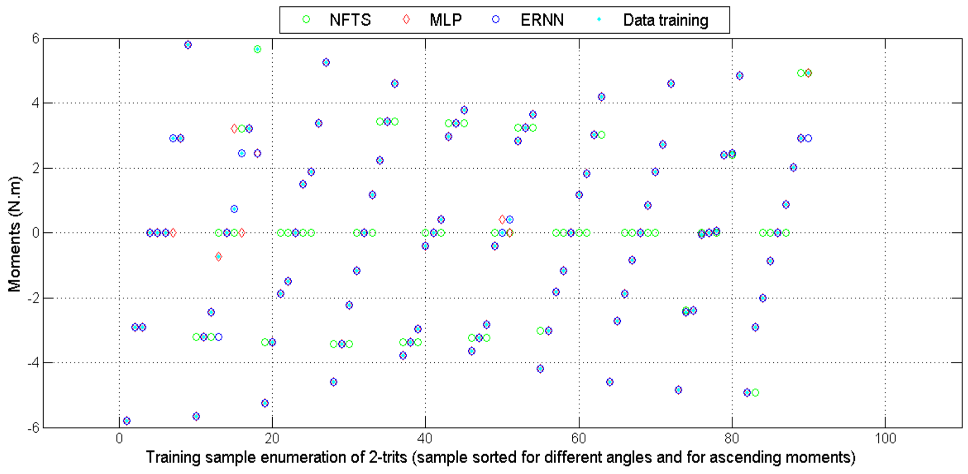


FIGURE 6. TRAINING PERFORMANCE OF 2-TERNARY MPRS WITH DIFFERENT ISA METHODS

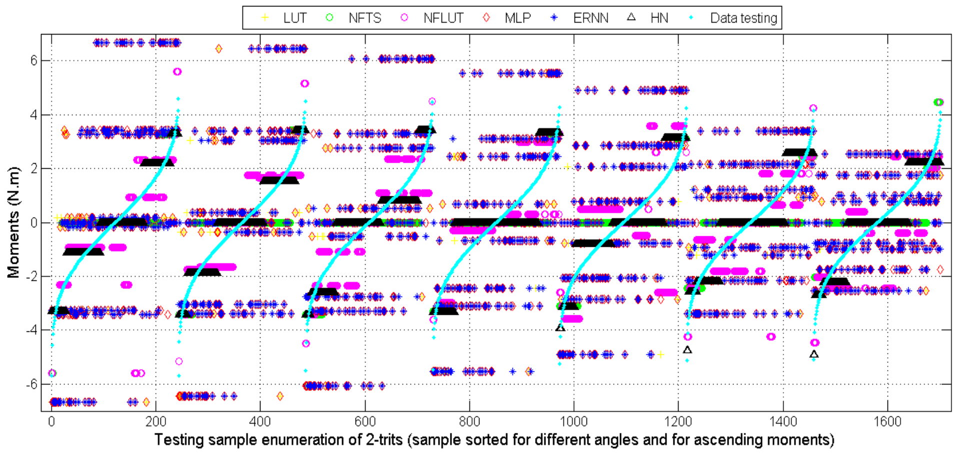


FIGURE 7. TESTING PERFORMANCE OF 2-TERNARY MPRS WITH DIFFERENT ISA METHODS

4.3 Results of 4-ternary MPRs

The comparison results between the 4-ternary models can be considered in the Table 8 as well as in the Figs. 8 and 9. The results show that HN is the best method concerning the generalization error $e_{g,}$ NFTS shows the best online computing time t_c, and LUT has the best preparation time t_p. The six proposed methods have FGE more than about 10%.

TABLE 8. PERFORMANCE COMPARISON OF THE 4-TERNARY MPRS OF THE CONSIDERED METHODS

Method/Description	LUT	NFTS	NFLUT	MLP	ERNN	HN
t_p (s)	0.3	43	43.5	318	75	0.3
t_c (s)	0.009	1e-4	3e-3	1.6e-3	1.6e-3	5.1e-3
e_m (N)	0	0.909	0	0.447	0.422	0
e_g (N)	1.664	0.909	0.611	1.681	1.697	0.589
STD of e_g (N)	0.985	0.513	0.368	0.991	1.000	0.368
FGE (%)	33.3	18.2	12.2	33.6	33.9	11.8

Note: the CPU has 32 bit operating system, dual core, 2.6 GHz, RAM 4 GB.

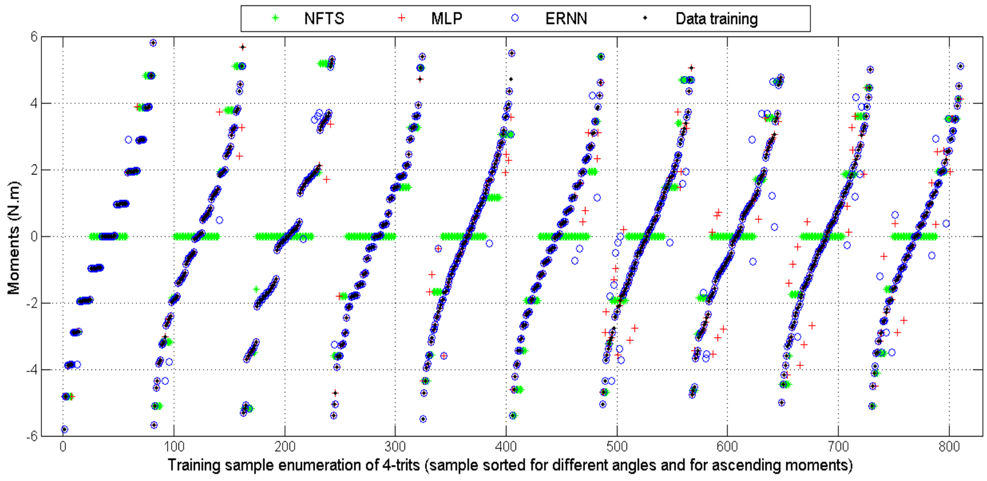


FIGURE 8. TRAINING PERFORMANCE OF 4-TERNARY MPRS WITH DIFFERENT ISA METHODS

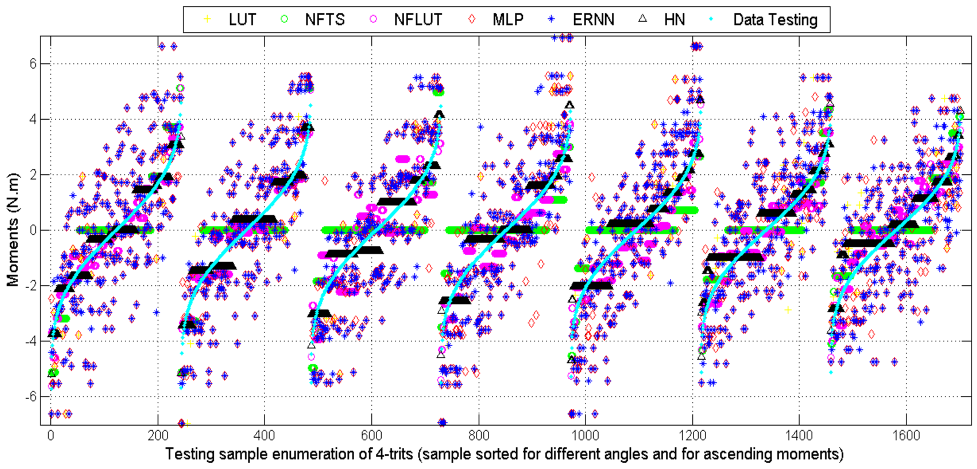


FIGURE 9. TESTING PERFORMANCE OF 4-TERNARY MPRS WITH DIFFERENT ISA METHODS

4.4 Results of 6-ternary MPRs

The results of the 6-ternary models can be seen in the Table 9 as well as in the Figs. 10 and 11. The results demonstrate that HN and NFTS are the best method concerning the best preparation time t_p and online computing time t_c respectively, and the HN has the best generalization error e_g. Concerning minimum requirements i.e. real-time computing and FGE, the NFLUT is the only method recommended for ISA solution.

TABLE 9. PERFORMANCE COMPARISON OF THE 6-TERNARY MPRS OF THE CONSIDERED METHODS

Method/Description	LUT	NFTS	NFLUT	MLP	ERNN	HN
t_p (s)	3.2	185	186	2622	229	1.4
t_c (s)	0.013	2.1e-4	5e-3	1.8e-3	1.8e-3	0.022
e_m (N)	0	0.917	0	0.448	0.377	0
e_g (N)	0.891	0.932	0.498	0.963	0.962	0.484
STD of e_g (N)	0.533	0.524	0.284	0.569	0.587	0.321
FGE (%)	17.8	18.6	9.9	19.3	19.2	9.7

Note: the CPU has 32 bit operating system, dual core, 2.6 GHz, RAM 4 GB.

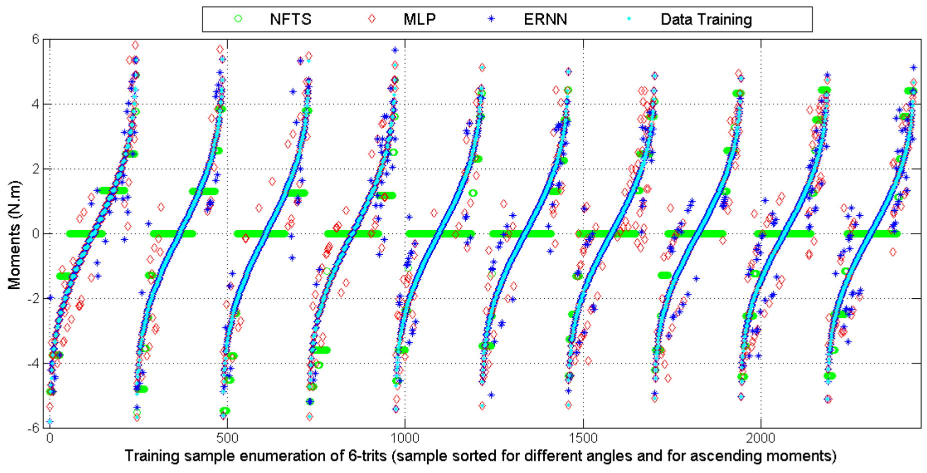


FIGURE 10. TRAINING PERFORMANCE OF 6-TERNARY MPRS WITH DIFFERENT ISA METHODS

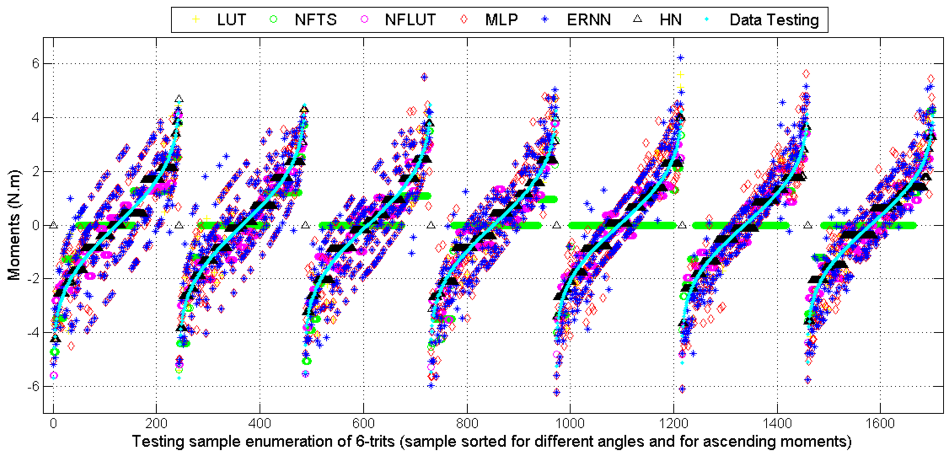


FIGURE 11. TESTING PERFORMANCE OF 6-TERNARY MPRS WITH DIFFERENT ISA METHODS

4.5 Results of 8-ternary MPRs

The performance of the 8-ternary models is shown in the Table 10 and in the Figs. 12 and 13. The results demonstrate that HN and NFTS are still the best method concerning the best preparation time t_p and online computing time t_c respectively, and the ERNN has the best generalization error e_g. Concerning minimum requirements i.e. real-time computing and FGE, the ERNN and MLP are the methods that recommended for ISA solution.

TABLE 10. PERFORMANCE COMPARISON OF THE 8-TERNARY MPRS OF THE CONSIDERED METHODS

Method/Description	LUT	NFTS	NFLUT	MLP	ERNN	HN
t_p (s)	48	965	983	11358	639	24
t_c (s)	0.35	1.4e-3	1.4e-2	2.5e-3	2.5e-3	0.121
e_m (N)	0	0.658	0	0.464	0.377	0
e_g (N)	0.805	0.985	0.528	0.515	0.389	0.329
STD of e_g (N)	0.499	0.585	0.328	0.431	0.347	0.262
FGE (%)	16.1	19.7	10.5	10.3	7.8	6.6

Note: the CPU has 32 bit operating system, dual core, 2.6 GHz, RAM 4 GB.

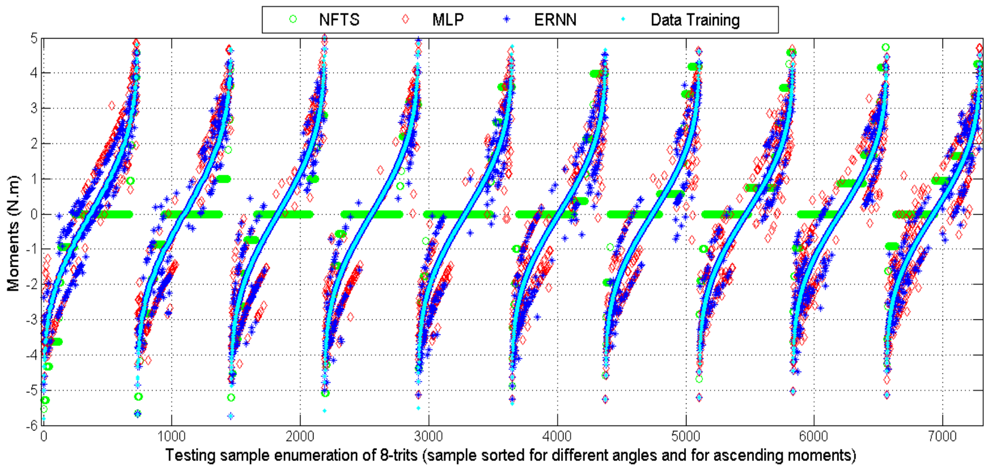


FIGURE 12. TRAINING PERFORMANCE OF 8-TERNARY MPRS WITH DIFFERENT ISA METHODS

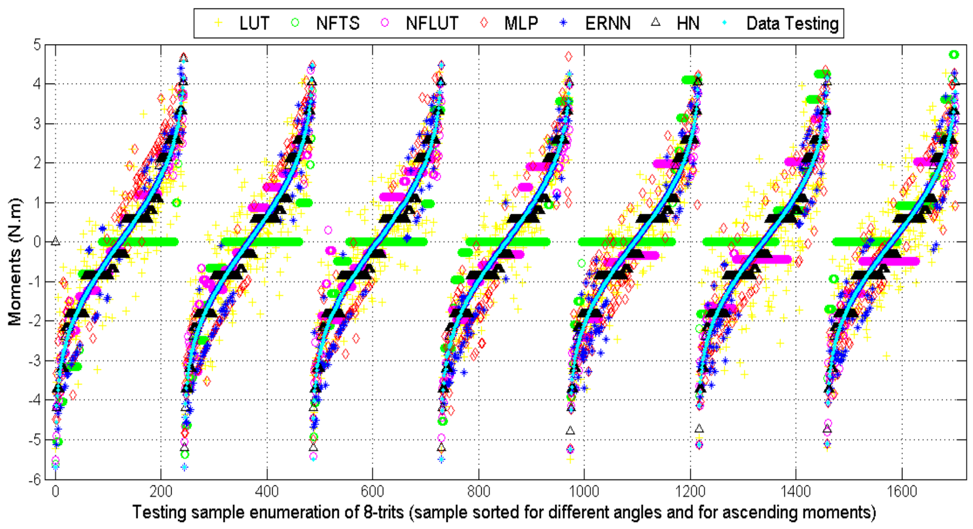


FIGURE 13. TESTING PERFORMANCE OF 8-TERNARY MPRS WITH DIFFERENT ISA METHODS

4.6 Results of 10-ternary MPRs

The results of using AI methods to the 10-ternary MPRs mechanism can be seen in the Table 11 as well as in the Figs. 14 and 15. The results demonstrate that HN and NFTS are still the best method concerning the best preparation time t_p and online computing time t_c respectively, and the HN has the best generalization error e_g. Again, concerning minimum requirements of ISA solution i.e. real-time computing and generalization error, the ERNN and MLP are the recommended methods.

TABLE 11. PERFORMANCE COMPARISON OF THE 10-TERNARY MPRS OF THE CONSIDERED METHODS

Method/Description	LUT	NFTS	NFLUT	MLP	ERNN	HN
t_p (s)	2.1e3	7.1e3	7.3e3	3.1e4	7.8e3	892
t_c (s)	2.8	1.9e-3	3.8e-2	3.3e-3	3.3e-3	0.203
e_m (N)	0	0.622	0	0.346	0.300	0
e_g (N)	0.551	0.998	0.3966	0.379	0.335	0.301
STD of e_g (N)	0.361	0.567	0.241	0.337	0.303	0.158
FGE (%)	11.0	19.9	7.9	7.6	6.7	6.01

Note: the CPU has 32 bit operating system, dual core, 2.6 GHz, RAM 4 GB.

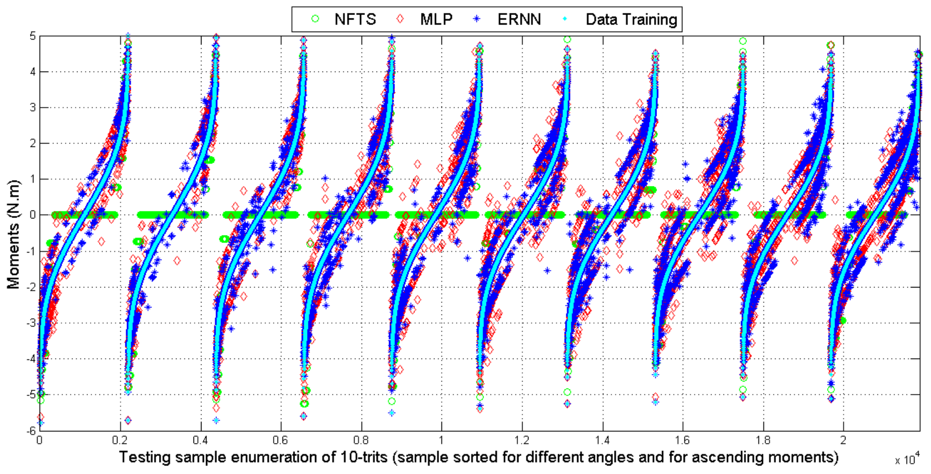


FIGURE 14. TRAINING PERFORMANCE OF 10-TERNARY MPRS WITH DIFFERENT ISA METHODS

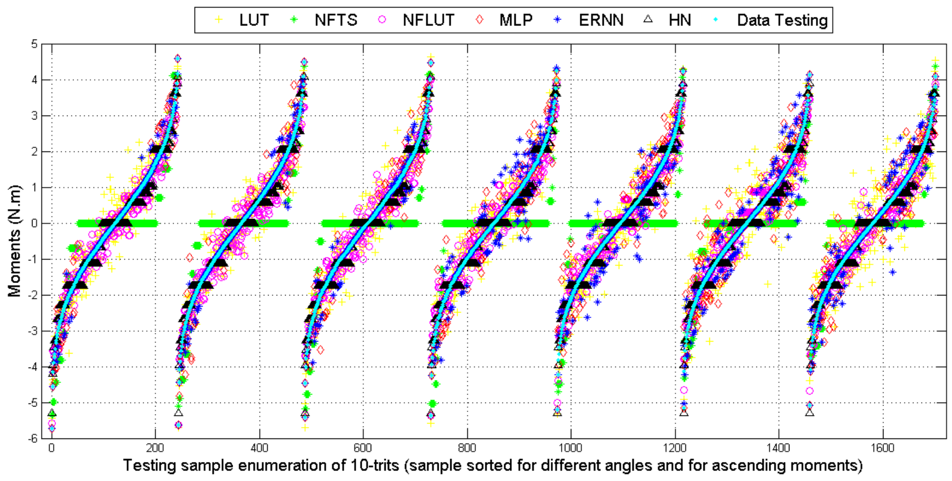


FIGURE 15. TESTING PERFORMANCE OF 10-TERNARY MPRS WITH DIFFERENT ISA METHODS

4.7 Summaries of six models of *m*-ternary MPRs

Summaries of the proposed methods in this section are based on the requirement of ISA solutions which are real-time control (i.e. less than 5ms) and generalization error (around 10% of FGE).

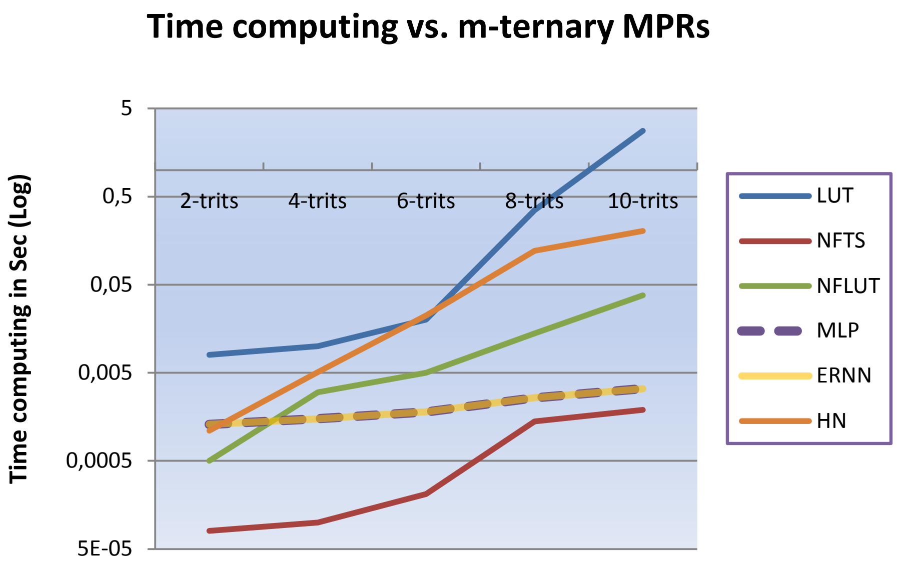


FIGURE 16. ON-LINE TIME COMPUTING TC OF M-TERNARY MPRS

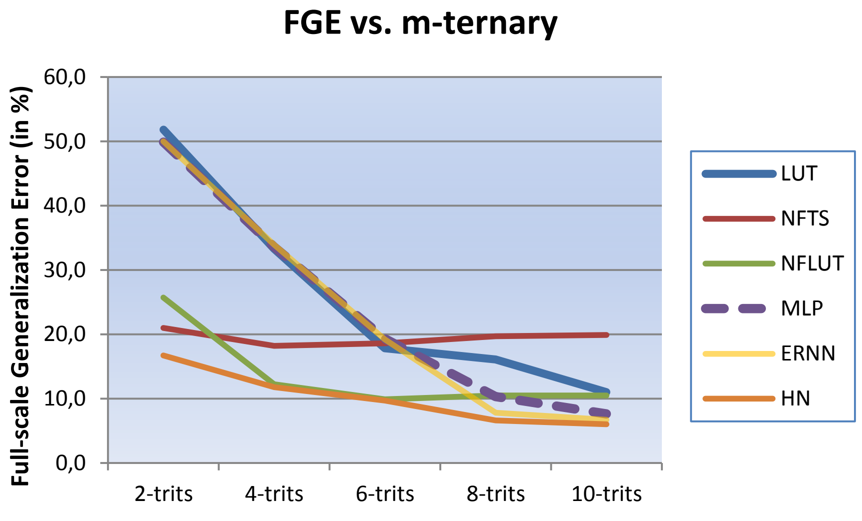


FIGURE 17. FGE OF M-TERNARY MPRS

According to the facts from the sections 4.2 to 4.5 as well as from additional Figs. 16 and 17 (whereas the figures explain a relation between time computing t_c vs. *m*-ternary MPRs and the generalization error FGE vs. *m*-ternary MPRs), the summaries of using six ISA solutions in *m*-ternary MPRs can be concluded as follows:

1. For 2-ternary MPRs: HN, NFTS and NFLUT methods show better performance concerning the real-time computation and generalization error, compared to the MLP, ERNN as well as LUT methods. In addition, HN demonstrates the lowest generalization error (FGE = 16,7%) and NFTS shows the lowest on-line computing respectively (t_c = 8e-5s).
2. For 4-ternary MPRs, HN shows the lowest generalization error. NFLUT and NFTS perform the second and third position. The MLP, ERNN and LUT methods have less accuracy in this generalization phase. In addition, only LUT does not fulfill the real time requirement. Other methods (NFTS, MLP, ERNN, NFLUT and HN) show their ability to control MPRs in real time.
3. For 6-ternary MPRs, only NFLUT meets the requirement as ISA solution, such as on-line computing and generalization phase. In addition, HN fulfills accuracy of generalization phase but in contrast other methods i.e. NFTS, ERNN and MLP, fulfill on-line computing requirement.

4. For 8-ternary MPRs, ERNN is the most accurate method for ISA solution, features the best generalization ability and requires a rather small computational time during the on-line phase. HN has better accuracy than ERNN, but require a larger off-line and on-line computational time respectively. NFTS has better on-line computational phase but less generalization phase compare to ERNN and HN.
5. For 10-ternary MPRs, ERNN and MLP are the suitable solution for ISA problem. They offer not only the accuracy in the generalization phase, but also the real-time approximation as shown in their on-line computing time t_c. The HN still shows the highest accuracy concerning the generalization error compared to others, but requires a larger on-line computing time.
6. In all *m*-ternary MPRs, we can also describe the facts that: a) NFTS features the shortest on-line computational time. However it is more inaccurate compared to NFLUT, MLP, ERNN and HN in generalizing phase; and b) HN features the lowest generalizing phase but it has bigger on-line computing time.

Chapter 5
Comparison of Proposed AI Methods with Other Methods

In general, most of the artificial intelligence methods with learning capability can be applied for general approximator of non-linear control, such as ISA problem. To be practically applied to the ISA of MPRs mechanisms, the proposed AI methods must fulfill some requirements of ISA, such as real-time control procedure and small full-scale generalization error (this is already explained in Section 4.1).

Additionally, most of the AI methods can provide the solution of the ISA of ternary MPRs. From literature, some of them are: Boltzmann Machine (BM), Radial Basis Function Network (RBFN), Jordan Recurrent Neural Network (JRNN), Wavelet Network (WN) and Support Vector Machines (SVMs). The solutions mentioned above are based on learning capability, stochastic model, signal analysis as well as statistic approach. Brief summaries of those methods are reported in the following sections.

5.1 Boltzmann Machine (BM)

BM method is named in honor of a 19th century Austrian physicist Ludwig Boltzmann by its inventors, Geoffrey Hinton and his co-workers. BM is a stochastic machine which consists of stochastic neurons. As shown in the Fig. 18, the BM has two groups of neurons, i.e.: hidden neurons and visible neurons. Both groups are stochastic with probabilistic firing algorithm. In this scheme, the output of BM is actually the visible units that should be updated in the closed loop network, as shown in the Fig. 18.

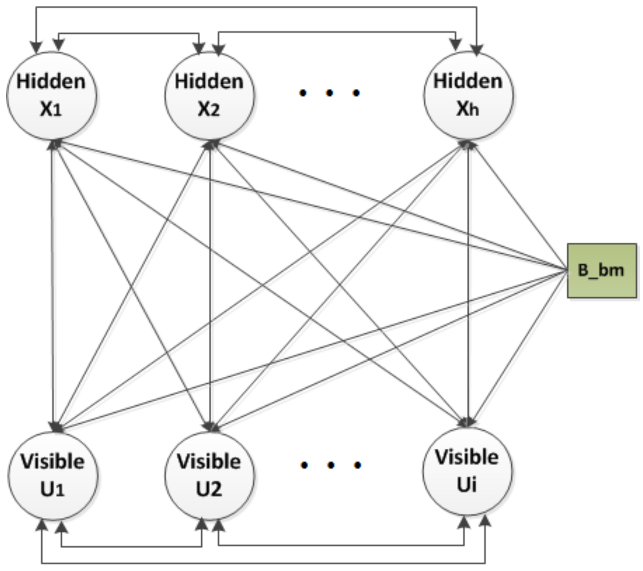


FIGURE 18. ARCHITECTURE OF BOLTZMANN MACHINE, WITH H NUMBER OF HIDDEN NEURONS AND I NUMBER OF VISIBLE NEURONS

Normally BM has two possible states (-1 and +1 or 0 and 1 for off and on condition), but it is possible to adapt BM into ternary mode by modifying its firing algorithm. Moreover, another feature of BM is the use of symmetric weight connection between its neurons. This form is motivated by statistical physics considerations.

The visible part of BM provides a direct interface between the network and the environment of application (here equivalent with the ternary states of proposed MPRs). On the other hand, the hidden neurons always operate freely. They are used to figure out a number of constraints in the network. Moreover, the hidden part in this network completed the task by capturing statistical correlation in the visible neurons. These neurons are clamped or connected directly to the output patterns. The network represent here is the unsupervised learning procedure that performs pattern classification of the states of the MPRs.

In brief, the goal of BM is to produce a Neural Network that categorizes input patterns according to a Boltzmann distribution [31], where the assumptions are made as follows:

- Each environmental vector persists long enough for network to reach thermal equilibrium. The thermal equilibrium is the condition where the desired visual states are achieved, by cooling/reducing the parameter, called as T parameter, slowly until the target is found.

- There is no structure in the sequence in which environmental vectors are clamped or connected to the visible neurons of the network.

Because of the simplicity of the network, BM can be developed over the deterministic Hopfield model that is explained in Section 3.5, as one of the ISA solutions. As on-line algorithm such as HN, BM replaces the activation function of HN, which is a deterministic activation function, with a probabilistic activation function, that can be written as:

$$P_{HN} = 1/\left(1 + exp(-\Delta E/T)\right) \qquad (5.1)$$

This is the stochastic function for binary case. For ternary case, Eq. 5.2 can be modified as:

$$P^{*}{}_{HN} = 2/\left(1 + exp(-\Delta E/T)\right) - 1 \qquad (5.2)$$

where T is the parameter, acts like temperature; and ΔE is the energy gap resulted from the flip of the state in the cycle process [30].

To explain (5.2), we calculate the output function with different temperature *T*. The results can be seen in Fig. 19 below.

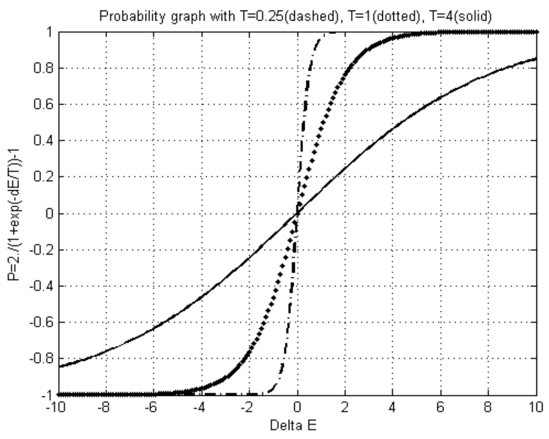


FIGURE 19. GRAPH OF DIFFERENT TEMPERATURE T IN TERNARY PROBABILISTIC ACTIVATION FUNCTION

In the learning process of BM, *T* starts with high value and decreases to a very small value, the probability of each output node is firmly changed from +1, 0 to -1 in ternary case. In other words, the network will move to the next state without being able to jump back. This process is called also as simulated annealing.

Recently, many scientists compare the works of HN and BM in the same applications. The reason is because the potentialities of both methods use the equivalent network and give the comparable results (concerning full-scale generalization error *FGE* and time computing t_c). In brief, similarities and differences between BM and HN can be described as follows:

Similarities:

- Connections between neurons (weights) are symmetric
- Neuron states are bipolar
- Weights and biases are selected at random for asynchronous update
- There is no self-feedback

Differences:

- Architecture of BM permits the use of hidden layer
- BM uses probabilistic activation function while HN with a deterministic activation function

5.2 Radial Basis Function Network (RBFN)

RBFN, as well as MLP, is an example of nonlinear layered feedforward neural networks which is a universal approximator too. The name radial basis function is coming from an approach to approximate the function based on adaptive function interpolation (Broohead and Lowe 1988). Similarly, Moody and Darken proposed a fast learning neural network structure with locally tuned processing units.

RBFN was enthusiastically welcome by NNs society, because it demonstrated the improved capability of solving classification problems, which is not done by a global function (like sigmoid function in MLP). For this reason, some locally classified basis function such as Gaussian functions, wavelets or B-spline functions are commonly proposed. Here, Moody [32] proposed one activation function like shown in (5.3),

$$f_h = exp(-\|x_i - c_h\|^2 / \sigma_h^2) \tag{5.3}$$

which is similar to the Gaussian density function centered at c_h and spread around centre σ_h.

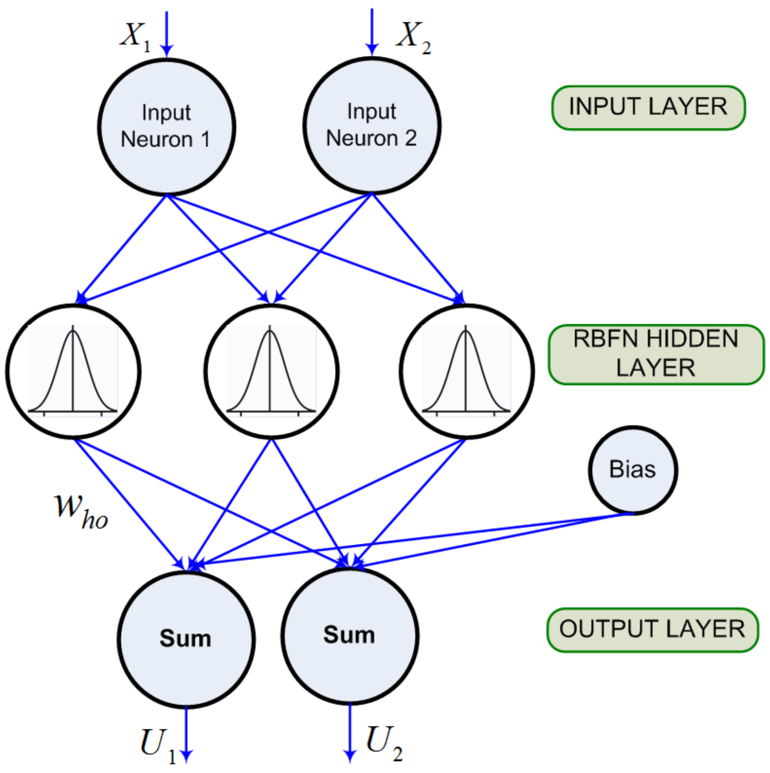


FIGURE 20. RBFN ARCHITECTURE

Furthermore, the RBFN is a feedforward network with three layers and basically it is similar to the MLP with one hidden layer. The common configuration of the RBFN can be seen in Fig. 20. The input layer of the RBFN is directly connected with the hidden layer with gain one. So the weighting parameters (including biases) are only placed to the connection between hidden and output layers. In contrary to MLP, the output layer of RBFN is a summation of all output functions, which is directly associated to the output desired. This makes the learning parameter of RBFN different from the training algorithm on the backpropagation networks (MLP). The main important issue here is how to select each neuron in the hidden layer (in a Gaussian function: parameters center c_h and spread around centre σ_h). To select those parameters, we can propose clustering algorithm, such as k-means clustering, which is capable to determine the optimal position of center and spread parameter.

Moreover, the output network can be found easily by (5.4a-5.4b) as follows:

$$U_i = b + \sum w_{ho}\, exp\left(- \|x_j - c_h\|^2 / \sigma_h^2\right) \tag{5.4a}$$

$$U_{i\_3state} = round\left(b + \sum w_{ho}\, exp\left(- \|x_j - c_h\|^2 / \sigma_h^2\right)\right) \tag{5.4b}$$

where x_j is the vector input of the network; w_{ho} and b are the weighting and bias parameters between hidden and output layers. Here, only w_{ho} and b are tuned in training process. The output U_{i_3state} of (5.4b) is exactly the 3-state outputs of the desired outputs of MPRs.

Moreover, Haykin [31] explained some differences between RBFN and MLP such as:

- RBFN basically has a single hidden layer, while MLP can have more than one hidden layers.
- In MLP, hidden and output neurons have the same underlying function. In RBFN, they are specialized into distinct functions.
- In RBFN, the output layer is linear, but in MLP, all neurons are nonlinear.

- The hidden neurons in RBFN calculate the Euclidean norm of the input vector and the center, while in MLP the inner product of the input vector and the weight vector are calculated.
- The MLP constructs global approximations to nonlinear input–output mappings, while the RBFN constructs a local approximation to nonlinear input–output mappings using localized nonlinear function (e.g. Gaussians).

Concerning the structure of RBFN, the training procedure includes two phases below:

- Phase one: Gaussian parameters must be initialized using such as unsupervised clustering algorithm, linear vector quantization (LVQ), etc.
- Phase two: Adaptive training algorithm for updating the weighting parameters (BPA, LMA).

By the two-phase explanation above, the RBFN increases the number of offline computing time in the learning phase. So this network needs at least two steps optimization processes, which are finding the optimal Gaussian parameters in the hidden layers and updating the weight parameters in the output layers. Once the optimal parameters are found, the RBFN performance could be determined easily. In general, for finding better minimum error or better accuracy in the similar application, using MLP is more suggested than RBFN [25, 31].

5.3 Jordan Recurrent Neural Network (JRNN)

In brief, the JRNN is a NN with short-term memory features that takes the output network as feedback to the state units. Each output is connected to one state unit with the constant weight. When the previous output of a network is crucial in determining the next learning, as in the design of a robot trajectory, a Jordan network [16], which has similar architecture with Elman network as explained in Section 3.4.2, seems more appropriate than Elman's. But, if the past internal neural responses are more important as in dynamic control problems, then an Elman network may be preferred [15].

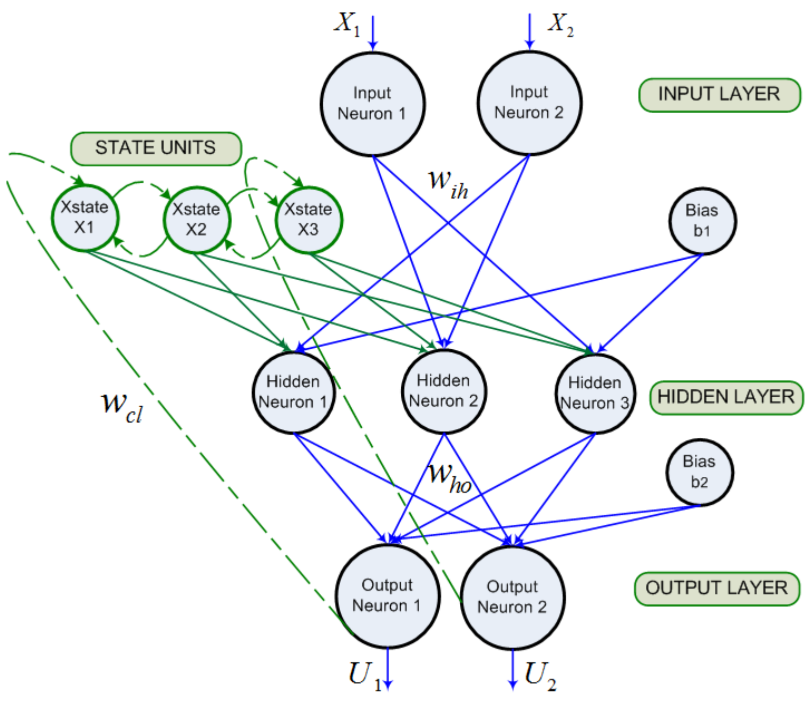


FIGURE 21. JORDAN NETWORK (NOT ALL CONNECTIONS ARE SHOWN)

If we considered JRNN that shown in Fig. 21 as one of the ISA solution, we must derive output network $U(t)$ by modifying MLP equation (3.3) as follows:

$$U(t) = round(G(\sum w_{ho}\ G(\sum w_{ih}^T X_i(t) + w_{cl}U(t-1) + b_1) + b_2)) \tag{5.9}$$

where $X_i\ (t)$ and $U(t)$ are input and output of the network; b_1 and b_2 are the biases input in the hidden and the output layers; w_{ho}, w_{ih} and w_{cl} are the weighting related to the hidden-output, input-hidden and the constant weighting of the state units respectively; additional $U(t-1)$ is the short-term output, where the typical initial $U(0) = null\ vector$.

The activation function of hidden neuron, as well as the output neuron $U(n)$, uses the sigmoid function *tanh*. In addition, we can choose the constant parameter w_{cl} as follows:

- $w_{cl} = 1$, the network become a fully connected Jordan NNs
- $w_{cl} = \frac{1}{No\_of\_Hidden}$, the network become a normalized context-fully connected Jordan NNs
- $w_{cl} = 0$, the network become a MLP

Concerning the comparison between ERNN and JRNN, so far we do not know which feedbacks (Elman or Jordan feedback) have better influence to the ISA solution. This makes JRNN a potential competitor to ERNN.

5.4 Wavelet Neural Network (WNN)

Wavelets are a class of function used to localize a given function or continuous signal in both position and scaling. The major advantage afforded by wavelets is the ability to perform local analysis — that is, to analyze a localized area of a larger signal. Compared to other signal analysis (i.e. Fourier, etc.), the wavelets have strong capability to reveal the aspects of data such as trends, breakdown points, and self-similarities. More detail about wavelets analysis and the comparison with Fourier analysis and other mathematical analysis can be seen in Wavelet Toolbox description [40].

In general, wavelets form the basis of the wavelet transform (WT) (Daubechies I. [34]), whereas a WT is a representation of basis function called wavelets where they dilated (or scaled by factor of *a*) and translated (or shifted by factor of *b*) into a finite-length of prototype wavelet (known as mother wavelet Ψ), that can be written as

$$\Psi_k(x) = \frac{1}{\sqrt{a_k}} \psi\left(\frac{x_j - b_k}{a_k}\right), a_k \neq 0 \qquad (5.10)$$

here $\Psi_k(x)$ represents the family of wavelets obtained from the single $\psi(x)$ and a_k, b_k are the scale and translate values in the time or frequency domain; x_j is the input wavelet network $j = 1, 2$ and parameter wavelet $k = 1, 2, .., K$; respectively, with K hidden unit of the wavelet layer.

In general, WT are classified into continuous WT (CWT) and discrete WT (DWT). CWT is designed to work with functions defined over the whole real axis, while DWT deals with functions that are defined over a range of integers (example: t =0, 1,…, *N*-2, *N*−1, where *N* denotes the number of values in the time series).

Concerning ISA solution, WT replaces the activation function in the neural network. This activation function has similar purpose with the sigmoid function in MLP or Gaussian function in RBFN. Furthermore, a number of publications reported on the analysis and applications using Wavelet Neural Network. Zhang Q. et al (1992) introduced Wavelet Network as universal approximator; Rao S.S. et al (1994) proposed feedback in Zhang's work, and known as Recurrent Wavelet Neural Network (RWNN); and Abiyev R.H. (2008, 2011) combined Fuzzy and Wavelet Neural Network (FWNN) for the identification, control and time series prediction problems.

The structure of WNN, as universal approximator, explained in this thesis is depicted in Fig. 22. This is a feed-forward neural network with one hidden layer and whose output layer consists of several linear combiners. The hidden layer consists of wavelets, whose activation functions are constructed from a wavelet basis.

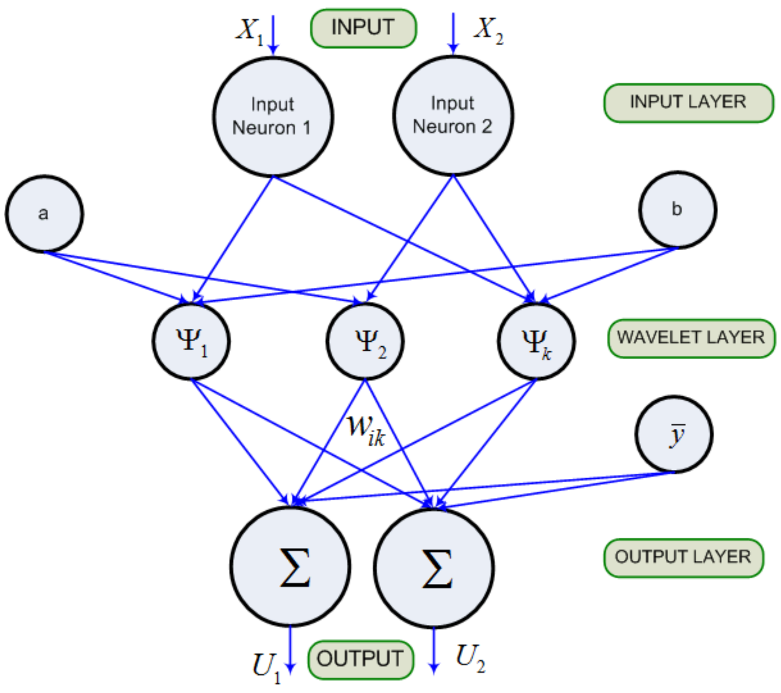


FIGURE 22. WAVELET NEURAL NETWORK (NOT ALL CONNECTIONS ARE SHOWN)

From Fig. 22, the ternary state-output of WNN U_i can be defined as

$$U_i = round(\overline{y_k} + \sum_{i=1}^{m} w_{ik}\, \Psi_k(x)), \text{ for } k= 1, 2, ..., K \qquad (5.11)$$

where i = 1, 2, ..., m output ternary; w_{ik} is the weight coefficients between wavelet layer and output layer; and the additional value $\overline{y_k}$ is needed to deal with the functions whose mean are non-zero. All parameters in Eqs. (5.10) and (5.11) (a_k, b_k, w_{ik} and $\overline{y_k}$) can be chosen properly in the beginning of iteration by some techniques, for instance orthogonal least square procedure [35] and clustering method [39]. For updating the parameters, some learning procedures can be used, such as stochastic gradient algorithm and genetic algorithm [35, 38].

5.5 Support Vector Machines (SVMs)

Originally, SVMs were designed to solve problems in pattern recognition, as binary classifier, by determining a hyperplane that separates two states or groups (positive and negative groups) and optimizing the separated margin between them. This can be done clearly by using the theory of statistical learning and the method of structural risk minimization (Vapnik, 1971, 1995). The concept of SVM states that the mapping of an input space X into a properly high-dimensional space (called as feature space), using a non-linear mapping function $\rho(X)$ could be more linearly separable than in the low dimensional input space. The different between input space and feature space can be described in Fig. 23.

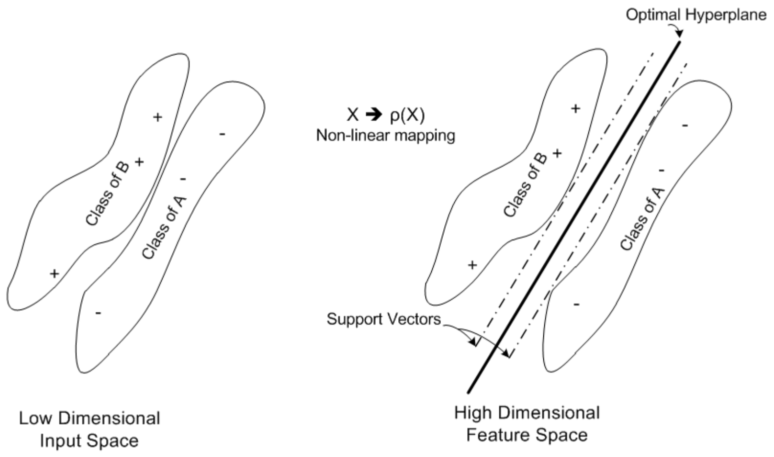


FIGURE 23. MAPPING FROM INPUT SPACE INTO FEATURE SPACE

A brief description on how to change an input space to the feature space of the SVM can be shown on Fig 23. The left figure is a low dimensional data (input space) while the right one is the high-dimensional data (feature space). A training set $D = \{X_i, U_i\}$ is given, where the input vectors $X_i = \left({X_i}^{(1)}, {X_i}^{(2)}, \dots, {X_i}^{(n)}\right)$ and the output vectors $U_i \in \{-1, +1\}$; with $i = 1, \dots, n$-dimensional vector.

Moreover, optimal hyperplane separates the equal distance between the data point which are nearest to the boundary of the two classes. Such data points must satisfy the classified equations according to Vapnik's formulation:

$$w^T X_i + b < 0 \, for \; U_i = -1 \qquad (5.12a)$$

$$w^T X_i + b > 0 \, for \; U_i = +1 \qquad (5.12b)$$

Nonlinear mapping from input space to feature space is carried out using the kernel function family as follows

$$K(X) = \{K_1(X), K_2(X), \dots, K_n(X)\}, \qquad (5.13)$$

where the linear discriminant function can be defined as

$$\sum_{i=1}^{n} w_i \, K_i(X) + b = 0 \qquad (5.14)$$

The structure of SVM, as universal approximator is depicted in Fig. 24. Similar to three layer network, the SVM is a feed-forward network with one hidden layer with kernel activation function. The output layers consist of several linear combiners Σ with updated weight w_{ij}.

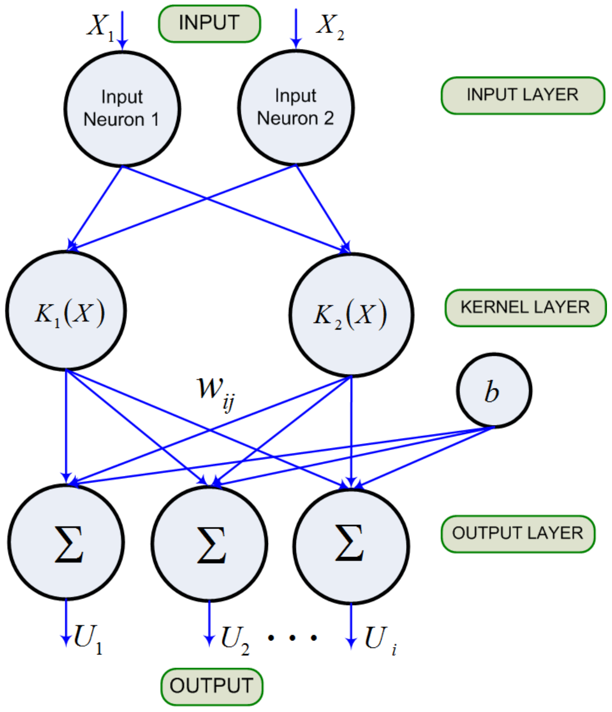


FIGURE 24. BASIC ARCHITECTURE OF SVM

The outputs of SVM can be written as:

$$U_i = round\left(b_i + \sum_{i=1}^{m} \sum_{j=1}^{2} w_{ij}\, \mathrm{K}_j(X)\right), \tag{5.15}$$

for j= 1, 2; i=1, 2, ..., m ternary output and input $X = [X_1, X_2\,]$.

Moreover, in engineering SVM has been used in several applications as universal approximator. For example: Cao et al (2001) proposed SVMs in financial time series forecasting of S&P daily index; Mukherjee et al (1997) investigated SVM and other predicted method for chaotic time series prediction. Both Cao and Mukherjee used several methods as competitors of SVMs and they experimentally proved that SVMs perform better generalization capabilities than other compared methods.

Chapter 6

Conclusion and Future Works

As conclusion, this thesis presented: 1) five planar massively parallel robots (MPRs) with 2, 4, 6, 8 and 10 three-state force actuators and one continuous degree of rotational motion; 2) one brute-force method, two Neuro-Fuzzy methods, two Neural Network methods and one Hopfield Network method for the solution of inverse static analysis of the considered MPRs. Thanks to the partitioned and spatially distributed actuator architecture, the considered discrete robot (specially for 6, 8 and 10-ternary MPRs) features rather sufficient, identical and accurate torque generation capabilities, compared to the standard CSL mechanism (actuated by a single continuously regulated force generator).

Comparison among the considered inverse static analysis methods highlighted that: 1) Elman type Recurrent Neural Networks and Neuro-Fuzzy Takagi-Sugeno methods are both suitable for real-time control applications, with the former providing more accurate solutions and generalization capabilities while the later involving less on-line computational time; 2) Hopfield Network method is the most powerful method for 2, 4, 6, 8 and 10-ternary MPRs, offers the lowest generalization error but still has a significant on-line computing time. 3) Both ERNN and MLP methods can approximate the output of 8 and 10-ternary with good accuracy and in real-time computing. 4) The brute force method requires less time preparation but significant on-line computational time besides featuring rather limited generalization capabilities for 2, 4, 6, 8 and 10-ternary MPRs.

As future works on designing the ternary MPRs, some highlights could be proposed for example: 1) increasing the number of *m*-ternary models, such as 12-ternary or more, will be a good advantage to test both NF and RNN methods, as well as proposing a ternary MPRs with more than one DOF; 2) comparing the performances of neural network methods, such as: ERNN, MLP and HN, as proposed in this thesis, with Boltzmann machine, RBFN, Jordan RNN, WNN and SVM that explained in Chapter 5; 3) replacing Hill Climbing as a local search procedure for finding the initial optimized parameters, with extended version of global search algorithm such as genetic algorithm (GA) or multi-objective optimization (MO) strategy.

References

[1] Pieper D.L., “The Kinematics of Manipulators under Computer Control”, PhD Thesis, Stanford University, Stanford, CA, 1968.

[2] Roth B., Rastegar J. and Sheinman V., “On the Design of Computer Controlled Manipulators”, *First CISM-IFTMM Symposium on Theory and Practice of Robots and Manipulators*, pp. 93-113, 1973.

[3] Chirikjian G. S.,”A Binary Paradigm for Robotic Manipulators*”, Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pp. 3063-3069, 1994.

[4] Chirikjian G.S., Lees D.S., “Inverse Kinematics of Binary Manipulators with Applications to Service Robotics”, *Proceedings of the 1995 IEEE International Conference on Intelligent Robots and Systems*, pp. 65-71, 1995.

[5] Lees D.S., Chirikjian G.S., “A Combinatorial Approach to Trajectory Planning for Binary Manipulators”, *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pp. 2749-2754, 1996.

[6] Chirikjian G. S.,”Inverse Kinematics of Binary Manipulators Using a Continuum Model*”, Journal of Intelligent and Robotic Systems*, vol.19, pp.5-22, 1997.

[7] Ebert-Uphoff I., Chirikjian G.S., “Inverse Kinematics of Discretely Actuated Hyper-Redundant Manipulators Using Workspace Densities”, *Proceedings of*

the 1996 IEEE International Conference on Robotics and Automation, pp. 139-145, 1996.

[8] Suthakorn J. and Chirikjian G. S.,"A New Inverse Kinematic Algorithm for Binary Manipulators with Many Actuators", *Advanced Robotics*, vol. 15, n. 2, pp. 225-244, 2001.

[9] Sujan V.A., Lichter D., Dubowsky S., "Lightweight Hyper-Redundant Binary Elements for planetary Exploration Robots", 2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, pp. 1273-1278, 2001.

[10] Lichter D., Sujan V.A., Dubowsky S., "Computational Issues in the Planning and Kinematics of Binary Robots", *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, pp. 341-346, 2002.

[11] Yang, P., "Design and Control of Bundles of Binary Actuators for Manipulator Actuation," PhD Dissertation, The Ohio State University, 2001a.

[12] Yang P., Waldron K.J., "Massively Parallel Actuation", 2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, pp. 868-873, 2001b.

[13] Mukherjee S. et Murlidhar S., "Massively parallel binary manipulators", *Journal of Mechanical Design*, vol. 123, n. 1, pp. 68-73, 2002.

[14] Elman J.L., "Finding structure in time", Cognitive Science 14, pp. 179-211, 1990.

[15] Jordan M.I., “Serial order: A parallel distributet processing approach”, Neural-Networks Models of Cognition, Elsevier, pp. 471-495, 1997.

[16] Palit AK, Babuška R, “Efficient training algorithm for Takagi-Sugeno type Neuro-Fuzzy network,” Proc. of FUZZ-IEEE, Melbourne, Australia, vol. 3: 1538-1543, 2001.

[17] Palit AK., Popovic D., “Nonlinear combination of forecasts using ANN, FL and NF approaches,” FUZZ-IEEE, 2:566-571, 2002a.

[18] Palit AK, Doeding, Anheier, Popovic, “Backpropagation based training algorithm for Takagi-Sugeno-type MIMO neuro-fuzzy network to forecast electrical load time series,” Proc. Of Fuzz-IEEE, Honolulu, Hawai, vol. 1:86-91, 2002b.

[19] Palit AK., D. Popovic, “Computational Intelligence in Time Series Forecasting, Theory and Engineering Applications”, Springer, 2005.

[20] Toha S.F., Tokhi, M.O., “MLP and Elman Recurrent Neural Network modeling for the TRMS”,IEEE, Cybernetic Intelligent Systems, 2008.

[21] Xiaosong D, Popovic D, Schulz-Ekloff, Oscillation resisting in the learning of Backpropagation neural networks, Proc. of 3rd IFAC/IFIP, Ostend, Belgium, 1995.

[22] Takagi, Sugeno, “Fuzzy identification of systems and its applications to modeling and control,” IEEE Trans. Syst., Man, Cybern., vol. SMC-15, pp. 116–132, Jan. 1985.

[23] Juang C.F., “A TSK-Type Recurrent Fuzzy Network for Dynamic Systems Processing by Neural Network and Genetic Algorithms”, IEEE Transactions on fuzzy systems, Vol. 10, No. 2, 2002.

[24] Pasila F., “Forecasting of Electrical Load using Takagi-Sugeno type MIMO Neuro-Fuzzy network”, Master thesis, Bremen, 2006.

[25] Flake, G.W., “Square Unit Augmented, Radially Extended, Multilayer Perceptrons” , Lecture Notes In Computer Science; Vol. 1524, Neural Networks: Tricks of the Trade", pages: 145 - 163,Springer-Verlag London, 1998.

[26] Hagan, M.T., Menhaj, M.B., “Training Feedforward Networks with the Marquardt Algorithm”, IEEE Transactions on Neural Networks, Vol. 5, No. 6, November 1994.

[27] Ogata, K., “Modern Control Engineering, Fifth Edition”, Prentice Hall, 2010

[28] Hopfield, J.J., Tank, D.W., ‘Neural Computation of Decisions in Optimization Problems’, Biol. Cybern: 52, 141-152, Springer-Verlag, 1985

[29] Hopfield, J.J., ‘Neurons with graded response have collective computational properties like those of two-state neurons’, Proc. Natl. Acad. Sci. USA 81, 3088-3092, 1984

[30] Ackley D.H., Hinton G.E., ‘A Learning Algorithm for Boltzmann Machines’ , Cognitive Science 9, p. 147-169, 1985

[31] Haykin S., ‘Neural Network: A Comprehensive Foundation’, Prentice Hall, 1999

[32] Broomhead, D.S. and Lowe D., ‘Multivariate functional interpolation and adaptive networks’, Complex Systems 2:321-355, 1988

[33] Jang J.S.R., “ANFIS: Adaptive network Based Fuzzy Inference System,” IEEE Trans. On SMC., 23(3):665-685, 1993.

[34] Daubechies I., Ten Lectures on Wavelets, SIAM, 1992.

[35] Zhang Q. and Benveniste A., “Wavelet Networks”,IEEE Trans on Neural Networks, Vol. 3, No. 6, 1992

[36] Rao SS. and Kumthekar B., “Recurrent Wavelet Networks”, IEEE, 1994

[37] Abiyev RH. and Kaynak O., “Identification and Control of Dynamic Plants Using Fuzzy Wavelet Neural Networks”, IEEE International Symposium on Intelligent Control, USA, 2008

[38] Abiyev RH., “Fuzzy wavelet neural network based on fuzzy clustering and gradient techniques for time series prediction”, Neural Comput & Applic, 20:249–259, 2011

[39] Khao T.Q.D. et al, “Application of Wavelet and Neural network to Long-Term Load Forecasting”, POWERCON, Singapore, pp.840-844, 2004

[40] Poggi J.M. et al, “Wavelet Toolbox, for use with Matlab”, User’s Guide, MathWorks, 1996

[41] Kumar V. et al, “Adaptive control of Inverted Pendulum using Neuro-Fuzzy Inference”, ICWET 2010 – TCET, Mumbai, India, 2010

[42] Cheng K.H. et al, “Recurrent neuro-fuzzy hybrid-learning approach to accurate system modeling”, Fuzzy Sets and Systems 158 (2007) 194 – 212, Elsevier, 2007

[43] Popovich D. et al, “Electrical Load Forecasting Using a Neural-Fuzzy Approach”, Nat. Intel. for Sched., Plan. and Pack. Prob., SCI 250, pp. 145–173, Springer-Verlag Berlin, 2009

[44] Vapnik V., “On the uniform convergence of relative frequencies of events to their probabilities”, Theory of Probability, Vol. 16, No.2, 1971

[45] Vapnik V., “The Nature of Statistical Learning Theory”, Springer-Verlag, New York, 1995

[46] Mukherjee S. et al, “Nonlinear Prediction of Chaotic Time Series Using Support Vector Machines”, Proc. Of IEEE NNSP, Florida, 1997

[47] Cao L. et al, “Financial forecasting using Support Vector Machines”, Neural Computation & Application, 10:184-192, 2001

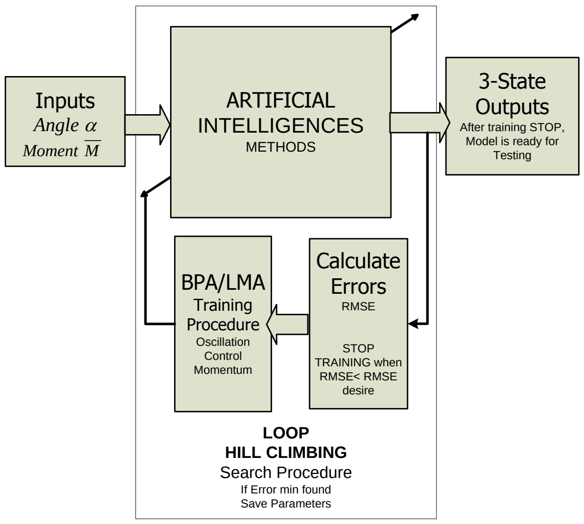
[48] Di Canio, “Analisi di un sistema di attuazione iper-ridondante e a regolazione discreta”, Laurea Thesis, University of Bologna, 2011

[49] Yunfeng W, Chirikjian G.S., “Workspace Generation of Hyper-Redundant Manipulators as a Diffusion Process on SE(N)”, IEEE Transaction on Robotics and Automation, 2004

APPENDIX A

HILL CLIMBING PROCEDURE

The randomized Hill Climbing procedure (HC) is a local search algorithm and tries to find the best local minimum from the huge iteration procedure, by permitting the best training parameters that minimize the error function (SSE) and neglecting the others.



The following steps of randomized HC are carried out:

a. Make the loop : 1) case NN: No of neurons X_{hidden} in hidden layer from 5-50; 2) case NF : No. of N rules from 2 – 15.

b. Make the loop : No of experiments of each initial parameters = 100 .

c. Make the loop : No of iteration, each experiment has iteration from 10 to 1000.

d. Calculate training output and find the suitable prediction output.

e. Save the parameters after step ***d*** completed. If the next iteration produce better result, replace the old parameters, otherwise the new parameters are neglected.

f. Determine the resultant moment prediction and calculate the prediction performance error.

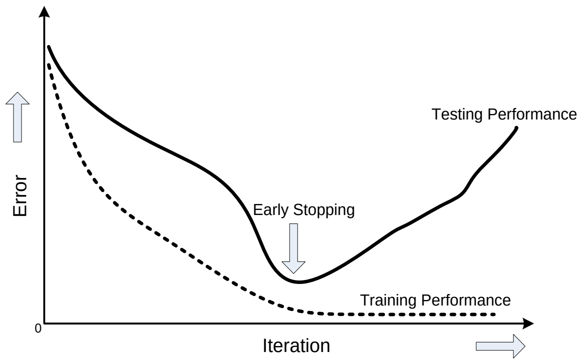
g. Early stopping criteria (see Appendix B): Terminate the program if the requirements are satisfied in step *f.*

h. Repeat again step ***a*** if there is no satisfied results until the loops are finished and save the best parameters.

APPENDIX B

EARLY STOPPING PROCEDURE

In practice, the principle of early stopping procedure is that the network should be trained until it has learned most of the important information for prediction step. This is of course difficult to find out because there is no best approach how to do this. In general, a high enough number of training steps or epochs are good enough, in the sense that the network has learned well in a specific region. Furthermore, reaching the local minimum of the objective function is accepted as the training efficiency merit [Palit, 2005, p.116; Pasila, 2006, p.15], so that after reaching this minimum value, the error function will steadily decrease until the minimum is reached. If there is no further decrease of the error function, this would then be an indication to stop the training iteration.



In our Neuro-Fuzzy or Neural Network simulations, early stopping can be done by doing several training with different number of epochs (for example in search programming strategy, iteration starts from 10 to 1000). In the iteration process, we collect the results of training and testing error from the same iteration. The suitable number of iteration which gives minimal value of desired error can be easily found by memorized all values of training and testing performances.