MINES: Mutual Information Neuro-Evolutionary System

Behzad Behzadan

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of the

University College London.

Department of Computer Science

University College London

March 6, 2011

Declaration

The candidate confirms that the work submitted is his own and that appropriate credit has been given where reference has been made to the work of others.

Abstract

Mutual information neuro-evolutionary system (MINES) presents a novel self-governing approach to determine the optimal quantity and connectivity of the hidden layer of a three layer feed-forward neural network founded on theoretical and practical basis. The system is a combination of a feed-forward neural network, back-propagation algorithm, genetic algorithm, mutual information and clustering. Back-propagation is used for parameter learning to reduce the system's error; while mutual information aides back-propagation to follow an effective path in the weight space. A genetic algorithm changes the incoming synaptic connections of the hidden nodes, based on the fitness provided by the mutual information from the error space to the hidden layer, to perform structural learning. Mutual information determines the appropriate synapses, connecting the hidden nodes to the input layer; however, in effect it also links the back-propagation to the genetic algorithm. Weight clustering is applied to reduce hidden nodes having similar functionality; i.e. those possessing same connectivity patterns and close Euclidean angle in the weight space. Finally, the performance of the system is assessed on two theoretical and one empirical problems. A nonlinear polynomial regression problem and the well known two-spiral classification task are used to evaluate the theoretical performance of the system. Forecasting daily crude oil prices are conducted to observe the performance of MINES on a real world application.

Acknowledgements

I would like to express my sincere appreciation and great thanks to my dear supervisor Dr. Robert Elliott Smith for his valuable advice, guidance, constructive suggestions and encouragement trough the course of this work. I express my deep sense of gratitude to Professor Philip Treleaven, acting as my second supervisor, for his conceptual and theoretical assistance. I am extremely grateful to Dr. Peter J. Bentley for his important suggestions examining my M.Phil.-to-Ph.D. transfer viva. Last but not least, my everlasting gratitude goes to my lovely parents who have been my supporters at all times and to my dearest brother and sister who are truly my enduring and reliable friends.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Neural networks (NNs) are extensively used in various fields of study and had been one of the most popular areas of research in the past years [50][45]. The applications of NNs are broad and diverse. In a comprehensive study, Paliwal and Kumar (1999) [45] categorise the applications of NNs into the following areas where traditionally, statistical techniques were used: accounting and finance, health and medicine, engineering and manufacturing, marketing and decision making. Indeed, NNs can be used as an alternative "to traditional statistical models" [45], such as regression, and are also widely used in classification [83] and business applications [50].

A three layer *feed-forward neural network* (FFNN), with an input layer, one hidden layer and an output layer, is the simplest form of a NN comprising of hidden nodes. Despite their simplicity, three layer FFNNs are widely used due to the "universal approximation theorem"[1] [26], asserting that any continuous function defined on a unit hypercube can be desirably approximated by a composed linear sum of continuous functions of one variable. The theorem simply states that, a NN with one hidden layer is sufficient for any mapping problem (see Section 2.2.2.1 for further detail). However, the theorem does not propose a way to realise how a single hidden layer is optimal "in the sense of learning time, ease of implementation, or (more importantly) generalization" [26].

To obtain an optimal hidden layer, the NN is required to be trained by a simultaneous consideration of the number of hidden nodes, the synaptic connections and the associated synaptic weights. Nonetheless, as Haykin (2009) [26] notes, "there are currently no well defined rules" to develop a "specialized structure" for a NN at hand. Hence, the issue of configuring the appropriate structure of a hidden layer remains an open problem in the field of NN training. The underlying research illuminates a novel and theoretically well founded technique to determine the structure of the hidden layer and the contributed synaptic weights in a three layer FFNN.

In this chapter, the main ideas and motivations of *mutual information neuro-evolutionary system* (MINES), is briefly explained. The scope of the current research - i.e. the limitations, main objectives and hypothesis - is addressed. A brief overview of the system is also presented. Finally, the involved challenges and the achieved contributions of the underlying research is discussed.

[1]The theorem is also called the "Kolmogrov theorem." [21]

1.1 Problem Statement

The performance of a NN is highly dependent on the structure and the contributed weights. Weight adjustment is usually carried out by parameter learning techniques, while structure is sought to be optimized through structural learning [62]. However, the two techniques are not totally independent of each other and there is an intertwined relation between them. For instance, the outcome and efficiency of a NN is highly influenced by the structure, making weight adjustment insufficient as the only technique used for NN optimization. In fact, the two procedures must be carried out simultaneously in order to achieve a close to optimal NN for a given problem at hand. Concerning this, and by focussing on *neuro-evolution* (NE), Stanley and Miikkulainen [63] assert that structural optimization along with adjustments to synaptic weights leads to a superior performance.

To find a desirable NN, the hypothesis space must therefore explore all free parameters constituting the set of entire possible configurations [70] and synaptic weights, having the predefined number of input and output layer nodes, but flexible hidden layers size. However, difficulty arises in such situations: first, the number of possible structures or configurations, alongside synaptic weights, is too large to exhaustively search; second, for many structural configurations, the decision surface is fairly likely to fall into a local minimum, especially when conventional parameter learning techniques such as *back-propagation* (BP) are used; third, the number of potential synaptic connections increases exponentially with the number of existing nodes (see Section 2.2.2).

Reducing the number of free parameters make search the hypothesis space more possible. Focusing on a three layer FFNN, the number of free parameters can typically be reduced (see Section 2.2.2) by decreasing the number of hidden nodes (node-redundancy), eliminating synaptic connections (pruning) and imposing same synaptic weights on some connections (weight-sharing). Conventionally, the pre-mentioned techniques are often carried out on a trial and error basis to find optimal configurations with respect to the underlying problem. In other words, the procedures are often problem based in a way that each technique must be applied several times to a problem to reveal the optimal number of parameters and configurations.

In a different approach - with the aid of information theory - a light NN structure can be obtained by avoiding hidden nodes having high pair-wise *mutual information* (MI). The NN, otherwise, would often possess more than necessary hidden nodes. To reduce the number of hidden nodes, they should be substituted by fewer, jointly sharing less information, while in total conveying the same level of information from the error space as before. MINES pursues a conceptually similar approach utilizing an evolutionary algorithm - the *genetic algorithm* (GA). The goal is to evolve new hidden nodes, mutually possessing less information in pairs, but still capable of reducing the total error of the network. The evolution, which is only applied to the receptive connectivity patterns, may reduce the number of contributed hidden nodes, or it may only change the connections without significantly affecting their quantity.

1.2 Assumptions and Limitations

MINES, in concept, is able to be extended to a multi layer FFNN (see Section 7.2). The *assumption* of a three layer FFNN, with an input layer, one hidden layer and an output layer, is to focus on the evaluation of the basic strength and capabilities of the ideas adopted through minimizing the involved factors. Nevertheless, the imposed limitation is also justifiable as the universal approximation theorem (addressed at the beginning of this chapter) assures that a FFNN with one hidden layer is potentially sufficient for any mapping problem.

It is worth noting that, although GA, BP and clustering have all been combined in MINES, the central aim of the research is not to demonstrate the benefit of the concurrent adaptation of these systems; rather, they mainly provide a means for the MI to appropriately participate in both the structural and parameter learning procedures (see Figure 1.1) to demonstrate that the information provided from the error space is able to determine the configuration of the hidden layer.

1.3 Hypothesis and Objectives

The central hypothesis of the underlying research is to demonstrate that *by aligning the MI of the output of a hidden node and the corresponding residual error of the system, the proper receptive fields connectivity pattern of the hidden node, in relation to the other incorporating elements, can be determined so that the evolving hidden layer would form an optimal, or close to optimal, FFNN.*

In brief, the most important objectives of this thesis are primarily to illustrate the following properties:

- A single hidden layer can properly be structured with the aid of MI by simultaneously aligning the outputs of the incorporating hidden nodes with the residual errors.
- A concurrent parameter and structural learning can be achieved by indirectly connecting the BP to the GA via MI.
- The comprising system of NN, BP, GA, MI and clustering can result to an optimal, or close to an optimal, NN.
- MI is a competent statistical tool which is capable of conveying information from the error space to the hidden layer.
- Even though for continuous random variables MI cannot be measured precisely, when combined with appropriate techniques, the dependency of the underlying variables (i.e. their linear or non-linear co-movements) are eventually revealed, albeit being estimated by the naive histogram technique [64].
- On the application of daily crude oil price forecasting, MINES is automatically capable of discriminating between the most strong input signals and the rest through searching possible receptive fields connectivity patterns.

1.4 System Overview

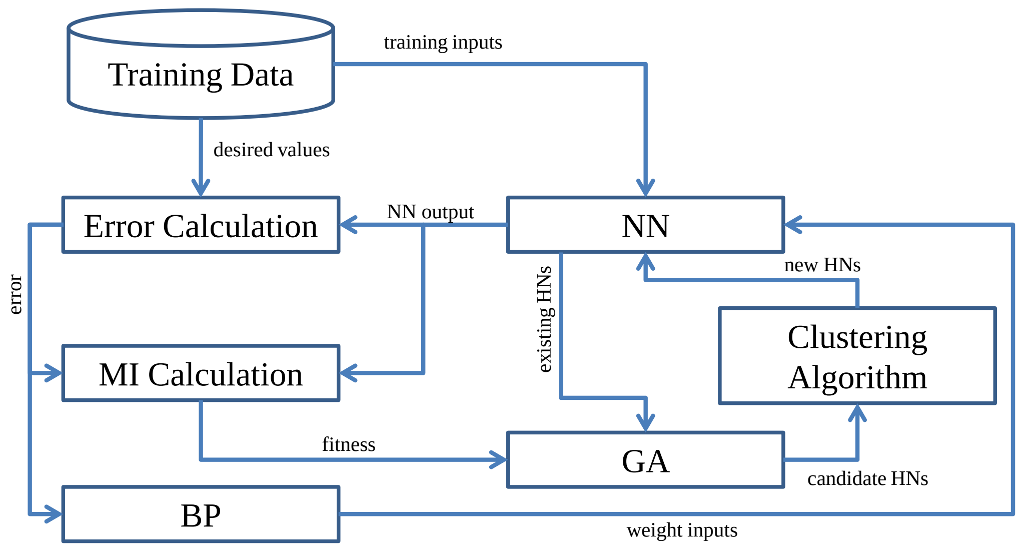


Figure 1.1: MINES diagram

As it will be further described in Chapter 4, MINES achieves the addressed objectives by a simultaneous consideration of parameter and structural learning to train a three layer FFNN. It alternately uses GA and BP alongside the other: the GA is responsible for changing the structure, while BP reduces the cost function. Each GA individual is associated to a hidden node (not necessarily unique) in the FFNN and controls the receptive connectivity pattern of that hidden node. The fitness of each GA individual is the MI between output of the associated hidden node and the NN's residual error. The residual error is the remaining error of the system after the exclusion of a specific hidden node. In each stage, the relative ranks of the cooperating hidden nodes, provided by the respective MIs, decide the final integrated hidden layer. MINES intends to create a hidden layer comprised of nodes being highly correlated with the error space. In this perspective, hidden nodes with low MI need to be replaced by those delivering higher information.

It must be noted that, the MI depends on the total state of the representing hidden node, including its receptive connectivity patterns and the associated synaptic weights. This is where GA and BP are indirectly connected through MI: the former controls the connectivity patterns, while the latter adjusts the synaptic weights. To this end, a successful combination of structural and parameter learning has been established by alternatively running BP for some iteration steps, followed by applying a GA generation. Since the GA evolution is based on the fitness function, being the MI, the role of MI is revealed as the indirect connector of the GA and BP in the following fashion: (1) BP is paused, run after several iterations; (2) MIs are measured, providing the fitness values; (3) GA determines the appropriate local structure; (4) BP is resumed from usually a new point in the weight space, adjusted previously by GA; (5) GA is re-applied to the current population; (6) the whole training procedure stops when the system converges.

Note that, in step (5) in above, individuals of the current population have same connectivity patterns, but different synaptic weights in comparison to the previous generation. Connectivity patterns are encoded directly from related genotypes (individuals) to the corresponding phenotypes (hidden nodes), while synaptic weights are encoded indirectly.

It is worth mentioning that, in MINES the incoming weight vectors are clustered to eliminate hidden nodes with similar functionality. Similar hidden nodes do not often need to be incorporated into a NN with the intention of achieving a light structure.

1.5 Challenges

Coordinating of the cooperative systems, implemented in MINES, is one of the foremost challenges of the underlying research. MIs, on the other hand, must be estimated for continuous random variables. Given a hidden node, the residual errors and the produced values of the hidden node are considered as two continuous random variables which their MIs must be estimated. The collaborating systems in MINES must therefore be chosen so that they could sufficiently disclose the real values of MI to the system over long run. As a result, adjusting the involved parameters of the systems incorporated in MINES is regarded as another challenge of the research. The latter must be addressed properly to correctly tune the final system.

1.6 Contributions

To the greatest extent, this thesis contributes to the field of machine learning by applying information theory in the area of NN training. More specifically, the most important and outstanding contributions of the current research can be summarised as follow:

- A three layer FFNN is converged to an optimal, or close to optimal, NN through evolving the receptive fields of the hidden layer by maximising the MI measured between a hidden node and the associated residual error assigned as the fitness function of GA individuals (see Chapters 5 and 6).

- A novel NN training system, simultaneously hybridising structural and parameter learning, is achieved by employing MI conveying informations from the error space to the hidden layer (see Chapters 5 and 6).

- An optimal, or close to optimal, number of hidden nodes is automatically achieved through the system's convergence (see Chapters 5 and 6).

- The proposed system, MINES, is a multi-applicable system performing well on three different problems from three different areas; including regression, classification and business forecasting (see Chapters 5 and 6).

- MINES shows to be robust against overfitting on the experimented polynomial regression problem (see Sections 5.1.5 and 5.1.6).

- MINES is incorporated with an automatically pruning technique distinguishing the most important input signals from the rest, when applied to the daily crude oil forecasting problem (see Section 6.4).

The most general and conceptual idea of MINES, accompanied by some carried out preliminary tests on the polynomial regression problem [56], was presented in the *eleventh congress on evolutionary computation (IEEE CEC 2009)* [60], held in *Trondheim, Norway in 2009*.

1.7 Thesis Outline

Chapter 2 deals with the most fundamental understandings required to comprehend the general concept of the underlying research area. The Chapter starts with a short description of artificial NNs followed by a broad introduction of parameter and structural learning utilised in NN training. It then focuses on a concise review of evolutionary algorithms, which have been introduced subsequently, to highlight their roles in NN training (related literature is provided extensively in Section 3.2). Finally the chapter is concluded by presenting the motivations behind the proposing system by a thorough comparison with the preceding systems.

Chapter 3 provides a broad literature of the works associated to MINES in four categories: information based systems, neuro-evolutionary systems, hybridization of GA and BP, and pruning, weight-sharing and neurogenesis.

Chapter 4 explains the implementation and stepwise methodology used in MINES. The details of GA and clustering are presented precisely. The effect of setting the number of GA generations and BP iterations are generally explained. A separate section elucidates the mechanism which leads to the convergence of MINES toward a steady FFNN.

Chapter 4 proposes the mathematical foundations required to support the implementation of MINES. The implementation demands an extensive understanding about the creation of a FFNN together with an in depth knowledge of BP algorithm and the way weight vectors with similar functionality are clustered. The chapter covers the concept of MI, defined based on conditional probability, while distinguishing its properties in relation to the classical Pearson's correlation. An exhaustive mathematical definition of entropy and MI, followed by the histogram technique used for the calculation and approximation of the underlying measurements, are provided afterwards.

Chapter 5 and 6 experiment the applications of MINES on three diverse areas: the polynomial regression problem, the two-spiral classification task and the daily crude oil price forecasting as a real world business application.

Chapter 7 summarises the significance of MINES and the specification of the system concisely alongside the future directions.

Chapter 2

Background

This chapter provides a fundamental overview of the topics discussed in the literature and the systems involved in MINES. The chapter starts with a brief overview of the artificial NN, followed by a conceptual discussion of the fundamental procedures involved in NN training making input-output mapping possible. In the widest scope, the training procedures can be divided into two categories: parameter learning and structural learning. The BP algorithm, as the most important and widely used parameter learning technique, is presented in detail. Thereafter, some important structural learning techniques (node-redundancy, pruning, weight-sharing and *cascade correlation neural network* (CCNN))are explained. Evolutionary techniques and *learning classifier systems* (LCSs) were briefly addressed as alternative tools for NN training (additional literature is presented in Section 3.2).

Section 2.3 introduces MI based on information theory and present it as a strong statistical tool called a “generalized measure of correlation” [64] in comparison to the “Pearson correlation”. The advantage of using MI over the conventional correlation was discussed following by an example presenting that a vanishing Pearson correlation does not necessarily imply the independence of the underlying variables.

Subsequently, Section 2.4 deals with the mathematical definition and calculation of entropy and MI. Section 2.4.1 introduces the entropy and MI for discrete random variables. Section 2.4.2 defines the histogram technique, using estimation from discrete case, to approximate entropy and MI for continuous random variables. The approximation is required for the calculation of the fitness values of the GA individuals in MINES. Section 2.4.3 explores those parameters affecting the accuracy of the histogram technique when calculating entropy and MI. The discovery of the parameters aids to understand correct settings required for the calculation of MI in MINES. It will be shown that the most important parameter in this regard is the number of histogram bins and the size of underlying data set.

Finally, Section 2.5 introduces the motivators of MINES while presenting the basic ideas leading to the creation of the system. The rationales behind incorporating the subcomponents forming the integrated system are also discussed.

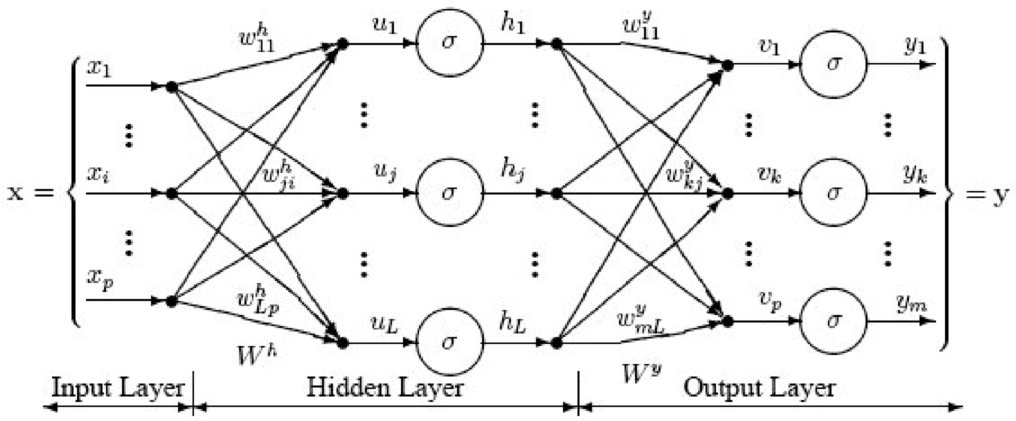


Figure 2.1: A three layer fully connected FFNN (URL: http://www.dtreg.com/mlfn.htm).

2.1 Artificial Neural Networks

The artificial NN was originally motivated from the human brain. It is primarily a highly simplified representation of the way the human brain works. Human brain mechanisms are nonlinear and highly parallel, completely different from conventional digital computers. The brain has the ability to create its own mechanisms through experience with its environment. Neurons are the main components of the human brain which process and transmit information, while communicating via chemical and electrical synapses [26]. Likewise, NNs are formed by computing cells, called neurons, which can align in-line or parallel to each other. NNs can, to some extent, replicate the parallel processing ability of the human brain. Each neuron consists of some input and output terminals, through which neurons interact with each other. These resemble synapses of the nerve cells in the human brain. Similar to the human brain, NNs learn from their environment and can store what they learn by adjusting the inter-neuron connections or weights [26].

2.2 Neural Network Training

Training a NN is a process of finding synaptic weights and connections that reduce the error of the network over the training data. In its broadest sense, NN training is divided into two categories: parameter learning and structural learning. The former concentrates on the adjustment of synaptic weights, while the latter mainly deals with the number of hidden layers, number of hidden nodes and the configuration of synaptic connections. Evolutionary techniques can also be used in NN training and therefore can be considered separately.

2.2.1 Parameter Learning

In parameter learning within a NN the goal is to find the suitable weights. This usually involves an optimization procedure over continuous variables. Evolutionary techniques - such as GA - can be exploited in this regard to adjust synaptic weights in a NN [40], but the typical procedure is BP.

2.2.1.1 Back-Propagation and Algorithm

BP is a continuous parameter learning algorithm based on a method called the "steepest descent" [26]. The BP algorithm usually starts from a random point in the weight space. It reduces the error by employing a point to point search technique towards a minimum point in the neighbourhood by using an approximation in the direction of the tangent line to the weight space at the current point. The approximation is provided by a parameter called "learning rate", being usually set by the user[1].

The learning rate cannot be chosen unboundedly as this may cause oscillation [23] [26] in the learning process: a big learning rate may point out to a point in the error space producing higher error compared to the current position. In fact, the smaller the learning rate, the smoother the learning path towards the next adjacent minimum [26]. However, a very small learning rate increases the number of training iterations and time [23] [26].

To use a large learning rate, but at the same time avoid system instability, another parameter called "momentum constant" is usually adopted to the BP algorithm. The momentum constant is especially beneficial where the curvature of the error space is almost flat [23]; however, it can also smooth the path where the weight approaches its minimum.

Algorithm

Given a FFNN, the output value of node k is compared to the respective desired values to obtain the node's error [25]:

$$e_k(n) = d_k(n) - y_k(n) \tag{2.1}$$

where, in Equation 2.1, n is the training iteration, e_k is the node error and y_k and d_k are, respectively, the output and the desired values.

Using Equation 2.1, the instantaneous value of the total error energy - i.e. the *mean squared error* (MSE) - is defined as [25]:

$$\zeta(n) = 1/2 \sum_{k=0}^{l-1} (e_k(n))^2 \tag{2.2}$$

where l is the number of output nodes of the NN.

Equation 2.2 is used for sequential training, whereas in the batch mode the sum of MSEs over the entire epoch is considered in the training procedure[2] [25]; i.e.:

$$Z(n) = \sum_{p=0}^{s-1} \zeta_p(n)$$

where s is the size of the training epoch and ζ_p is the instantaneous value of the total error energy for pattern p.

The method of the steepest descent requires to obtain the derivative of $\zeta(n)$ with respect to the elements of the weight matrix. For a NN with no hidden layer - only comprised of input and output

[1]MINES, in contrary, is adopted with an automatically adjusted learning rate system explained in Section 4.2.1.

[2]Sequential training is a process in which input vectors are presented to the network one-by-one; for each presentation respective weights adjustments are applied. In the batch training all inputs are presented to the network to calculate the cost function of the network; weights are then adjusted for the entire batch of inputs [26].

layers - the sensitivity of the cost function with respect to the synaptic weights are obtained as follow:

$$\frac{\partial \zeta(n)}{\partial w_{ij}(n)} = -\sum_{k=0}^{l-1} e_k(n)\varphi'_k(v_k(n))\frac{\partial v_k(n)}{\partial w_{ij}(n)} \tag{2.3}$$

where w_{ij} is the receiving synaptic weight of node i, incoming from node j from previous layer, v_k is the induced local field of node k and φ' is the derivative of the activation function. All the elements of the weight matrix, subsequently, are shifted equal to the amount of the assigned learning rate in the direction of the derivative vector. Usually, the procedure continues til the MSE goes lower than a predefined threshold.

Each layer (neuron) of a multilayer NN can be regarded as a NN with no hidden layer; considering its input layer as the output layer of the previous neuron and its output layer as the input layer of the subsequent neuron. Hence, the error calculation formula for a multilayer NN is similar to a NN with no hidden layer. However, the training procedure of a multilayer NN converts to a recursive form called BP algorithm. This is because the weight adjustments of each layer depends on the modification of the previous layer, as will be shown accordingly.

Define L as the final output layer of the network. The sensitivity of the cost function with respect to the nodes' incoming weights in layer L is calculated as:

$$\frac{\partial \zeta(n)}{\partial {w_{ij}}^{L}(n)} = e_i(n){\varphi'_i}^{L}({v_i}^{L}(n)){y_j}^{L-1}(n) \tag{2.4}$$

where n is the iteration number that the steepest descent algorithm is currently working on, e_i is the error of output node i; ${\varphi'_i}^{L}$ is the derivative of the activation function of output node i, ${v_i}^{L}$ is the induced local field of node i and ${y_j}^{L-1}$ is the output of node j in the previous layer.

By defining delta as follow:

$${\delta_k}^{L}(n) = {\varphi'_k}^{L}({v_k}^{L}(n))e_k(n)$$

Equation 2.4 becomes more compact:

$$\frac{\partial \zeta(n)}{\partial {w_{ij}}^{L}(n)} = {\delta_k}^{L}(n){y_j}^{L-1}(n) \tag{2.5}$$

For a layer back, the delta is calculated with the following formula:

$${\delta_i}^{L-1}(n) = {\varphi'_i}^{L-1}({v_i}^{L-1}(n))\sum_{k=0}^{L-1} {\delta_k}^{L}(n){w_{k,i+1}}^{L}(n) \tag{2.6}$$

Equation 2.6 makes derive the derivative of the cost function with respect to layer $L-1$:

$$\frac{\partial \zeta^{L}(n)}{\partial {w_{ij}}^{L-1}(n)} = -{\delta_i}^{L-1}(n){y_j}^{L-2}(n) \tag{2.7}$$

Thus, from Equations 2.5 and 2.7, the BP algorithm can be calculated recursively for every layer since delta is defined in a recursive form in Equation 2.6.

2.2.2 Structural Learning

Structural learning, in contrast to the parameter learning, deals with the manner in which the neurons and nodes of a NN are structured [26]. Essentially, structural learning is engaged with discrete parameters, mainly the number of nodes and their connections [62]. Structural learning is an important practice as it can reduce the number of free parameters in the network.

In a FFNN, the number of free parameters increases exponentially with the nodes' existing synapses. For instance, in a fully connected FFNN, when the number of input layer nodes increases by one, the number of synapses, φ, increases so that:

$$\varphi = 2^{\eta} \tag{2.8}$$

where η is the number of existing hidden nodes in the next layer. The reason is that, for each hidden node there exist two states with regard to the newly added input node: being connected or not.

In the same manner, when the number of hidden nodes increases by one, the number of synapses increases so that:

$$\varphi = 2^{\iota} + 2^{o} \tag{2.9}$$

where ι is the number of nodes of the previous layer (which could be the input layer or the preceding hidden layer) and o is the number of nodes of the immediate next layer (which could be the output layer or the succeeding hidden layer). The reason is again the state of the newly added hidden node which can be connected or disconnected to the existing nodes of the immediate preceding or succeeding layer.

The above discussion indicates that, in a FFNN any attempt to reduce the number of free parameters would be of interest, subject to not compromising the performance of the system. Nonetheless, there is currently no all-embracing technique to illustrate a way to reduce free parameters in NNs [26]. However, there exist some common ad hoc procedures to address the problem. Some of the important techniques are as follow:

1. Reducing the number of nodes (Node Redundancy)

2. Fixing the number of nodes, but reducing the number of synapses or connections (Pruning)

3. Fixing both the number of nodes and connections, but constraining the choice of synaptic weights (Weight-sharing)

4. Incrementally adding new nodes (CCNN)

Note that, depending on the system being designed, a combination of above techniques can also be used to develop a specialized NN. Other alternative techniques can also be addressed from the fields motivated from natural biology applied in NN. Neurogenesis, for instance, is an example in this regard. Neurogenesis or the artificial morphogenesis of NNs is a process of creating neurons structure based on information of a chromosome [40] (a literature applying neurogenesis to NN structural learning is surveyed in Section 3.4).

The above mentioned techniques (except Neurogenesis) are explained thoroughly in the subsequent sections.

2.2.2.1 Node Redundancy

In a NN, the number of nodes of the input and output layer are often fixed since they are imposed by the implementing model. Hence, the practice of reducing the number of nodes is usually restricted to the hidden layers. In fact, the number of hidden nodes is an important parameter in NN structural learning. Excess hidden nodes often lead to overfitting and poor generalization [23] [5]. A network with a shortage of hidden nodes on the other hand is not able to train well and is over general for a given task. Generalization is a crucial aim of any NN training.

The issue of the number of the contributed hidden nodes becomes more highlighted when a FFNN with a single hidden layer is used. It has been shown by Hornik et al (1989) [30] that a FFNN with one hidden layer is a "universal approximator" - i.e. when equipped with BP algorithm, it can approximate any function, given that sufficient number of hidden nodes are provided [79]. In terms of forecasting accuracy, additional hidden layers provide no advantage; though, may result in faster learning in certain cases [47]. However, Boers and Kuiper (1992) [4] mathematically presented that a FFNN with additional hidden layers (called a modular NN) is more likely to find weights with better generalization ability among the entire weight space (see Section 3.4). Nevertheless, their claim is only valid when every pattern in the epoch is invariant to the input-nodes' permutations. This makes the hidden nodes be also invariant to any permutation among themselves, provided that all the incoming and outgoing synaptic weights goes under the same permutation.

Pendharkar and Rodger (2003) [47] point out that the hidden layer determines the complexity of the forecasting model. They approximated the number of hidden nodes, η, for a fully connected FFNN to be as twice as the number of input nodes, ι, plus one; i.e.:

$$\eta = 2\iota + 1 \tag{2.10}$$

However, Equation 2.10 is only a good approximation for a small sample size; for a bigger sample size the number of hidden nodes must be increased.

The number of contributed hidden nodes strictly plays an important role in a fully connected NN. However, this is not a crucial issue in a partially connected NN as long as appropriate synaptic connections have been established. In fact, a partially connected NN may have the same performance as a fully connected one, while is benefiting from fewer or more number of hidden nodes. Section 2.2.2.2 provides more information about partially connected FFNNs.

2.2.2.2 Pruning

In a FFNN the number of free parameters can be decreased by reducing the number of connections, without reducing the number of hidden nodes. In this fashion, usually the number of hidden layer nodes is determined in advance while the synaptic connections are subject to change. A typical way to this practice is pruning. Pruning is a process of disconnecting some of the synaptic connections of a usually fully connected NN with the ultimate goal of achieving an optimal structure [5]. Pruning, therefore, results in a partially connected NN with lighter architecture having fewer synaptic connections.

In a partially connected FFNN the *degree of connectivity* - or the free parameters - is lower than a fully connected one. Indeed, a fully connected FFNN has the highest degree of connectivity with the given number of nodes for each layer. The degree of connectivity is closely related to the number of synaptic weights which ought to be optimized [79]. Usually, lower number of free parameters leads to lower training time along with a better performance over new data (i.e. a better generalization performance) [5]. In fact, "a network that is too big for a particular classification task is more likely to overfit the training data" and show a poor generalization performance [5]. Therefore, the smallest network performing well on the training data mainly presents a better generalization capability. Pruning additionally improves the accuracy of a NN by eliminating irrelevant input signals. It also has a significant role in the ability of local-learning over the input space. Furthermore, when equipped with BP algorithm, a partially connected FFNN is usually regarded as "a class of the most stable" NNs which "is widely used and admitted in most applications" [79].

2.2.2.3 Weight-Sharing

Weight-sharing is an applied constraint technique in which some synaptic connections are constrained to have the same weights. Weight-sharing is a process through time: the constrained weights often differ from one training step to another, but within every step they are updated in a fashion that they share the same synaptic weights [52]. The shared weights can be imposed to the synaptic connections of any layer or any position in the network. When using the standard BP algorithm, the weight-sharing technique requires updating weights in a way that the shared synapses are set to the sum or the average of the constrained weights [52]. A very advanced and interesting example of the weight-sharing technique can be found in the work conducted by Saito and Nakano (2002) [56] (see Section 3.4.2).

2.2.2.4 Cascade Correlation Neural Network

In CCNN hidden nodes are incorporated in the network in an incremental, bottom-up fashion. CCNN illustrates a structural learning technique to determine the minimal number of required hidden nodes by providing a path in the weight space possessing the highest absolute correlation between the most recently added hidden node's output and the network's error. The motivation is to train hidden nodes so

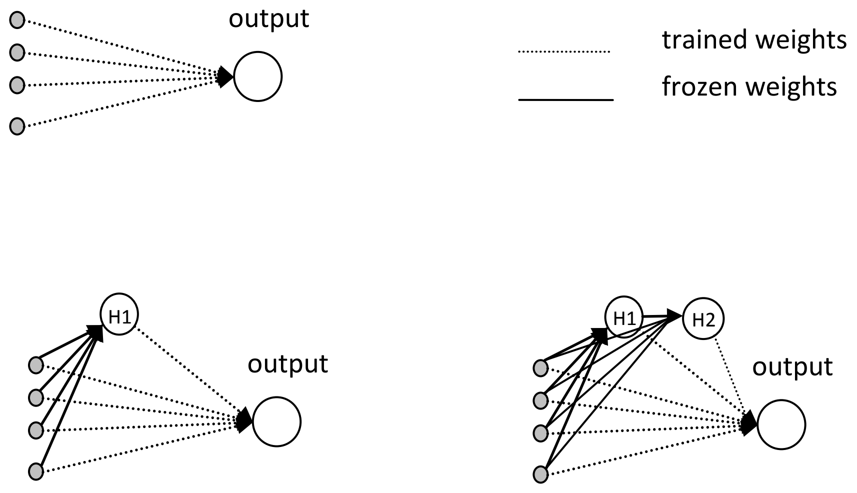


Figure 2.2: The architecture of CCNN [52]

that they can detect the residual error of the existing network. It also tries to improve the speed of the BP learning algorithm [14].

The conventional CCNN procedure starts with a NN which has no hidden nodes: all input nodes are connected to the output nodes. The Quickprop algorithm, which is the same as the delta rule (see Equation 2.3) with no hidden node, is then performed to train the direct input-output connections [14]. If the error is greater than a desired upper bound, a hidden node is added. The added hidden node receives connections from all the input nodes. Note that at this stage the output of the hidden node is not yet connected to the network's output-node, it is only connected to the inputs. The incoming synapses of the hidden node are adjusted so that the correlation, or covariance, between the output of the added hidden node and the network's error becomes maximized. More precisely, the covariance of the output of the most recently added hidden node and the network's error is calculated according to the following formula [52]:

$$cov(V, E) = \sum_{i=1}^{p}(V_i - \bar{V})(E_i - \bar{E}) \tag{2.11}$$

where in that p is the number of input patterns, while V_i and E_i are, respectively, the hidden node's output and the error of the i^{th} pattern; whereas, $\bar{V}$ and $\bar{E}$ are the mean values of V_i and E_i, respectively. Afterwards, the hidden node is connected to the output node. While the previously adjusted incoming weights are kept frozen, the hidden nodes' outgoing synapses are trained by the "Quickprop" process to minimize the network's MSE. The procedure continues in such a way that each newly added hidden node is connected to every node in the input layer and also to all the formerly added hidden nodes; this is when the system is cascaded. When a new hidden node is added, the incoming synaptic weights of the formerly added hidden nodes remain unchanged; just the newly created synapses, and those existing directly between the input and output layer nodes, are trained.

Figure 2.2 illustrates the architecture of CCNN in three stages. The first stage (the top left corner of the figure) is when the system starts from no hidden nodes. The second stage (the bottom left corner of the figure) is when one hidden node is added to the system. The third stage (the bottom right corner of the figure) is when a second hidden node comes into the operation in a newly cascaded layer.

2.2.3 Evolutionary Learning

Evolutionary learnings adopt the application of evolutionary techniques in NN learning or related fields. Evolutionary techniques are based on natural selection and genetics. Darwin's principle of natural selection is used computationally in evolutionary algorithms through artificial gene recombination, mutation, selection and replacement. The principle implies that favourable heritable traits become common in successive generations. All evolutionary computing techniques are faithful to the idea of finding the fittest solution by evolving an initially random population of solutions in a way that fitter outcomes are generated over time.

GA can be referred as the most popular class of evolutionary algorithm invented by John Holland (1975) [29]. They use techniques inspired by evolutionary biology such as inheritance, mutation, selection, crossover (also called recombination). GA usually starts with a randomly created initial population. Members of the population are then evaluated according to an objective function; its value is called fitness or the objective value. A subset of the population is selected based on the fitness of the members. A new population is created from the previously selected members by crossover and mutation. In GA, selection is done by providing more chance to the fitter members which results to the increase of the average fitness in each generation. Crossover makes the algorithm search the space of possible solutions, while mutation avoids the system trap in local minima (literature of the applications of GA in NN learning can be found in Chapter 3).

The application of GA in NN is usually known as NE. NE is the "artificial evolution of NNs using genetic algorithms" [63]. GA can be applied to evolve both the weights and architecture of a NN. When GA is exploited to evolves the weights, the architecture is usually predefined and fixed. However, the GA can also be used to evolve the structure. Nevertheless, as Yao [76] note "there is no systematic way to design an optimal (near optimal) architecture for a particular task". A constructive algorithm starts with a minimal network (i.e. a network with minimal number of hidden nodes and synaptic connections) to form the structure; whereas, a destructive algorithm does the opposite: starts with a maximal network and prunes the unnecessary connections and nodes.

Systems using NE are divided to those using a direct or an indirect encoding scheme between genotypes and phenotypes. In the direct encoding all connections, nodes and weights of the phenotype are exactly coded in the genome [16]. The direct encoding is employed by most of the NE systems [63]. In contrast, indirect encoding is based on the rules coded in the genome to construct a phenotype [12]. Indirect encoding has the advantage of employing more compressed codes than the direct encoding since there is no need to explicitly code all the connections, nodes and information into the genome (literature of NE can be found in Chapter 3).

Memetic Algorithm (MA) is another branch of evolutionary systems for combinatorial optimization problems. "MA is inspired by Dawkins notion of a meme defined as a unit of information that reproduces itself while people exchange ideas" [39]. Similar to GA, where genes are passed to the next generation, memes are also passed generation by generation. However, before being passed to the next generation, memes are usually "adapted by the people who transmit them". MA can be considered as a hybridization of GA and a local search in the space of locally optimal solutions instead of the space of all candidate solutions. Hence, in any generation, the population of individuals consists solely of local optima.

LCS, first introduced by John Holland [29], is a "rule-based" system applying GA on the reinforcement learning. A rule, also called a classifier, is formed by a "condition" and "action" string [59]. Typical LCS conditions and actions are strings of characters from the ternary alphabet $\{0, 1, \#\}$; where in that # acts as a wild-card. The possible rules are searched by evolutionary computing techniques and heuristics (structural learning), while reinforcement learning assigns utilities to them (parameter learning). LCS has a close analogy with NN [61]: rule's condition can be observed as the collection of the node's input-signals, node's connectivity-patterns and node's synaptic-weights; while, rule's action can be regarded as the node's output signal. Due to their analogy with NNs, the study of LCSs would introduce alternative techniques that can be used in NN learning.

Due to the complexity of LCS, Wilson [71] presented an alternative system called ZCS: the zeroth-level classifier system. ZCS - a successor of LCS and a predecessor of XCS - inherits much of the Holland's original framework but is simplified to increase understandability and performance.

XCS, introduced by Wilson [72], was a form of LCS altered the former way of calculating the rule's fitness. In XCS, the rule's fitness of GA is based on its accuracy of prediction in payoff rather than on the expected payoff itself. In general, XCS performs better than its predecessor LCS.

The most recent successor of XCS is the novel system introduced by Smith and Jiang [62] called *mutual information learning classifier system* (MILCS). MILCS, in contrary to XCS, is directly based on supervised learning. It uses MI to maximize correlation between outputs and error. Hence, it is advantageously explicit compared to XCS about accuracy and generalization. Although developed from XCS, MILCS has also an interesting similarity to the CCNN [62]. In the process of training a focussed hidden node, the CCNN usually sets the incoming weights of the hidden node to values close to zero when maximizing the correlation of its output and the network's residual-error. Moreover, later variations of CCNN, in contrast to the original prototype, do not necessarily employ a fully connected system. In fact, in CCNN some of the hidden nodes might be disconnected (or connected by values close to zero) to the preceding nodes. In this sense, MILCS is comparable to CCNN since the latter can be pruned and go through a type of structural learning.

2.3 Mutual Information versus Correlation

Correlation tends to find the co-movement of two random variables based on their co-placement above or under their expected values. Hence, it is quite likely that the calculated correlation turns out to be zero, while in reality a tense interrelationship exists between the variables. This is stated more mathematically by Freund and Walpole (p. 452) [19]:

"if two random variables are uncorrelated they are not necessarily independent"

Section 2.3.1 demonstrates the above fact by a numerical example.

Two discrete random variables X and Y , by definition, are independent if for all x and y in their range (p. 104, 126) [19]:

$$p(x, y) = p(x)p(y) \tag{2.12}$$

where in that:

$$p(x, y) = p(X = x, Y = y) \tag{2.13}$$

is the joint distribution (probability) function of X and Y, and:

$$\begin{cases} p(x) = p(X = x) \\ p(y) = p(Y = y) \end{cases}$$

By dividing both sides of Equation 2.12 by $p(y)$, we have:

$$p(x) = \frac{p(x, y)}{p(y)} \tag{2.14}$$

However, the right hand side of Equation 2.14 is by definition (p. 54) [19] equal to the conditional probability of the event $X = x$ given $Y = y$. Hence:

$$p(x) = p(x \mid y) \tag{2.15}$$

Equation 2.15 indicates that, given that X and Y are independent, the conditional probability of x given y has nothing to do with y. Since a similar argument holds true for Y, it can be concluded that, when X and Y are statistically independent, "the occurrence or non-occurrence of either of them does not affect the probability of the other" (p. 58) [19].

As an absence of correlation (a zero correlation) does not guarantee the independence of the underlying variables.Hence, a superior measurement must be utilized to verify their dependence. Since statistical dependency is defined based on conditional probability, the new measurement logically needs to be based on the conditional probability to fully examine the existing dependency.

MI, in contrast to correlation, tends to calculate the interdependence of two variables by using the conditional probability. In fact, MI vanishes if, and only if, all samples are statistically independent [27]. MI, therefore, has many significant advantages over the conventional correlation. For example, MI is able to measure the general dependency of variables, while correlation can only measure their existing linear relations [38][11]. Since it deals with probabilities, MI is applicable on both symbolic and numerical sequences [38]. This is clearly in contrast to correlation, which is based on simple algebraic operations, restricting its application only to numerical random events. MI also shows more sensitivity, making it desirable, especially, when high precision is required. Moreover, since it is defined by a logarithmic term, MI is convenient for many mathematical manipulations.

Another important attribute of MI is its invariance property under invertible transformations of random variables [26]. To state this property in mathematical terms, suppose U and V are two invertible transformations of random variables X and Y:

$$\begin{cases} U : x \longrightarrow f(x) \\ V : y \longrightarrow g(y) \end{cases}$$

where x and y are the sample values of X and Y, respectively. The invertible property of MI states that the MI of X and Y is equal to the MI of U and V:

$$I(X, Y) = I(U, V) \tag{2.16}$$

The above mentioned property is completely in contrast to the general characteristic of correlation being not invariant with respect to the transformation of coordinates [27]. If it can be named, the only disadvantage of MI over conventional correlation is attributed to its sign: MI is always positive[3], whereas correlation can either be positive or negative, defining correlation and anti-correlation [27], respectively. Hence, when using MI, accompanying techniques are required to generally differentiate between a co-movement and an anti-co-movement.

[3]In the continuous case, the MI can also be negative, but when variables are discrete or when the MI is estimated through discrete variables (like the histogram technique introduced in Section 2.4.2.), it is always positive[35]

2.3.1 Dependent but Uncorrelated Series

Consider the following two series with equi-probable outcomes (i.e. equal to 25% each):

$$X = (-1, -\frac{1}{2}, 0, \frac{1}{2}, 1)$$

$$Y = X^2 = (1, \frac{1}{4}, 0, \frac{1}{4}, 1)$$

Then:

$$E(X) = E(X^3) = 0 \tag{2.17}$$

Using Equation 2.17, it can be shown that the covariance of X and Y is zero:

$$Cov(X, Y) = E(XY) - E(X)E(Y) = E(X^3) - E(X)E(X^2) = 0$$

meaning that the two variables are uncorrelated. However, it is far obvious that the two series are perfectly related, as one is the squared of the other one. For instance, considering the event A:

$$A = (X = -\frac{1}{2}, Y = 1)$$

we have:

$$\begin{cases} p(X = -\frac{1}{2}, Y = 1) = \frac{2}{16} = \frac{1}{4} \\ p(X = -\frac{1}{2}) = \frac{4}{16} = \frac{1}{4} \\ p(Y = 1) = \frac{8}{16} = \frac{1}{2} \end{cases}$$

However:

$$p(X = -\frac{1}{2})p(Y = 1) = \frac{1}{8}$$

indicating that for the event A:

$$p(x, y) \neq p(x)p(y)$$

confirming that X and Y are not statistically independent.

2.4 Measuring Entropy and Mutual Information

The MI plays a central role in the functionality of MINES. It is assigned as the fitness function of the GA individuals (see Section 4.4.2). To this sense, correct measurement of MI is crucially important. In theory, when probabilities are known, the MI can be measured accurately. However, in general, the probabilities are not known and have to be estimated from measurements [64].

In Section 2.4.1, the definition of entropy and MI and their mathematical relations for discrete random variables will be given. In addition, assuming the probabilities are known, the way entropy and MI are measured are addressed. The formulas and the detailed proofs are essential to explore since they provide the basis of understanding the mathematical nature of entropy and MI required for their estimation in continuous case.

Section 2.4.2 deals with the estimation of entropy and MI for continuous random variables. Equations 2.22, 2.23, 2.29 and 2.30, derived for discrete random variables in Section 2.4.1, are also true in continuous case. Nevertheless, the proofs have not been provided again since an estimation from the

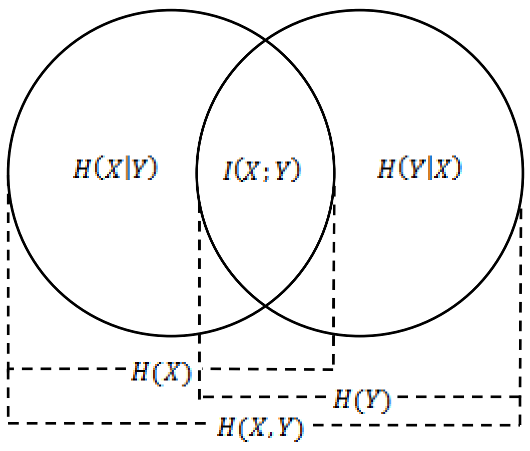


Figure 2.3: Relationships of the entropies and MI between two random variables X and Y

discrete case (i.e. the histogram technique) has been used to measure the entropy and MI for continuous random variables. The estimation of entropy and MI is often necessary for continuous random variables as the "explicit knowledge of respective probability distributions" are not usually known [64].

When finite size samples are used, the histogram technique is susceptible to "systematic errors". For this reason, Section 2.4.3 provides a very detailed and exact mathematical proof of the errors involved in the measurement of entropy and MI when dealing with histogram technique. Further information about calculating the MI of a hidden node in MINES is provided in Section 4.3.1.

It must be noted that the mathematical formulas derived in Sections 2.4.2 and 2.4.3 are, in basic, based on the consolidation works of Herzel et al. (1994) [28], Herzel and Große (1995) [27], Roulston (1999) [53] and Steuer et al. (2002) [64]. The formulas have been provided with additional explanations than the original works, and in some cases have been reordered, to make them easier to understand. Also, different mathematical symbols may have been used in some cases.

2.4.1 Discrete Case

Suppose the discrete random variable, X, is defined over the sample space S_X as:

$$X : S_X \longrightarrow R_X = \{x_1, \ldots, x_{M_X}\}$$

Define:

$$I = \{1, \ldots, M_X\}$$

For $i \in I$, the event $X = x_i$ is defined as:

$$\{s \in S_X \mid X(s) = x_i\}$$

The amount of information gained after observing the event $X = x_i$ is defined as follow [26]:

$$I(x_i) = \log(1/p_i) = -\log(p_i) \tag{2.18}$$

where

$$p_i = p(X = x_i) = p(\{s \in S_X \mid X(s) = x_i\}) \tag{2.19}$$

is the probability of the occurrence of the event. This is equivalent to the amount of uncertainty we had before the occurrence of x_i. In other words, the lower the probability p_i, the more surprise there is in the happening of x_i [26].

The mean value of $I(x_i)$, called the entropy of X and denoted by $H(X)$, is the average of information conveyed per message x_i. In other words, $H(X)$ is a measurement of the uncertainty about X; or, equivalently, is "a measure of how evenly" the states of X are distributed [9]. The entropy of X is mathematically defined as:

$$H(X) = E(I(X)) = -\sum_{i \in I} p_i \log(p_i) \tag{2.20}$$

It must be noted that in Equation 2.20 the sum is over all states that X can assume [53]. If "the outcome of a measurement" is completely known (i.e. there exists an i so that $p_i = 1$), the entropy of X becomes zero, whereas it becomes "maximal" if all probabilities are equal [9].

In the discrete case the elements of R_X are exactly known; hence, the probability values p_i ($i \in I$), are easy to evaluate and the calculation of entropy is straightforward. Consider Y as another discrete random variable defined over the sample space S_Y:

$$Y : S_Y \longrightarrow R_Y = \{y_1, \ldots, y_{M_Y}\}$$

Define:

$$J = \{1, \ldots, M_Y\}$$

The joint entropy of X and Y, $H(X, Y)$, is the total uncertainty about X and Y; defined as:

$$H(X,Y) = -\sum_{i \in I} \sum_{j \in J} p_{ij} \log(p_{ij}) \tag{2.21}$$

where p_{ij} is the joint probability function of X and Y defined on the following sample space:

$$S_X \times S_Y = \{(s, s') \mid s \in S_X, s' \in S_Y\}$$

and:

$$p_{ij} = p(x_i, y_j) = p(X = x_i, Y = y_j) = p(\{(s, s') \in S_X \times S_Y \mid X(s) = x_i, Y(s') = y_j\})$$

The entropy and the joint entropy make define the MI possible. The MI between X and Y, denoted by $I(X, Y)$, is defined as:

$$I(X,Y) = H(X) + H(Y) - H(X,Y) \tag{2.22}$$

It is seen that Equation 2.22 is symmetric with respect to X and Y:

$$I(X,Y) = I(Y,X) \tag{2.23}$$

It should be noted that in Equation 2.23, $H(X)$ and $H(Y)$ are also defined on $S_X \times S_Y$ - the joint sample space of X and Y. In fact:

$$\begin{aligned} p_i = p_X(x_i) &= p(\{(s, s') \in S_X \times S_Y \mid X(s) = x_i\}) \\ &= p(\bigcup_{j \in J} \{(s, s') \in S_X \times S_Y \mid X(s) = x_i, Y(s') = y_j\}) \\ &= \sum_{j \in J} p_{ij} \end{aligned}$$

Hence:

$$H(X) = -\sum_{i\in I} p_i \log(p_i) = -\sum_{i\in I}\sum_{j\in J} p_{ij}\log(p_i) \tag{2.24}$$

Likewise:

$$\begin{aligned} p_j = p_Y(y_j) &= p(\{(s,s') \in S_X \times S_Y \mid Y(s') = y_j\}) \\ &= p(\bigcup_{i\in I}\{(s,s') \in S_X \times S_Y \mid X(s) = x_i, Y(s') = y_j\}) \\ &= \sum_{i\in I} p_{ij} \end{aligned}$$

and:

$$H(Y) = -\sum_{j\in J} p_j \log(p_j) = -\sum_{i\in I}\sum_{j\in J} p_{ij}\log(p_j) \tag{2.25}$$

Therefore, by using Equations 2.21, 2.24 and 2.25, an expansion form of Equation 2.22 is obtained:

$$I(X,Y) = -\sum_{i\in I}\sum_{j\in J} p_{ij}\big(log(p_i) + \log(p_j) - \log(p_{ij})\big) = -\sum_{i\in I}\sum_{j\in J} p_{ij}\log(\frac{p_i p_j}{p_{ij}}) \tag{2.26}$$

The definition of conditional entropy provides a better understanding of MI. For a point $j \in J$, the conditional entropy of X given $Y = y_j$, denoted by $H(X \mid Y = y_j)$, is the amount of uncertainty about X given $Y = y_j$:

$$H(X \mid Y = y_j) = -\sum_{i\in I} p_{i|j}\log(p_{i|j}) \tag{2.27}$$

where $p_{i|j}$ is the conditional probability function of X and Y defined as:

$$p_{i|j} = p(x_i \mid y_j) = p(X = x_i \mid Y = y_j) = \frac{p(X = x_i, Y = y_j)}{p(Y = y_j)} = \frac{p(x_i, y_j)}{\sum_{i\in I} p(x_i, y_j)} = \frac{p(x_i, y_j)}{p_Y(y_j)} = \frac{p_{ij}}{p_j}$$

Given the whole sample space of Y, the conditional entropy of X given Y, $H(X \mid Y)$, is defined as the amount of uncertainty about X given that the whole sample space of Y has occurred:

$$\begin{aligned} H(X \mid Y) = -\sum_{j\in J} p_Y(y_j) H(X \mid Y = y_j) &= -\sum_{j\in J} p_j \sum_{i\in I} p_{i|j}\log(p_{i|j}) \\ &= -\sum_{j\in J}\sum_{i\in I} p_j p_{i|j}\log(p_{i|j}) \\ &= -\sum_{j\in J}\sum_{i\in I} p_{ij}\log(p_{i|j}) \end{aligned}$$

Note that:

$$\log(p_j) - \log(p_{ij}) = \log(\frac{p_j}{p_{ij}}) = \log(p_{i|j})$$

Therefore:

$$\begin{aligned} H(X) - H(X \mid Y) &= -(\sum_{i\in I}\sum_{j\in J} p_{ij}\log(p_i) - \sum_{i\in I}\sum_{j\in J} p_{i|j}\log(p_{ij})) \\ &= -(\sum_{i\in I}\sum_{j\in J} p_{ij}\log(p_i) - \sum_{i\in I}\sum_{j\in J} p_{i|j}\log(\frac{p_{ij}}{p_j})) \\ &= -\sum_{i\in I}\sum_{j\in J} p_{ij}\log(\frac{p_i p_j}{p_{ij}}) \end{aligned} \tag{2.28}$$

From Equations 2.26 and 2.28 the following new formula is obtained:

$$I(X,Y) = H(X) - H(X \mid Y) \tag{2.29}$$

Since $I(X,Y)$ is symmetric, the following equation also holds true:

$$I(Y,X) = H(Y) - H(Y \mid X) \tag{2.30}$$

Equations 2.29 and 2.30 indicate that $I(X,Y)$ is the amount of information in X discovered by knowing that Y has occurred; or vice versa: it is the amount of information in Y discovered by knowing that X has occurred. In other words, MI measures the interdependence of two variables: X and Y.

The relationships embodied in the formulas related to the entropies and MI over the joint distribution of X and Y are depicted in Figure 2.3.

2.4.2 Continuous Case

The histogram technique, also called the "naive algorithm" [64], is "the most straightforward and widely used approach" for the estimation of entropy and MI; although, other techniques using kernel density functions [41][9] can alternatively used. In the histogram technique the calculation of entropy and MI is based on the binning of data into discrete intervals. The binning of the data makes estimation of the entropy and MI possible based on the calculations provided for the discrete case.

Suppose X is a continuous random variable defined over a sample space S_X with a bounded range between the two real numbers a and b, inclusive b:

$$X : S_X \longrightarrow R_X = (a,b] \tag{2.31}$$

Note that, $(a,b]$ is an interval in the real numbers and, potentially, X can get any real number in that. Nonetheless, in most applications, the range of X, R_X, usually varies over a finite size range. For instance, when the size of S_X is finite, R_X would have a finite size too. As a result, we may assume that R_X has a size equal to N:

$$R_X = \{x_1, \dots, x_N\}$$

It is worth noting that, in contrary to the discrete case, the values of $x_1, \dots, x_N$ are not known in advance. This is due to the continuous random variable X which is likely to get any real numbers in the interval $[a,b]$. As a result, it is impossible to calculate the entropy of a continuous random variable in the same manner to the discrete case. However, having known that the range of X is bounded, it can be divided into deliberate number of bins (intervals), each, having a specific length. Suppose $(a,b]$ is divided into M_x bins labelled in the following fashion:

$$1 = (a = r_0, r_1], \dots, M_X = (r_{M_{X-1}}, r_{M_X} = b] \tag{2.32}$$

In general, for each $i \in I = \{1, \dots, M_X\}$ an interval is defined as follow:

$$M_i = (r_{i-1}, r_i]$$

Suppose n_i is the number of elements in R_X mapped by X into the interval i. More mathematically:

$$n_i = |\Theta_i|$$

where

$$\Theta_i = |\{x_k \in R_X \mid r_{i-1} < x_k \leqslant r_i\}| \tag{2.33}$$

Hence:

$$R_X = \bigcup_{i=1}^{M_X} \Theta_i$$

Note that, since the bins have no intersection, for all distinct indices h and l in I:

$$\Theta_h \cap \Theta_l = \varnothing$$

Therefore, each $x_k \in R_X$ can only belong to a single Θ_i. The probability that X gets the value x_k, $p_k = p(X = x_k)$, can be estimated by the amount of concentration of the points (outcomes of X) around x_k. The more points are around x_k, the more likely is for X to get the value x_k in the range. If $x_k \in \Theta_i$, then the interval $i = (r_{i-1}, r_i]$ can be chosen as the area around which the concentration of the points are measured. Subsequently, the number of points, n_i, falling in the interval i, gives an estimation of the concentration of points around x_k. In fact, the bigger is n_i, the more probable is that the event $X = x_k$ occurs. So, we may define:

$$q_i = \frac{n_i}{N} \tag{2.34}$$

It must be considered that, q_i is zero if and only if for each $r \in M_i$ the vent $X = r$ is empty; more mathematically:

$$q_i = 0 \iff \forall r \quad (r_{i-1} < r \leqslant r_i): \quad \{s \in S_X \mid X(s) = r\} = \varnothing$$

In fact, q_i, is the probability that the random variable X maps an event into the interval $(r_{i-1}, r_i]$. Therefore, q_i can be used as a reasonable and a practically computational estimator of p_k, being usually an unknown probability. In other words:

$$\begin{cases} p(X = x_k) = p_k \approx q_i \\ x_k \in \Theta_i \end{cases} \tag{2.35}$$

Therefore, an estimation of entropy in the continuous case [53] is given as:

$$H_{est}(X) = -\sum_{i \in I} q_i \ln(q_i); \qquad I = \{1, \ldots, M_X\} \tag{2.36}$$

Consider Y to be another continuous random variable with a finite size range:

$$R_Y = \{y_1, \ldots, y_N\}$$

Then, with a similar argument for the joint probability function of X and Y, it can be shown that:

$$\begin{cases} p(X = x_k, Y = y_{k'}) \approx p_{ij} \approx q_{ij} \\ \begin{cases} x_k \in \Theta_i \\ y_{k'} \in \Theta_j \end{cases} \end{cases} \tag{2.37}$$

where:

$$q_{ij} = \frac{n_{ij}}{N} = \frac{|\Theta_{ij}|}{N} \tag{2.38}$$

and:

$$\Theta_{ij} = \Theta_i \cap \Theta_j$$

Therefore, the estimated joint entropy of X and Y is given as:

$$\begin{cases} H_{est}(X,Y) = -\sum_{i\in I}\sum_{j\in J} q_{ij}\ln(q_{ij}) \\ \begin{cases} I = 1,\dots,M_X \\ J = 1,\dots,M_Y \end{cases} \end{cases} \tag{2.39}$$

2.4.3 Histogram Accuracy

Equation 2.34, derived earlier, gives an estimation of the probability $p(X = x_k) = p_k$. Note that, to derive the formula it was assumed that $x_k \in \Theta_i$. However, prior to binning the interval $(a, b]$, it is not obvious which elements of R_X falls in Θ_i. So, suppose p_i is the probability that x_k falls in the interval i. Then, n_i would be a binomial random variable with size N and the probability p_i; i.e. for each $k \in \{1,\dots,N\}$ the chance of success that x_k falls in the interval i is p_i, and $1 - p_i$ otherwise. As noted earlier, n_i counts the number of elements falling in the interval i for any subset of R_X. Therefore:

$$P(n_i = k) = p(\{\Theta_i \mid \quad |\Theta_i| = k\}) = \binom{N}{k}(p_i)^k(1-p_i)^{n-k} \tag{2.40}$$

Since $i \in \{1,\dots,M_X\}$, there exist M_X of n_i binomial distributions. From the properties of the binomial distributions for expected value and variance we have:

$$E(n_i) = Np_i \tag{2.41}$$

$$Var(n_i) = Np_i(1-p_i) \tag{2.42}$$

Since $var(n_i) = E({n_i}^2) - (E(n_i))^2$, it follows that:

$$E({n_i}^2) = Np_i(1-p_i) - N^2{p_i}^2 \tag{2.43}$$

Suppose ε_i is the ratio of the error over $p_i \neq 0$, i.e.:

$$\varepsilon_i = \frac{q_i - p_i}{p_i} = \frac{n_i}{Np_i} - 1 \tag{2.44}$$

Then, $\ln(q_i)$ can be expanded to the second order by the Taylor series around the true point $p_i \neq 0$:

$$\ln(q_i) \approx \ln(p_i) + \frac{1}{p_i}(q_i - p_i) - \frac{1}{2{p_i}^2}(q_i-p_i)^2 = \ln(p_i) + \varepsilon_i - \frac{1}{2}{\varepsilon_i}^2 \tag{2.45}$$

Hence, by obtaining q_i from 2.44 and substituting $\ln(q_i)$ from 2.45:

$$\begin{aligned} H_{est}(X) = \sum_{i\in I^c} q_i\ln(q_i) &\approx -\sum_{i\in I^c}(1+\varepsilon_i)p_i(\ln(p_i) + \varepsilon_i - \frac{1}{2}{\varepsilon_i}^2) \\ &= -\sum_{i\in I^c}[p_i\ln(p_i) + p_i\varepsilon_i(1+\ln(p_i)) + \frac{1}{2}p_i{\varepsilon_i}^2 - \frac{1}{2}p_i{\varepsilon_i}^3] \end{aligned} \tag{2.46}$$

where

$$I^c = \{i \in I \mid p_i \neq 0\}$$

The true entropy of the system, which is a part of Equation 2.46 is shown as:

$$H_\infty(X) = -\sum_{i \in I^c} p_i \ln(p_i) \tag{2.47}$$

It is worth noting that, changing the size or length of the bins results the true entropy to change. Therefore the value of H_∞ depends on how R_X is binned. H_∞ is called the true entropy since it was assumed that for the performed bins the probability that x_k belongs to Θ_i ($k = 1, \dots, N$) is exactly known and is equal to p_i. Therefore:

$$H_{est}(X) \approx H_\infty(X) - \sum_{i \in I^c} [p_i\varepsilon_i(1 + \ln(p_i)) + \frac{1}{2}p_i{\varepsilon_i}^2 - \frac{1}{2}p_i{\varepsilon_i}^3] \tag{2.48}$$

Note that in the above equation p_i is a value and not a random variable; only ε_i is a random variable, changing based on the binning procedure, which in turn is a transformation of the binomial random variable n_i. Therefore, everything except n_i can come out of the expected value:

$$E(H_{est}(X)) = E(H_\infty(X)) - \sum_{i \in I^c} [p_i(1 + \ln(p_i))E(\varepsilon_i) + \frac{1}{2}p_iE({\varepsilon_i}^2) - \frac{1}{2}p_iE({\varepsilon_i}^3) \tag{2.49}$$

Using Equations 2.41, 2.43 and 2.44, the expected value of ε_i is calculated as:

$$\begin{aligned} E(\varepsilon_i) = E(\frac{q_i - p_i}{p_i}) = E(\frac{q_i}{p_i} - 1) = \frac{1}{p_i}E(q_i) - 1 = \frac{1}{p_i}E(\frac{n_i}{N}) - 1 &= \frac{1}{Np_i}E(n_i) - 1 \\ &= \frac{1}{Np_i}Np_i - 1 = 0 \end{aligned} \tag{2.50}$$

Likewise, the expected value of ${\varepsilon_i}^2$ is obtained as:

$$\begin{aligned} E({\varepsilon_i}^2) = E\Big((\frac{q_i}{p_i} - 1)^2\Big) = \frac{1}{p_i^2}E(q_i^2) - \frac{2}{p_i}E(q_i) + 1 &= \frac{1}{p_i^2N^2}E(n_i^2) - \frac{2}{p_iN}E(n_i) + 1 \\ &= \frac{1 - p_i}{p_iN} + 1 - 2 + 1 \\ &= \frac{1 - p_i}{p_iN} \end{aligned} \tag{2.51}$$

If M_X is small then for each p_i, the probability that points fall in the interval i, is high. If, additionally, N is large then, based on Equation 2.44, ε_i is small. Therefore, we may discard the small term $E(\varepsilon_i^3)$ and substitute the values of $E(\varepsilon_i)$ and $E({\varepsilon_i}^2)$, respectively, from Equations 2.50 and 2.51 into Equation 2.49 to obtain:

$$\begin{aligned} E(H_{est}(X)) &\approx H_\infty(X) - \sum_{i \in I^c} \frac{1 - p_i}{2N} \\ &= H_\infty(X) - \frac{\sum_{i \in I^c} 1 - \sum_{i \in I^c} p_i}{2N} \\ &= H_\infty(X) - \frac{M_X^c - 1}{2N} \end{aligned} \tag{2.52}$$

where in that

$$M_X^c = |I^c| = |\{i \in I \mid p_i \neq 0\}|$$

It is worth noting that, in the derivation of Equation 2.52, the distribution of the random variable X has not been used. Therefore, when using the histogram based approach, the error in estimating entropy, to the second order, is independent of the actual distribution of the observed random variable [28].

Equation 2.52 indicates that when N is sufficiently larger than M_X^c, the average of the estimated entropies is deliberately close to the actual entropy.

Similar to the procedure applied to measure the entropy, the MI of two continuous random variables can be estimated using the histogram technique. Suppose X and Y are two continues random variables. According to Equation 2.20 the estimated MI of the two random variables is given as:

$$I_{est}(X,Y) = H_{est}(X) + H_{est}(Y) - H_{est}(X,Y) \tag{2.53}$$

Therefore:

$$E\big(I_{est}(X,Y)\big) = E\big(H_{est}(X)\big) + E\big(H_{est}(Y)\big) - E\big(H_{est}(X,Y)\big) \tag{2.54}$$

While estimating the MI, the average error can be derived from Equation 2.54 using Equation 2.52:

$$E\big(I_{est}(X,Y)\big) = H_\infty(X) - \frac{M_X^c - 1}{2N} + H_\infty(Y) - \frac{M_Y^c - 1}{2N} - H_\infty(X,Y) + \frac{M_{XY}^c - 1}{2N}$$

where:

$$\begin{cases} M_X^c = |I^c| = \big|\{i \in I \mid p_i \neq 0\}\big| \\ M_Y^c = |J^c| = \big|\{j \in J \mid p_j \neq 0\}\big| \\ M_{XY}^c = \big|\{i \in I; j \in J \mid p_{ij} \neq 0\}\big| \end{cases} \tag{2.55}$$

Hence:

$$E\big(I_{est}(X,Y)\big) = I_\infty(X,Y) + \frac{M_{XY}^c - M_X^c - M_Y^c + 1}{2N} \tag{2.56}$$

In the special case that:

$$M_X^c = M_Y^c = M$$

then

$$M_{XY}^c = M \times M = M^2$$

Hence:

$$E\big(I_{est}(X,Y)\big) = I_\infty(X,Y) + \frac{(M-1)^2}{2N} \tag{2.57}$$

Equation 2.57 shows that, assuming M to be fixed, when $N \to \infty$ the estimated MI, on average, deliberately approximates the true MI.

2.5 Basics of MINES

This section explains the fundamental motivations of MINES and the integrating components implemented in the final system. In Section 2.5.1 some of the most important motivators of MINES are introduced. Section 2.5.2 deals with the rationales and logics of the incorporation of the sub-systems combined in MINES.

2.5.1 MINES Motivators

The MI between a hidden node's output and the residual error introduced in MINES is in effect a measure of the error prediction by the network's components (i.e. the hidden nodes). Since MI is based on the conditional probability, it becomes clear that why a high MI detected by a hidden node acts as an error predictor. Prior to MINES, MILSC has adopted a similar technique of aligning rules and respective errors by adopting MI as a stronger "second order statistic" than correlation [59]. MILCS is, in fact, the most important motivator of the implementation of MINES. MINES and MILCS have a sensible analogy. They both share the idea of using information theory between an evolving structure and the residual error; both techniques are based on specializing a new component (hidden nodes in MINES and rules in MILCS) to detect the system's error as much as possible by increasing the respective MIs. They both use GA to find the fittest member of the population. Supervised learning in both systems makes training possible.

MILCS, itself, has inspired from XCS and CCNN. XCS proposes a new technique central to focussing on the prediction of the error rather than the error itself. The use of MI as an error detector in MILCS makes the system work based on the prediction of the error. CCNN, on the other hand, proposes the idea of aligning a hidden node's output with the network's residual error using the Pearson's correlation. In a similar concept, MILCS aligns a new rule to the residual error, but with the aid of MI rather than Pearson's correlation.

MINES also takes the idea of aligning a new hidden node with the residual error from CCNN. However, there are key differences between the two system. First, CCNN focuses on a single hidden node at a time added to a new hidden layer creating a cascaded network; while MINES works on the configuration of several hidden nodes simultaneously. Second, CCNN is based on Pearson correlation, which is usually feeble in capturing the co-movements of two random variables. In a different manner, MINES is central to MI, which is a stronger measurement of the interrelationships of two random variables (see Section 2.3). Third, in CCNN, the calculation of covariance in Equation 2.11 is to some extent biased. This is because, for each i, the term E_i already exists in $\bar{E}$; this will influence some self-correlation in the system[4]. Adversely, the methodology in MINES does not deal with any self-correlation since only the MI between the output of a hidden node and the respective residual error is considered.

Regardless of the above mentioned dissimilarities, MINES and CCNN follow different attitudes to eliminate the existing error of the system. CCNN aggressively reduce the network's error [74] by exploiting the addition of hidden-layer-nodes in a greedy fashion. On the contrary, the hidden nodes in MINES are designed so that they collectively reduce the error in each stage. This is because, MINES is focused on the creation of one hidden layer at a time, but configuring several hidden nodes.

It is additionally worth noting that, the concentration of the CCNN on a newly incorporated hidden node is expected to cause system to have a tendency to overfitting [8]. In addition, the several cascaded hidden layers, which are usually fully connected, makes CCNN complex in structure, resulting

[4]It is worth considering that, Equation 2.11 tries to reduce the self correlation by subtracting E_i from the average error. However, this does not seem to entirely work as still the term E_i would exist in the formula.

in long propagation delays [48]. In contrast, MINES uses pruning techniques to restrict dealing with a necessarily fully connected network.

2.5.2 MINES Constituents

As mentioned in Section 1.4, MINES is a combination of a FFNN, GA, BP, MI and clustering. The main objective of the combined components is to construct an optimal hidden layer. Since the performance of a hidden node depends on its connectivity pattern and synaptic weights, structural and parameter learning must be carried out in parallel. For this purpose, GA in MINES designs the structure, while BP trains the synaptic weights. MI, on the other hand, conducts and links the two practice. As upcoming literature shows, the idea of adopting the two practice concurrently has been exploited in various systems and in different fashions. However, the notable difference is that the MI in MINES indirectly associates GA to BP through the fitness of GA individuals: a high MI indicates that the respective hidden node provides more information from the error space due to a collectively strong connectivity pattern and synaptic weights.

Section 2.5.1 addressed some of the reasons of employing MI in MINES over other statistics (like Pearson's correlation). However, further explanations against the advantage of using MI is explicitly provided in Section 2.3.

The subject of using a three layer FFNN in MINES has already justified in Chapter 1 and Section 2.2.2.1 with reference to the "universal approximation theorem".

With respect to the rationale of exploiting GA for the structural learning in MINES, literature (see Section 3.2) suggests that GA is a widely used and standard tool coping well with combinatorial search problems; alongside an admiring ability in working simultaneously on different types of variables. Hence, it is a powerful candidate to integrate into a NN with the intention of appropriately configuring hidden nodes and connectivity patterns. It is additionally worth considering that, for continuous random variables, the calculation of MI may never happen to be exact. Nevertheless, the estimated error is less volatile when an appropriate histogram approximation is used over long run (see Section 2.4.3). In this sense, the GA is even more advantageous to be used as it copes usually well with noisy systems [15]; suggesting that, a few plausible miscalculations of MIs do not usually result in a lethal fail of the entire learning procedure.

Concerning the motives of utilizing BP for the parameter learning in MINES, literature stresses that the ability of GA in structural learning does not necessarily confirms a robust capability in parameter learning. Therefore, BP, as a computationally efficient technique for computing the gradients of the cost function, expressed based on the existing synaptic weights [26], is best to be used alongside GA.

Respecting the clustering of the hidden nodes in MINES, Section 4.5.1 prepares solid mathematical arguments indicating the rationale of using the existing angle between hidden nodes as a similarity measurement. Clustering in MINES aims to result a lighter NN by eliminating unnecessary hidden nodes.

2.6 Summary

In this background chapter, the most fundamental aspects of NN training were covered. The parameter and structural learning were addressed as the two main divisions of NN training. It was expressed that the two practice have a strong interactional effects on each other.

Within the parameter learning, BP algorithm is the most important and widely used technique. The BP formula was mathematically presented in detail (the derived formula is used in Section 4.2 when implementing a FFNN).

Structural learning as another important practice of NN training was subsequently covered. Some of the most important and common techniques (node-redundancy; pruning; weight-sharing and CCNN) were explained in this relation. Meanwhile, it was intended to explicate the fundamental reasons which make the structural learning to have a vital role on the performance of a NN.

Evolutionary algorithms and LCSs are addressed as alternative techniques which can be used in structural learning. The reader needs to refer to Chapter 3 for an overview of the selected applications of these techniques within NNs.

MI, a competitor of the Pearson's correlation, has been introduced subsequently. It has been explained and demonstrated by an example that correlation is only a good measurement of linear dependencies, whereas MI could capture any linear or nonlinear co-movement with the aid of conditional probability[5].

An important feature of MI is reflected in Equation 2.16 introducing the invariant property of MI with respect to the change of coordinates. This feature has been used in calculation of MI in MINES (explained in detail in Section 4.3.1), when obtaining the fitness value of GA individuals by linearly transforming the underlying data to a fixed interval.

The definition of the entropy and MI, and their mathematical relations, were derived for discrete random variables. Based on the discrete case, the histogram technique was introduced for the approximation of entropy and MI in continuous case. For finite size data, the approximation is never exact. To this sense, Equation 2.57 indicates that the smaller the number of bins in comparison to the size of the data set, the better the approximation over long run of measurements.

Finally, MILCS, XCS and CCNN were introduced as the most direct motivators of MINES. The rationale of incorporating the systems implementing MINES (i.e. FFNN, GA, BP, MI and the clustering system) were discussed afterwards.

[5]A classical example of the application of Pearson's correlation in NN structural learning is CCNN (see Section 2.2.2.4). In contrast to CCNN, MINES introduces a novel system of applying MI in structural learning.

Chapter 3

Literature Review

In this chapter literature which is conceptually related to MINES is covered extensively. The following works are sectioned into four categories and are surveyed in detail in each section. The underlying classification is just to clarify the primarily reasons that a particular literature has been taken into account; otherwise, there are a lot of intersections between the categories. The drawbacks and limitations of each work have been assessed and addressed in the context of each part.

The surveyed literature is categorised into the following groups accordingly:

1. literature applying information based systems in the field of NNs ([77][57][82])

2. literature dealing with the sole application of GA within NNs for both parameter and structural learning ([42][43][63])

3. literature emphasizing the hybridization of GA and BP ([3][23][1])

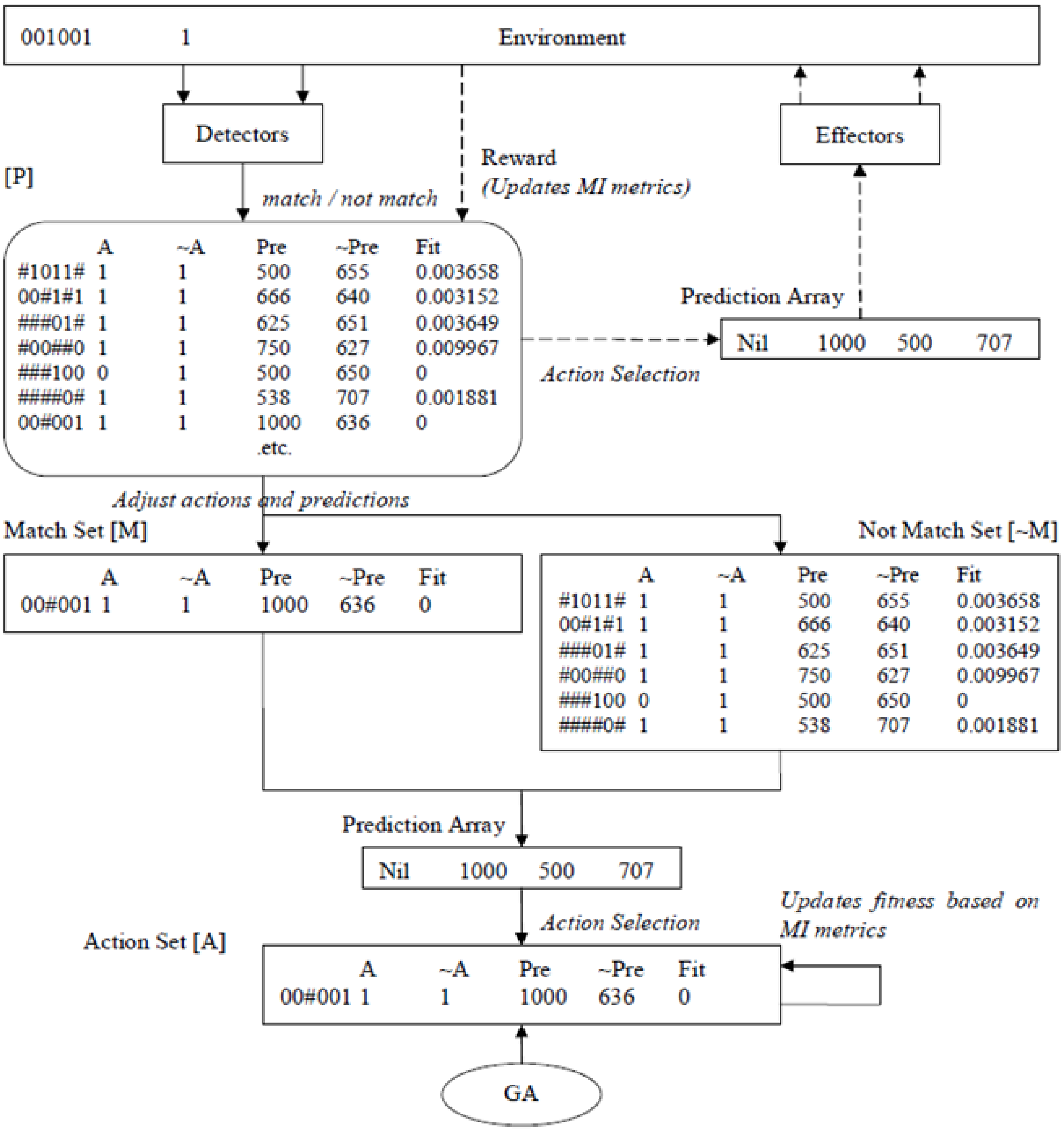
4. literature surveying structural learning by emphasizing pruning, weight-sharing and Neurogenesis ([5][56][4])

3.1 Information Based Systems

As stated in Section 1.1, MINES is a FFNN combined with BP, GA and clustering to exploit MI as an error detector. In this respect, MI has a central role. In this section, related literature ([77][57][82]) has been provided to partly survey the application of MI and, in general, the information theory within NNs. The literature mainly aims to demonstrate the nature of applying information theory to NN training. It also present the capability of MI in measuring the interrelationship of two random variables. As an alternative tool, the work presented by Fogel [17], representing an information criterion (AIC) to select the optimal NN, is also explored. The demonstrated mathematical foundation of his work, suffering from considerable inherent limitations, reveals the capabilities of MINES using information theory.

3.1.1 MILCS

MILCS [62] presents a novel approach to ”design representations and operators for the LCS”. It is inspired by CCNN while utilizing MI as fitness feedback. Unlike most traditional LCSs, originating from reinforcement learning, MILCS is specifically designed for supervised learning. In a conventional



[P]

	A	~A	Pre	~Pre	Fit
#1011#	1	1	500	655	0.003658
00#1#1	1	1	666	640	0.003152
###01#	1	1	625	651	0.003649
#00##0	1	1	750	627	0.009967
###100	0	1	500	650	0
####0#	1	1	538	707	0.001881
00#001	1	1	1000	636	0
			.etc.		

Prediction Array

Nil	1000	500	707

Match Set [M]

	A	~A	Pre	~Pre	Fit
00#001	1	1	1000	636	0

Not Match Set [~M]

	A	~A	Pre	~Pre	Fit
#1011#	1	1	500	655	0.003658
00#1#1	1	1	666	640	0.003152
###01#	1	1	625	651	0.003649
#00##0	1	1	750	627	0.009967
###100	0	1	500	650	0
####0#	1	1	538	707	0.001881

Prediction Array

Nil	1000	500	707

Action Set [A]

	A	~A	Pre	~Pre	Fit
00#001	1	1	1000	636	0

Figure 3.1: Work Cycle of MILCS [62]

LCS, each classifier has a condition and a respective action which is fired once the condition is matched. In the process of developing a new node in the conventional CCNN, only the magnitude, and not the sign, of the correlation with the error is used. Hence, in the presence of a positive correlation with the error, a negative weight is formed to that node; and vice versa. As MI is known to be non-negative, rules, like nodes, also need to have a positive and a negative output. Therefore, in MILCS two actions are given to each rule to conform to the CCNN analogy: one for when the rule matches, and one for when the rule does not match. Both are updated via simple supervised learning. Similarly, there also exist two predictions for each corresponding action.

In MILCS, features like rule representations, population sets, matching algorithm, conflict resolution on action selection, subsumption and GA operators, have a similar resemblance of XCS. Each rule is formed by a condition string and an action string. The condition string is formed by alphabet 0, 1, #. The ”#” symbol indicates ”dont know” so it could be either 0 or 1. The action string is the associated class.

The MILCS learning process (see Figure 3.1) starts by initializing the population [P]. A classifier will be then removed from [p] to form [Ṕ]. The condition of each rule in [Ṕ] is then compared with the detector string. If the bit at every non-# position of a rule matches the corresponding bit of the detector string, it is placed in match set [Ḿ]. Otherwise, it is placed in not match set [˜Ḿ]. For each action in [Ḿ] and [˜Ḿ] a prediction is formed to calculate the system’s output. Since each MILCS rule has two actions, two predictions exist for the matched action and also two for the not matched action. In a different approach from XCS, using a fitness-weighted average of the predictions of classifiers, MILCS simply selects the highest prediction of the classifiers. The maximum-reward-prediction action is always selected in MILCS. MILCS adopts a non-panmictic GA to update rules in [A]. Moreover, inspired from CCNN, each rule is temporarily removed from the system before being updated; the rule will be then back again in to the system.

3.1.2 Yao and Liu (2004)

The research performed by Yao and Liu [77] utilizes the MI for communicating between NNs forming an ensemble. The study indicates that MI can be effectively used to eliminate those members of ensemble having similar functionality. In this concept, the research shares the same outlook as MINES: MI can be used to measure the similarities of two random variables. In MINES the random variables are the hidden nodes and the residual errors; whereas, here, the NNs of the ensemble, themselves, considered as the random variables.

In the system presented by Yao and Liu, a set of NNs is trained (instead of using a single one) to subdivide a given task and solve it in a more efficient way. However, the system is designed so that the elements of the ensemble (i.e. NNs) with low pair-wise MIs become dominated in the population. The MI is measured between the final output of the ensemble and the outputs of each individual NN. A GA is set to evolve the ensemble based on the negative correlation learning between individual NNs. The GA fitness is defined as the pair-wise MI of NNs. The fitness needs to be minimized to achieve a negatively correlated population. Negative correlation learning makes partial learning of training data possible for each NN. The standard BP algorithm [54] has been used for weight adjustments in sequential mode.

3.1.3 Error Entropy Minimization

An alternative application of information theory, with regard to NN learning, is the *error entropy minimization* (EEM) algorithm [57]. EEM concentrates on the minimization of the error between the output of a NN and the desired targets. A BP formula was derived from the Renyi’s Quadratic Entropy for weight updating. The work, however, does not cover any structural learning technique. In addition, the presented formula is computationally complex - due to the existence of a term with an order of N^2, where N is the dimension of the input space. The quadratic term imposes limitations on the number of samples in order to obtain results in a reasonable time.

3.1.4 Zhang and Hu (2005)

The application of information theory in connecting input and output spaces is a standard technique explained explicitly by Haykin (2009) [26]. Nevertheless, to further emphasize the importance of using MI as a strong tool to measure correlation, the work performed by Zhang and Hu (2005) [82] is of interest. It is worth noting that the work does not deal with structural learning in any sense. The research examines the application of a NN for feature selection using a GA combined with MI. The MI between input data and output of the network is used as a criterion for the mutation of the GA. The tolerance distance between the two standard deviations is divided in three. Each gene in the GA is represented in a binary chromosome so that its bit value is determined by the location where the MI of its associated input-output-pair lies in any of those three categories. The motivation behind this is to include patterns highly correlated to outputs into the input feature subset, and vice versa. After selecting an optimal input subset, the network having a fixed architecture is trained by BP using pre-set parameters (such as learning rate and momentum constant). It was claimed that the proposed method could reduce the dimensionality of inputs accompanied by speeding up NN training and achieving better performance.

3.1.5 Akaike Information Criterion Neural Network

Fogel (1991) [17] adopted *Akaike information criterion* (AIC) to find a maximum likelihood for the selection of an optimal NN, named AICNN[1]. AIC is a statistical measure of the goodness of fit defining the likelihood of two density functions. The research has been presented to draw a distinction between the applications of information theory within NNs and those implementing only statistical models.

The following quantity:

$$I\big(g(x), f(x)\big) = \int g(x) \ln\left(\frac{g(x)}{f(x)}\right) dx \tag{3.1}$$

is described as “the mean information for discrimination between $g(x)$ and $f(x)$”. The smaller the value of $I\big(g(x), f(x)\big)$, the more resemble is $f(x)$ to $g(x)$. Therefore, minimizing $I\big(g(x), f(x)\big)$, with respect to $f(x)$, leads to a good approximation for $g(x)$. To this respect, the following equation is defined:

$$S\big(g(x), f(x)\big) = \int g(x) \ln\Big(f(x)\Big)\, dx \tag{3.2}$$

which makes rewrite Equation 3.1 to:

$$I\big(g(x), f(x)\big) = S\big(g(x), g(x)\big) - S\big(g(x), f(x)\big) \tag{3.3}$$

Equation 3.3 indicates that in order to find an $f(x)$, which finely approximates $g(x)$ it is sufficient to maximize $S\big(g(x), f(x)\big)$. Maximization of $S\big(g(x), f(x)\big)$ leads to minimization of $I\big(g(x), f(x)\big)$. The best approximation happens when $I\big(g(x), f(x)\big) = 0$, or equivalently when $g(x) = f(x)$.

In order to estimate $S\big(g(x), f(x)\big)$ it is required to define the mean-log-likelihood. Suppose $g(x)$ is the true density function of a random variable x; and $f(x_i \mid \theta)$ is the parametric density function of the random variable x_i, which x_i is the $i_t h$ observation of random variable x. Suppose n observation of the

[1]This name has not been given to the proposed system by the author himself, but the AIC acronym has been used alone.

random variable x is available; then the mean-log-likelihood is defined as:

$$1/n \sum_{i=n}^{n} \ln \left(f(x_i \mid \theta)\right) \tag{3.4}$$

When $n \to \infty$, the mean-log-likelihood tends to $S\big(g(x), f(x)\big)$. Hence, the mean-log-likelihood approximates $S\big(g(x), f(x \mid \theta)\big)$. Therefore, to best approximate $g(x)$, it is sufficient to maximize the mean-log-likelihood. The maximum likelihood estimation, $MLE_{\hat{\theta}}$, is the sample statistic that maximizes the mean-log-likelihood. To also include the number of free parameters, the AIC is defined as:

$$AIC_{\hat{\theta}} = -2\ln(MLE_{\hat{\theta}}) + 2p \tag{3.5}$$

where p is the number of independently adjusted parameters to obtain $\hat{\theta}$. Minimization of $AIC_{\hat{\theta}}$ acts as a criteria to select the best model that approximated $g(x)$.

To apply the AIC to select the best NN, a three layer FFNN equipped with logistic activation function having one final binary ($\{0, 1\}$) output was used. The induced local field of the final output node of the NN was considered as the value of a random variable from the input patterns to $\{0, 1\}$. This made possible derive a maximum likelihood criterion based on the mean and variance of the associated random variable of the single final output node. Then, a GA was used to evolve the weights and bias of each corresponding NN. Each member of the population represents a vector of weights and bias translating to a three layer FFNN with the sum of squared error of the network being assigned as the fitness value that should be minimized. In other words, the parameter learning is obtained by GA in a way that each chromosome represents a NN containing its weights and bias values as the genes. When the final evolved network is achieved, the AIC is compared to another NN having a different number of hidden nodes that has gone through the same procedure (for instance, a NN with 10 hidden nodes versus another one having 8). The decision on the optimal NN is then made based on the NN having the smaller value of $AIC_{\hat{\theta}}$.

Although the Fogel's presented work is based on a justifiable mathematical background, it is restricted by some considerable limitations. For instance, the structural learning is limited to the number of hidden nodes. In fact, the structural learning is obtained indirectly by adopting $AIC_{\hat{\theta}}$ as the main criterion to choose the NN which best approximates the optimal one. Moreover, the Gaussian assumption of the induced local fields of the final output node is essential to derive the required statistics. Hence, as Fogel himself asserts in conclusion, the assumption may not hold for small size networks. It is further emphasized by the author that there exist some computational difficulties in calculating the final statistical formula, and the technique is only limited to binary classification applications.

3.2 Neuro-Evolutionary Systems

The ultimate goal of aligning the output of a new hidden node with the residual error is to define a desired structure for the NN at hand. As mentioned earlier (see Section 2.5.2), GA is a proper candidate to be used for NN structuring. Therofre, works focusing on the application of GA in NN learning have been provided. The first literature ([49]) is only focused on the adaptation of GA in CCNN with the intention of training the synaptic weights, while the rest ([42][43][63]) use GA for both structural and parameter learnings in NNs. The literature aims to justify the utilization of GA in MINES with the purpose of searching the hypothesis space being mainly the connectivity patterns, while at a same time indicating the possibility of adopting GA as a system adjusting synaptic weights.

3.2.1 Genetic Cascade Correlation Neural Network

An interesting application of GA to parameter learning problem within the CCNN is the *genetic cascade correlation neural network* (GCCNN) presented by Potter [49]. The GA in this program was, effectively, substituted for the BP algorithm, which was responsible for training the non-frozen synaptic weights. Therefore, the research does not present a new approach to the structural learning algorithm within the conventional CCNN. In this aspect, the literature is totally incomparable to MINES, which utilizes GA primarily for the structural learning. The work, however, is beneficial to explore since, unlike the conventional CCNN, it additionally focuses on the outgoing weights of the hidden nodes. In this concept, the system can be exploited in future extension of MINES by adopting an extra GA to select suitable weights and connections for the outgoing synapses of the hidden nodes (see Section 7.2). Moreover, the research applies the two-spiral problem as the benchmark of evaluating the performance of the system; which similarly has been used in MINES.

GCCNN represents the adaptation of two GA at a same time to train the incorporated synaptic weights: one for the incoming and another for the outgoing weights. It was motivated from the advantage of using a GA over the Quick BP to lower the possibility of a suboptimal local minimum convergence. In addition, GA is able to support applications in which gradient information is not available. In other words, the constraint of differentiability of nodes' functions in BP is not the case for GAs.

In Potter's system, two populations were used. Each individual in the first population represents all incoming weights of a hidden node; i.e. synaptic weights of all input nodes, including bias, and all previously fitted hidden nodes. Individuals in the second population represent the outgoing weights of the output nodes. The fitness used for the first population is the correlation of the output of the hidden nodes with the network's MSE. Individuals in the second population use the MSE of the network as their fitness value. Synaptic weights in both populations are initialized randomly between -1 and 1. However, in each generation the output connections of the hidden nodes are heuristically updated in the range of $(0, -C)$; where C is the covariance of the hidden node and error. The negative sign is used in order to quickly reduce the error in case the hidden node is positively or negatively correlated with the error produced from another hidden node. "Similarly, if the correlation is negative a positive connection weight will best reduce the error". When applied to the two-spiral problem algorithm, GCCNN was claimed to require fewer number of training epochs compared to the original CCNN using BP algorithm; though CCNN was able to solve the problem with fewer hidden nodes.

3.2.2 Symbiotic Adaptive Neuro-Evolution

A classic system which simultaneously applies GA on structural and parameter learning is "symbiotic, adaptive neuro-evolution" (SANE), introduced by Moriarty & Miikkulainen [42]. However, SANE only partly addresses structural learning in a fashion that it is restricted to the connections and combinations of a fixed number of hidden nodes: the total number of hidden nodes participating in the NN and the total number of connections made to the input and output layers are all predetermined and fixed during the evolution.

SANE is comprised of a three layer FFNN with each hidden node representing a phenotype encoded in a genome. The goal is to evolve hidden nodes to eventually form a complete NN. This was a different system from most of the NE systems where each phenotype represents a complete NN evolving independently; the population eventually converges towards the fittest network. An advantage of the new system is to deal with a diverse population; none of the single hidden nodes can perform individually well and overcome the others. Each genome of a hidden node is a bitwise chromosome consisting of two parts: the first part allocates the connections; the second part assigns weights to the connections. Connections are between hidden nodes and the nodes in the input and output layers. When the connection-gene has a value less than 127, then the connection is made between that hidden node and an input-node. The associated input-node is the one having a label equal to the value of the connection-gene mode total number of input-nodes. In other hand, if the value of the connection-gene is greater than 127, the connection is made between the hidden node and an output-node having a label equal to the mode of the gene's value and total number of output-nodes. It is worth noting that although the state of the hidden-nodes' connections change, the total number of connections made to the input and output layers are predetermined and fixed for all hidden nodes during the evolutionary process. The total number of connections, therefore, must be determined before running the system. It is additionally worth mentioning that the total number of hidden nodes participating in the NN is also initially determined and fixed during the evolution.

SANE algorithm starts with a random population of individuals each representing a hidden node. A set of individuals, forming the hidden layer, is randomly selected from the population. The size of the selected set is equal to the predetermined number of network's hidden nodes. The network's performance is then evaluated (the network's fitness) and the evaluation is added to the fitness of all hidden nodes participating in the network. By repeating the process, each individual will participate in a network having a different topology and weights. When each hidden node has participated in sufficient number of networks, the process stops. In that stage, each individual's fitness is equal to the sum of all networks' fitness (evaluation score) that the hidden node has participated in. A new fitness (average fitness) is then substituted to the former by dividing the former by the total number of networks which the associated hidden node has contributed to. A high average fitness indicates that the related hidden node has cooperated well in association with other individuals in the population. A one point cross-over takes place to create two offspring by selecting two parents for mating based directly on their fitness (selection by rank).

SANE was applied to the pole-balancing problem which intends to balance a pole centred on a chart by moving it the left or right. A FFNN consisting of 5 input nodes including bias, 8 hidden nodes and 2 output nodes was evolved by SANE to solve the underlying task. Each hidden node was set to have a total of 5 connections to the input and output layer. One of the output-nodes is assigned to move the chart to the right while the other to the left. The unit with the greatest magnitude determines which action (right or left) must be chosen. This method differs from similar systems where the action of an output is interpreted as a probability of performing the action [69]. The reason for using an absolute two-way decision system is due to the SANE's strength in sampling lots of points in the problem space discarding uncertainty in making a decision.

As noted earlier, SANE does not use BP, but weights are directly changed by the GA. The avoidance of using BP, together with the weakness of partial structural learning, makes SANE inappropriate for problems requiring exact solutions [43]. The rather complicated technique used in SANE (and in its successor: the Hierarchical SANE) signifies the robustness technique used in MINES centred on applying information theory for structural learning.

3.2.3 Hierarchical Symbiotic Adaptive Neuro-Evolution

Hierarchical SANE - a successor of SANE - emphasises on the importance of the concurrent evolution of the hidden nodes within the associated network architecture. In SANE the fitness of each hidden node is accumulated by participating in different NNs. The technique results in a weak evolution of hidden nodes, not optimally forming the final NN. To compensate this weakness, Hierarchical SANE deals with two GA at the same time: one for evolving the hidden nodes and the other for converging to an optimal NN. The aim was to improve the deficient structural learning of original SANE by aiding hidden nodes to contribute to each other. Yet, it still suffers from a fixed topology in the number of hidden nodes during the evolutionary process.

Moriarty and Miikkulainen (1998) [43] introduced the "hierarchical evolution of NNs" by addressing the difference between two types of NE, where in one each individual represents a complete NN while in the other just a single node. They pointed out that evolving individual nodes lead to a more efficient generic search resulting to a more diverse population and "faster exploration" of the search space. However, SANE is unable to find a finely tuned NN which makes it inappropriate for applications requiring precise solutions. Instead, it is mainly suitable for problems that can be addressed by lots of solutions. In contrast, evolving a population of NNs might happen slowly and also converge prematurely. Yet, when an area of a good solution is found the final result is usually a desirable NN.

The hierarchical methodology of NE integrates the two types of evolution in a single construction: a network evolutionary system is incorporated on top of the original SANE algorithm. The new technique is named Hierarchical SANE because the networks' evolution comes after the original evolution of the hidden nodes. Hierarchical SANE, thus, deals with two separate sets of populations: the population of hidden nodes, and the population of NNs.

In the original SANE algorithm a predefined number of hidden nodes are chosen randomly to incorporate into a network. The first drawback of this method is the chance of losing a good hidden node due to an inefficient combination of nodes selected by GA. The second problem lies in the inconsistency of the networks' structure, since, in each generation, a variety of different hidden nodes are selected. Even though, this inconsistency is advantageous in early stages of evolution, it turns to be problematic when global optima need to be found at later stages.

Hierarchical SANE improves the problems of random selection of hidden nodes by specifying those collaborating with each other. For this purpose, each individual in the population of NNs is assigned a chromosome of genes addressing the chromosome of a hidden node. Therefore, the number of genes on the chromosome of a NN is equal to the number of connections of the hidden nodes. More precisely, in Hierarchical SANE, the random selection of hidden nodes happening in original SANE has been substituted with a deliberate selection according to the pointers of the networks' chromosomes. Thus, each individual in the population of NNs is pointing to a number of hidden nodes being a fixed predefined number for all networks. During the evolution, the population of the NNs converges to an optimal individual which specifies which hidden nodes are the best to cooperate. The fitness of each individual is the evaluated performance of the associated NN. Since instead of bits the chromosomes are made up of address pointers, crossover only occurs between the pointers by swapping the parents' pointers.

SANE and Hierarchical SANE are, in a sense, similar to MINES as in them each GA individual represents a single hidden node. However, in both systems the total number of hidden nodes and the total number of hidden nodes' synapses (both incomings and outgoings) are fixed and must be defined by the user in advance. The evolution of the system is in effect restricted to the state of the hidden nodes' connections. In MINES, on the other hand, the total number of hidden nodes and incoming synapses and also the state of the incoming connections are all subject to change being automatically found by the system. Another noticeable contrast is the fitness of the individuals which in MINES is the MI between the hidden nodes and the error whilst in SANE and Hierarchical SANE is an average feedback of the performance of the networks in which the particular hidden node has participated.

3.2.4 Neuro-Evolution of Augmenting Topologies

Neuro-evolution of augmenting topologies (NEAT) introduced by Stanley and Miikkulainen [63], is another system that performs structural and parameter learning concurrently using GA. In a similar fashion to SANE, in NEAT, a GA controls and dominates both structural and parameter learning.

Stanley and Miikkulainen [63] highlighted that one of the drawbacks of the most traditional systems using NeuroEvolution (NE) is to usually adopt a fully connected fixed network structure with one hidden layer; and constraining the evolution to just the weights space. It was discussed that the topology of the network, on the other hand, has an undeniable effect on the performance of the system based on following literature: Chen et al. (1993), Zhang and Miikkulainen (1993), and Gruau et al. (1996). The literature disputes that the effect could be positive or negative subject to the type of the employed learning procedure of the weights; which mainly is based on the gradient descent or evolution. In contrary to the literature, Stanley and Miikkulainen argued that "if done right, evolving structure along with connection weights can significantly enhance the performance of NE". They claimed that the evolution of the structure is beneficial in terms of learning speed when the evolved topology minimizes the dimensionality of the weight space. Their presented algorithm is based on a bottom-up evolutionary process so that a minimized topology grows incrementally rather than a decrementing top-down process. NEAT was claimed to be unique in the sense that "structures become increasingly more complex as they become more optimal; strengthening the analogy between GAs and natural evolution".

In NEAT each genome represents the whole structure of a NN. The network's connectivity is coded linearly in each genome. Each node in the NN is represented by a gene called node-gene. The connection of two nodes is expressed by a connection-gene. Each connection-gene carries complete information of the state of the two connected nodes: the in-node, the out-node, the connection weight, connectivity status (enable bit), and an innovation number making possible find relevant genes. A genome, therefore, has the list of all node-genes and connection-genes.

Mutation, which expands the size of the genome, happens in two ways: add-connection; add-node. Add-connection mutation reconnects to previously connected nodes by an extra edge having a random weight. In add-node mutation, an existing connection is split by the new mutated node. The previous connection is disabled and the incoming and outgoing weights of the new node are set to 1 and to the pre-existing weight, respectively. Inducing nonlinearity to the network, the add-node mutation gives GA a more relaxed time to evolve the network's topology as it has a balancing effect on the add-connection mutation by being absorbed quickly into the network.

In order to find suitable genomes for crossover, NEAT presents a way to overcome the competing conventions problem. Competing conventions arises when whole or parts of genomes refer to an identical structure but they have different encodings. In such case, the conventional crossover might result a chromosome missed some of its parents' vital information as the result of their encodings' symmetry. Whenever the size of the genome is increased by a newly created gene, NEAT makes a record of that gene by assigning a global innovation number to it which never changes. The process, called historical markings, makes tracking genes with the same historical ancestors possible. In fact, two genes in two different chromosomes have a same global innovation number if they are inherited from a single ancestor (genome) in the past. NEAT compares the genes of two genomes and lines up those having similar innovation number. These are called the matching genes. The new genome is a result of a random selection of the matching genes and a total inheritance from the remaining genes (excess and disjoint) of the fitter parent.

Although mutations and crossovers create a diverse population of genomes representing different structures, problem arises when networks are trained. Simple structured genomes train faster than the complex ones. Moreover, mutations, themselves, can reduce the fitness by bringing nonlinearity to the network (in the add-node mutation) or by adding a suitable weight. It results in a quick drop in some individual's fitness making them disappear sooner than the time needed for optimization. In order to overcome this problem, called protecting innovation through speciation, NEAT clusters population so that members with similar structures are set together. Genomes in a cluster are assumed to be close together because they carry more common genetic information. Fitness sharing, then, applies by assigning the same fitness to all specious in a set.

In NEAT, the difficulty of finding tuned, local optima arising from not using a classical parameter technique, such as BP, is compensated for by adding additional nodes, connections, and structure, thus adding new dimensions to the weight space. However, a combination of BP and GA together could probably accelerate the training procedure.

3.3 Hybridization of Genetic Algorithm and Back-Propagation

Due to the interactional effect of the structural and parameter learning, which is discussed thoroughly in Section 2.2, an optimal NN is more likely to be found when the two procedures are jointly applied. MINES adapt this idea by alternately applying the GA and BP in parallel with the other. In this concern, literature ([3][23][1]) applying a combination of GA and BP for NN learning are reviewed in this section. The literature mainly focuses on the advantage of using GA and BP in partnership; emphasizing on the benefit of the hybridization of the two techniques.

3.3.1 Belew et al. (1990)

The research presented by Belew et al. (1990) [3] is one of the thorough works investigated the combination of GA and BP for parameter learning. The work is entirely valuable as it broadly investigates in detail the ability of GA in conjunction with BP in training a three layered FFNN. The work highlights the strength and limitation of both techniques. We, specially, present the work in detail since the GA is in central importance in the way structural learning is functioned in MINES.

Belew et al. (1990) presented the idea of combining GA with BP in a way that GA was used to find the initial weights of a NN by proceeding a global search over the weight space while BP was, subsequently, applied to find the optimum weights locally. In other words, GA was responsible to find "good initial weights" in order for the BP to "converge on a solution". It was asserted that gradient descent procedures are likely to trap in a local minimum therefore, sampling methods, such as GA, helps the system to search over a broader domain.

The work starts with a brief explanation of weight encoding in GA. Each weight is assumed to get a real value in the interval [m, M]. By allocating B bits to each weight the interval is divided into 2B subintervals. This means that each weight can vary over 2B values. In order to select a real value from the original interval a uniform distributed random variable over the subintervals is used. In order to encode a three layer FFNN binary-wiring-diagram was suggested. If the network consists of I input-nodes, 1 bias, H hidden nodes, and O outputn nodes, then $H \times (I + 1 + O) + O$ binary strings are needed to cover all links. Each individual in the GA represents a single NN which goes through evolution by allocating the respective MSE as its fitness value.

The first experiment followed by setting up weights from the interval $[-105, +105]$ by allocating 8 bits to each weight meaning that the length of each subinterval was about 800; which corresponds to a poor accuracy. Then, the GA was assigned to create a population of initial weights while the Conjugate Gradient (CG) algorithm was used afterwards to train the system. The obtained MSEs were assigned as the individuals' fitness fed to the GA to create new initial weights and repeat the cycle. The average performance of CG combined with GA (CG+GA) was compared with: the average performance of CG when used alone over several runs (CG_avg); the performance of CG having the lowest MSE (CG_min); and finally the average performance of GA when used alone for several runs (GA_avg). In all cases the CG+GA showed a better performance after the first 30 generations while it converged to a significantly better result at the end of 60 generations.

One important result from this experiment was the observation of large weights (usually larger than

104) in the trained network using CG+GA. An explanation for this effect is the uniform division of the weights' intervals. This makes big numbers to be more available for the GA than the small ones. Moreover, when the magnitude of a weight exceeds a certain threshold, the derivative of the node-function will lead close to zero making BP incapable of changing the weights. Good initial weights, consequently, are located in a donut shape as both very small and large weights are undesirable. Small weights are not suitable since the hidden nodes will produce similar outputs preventing the NN to have a good diversity. In the other hand, large weights should be avoided as it makes BP considerably less sensitive to the error space.

In the next experiment, selection of weights was restricted to the interval $[-\frac{1}{2}, \frac{1}{2}]$. Again the GA selected initial weights while BP used to train networks. The MSE of each network was assigned as the individual's fitness and the cycle repeated for 400 epochs. The results were compared to when GA and BP were used alone for 5000 random restart experiments. When GA was used alone the average of the population's MSE was very close to the average of the MSE over 5000 BP run. Furthermore, the best performance of the 5000 BP run - in terms of minimum MSE - was close to the compound use of GA and BP (GA+BP). However, this does not devalue the use of GA+BP as the compound system was able to find a good solution in a substantial fewer generations.

In the last experiment, a method of exponentially binning the weights was examined. In this case, each weight was binned over 10 bits while the Euclidian length of initial weight vectors were restricted between 0.01 and 12. In contrast to the previous experiments, such a division technique made picking both small weights and large weights possible. The result showed a bigger and faster (in terms of generations) decline in the MSE for the compound use of BP and GA. Also, the exponential selection had a dramatic effect on the best performance of the experiments when BP was used alone.

In conclusion, we point out that, although some of the experiments done by Belew et al. (1990) adopted a joint combination of GA and BP, their systems, in contrary to MINES, does not deal with structural learning; it only deals with fully-connected networks. GA is only used to find the starting points which are suitable for BP to initiate the training from. More importantly, in the system, each individual in the GA's population represents a complete NN assigning the MSE as its fitness, whereas in MINES each individual represents only one hidden node with MI as the fitness.

3.3.2 Han and May (1996)

Han and May (1996) [23] proposed a system employing the GA as the central unit which controls all important BP parameters and also the number of incorporated hidden nodes. The work emphasizes that structural and parameter learning has a huge effect of NN performance. In the presented system the GA was integrated into a three layer NN to find an optimal NN by both changing the structure and learning parameters. The structural learning dealt just with the suitable number of hidden nodes, and not their connectivity pattern of receptive fields. In the case of parameter learning, they tried to find the optimum values involved in BP algorithm: learning rate, momentum constant, and training tolerance. The GA, then, was simultaneously changing these four parameters in order to find the desired performance. The GA objective function was set to be derived regression from the four involved factors. They concluded that since BP algorithm is quite dependent on the four aforementioned parameters, their technique was able to find better weights, and hence better results, through changing them with the aid of GA.

3.3.3 Azzini and Tettamanzi (2006)

Azzini and Tettamanzi (2006) [1] presented a NE system working alongside BP. The idea was to use GA to find a solution near the global optimum while BP was, subsequently, used as a tuning device to locate the closest local minimum. However, the structural learning of the system does not cover the evolution of the receptive fields. In other words, there is no pruning process and the system only deals with fully-connected NNs. The work, however, is presented here, since, in an approach similar to MINES, it utilizes both the GA and BP for structural and parameter learning. In addition, the system is worth considering as it has been tested on a real financial problem.

In the system presented by Azzini and Tettamanzi (2006), each individual in the population (genome) represents a *multi layer perceptron* (MLP) NN. The GA initially finds a solution near the global optimum in the search space where each point represents the topology of a MLP. BP is, subsequently, used as a tuning device to locate the closest local minimum. Each genome has all necessary information needed to create the associated network: number of layers, number of nodes of each layer, and the weight matrix of each layer. In each genome there is a bit to activate or deactivate the BP process. BP, therefore, is not necessarily used in each generation. The problem data is divided into three sets: training set used to train the network, test set used to avoid overfitting, and validation set used to examine network's generalization performance. During evolution, whenever an individual decides to use BP, the training set is used for training while subsequently the MSE over the test set is calculated to evaluate part of the individual's fitness. When BP is not used, the MSE is calculated over the joint training and test sets. The fitness of an individual does not totally depend on the network's MSE. The cost of the network, which is a measurement of the complexity of the network's topology, is also used in the fitness. The overall fitness is formulated as:

$$\begin{cases} f = \lambda kc + (1-\lambda)MSE \\ 0 \leq \lambda \leq 1 \end{cases} \tag{3.6}$$

In formula 3.6, λ is the desired trade-off parameter between the cost and error; k is a constant used for scaling which makes the cost comparable to MSE; c is the cost of the network; and MSE is the network's error calculated as explained earlier. The cost of the network is defined as:

$$c = \alpha(\text{number of hidden nodes}) + \beta(\text{number of synapses})$$

Since the system works on MLP networks, the formulated cost is likely to reasonably measure the complexity of a MLP's structure.

Based on the state of the BP being either off or on in a genome, a direct or an indirect encoding is conveyed to the associated phenotype, respectively. In the underlying research, the selection method of breeder GA (Muhlenbein and Schlierkamp-Voosen, 1993, cited in [1]) was used. The method is based on removing half of the worst members of the population while duplicating the remaining individuals to compensate those eliminated.

Both weights and topology of the networks undergo mutations. Weight mutations happen in each layer by adding a noise proportional to the variance of the layer multiply by the standard normal distribution. Immediately after the weight mutation, nodes with little influence on the network's output are removed based on the following pseudo-code:

```
for $i = 1$ to $l - 1$ do
if $N_i > 1$
    for $j = 1$ to $N_i$ do
    if $\|W_i^j\| < (avg_k(\|W_i^j\|) - r(stdev_k(\|W_i^j\|))$
        delete the $j_{th}$ neuron
```

where N_i is the number of nodes in the i_{th} layer, W_i^j is the j_{th} row of the matrix W^i, and r decides how many standard deviation bellow the layer's average the weight vector's norm must be in order to be deleted. The topology mutation is performed in three ways: insertion of one hidden layer, deletion of a hidden layer, and adding an extra node. The insertion of an extra node is a neutral mutation since the incoming and outgoing weights of the duplicated node are assigned in such a way that the overall network's output does not change.

Being aware of the possible problems that might arise by crossing over the individuals, which each represents a NN, a vertical crossover merging both the weights and the structure of the parents was adopted. Since selected parents might have different number of layers, the parent with the less number of layers undergoes some layer-insertion mutation till both own equal number of layers. The new offspring, afterwards, is generated in a way that the number of nodes in each layer is the sum of the nodes of the parents' respective layers except for the input and output layers being unchanged for all NNs. The weight matrix of the input layer is obtained by appending its respective parents' weight matrices. For the rest of the layers, the weight matrices are the block diagonal matrices of the parents. However, the elements of the matrices have been halved in order to produce an output closed to the parents.

The evolved NN was used as a factor model (statistical model) to examine the return of a statistical arbitrage. Factor models represent the return of a financial instrument as a function of other financial instruments' returns. In the statistical arbitrage, the arbitrageur builds a portfolio of correlated instruments. "When the price of one of the instruments diverges" lower than "the value predicted by the model, the arbitrageur" longs "that instrument and short the others"; the opposite is done when "the price is higher". "If the model is correct, the price will tend to revert to the value predicted by the model, and the arbitrageur will profit".

Azzini and Tettamanzi [1] examined a factor modelling problem of Dow Jones Industrial Average (DJIA) against some foreign exchange rates, stock of companies as a representative of the relevant market segment, and commodities. In order to find the best solutions they used experiments by changing the following settings: probability of inserting a layer, probability of deleting a layer, and probability of inserting a neuron. For each set of settings 10 experiments was run. A noticeable finding was that for all simulations the best result was obtained when BP was adopted. The best obtained MLP showed a satisfactory performance in predicting the values of DJIA compared to the actual values. By comparing these predicted (expected) results to the actual values a strategy of making an arbitrage against the pre-mentioned instruments was proposed. The strategy proved to be fairly successful by making 5.75% annual return on the capital.

To compare the technique presented by the surveyed literature with the one proposed by MINES we proceed as follow. In MINES the interaction between the GA and BP is indirect through MI and clustering of the weights. BP trains the weights which cause the MIs to change. This in turn affects the individual's fitness reflecting the interaction between BP and GA. BP also changes the angles of the weight vectors affecting them to be clustered differently expressing another interaction between GA and BP. On the contrary, in the method introduced by Azzini and Tettamanzi [1], each phenotype represents a complete NN being a direct encoding of its genotype. The interaction between the GA and BP also takes place directly through the network's MSE which partly affects the individual's fitness when BP is chosen for the weight training. Most essentially, in contrast to MINES, the surveyed literature, as mentioned earlier, only deals with a fully connected network. Furthermore, the system incorporates an algorithm for eliminating ineffective hidden nodes. In this respect, the system is similar to MINES in which close hidden nodes are removed. However, in the two systems, the theories of acknowledging the removing hidden nodes are fundamentally different. In one hidden node are eliminated when the norm of its incoming weights as a vector is comparably less than a threshold whereas in the other (MINES) the hidden nodes are removed when the angles of the incoming weight vectors are less than a threshold.

As a downside of the presented technique the use of vertical crossovers can be mentioned. A vertical crossover usually requires some layer-insertion mutation. As a result, an undesirable deep structure might arise resulting in a slow propagation computation. A further drawback is related to the algorithm of eliminating unnecessary weights, which is based on the length of the weight-vectors. This method does not completely help reduce similar vectors. The lengths of two vectors might lie in the required boundary indicating that they should not be eliminated while in reality they may have a close angle producing similar outputs (see Section 4.5 for more details).

3.4 Pruning, Weight-Sharing and Neurogenesis

The three common structural learning techniques (node-redundancy, pruning and weight-sharing), addressed in Section 2.2, have been used in some of the pre-surveyed literature. In fact, most of the literature simultaneously applies a combination of node-redundancy and pruning - two practices that also happens in MINES (SANE [42] and Hierarchical SANE [43] are two classical examples in this relation—). In addition to these, the research provided by Cantú-Paz [5] is presented here to specifically assess the pruning technique. To reveal the essence of weight-sharing, as an alternative structural learning technique, the system proposed by Saito and Nakano (2002) [56] is surveyed in detail. The literature, additionally, reveals the great deal of complication exists in solving the presented regression problem which has also implemented by MINES. A further structural learning technique, based on Neurogenesis, presented by Boers and Kuiper [4], are also surveyed to demonstrate an additional system of structural learning. The research is, in addition, valuable as it applies information theory to examine the structural characteristics of a NN.

3.4.1 Cantú-Paz (2003)

The work investigated by Cantú-Paz (2003) [5] contains a full exploration of the advantage of pruning technique in NNs. In this experiment a network is first trained with BP algorithm to a desired level of accuracy, then pruning is applied with or without retraining. Hence, the method does not give the chance of a going through structural learning in conjunction with the employed parameter learning (BP in this case). In fact, there is no real interaction between the structural learning (pruning in this case) and parameter learning (BP). However, the work can be used as a benchmark to give explanation for the adaptation of the automatic pruning technique used in MINES.

Cantú-Paz [5] examined NN pruning by using four evolutionary algorithms on FFNN with on hidden layer. The main goal of the research is to improve the generalization capability in classification problems. The four employed evolutionary algorithms are: simple GA (sGA); compact GA (cGA); extended compact GA (ecGA); Bayesian Optimization Algorithm (BOA). The last three algorithms are categorized as distribution estimation algorithms (DEAs).

The general algorithm starts by training a fully-connected network using BP for a number of epochs varying for each experiment. One of the above mentioned evolutionary algorithms, afterwards, is applied to prune the network based on the state of a particular bit allocated to each weight. Two different types of strategy can be applied in this stage: retraining the network for one or two or more epochs after each generation to "repair the damage caused by pruning", or allowing the system to go through the evolutionary process without retraining. The fitness of an individual is the accuracy of the associated network over the training data.

The sGA works on binary strings. Binary selection, uniform crossover, and bitwise point mutation are performed by sGA. Since each individual represents a complete network, problem might arise when two identical genomes have different encodings. A simple solution to this problem is to employ an exponentially-sized population.

Each individual in the cGA is a vector $p = (p_1, \ldots, p_k, \ldots, p_l)$ with the length l equal to the number of weights of the representing network. The compact GA assumes that each gene (bit) on the genome (vector), which represents a weight in the phenotype (network), is independent from the others. For each k, P_k is the probability of getting a value 1; or, equivalently, the probability of having the k_th weight connected. The population is, therefore, modelled as a product of Bernoulli distributions. Offspring are obtained by selecting different positions of p and concatenating the obtained values. When the fittest offspring has a 1 in its k position, the parent p_k is increased by $1/n$, where n is the population size. The iteration terminates when all elements of p are either zero or one.

The ecGA focuses on a set (partition) S with the size m which is a subset of $\{p_1, \ldots, p_l\}$. Suppose the number of subsets of S is m and the distribution of variables of the i_th subset is $P - i$. The problem is, in fact, to focus on the subsets of a subset (partition) of $\{p_1, \ldots, p_l\}$. By assuming that the subsets are independent, the population can be modelled as $P = \prod_{i=0}^{m} P_i$ where each P_i represents an individual. The distribution of P_i is distributed in a table with $2^k - 1$ entries where k is the size of the set P_i. The aim is to find a suitable P which can satisfactory model the population. One method is to merge subsets minimizing the complexity of the networks based on information theory. The search ends when there are no more subsets that can be merged.

The BOA uses a Bayesian network to determine the relations among selected individuals. In BOA the maximum number of incoming weights to each node must be stated by the operator.

The final pruned networks obtained by the four explained models were tested on several instances of the UCI machine learning repository data sets. The results suggest that pruning is considerably beneficial when the performance of the fully connected NN is poor. When the fully connected NN performs well pruning does not show a noticeable improvement. Even in two cases the pruned networks carried out by cGA and ecGA resulted significantly worse than the fully connected ones. Among the four tested systems ecGA is the fastest while sGA and BOA are mainly the slowest. However, ecGA occasionally results a poorer network while sGA and BOA do not.

Further experiments carried out by retraining the networks for some epochs (in contrary to the above experiments) after pruning. Retraining does not show to considerably improve the generalization ability of the pruned network. However, it dramatically reduces (up to 40%) the number of evolutionary generations with the cost of increasing the total operating time. Retraining for more than one epoch has no visible effect in improving overfitting or reducing the number of generations. The final conclusion recommends that the best evolutionary method for pruning a NN is sGA over a small population.

3.4.2 Bidirectional Clustering of Weights

The technique presented by Saito and Nakano is based on a very complicated and sophisticated mathematical foundation which presents structural learning solely by adopting weight-sharing. The work is extremely valuable to present as it can be compared to MINES solving the same application using a far simpler technique.

Saito and Nakano (2002) presented a way to automatically create a three layer NN by clustering the weights and weight-sharing. The technique, called *Bidirectional Clustering of Weights* (BCW), employs the second order optimal criteria to merge and split clusters. Bidirectional Clustering starts with a fixed set of weights

$$w = (w_1, \ldots, w_d, \ldots, w_D)^T \tag{3.7}$$

clustered to subsets that have no intersection so that the sum of the subsets is equal to the original set of the weights. In fact, if $S_1, \ldots, S_g, \ldots, S_G$ is a set of indices corresponding to weight clusters, then:

$$\begin{cases} S_g \neq \emptyset \\ S_g \cap S_{\acute{g}} = \emptyset \qquad g \neq \acute{g} \\ S_1 \cup \ldots \cup S_G = \{1, \ldots, D\} \end{cases}$$

A common weight is associated to each cluster so that all the cluster's weights have the same value as the common vector (the concept of weight-sharing). Therefore, a transformational matrix A, having elements of 0 and 1, mathematically, maps the original weight space w in expression 3.7 to the common weight space:

$$u = (u_1, \ldots, u_g, \ldots, u_G)^T$$

so that:

$$w = Au$$

Considering ${e_d}^D$ as a D-dimensional unit vector ($D \times 1$) that its d element is 1 while the rests are 0, then:

$$A = [\sum_{\{d|d \in S_1\}} {e_d}^D, \ldots, \sum_{\{d|d \in S_G\}} {e_d}^D] \tag{3.8}$$

More intuitively, the j_{th} column of A is related to the cluster S_j and row i of column j is equal to 1 if $w_i \in S_j$ and 0 otherwise.

The Bidirectional Clustering problem is to find a suitable set of clusters which minimizes the error function, denoted by $E(Au)$. A successive number of merges and splits of clusters are repeated based on the second order optimal criteria to find the optimal set of clusters escaping local optima. Merging of clusters happens by a one-step bottom-up clustering, being a merger between a pair of clusters reducing the total number of clusters by one. Suppose a pair of clusters S_g and $S_{\acute{g}}$ is merged into a single cluster $S_g \cup S_{\acute{g}}$. Since all the weights in the merged cluster should be equal to its associated common weight it requires that the previous common weights of S_g and $S_{\acute{g}}$ become a new single common weight in the merging process. More precisely we should have:

$$(u + \Delta u)^T {e_g}^G = (u + \Delta u)^T e_{\acute{g}}^G \tag{3.9}$$

In order to find a solution for the merge operation, the Bidirectional Clustering uses the second order Taylor expansion of error function around u:

$$E(A(u + \Delta u)) - E(Au) \approx g(w)^T A \Delta u + 1/2 \Delta u^T A^T H(w) A \Delta u \tag{3.10}$$

Where in that, $g(w)$ and $H(w)$ are the gradient and Hessian matrix of the error function respectively. If u is a trained vector then $A^T g(Au) = 0$. By using the Lagrange multiplier method and using Equation 3.9, the minimum value of expression 3.10 around a trained weight w is:

$$\frac{\left(u_g - u_{\acute{g}}\right)^2}{2\left({e_g}^G - {e_{\acute{g}}}^G\right)^T \left(A^T H(w) A\right)^{-1} \left({e_g}^G - {e_{\acute{g}}}^G\right)} \tag{3.11}$$

Equation 3.11 was called the dissimilarity between S_g and $S_{\acute{g}}$ showing by $DisSim(S_g, S_{\acute{g}})$. In order to find a suitable pair of clusters to merge, the Bidirectional Clustering calculated the dissimilarity of all pair of clusters and then selects the one which has the minimum dissimilarity.

The top-down clustering in contrary to the bottom-up clustering is the operation of splitting a cluster to two new clusters. Considering the total number of clusters to be G, after splitting S_g into two new clusters $\acute{S}_g$ and S_{G+1} (i.e. $S_g = \acute{S}_g \cup S_{G+1}$) the total number of clusters will increase by one. Suppose, before splitting, the network has been trained and u is the trained common weight vector. Immediate after splitting of S_g, the new common weight vector, denoted by v, has the form of:

$$v = \left(u^T, u_g\right)^T$$

The reason is that, prior to splitting, all the weights in the cluster S_g have the common weight u_g. Therefore, just after splitting, weights of the clusters $\acute{S}_g$ and S_{G+1} will again have the common weight

u_g. However, it is worth mentioning that after training, u_g, which is associated to S_{G+1}, will have a different value than those associated to $\acute{S}_g$. The matrix A, in the second order Taylor expansion, is substituted by a new matrix B. B is similar to A except that its column g has been changed according to $S_{\acute{g}}$ and also has a new column at the end at column: $G+1$ related to S_{G+1}:

$$B = [\sum_{\{d|d \in S_1\}} e_d{}^D, \dots, \sum_{\{d|d \in S_{\acute{g}}\}} e_d{}^D, \dots, \sum_{\{d|d \in S_G\}} e_d{}^D, \sum_{\{d|d \in S_{G+1}\}} e_d{}^D]$$

It should also be noted that since v is not a trained weight vector $B^T g(Bv) \neq 0$. Therefore, the following optimally condition on u holds:

$$0 = g(Bv)^T \sum_{\{d|d \in S_g\}} e_d{}^D = g(Bv)^T \Big(\sum_{\{d|d \in \acute{S}_g\}} e_d{}^D + \sum_{\{d|d \in S_{G+1}\}} e_d{}^D \Big) \tag{3.12}$$

From Equation 3.12 we will have:

$$B^T g(Bv) = kf \qquad k = g(Bv)^T \big(\sum_{\{d|d \in S_{G+1}\}} e_d{}^D \big) \qquad f = e_{G+1}^{G+1} - e_d^{G+1} \tag{3.13}$$

By substituting formula from 3.13 in the Taylor expansion, which is similar to Equation 3.10 except having matrix B instead of A, we get:

$$min\Big(g(Bv)^T B\Delta v + 1/2\Delta v^T B^T H(Bv) B\Delta v\Big) = -1/2k^2 f^T \big(B^T H(Bv)B\big)^{-1} f \tag{3.14}$$

Therefore, the general utility of splitting S_g into $S_{\acute{g}}$ and S_{G+1} is defined as:

$$GenUtil\big(S_g, S_{G+1}\big) = -1/2k^2 f^T \big(B^T H(Bv)B\big)^{-1} f$$

which should be maximized. In order to simplifying different ways of splitting, only the splitting of one element has been considered; for a single $d \in S_g$, we have:

$$Util(S_g, \{d\}) = \big(g(Bv)^T e_d{}^D\big)^2 f^T \big(B^T H(Bv)B\big)^{-1} f$$

Bidirectional Clustering has the advantage of escaping local minima if both bottom-up and top-down clustering are used together. A single usage of either of them is likely to stick in a local minimum. The Bidirectional Clustering starts from an initial set of clusters. The clusters, then, go under a number of merges or bottom-up clustering. At this stage, in order to reduce the computations the network does not retrain. The top-down clustering or splitting the clusters happens afterwards while the network is retrained. Again, a bottom-up clustering is performed, while, in contrast to the initial merging step, the network is also retrained. If the overall error of the network is increased, the initial network architecture is chosen as the optimum; otherwise the procedure again starts from the second step.

The final obtained NN resulted from applying BCW was tested on a multivariate polynomial-type function. The following polynomial was used:

$$y = 2 + 3x_1x_2 + 4x_3x_4x_5 \tag{3.15}$$

In the training data each $x_i (i = 1, \dots, 5)$ was randomly created in the $(0, 1)$ interval. For a training pattern $X = (x_1, \dots, x_5)$ the corresponding desired value is y, obtained from Equation 3.15. The size of epoch was 5000. As explained earlier a three layer NN was used for this experiment; the number of suitable hidden nodes was set to 2 acquired by the cross-validation technique. It is worth noting that weight-sharing just applied between the connections of the input-layer and the hidden layer; the common weights are 0 and 1.

BCW, as stated by the authors, starts with plenty number of weights. This may be regarded as a drawback as it may result in a wide NN architecture, producing delay in forward and backward propagation. An additional disadvantage is the use of the Hessian matrix making the system computationally slow.

3.4.3 Boers and Kuiper (1992)

Boers and Kuiper (1992) [4] used Lindenmayer grammar to develop and structure NNs by combining GA and BP. Their work is interesting to explore as it clearly declares a way GA can be used in determining the structure of a FFNN while BP is combined to perform parameter learning. The presented work is also attractive as it illuminates an alternative structural learning technique based on neurogenesis. Besides, the main objective of the work is to demonstrate that "the initial structure given to a NN greatly determines its performance".

Lindenmayer grammar was coded in GA chromosomes forming a population of genotypes which subsequently translated to create the structure of each associated phenotype. BP was utilized as the learning procedure and the resulting error was assigned as the fitness of the GA individuals. In Lindenmayer grammar each node is specified by eight letters from A to H, while the digits 1 to 5 are used to indicate a skip within the string; comma (,) is placed between two letters when there is no connection between corresponding nodes; and finally square brackets ([,]) are employed to specify that two adjoining modules are connected. For instance [A2[B,C]D]E is a FFNN which in that the node A is connected to the both nodes B and C which themselves are connected to the node D which itself is connected to the output node E. Moreover, nodes B and C are not connected (since there is a ',' between them) and the digit 2 after A indicates that the node A is also directly connected to the node E. Since the pre-mentioned rule must be understandable for a GA, Boers and Kuiper employed a table to code each character to a 6-bit string (for instance code 100100 is accounted as the first A in the table). The genetic information then specifies the structure of a FFNN with its weights being trained by BP.

Boers and Kuiper justified that a modular NN can be advantageous over a NN having a single hidden layer. It was claimed that when a network with one hidden layer, containing n hidden nodes, finds a solution, a permutation of the nodes in the hidden layer together with all their incoming and outgoing weights, will also result in the same solution. Therefore, the existence of a solution guarantees the existence of at least $n!$ more solutions. When a network has more hidden layers the number of permutations of the successive layers must be multiplied. The total number of permutations of a network with i hidden layers containing n_i nodes in each layer (ignoring nodes with exactly similar weights, and different solutions with respect to every permutation) is given by:

$$\prod_{i=1}^{n} n_i! \tag{3.16}$$

However, the authors failed to notice that the derived formula is only valid when the trained NN is invariant to the permutation of the incorporated signals of the input pattern vectors. Suppose, the input pattern vector p contains k signals:

$$p = (s_1, \ldots, s_k)$$

Given a permutation:

$$\varphi : \{1, \ldots, k\} \rightarrow \{1, \ldots, k\}$$

the signals can be permuted as follow:

$$p_\varphi = \left(s_{\varphi(1)}, \ldots, s_{\varphi(k)} \right)$$

The required assumption is that any solution of the NN be indifferent to the presentation of the input vectors p_φ and original input vectors p. In other words, the order of the presentation of the input signals should not matter when the network has reached to find a solution. For some theoretical problems concerning a random creation of input patterns, such as the polynomial regression problem introduced in Section 5.1, this assumption may hold true. However, the assumption is not valid as a general rule; for instance, the decomposed signals used as input patterns in the crude oil forecasting problem, introduced in Chapter 6, are not invariant to permutation.

Assuming that the obtained permutation formula for the hidden nodes indicates the number of potential solutions, the entropy of the weight space can be used to demonstrate that a better generalization capability can be achieved when the number of potential solutions is high. In reality, a NN performs an input-output mapping from the input space to the output space. The mapping depends on both architecture and synaptic weights. When the architecture is fixed, the collection of all possible sets of synaptic weights defines the weight space. Each set of synaptic weights declares a probability that determines the possibility of a correct mapping with the employed NN consisting of the given synaptic weights within the fixed architecture. So, a random variable can be defined over the weight space to the set of all possible input output mappings. Consequently, the weight space determines a probability distribution over all potential mappings. Boers and Kuiper continue:

> "The entropy of this distribution is a quantitative measure of the diversity of the mappings realizable by the architecture under consideration. Learning from examples reduces the intrinsic entropy of the untrained network by excluding configurations which realize mappings incompatible with the training set. The residual entropy of the trained network measures its generalization ability."

Therefore, the residual entropy is the entropy of those sets of synaptic weights resulting a desired mapping which is usually different from the mappings defined on the training set.

Zero residual entropy indicates that the employed architecture confines the synaptic weights to some limited configurations within the desirable mappings. The residual entropy will not be zero when the network architecture gives several different mappings. Non-zero residual entropy indicates that small deviations from the training input space leads to large errors in the output space. So the system will be poor in generalization, because a strong generalization requires the network to perform well on the unseen data.

Therefore, an optimal architecture is the one that would result to zero - or close to zero - residual entropy. A non-zero residual entropy can be imagined as "a weight space with a large number of minima, each implementing exactly the mapping of the training set", but only one (or none) representing the actual sought mapping. "It also shows the importance of the selection of the training set". Suppose $W = \{w_1, \ldots, w_n\}$ is an arbitrary set of synaptic weights corresponding to a network architecture at hand. Then:

$$W = \{w \mid \text{w defines the synaptic weights of the network architecture}\}$$

is the collection of all possible sets of synaptic weights w incompatible with the given architecture. The a priori probability of a desired mapping f is defined as (Solla, 1989, cited in [4]):

$$P_f = \frac{\Omega_f}{\Omega_W} \tag{3.17}$$

Where Ω_W is the total volume of the weight space W confined with the network architecture at hand, and:

$$\Omega_f = \int_{\Omega_W} \Theta_f(w)\,\mathrm{d}w \tag{3.18}$$

where:

$$\Omega_f(w) = \begin{cases} 1 & \text{if } w \text{ implements the mapping } f, \\ 0 & \text{otherwise.} \end{cases} \tag{3.19}$$

In fact, Ω_f is the volume in the initial weight space which implements the desired mapping. Likewise, P_f is the probability of a correct implementation of the desired mapping within the possible choices of synaptic weights in W. If $P_f = 0$, the network is unable to learn the desired mapping; hence, $P_f \neq 0$ is a necessary condition for the network to learn. In NN learning, the mapping defined by the training set is usually unique as the learning procedure is applied to a fixed training set consisting of the input and respective desired patterns. In contrast, the generalization ability of the trained NN necessitates performing correct mapping on new data. Entropy can be used to measure the diversity of all possible mappings f within a chosen architecture:

$$S = -\sum_{\{f\}} P_f \ln(P_f)$$

When a suitable initial structure is chosen, P_f is usually large for a given mapping f, resulting in a low entropy over the space of possible mappings. When the network with the given structure is not yet trained with a parameter learning technique, the measured entropy over the space of all possible mappings f is called the intrinsic entropy of the network. "It is the intrinsic entropy of the untrained network that needs to be eliminated via a learning process". The purpose of training is to confine the configuration of weight space to the region $\{w \mid \Theta_f(w) = 1\}$ for all desired mappings.

The presented mathematical formula can be used in the future application of MINES. At the moment synaptic weights are trained unboundedly by just BP algorithm during two successive GA generations.

To realize better which synaptic weights are suitable for the fixed architecture at hand during two successive GA generations, synaptic weights must be confined (digitalized) within some boundaries. This can be implemented by simply picking up weights manually from a pool of binned weights. For example every synaptic weights can be chosen from $[-1, -0.8], [-0.8, -0.6], \ldots, [-0.2, 0] \ldots$, or alternatively, the activation function can be chosen as a stepwise function. Then to find out which weights define a better performance, some mappings from training or validation sets can be chosen. Also mappings can be chosen by eliminating some of the training input patterns to find out which training set is more valid to be chosen. Then P_f is calculated by creating a NN with synaptic weights chosen from the binning intervals. Subsequently, the entropy over selected mappings is calculated. Those synaptic weights result to lower entropy are those which provide the best generalization performance.

3.5 Summary and Discussion

The surveyed literature indicates that in spite of a lot of attempts to define an optimal structure for a NN, there is still no complete, focused work taking into account the relation of hidden layer and the network's error with respect to the quantity of the contributed hidden nodes, their receptive fields and synaptic weights. More importantly, only a few works have paid any attention to the theoretic aspects of this problem. In fact, there exist no system to automatically find the quantity and the structure of the hidden layer based on a trustworthy theory.

Most of the literature only partly answers the structural learning and the only one ([63]) which broadly deals with this matter suffers from discarding a robust parameter learning technique such as BP. In other words, most of the research ([17][49][42][43][63]) does not adopt any gradient descent algorithm; they use GA for both weight training and, if necessary, structural learning. In effect, in these works, the role of NN is demoted to the process of forward-propagations for error calculation while the GA has almost dominated the entire program. This neglects the natural and powerful parameter learning capability of the BP.

Additionally, in most of the works the fitness function of GA is directly defined based on the cost function of the underlying NN - such as MSE. Even though the GA is able to construct a desirable structure in this regard, it is not theoretically clear how the forming structure makes reduce the cost function. In other words, the success of the system is heavily rooted in the stochastic search technique adopted by GA regarded as a black box. *Indeed, there is no lucid explanation for the reason a typical structure can outperform the rest.* The systems do not propose any well organised rule to the individuals forming the structure aiming to reduce the cost function.

In contrast to these systems, MINES provides an indirect interaction between individuals and the NN's cost function by introducing an intermediate variable, acting as the fitness function, communicating between the cost function and the individuals' chromosomes. In some of the surveyed literature, MI has been used successfully in disclosing the existing correlations among series. However, MI has not been utilized to form the structure of a hidden layer with the information gained from the error space. The only exception in this regard is the research conducted by Smith and Jiang [62], MILCS [62]; which, in concept, shares the most similar algorithm to what proposed by MINES (the subject of this research).

The promoted procedure in MILCS, a successor of XCS, is being used similarly in our algorithm, except that in MINES neurons are the main operator of the network which is in analogy to rules in MILCS. It is also worth mentioning that both MILCS and MINES take a key point from CCNN in the sense that the output of the new structures (rules in MILCS and hidden nodes in MINES) should align to the error of the existing system; in CCNN aligning means a high correlation coefficient whereas in MILCS and MINES it is a high MI.

Chapter 4

MINES Implementation

This chapter deals with the implementation of the systems incorporated in MINES. Implementation of MINES required an extensive amount of coding and a wide range of programming to properly connect the incorporating systems. The GA package was pre-written and equipped with both binary and real variables (however, only the binary variables are used to encode the connectivity patterns of the hidden nodes). For the clustering the well-known Weka package [22] is used. The technical details of the systems collaborating in MINES will be discussed in macro and micro levels. It is expected that, at the end of the chapter, the reader becomes clear about how the multi-collaboration of the systems results in an optimal, or close to optimal, FFNN in MINES. The reader would, furthermore, realize how structural and parameter learnings can be applied concurrently by maximising the MI of the hidden layer and error space.

Section 4.1 gives a short overview of the incorporating systems of MINES.

Section 4.2 explains the important implementation steps in creating a multi-layer FFNN which was used in MINES. The implementation of the adaptive learning rate and its capabilities and the primarily tests to evaluate the performance of the implemented FFNN are explained.

Section 4.3 starts with the macro algorithmic steps of MINES, evolving the underlying FFNN. Additional details of steps are provided in micro levels in a separate subsection.

Section 4.4 deals with the technical details of important steps of GA. The rationale behind adopting MI as the fitness function of GA individuals is discussed separately.

Section 4.4.1 explains the connectivity patterns of hidden nodes and the way GA individuals get associated to the hidden layer.

Section 4.5 provides the pseudo-code of a new clustering algorithm of the weight vectors based on their angles with the input-vectors in MINES. Section 4.5.1 explicitly formulates the mathematical foundations which the new clustering algorithm is derived from.

Section 4.6 highlights the effect of the number of GA generations and BP iterations on the performance and convergence of MINES.

Section 4.7 explicates how the incorporating systems of MINES would result the system to converge to a fixed topology by optimizing the associated MIs.

Section 4.8 provides remarks and summary of the chapter.

4.1 Overview

MINES combines a three layer FFNN with a GA used for structural learning and BP for parameter learning, connected via the MI between the error space and the hidden layer. The GA intends to select the right synaptic weights to evolve the structure of the hidden layer; i.e. the number of hidden nodes and their receptive fields. The MI of the output of each hidden node and the network's residual error is the associated fitness of each GA individual. Given each hidden node, the network's residual error is the error occurring when the specific hidden node is excluded from the original network. This can be achieved through disconnecting the hidden nodes' synaptic weight outgoing from the hidden layer to the final output-layer. Clustering is applied to the synaptic weights to merge hidden nodes having same functionality. Finally, weights are adjusted through BP learning algorithm.

As mentioned earlier (see Section 1.1) there is no concerning barrier to extend the employed technique to a multi-layer FFNN. However, a three layer FFNN makes the evaluation of the implemented method less complicated. Therefore, the implementation is restricted to a single hidden layer, focussing on the number of incorporated hidden nodes, the configuration of the receptive fields and the associated synaptic weights.

4.2 Implementation of Multilayer Feed-Forward Neural Network

The first step in the implementation of MINES was to create a multilayer FFNN. To achieve this, a FFNN with no hidden layer was first created in Java language, using the Colt library [18]. The number of nodes in the input layer and the output layer are defined by two integer variables; input nodes include one extra node as bias. The synaptic weights of the implemented NN is a two dimensional matrix (DoubleMatrix2D), having double variables, with the number of rows and columns, respectively, equal to the number of output and input nodes. The multiplication of the matrix and the input pattern results the induced local field vector. The activation function is applied to the induced local field vector to result in the output vector of each layer. The output vector is compared to the desired vector to calculate the error of each node and subsequently the overall MSE of the network. The BP algorithm is applied to the synaptic weights to recursively train each layer to the desired error level.

4.2.1 Dynamic Learning Rate

We have adopted a dynamic learning rate for the BP procedure. The BP starts with an initial learning rate. After some iterations, the system observes the error reduced. If the error of the network is not decreased by a predetermined certain amount, the initial learning rate increases by a fixed percentage. Whenever the learning rate increases, the next plausible time to increase learning rate decreases by a fixed number of iterations. The procedure of increasing the learning rate, and accordingly the time decline of consecutive plausible increases, continues till the network's error happens to increase instead of decreasing; i.e. when the BP algorithm falls in a local minimum. By the time of this, to bring the BP back to the right position in the weight space, the currently set learning rate decreases by a certain fraction, while the weights are seized to become updated. At this stage, the next time to increase the learning rate is postponed by a predefined number of iterations, since the learning rate is currently over increased.

The applied algorithm is notably beneficial as it often prevents the BP algorithm trap in a local minimum - where error resonates in an area of the weight space. Moreover, the algorithm makes use the maximum possible learning rate since it is dynamically adjusted during the training procedure. It is worth mentioning that, usually after a long training period the learning rate converges to a stable value.

4.2.2 Preliminary Performance Tests

The performance of the implemented FFNN was initially tested on the XOR and Unique Square problem (2 Input nodes, 2 hidden nodes and 1 output node was required to solve the problems). The trained network showed a quite acceptable performance on these two tests. In order to further test the ability of the NN on a harder problem, the standard two-spiral problem was used (see Sections 5.2.1 and 5.2.2 for the formula and definition of the two-spiral problem, respectively). The NN having one hidden layer with 30 hidden nodes, equipped with tangent hyperbolic as the activation function, was able to classify correctly the training set with the accuracy of 93% after $130,000$ BP iterations, while the final MSE was reduce to 0.03. To achieve this result, a dynamic learning rate (see Section 4.2.1) had to be incorporated in the system . Without the aid of the dynamic learning rate the system was not able to reduce the MSE to a value less than 0.1, and could not make a classification better than 55%. It must be noted that, the two-spiral problem has also used as the benchmark classification performance of MINES (see Section 5.2).

4.3 MINES Step by Step

The major steps of MINES algorithm are as follow (see Figure 4.1):

1. Initialize GA with a random population of binary variables. The binary variables define the receptive fields connectivity patterns of hidden nodes to the corresponding input nodes (1 means connected, 0 means disconnected). The binary variables are the *genotypes*, where each *gene* is a bit defining the state of connection of an input-layer-node to the associated hidden node (the *phenotype*) in the implemented three layer FFNN.

2. Assign to each GA individual a randomly generated weight vector, with one weight for each gene in the corresponding genotype. Note that, the weight vectors are assigned outside the GA; meaning that, they are not directly defined in the individuals' genotypes. However, when a corresponding gene in the genotype is 0, the value of the associated weight is also set to 0; whereas, when a gene is set to 1, the associated weight keeps its assigned value.

3. Group GA individuals having identical binary (connectivity) variables. Then, cluster each group based on the angles of the respective weight vectors. Within each group, weight vectors having close angles are bundled together.

4. Each cluster represents a hidden node in the implemented three layer FFNN. In fact, each hidden node is a phenotype which its characteristic - i.e the receptive fields connectivity patterns and the

respective synaptic weights - is exactly defined according to the individuals of the corresponding cluster. The receptive field connectivity pattern of each hidden node is determined by the unique binary variable of the individuals in the corresponding cluster; while the respective synaptic weights incoming from input nodes to the hidden nodes (i.e the weight vector of the hidden node) is the average of the weight vectors assigned to the individuals in the cluster.

5. Implement a three layer FFNN by assigning outgoing-weights to each hidden node (i.e. phenotype), connecting the hidden node to the final output nodes of the network.

6. Repeat the following steps until a convergence criterion is met:

   (a) Train the FFNN for some number of iteration steps of BP over a given set of training data.

   (b) Obtain the fitness of each GA individual by directly calculating the MI of the representing hidden node and the residual error. Note that for each hidden node - and hence for all associated individuals in its cluster - the MI is measured between the output of the hidden node and the corresponding residual error. For each hidden node, there is a corresponding residual error which is the remaining error of the FFNN when all the synaptic weights, outgoing from that hidden node to the final output-nodes of the network, are disconnected.

   (c) Set the fitness of each GA individual to the MI of the output of its representing hidden node and the respective residual error calculated in the previous stage.

   (d) Create a pre-specified number (λ) of new GA individuals through mutation and crossover of the selected parents from the population. This stage is called the GA generation-plan (GP) (see Section 4.4).

   (e) Assign weights to the binary variables of newly created offspring. After this stage, each new offspring can potentially represent a hidden node since its receptive field connectivity pattern and its weight-vector is completely defined. Depending on the adopted technique, the new offspring may represent a new hidden node or it may be only assigned to an existing hidden node in the implementing FFNN.

   (f) Step *b* is applied to the new offspring or, if necessary, also to the old individuals to update the fitness of GA individuals.

   (g) According to the calculated fitness of individuals, a new population is created by replacing some of the old individuals with the new offspring (see Section 4.4 for details).

   (h) Group GA individuals having identical binary (connectivity) variables. Then, cluster each group based on the angles of the respective weight vectors. Within each group, weight vectors having close angles are bundled together.

   (i) Implement a new three layer FFNN by assigning one hidden node to each cluster obtained in the previous stage.

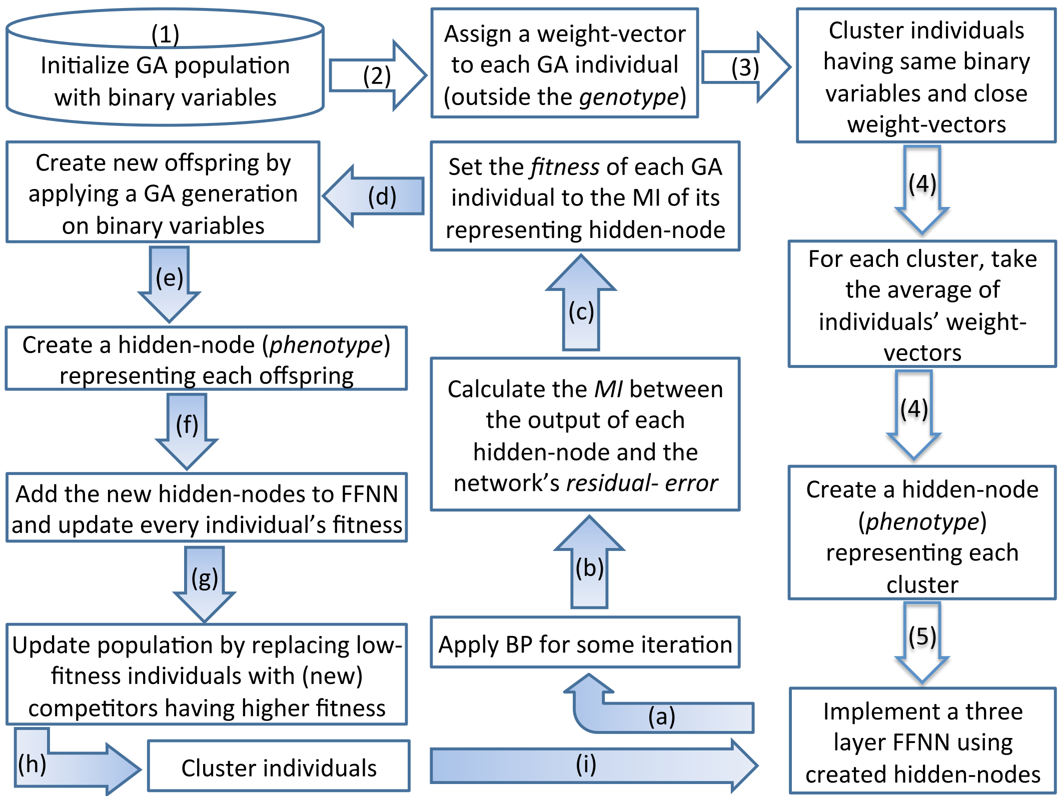

(1)
Initialize GA population with binary variables
(2)
Assign a weight-vector to each GA individual (outside the *genotype*)
(3)
Cluster individuals having same binary variables and close weight-vectors
(4)
For each cluster, take the average of individuals' weight-vectors
(4)
Create a hidden-node (*phenotype*) representing each cluster
(5)
Implement a three layer FFNN using created hidden-nodes
(a)
Apply BP for some iteration
(b)
Calculate the *MI* between the output of each hidden-node and the network's *residual- error*
(c)
Set the *fitness* of each GA individual to the MI of its representing hidden-node
(d)
Create new offspring by applying a GA generation on binary variables
(e)
Create a hidden-node (*phenotype*) representing each offspring
(f)
Add the new hidden-nodes to FFNN and update every individual's fitness
(g)
Update population by replacing low-fitness individuals with (new) competitors having higher fitness
(h)
Cluster individuals
(i)

Figure 4.1: Steps involved in MINES implementation. The numbers and letters in the arrows correspond to steps listed in Section 4.3.

4.3.1 Details of Steps

The following passage goes through the details of the major steps of MINES methodology in the same order brought up earlier in Section 4.3:

1. The initial size of the GA population, which remains constant during evolution, usually depends on the number of input nodes, n, of the implemented NN. As mentioned earlier, for every hidden node the number of potential synaptic connections increases exponentially with the number of input nodes. Hence, it might be improbable for GA to search through all combinations, especially, when n is large. In this case, increasing the size of the initial population may assist GA to examine more connections. With a larger population more offspring are created and assessed during the evolutionary process. This, in turn, increases the chance of exploring more combinations of receptive fields, making GA to converge through a more diverse population. However, a disproportionately large population not only prolongs the GA convergence, but also often decelerates the BP training process. The latter is due to the incorporation of many hidden nodes in the network, when n is large. Another way to increase the chance of searching through various combinations of receptive fields is to produce more offspring per GA generation. The more offspring are created in a generation, the more the chance of creating a new connectivity pattern. However, there is a limit for λ. For instance, the number of offspring created in each generation depends on the size of the set R in the replacement plan (see Section 4.4): it is not sensible to have $\lambda \geq r$ (r is the size of the set R).

2. The receptive weights are randomly generated between $-w$ and $+w$ (usually $w = 1$). The BP modifies the weights when parameter learning is applied.

3. Potentially, clustering of weight vectors could also be applied based on their relative distance in the space; for vectors X_i and X_j the distance would be defined as $\| X_i - X_j \|$. Vectors with close angles, but large distance, would be clustered separately using the latter technique. This is, however, unnecessary and could be regarded as a drawback (see Sections 4.5.1 and 4.5).

4. The synaptic weights of each newly clustered hidden node is calculated as the average of the former synaptic weights of the hidden nodes associated to the individuals of the current cluster. It is worth pointing out that, if instead of the average, sum of the synaptic weights is used, the BP algorithm is likely to fail. This is because, the magnitude of the induced local fields may exceed the activation functions thresholds. When the activation function is tangent hyperbolic or sigmoid, large synaptic weights produce values close to 1 or -1; preventing BP to update weights properly. This is because BP deals with derivative of activation functions, and tangent hyperbolic and sigmoid have derivatives close to 0 when dealing with values close to 1 or -1.

5. The outgoing weights are created randomly for the first time. After that, each newly created hidden node inherits the average of its parents assigned outgoing weights; other hidden nodes keep their old outgoing weights.

6. A convergence criterion is met when: 1. The MSE cannot be significantly reduced for some successive generations by BP; 2. The number of hidden nodes do not change for some successive generations; 3. No major improvement can be observed on the performance of the trained NN on the validation set.

   (a) The number of BP iterations could be a pre-specified fixed number of iterations or dynamic changing in each generation. When a fixed number of BP iterations is used, the MSE of current generation may happen to be higher than the previous one. Therefore, the MIs between hidden nodes and residual MSEs becomes less comparable to those measured in the previous generation. This is because the synaptic weights are not trained enough to produce the same level of error as before. Hence, the hidden nodes' outputs and the entropies of residual errors become less comparable to the former generation. This may cause a hidden node which is strong in error detection, showing a low MI in current generation, but a high one in previous generations, be substituted by a tenuous one showing the opposite (i.e. a high MI in current generation, but a low one in previous generations). In fact, a rather uniform descending MSE across GA generations may guarantee that the trained NN in each generation is in a point in the weight space that is at least as good as the one in the former generation.

   (b) The MI is calculated by the histogram technique introduced in Section 2.4.2. The technique requires binning of the range of the random variables which their MI is ought to be calculated. Given a hidden node, one random variable is the function providing the output values of the hidden node, whereas the other is the one producing the corresponding residual error values. In fact, both random variables are functions of input patterns and associated weight vectors. For each pattern in the training epoch, each random variable produces values in its range. When the activation functions are squashing, the produced outputs (i.e. hidden nodes or residual error values) are bounded in consistent with the range of the squashing function. Hence, it is easy to assign an upper bound and a lower bound for the random variables under consideration and apply the binning based on the bounded interval. This, however, is not the case when the activation function is unbounded - like the identity function. Binning the range of an unbounded function requires an infinite number of bins which is computationally impossible. A straightforward way to solve the issue is to assign a sufficiently large upper bound and lower bound covering most of the produced outputs in the range; all other outputs falling outside the assigned bounded span would be correspondingly transferred to one of the endpoints of the interval. However, a more effective way to bin the range of a boundless activation function is to transform the outputs to a constrained interval. MINES adopts linear transformation, although both linear or non-linear can be employed. It is particularly worth noting that the transformation does not change the value of MI due to its exclusive property of being invariant under invertible transformations (see Equation 2.16). The minimum and the maximum values of the activation function produced over all patterns in the epoch are first obtained and transformed to the new minimum and maximum. Other output values will then be also transformed in the same manner between the new minimum and maximum. The new minimum and maximum can be assigned to fixed for all hidden nodes and residual errors through the whole evolutionary process (for instance Min=0 and Max=1).[1]

(c) The calculated MIs may be rounded to some decimals places before being assigned to the respective individuals.

(d) The effect of using various number of λ has been previously investigated.

(e) As explained earlier, a new offspring must represent a hidden node in the NN according to the individual's connectivity variables. However, the offspring would not automatically represent any synaptic weight (receptive or outgoing) as the GA evolution is only applied to the connectivity variables. Since the performance of a hidden node depends on both the connectivity patterns and the synaptic weights, it is essential to assign weights to the existing connections. Various techniques can be adopted to determine synaptic weights to the connectivity variables of a new offspring:

i. Weights can be created randomly and assigned to the existing connectivity variables.

[1]The hidden nodes in MINES are specialized so that each detects a different part of the error space. Consequently, the outputs of the hidden nodes usually do not overlap, but each span a different part of the activation function's range. Moreover, as the NN is being trained and evolved, the concentration of the outputs of each hidden node may shift from one former area to another one in the range. The linear transformation of the output values of hidden nodes and residual errors to a fixed interval, such as [0, 1], resolves this inconsistency which, otherwise, would make the comparison of individuals inefficient. The binning is, then, performed uniformly over the fixed interval with a predefined number of bins.

ii. GA can evolutionary create weights at the same time as evolving connectivity variables.

iii. The weight vector associated to a new offspring can simply be the average of the weight vectors of its generating parents. If a synaptic connection is disconnected in both of the parents but is enabled in the new offspring, a randomly generated weight will be assigned to the connection.

After that, the new offspring is assigned to a hidden node in one of the following ways:

i. A new hidden node is created based on the offspring's incoming and outgoing weight vectors and is added to the NN. In his case, the new hidden node would only represent the originated offspring and not any other individuals in the population; though this might be subject to change in future generations. Consequently, this technique temporary adds r (the generation overlap size) hidden nodes to the NN as each newly created offspring represents a unique hidden node. However, the number of hidden nodes may change in subsequent steps (after the update plan (UP)) when clustering is applied to the new population.

ii. The incoming weight vectors associated to new offspring are clustered with the existing hidden nodes prior to incorporating in the NN. Clustering results new hidden nodes be added to the NN so that each individual in the population is associated to a hidden node.

The advantage of method i over method ii is that the new offspring is evaluated solely as a single hidden node in the incorporated NN. The fitness of the new offspring is, hence, a direct measurement of the potency of its connectivity patterns and synaptic weights in acquiring information from the error space. Additionally, to restrict distortion in the fitness of the existing hidden nodes and also to prevent change in the present MSE of the network, the outgoing weight of each new offspring can be assigned to 0 in steps f, g, h, but set to the parent's average in the subsequent steps.[2] Hence, adding hidden nodes would impose no distortion in the fitness of the existing individuals; neither no change in the current MSE of the network. Method i is also less computationally expensive as does not deal with an extra clustering step required in method ii. Therefore, overall, the method i is preferred over the method ii and has been used in the upcoming presented experiments.

(f) As mentioned in the previous stage, depending on the adopted technique when introducing a new hidden node, the fitness of old individuals may change. Hence, fitness calculation may be applied on both new and old individuals.

(g) Before the replacement plan (RP), individuals may first get sorted based on the magnitude of the outgoing weights of the associated hidden nodes. The r individuals then are selected from the sorted individuals so that those with smaller outgoing weights have more chance to get selected to form the set R. This extra step insures that in the next stage, the update plan (UP), individuals with associated large outgoing weights are not replaced. Problem,

[2]This is because a zero weight is little sensitive to the MSE, and hence it would slow down the BP training process. Therefore, it is better to change the outgoing weight of the new hidden node to the parent's average before starting the next BP training process.

sometimes, arises when an individual with a small fitness, but a large outgoing weight, leaves the population in favour of a new offspring. If the cluster representing the hidden node associated to the individual only contains the individual itself, the hidden node becomes omitted from the NN. The omission of a hidden node with large outgoing weight, hugely, increases the MSE. Despite the huge distortion that this increase of the MSE imposes to the system, it usually takes a long time for the BP to reduce the MSE to the former value.

4.4 Genetic Algorithm in MINES

The GA algorithm is mainly implemented based on the procedure described by Deb [10]. Initially, the fitness of all individuals is evaluated to determine the best individual (usually the one with the highest or lowest fitness). The first step, afterwards, is the selection plan (SP): a subset P of randomly selected individuals with size μ, the predetermined number of parents per generation, containing the best individual, is chosen. The second step is called the generation plan (GP): a predetermined number of offspring, λ, forming the set C, is created in each generation. Each new offspring is the result of the recombination of two parents, chosen from P, with two-point binary crossover accompanied by a possible point mutation. The third step is the replacement plan (RP): the set R, containing r individuals selected randomly from the population, is formed. The last step is the update plan (UP): the r individuals of the last step are replaced with the best of created offspring and the selected competitors from the set C and R, respectively; r is usually called the generation overlap size.

The bigger is the generation overlap size, the higher is the selective pressure. This is because, when the generation overlap size is large, new offspring are compared with a larger portion of old population before being replaced. When the generation overlap size is small, the convergence of the system is usually slower. When the generation overlap size is relatively bigger, the convergence is faster, but the diversity of the population may not preserve for a long time.

The presence of the best individual in P, in the SP, makes the algorithm to become “somewhat greedy” [10]. However, the bigger is μ, the less is the chance of the best individual to be selected as a parent in the generation plan. Therefore, by adjusting μ, the degree in which the algorithm behaves greedily can be tuned.

4.4.1 Representation

Each GA individual in MINES is associated with a single hidden node in the network. However, the converse is not necessarily true: a hidden node in the network may represent one or more individuals in the population. A vector of bits is assigned to each hidden node representing its connectivity to the input nodes including bias. In fact, the connectivity pattern of synaptic weights is controlled by an array of bits (i.e. a set of binary variables), in which 1 represents an existing connection and 0 means that the synaptic weight is disconnected; for instance, if the input layer consists of n nodes (including bias) there will be 2^n unique (distinctive) connections. Hence, the structure of the hidden layer is largely controlled by GA individuals. The binary variables control the connectivity patterns of hidden nodes' receptive fields so that the convergence of GA individuals affects the quantity of the hidden nodes and the diversity of connectivity patterns. The terms *receptive fields connectivity pattern*, *receptive fields connectivity*, *receptive connectivity* or *connectivity patterns* are all referred to the state of the synaptic connections of a hidden node incoming from the input layer.

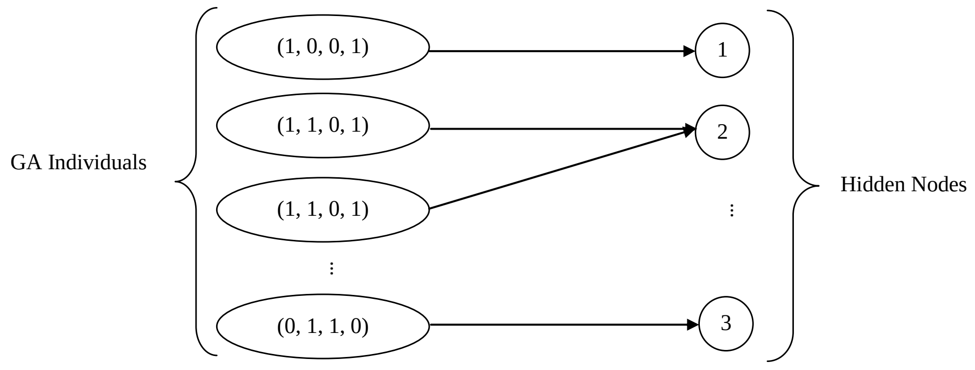


Figure 4.2: Relations of GA individuals and associated hidden nodes in MINES

Diagram 4.2 depicts the relation of GA individuals and hidden nodes in MINES. Each GA individual has a connectivity variable of 4 bits in this example, implying that the number of input nodes (including bias) of the corresponding NN is 4. For instance, the first individual has the connectivity variable of $(1, 0, 0, 1)$; which means the representing hidden node (hidden node 1) is connected to the first and last but not to the second and the third input nodes. The diagram also shows that the second and the third GA individuals represent a single hidden node (hidden node 2); the two individuals are in fact in the same cluster. Individuals in a single cluster have identical connectivity variables and also close synaptic weights.

4.4.2 Mutual Information as Fitness Function

The fitness of each GA individual in the population is the MI between the output of the associated hidden node and the residual error of the NN. The residual error guarantees that the specific hidden node has not taken part in the error space. Consequently, the MI does not measure the self correlation that would otherwise exist between the output of the hidden node and the total error of the network.

The primary objective of using MI as the fitness of GA individuals is to directly convey information from the error space to the hidden layer. A high MI implies that the contributed hidden node correlates largely with the residual error. Therefore, pruning the receptive fields, and modifying the incoming and outgoing synaptic weights of the hidden node would largely reduce the residual error and the total error, respectively. For instance, if the residual error caused by excluding hidden node i is bigger than hidden node j, we may conclude that the contribution of hidden node i in the implemented NN is more important than hidden node j.

In fact, assigning MI as the fitness function of GA individuals brings a competition among hidden nodes to acquire relatively more information from the error space. The competition also makes individuals who are less able to convey information leave their position in favour of the more competent ones. The capability of the competent individuals comes from both their receptive fields connectivity patterns and their weight vectors.

4.5 Clustering to Form Hidden Layer

Individuals with a same GA encoding are clustered, using the weight vectors of their associated hidden nodes' incoming synapses. Distance in this clustering operation is the absolute angle between the weight vectors. Each cluster is translated to a single node in the hidden layer. In fact, one or several individuals may represent a single hidden node. Note that due to this process, hidden layer size grows or shrinks during the evolution, until an effective number of hidden nodes is obtained. This is similar to the macro-classifier concept in MILCS and XCS [71].

Section 4.5.1 mathematically discusses that the angle between weight vectors can be used as a similarity measurement of their functionality, assuming that the vectors have close length. The following pseudo-code is introduced - with the same notations used in Section 4.5.1 - as a new clustering algorithm of hidden nodes producing similar induced local field:

```
1. Use Simple K-Means or X-Means for clustering.
2. Change the distance function of Simple K-Means [73] or X-Means [46] to the following:
    (a) For every hidden node $i$ and $j$ calculate $\theta_{ij}$, which is the angle of two non-zero weight vectors $X_i$ and $X_j$; where:
        $\theta_{ij} = \arccos\left(\frac{X_i.X_j}{\|X_i\|\|X_j\|}\right)$
    (b) If $\theta_{ij} \geq \pi/2$, then convert $\theta_{ij}$ to $\pi - \theta_{ij}$
3. Cluster the hidden nodes using the function defined in step 2.
4. For every cluster, calculate the average of the weight vectors of the cluster:
    $\overline{X} = 1/n \sum_{i=1}^{n} X_i$
5. Use $\overline{X}$, obtained in previous step, as the weight vector of the representing hidden node of the cluster.
```

As it is observed in the above pseudo-code, clustering of hidden nodes is only performed based on the angles of weight vectors and not their norms. This is in contrast with the earlier discussion suggesting that clustering must be performed based on both the angles and the norms of the weight vectors. However, since in the above algorithm the average of weight vectors of each cluster is used, in the case when a weight vector has a far distant norm from the others, this will also be reflected in the averaged weight vector. Therefore, the final representing hidden node is a good approximation of the hidden nodes forming the cluster.

4.5.1 Angle Indicating Similarity

For a given hidden node, the incoming synaptic weights can be shown in vector from as:

$$X = (x_1, \dots, x_n)$$

X is called the weight vector of the hidden node; where x_k ($k = 1, \ldots, n$) is the incoming synaptic weight from input node k to the corresponding hidden node and n is the number of nodes in the input layer. The induced local field vector for a given input pattern vector:

$$P = (p_1, \ldots, p_n)$$

is calculated as the inner product of the two vectors X and P:

$$X.P \tag{4.1}$$

Equation 4.1 can also be written in matrix multiplication form:

$$XP^T$$

Suppose $\parallel . \parallel$ is the vector norm's function from the vector space to the real numbers, and

$$0 \leq \theta \leq \pi \tag{4.2}$$

is the angle between the vectors X and P; then:

$$X.P = \parallel X \parallel \parallel P \parallel \cos\theta \tag{4.3}$$

For two hidden nodes i and j, suppose X_i and X_j are the associated non-zero weight-vectors. Given an input-pattern vector P, the corresponding induced local fields are:

$$X_i.P = \parallel X_i \parallel \parallel P \parallel \cos\theta_{ip} \qquad 0 \leq \theta_{ip} \leq \pi \tag{4.4}$$

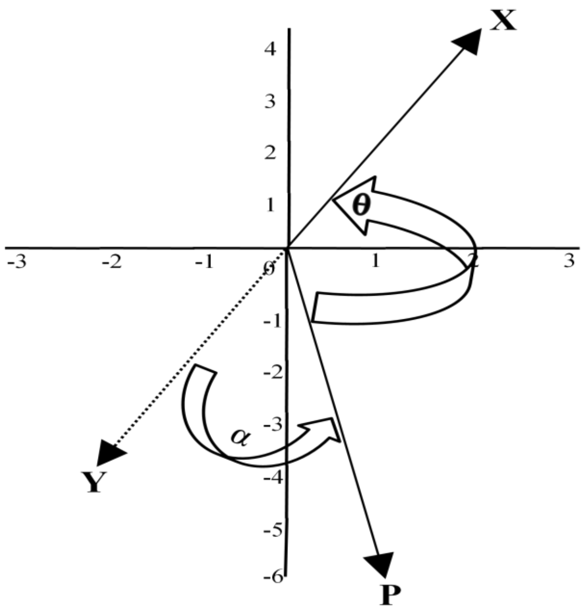


Figure 4.3: The above diagram shows the angle between X (the weight-vector of a hidden node) and P (an input-vector) in a two dimensional space. It is assumed that the number of input nodes, including bias, is 2. The receptive connectivity of the hidden node is $(1, 1)$ with the weight-vector of $(2, 4)$. The diagram also shows that the coordinates of the input-vector is $(1, -6)$. The angle between the two vectors in Radian is calculated as: $\theta = ArcCos(\frac{(2,4)(1,-6)}{\|(2,4)\|\|(1,-6)\|}) \approx 2.51$. If the other weight-vector, Y, lies in the same line as X, but with opposite direction, the angle between P and Y would be: $\alpha = \pi - 2.51 \approx 0.94$. When the activation function is odd, α changes the sign, but not the magnitude of the original induced local field.

$$X_j.P = \| X_j \| \| P \| \cos\theta_{jp} \qquad 0 \leq \theta_{jp} \leq \pi \tag{4.5}$$

where θ_{ip} and θ_{jp} are the angles between vectors X_i and P, and X_j and P, respectively. Suppose for every P in the epoch, the absolute value of the induced local fields produced by X_i and X_j are equal:

$$|X_i.P| = |X_j.P| \tag{4.6}$$

By applying Equations 4.4 and 4.5 in Equation 4.6 we have:

$$\| X_i \| \, |\cos\theta_{ip}| = \| X_j \| \, |\cos\theta_{jp}|$$

A trivial answer for this equality is:

$$\begin{cases} \theta_{ip} = \pi/2 \\ \theta_{jp} = \pi/2 \end{cases} \tag{4.7}$$

which is an insensible answer: it means the two weight vectors are orthogonal to the input space. Hence, all the induced local fields are zero. In this case, the output of the respective hidden nodes would also be zero if the activation function, φ, intersects the origin of the Cartesian space; i.e. $\varphi(0) = 0$. On the other hand, if $\varphi(0) \neq 0$, it only shifts the induced local fields of the next layer. This, however, is unnecessary as the existing bias of the next layer would, itself, apply an affine transformation to the layer. Therefore, this trivial solution for Equation 4.6 can be ignored, but two other trivial answers still exist:

$$\begin{cases} \| X_i \| = \| X_j \| \\ \theta_{ip} = \theta_{jp} \end{cases} \tag{4.8}$$

and:

$$\begin{cases} \| X_i \| = \| X_j \| \\ \theta_{ip} = \pi - \theta_{jp} \end{cases} \tag{4.9}$$

Since Equations 4.8 and 4.9 must hold for every pattern in the epoch, it is required that:

$$\theta_{ip_h} = \theta_{jp_h}$$

and

$$\theta_{ip_h} = \pi - \theta_{jp_h}$$

where θ_{ip_h} is the angle between X_i and P_h, while θ_{jp_h} is the angle between X_j and P_h, and P_h is the pattern in the epoch with size s; $h = 1, \ldots, s$. However, it is very unlikely to have equal angles between all input patterns and the weight vectors X_i and X_j when dealing with large epochs. The reason is that the two weight vectors are fixed in the space, while the input pattern vectors are likely to occupy the space dispersedly. The larger the epoch, the more possible for the input pattern vectors to occupy the space unevenly. The case is even less possible for nonlinear or complex data, like most of the real world problems. Since there is a nonlinear relation between vector components of nonlinear input vectors, it is unlikely to have a linear relation between their angles with two fixed weight vectors. The problem is exacerbated when the input space is a high dimensional one; i.e. when the number of input nodes is high. In this case the input pattern vectors can occupy the high dimensional space freely and hence are unlikely to have equal angles with the two weight vectors.

Thus, ignoring the special cases in Equations 4.7, 4.8 and 4.9, a general answer to the problem is:

$$\frac{\| X_i \|}{\| X_j \|} = \frac{|\cos \theta_{jp}|}{|\cos \theta_{ip}|} \tag{4.10}$$

Since the left hand side of Equation 4.10 is fixed, with the same argument as before, it can be said that it is unlikely to have an input space and vectors X_i and X_j that fulfil equality 4.10. As a result, we may derive the following conclusion: instead of considering the angles between input pattern vectors and weight vectors, it is better to consider the angle between the weight vectors themselves.

Defining θ_{ij} as the angle between X_i and X_j, in limits, for every input pattern vector P in the epoch we have:

$$|X_i.P| \to |X_j.P|$$

if:

$$\begin{cases} \| X_i \| \to \| X_j \| \\ \theta_{ij} \to 0 \end{cases} \tag{4.11}$$

or:

$$\begin{cases} \| X_i \| \to \| X_j \| \\ \theta_{ij} \to \pi \end{cases} \tag{4.12}$$

Since in practice we do not deal with the limits of the weight vectors and the angles, a parameter taking into account the distance of the weight vectors' norms and their angles is used:

$$\begin{cases} c = \lambda |\ \|X_i\| - |\ \|X_j\|\ | + (1-\lambda)\,\theta ij \\ 0 \le \lambda \le 1 \end{cases} \tag{4.13}$$

Equation 4.13 introduces a distance function measuring how far the induced local fields of two weight vectors x_i and x_j are relative to each other; λ is the desired trade-off parameter between the absolute distance of the vectors' norms and their angle.

In Equation 4.6, the absolute value of the induced local fields has been used while, in general, NNs deal with real values and not necessarily with absolutes. However, considering the activation functions of the hidden nodes, it can be shown that the use of absolute values do not confine the generality of the aforementioned arguments. If the activation function, φ, is even - i.e. for every t in the domain $\varphi(-t) = \varphi(t)$ - the sign of the induced local fields can be ignored. If φ is odd, then for every t in the domain: $\varphi(-t) = -\varphi(t)$. In this case it is obvious that the signs are vanished if condition 4.11 happens.

For the case of condition 4.12, the signs can be ignored provided that the sign of one of the hidden nodes' outgoing weight is changed. Since $\theta_{ij} \to \pi$, the vectors X_i and X_j are almost in the same line, but in opposite direction. Therefore, for every vector P in the epoch: $\theta_{ip} = \pi - \theta_{jp}$. So:

$$\begin{aligned} X_i.P = \| X_i \| \| P \| \cos \theta_{ip} &\to \| X_j \| \| P \| \cos(\pi - \theta_{jp}) \\ &= \| X_j \| \| P \| \cos \theta_{jp} \\ &= -X_j.P \end{aligned}$$

Hence, by changing the sign of the outgoing weight of hidden node i (or j), the new NN would be identical to the previous one. So, the sign of the induced local fields can be ignored even when the activation function is odd. Since most of the employed activation functions are often even or odd, the above two cases are usually considered when evaluating the performance of hidden nodes' receptive weight vectors.

4.6 Number of Generations and Back-Propagation Iterations

As stated earlier MINES adopts structural and parameter learning concurrently: every GA generation, defining the connectivity patterns (structural learning), is followed by some number of BP iterations, adjusting synaptic weights (parameter learning). Applying structural learning alone would result the contributed weights untrained; leading to a large error, making supervised learning unsuccessful. On the other hand, using parameter learning in isolation would leave the NN structure unsuitable for the problem at hand. Hence, and also according to the extensively surveyed literature, the two techniques must be carried out simultaneously. However, two natural questions would arise in this regard: 1. how many GA generations must be applied before starting a parameter learning session; 2. what is the appropriate number of BP iterations required after a course of structural learning.

Regarding to the first question, we may say that one GA generation is enough to be applied in each session. If it is needed, increasing the number of new offspring per generation would provide enough diversity leaving additional number of GA generations unnecessary. Moreover, several number of consecutive GA generations, without reducing the MSE, may result in a premature structural convergence.

With respect to the second question, the appropriate number of BP iterations is harder to determine. Nevertheless, it seems straightforward to presume that few number of BP iterations are inadequate as it would only cancel out the jump in the MSE, resulted from the previous distortions caused by the GA or clustering to the synaptic weights. On the other hand, too many BP iterations would over-tune the synaptic weights. Over-tuning is time consumingly worthless since the finely adjusted synaptic weights would be changed in the next GA generation, and when re-clustering is applied. In fact, a small change in the finely tuned synaptic weights would result in a huge jump in the MSE comparing to the previous generation; a lot of number of iterations are sometimes required to reduce the error to the former levels.

Therefore, the number of BP iterations must be selected so that it would properly reduce the error below the previous levels of former generations while at the same time preventing over-tuning. In this respect, an ad hoc rule of thumb is to use a moderate fixed number of BP iterations after each generation, with the flexibility of applying further BP iterations to reduce the error to former levels whenever necessary.

4.7 Convergence of MINES

The convergence of MINES is a combination of a direct and an indirect convergence of the GA. As stated earlier, each individual in the population is associated to a single hidden node in the NN. However, this does not mean that the total number of hidden nodes is equal to the total number of individuals. In fact, two or more individuals may point to a single hidden node in the network (see Figure 4.2). In other words, the total number of GA individuals remains constant during evolution. What makes change in the quantity of the contributed hidden nodes is a reduction in the number of distinct connectivity patterns and the number of clusters. The total number of hidden nodes, in each generation, is equal to the total number of created clusters.

Direct convergence of GA is the result of the convergence of the population towards individuals possessing higher MI. Since mutation and crossover is applied to the connectivity pattern of the individuals, connections providing higher fitness values would dominate the rest. The direct convergence of the GA is translated in MINES by primarily grouping individuals having same connectivity pattern into a single cluster; although, subsequent clustering may also be applied based on the incoming and outgoing synaptic weights. The less is the diversity of the connectivity patterns in the population, the more likely is to have fewer hidden nodes.

Indirect convergence of the GA is affected by both clustering and BP process. It should be noted that, individuals in the same cluster share the same fitness value since they represent a single hidden node. Clustering, therefore, influences the GA convergence by grouping those individuals sharing same fitness value. The less the number of clusters, the fewer the number of distinct fitness values. It is worth considering that clustering may cause some distortion to the GA evolution. This is because, an individual may change its former cluster and get re-clustered to represent a new hidden node.

On the other hand, clustering of the weight-vectors is directly affected by the BP process. BP modifies all synaptic weights during NN training. The change in the synaptic weights of the hidden nodes alter the angles. Change in the angle of the weight-vectors directly affects clustering and indirectly influences the convergence of GA. Additionally, the modification of the synaptic weights, in general, alters the error of the network and the residual errors of the hidden nodes. Hence, the MIs between hidden nodes and the residual errors, being the fitness of the associated individuals, is directly altered by BP. Concerning this, BP has again an indirect effect on the convergence of the GA, which may partly distort the evolution too.

The convergence of MINES eventually happens when no newly created hidden node is able to be replaced in the evolving NN. This is mainly due to the direct and indirect convergence of the GA mentioned earlier. However, the level of the network's error also plays an important role in the convergence of MINES. When the MSE is sufficiently low, the current incorporated hidden nodes in the network would capture most of the network's error. It equivalently means that these hidden nodes have a considerably high MI with the residual errors. Hence, any newly created hidden node would have a lower chance to possess higher MI than those already incorporated.

In Section 4.6 the suitable number of BP iterations and GA generations were discussed. The number of BP iterations also plays an important role in the way the system is converged. This is because, the error is directly controlled by the BP process. The more is the number of BP iterations after each generation, the faster the error is reduced. A quick reduction in the error is equivalent to granting more chance to the current incorporated hidden nodes to not leave their position in favour of new ones. In other words, the current incorporated hidden nodes would always benefit from a host's advantage of being already a part of the network. Contrariwise, when fewer number of BP iterations are used, more connections find the chance of being incorporated into the network. This is due to a gradual reduction of the error, allowing the GA search for new connections around various points in the weight space. When higher number of BP is used, the NN is likely to get trapped in a local minimum without letting GA to search for alternative connections.

4.8 Summary and Remarks

In this Chapter, it was explained that MINES is a combination of a FFNN incorporated with GA using MI as the fitness function while exploiting a clustering system to eliminate similar weight vectors. A general overview of the implementation of a FFNN using Java language was given. Although the implemented NN is designed to have a multilayer structure, only a three layer FFNN has been used in MINES. A dynamic learning rate was added to the implemented FFNN. The conceptual procedure of the added utility was explained. It was shown that the new system is far more capable than the old one as it could classify the two-spiral problem with 93% accuracy on the training set using 30 hidden nodes.

It was mathematically proved that, when weight vectors have a close length, their angle can be used as a decisive factor defining their similarity. Equation 4.13 defines the similarity factor c of two weight vectors based on their Euclidean length and sharing angle. The equation clarifies why clustering of weight vectors based on their angle is fundamentally true in MINES (refer to Section 4.5 for details of the utilized algorithm).

In MINES the genetic evolution is only applied to the connectivity patterns of hidden nodes through mutation and crossover on the binary variables of GA individuals. Every hidden node is associated to one or several GA individuals; also, every individual is related to hidden node defining its connectivity pattern to the input nodes. The weight vectors of hidden nodes is defined outside the GA mainly by BP algorithm and the clustering system. The clustering is applied by X-means or Simple K-means algorithm based on the angle of the weight vectors. After every GA generation evolving the connectivity patterns, some number of BP iterations are applied to adjust and parametrically train the synaptic weights.

It is recommended to only use one GA generation before running BP algorithm. However, the number of BP iterations is essential to the convergence of MINES. In general, the more is the number of BP iterations, the faster is the convergence.

The convergence of MINES is a combination of a direct and an indirect convergence of the GA. The reduction of the number of distinct connectivity patterns defines the direct convergence of the GA; whereas, BP and clustering would also impose a reduction in the number of clusters, which is attributed to the indirect convergence of the GA.

Chapter 5

Preliminary Testing of MINES

In this chapter the performance of MINES is tested on two theoretical problems: a regression and a classification. Section 5.1 deals with preliminary testing of MINES on a regression problem, while Section 5.2 shows the ability of the system on the two-spiral classification task.

5.1 Preliminary Testing on Regression Problem

This section aims to test the performance of MINES on a regression problem used by Saito and Nakano in 2002 [56][1]. The problem is also used as the main benchmark of testing and tuning the parameters of MINES. A brief structure of the section is as follow:

- Section 5.1.1 brings an overview of the importance of law discovery and its connection with NNs.
- Section 5.1.2 introduces the underlying polynomial regressed by MINES. The rationales and logics of choosing the polynomial regression problem as the main theoretical benchmark of MINES will be addressed.
- Section 5.1.3 explains the most important parameters of MINES that may influence the experimental performance of the system on the polynomial regression problem. It will be argued that the number of BP iterations and the generation overlap size are two important influential factors that the system's stability must be tested against them.
- Section 5.1.4 examines the performance of the system based on the experimental settings. While changing the influential parameters (i.e. the number of BP iterations and the generation overlap size), the stability of the system is assessed according to the obtained number of distinct connectivity patterns, the number of evolved hidden nodes, the *root mean squared errors* (RMSEs) on the training and test sets, the number of GA generations before convergence and finally the total run time of the system. The findings make define most of the important settings possible for the current and future problems.
- Section 5.1.5 deals with the results of one of the best performers of MINES in connection to the correct discovery of the underlying polynomial.

[1]The polynomial first introduced by Sutton and Matheus in 1991 [66]

- Section 5.1.6 compares the performance of MINES to those obtained by Saito and Nakano [56].

5.1.1 Law Discovery with Neural Network

The discovery of numeric laws is regarded as the essence of data mining [55]. The Kepler's third law ($T = kr^{3/2}$) is a famous example in this regard, explaining the relation of the distance to the sun (r) and the revolution period (T) of five planets, using a constant (k). Conventional literature extracting such rules have all a basic strategy in common: they recursively combine two variables into a new one by using multiplication, division or some predefined prototype functions. However, the strategy may turn out to be inefficient when the number of variables at hand is large. In this case, the combinatorial process may become systematically outsized, causing the desired law be missed sometimes. The former is due to the combination of two variables into one which usually must be done in order. In addition, in these systems, the law may not be discovered properly when the attending powers in the desired law are not integers. To overcome this, "some appropriate prototype functions", like $r^{3/2}$, must usually be prepared in advance. Nevertheless, a priori information is rarely available [55].

An alternative approach, contrasting the above mentioned methods, relies on using a NN - as a regression processor - for data mining rule extraction. Define:

$$\mathbf{X} = (x_1, \ldots, x_K)$$

and

$$\mathbf{W} = (W_0, W_1, \ldots, W_J)$$

as two vectors of independent variables in $\mathbb{R}^K$ and $\mathbb{R}^{J(K+1)}$, respectively; where:

$$\mathbf{W_0} = (w_0, w_1, \ldots, w_J)$$

and, for $j = 1, \ldots, J$:

$$\mathbf{W_j} = (w_{j1}, \ldots, w_{jK})$$

Hence, a polynomial function can be formulated as:

$$f(\mathbf{X}; \mathbf{W}) = w_0 + \sum_{j=1}^{J} w_j x_1^{w_{j1}} \ldots x_k^{w_{jk}} = w_0 + \sum_{j=1}^{J} w_j \prod_{k=1}^{K} x_k^{w_{jk}} \tag{5.1}$$

All elements of $\mathbf{X}$ can be assumed to be positive, "by adding an adequate value" to each x_k whenever is necessary [55]. The imposed assumption does not reduce the generality of the problem as the added value can be regarded as shifting the whole data points in the space; the only requirement is to have a finite number of data points. Hence, we may suppose:

$$\forall k \quad 0 < x_k$$

Then:

$$f(\mathbf{X}; \mathbf{W}) = w_0 + \sum_{j=1}^{J} w_j \exp(\ln(\prod_{k=1}^{K} x_k^{w_{jk}})) = w_0 + \sum_{j=1}^{J} w_j \exp(\sum_{k} w_{jk} \ln(x_k)) \tag{5.2}$$

Given that the number of data points are finite, by dividing each x_k by a sufficiently large constant, it can be further assumed that each data point is smaller than one; i.e.:

$$0 < x_k < 1 \tag{5.3}$$

The assumption is again valid as it only represents a uniform scaling of the data in the space[2]. Assuming for each k, $\ln(x_k)$ be an input signal, the right hand side of Equation 5.2 is equivalent to the output of a three layer FFNN having K input nodes, J hidden nodes and one output node; w_0 is the bias term to the hidden layer, while there is no bias to the input layer. All hidden nodes functions are exponential while the output node function is identity.

A common regression problem is about finding a curve in the $\mathbb{R}^{K+1}$ that best fits a number of independent variables from the set $\mathfrak{I}$ to their corresponding dependent variables from the set $\mathfrak{D}$; where:

$$\mathfrak{I} = \left\{\mathbf{X}_i = (x_{1i}, \ldots, x_{ki}) \in \mathbb{R}^K \mid i = 1, \ldots, N\right\}$$

and

$$\mathfrak{D} = \left\{y_i \in \mathbb{R} \mid i = 1, \ldots, N\right\}$$

In order to exploit a NN to find the required regressive function, the set

$$\mathfrak{E} = \left\{(\mathbf{X}_i, y_i) \mid i = 1, \ldots, N\right\}$$

is assumed to be the training epoch, where $\mathbf{X}_i$ is the input vector and y_i is the respective desired value. The training error is the usual MSE over each y_i, the i^{th} desired value, and the output of the network obtained by presenting $\mathbf{X}_i$.

5.1.2 Polynomial Discovery

The polynomial used by Saito and Nakano [56], in Equation 3.15 of Section 3.4, has been selected to test the regression capabilities of MINES. For convenience, the polynomial is again recalled here (note that the polynomial is nonlinear):

$$y = 2 + 3x_1x_2 + 4x_3x_4x_5$$

There are many rationales to choose the above regression problem as a performance benchmark of MINES. First, as provided literature shows ([56]), the underlying regression problem is quite challenging to be solved by a FFNN. The complicated techniques, rooted in advanced mathematical foundations, applied by Saito and Nakano [56] confirms that the problem is important to be explored. In general, the applications of NNs in regression is of interest. The technique can potentially be used in data mining and even stock price assessment; discussed Nakano and Saito in 2002 [44]. Second, the selected regression problem leaves no ambiguity about optimality of the objective FFNN. In other words, it is fairly straightforward to determine the optimal NN regressing the above polynomial. The optimal NN can be used as the benchmark to evaluate the performance of MINES and facilitate comparisons. Any obtained regression formula from the adopted NN, being different from the optimal one, can be easily compared with

[2]Equation 5.3, in Section 5.1.3, has been used to produce unambiguous artificial data set as inputs of MINES.

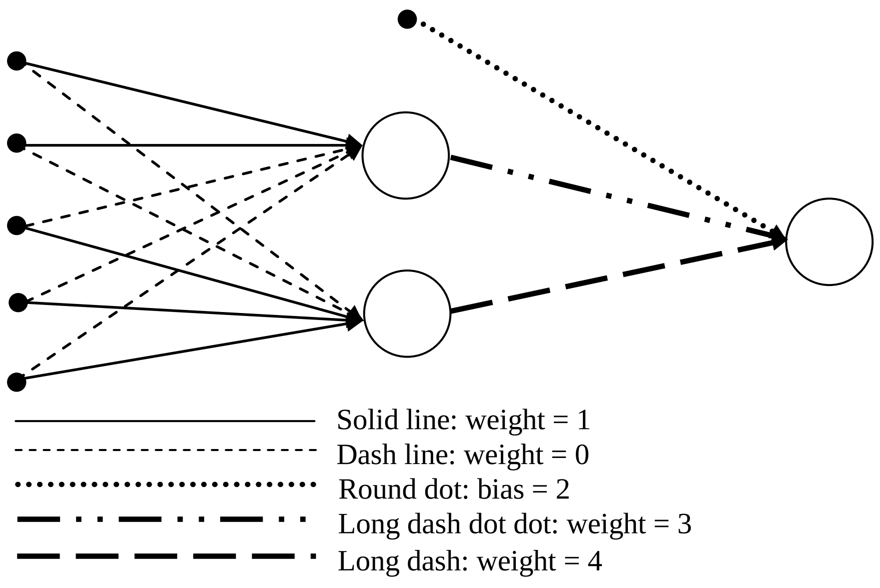


Figure 5.1: NN structure implementing polynomial regression problem

the original polynomial. This is clearly advantageous as in many NN applications a lot of uncertainty arises about the type of an optimal NN in relation to the problem. As it will be shown later in this section, the presented polynomial guarantees and demonstrates the optimal NN regressing it (see Equations 5.5 and 5.6). Third, the desired polynomial can be achieved through a partially connected NN, where zero values of synaptic weights are, in fact, disconnected synaptic connections. Advantageously, this is in line with the learning algorithm that MINES pursues: it constructs the hidden layer from hidden nodes having various connectivity patterns. Fourth, it has been repeatedly stated that [44][56][67] the number of hidden nodes in the NN solving the regression problem cannot be determined in advance. It is asserted that “a criterion” is needed to “choose” hidden nodes “from several candidates” and the task must be resolved by trial and error. In the literature, the decisive factor to find the optimal number of hidden nodes was cross-validation. The polynomial regression problem is, therefore, a right choice for MINES since the ability of the system configuring the required hidden layer can be extensively investigated.

Using Equation 5.2, Equation 3.15 can be transformed to the following new equality:

$$y = 2 + 3\exp(\ln(x_1) + \ln(x_2)) + 4\exp(\ln(x_3) + \ln(x_4) + \ln(x_5)) \tag{5.4}$$

subject to holding that $x_i \geq 0$ $(i = 1, \ldots, 5)$. Suppose $X = (\ln(x_1), \ldots, \ln(x_5))$ is an input vector to a FFNN, having 5 input nodes. The FFNN, regressing the above polynomial, is an optimal one if it can exactly replicate Equation 5.4 by obtaining the following bias and weight values through the training procedure:

$$neuron\,1 : \begin{cases} bias : 0 \\ hidden\ node\,1 : \begin{cases} w_{11} = w_{12} = 1 \\ w_{13} = w_{14} = w_{15} = 0 \end{cases} \\ hidden\ node\,2 : \begin{cases} w_{21} = w_{22} = 0 \\ w_{23} = w_{24} = w_{25} = 1 \end{cases} \end{cases} \tag{5.5}$$

$$neuron\,2: \begin{cases} bias: w_0 = 2 \\ w_1 = 3 \\ w_2 = 4 \end{cases} \tag{5.6}$$

where w_{jk} ($j = 1, 2;\ \ k = 1, \ldots, 5$) is the synaptic weight of hidden node j incoming from the input node k; w_0 is the bias term of the hidden layer, while w_1 and w_2 are the outgoing weights of the hidden nodes to the single output node (see Figure 5.1).

5.1.3 Settings of Involved Parameters

Since MINES is a combination of several systems, many settings are required when starting a training course. In general, it would be a great challenge (see Section 1.5) to coordinate all the involved systems and find appropriate settings optimally working in collaboration. Hence, all important settings are experimentally examined in this section and their effect are discussed. The experiments will reveal parameters the system is mostly sensitive against, aiming to provide a reliable indication to choose appropriate settings for subsequent problems.

To train the objective NN, the training epoch was consisted of 1000 patterns with each pattern being a vector of 5 elements:

$$X = (x_1, \ldots, x_5)$$

so that:

$$x_i = \ln(r_i)$$

where r_i is created randomly in the interval $[0, 1]$, according to Equation 5.3. For each pattern, the desired value is the corresponding value y in Equation 3.15. Similarly, 250 extra data, forming the test set, was created to evaluate the performance of the trained NN. The proportion of the test set and training set was chosen at random.

As explained in Chapter 4, MINES is equipped with a dynamic learning rate adjusted accordingly during the training progress. Hence, the system is mainly insensitive to the initial setting of the learning rate. This is unlike other NN systems, using a fixed learning rate, making them vulnerable to the allocated value. Therefore, the concentration in MINES is not usually on the initial value of the learning rate as it will be adjusted automatically whenever is necessary. However, to avoid the system pause when training is started, the initial learning rate is usually set to a sufficiently small value. In the training procedure of polynomial discovery, the initial learning rate was set to 0.001 for all carried out experiments. Also, a fixed momentum constant of 0.1 was uniformly used to further facilitate training. The hidden and output layer activation functions are, respectively, exponential and identity; as discussed in Section 5.1.1.

As explained in Section 2.4.2, the histogram technique for calculating MI requires binning of the range of the involved random variables. For this purpose, the output range of the hidden nodes and the residual errors must be binned during training procedure in MINES. The required number of bins can potentially be a disputable factor. However, based on the extensive mathematical literature provided in Section 2.4.3, leading to Equation 2.57, we may conclude that, over several estimation of MI and for a sufficiently large number of data points, a low deviation from the mean is more likely to be obtained when a relatively small number of bins is used. Nonetheless, the argument holds true providing that the range assigned to hidden nodes and residual errors are kept fixed when training is in progress[3]. This is usually not the case as activation functions may not have a bounded range, like the exponential function used in polynomial discovery; similarly, there is no guarantee to find a boundary for the residual errors. To overcome these uncertainties all values are transformed to the interval $(0, 1)$ as explained in details in Section 4.3.1. Having fixed the boundary ranges of hidden nodes and residual errors to $(0, 1)$, we have used a fixed number of bins equal to 5 for all variables involved in the calculation of MI. This means the interval $(0, 1)$ has been divided to 5 equal subintervals with the length of 0.2 each. According to the provided literature and also based on some preliminary experiments of MINES, the appropriate number of bins are set to 5, or 10.

The size of the GA population is 32. Since the NN has 5 input nodes, 32 individuals seem to provide enough diversity to explore various receptive connectivity patterns. A larger population often prolongs the evolutionary convergence without providing substantial benefits. The number of new offspring per generation is set to five[4].

The generation overlap size, r, introduced in Section 4.4, is another important parameter defining the selective pressure of the individuals in each generation. Different values of generation overlap size are expected to affect the nature and speed of convergence. This is because, the generation overlap size defines the proportion of the population which are compared to the new offspring before being replaced. For this reason, generation overlap size of 2, 4, 8, 16 and 24 have been used to investigate contrasts in the obtained results.

The number of BP iterations after each generation is another important factor determining the selective pressure in the GA (see Section 4.6). Hence, experiments have been conducted, by adopting various number of BP iterations (5, 10, 20, 30, 40, 50 and 100), to reveal any meaningful difference. For each experiment, the cycle of applying one GA generation followed by several BP iterations has been repeated for 200 times; though, the system usually converges in earlier steps. At the end of the 200 repeats (i.e. generations), the GA stops and the system does not go through further structural learning application. However, the system continues on parameter learning procedure to finely tune the final synaptic weights by applying the BP algorithm for 1000 continuous iterations. It is worth mentioning that, a similar approach of finely tuning weights, after obtaining a steady structure, has also been adopted by Saito and Nakano (2002) [56], when introducing BCW, and also by Tanahashi et al. (2005) [67], in the newer version of BCW[5].

Each set of experiments were performed on 30 randomly created seeds. We have used 30 exper-

[3]In derivation of Equation 2.57, it was assumed that the underlying variables - namely X and Y - have a bounded range. For the random variable X, this can clearly be observed from Equations 2.32 and 2.32. The random variable Y, was also assumed to be the same.

[4]Some preliminary tests have been conducted in this regard by experimenting 1, 2, 4 and 5 new offspring per generation. In general MINES does not show any meaningful sensitivity over changing the number of new offspring per generation, but sometimes the convergence of the system is prolonged.

[5]"We pruned the near-zero ... common weights and *retrained* to get the *final* common weights ..." [67].

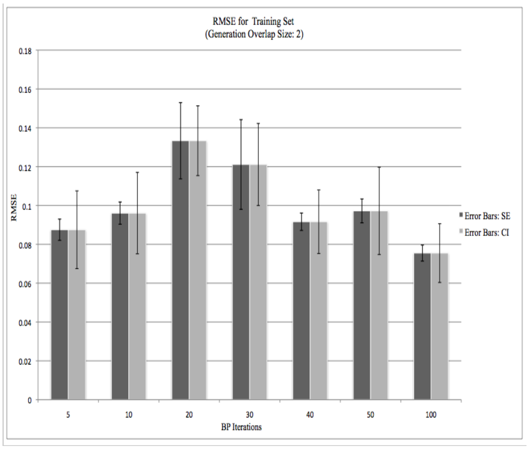


Figure 5.2: Root mean squared error of training set across various number of BP iterations, when the generation overlap size is 2. CI represents a 90% confidence Interval bar.

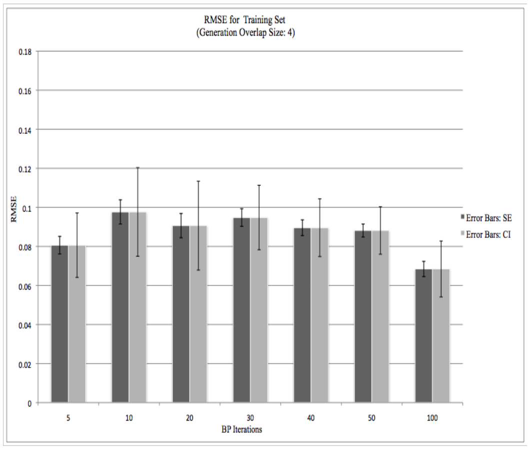


Figure 5.3: Root mean squared error of training set across various number of BP iterations, when the generation overlap size is 4. CI represents a 90% confidence Interval bar.

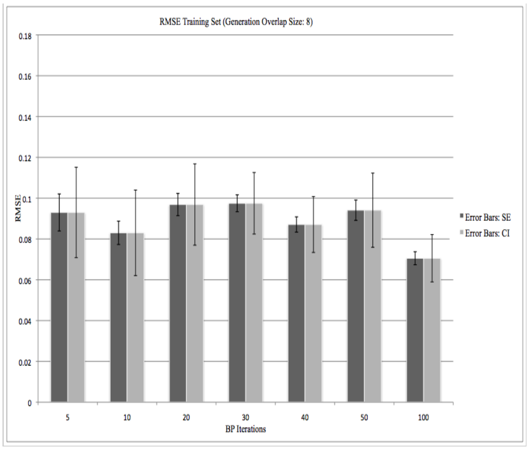


Figure 5.4: Root mean squared error of training set across various number of BP iterations, when the generation overlap size is 8. CI represents a 90% confidence Interval bar.

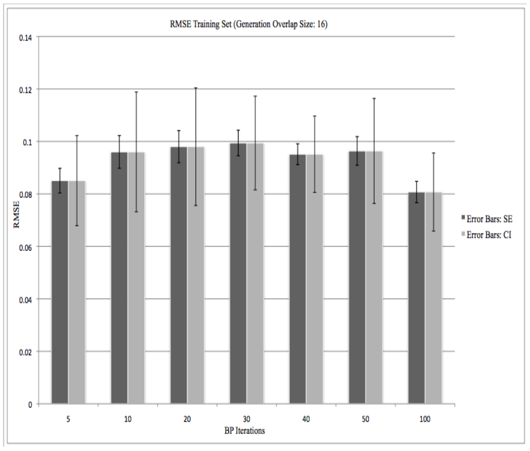


Figure 5.5: Root mean squared error of training set across various number of BP iterations, when the generation overlap size is 16. CI represents a 90% confidence Interval bar.

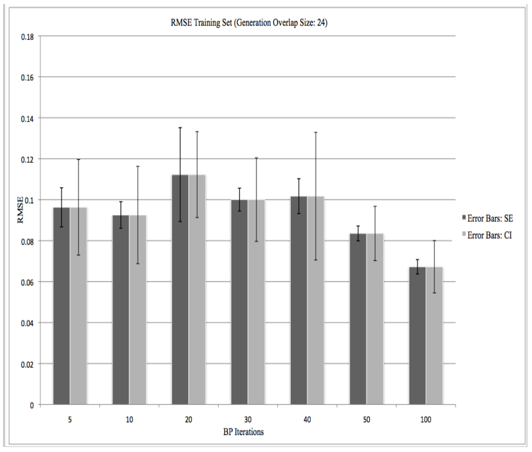


Figure 5.6: Root mean squared error of training set across various number of BP iterations, when the generation overlap size is 24. CI represents a 90% confidence Interval bar.

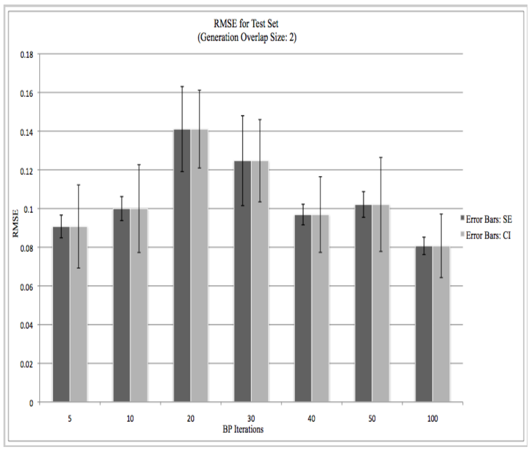


Figure 5.7: Root mean squared error of test set across various number of BP iterations, when the generation overlap size is 2. CI represents a 90% confidence Interval bar.

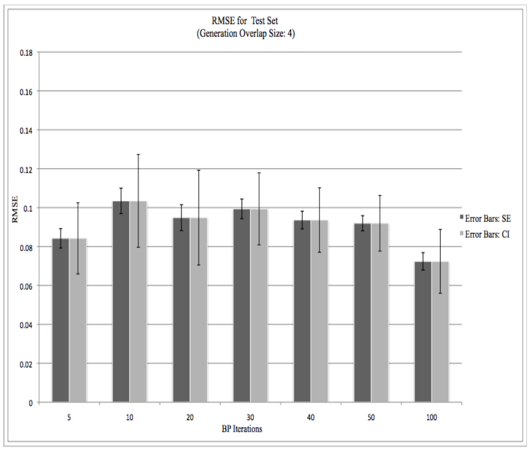


Figure 5.8: Root mean squared error of test set across various number of BP iterations, when the generation overlap size is 4. CI represents a 90% confidence Interval bar.

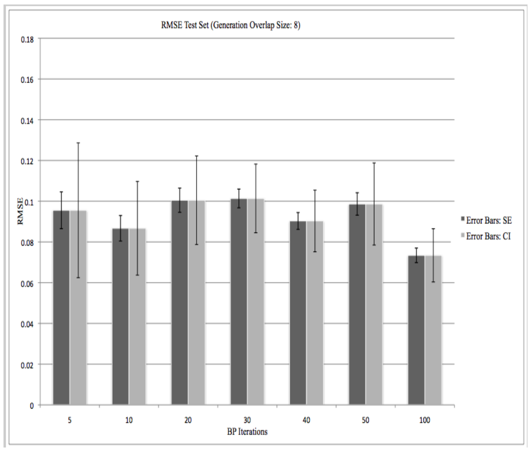


Figure 5.9: Root mean squared error of test set across various number of BP iterations, when the generation overlap size is 8. CI represents a 90% confidence Interval bar.

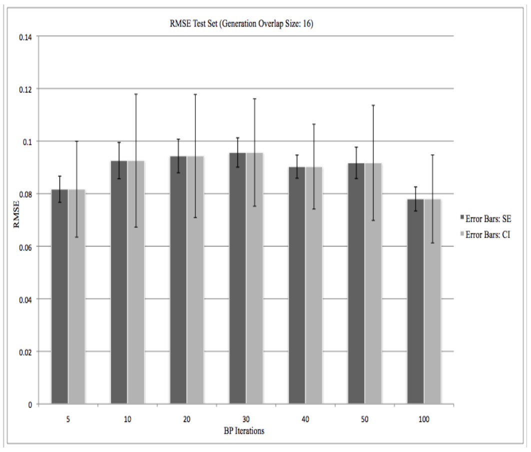


Figure 5.10: Root mean squared error of test set across various number of BP iterations, when the generation overlap size is 16. CI represents a 90% confidence Interval bar.

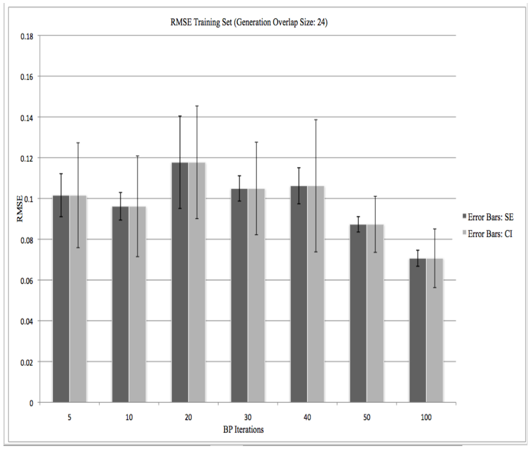


Figure 5.11: Root mean squared error of test set across various number of BP iterations, when the generation overlap size is 24. CI represents a 90% confidence Interval bar.

iments since, when applying the t-test, any significant difference can be observed by a higher level of confidence than what, otherwise, would be provided by fewer number of experiments.

5.1.4 Experimental Results of Settings

This section aims to reveal the optimum number of BP iterations and generation overlap size which statistically result the best performance on the carried out experiments. The focus is first on the suitable number of BP iterations and later on the generation overlap size. To this end, the RMSE, the number of distinct connectivity patterns and the number of final hidden nodes are observed when the number of BP iteration is changed across different values for generation overlap size. Table 5.1 lists most of the important settings of MINES used for all of the incoming experiments of this section.

Figures 5.2 to 5.6 show the RMSE of MINES on the training set, when converged on the polynomial regression problem. The figures represent the performance of the system in reducing the training error when the BP iteration is the changing parameter under consideration (the number of BP iteration is changing from 5 to 100 in each figure). To consider the probable influence of the generation overlap size, experiments are also run for generation overlap size of 2, 4, 8, 16 and 32. The figures reveal that, except for the BP iteration of 100, there is no significant difference in the ability of the system to reduce the training error across various generation overlap size. Considering the low values of RMSEs (about 0.1 on average), it can be claimed that MINES shows a good performance on approximating the underlying polynomial on the training set.

Similar to above, Figures 5.7 to 5.11 show the RMSE of MINES on the test set, while regressed on the polynomial regression problem. Again, it is observed that the overall generalization performance of the system is very good over all generation overlap sizes. It is also observed that, except for the BP iteration of 100, the RMSEs do not present a significant difference when the system is run for various number of BP iterations.

Since in all experiments the system is able to reduce the RMSE of the system significantly, the concentration is now shift to the ability of the system to reduce the number of hidden nodes. This is because the underlying polynomial can be regressed optimally with a NN with only two hidden nodes (see Figure 5.1). Hence, we may conclude that the fewer is the number of evolved hidden nodes, the more optimal is the system's architecture. The statement is however crude; although, it provides a simple indication to proceed our investigation. In fact, a NN with 3, 4 or even 5 hidden nodes could be as optimal as the one shown in Figure 5.1 depending on the way the receptive connections are formed. Nevertheless, it is very obvious that a NN with considerably more number of hidden nodes is far from an optimum

Population Size	No. of Generations	No. of Parents	No. of Offspring	Recombination Type	Mutation Rate	Clustering Type	No. of Clusters
32	200	2	5	2-point Crossover	1%	X-Means	Max=3

Table 5.1: Important Settings of Polynomial Regression Problem

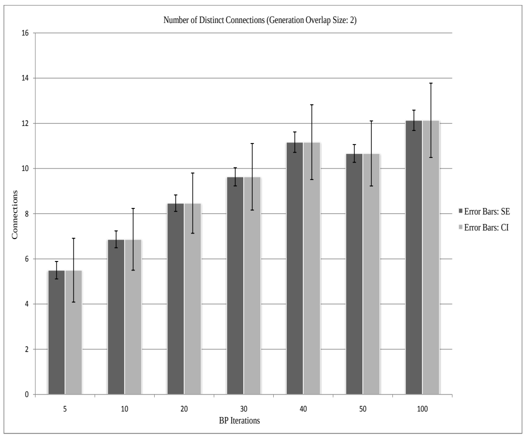


Figure 5.12: Distinct number of connectivity patterns across various number of BP iterations when the generation overlap size is 2. CI represents a 99.9% confidence Interval bar.

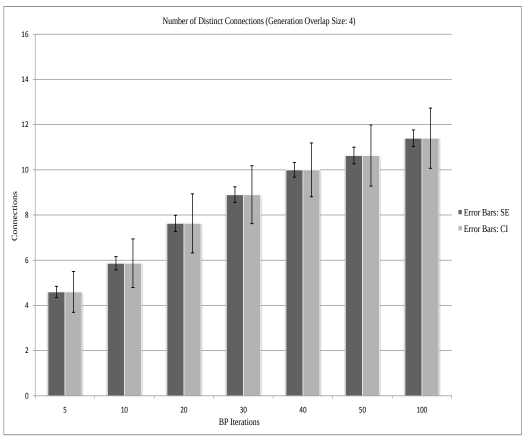


Figure 5.13: Distinct number of connectivity patterns across various number of BP iterations when the generation overlap size is 4. CI represents a 99.9% confidence Interval bar.

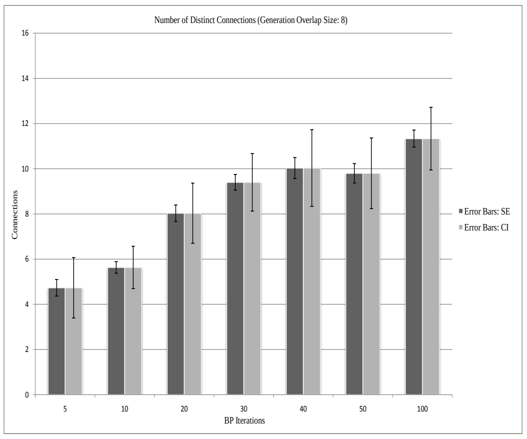


Figure 5.14: Distinct number of connectivity patterns across various number of BP iterations when the generation overlap size is 8. CI represents a 99.9% confidence Interval bar.

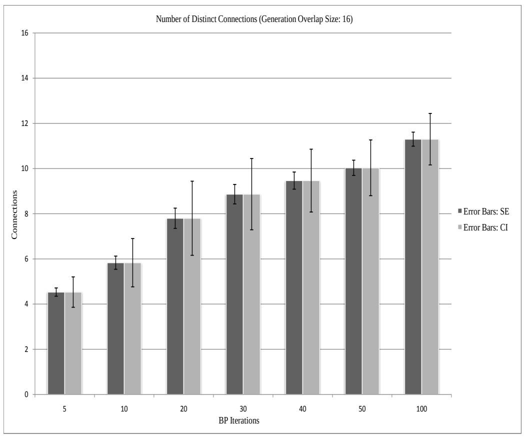


Figure 5.15: Distinct number of connectivity patterns across various number of BP iterations when the generation overlap size is 16. CI represents a 99.9% confidence Interval bar.

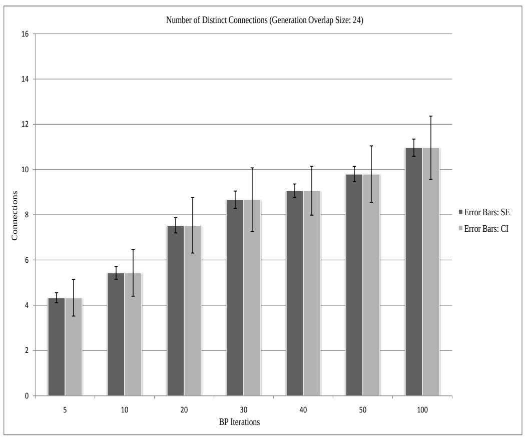


Figure 5.16: Distinct number of connectivity patterns across various number of BP iterations when the generation overlap size is 24. CI represents a 99.9% confidence Interval bar.

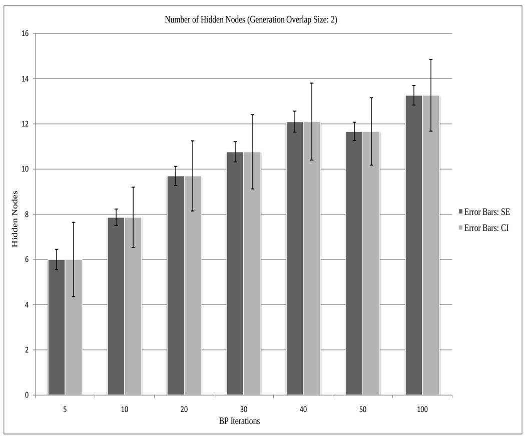


Figure 5.17: Number of obtained hidden nodes across various number of BP iterations when the generation overlap size is 2. CI represents a 99.9% confidence Interval bar.

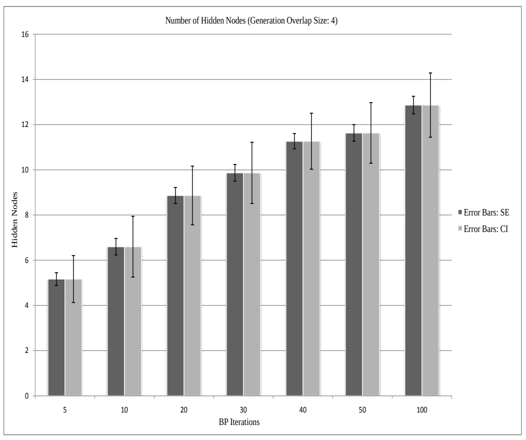


Figure 5.18: Number of obtained hidden nodes across various number of BP iterations when the generation overlap size is 4. CI represents a 99.9% confidence Interval bar.

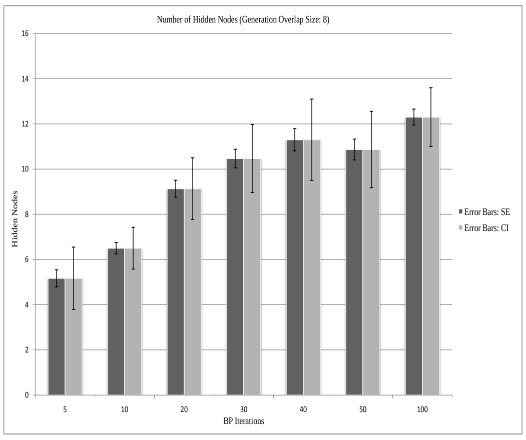


Figure 5.19: Number of obtained hidden nodes across various number of BP iterations when the generation overlap size is 8. CI represents a 99.9% confidence Interval bar.

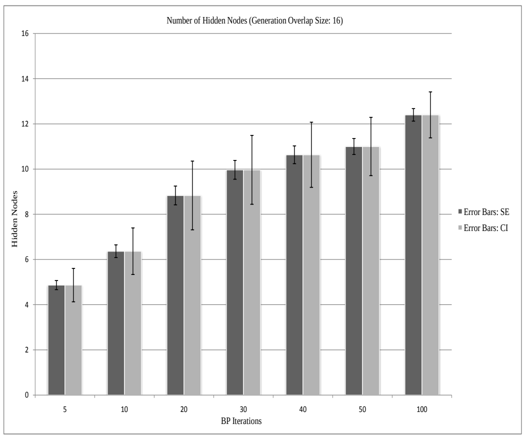


Figure 5.20: Number of obtained hidden nodes across various number of BP iterations when the generation overlap size is 16. CI represents a 99.9% confidence Interval bar.

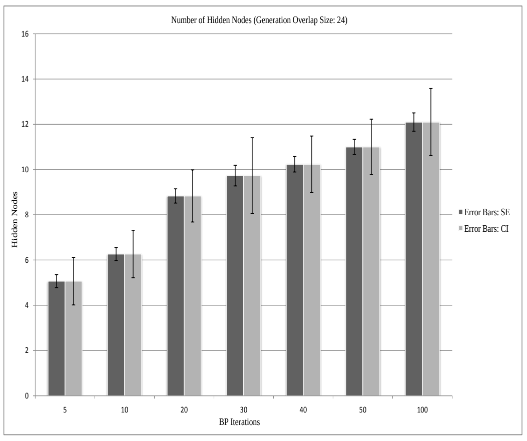


Figure 5.21: Number of obtained hidden nodes across various number of BP iterations when the generation overlap size is 24. CI represents a 99.9% confidence Interval bar.

configuration in comparison with the one in Figure 5.1.

Figures 5.12, 5.13, 5.14, 5.15 and 5.16 show the number of distinct connectivity patterns of the evolved network for various number of BP iterations when the generation overlap size is 2, 4, 8, 16, 24, respectively. Similarly, Figures 5.17, 5.18, 5.19, 5.20, 5.21 show the number of evolved hidden nodes for various number of BP iterations when the generation overlap size is 2, 4, 8, 16, 24, respectively. The figures are provided with the associated *standard errors* (SE) and the 99.9% *confidence interval* (CI) bars.

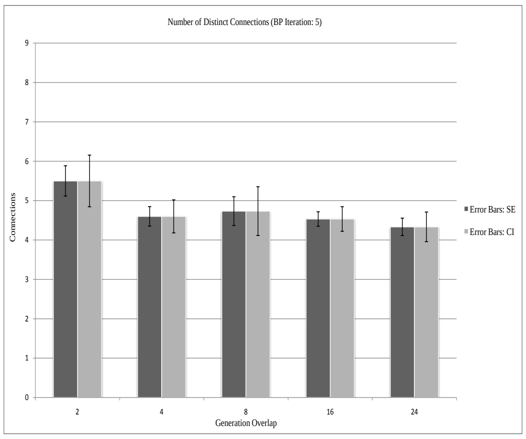
Wherever SE bars overlap, the two experiments are not significantly different. However, the converse is not necessarily true: when error bars do not overlap, it does not necessarily imply that experiments are significantly different. In this case, confidence intervals might provide more information: when confidence intervals do not overlap, the experiments are certainly significant. Again, the converse is not necessarily true: it is not valid to conclude that when CI bars overlap, there is no significant difference. Hence, when comparing standard error and confidence interval bars, no certain conclusion can be derived on situations where error bars do not overlap, but CI bars do.

Figures 5.12 to 5.16 and 5.17 to 5.21 indicate that, as the number of BP iteration goes up, there is almost a constant increasing trend in both the number of distinct connectivity patterns and the number of obtained hidden nodes, across all values of generation overlap size. In other words, the more is the number of BP iterations, the higher is the number of distinct connectivity patterns and the number of obtained hidden nodes; which is in line with our expectation explained earlier in Section 4.6. More specifically, the figures reveal that, across all values of generation overlap size, when 5 or 10 number of BP iterations are used, the number of distinct connectivity patterns and obtained hidden nodes are significantly different (with 99.9% confidence) from results corresponding to 30, 40, 50 and 100 number of BP iterations.

An important conclusion is that either 5 or 10 number of BP iterations should be used to obtain best results - in terms of lower number of distinct connectivity patterns and obtained hidden nodes - with respect to the underlying polynomial regression problem. Note that, it is not possible to statistically realize (at least with the aforementioned level of confidence) whether there is a significant difference between the results obtained when the number of BP iteration is 5 or 10; this is because, although the standard error bars do not overlap, the CI bars do.

So far, our findings indicate that when the number BP iterations is 5 or 10, MINES obtains the lowest number of distinct connectivity patterns and hidden nodes, over various values of generation overlap size for the regression problem outlined in Equation 3.15. Since the underlying polynomial can optimally be regressed with two hidden nodes, we may consider the aforementioned number of BP iterations the best, provided that the system also performs satisfactory on other aspects. For this reason, the error on the training and test sets, the convergence speed and the total run time of the system are also taken into account. Having fixed the number BP iterations on 5 or 10, the performance of the system will be assessed against the aforementioned factors, across various values for generation overlap size.

Figures 5.22 and 5.28 present the number of distinct connectivity patterns, over various values of



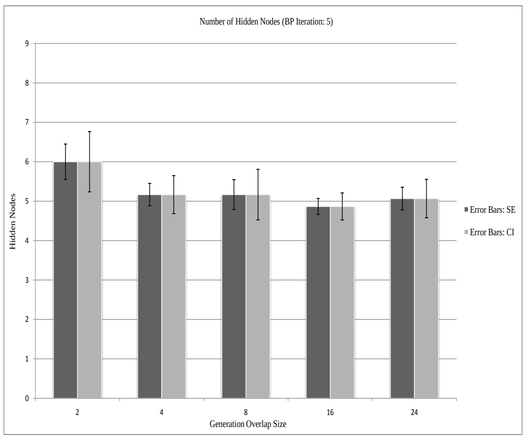
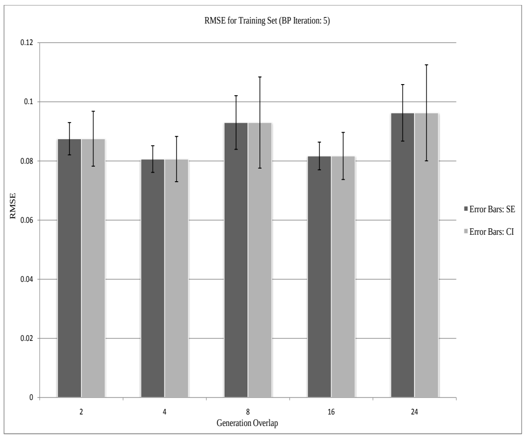


Figure 5.22: Number of distinct connectivity patterns across various number of generation overlap size when the BP iteration is 5. CI represents a 90% confidence Interval bar.

Figure 5.23: Number of obtained hidden nodes across various number of generation overlap size when the BP iteration is 5. CI represents a 90% confidence Interval bar.



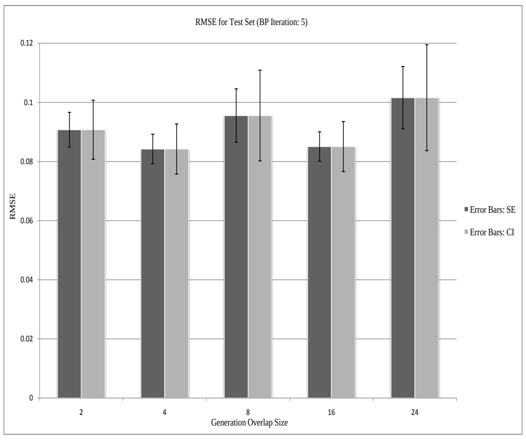
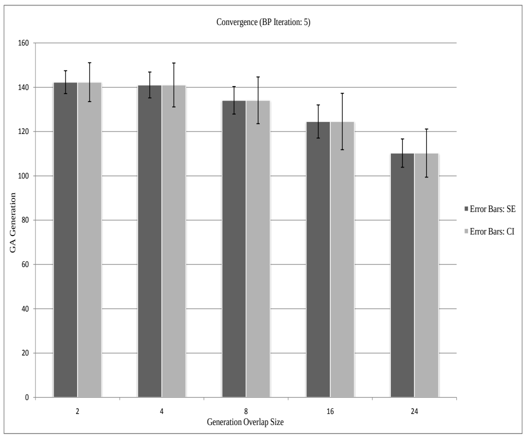


Figure 5.24: RMSE on the training set across various number of generation overlap size when the BP iteration is 5. CI represents a 90% confidence Interval bar.

Figure 5.25: RMSE on the test set across various number of generation overlap size when the BP iteration is 5. CI represents a 90% confidence Interval bar.



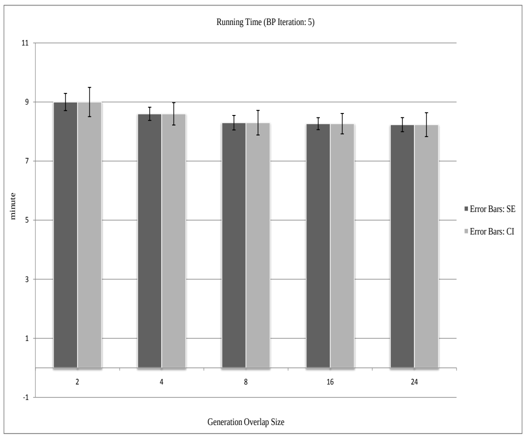


Figure 5.26: Convergence of MINES, measured based on the number of GA generations, across various number of generation overlap size when the BP iteration is 5. CI represents a 90% confidence Interval bar.

Figure 5.27: Run time required for the training of MINES for 200 cycles, across various number of generation overlap size when the BP iteration is 5. CI represents a 90% confidence Interval bar.

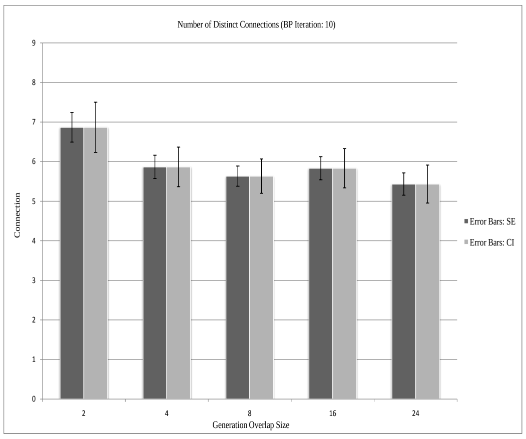


Figure 5.28: Number of distinct connectivity patterns across various number of generation overlap size when the BP iteration is 10. CI represents a 90% confidence Interval bar.

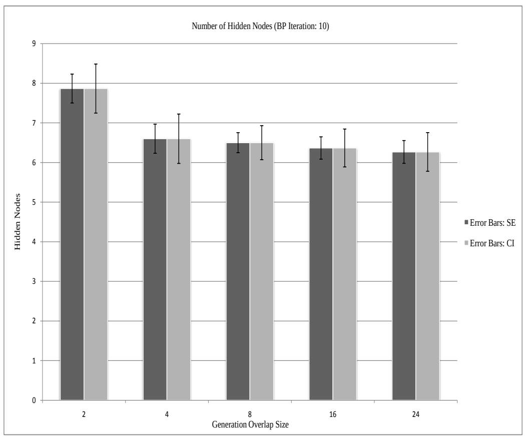


Figure 5.29: Number of obtained hidden nodes across various number of generation overlap size when the BP iteration is 10. CI represents a 90% confidence Interval bar.

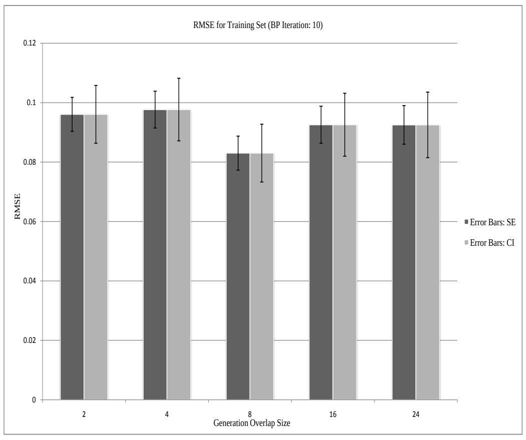


Figure 5.30: RMSE on the training set across various number of generation overlap size when the BP iteration is 10. CI represents a 90% confidence Interval bar.

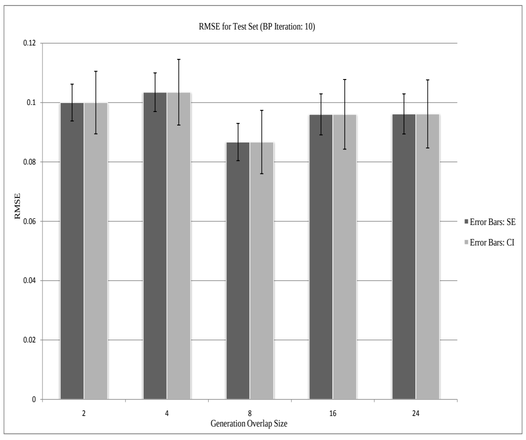


Figure 5.31: RMSE on the test set across various number of generation overlap size when the BP iteration is 10. CI represents a 90% confidence Interval bar.

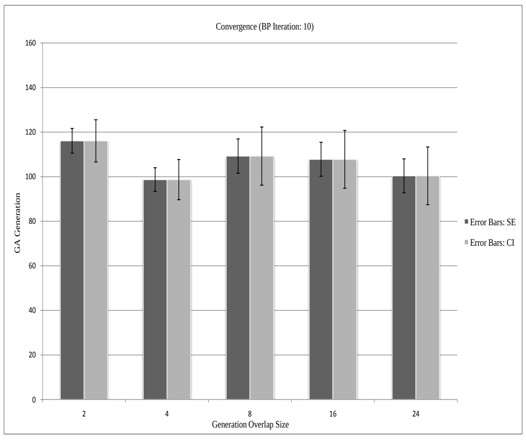


Figure 5.32: Convergence of MINES, measured based on the number of GA generations, across various number of generation overlap size when the BP iteration is 10. CI represents a 90% confidence Interval bar.

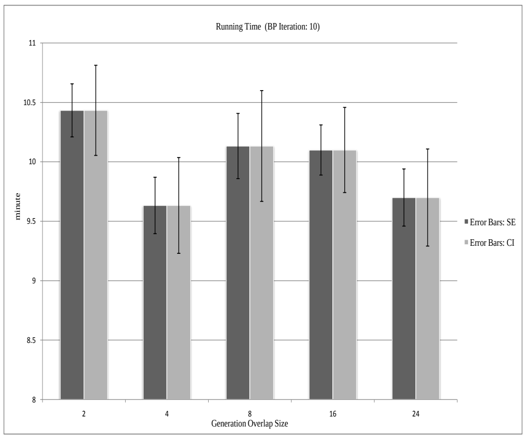


Figure 5.33: Run time required for the training of MINES for 200 cycles, across various number of generation overlap size when the BP iteration is 10. CI represents a 90% confidence Interval bar.

generation overlap size, when the number of BP iteration is 5 and 10, respectively. The CI bars represent a 90% level of cofidence for all the following represented figures. The presented figures indicate that, for both 5 and 10 number of BP iterations, the number of distinct connectivity patterns is significantly different when the generation overlap size is 24. On the other hand, Figure 5.29 additionally indicates that, when the number of BP iterations is 10, the number of hidden nodes is significantly different when the genreation overlap size is 24. However, this may not be the case when the number of BP iterations is equal to 5 as the CI bars overlap. Overal, we may conclude that the genreation overlap size of 24 is the best choice since in both cases (i.e. 5 and 10 number of BP iterations) it provides the least number of distinct connectivity patterns (though it may not necessarily result a significantly least number of hidden nodes).

Another important factor to decide the proper number of generation overlap size is the performance of the system on the unseen data. For this purpose, Figure 5.25 and 5.31 have been represented. The figures, however, do not present significant differences across different values for generation overlap size. Similarly, Figures 5.24 and 5.30 provides no indication to confirm that the performance of the system is better on the training set with respect to the experimented values of generation overlap size.

The convergence of the system may also be of interest when considering various number of generation overlap sizes. Figures 5.26 and 5.32 provide results in this regard. Figure 5.26 indicates that when the number of generation overlap size is 24, the system converges in earlier generations. However, the result is different when the number of BP iterations is 10. Figure 5.32 does not represent any significant difference among values of generation overlap size.

Figures 5.27 and 5.33 represent the total run time required for 200 training cycles, succeeding by 1000 continuous BP iterations in the final generation, using an Intel(R) Pentium(R) D CPU 2.80GHz processor equipped with 1024 MB RAM. No difference can be observed in relation to any value for generation overlap size, when the number of BP iterations is 5. When the number of BP iterations is 10, the generation overlap size of 4 requires the least run time.

As a final conclusion we may comment that the results obtained over different values for generation overlap size are mixed. The differences, however, are not extremely crucial since the results are all sufficiently acceptable. Nevertheless, to obtain the least number of distinct connectivity patterns and hidden nodes, the generation overlap size of 24 is more advisable to be used.

Figures 5.34 and 5.35 show an example graph of evolutionary learning process of MINES for the polynomial regression problem when the number of BP iteration after each generation is 5 and the generation overlap size is 2. Figure 5.34 shows the RMSE on the training and test set while the system is evolving for 200 GA generations. It can be seen that the RMSE has an overall declining trend, while sometimes in some generations there are jumps in the RMSEs. The jumps in the RMSEs are usually due to change in the number of connections or hidden nodes imposed by the evolution. The figure reveals that RMSEs on the training and test sets monotonically declines after generation 150. The reason for this monotonic decline can be explained by looking at Figure 5.35, which presents the effect of MINES evolution on the number of distinct connections and number of hidden nodes of the evolving FFNN.

The figure indicate that the system is converged after generation 150. Hence, there is no change in the number of connections and hidden nodes from generation 150 to 200. This stability in the structure of evolving FFNN makes the system reduce error using BP, while there is little distortion on synaptic weights after each GA evolution.

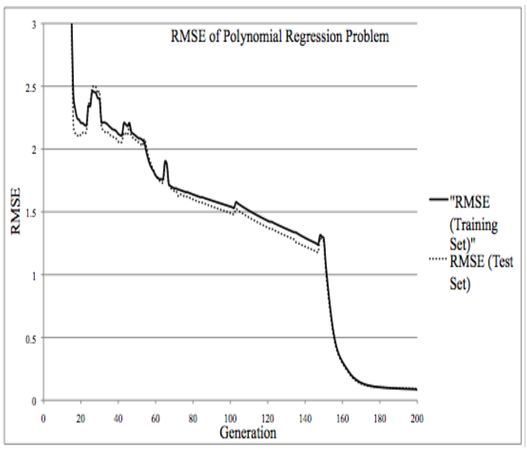


Figure 5.34: RMSE on training and test sets when the BP iteration is 5 and the generation overlap size is 2.

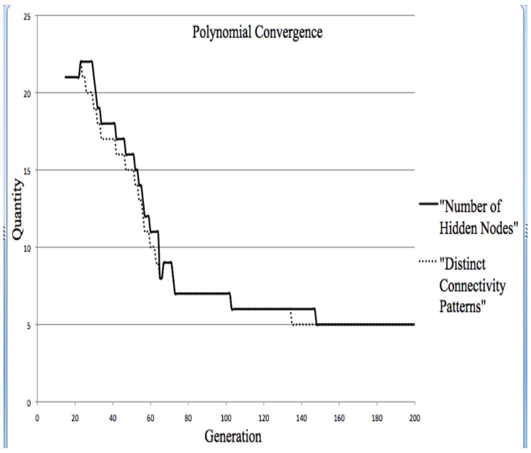


Figure 5.35: Convergence of MINES to 5 hidden nodes when the number of BP iteration is 5 and the generation overlap size is 2.

5.1.5 Optimum Polynomial Results

The aim of this section is to explore the ability of MINES in discovering a close to optimal NN structure regressing the underlying polynomial. As Figures 5.22 and 5.28 reveal, the number of obtained hidden nodes varies over the 30 number of carried out experiments. However, to observe the ability of the system in discovering the polynomial an obtained result with two hidden nodes has been selected to compare with the optimum NN depicted in Figure 5.1.

The obtained connectivity patterns and synaptic weights are presented here in a similar fashion used in Equations 5.5 and 5.6:

$$neuron\,1: \begin{cases} bias: \text{no connection (an absolute 0 weight)} \\ hidden\ node\,1: \begin{cases} w_{11} = 0.9656 \\ w_{12} = 0.9656 \\ w_{13} = 0.0132 \\ w_{14} = 0.0129 \\ w_{15} = 0.0129 \end{cases} \\ hidden\ node\,2: \begin{cases} w_{21} = -0.0075 \\ w_{22} = -0.0074 \\ w_{23} = 1.0057 \\ w_{24} = 1.0056 \\ w_{25} = 1.0056 \end{cases} \end{cases} \tag{5.7}$$

$$neuron\,2: \begin{cases} bias: w_0 = 1.7550 \\ w_1 = 3.2261 \\ w_2 = 3.9447 \end{cases} \tag{5.8}$$

In compact form, Equations 5.7 and 5.8 result the following polynomial:

$$\begin{aligned} y &= 1.7550 \\ &+ 3.2261{x_1}^{0.9656}{x_2}^{0.9656}{x_3}^{0.0132}{x_4}^{0.0129}{x_5}^{0.0129} \\ &+ 3.9447{x_1}^{-0.0075}{x_2}^{-0.0074}{x_3}^{1.0057}{x_4}^{1.0056}{x_5}^{1.0056} \end{aligned} \tag{5.9}$$

When weights in Equations 5.7 and 5.8 are rounded to no decimal place, Equation 5.9 is transformed to the exact desired polynomial represented in Equation 3.15.

The RMSE on the training set and the test set are 0.0202 and 0.0190, respectively, for the evolved NN above. Figure 5.36 demonstrates the declining trend of RMSE of the NN over training and test sets. The figure shows that before generation 130 sometimes there is a jump in the RMSE in some generations. The jump in the RMSE usually happens due to a change in the number of hidden nodes after some GA generations when new connectivity patterns are created in the population. The adjustment of the synaptic weights after the clustering of weight vectors may also raise the RMSE in some generations. It is observed that, after the generation 130, when the number of hidden nodes remains still (see Figure 5.37), the RMSEs on the training and test sets decline constantly without being raised considerably in any cycle.

Figure 5.37 demonstrates the converging trend of MINES for the polynomial regression problem. As it can be seen, the general trend is declining for the quantity of the distinct connectivity patterns and

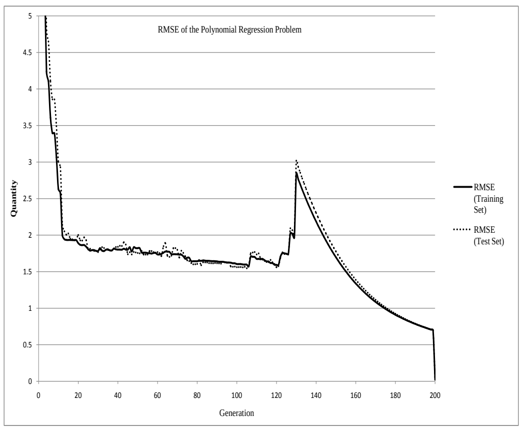


Figure 5.36: Declining trend of RMSE on the training and test sets on the polynomial regression problem. There is a steady decline after the convergence of the system on generation 130.

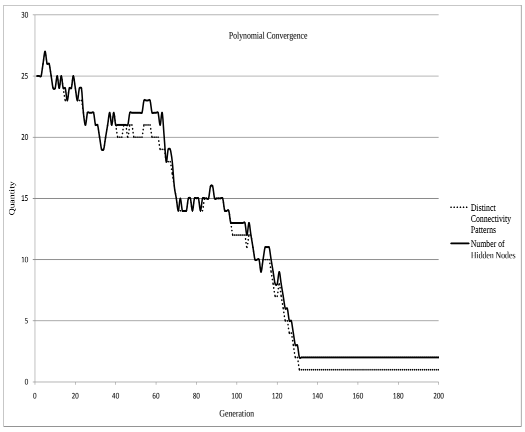


Figure 5.37: Convergence of MINES to 1 distinct connectivity pattern and 2 hidden nodes after 130 GA generations on the polynomial regression problem.

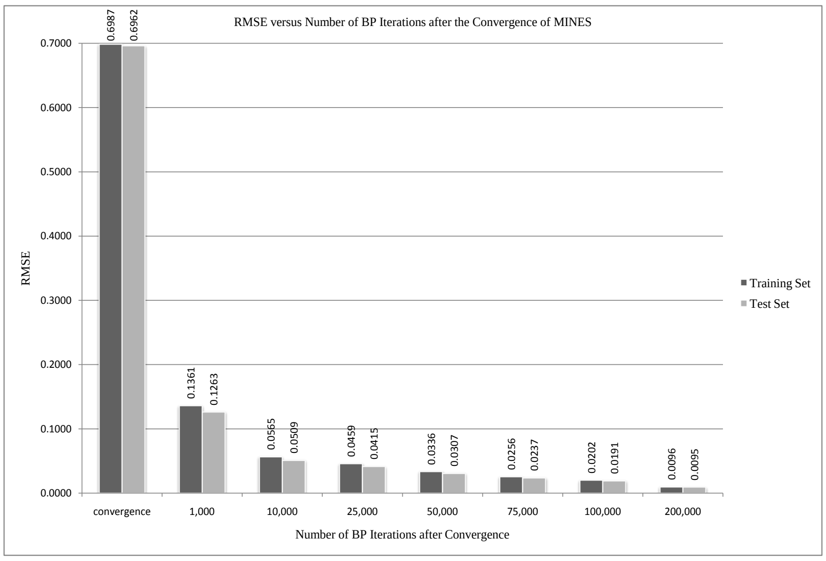


Figure 5.38: RMSE on the training and the test sets for different number of BP iterations when the system is converged after 200 GA generations.

the incorporated hidden nodes, although some minor jumps is observed before the convergence due to the clustering and the GA evolution. However, after the generation 130, when the system is converged, there is absolutely no change either in the number of distinct connectivity patterns or the number of hidden nodes for over 70 generations. The latter reveals the robust capability of MINES in converging towards a fixed number of hidden nodes that can be sustained for many number of generations.

Figure 5.38 represents the RMSE of the above NN on the training and test set when the system has gone through the parameter learning for different number of BP iterations after the convergence in generation 200. The figure reflects a very interesting feature: the system does not overfit, on the underlying experiment, regardless of the length of the applied parameter learning after the convergence. This is an interesting effect since it is often expected that conventional NNs perform worse on the test set when the error on the training set comes lower than a certain threshold.

Experiment	Hidden Nodes	BP Iterations	RMSE (Training Set)	RMSE (Test Set)
1	3	convergence	0.4135	0.4135
		$1,000$	0.1251	0.1272
		$50,000$	0.0288	0.0279
		$100,000$	0.0172	0.0171
2	3	convergence	0.6160	0.6413
		$1,000$	0.2693	0.2952
		$50,000$	0.0613	0.0634
		$100,000$	0.0346	0.0356
3	4	convergence	0.6155	0.5931
		$1,000$	0.2079	0.2234
		$50,000$	0.0315	0.0339
		$100,000$	0.0254	0.0271
4	5	convergence	0.8118	0.7912
		$1,000$	0.2675	0.2847
		$50,000$	0.0155	0.0162
		$100,000$	0.0112	0.0115
5	5	convergence	0.9035	0.8602
		$1,000$	0.3500	0.3794
		$50,000$	0.0330	0.0342
		$100,000$	0.0263	0.0275

Table 5.2: RMSEs on training and test sets for 5 random experiments.

5.1.6 Comparison of the Obtained Results

The carried out experiments obtained by MINES are compared to those resulted from BCW introduced by Saito and Nakano[56]. The comparison, however, is highly superficial as the natural objectives of the two systems are in essence completely different.

In BCW the number of incorporated hidden nodes (denoted by "J") must be defined in advance. For example, in the application of polynomial discovery, the number of hidden nodes was manually set to 2 (i.e. J=2). BCW does not offer any new technique to estimate the number of required hidden nodes; the authors suggest "employing a model selection technique such as cross-validation" to find the optimal number of hidden nodes. On top of that, even the optimal number of clusters, denoted by G^*, must be defined beforehand. G^* represents the total number of "common weights" used in the weight sharing process. For instance, for the underlying polynomial (Equation 3.15) the "true G^*" is 2 since only two common weight values of 0 and 1 can completely result the exact polynomial. Since the "optimal G^*" is not known "in advance", BCW suggests that the "predetermined number of clusters", G^*, be found by trying different number of clusters, G, that "minimizes the generalisation error". For this reason, various values of G (2, 3, 5 and 10) were experimented to reveal their performance on the training and test sets. The RMSE on the training data showed a declining trend, in order for 3, 5 and 10 number of clusters, while, in the same order, increases on the test set. The latter was used as a justification of adopting two clusters for the underlying experiment since the RMSE on the test was lowest when $G = 2$. The justification, however, may be unreliable as the experiments do not provide any test of significance in spite of dealing with very close numbers: the RMSEs on the test set reported to be 0.0206, 0.0373 and 0.0384, respectively, for 3, 5 and 10 clusters. In other words, it appears that only one experiment has been carried out for each experimental number of clusters; as if the system is not sensitive to any type of settings, such as the initial randomly created weights and the parameters involved in the "retraining". Further experiments, however, reveal that when $G^* = 2$, the BCW provides the same level of generalization performance over various experiments; the RMSE on the test set is 0.012 for all the 5 experimented run, although it is not clear that what kind of variables are initialized in each run. The training iteration is terminated when each element of the gradient vector is less than 10^{-6}.

MINES, in contrast to BCW, automatically determines the number of hidden nodes. However, since it partly relies on an evolutionary system to explore the connectivity patterns, the number of obtained hidden nodes is not always fixed using different settings. Therefore, to compare the performance of MINES with the BCW, the same NN used in Section 5.1.5 has been used here. The RMSEs on the test set is 0.0190 when $100,000$ number of BP iterations was used after the convergence of the system. Not only the performance does not show a significant difference with the one reported by BCW, but also it can be further improved if additional BP training is applied. However, the latter comes with the cost of extra run time. MINES could improve the generalization performance to 0.0094, when $200,000$ number of BP iterations are used. The performed experiments do not show any sign of overfitting for the underlying polynomial regression problem, suggesting that even better performance is likely to achieve by continuing further BP training.

The total run time for each experiment performed by BCW was about 40 minutes using a 2 GHz Pentium. In MINES, in contrast, the training procedure takes only about 9 to 10 minutes for 200 cycles (see Figures 5.27 and 5.33), using a 2.8 GHz Pentium . In terms of convergence, the required time is even less as the system converges at only about 100 to 110 generations. However, as stated in Section 5.1.3, the final number of BP iterations after the last generation was equal to 1000 for the presented experiments. Nevertheless, this number can increase to provide a better performance which in return the run time is extended. For $100,000$ number of BP iterations the total run time would increase to about 45 minutes which is still acceptable for the very good performance that the system provides. This increases to about 90 minutes when $200,000$ number of BP iterations are used.

It can be argued that the above results are only valid for the obtained NN with 2 hidden nodes and other networks evolved with different number of hidden nodes may show a considerably worse performance. The argument, however, does not seem to be in line with the experiments presented in Section 5.1.5, implying that MINES is a very stable system on the underlying regression problem. Therefore, we may expect the system to be robust against overfitting, on the experimented polynomial problem, regardless of the number of hidden nodes. In other words, it would be expected that, when the system is converged, reducing the RMSE on the training set provides a better generalization performance on the test set.

Table 5.2 confirms the above claim by reducing the RMSE on the training set for 5 randomly evolved NN by MINES - the generation overlap size is 24 while the number of BP after each generation is set to 5; other variables are exactly the same as those reported in Section 5.1.3. It is observed that the number of hidden nodes vary from 3 to 5. After the convergence, the error on the training set has been reduced for $1,000$, $50,000$ and $100,000$ number of iterations. Results on the test set indicate that there is a constant improvement in the generalization performance when the error on the training set is reduced. The latter may suggest even better performance is able to achieve subject to lowering the training error sufficiently.

5.1.6.1 Comparison to CCNN

As mentioned in the literature and in the beginning of this chapter, MINES has a close analogy with the CCNN: both systems follow a dynamic structural and parameter learning procedures in which the interrelationship (correlation in CCNN and MI in MINES) of the output of a focused hidden node and the respective residual error is the main factor making the system converge. Therefore, we have selected CCNN to compare the performance of MINES with one of the alternative systems described in the literature.

Statistic	RMSE of Training Set	RMSE of Test Set	No. of Layers	No. of Nodes	Run Time (min)
mean	1.4858	1.4835	83	170	5
standard deviation	0.3530	0.3723	3	8	1

Table 5.3: CCNN Results on the Polynomial Regression Problem.

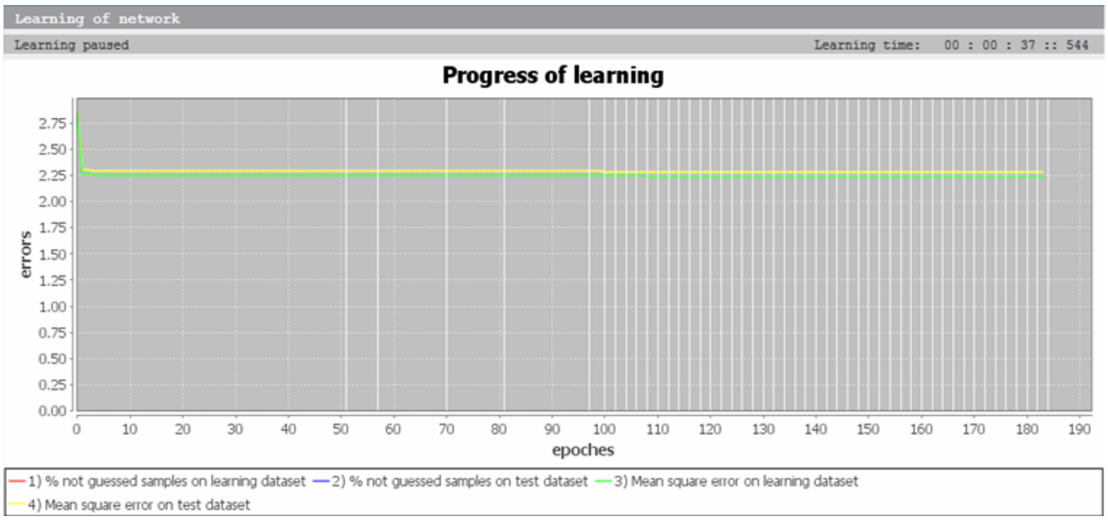
The CCNN software can be found in [68], which is the world's largest provider of hosting for open source software development projects. The experiments presented here are the best obtained by changing all the involved parameters listed in Figure 5.41. There are two sets of settings: default settings and those which have been used to run the system.

Statistic	RMSE of Training Set	RMSE of Test Set	No. of Nodes	Run Time (min)
mean	0.6721	0.8602	4	15
standard deviation	0.1912	0.6560	1	3

Table 5.4: MINES Results on the Polynomial Regression Problem for 5 random experiments extracted from Table 5.2

.

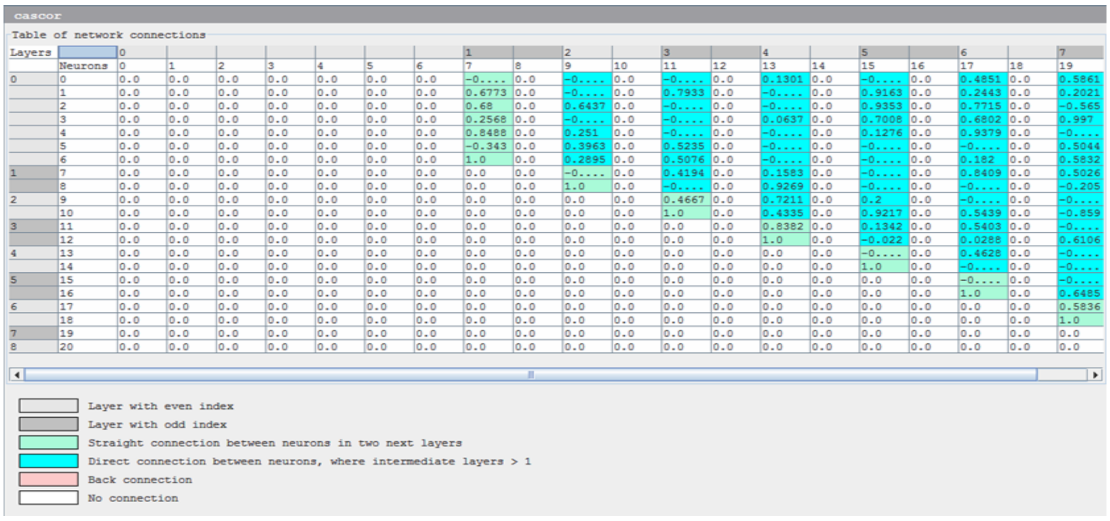
Figure 5.40 shows how MSE of training and test sets are lowered when the CCNN is under learning



Learning of network

Learning paused Learning time: 00 : 00 : 37 :: 544

Figure 5.39: The trend of MSE of the CCNN for training and test sets when applied on the polynomial regression problem.



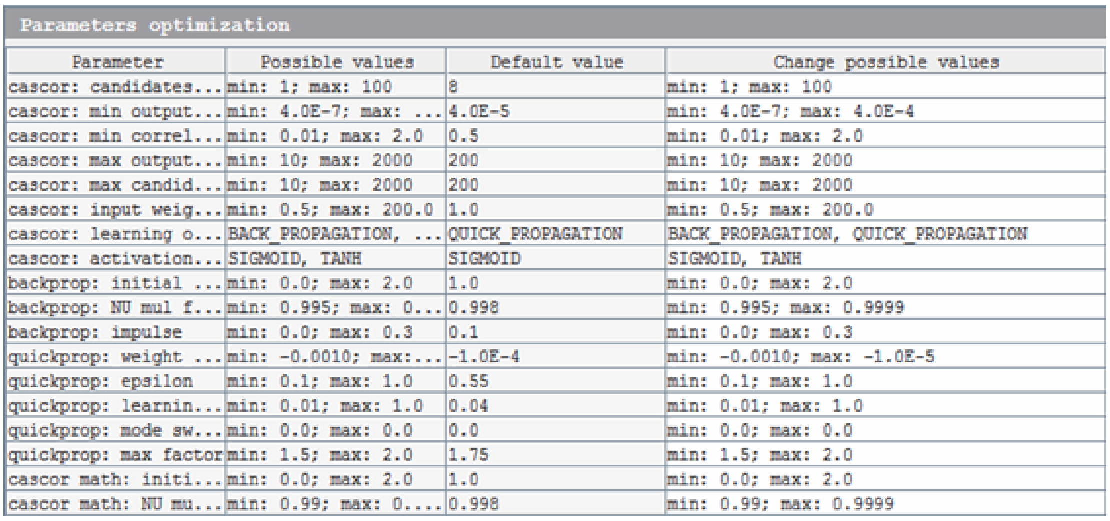
cascor

Table of network connections

Layers		0							1		2		3		4		5		6		7
	Neurons	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0....	0.0	-0....	0.0	-0....	0.0	0.1301	0.0	-0....	0.0	0.4851	0.0	0.5861
	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6773	0.0	-0....	0.0	0.7933	0.0	-0....	0.0	0.9163	0.0	0.2443	0.0	0.2021
	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.68	0.0	0.6437	0.0	-0....	0.0	-0....	0.0	0.9353	0.0	0.7715	0.0	-0.565
	3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2568	0.0	-0....	0.0	-0....	0.0	0.0637	0.0	0.7008	0.0	0.6802	0.0	0.997
	4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8488	0.0	0.251	0.0	-0....	0.0	-0....	0.0	0.1276	0.0	0.9379	0.0	-0....
	5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.343	0.0	0.3963	0.0	0.5235	0.0	-0....	0.0	-0....	0.0	-0....	0.0	0.5044
	6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.2895	0.0	0.5076	0.0	-0....	0.0	-0....	0.0	0.182	0.0	0.5832
1	7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0....	0.0	0.4194	0.0	0.1583	0.0	-0....	0.0	0.8409	0.0	0.5026
	8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	-0....	0.0	0.9269	0.0	-0....	0.0	-0....	0.0	-0.205
2	9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4667	0.0	0.7211	0.0	0.2	0.0	-0....	0.0	-0....
	10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.4335	0.0	0.9217	0.0	0.5439	0.0	-0.859
3	11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8382	0.0	0.1342	0.0	0.5403	0.0	-0....
	12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	-0.022	0.0	0.0288	0.0	0.6106
4	13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0....	0.0	0.4628	0.0	-0....
	14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	-0....	0.0	-0....
5	15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0....	0.0	-0....
	16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.6485
6	17	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5836
	18	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
7	19	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

⬜	Layer with even index
▩	Layer with odd index
🟩	Straight connection between neurons in two next layers
🟦	Direct connection between neurons, where intermediate layers > 1
🟥	Back connection
⬜	No connection

Figure 5.40: The architecture of CCNN when applied on the polynomial regression problem. Note that this is a snapshot structure at the middle of learning process and not the structure at the convergence.



Parameters optimization

Parameter	Possible values	Default value	Change possible values
cascor: candidates...	min: 1; max: 100	8	min: 1; max: 100
cascor: min output...	min: 4.0E-7; max: ...	4.0E-5	min: 4.0E-7; max: 4.0E-4
cascor: min correl...	min: 0.01; max: 2.0	0.5	min: 0.01; max: 2.0
cascor: max output...	min: 10; max: 2000	200	min: 10; max: 2000
cascor: max candid...	min: 10; max: 2000	200	min: 10; max: 2000
cascor: input weig...	min: 0.5; max: 200.0	1.0	min: 0.5; max: 200.0
cascor: learning o...	BACK_PROPAGATION, ...	QUICK_PROPAGATION	BACK_PROPAGATION, QUICK_PROPAGATION
cascor: activation...	SIGMOID, TANH	SIGMOID	SIGMOID, TANH
backprop: initial ...	min: 0.0; max: 2.0	1.0	min: 0.0; max: 2.0
backprop: NU mul f...	min: 0.995; max: 0...	0.998	min: 0.995; max: 0.9999
backprop: impulse	min: 0.0; max: 0.3	0.1	min: 0.0; max: 0.3
quickprop: weight ...	min: -0.0010; max:...	-1.0E-4	min: -0.0010; max: -1.0E-5
quickprop: epsilon	min: 0.1; max: 1.0	0.55	min: 0.1; max: 1.0
quickprop: learnin...	min: 0.01; max: 1.0	0.04	min: 0.01; max: 1.0
quickprop: mode sw...	min: 0.0; max: 0.0	0.0	min: 0.0; max: 0.0
quickprop: max factor	min: 1.5; max: 2.0	1.75	min: 1.5; max: 2.0
cascor math: initi...	min: 0.0; max: 2.0	1.0	min: 0.0; max: 2.0
cascor math: NU mu...	min: 0.99; max: 0....	0.998	min: 0.99; max: 0.9999

Figure 5.41: The parameters used for the CCNN when applied on the polynomial regression problem.

process. As can be seen from Table 5.3, the average RMSE on training and test sets are 1.4858 and 1.4835, respectively. The results are significantly worse than what was obtained by MINES, listed in Table 5.4, representing an average RMSE of 0.6721 on the training set and 0.8602 on the test set. Note that in in Table 5.4 the system does not even go under fine tuning of weights after convergence.

Figure 5.40 illustrates the architecture of the CCNN when applied on the polynomial regression problem. The architecture shows the number of layers, nodes (neurons) and synaptic weights of the CCNN in the middle of learning. The final number of layers and nodes at the convergence have been reported in Table 5.3. When compared to Table 5.4, it is observed that CCNN on average requires 170 nodes to solve the problem, while MINES only needs 4 hidden nodes incorporated in a single layer.

5.2 Preliminary Testing on Classification Problem

To test the performance of MINES on a classification task, the two-spiral problem has been selected. The problem is a quite well-known and challenging classification task used by researches for over 20 years [6] "as a benchmark for NNs" [34]. It requires the network to learn a very complicated nonlinear mapping to distinguish points on the two intertwined spirals. There are claims that a standard FFNN with BP algorithm is unable to solve the problem, unless it is modified with query learning [2] or short-cut connections providing "direct information pathway" [37]. The problem, proposed first by Alexis Wieland in MITRE Corporation [58], was historically used as a benchmark to extensively test the performance of the CCNN [13][14] being a novel system at the time. The evaluation of the CCNN on the two-spiral problem also "inspired a long sequence of follow-up studies by other researchers" [6]. The researches conducted by Shultz and Elman in 1994, Shultz et al. in 1995, Setiono in 1995, Teng and Wah in 1996, Hwang et al. in 1996, Treadgold and Gedeon in 1997 and in 1999, Chen et al. in 1997, Potter and De Jong in 2000, Su and Guan in 2000, Lahnajrvi et al. in 2002 and Thangavel and Kathirvalavakumar in 2003 compared the performance of their designed system to the CCNN applied to the two-spiral problem [6]. An additional example is the Genetic Cascade-Correlation Learning Algorithm [49] surveyed in detail in Section 3.2.1.

The conceptual resemblance of the CCNN and MINES in concentrating on the prediction of the expected payoff by aligning the output of hidden nodes with the corresponding residual errors makes the two-spiral problem to be a natural choice for evaluating the classification performance of MINES.

In this section, first, the mathematical formula required to create data points of the two-spiral problem will be derived completely. Subsequently, it will be shown that MINES is able to classify the two spirals with an astonishing accuracy both on the training and test sets.

5.2.1 Spiral Formula

An Archimedean spiral, as described by Hansen [24], is a curve in the x-y plane being the result of the motion of a point P, rotating with a constant velocity, about a fixed point O ; while simultaneously moving constantly away from O. Without loss of generality, O can be considered to be the origin of the x-y plane, and P can be assumed as a vector in the same plane with the starting point O and the end point P. Suppose, θ and r are the Euclidean angle and the length of the vector P, respectively; defined, naturally, in the x-y plane. Since, by definition, the angular and the linear velocity of P are constant, θ and r must vary linearly through time. Hence, by considering t to be an arbitrary time unit, the following equations must hold:

$$\theta = \alpha + \beta t \tag{5.10}$$

$$r = \alpha' + \beta' t \tag{5.11}$$

where α, α', β, β' are some arbitrary constants. By eliminating t from Equations 5.10 and 5.11 a new formula is obtained directly defining r based on θ:

$$r = (\alpha' - \alpha\frac{\beta'}{\beta}) + (\frac{\beta'}{\beta})\theta \tag{5.12}$$

By defining the following new constants:

$$a = \alpha' - \alpha\frac{\beta'}{\beta} \tag{5.13}$$

$$b = \frac{\beta'}{\beta} \tag{5.14}$$

the formula 5.12 becomes more compact:

$$r = a + b\theta \tag{5.15}$$

In Equation 5.15, θ is considered as the independent parameter resulting the dependent parameter r. Suppose that the coordinates of the point P, in the x-y plane, is (x, y). The polar coordinates of x and y is, then, defined based on r and θ:

$$x = r\cos\theta \tag{5.16}$$

$$y = r\sin\theta \tag{5.17}$$

where θ must be defined in radian. Equations 5.16 and 5.17 make possible plot an spiral in the x-y plane: different values of θ provides the points of the plotted spiral. The density of the points can increase or decrease by incrementing or decrementing the angle, θ, respectively. When θ is positive, assuming b is always positive, the spiral's rotation is anticlockwise; while when it is negative the direction of rotation is clockwise. In order to change the starting point of the spiral a phase difference, ω, must be added to the angle:

$$r = a + b(\theta + \omega) \tag{5.18}$$

Equation 5.18 is a more general from of Equation 5.15; the sign of ω determines the angular direction that the starting point is shifted towards: positive and negative ω makes anticlockwise and clockwise shifts, respectively. Assuming the starting point of the spiral is (-α, 0) and the spiral intersects the x axis for the second time at the point (β, 0), a phase difference equal to -π shifts the starting point to (β, 0). To better understand the role of a, in Equations 5.15 and 5.18, it should be noted that a positive a shifts the whole spiral to the left of the plane, whereas a negative sign makes a shift to the right. Using Equations 5.13 and 5.14, the derivative of r with respect to θ can be obtained from 5.15 or 5.18:

$$\frac{dr}{d\theta} = b \tag{5.19}$$

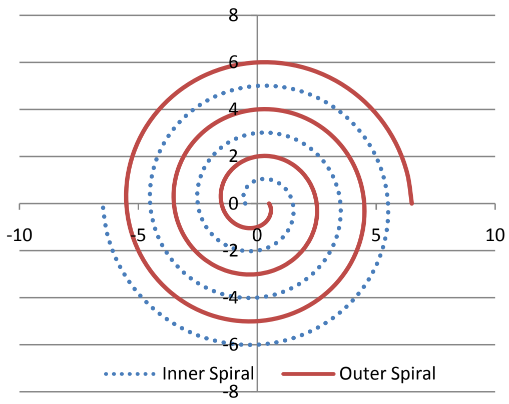


Figure 5.42: Classical two intertwined spiral problem consisting of 97 points on each spiral. The spirals rotate 3 complete times around the origin and π Radians relative to each other

Equation 5.19 makes possible explain the role of b in Equations 5.15 and 5.18: b is a constant which determines how fast a vector-point p on the spiral changes in length with respect to θ. Hence, for a fixed θ in Equation 5.18, the bigger is b the bigger is the spiral.

5.2.2 Two-Spiral Problem

The original two-spiral problem consisted of 194 input data distributed equally over two intertwined spirals [37] rotating π Radians relative to each other. Each spiral is made of 97 points and rotates 3 complete times around the origin of the $x - y$ plane. In order to equally distribute the points, each round must be equally divided into 32 divisions. Hence, the angular distance of two successive points is:

$$\delta = 2\pi/32 \tag{5.20}$$

If the starting and the end point of the spiral is known, the constants a and b in Equation 5.15 can be found. The original two-spiral problem requires that the $x - y$ coordinates of the starting (s) and the end point (t) of the outer spiral be defined as:

$$\begin{cases} s = (0.5, 0) \\ t = (6.5, 0) \end{cases}$$

Since s and t are two vector points lying on the X axis, we have:

$$\begin{cases} \theta_s = 0 \\ r_s = 0.5 \end{cases} \tag{5.21}$$

and

$$\begin{cases} \theta_t = 6\pi \\ r_t = 6.5 \end{cases} \tag{5.22}$$

By applying Equation 5.21 in 5.15, the constant a is obtained:

$$a = 0.5 \tag{5.23}$$

Similarly, by applying Equations 5.22 and 5.23 in 5.15, the constant b is obtained:

$$b = 1/\pi \tag{5.24}$$

Given that $i = 0, \ldots, 96$, the angle of a point p_i on the outer spiral, is given as:

$$\theta = i\delta \tag{5.25}$$

Similarly, for $i = 0, \ldots, 96$, the length of the vector point p_i can be obtained by applying Equations 5.23, 5.24 and 5.25 in Equation 5.15.

$$r = 0.5 + \theta/\pi \tag{5.26}$$

Finally, the x and y coordinates of p_i can be obtained, respectively, by applying Equations 5.16 and 5.17 for the calculated values of θ and r. For $i = 0, \ldots, 96$, any point q_i on the inner spiral is given as:

$$q_i = -p_i = (-x, -y)$$

Points on one spiral are assigned 0 to create the desired values, whereas for the other spiral are labelled 1.

The previous procedure were repeated to create 140 data points (70 for each spiral) of the test set by dividing each spiral's round to 23 division rather than 32. The points on the test set are those not included in the training set but lying between the original training points on the sketched spiral. Creating the test data in this regard is in line with what Chalup et al suggests: "Generalisation ability of the trained classifier is typically tested by evaluating it on a subset of all the other pixels of the image which were not included as spiral points in the training set".

5.2.3 Experimental Results and Comparison

The two-spiral problem was tested using MINES to assess the classification ability of the system. We first present one of the best results obtained by MINES on the two-spiral problem. The only change in the settings of the system was the use of Simple K-Means[6], set on 15, for the clustering part and the use of sinus and logistic functions as the activation function of hidden nodes and the final output node, respectively. Table 5.5 lists the exact settings used to for the aforementioned problem.

Population Size	No. of Generations	No. of Parents	No. of Offspring	Recombination Type	Mutation Rate	Clustering Type	No. of Clusters
32	200	2	5	2-point Crossover	1%	K-Means	15

Table 5.5: Important Settings of Two Spiral Classification Problem

[6]Clustering using X-Means results an unstable convergence on the two-spiral problem. The main reason is that the dimension of input space - i.e. the X-Y plane - is only 2. Hence, GA tends to converge very fast and immaturely when X-means clustering is applied. Simple K-Means, in contrast, slows down the convergence by applying a deliberate control on the clustering of weight vectors. This is especially beneficial when the dimension of input space is low.

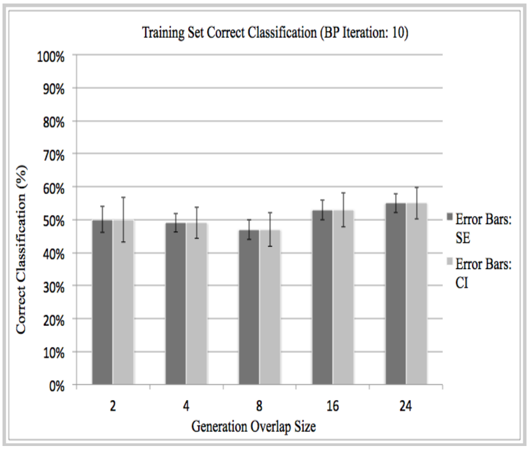


Figure 5.43: Correct classification of the two spiral problem on the training set when number of BP iteration is 10. CI represents a 90% confidence Interval bar.

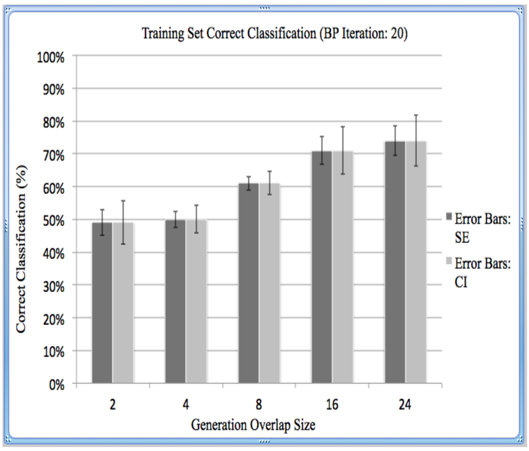


Figure 5.44: Correct classification of the two spiral problem on the training set when number of BP iteration is 20. CI represents a 90% confidence Interval bar.

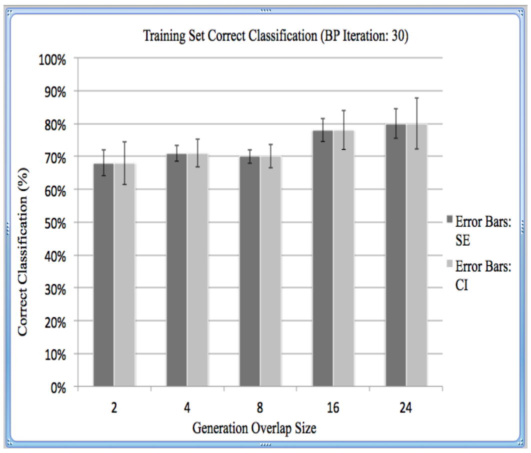


Figure 5.45: Correct classification of the two spiral problem on the training set when number of BP iteration is 30. CI represents a 90% confidence Interval bar.

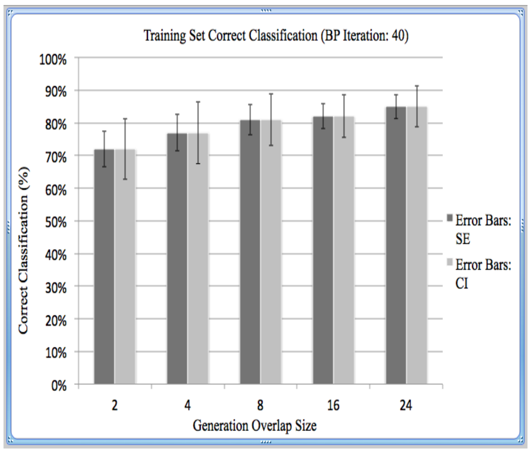


Figure 5.46: Correct classification of the two spiral problem on the training set when number of BP iteration is 40. CI represents a 90% confidence Interval bar.

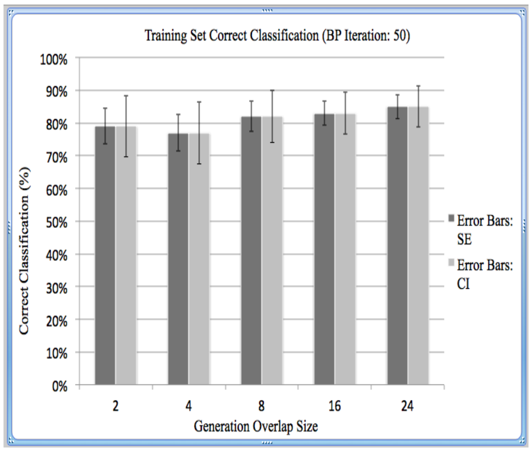


Figure 5.47: Correct classification of the two spiral problem on the training set when number of BP iteration is 50. CI represents a 90% confidence Interval bar.

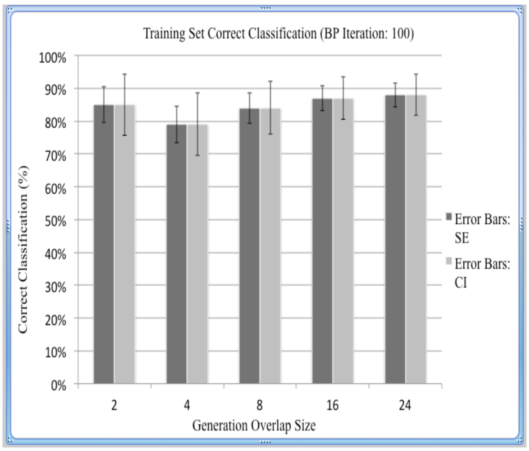


Figure 5.48: Correct classification of the two spiral problem on the training set when number of BP iteration is 100. CI represents a 90% confidence Interval bar.

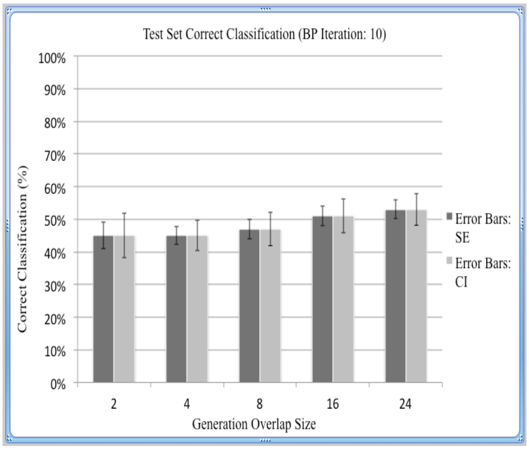


Figure 5.49: Correct classification of the two spiral problem on the test set when number of BP iteration is 10. CI represents a 90% confidence Interval bar.

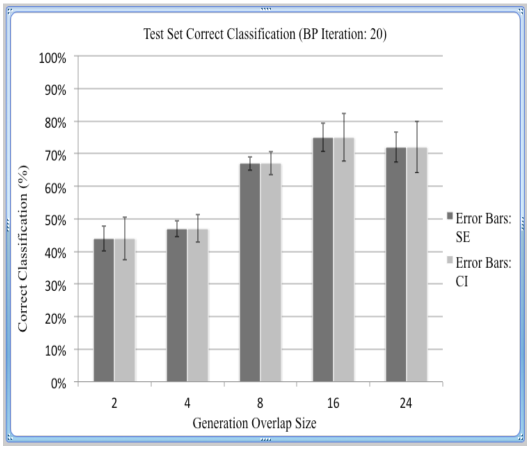


Figure 5.50: Correct classification of the two spiral problem on the test set when number of BP iteration is 20. CI represents a 90% confidence Interval bar.

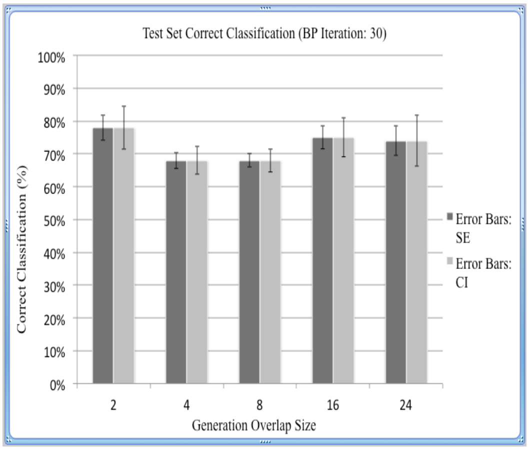


Figure 5.51: Correct classification of the two spiral problem on the test set when number of BP iteration is 30. CI represents a 90% confidence Interval bar.

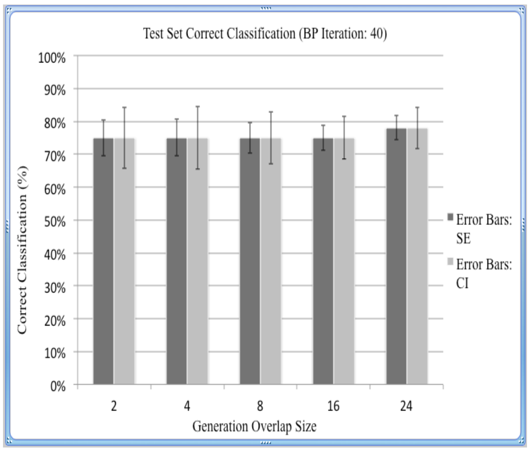


Figure 5.52: Correct classification of the two spiral problem on the test set when number of BP iteration is 40. CI represents a 90% confidence Interval bar.

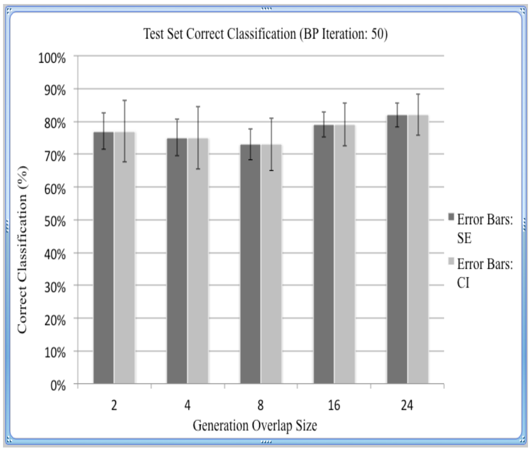


Figure 5.53: Correct classification of the two spiral problem on the test set when number of BP iteration is 50. CI represents a 90% confidence Interval bar.

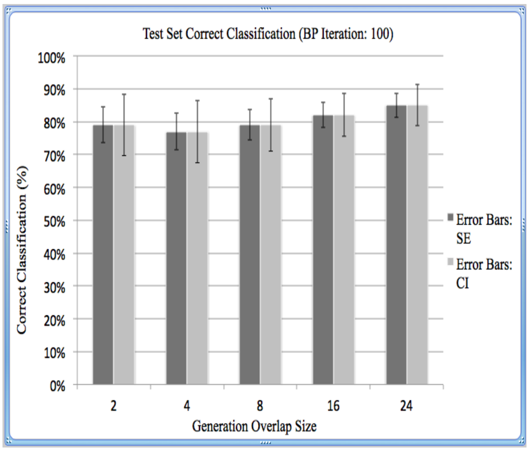


Figure 5.54: Correct classification of the two spiral problem on the test set when number of BP iteration is 100. CI represents a 90% confidence Interval bar.

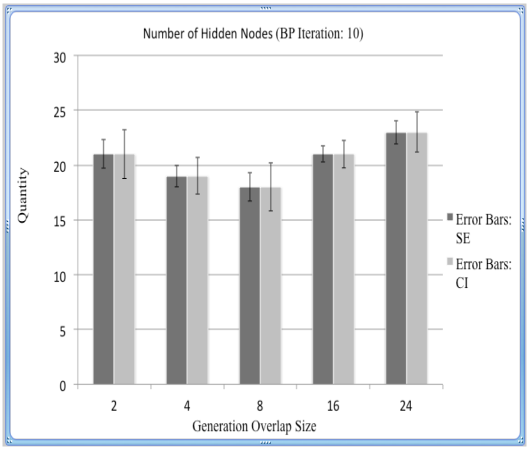


Figure 5.55: Correct classification of the two spiral problem on the test set when number of BP iteration is 10. CI represents a 90% confidence Interval bar.

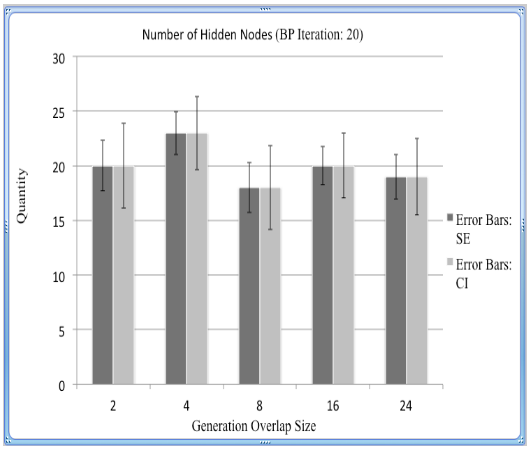


Figure 5.56: Correct classification of the two spiral problem on the test set when number of BP iteration is 20. CI represents a 90% confidence Interval bar.

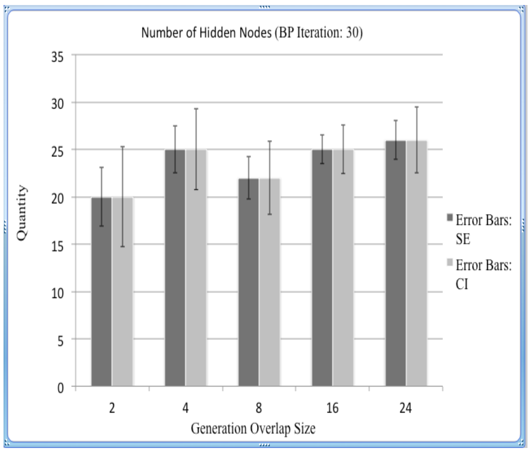


Figure 5.57: Correct classification of the two spiral problem on the test set when number of BP iteration is 30. CI represents a 90% confidence Interval bar.

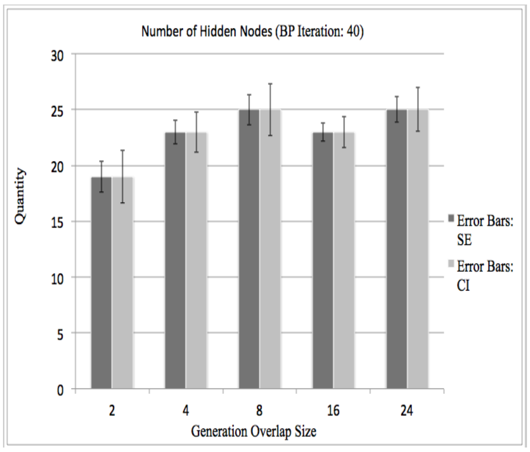


Figure 5.58: Correct classification of the two spiral problem on the test set when number of BP iteration is 40. CI represents a 90% confidence Interval bar.

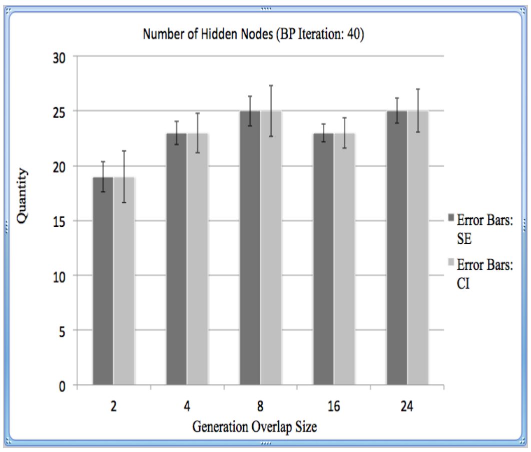


Figure 5.59: Correct classification of the two spiral problem on the test set when number of BP iteration is 50. CI represents a 90% confidence Interval bar.

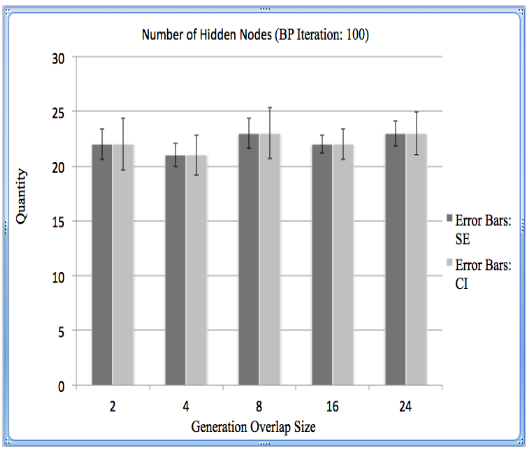


Figure 5.60: Correct classification of the two spiral problem on the test set when number of BP iteration is 100. CI represents a 90% confidence Interval bar.

The obtained results indicate that, MINES was able to successfully classify the two underlying spirals with 100% accuracy on the training set and 92% on the test set. It is observed from Figure 5.62 that the convergence of the system happened after 100 generations and on 15 hidden nodes (see Table 5.5 for the applied settings). The perfect performance of MINES on the two-spiral problem proves its ability as a robust classifier system. The result indicate that, MINES is a competent system even when the dimension of the input space is very limited - which is 2 in the two-spiral problem. This is clearly in contrast with our former experiment, the polynomial regression problem in Section 5.1, dealing with a substantially higher input dimension.

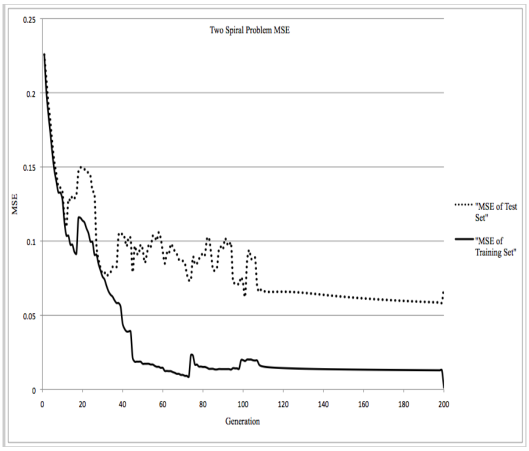


Figure 5.61: MSE on training and test sets for the two spiral problem.

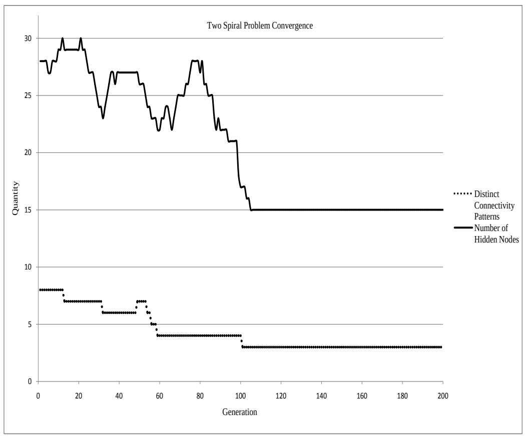


Figure 5.62: Convergence of MINES on the two-spiral problem.

To asses the effect of parameter settings on MINES performance when applied on the two-spiral problem, the number of BP iterations and the generation overlap size has been set to different values. Figures 5.43 to 5.48 show how change in the generation overlap size affects the two spiral classification performance of MINES on the training set over various BP iterations. Similarly, Figures 5.49 to 5.54 show the same set of experiments on the test set instead of the training set. It is observed from the figures that when the BP iteration is 10 not only the classification performance of the system is poor (around 40 to 50 percent) on both training and test sets, but also it is not significantly difference across various values of generation overlap size. A possible explanation is that the BP of 10 is simply not adequate to tune the synaptic weights over a certain threshold so that the effect of change in the generation overlap size can be seen. In contrary, it is observed that when the BP is 20, there is a significant difference on both training and test sets classification for generation overlap size of 8, 16 and 24. In fact, when the the generation overlap size is higher, new offspring are compared against more individuals in the old population before being replaced while forming the new population. The latter makes the system able to select better individuals during evolutionary process and result better performance. However, for higher values of BP iterations, as it is observed from Figures 5.45 to 5.48 and 5.51 to 5.54, the generation overlap size has smaller or even no significant effect. The observation may again be attributed to the effect of BP iteration on tuning the weights. In fact, due to the low number of possible connectivity patterns for the two spiral-problem, when weights associated to individuals are trained above a certain threshold the majority have a similar performance; hence, the generation overlap size cannot play a significant role.

Figures 5.55 to 5.60 show the number of incorporated hidden nodes in MINES after convergence on the two-spiral problem. As it is observed the number of hidden nodes varies from 15 to 30 with no significant difference neither for any value of generation overlap size and BP iteration.

5.2.3.1 Comparison to CCNN

As it was mentioned earlier, the two-spiral problem has historically used to assess the classification performance of CCNN or its variants. Due to the similarity of MINES and CCNN, we compare the classification results of MINES on the two-spiral problem with the original CCNN. The original classification performance of CCNN on two-spiral problem has been reported by Su and Guan (2002) [65].

Table 5.6 lists 20 random experiments applied on the two-spiral problem using the original CCNN configurations. The two-spiral data points are the same as those derived in Section 5.2.2. The activation function is the ”standard symmetric sigmoid”. The results in the table indicate that the average and the highest performance of the original CCNN on the two spiral problem is respectively 93.75% and 89.50% correct classification on the test set.

The best obtained results of MINES on the two spiral problem (see Section 5.2.3) was 100% on the training set and 92% on the test set. The average performance on the test set for 30 random experiments is about 85% (see Figure 5.54). It is in observed that the performance of MINES on the two-spiral problem is very close to those reported by Su and Guan (2002) [65] as the original performance of CCNN on the same problem.

5.3 Summary and Conclusion

In Section 5.1, it was shown that a general polynomial, having any power, can be modelled with a NN. A non-linear polynomial (with power one) was brought forward as the regression problem of MINES. The selection of the problem was justified based on numerous reasons. A great number of experiments were carried out to assess the sensitivity of the involved parameters of MINES. Based on the carried out experiments the optimum settings were identified. The experiments showed that MINES is always able to deliberately approximate the benchmark polynomial with another polynomial evolved from the convergence of the system. However, the exact optimum polynomial (with two hidden nodes) may not always be obtained. Nevertheless, the performance of one of the NNs exactly converging to the true polynomial was compared with the benchmark problem. The comparison suggested that MINES has an equivalent, or sometimes even better, performance. The big difference between MINES and the benchmark systems is that MINES does not need to have the number of hidden nodes defined in advance. Defining the number of hidden nodes in advance often requires an extra knowledge of the problem - at least in the case of polynomial discovery. MINEs would itself converge smoothly to a fixed number of hidden nodes. Finally, by performing several random experiments, it was additionally shown that MINES is robust to overfit on the underlying polynomial regression problem.

Section 5.2 started with an intuitive definition of spiral resulted from the trace of a rotating point on the plane about a fixed point, while, at the same time, getting away from it. Equation 5.12 defines the relation of the Euclidean angle and the length of a vector point on the spiral. Subsequently, Equations

Exp. No.	MSE (Test Set)	No. of Hidden Nodes	Correct Classification (%)	No. of Training Iterations	Network's Depth
1	0.65	14	88.54	2364	14
2	0.19	11	93.75	1947	11
3	0.31	14	93.23	2553	14
4	0.56	18	86.45	3088	18
5	0.24	16	90.62	2947	16
6	0.27	17	89.58	2910	17
7	0.50	15	82.81	2716	15
8	0.54	17	84.37	2987	17
9	0.36	14	88.54	2326	14
10	0.69	15	86.98	2488	15
11	0.34	16	93.75	3067	16
12	0.43	12	91.67	1774	12
13	0.59	14	87.50	2479	14
14	0.61	15	88.54	2326	15
15	0.36	18	89.58	3189	18
16	0.66	16	89.59	2882	16
17	0.25	15	91.67	2718	15
18	0.35	17	89.59	2602	17
19	0.51	13	92.70	2307	13
20	0.40	16	90.62	3110	16
High	0.69	18	93.75	3110	18
Low	0.19	11	82.81	1774	11
Mean	0.44	15	89.50	2639	15

Table 5.6: The original CCNN classification performance on the two spiral problem, reported by Su and Guan (2002) [65].

5.16 and 5.17 provides the polar coordinates of a point on the spiral in the $x - y$ plane. With these equations a spiral is possible t be sketched on the plane.

Using the derived equations, the two spiral training set consisting of 97 points for each spiral was obtained. The test set consists of 140 points lying between the training set points.

MINES demonstrated a 100% classification success on the training set accompanied by 92% correct classification of the test set. Furthermore, the system converged smoothly to 15 number of hidden nodes after about 100 generations.

The results indicate that MINES is a powerful system performing well on the two-spiral problem, regarded as one of the hardest classification tasks for FFNNs. The performance of MINES on the two-spiral problem is especially interesting due to the low dimension of the input space. The dimension of the input space in only 2 for the two-spiral problem. The convergence of GA on such a limited connectivity patterns indicates the well coordination of the incorporated systems in MINES.

Chapter 6

Application of MINES to Business Forecasting

Forecasting crude oil price has been selected as the real world business application of MINES. Daily *West Texas Intermediate* (WTI) crude oil spot prices are chosen as the input data for training and test sets. WTI price is the most famous benchmark widely used as the basis of many crude oil price formulas [78]. It is the US benchmark of the three major benchmarks (WTI, Brent blend, Dubai) which are used to price other types of crude oil based on their relative differences [20].

Prior to presenting data into the system, it is decomposed into its corresponding components based on the Empirical Mode Decomposition (EMD) method [32] (see Section 6.2). The same procedure has been used by Yu et al. [78][81][80], who decomposed the historical crude oil prices into 9 components (IMCs/IMFs) representing as the input signals to the underlying NNs. One of the studies ([78]) a single layer FFNN has then used to predict the crude oil prices, while in the others ([81][80]) each component was represented to a single FFNN forming an ensemble of network producing the final output price. In this chapter we follow the first approach ([78]) using MINES.

This chapter aims to demonstrate the following issues:

- Forecasting crude oil prices is possible by using the EMD technique.
- MINES performs well on forecasting crude oil prices, as a real world application, in conjunction with the EMD technique.
- MINES is able to automatically converge to a fixed number of hidden nodes which is capable to reasonably forecast the crude oil prices, using the EMD technique.
- MINES is able to automatically detect the most strong input signals from the rest (the input signals are the IMCs presented to the input nodes of MINES).

Although the first issue has earlier been proved using a fully connected FFNN [78], MINES demonstrates its successful performance (the second issue) on a partially connected FFNN.

It must be noted that in the aforementioned studies ([78][81][80]), it is repeatedly emphasized that the number of hidden nodes is assigned based on trial and error. MINES, in contrary, is able to automatically converge to the required number of hidden nodes with respect to the third issue.

In one of the underlying literature ([78]), a single three layer FFNN was used to reveal the interrelationships of the IMCs presented as the input signals. A manual pruning technique was then adopted to eliminate weak input signals. The “selection criterion” [78] was based on the synaptic weights of the trained FFNN. The importance of an input component i (e.g. IMC$_i$) is denoted by s_i, defined as:

$$s_i = \sum_{j=1}^{n} |w_{ij}|$$

where n is the number of hidden nodes, while w_{ij} is the synaptic weight of the input node i connected to the hidden node j. Subsequently, the final relative importance of input node i, providing the input signal IMC$_i$, is defined as:

$$\tilde{s_i} = \frac{s_i}{\sum_{i=1}^{m} s_i} \tag{6.1}$$

where $\tilde{s_i}$ is the “normalized input strength” and m is the number of input nodes, which is equal to the number of IMCs.

Regarding to the fourth issue, it will be shown that MINES automatically evolves the synaptic connections of the final FFNN in a way that important input nodes possess more connections and stronger synaptic weights in comparison to the weak ones.

6.1 Empirical Mode Decomposition

EMD subdivides a single series - such as a series of historical data - into several parallel sub-series. The sub-series can then be represented into a NN. “When the underlying problem is very difficult and complex, single series representation for NN learning may be inadequate” [78]. EMD, which was initially proposed for the study of ocean waves, decomposes a signal into some oscillatory functions, named intrinsic mode functions (IMF) - or intrinsic mode components (IMC). The original data will be then the sum of the IMFs and a residue which is either the mean trend or a constant.

The focus of EMD is initially on the local oscillations of signals between two consecutive extrema [51]. Each IMF has the same number of extrema and zero-crossings and will be symmetric with respect to the local mean. IMFs impose a general separation of the data into locally non-overlapping time scale components so that signals of the same time scale would never occur at the same locations in two different IMF components. Furthermore, IMF components, which span from shortest to longest period, are orthogonal to and independent of each other [84]. In fact, IMFs are decomposed based on a simple assumption: any data consist of different simple intrinsic modes (linear or nonlinear) of oscillations. At any given time in the original data, there might be simultaneously many different modes of oscillation interacting and superimposing on each other. Once separated, each mode reveals its own characteristic which is independent of the others [31].

Given a signal $X(t)$, the general algorithm of EMD is given as follow [51]:

1. identify all extrema of $X(t)$
2. interpolate between two successive minima (respectively maxima) an envelope $e_{\text{min}}(t)$ (respectively $e_{\text{max}}(t)$)
3. compute the mean $m(t) = \frac{e_{\text{min}}(t)+e_{\text{max}}(t)}{2}$
4. extract the detail $d(t) = x(t) - m(t)$

5. iterate on the residual $m(t)$

In contrast to the Fourier analysis, the EMD method neither assumes a time series is linear or stationary prior to analysis [32], nor "eliminates any harmonic of the fundamentals required by the non-stationary and non-linearity of the data" [33].

6.2 Crude Oil Decomposition

Daily data from January 1, 1998 till December 31, 2004 is used for training set (1751 input vectors), while daily data from January 3, 2005 to October 30, 2006 is used for testing set (459 input vectors). Data is freely accessible from the website of Department of Energy (DOE) of United States[1].

Figure 6.1 shows the historical data of the training set. It can be seen that the data is highly non-linear and noisy. The EMD reveals the general trend of the data in order of different time scales. Initially, data with the highest frequency is extracted to represent the most noisy IMC having the shortest wave-length. The remaining data is subdivided again into a second series possessing a shorter frequency, but longer wave-length with respect to the former. The procedure continues until the underlying data is left with a rather monotonous curve - with no local minimum and maximum, but only the end-points - representing the most general trend of the initial data set. In other words, the EMD procedure filters [33] the underlying data into sub series oscillating from fine to coarse. Figures 6.2 to 6.10 depict such separation of the data, starting from the first component with the highest number of oscillations reduced gradually to the last component with no frequency.

As mentioned earlier each IMC is perpendicular to the other. Furthermore, as it is observed, each component has a different oscillatory time period in comparison to the other, but an approximately constant frequency within itself. As a result, it can be assumed that every IMC, in effect, represents an independent exogenous factor affecting the original data set at different time scales. A NN would reveal the nonlinear relation of these factors interacting on each other resulting the desired data point.

[1]http://www.eia.doe.gov/

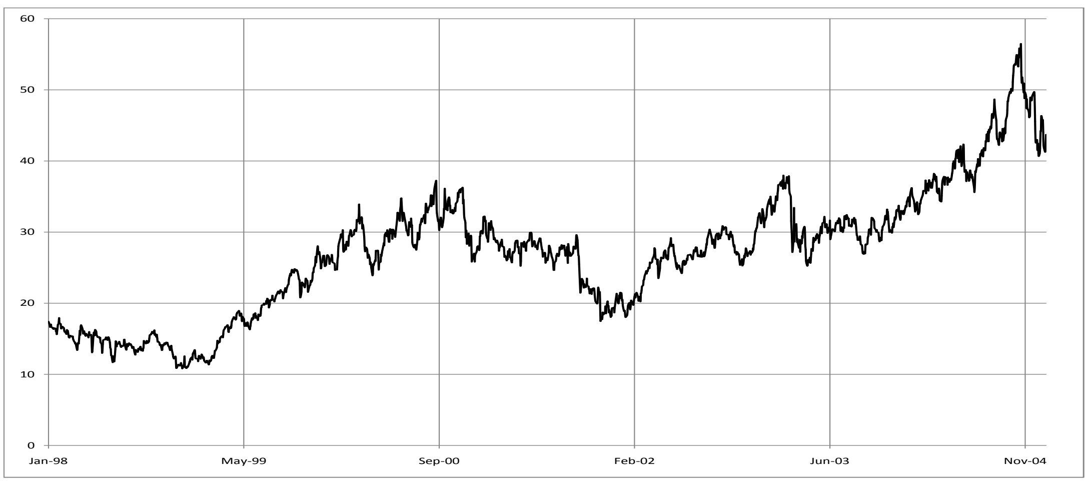


Figure 6.1: Daily WTI crude oil data, from January 1, 1998 to December 31, 2004, going under EMD decomposition to comprise the training set used for one-step-ahead forecasting.

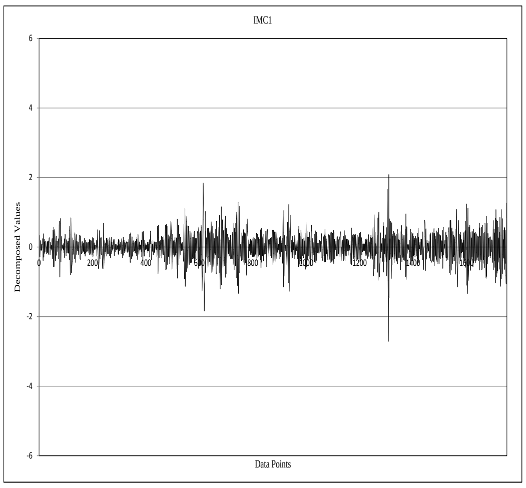


Figure 6.2: IMC1

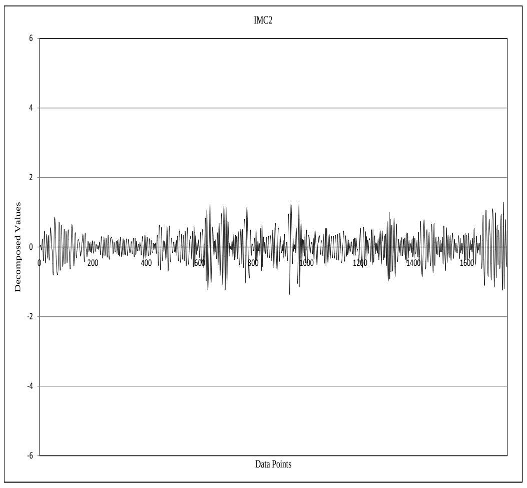


Figure 6.3: IMC2

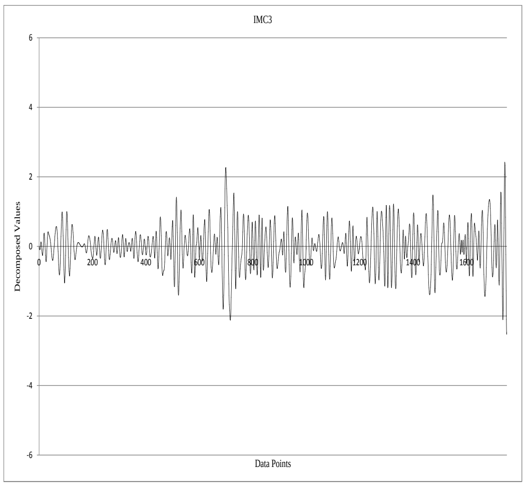


Figure 6.4: IMC3

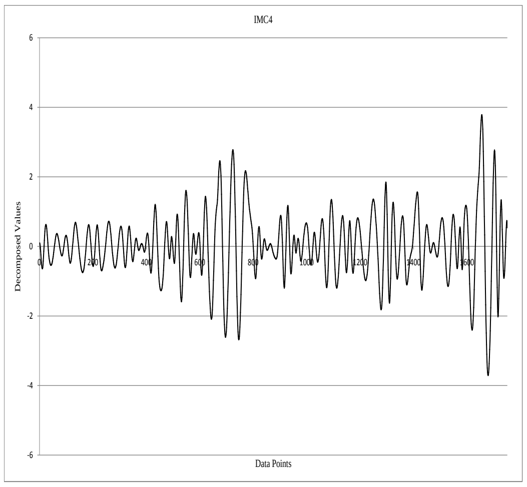


Figure 6.5: IMC4

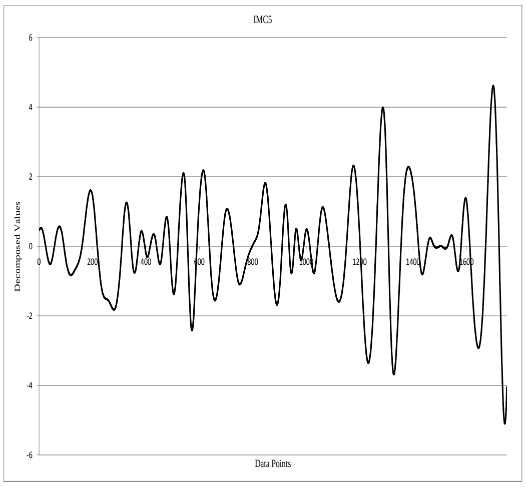


Figure 6.6: IMC5

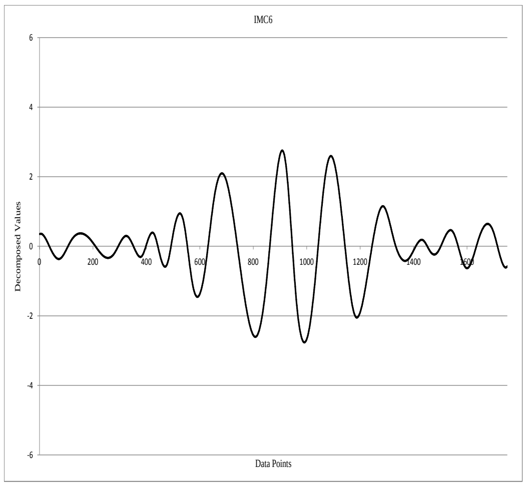


Figure 6.7: IMC6

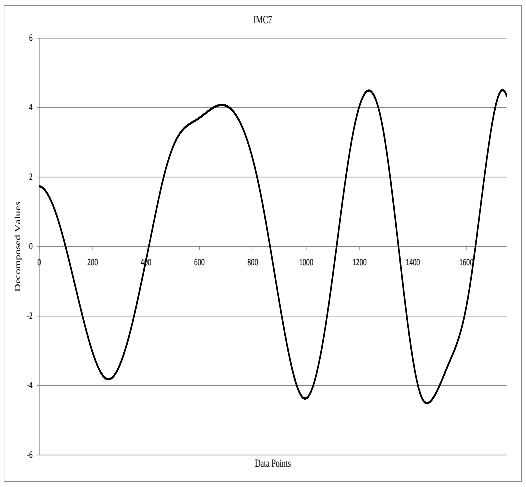


Figure 6.8: IMC7

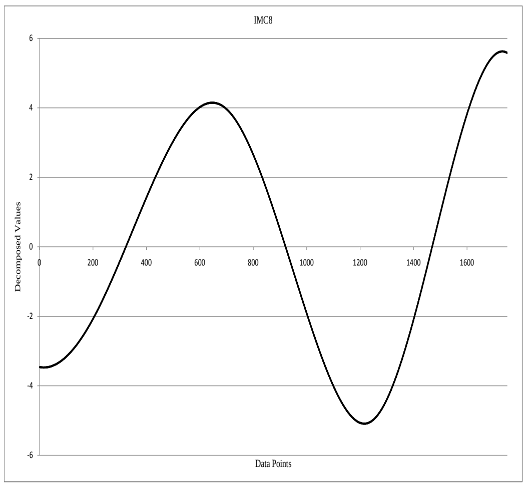


Figure 6.9: IMC8

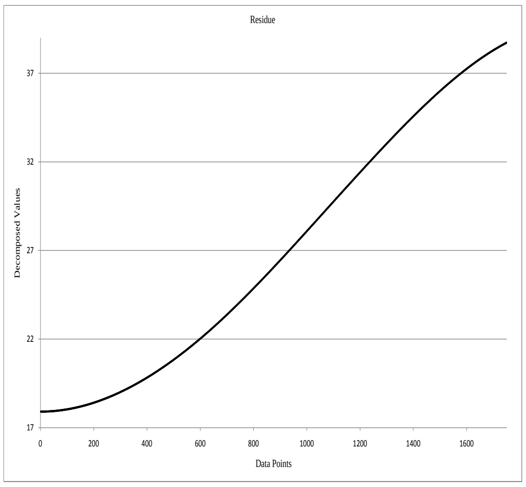


Figure 6.10: IMC9

One may be interested to know how in reality the subdivided data modes are adhered to the original data points. In other words, are the components of the EMD "physically meaningful" [33] or they are just mathematically manipulated to sum up the original data set? The answer is: the EMD method is an "intuitive and direct" derivation of the data which is "*posteriori*" based and "adaptive" [33]. The matter will be understood better when the partial sum of the IMCs are compared to the original data, as it will be discussed in the following context.

The reconstitution of the components to the original data demonstrates the "intrinsic meaning" [33] of each IMC. In Figure 6.11 the original data and the IMC-9 (the residual component of the EMD process) are shown in doted and solid lines, respectively. The residue term, representing the global trend of the data, is obtained after removing all possible oscillations. Figures 6.12 to 6.19 present the original data and the partial sum of the underlying IMCs in doted and solid line, respectively. For instance, the doted curve in Figure 6.12 is the result of the partial sum of the IMC-9 (the residue) and the IMC-8; which is in effect the "smoothest trend of the data" [33]. In each subsequent step, where the next component is added, the partial trend becomes more volatile, but resembles more to the original data. When the procedure of adding new components meets the last one (i.e. IMC-1), the partial sum coincides with the original data. In fact, ignoring the computational "round-off error", Figure 6.19 is identical to Figure 6.1.

6.3 Data Preparation and Performance Measurements

Given a time series $X = \{x_i \mid i = 1, 2, \ldots, N\}$ an l-step ahead prediction requires forecasting x_{i+l} given that x_i is presented to the system. Therefore, $l = 1$ means one single-step ahead prediction and $l = 30$ represents 30-step ahead prediction. More specifically, to make an l-step ahead prediction with a FFNN, x_{i+l} is assigned as the associated desired value of each input vector x_i for $i = 1, 2, \ldots, N - l$. More precisely the inputs and desired are:

$$\begin{cases} \text{Inputs} = \{x_1, \ldots, x_{N-l}\} \\ \text{Desireds} = \{x_l, \ldots, x_N\} \end{cases}$$

Hence, for an l-step-ahead prediction the epoch-size is equal to $N - l$; where N is the size of the given time series and l is the number of prediction steps ahead. Suppose the data point x_i is decomposed into k components and one residue using EMD:

$$x_i = \sum_{j=1}^{k} c_j(t) + r_t \tag{6.2}$$

where $c_j(t)$ is the j^{th} component series, $r(t)$ is the residue series and k is the total number of decomposed series excluding the residue. To represent the decomposed series into a FFNN and make an l-step ahead prediction, each input pattern x_i is substituted by an input vector

$$x_i = (c_1(t), \ldots, c_k(t), r(t))$$

while each desired value will be x_{i+l} as before. Since each WTI daily spot price is divided into 9 components plus one residue, the total number of input nodes of MINES is 10, excluding bias. The activation function of the hidden nodes is tangent hyperbolic whereas for the single output node it is identity. Prior to being presented into the NN, each decomposed data series is scaled in the following way:

$$\tilde{c_i} = \frac{c_i - \mu_i}{\sigma_i}$$

where $\tilde{c_i}$ is the normalised series of the i^{th} decomposed series c_i, while μ_i and σ_i are, respectively, the mean value and the standard deviation of c_i.

In order to evaluate the prediction performance, the *normalized mean squared error* (NMSE) (Weigned, 1994) has been used over the testing data X:

$$NMSE = \frac{\sum_{i=1}^{N} (x_i - \hat{x}_i)^2}{\sum_{i=1}^{N} (x_i - \bar{X})^2} = \frac{1}{\sigma_X{}^2} \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{x}_i)^2 = \frac{MSE}{\sigma_X{}^2} \tag{6.3}$$

where x_i and $\hat{x_i}$ $(i = 1, \ldots, N)$ are the i^{th} desired and predicted value, respectively; while $\bar{X}$ and $\sigma_X{}^2$ are, respectively, the mean and variance of the test set X. In Equation 6.3, we have:

$$NMSE = 1 \quad \Longleftrightarrow \quad \forall i \quad \hat{x}_i = \bar{X} \tag{6.4}$$

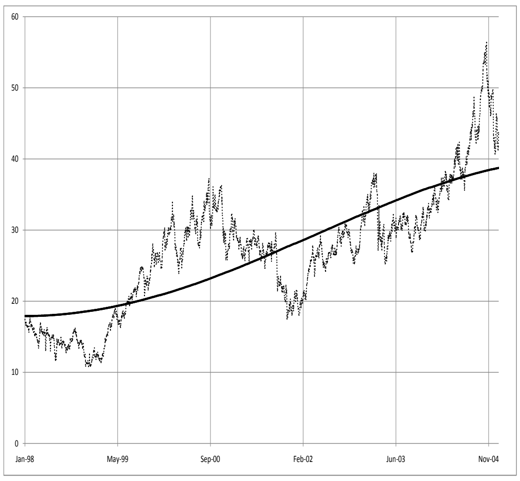


Figure 6.11: Data & IMC9

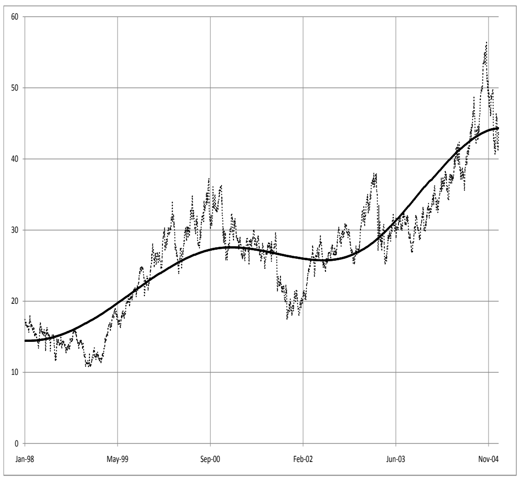


Figure 6.12: Data & Sum of IMC8 to IMC9

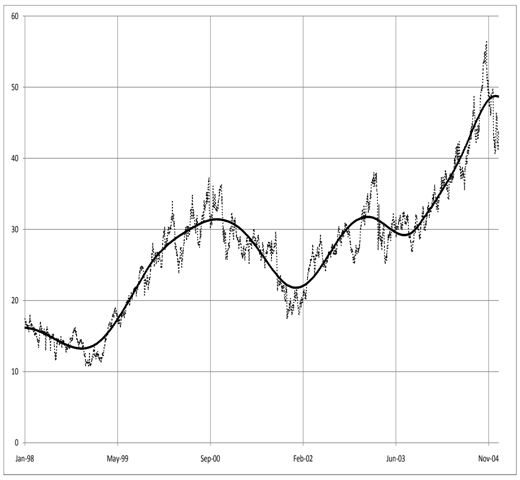


Figure 6.13: Data & Sum of IMC7 to IMC9

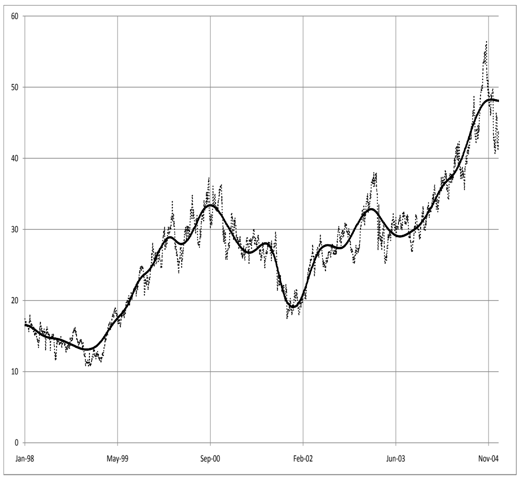


Figure 6.14: Data & Sum IMC6 to IMC9

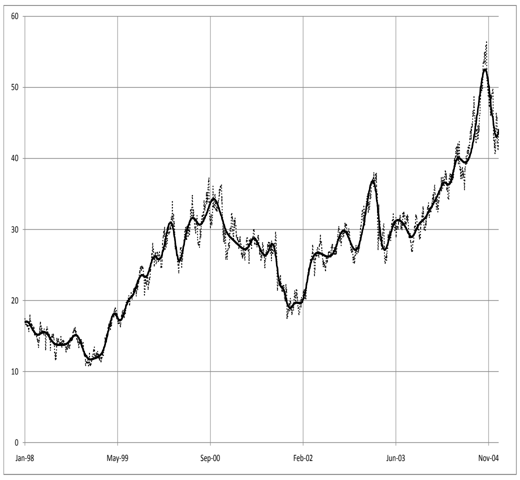


Figure 6.15: Data & Sum IMC5 to IMC9

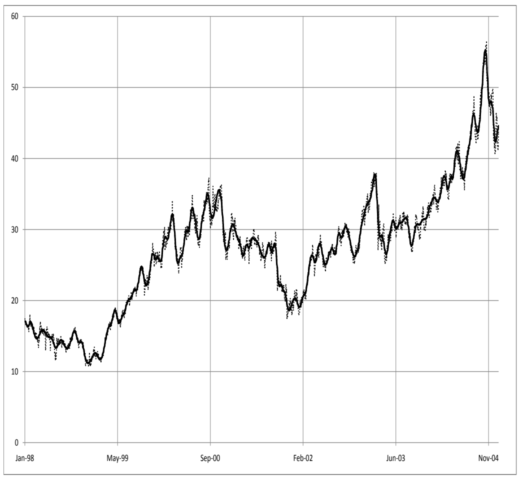


Figure 6.16: Data & Sum IMC4 to IMC9

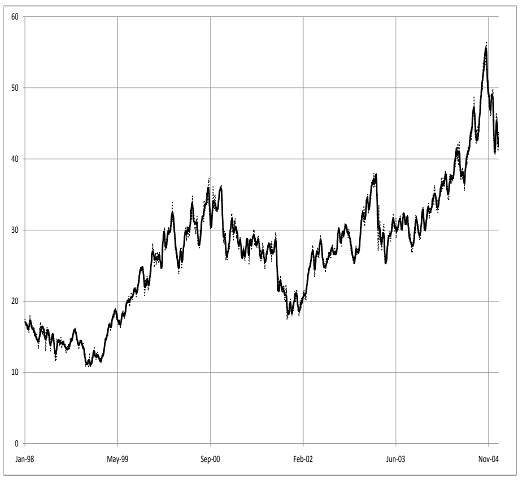


Figure 6.17: Data & Sum of IMC3 to IMC9

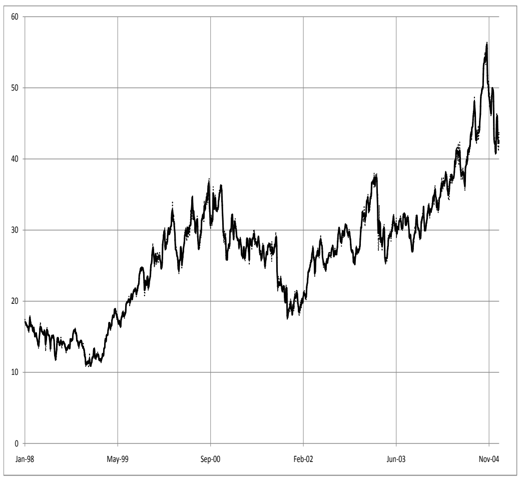


Figure 6.18: Data & Sum of IMC2 to IMC9

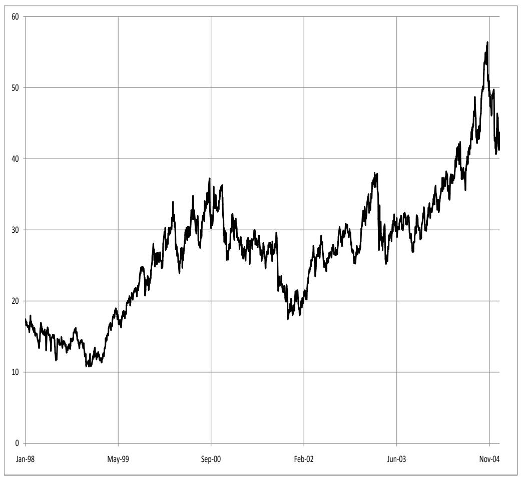


Figure 6.19: Data & Sum of IMC1 to IMC9

Thus, the closer the predicted values $\hat{x}_i$ $(i = 1, \ldots, N)$ to the mean value $\bar{X}$, the closer the NMSE to 1. The NMSE is 0 when the prediction is 100% accurate; i.e. $x_i = \hat{x}_i$ $(i = 1, \ldots, N)$. For a fixed data set X, ${\sigma_X}^2$ is constant regardless of the prediction performance. Hence, what changes the NMSE is the MSE over the test set, resulting from the output errors: $e_i = x_i - \hat{x}_i$ $(i = 1, \ldots, N)$. Therefore, when comparing a sequence of prediction performances over a fixed data set, the smaller is the MSE over the test set, the better is the performance. This is because on average the prediction errors are smaller.

NMSE is also related to R^2 which is defined as follow:

$$R^2 = \frac{\sum_{i=1}^{N} \left(\hat{x}_i - \bar{\hat{X}}\right)^2}{\sum_{i=1}^{N} \left(x_i - \bar{X}\right)^2} = \frac{\frac{\sum_{i=1}^{N} \left(\hat{x}_i - \bar{\hat{X}}\right)^2}{N}}{\frac{\sum_{i=1}^{N} \left(x_i - \bar{X}\right)^2}{N}} = \frac{\sigma_{\hat{X}}^2}{\sigma_X^2} \tag{6.5}$$

where $\sigma_{\hat{X}}^2$ and σ_X^2 are the variance of the series X and $\hat{X}$, respectively. The relation of NMSE and R^2 can be expressed as:

$$NMSE = R^2 \frac{MSE}{{\sigma_{\hat{X}}}^2} \tag{6.6}$$

Another measurement that is used to evaluate the prediction performance is the *directional statistic* (Dstat) [75] defined as:

$$Dstat = 1/N \sum_{i=1}^{N} b_i \tag{6.7}$$

where

$$b_i = \begin{cases} 1 & if(x_{i+1} - x_i)(\hat{x}_{i+1} - x_i) \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.8}$$

Dstat measures the percentage of the times that the forecasting direction was correct. For each point x_{i+1} and the associated forecasting point $\hat{x}_{i+1}$ $(i = 1, \ldots, N-1)$, the forecasting direction is solely measured based on the two possible states of being greater or smaller than the previous point x_i. Therefore, unlike NMSE, the magnitude of the error prediction is not taken into account by the Dstat; only the sign of the difference between forecasting and original points are of interest. Thus, Dstat measures the forecasting trend, which is in fact useful for real world applications such as buying or selling contracts [36]. However, the joint combination of both measurements - i.e. NMSE and Dstat - is superior to evaluate the overall performance.

6.4 Forecasting Results and Comparison

MINES was used to forecast daily crude oil prices using the settings listed on Table 6.2. The forecasting results can be seen from Table 6.1 which shows 1-step ahead forecasting for the daily decomposed WTI crude oil data. The table shows the average and the standard deviation of the number of distinct connectivity patterns and the number of hidden nodes for 10 random experiments after convergence of MINES on the training set, spanning from January 1, 1998 to December 31, 2004. The respective average and the standard deviation of the NMSEs and Dstats are also provided for the test set, covering daily data from January 3, 2005 to October 30, 2006.

As it can be observed, the average number of hidden nodes is 15, which is interestingly the same as what was claimed by Yu et al. (2007) [78] as the optimal number of hidden nodes for their experiments, using the same data set. It must be noted that, in contrast to MINES, which configures the required number of hidden nodes self sufficiently, Yu et al. (2007) [78] used a series of “trial and error” experiments to figure out the best number of hidden nodes for the employed three layer FFNN. Moreover, they adopted a fully connected FFNN which is clearly different from partially connected NNs evolving by MINES. This inconsistency explains the existing difference between the number of obtained hidden

Statistic	No. of Distinct Connections	No. of Hidden Nodes	NMSE	Dstat (%)	Run Time (min)
mean	12	15	0.0145	73	67
standard deviation	3	3	0.0015	4	11

Table 6.1: MINES experimental results for 10 random experiments, forecasting daily WTI crud oil prices. The number of distinct connections and hidden nodes are obtained at the convergence of the system. The NMSEs and Dstats are measured after the convergence of the system on the test set.

nodes by MINES over several experiments: depending on the diversification of the evolving receptive connectivity patterns the number of obtained hidden nodes would change. The standard deviation of 3 indicates that the number of hidden nodes on average varies from 12 to 18. The variation, however, does not significantly affect the performance of the system as it can be seen from the NMSEs on the test set. The average number of NMSEs on the test set is 0.0145, with the standard deviation of 0.0015. It means, on average, the NMSE varies from 0.0130 to 0.0160, indicating a fair tolerance of the goodness of fit when is considered accompanied with the Dstat varying from 69% to 77%.

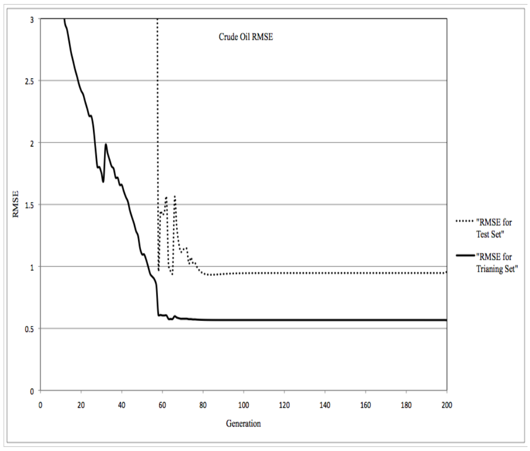


Figure 6.20: RMSE on training and test sets for the crude oil forecasting problem. For generations 1 to 60 the RMSE on the test set is very huge and hence are not shown on the graph.

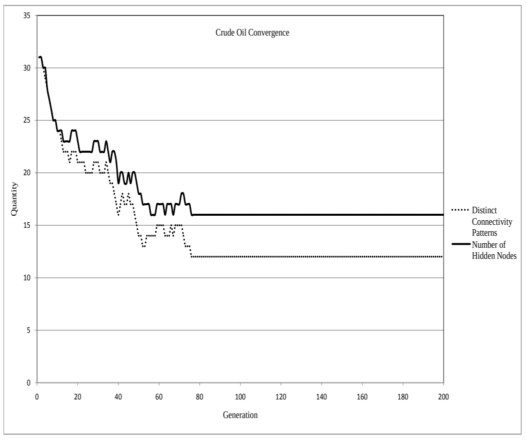


Figure 6.21: An example of MINES evolutionary convergence on steady number of hidden nodes and distinct connectivity patterns for the crude oil forecasting problem.

It is worth mentioning that, the experiments carried out by Yu et al. (2007) [78] does not provide any Dstat measurement; only NMSE has been reported for the performance evaluation. The reported NMSE is 0.0143 for a three layer FFNN with 9 input, 15 hidden and 1 output nodes; the architecture was denoted by (9:15:1). It can be observed that the reported NMSE is very close to the average values of NMSEs obtained by MINES. This is in spite of the fact that it is not obvious whether the reported value by Yu et al. (2007) is an average of several experiments or only one of the best obtained values. In a later work, however, Yu et al (2008) [81][80] report Dstat with their experiments, where the data of each IMC was fed to a single three layer FFNN before being averaged, or fed to another NN forming an ensemble, to forecast daily crude oil prices. The reported Dstat in this case is 77.45%, when the outputs of NNs are simply averaged and 86.99%, when an ensamble of NNs is used. In this sense, we may claim that the average Dstat of 73% obtained by MINES is an acceptable value, considering that only a single FFNN

Population Size	No. of Generations	No. of Parents	No. of Offspring	Recombination Type	Mutation Rate	Clustering Type	No. of Clusters
32	200	2	5	2-point Crossover	1%	X-Means	Max=3

Table 6.2: Important Settings of Daily Crude Oil Forecasting Problem

has been used and taking into account that the underlying data covers differnet time periods making them geenrally incomparable.

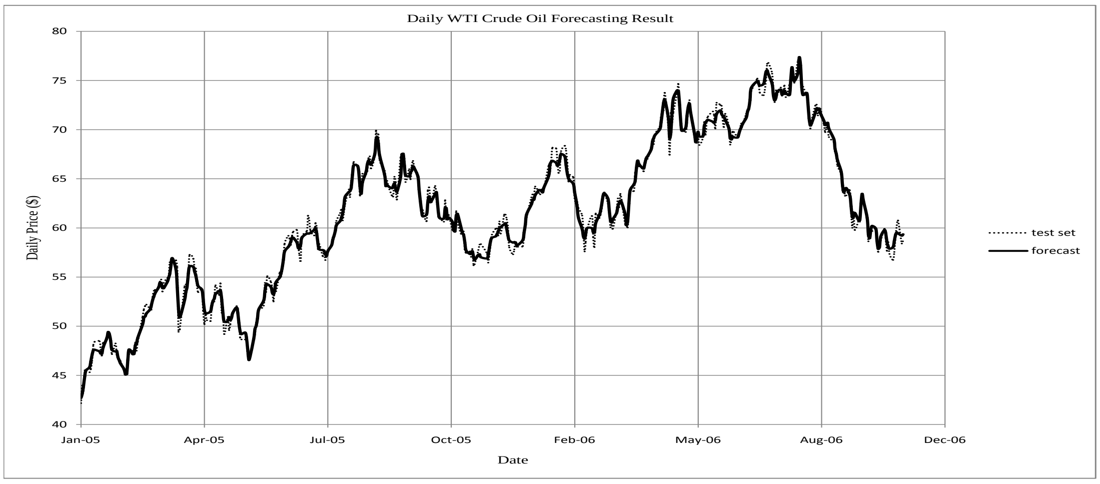


Figure 6.22: Crude oil forecasting graph

As mentioned at the beginning of Chapter 6, Yu et al. (2007) [78] used Equation 6.1 to eliminate three less important IMCs. The elimination improves the reported NMSE by reducing it from 0.0143 to 0.0084. However, there are serious criticism about Equation 6.1 as a true indicator of relative signal importance. First, the equation only considers the synaptic weights existing between input and hidden nodes; whereas the outgoing weights of the hidden nodes are also important to reveal the relative importance of the signals due to plausible negative correlation of the hidden nodes. Second, the use of the absolute values of the synaptic weights eliminates the internal correlation of the input signals and hidden nodes. Hence, we may conclude that Equation 6.1 represents a very crude estimation of the relative importance of the input signals, especially when the underlying data is highly non-linear.

Figure 6.23 indicates the relative signal strength for IMC-1 to IMC-9 reported by Yu et al. (2007) [78] using Equation 6.1. Similarly, Figure 6.24 shows the relative signal strength, using Equation 6.1, for 10 random experiments obtained by MINES; the zero column is associated to bias. It can be observed that, in both figures the IMC-9 is the strongest one whereas IMC-1 and IMC-2 are the weakest. However, for other input signals the results are mixed. One explanation for this can be attributed to the fully connected structure of the NN used by Yu et al. (2007) [78] in comparison to the partially connected NNs resulted from the convergence of MINES. Moreover, the number of hidden nodes of the NN used by Yu et al. is fixed on 15, while this is not the case for MINES as mentioned earlier. Finally, the results of Figure 6.23 is obtained from the average of 10 random experiments, whereas those reported by Yu et al. appears to be related to a selected experiment.

Due to the aforementioned problems of using Equation 6.1 for relative signal strength detection, a direct approach has been adopted in this research to measure the relative importance of the input IMCs. In this approach, each input signal is eliminated from the trained NN obtained by MINES. After the elimination of the input signal, the new NMSEs and Dstats are subsequently measured on the test set and compared to the initial NMSE and Dstat of the trained NN receiving all input signals. When an input

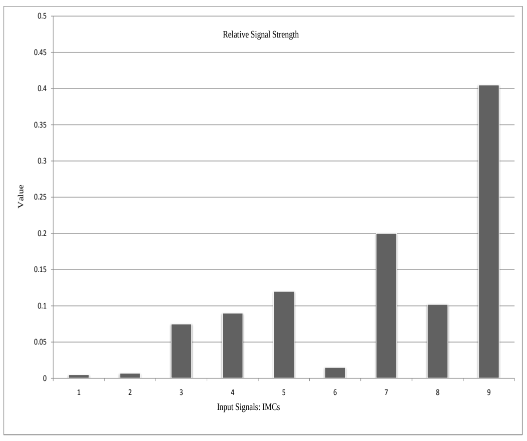


Figure 6.23: Relative signal strength for crude oil IMCs based on Equation 6.1 (Reported by Yu et al. (2007) [78]).

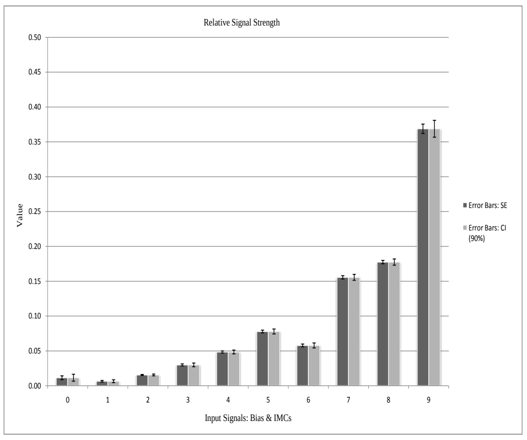


Figure 6.24: MINES Relative signal strength for crude oil IMCs based on Equation 6.1. The results are for 10 random experiments.

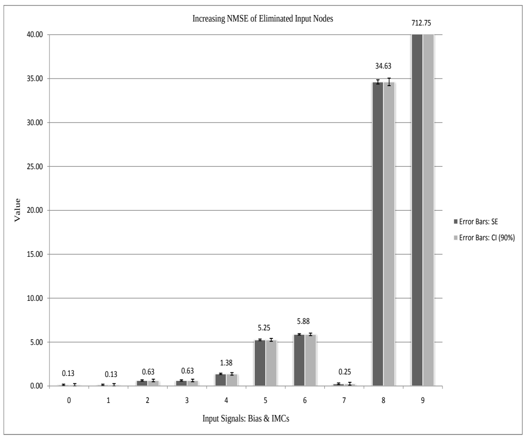


Figure 6.25: Columns shows the ratio of the NMSE increase on the test set when the respective input signal (or bias) is eliminated. The results are for 10 random experiments.

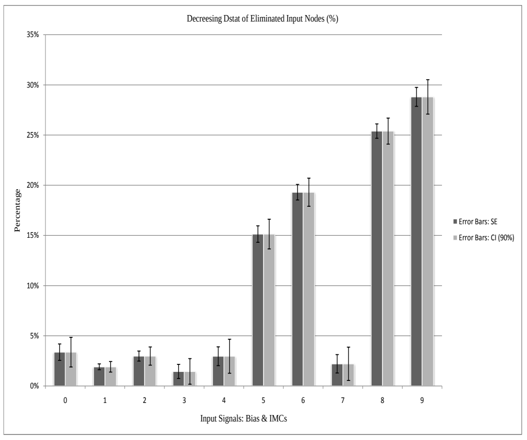


Figure 6.26: Columns shows the percentage ratio of the Dstat decrease, on the test set, after the elimination of the respective input signal (or bias). The results are for 10 random experiments.

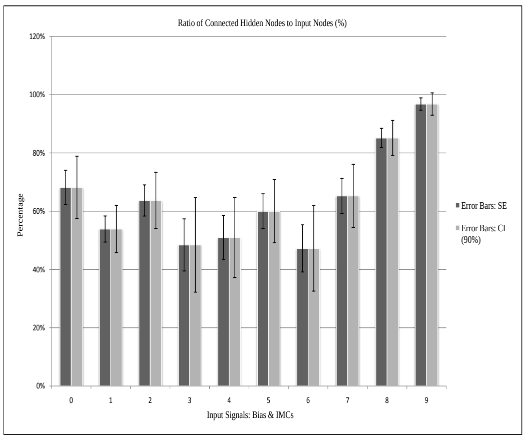


Figure 6.27: Ratio of connected hidden nodes to input nodes

signal (IMC) is eliminated, the NMSE and the Dstat increases and decreases, respectively, indicating a performance worsening.

Figure 6.25 and 6.26 show, respectively, the increase and the decrease of the NMSEs and Dstats over the elimination of the input signals over 10 random experiments after the convergence of MINES. According to Figure 6.25, the IMC-9 and IMC-8 have the most influential effect of the performance of MINES by increasing the ratio of the NMSE. For instance, the elimination of the IMC-9 results in about 700 times increase of the NMSE over the test set[2]. After the IMC-9 and IMC-8, IMC-6 and IMC-5 lie in the next stage of importance. Focussing on the Dstat, Figure 6.26 indicates that the IMC-9 and IMC-8 are also the most influential factors in the direction forecasting performance. For instance, the exclusion of IMC-9 would approximately result in a 30% decrease in the Dstat. Figure 6.26 again indicates that after the IMC-9 and IMC-8, IMC-6 and IMC-5 lie in the next stage of importance.

Following the above discussion we may expect that MINES assigns a relatively higher number of synaptic connections to the most important input signals to show a good forecasting performance. To this end, the number of synaptic connections outgoing from every input node to the all existing hidden nodes has been counted. The result has been normalized after being divided by the total number of hidden nodes. Figure 6.27 presents the ratio of the number of connected hidden nodes to the input nodes including bias (the zero column is related to the bias). It is observed that the IMC-9 and IMC-8 have the most number of connections with significant difference in comparison to the other input signals. In fact the input signal associated to the IMC-9 is nearly fully connected to all hidden nodes. In the next stage is the IMC-8 with more than 80% connection ratio. The results are fully in line with our expectation of the relative importance of the input signals: the IMC-9 and IMC-8 are the most important input signals in terms of forecasting performance according to both the goodness of fit (NMSE) and the directional trend (Dstat).

For other IMCs, Figure 6.27 shows no significant differnece in connection ratios. This seems in contrast with the relative importance of IMC-6 and IMC-5 on the overal Dstat; bringing the expectation that a significantly higher connection ratio should be observed for them in comparison to the IMC-0 to IMC-4 and IMC-7. However, it must be noted that although the IMC-6 and IMC-5 have a significant effect on the directional forecating performance, they do not considerably worsen the avergae goodness of fit (NMSE) of the predicted data. In addtion, it must be noted that the average connectivity in general is a pretty crude indicator; MINES may have found alternative ways to use these inputs that require limited connectivity. Therefore, we may conclude that the average connectivity is better to be used as an indicator of the most important input signals - like the IMC-9 and IMC-8 which have a considerable effect on both the goodness of fit and the direction of prediction.

From the aforesaid discussion, the following conclusion can be made: In the application of the daily crude oil price prediction, MINES is able to detect the most important and influential input signals (input nodes) from the rest by assigning more synaptic connections to them. The result is in line with the last objective brought forward in Section 1.3.

[2]Note that, in Figure 6.25, column 9 has a value of 712.75 which goes far beyond the rest of the columns. Hence, the vertical axis has been restricted to 40 to provide a better visualization comparability.

6.5 Comparison to CCNN

IN Section 6.4, the forecasting capability of MINES was compared to the the techniques applied by Yu et. al [78][81][80]. However, to compare the performance of MINES with another alternative system, the CCNN was used on the same data set to make one step ahead daily crude oil forecasting. The CCNN software can be found in [68], which is the world's largest provider of hosting for open source software development projects.

Figure 6.28 demonstrates the trend of MSE on the training set to forecast one step ahead daily crude oil prices. The presented figure is the result of one of the best performances when changing the involved settings parameters listed in Figure 6.30. It is observed from Figure 6.28 that after about 110 epochs the MSE starts increasing and begins to resonates. Hence, the system was stopped by the operator at generation 110 and results were extracted.

Statistic	No. of Layers	No. of Hidden Nodes	NMSE	Dstat (%)	Run Time (min)
mean	12	47	0.059	60	19
standard deviation	2	4	0.009	3	4

Table 6.3: CCNN experimental results of forecasting daily WTI crud oil prices. The NMSEs and Dstats are measured after the convergence of the system on the test set.

To achieve a better performance for the CCNN, both inputs and desired values are normalized before training. Therefore, the MSEs on the training and test sets achieved by the CCNN have been transformed in Table 6.3 to reflect the true values. Table 6.3 represents 20 random experiments that have been carried out by the CCNN when changing the influential parameters listed in Figure 6.30. To make the comparison of MINES and CCNN easier, a copy of Table 6.1 is again presented in Table 6.4. It can be seen from Table 6.3 that both the NMSE and Dstat obtained by CCNN is worse than those reported by MINES: the average NMSE on test set is 0.061 for CCNN, while the respective value for MINES is 0.0145; similarly, the Dstat for CCNN is just 60%, while the crresponding value for MINES is 73%.

Statistic	No. of Distinct Connections	No. of Hidden Nodes	NMSE	Dstat (%)	Run Time (min)
mean	12	15	0.0145	73	67
standard deviation	3	3	0.0015	4	11

Table 6.4: MINES Experimental results for 10 random experiments, forecasting daily WTI crud oil prices. The number of distinct connections and hidden nodes are obtained at the convergence of the system. The NMSEs and Dstats are measured after the convergence of the system on the test set.

Figure 6.29 shows part of the CCNN architecture when applied on the daily crude oil forecasting problem. The architecture shows the number of layers, nodes and synaptic connections. As it is seen from Table 6.3, CCNN on average converges to 13 layers and 15 nodes. It is worth noting that the first layer usually consists of several nodes connected directly to the final output node, while other layers

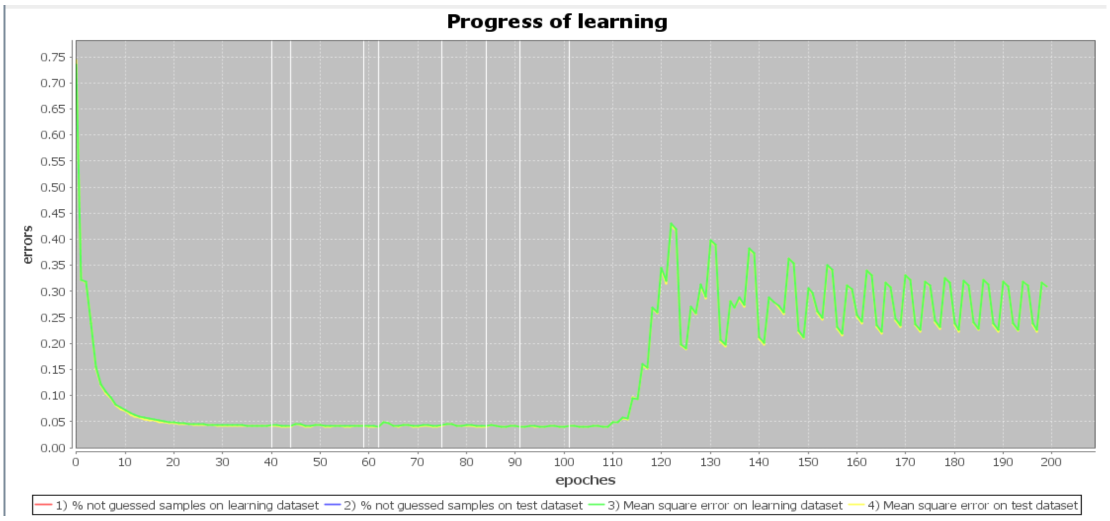


Figure 6.28: The trend of MSE of the CCNN for training and test sets when forecasting one step ahead daily crude oil prices. Note that, before training the system, both the inputs and desired values are transformed. Hence, the MSEs on the graph do not represent the true MSE before transformation. The true MSEs have been calculated sperately in Table 6.3.

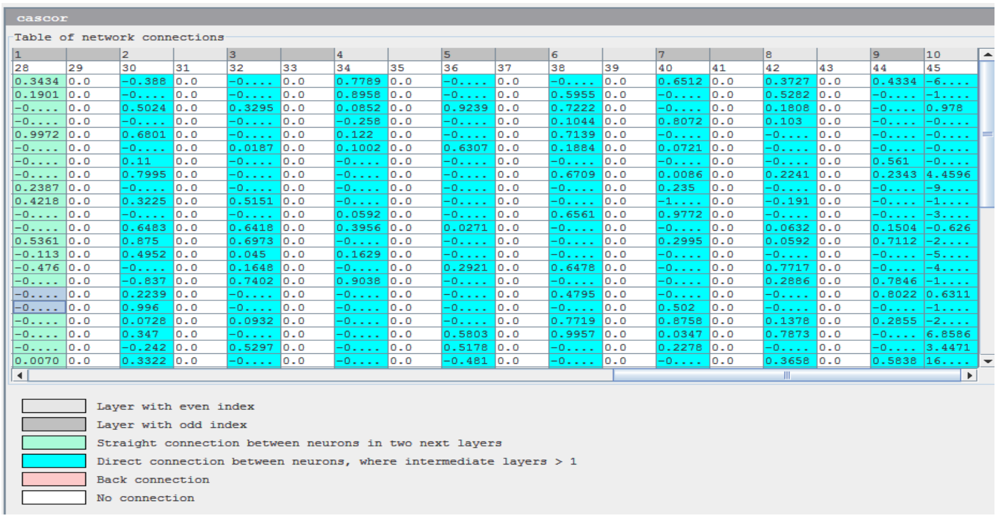
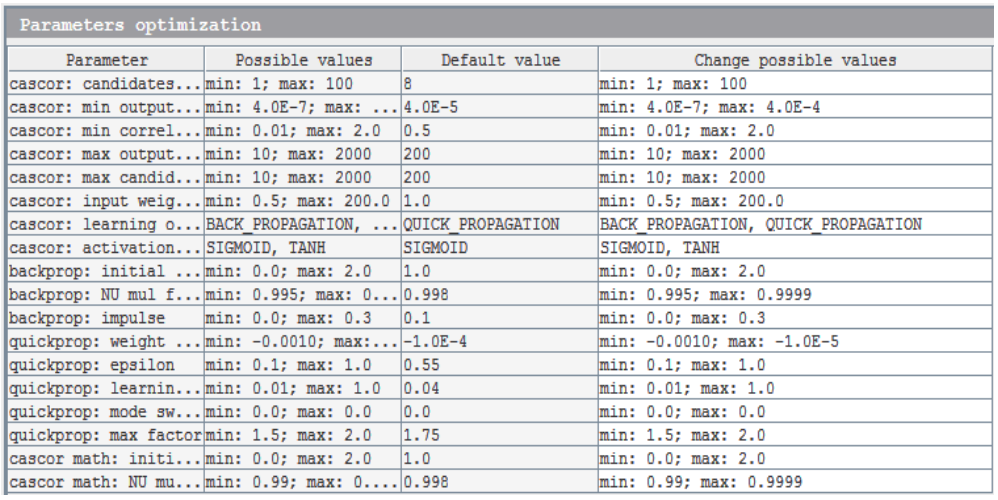

cascor

Table of network connections

1		2		3		4		5		6		7		8		9	10
28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
0.3434	0.0	-0.388	0.0	-0....	0.0	0.7789	0.0	-0....	0.0	-0....	0.0	0.6512	0.0	0.3727	0.0	0.4334	-6....
0.1901	0.0	-0....	0.0	-0....	0.0	0.8958	0.0	-0....	0.0	0.5955	0.0	-0....	0.0	0.5282	0.0	-0....	-1....
-0....	0.0	0.5024	0.0	0.3295	0.0	0.0852	0.0	0.9239	0.0	0.7222	0.0	-0....	0.0	0.1808	0.0	-0....	0.978
-0....	0.0	-0....	0.0	-0....	0.0	-0.258	0.0	-0....	0.0	0.1044	0.0	0.8072	0.0	0.103	0.0	-0....	-0....
0.9972	0.0	0.6801	0.0	-0....	0.0	0.122	0.0	-0....	0.0	0.7139	0.0	-0....	0.0	-0....	0.0	-0....	-0....
-0....	0.0	-0....	0.0	0.0187	0.0	0.1002	0.0	0.6307	0.0	0.1884	0.0	0.0721	0.0	-0....	0.0	-0....	-0....
-0....	0.0	0.11	0.0	-0....	0.0	-0....	0.0	-0....	0.0	-0....	0.0	-0....	0.0	-0....	0.0	0.561	-0....
-0....	0.0	0.7995	0.0	-0....	0.0	-0....	0.0	-0....	0.0	0.6709	0.0	0.0086	0.0	0.2241	0.0	0.2343	4.4596
0.2387	0.0	-0....	0.0	-0....	0.0	-0....	0.0	-0....	0.0	-0....	0.0	0.235	0.0	-0....	0.0	-0....	-9....
0.4218	0.0	0.3225	0.0	0.5151	0.0	-0....	0.0	-0....	0.0	-0....	0.0	-1....	0.0	-0.191	0.0	-0....	-1....
-0....	0.0	-0....	0.0	-0....	0.0	0.0592	0.0	-0....	0.0	0.6561	0.0	0.9772	0.0	-0....	0.0	-0....	-3....
-0....	0.0	0.6483	0.0	0.6418	0.0	0.3956	0.0	0.0271	0.0	-0....	0.0	-0....	0.0	0.0632	0.0	0.1504	-0.626
0.5361	0.0	0.875	0.0	0.6973	0.0	-0....	0.0	-0....	0.0	-0....	0.0	0.2995	0.0	0.0592	0.0	0.7112	-2....
-0.113	0.0	0.4952	0.0	0.045	0.0	0.1629	0.0	-0....	0.0	-0....	0.0	-0....	0.0	-0....	0.0	-0....	-5....
-0.476	0.0	-0....	0.0	0.1648	0.0	-0....	0.0	0.2921	0.0	0.6478	0.0	-0....	0.0	0.7717	0.0	-0....	-4....
-0....	0.0	-0.837	0.0	0.7402	0.0	0.9038	0.0	-0....	0.0	-0....	0.0	-0....	0.0	0.2886	0.0	0.7846	-1....
-0....	0.0	0.2239	0.0	-0....	0.0	-0....	0.0	-0....	0.0	0.4795	0.0	-0....	0.0	-0....	0.0	0.8022	0.6311
-0....	0.0	0.996	0.0	-0....	0.0	-0....	0.0	-0....	0.0	-0....	0.0	0.502	0.0	-0....	0.0	-0....	-1....
-0....	0.0	0.0728	0.0	0.0932	0.0	-0....	0.0	-0....	0.0	0.7719	0.0	0.8758	0.0	0.1378	0.0	0.2855	-2....
-0....	0.0	0.347	0.0	-0....	0.0	-0....	0.0	0.5803	0.0	0.9957	0.0	0.0347	0.0	0.7873	0.0	-0....	6.8586
-0....	0.0	-0.242	0.0	0.5297	0.0	-0....	0.0	0.5178	0.0	-0....	0.0	0.2278	0.0	-0....	0.0	-0....	3.4471
0.0070	0.0	0.3322	0.0	-0....	0.0	-0....	0.0	-0.481	0.0	-0....	0.0	-0....	0.0	0.3658	0.0	0.5838	16....

⬜	Layer with even index
▨	Layer with odd index
🟩	Straight connection between neurons in two next layers
🟦	Direct connection between neurons, where intermediate layers > 1
🟥	Back connection
⬜	No connection

Figure 6.29: The architecture of the CCNN at convergence, when applied on the crude oil problem.


Parameters optimization

Parameter	Possible values	Default value	Change possible values
cascor: candidates...	min: 1; max: 100	8	min: 1; max: 100
cascor: min output...	min: 4.0E-7; max: ...	4.0E-5	min: 4.0E-7; max: 4.0E-4
cascor: min correl...	min: 0.01; max: 2.0	0.5	min: 0.01; max: 2.0
cascor: max output...	min: 10; max: 2000	200	min: 10; max: 2000
cascor: max candid...	min: 10; max: 2000	200	min: 10; max: 2000
cascor: input weig...	min: 0.5; max: 200.0	1.0	min: 0.5; max: 200.0
cascor: learning o...	BACK_PROPAGATION, ...	QUICK_PROPAGATION	BACK_PROPAGATION, QUICK_PROPAGATION
cascor: activation...	SIGMOID, TANH	SIGMOID	SIGMOID, TANH
backprop: initial ...	min: 0.0; max: 2.0	1.0	min: 0.0; max: 2.0
backprop: NU mul f...	min: 0.995; max: 0...	0.998	min: 0.995; max: 0.9999
backprop: impulse	min: 0.0; max: 0.3	0.1	min: 0.0; max: 0.3
quickprop: weight ...	min: -0.0010; max:...	-1.0E-4	min: -0.0010; max: -1.0E-5
quickprop: epsilon	min: 0.1; max: 1.0	0.55	min: 0.1; max: 1.0
quickprop: learnin...	min: 0.01; max: 1.0	0.04	min: 0.01; max: 1.0
quickprop: mode sw...	min: 0.0; max: 0.0	0.0	min: 0.0; max: 0.0
quickprop: max factor	min: 1.5; max: 2.0	1.75	min: 1.5; max: 2.0
cascor math: initi...	min: 0.0; max: 2.0	1.0	min: 0.0; max: 2.0
cascor math: NU mu...	min: 0.99; max: 0....	0.998	min: 0.99; max: 0.9999

Figure 6.30: The parameters used to train the CCNN to forecast one step ahead daily crude oil prices.

(labeled by odd and even indices) consist of two sets of nodes (in each layer, the synaptic weights of the second sets of nodes are usually zero). It must be noted that in MINES there is only one hidden layer, whereas in CCNN there are usually several cascaded hidden layers. Table 6.4 shows that MINES on average requires 15 hidden nodes to solve the daily crude oil forecasting problem, while CCNN requires 47 nodes to converge.

6.6 Summary and Conclusion

This chapter started by introducing the EMD technique which is a novel procedure for data decomposition. After addressing the properties of the EMD, the daily WTI crude oil data, spanning from January 1, 1998 to December 31, 2004, was decomposed to 9 IMCs, using the introduced technique. While the IMC-9 represents the most general trend of the original data, partial sum of respective IMCs filters data with different levels. The more IMCs are added, the more the partial sum is fitted to the original data. The sum of all IMCs coincides exactly with the original data, ignoring the computational round-off errors.

The NMSE and Dstat were introduced as two important performance measurements for business forecasting applications. The NMSE mainly measures the goodness of fit while the Dstat gives the percentage of the times that the forecasting direction is correct. The two measurements must be used in parallel to provide a better overview of the performance assessment.

Using the decomposed IMCs as the input data, MINES showed a successful one-day ahead forecasting with an average performance of 0.0145 and 73% for NMSE and Dstat, respectively, on the test set covering January 3, 2005 to October 30, 2006. More importantly, MINES was able to converge smoothly to a fixed number of hidden nodes (15, on average) in all eperiments with a stisfactory performance on the test set (NMSE varies from 0.0130 to 0.0160, while Dstat ranging from 69% to 77%, on average). The average training time was only 67 minutes. The obtained results indicate a fair performance in comparison with experiments carried out by Yu et al. (2007) [78], reporting no Dstat, but an NMSE of 0.0143 using a three layer FFNN compersing of 9 input, 15 hidden and 1 output nodes.

Finally, to realise the relative importance of the input signals, NMSEs and Dstats were measured on the test set when each input signal was disconnected from 10 randomly trained NN evolved by MINES. The exclusion of the input signals, in general, resulted in a performance worsening, both in terms of NMSEs and Dstats. It was observed that, every input signal had a different level of contribution to the final performance of the trained NN. The IMC-9 and IMC-8 were the top most important signals amongst other, since their elimination reduced the forecasting performance considerably. By counting the number of synaptic connections between each input node and all existing hidden nodes, it was shown that MINES assigns significantly more number of connections to the aforementioned input signals. The observation shows that MINES is automatically capable of determining the most important input signals from the rest.

Chapter 7

Conclusion and Future Directions

7.1 Summary and Conclusion

The backbone of achieving a desirable NN for a particular problem at hand is to consider the structural configuration of nodes and synaptic weights in parallel. The performance of a NN, especially its generalization ability, is interdependent on the overall architecture of hidden nodes, their state of connections and the associated synaptic weights. Therefore, one may not achieve an optimal NN while performing any of the structural or parameter learning techniques in isolation. However, a concurrent training of synaptic connections and weights requires a theoretical basis that is able to connect the two practice in a consolidated way. Literature shows that one way to connect the two techniques is through employing GA optimizing the fitness function reflecting both the topology and error of the network (NEAT [63] is a classical example in this regard). Nonetheless, the GA cannot often compete with the BP algorithm with respect to the precision of weight tuning. In a different manner, the BP is connected to the GA, performing the structural learning, by introducing a cost function including the MSE of the network and another parameter indicating the network's configuration (the system proposed by Azzini and Tettamanzi [1] pursues a similar approach).

In a quite different approach, MINES exploits MI to indirectly link structural and parameter learning. MINES, in contrast to many proposed algorithms, does not define GA as the dual optimizer of both structure and synaptic weights; rather, in MINES, the GA only conducts structural learning, while weights are updated by the BP algorithm. However, the two practice are followed concurrently based on a theoretically well founded technique central to information theory. The technique defines a self-governing system determining the appropriate structure of hidden nodes while, simultaneously, training the synaptic weights. The use of BP for parameter learning enables MINES to tune synaptic weights more precisely compared to systems using GA for weight adjustment. Finally, the concurrent evolution of connectivity patterns and synaptic weights leads the system to acquire an optimal, or close to optimal, configuration.

The technique proposed by MINES is conceptually similar to some preceding systems such as MILCS and CCNN. They all concentrate on coordinating the remaining error of the system focused on a new component - i.e. hidden node in MINES and CCNN; rule in MILCS - so that it captures as much of the residual error as possible. In MINES and MILCS the components are assessed in parallel, whereas in CCNN each component is considered in isolate. The use of MI makes MINES more capable to detect the existing residual error compared to CCNN using Pearson's correlation. The reason is, MI captures all linear and non-linear interrelationships, while correlation only measures linear co-movements. The greater sensitivity of MI in disclosing the existing interdependencies makes MINES capable of configuring several hidden nodes simultaneously without needing to become cascaded like CCNN.

Since a three layer FFNN is a universal approximator, MINES concentration is mainly on the topology of the hidden layer. Having exploited the MI, a hidden node is evolved by being aligned with the respective residual error. The evolution is only imposed on the receptive connectivity patterns of hidden nodes. In fact, the receptive connectivity patterns play an important role in the flow of information from the input layer to the hidden node. Hence, MINES starts constructing the hidden layer from a pool of hidden nodes (i.e. the GA population) with various connectivity patterns. When connectivity patterns are not taken into account, the training procedure is limited to a NN with a fixed structure. In contrast, the procedure in MINES usually yields to a partially connected system, possessing extra advantageous over fully connected NNs.

MINES illuminates a novel approach, in comparison to conventional supervised learning methods in NNs, to shift the concentration from the expected payoff to the prediction of the expected payoff. As a second order statistic, MI makes BP reduce the training error by concentrating on the prediction of MSE rather than the MSE itself. MI, working based on conditional probability, acts as a local predictor of the expected payoff at any point in the weight space by measuring the interrelationship of hidden nodes with the respective residual errors. The interrelationships fundamentally determines the exclusion or inclusion of a hidden node from or into the comprising hidden layer. The network is evolved so that the incorporated hidden nodes are highly correlated with the network's error; hence, providing a system with minimal, or close to minimal, hidden layer size. The correlation is measured by MI which is more advantageous than the conventional Pearson's correlation used in other systems, such as CCNN.

MINES starts from a number of hidden nodes with various connectivity patterns evolved through shrinking - or, if necessary, expanding - the hidden layer, converging finally to a steady configuration. In this aspect, it is entirely distinguishable from almost all related techniques, starting from no hidden nodes (like CCNN [13][14]), or too many fully connected ones (like methods proposed by Cantu-Paz [5].

MINES's coding scheme of synaptic connections makes hidden nodes to be specialized in detecting the error space by which local learning is maximized. The evolution of synaptic connections also regularizes the smoothness of the error space. Since BP is a point-to-point search method, the more uniform is the error space, the faster and better it works. MINES, especially, is able to automatically distinguish the most important input signals from the less crucial ones by assigning more connections to them. Additionally, the partially connected networks created by MINES often show to be robust against overfitting on the experimented polynomial regression problem.

MINES employs a very standard GA without need of any modification for mutation and crossover. This is in contrast to some other techniques manipulating the GA to overcome some of the typical problems arising in the evolutionary process [63] (like the invention of a vertical crossover by Azzini and Tettamanzi [1]).

In MINES, the clustering of the weight-vectors makes the analysis of the training procedure more feasible. Since weight vectors are clustered based on their angles, the division of the weight space often happens with a minimum number of required vectors. Monitoring the clustered weight vectors facilitates understanding the learning behaviour of the system. Moreover, the clustering algorithm is derived from a solid mathematical foundation (see Section 4.5.1) based on the Euclidian angles of weight vectors (see Section 4.5). The technique does not suffer from innate imperfections and is expected to work reasonably well.

MINES showed successful performance on the chosen polynomial regression problem. The polynomial was non-linear and was regressed earlier using a complicated weight-sharing technique [56]. The problem was used as the primary criterion for testing the performance of MINES and setting the involved parameters. The main rationale for selecting the problem as the main system assessment benchmark was due to the feasibility of defining the optimal NN regressing the underlying polynomial. A clear understanding of the optimal NN would make the assessment of MINES possible by comparing the obtained results with the optimal NN. To this end, the number of obtained hidden nodes was one of the foremost factors indicating the closeness of the evolved NN with the optimal one, consisting of only 2 hidden nodes. The experiments suggest that, although MINES does not always find the optimal NN, it shows a good performance on both training and test sets. Even the system shows robust against overfitting on the several randomly performed experiments. The performance of one of the best obtained results was compared with the selected basis of comparison [56]. The result was as good as or even better than the benchmark.

On a quite different problem, MINES was tested on the two-spiral classification task. The problem is one of the hardest classification tasks which can be performed by a FFNN. It was initially introduced as the classic benchmark to test the performance of CCNN. In fact, the CCNN was able to nicely classify the two intertwined spiral points due to its highly nonlinear structure arising from its cascaded architecture. Due to the analogy between CCNN and MINES, the same problem was chosen to test the capability of the system on a tough classification subject. MINES was able to correctly classify the entire set of training points while smoothly converging to a fixed number of hidden nodes. The performance on the test set was 92% correct classification.

As a real world application, MINES was used to forecast the daily WTI crude oil prices. The underlying time series was decomposed into 9 IMCs while applying the EMD technique. The EMD technique was earlier shown to be effective on crude oil prediction using a fully connected FFNN [78], or an ensemble of fully connected FFNNs [81][80]. EMD proposes a fundamentally different approach to other similar techniques; notably, it is posteriori based derived from the nature of data. In effect, it filters the data according to the different existing oscillatory modes. As a result, each IMC usually has a different frequency when compared to the other; hence, may be regarded as the reflection of the variation of an independent exogenous factor affecting the underlying data trend. Supplying the decomposed IMCs as the input data to MINES, one step ahead prediction was achieved with an average of 73% success in forecasting correct directions, when converged smoothly to an average of 15 hidden nodes.

7.2 Future Directions

Although it was shown that MINES is a very capable system utilizing only one hidden layer, some applications may require additional hidden layers to meet their objectives. For this reason, it would be of interest to extend the proposed system to multi-layer FFNNs. Conceptually this regard would only require additional GA individuals to control the incoming synaptic connections of the hidden nodes in the extra layers. The optimization objective can still be the MI of each hidden node to the residual error. The residual error would naturally be the networks remaining error when the outgoing synapse of the underlying hidden node is disconnected from the immediate next hidden layer.

In the current version of MINES the incoming and outgoing synaptic weights of newly added hidden nodes are just the average of the respective synaptic weights of the offspring's parents. As a new line of research, a separate GA can come into operation acting solely on the synaptic weights of new offspring. This may enhance the capability of system to introduce new hidden nodes possessing both good connectivity patterns and synaptic weights at a same time. The fitness of GA individuals could be a compromised value of the MI between the newly added hidden node and the residual error and the total MSE of the network.

The evolutionary search on the hidden nodes connectivity patterns reduces the free parameter in MINES and eventually leads to a lighter NN. For synaptic weights, however, there is no restriction. Synaptic weights can continuously vary to any number that the BP algorithm implies. The only restriction comes from the activation functions; for instance, the tangent hyperbolic activation function makes values outside -1 and $+1$ ineffective. In a different practice, each hidden node could be restricted to adopt weights only in a specific interval. A GA may be employed for this purpose in a way that binary codes determine the interval which the associated hidden node is able to select weights from. For instance, the interval $(-1,+1)$ can be divided by 4 and the synaptic weight can only be selected from the interval $(-1, -1/2)$.

Currently, in MINES, individuals are initially grouped based on their connectivity patterns and then clustered based on the weight vectors. A drawback of the employed methodology is that individuals with different connectivity patterns producing similar outputs would be clustered differently. In fact, different connectivity patterns does not necessarily guarantee dissimilar functionality. Hence, in a different approach, individuals can only be clustered based on the angles and the size of weight vectors. The new technique would cause individuals with different connectivity patterns to be grouped in a single cluster. However then a problem would arise concerning the connectivity pattern of the representing hidden node.[1]. An immediate answer to the problem would suggest randomly using one of the connectivity patterns or the one producing higher MI and lower network's MSE.

In MINES clustering is applied according to the Euclidian angles of the weight vectors (see Section

[1]Suppose, two hidden nodes are clustered together so that one has a connectivity pattern of $(1, 1, 1)$ and the other $(1, 0, 1)$; what will be the connectivity pattern of the representing hidden node?

4.5). Nevertheless, other techniques could also be used as the basis of clustering. For instance, the MI between pairs of hidden nodes (the pair-wise MI) could possibly be used as an alternative. In this approach, hidden nodes possessing high pair-wise MI are clustered together. It must be noted that, in current version of MINES, the geometric approach was advantageous to monopolize the application of MI on error recognition[2].

In the present system of MINES, the calculation of MI is conducted via the histogram technique (see Section 2.4.2). However, for continuous random variables, the accuracy of the histogram technique is highly disputable; it was even called the "naive algorithm" [64]. Although Equation 2.57 indicates that over repetitive estimations the measured MIs would be more likely to be around the true value, a better technique to calculate the MI could further boost the performance of MINES. More exact calculation of MIs is expected to speed up the convergence of the system and result in more optimal NNs. For instance, the calculation of MI using "kernel density" estimators show a superior performance to the histogram method [64], which accordingly can be adopted in MINES.

As mentioned in Section 4.7, in MINES, the former incorporated hidden nodes often benefit from a host's advantage of detecting most of the error of the network when the MSE of the system is considerably low. The phenomenon makes newly created hidden nodes to have little chance to take in the network's architecture, which is when the system is converged. However, a lot of times the convergence of the system may not lead to the exact optimal structure (in the case of existence). One way to enhance the chance of obtaining the optimal NN, is to use conditional MI [7] when new offspring are created. For instance, suppose a new hidden node, denoted by h_i, is associated to a new offspring. In the new system, h_i will be added to the hidden layer, if its incorporation brings a higher MI in comparison to other currently incorporated hidden nodes. Otherwise, its MI will be measured with the residual error when an arbitrary hidden node, denoted by h_j, is deleted from the network. In effect, the following quantity is of interest:

$$\mathrm{MI}_{i|j} = \mathrm{MI}(h_i, e_{ij} \mid h_j)$$

where $\mathrm{MI}_{i|j}$ is the MI of h_i with the residual error subject to the exclusion of h_i and e_{ij} is the residual error of the network when h_i and h_j are excluded. A conceptual approach would conclude that the new hidden node h_i must be substituted instead of the former hidden node h_j if $\mathrm{MI}_{i|j}$ is bigger than MI_j; where MI_j is the usual MI of h_j with the residual error.

In Equation 3.15, representing the polynomial regressed by MINES, the power of all independent variables (i.e. x_i: $i = 1, \ldots, 5$) are one. However, to further examine its regression capability, MINES should be tested on a polynomial with arbitrary power. In this respect, the following polynomial:

$$y = 3 + 2x_1^{2/3}x_2x_3^{2/3}x_4^{1/2} + x_3^{1/2}x_4^{2/3}x_5$$

introduced by Tanahashi et al. (2005) [67] to test the newer version of BCW, can similarly be adopted in

[2]For instance, using pair-wise MIs would complicate the judgement about the performance of MI in error detection: a success in the system couild be interpreted due to a possible good collaboration between pair-wise MIs and those between HNs and the residual errors; while a failure in the system could be attributed to inaccurate estimations of the pair-wise MIs measuring the existing similarities between hidden nodes.

future application of MINES.

In Chapter 6, daily crude oil data was used to perform one-step ahead prediction. However, the long term predictability of MINES could particularly be of interest. In this concern, monthly crude oil prices can be used to assess the long term forecasting performance of MINES in future applications.

Bibliography

[1] Antonia Azzini and Andrea Tettamanzi. A neural evolutionary approach to financial modeling. In *GECCO: Genetic and Evolutionary Computation Conference*, pages 1605–1612, 2006.

[2] Eric B. Baum and Kevin J. Lang. Constructing hidden units using examples and queries. In *NIPS-3: Proceedings of the 1990 Conference on Advances in Neural Information Processing systems 3*, pages 904–910, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[3] Richard K. Belew, John Mcinerney, and Nicol N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. In Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II*, pages 511–547, Redwood City, CA, 1990. Addison-Wesley.

[4] E. J. W. Boers and H. Kuiper. Biological metaphors and the design of artificial neural networks. Master's thesis, Leiden University, Niels Bohrweg 1, 2333 CA, Leiden, The Netherlands, 1992.

[5] Erick Cantú-Paz. Pruning neural networks with distribution estimation algorithms. In *GECCO: Genetic and Evolutionary Computation Conference*, pages 790–800, 2003.

[6] Stephan K. Chalup, Craig L. Murch, and Michael J. Quinlan. Machine learning with aibo robots in the four-legged league of robocup. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 37(3):297–310, 2007.

[7] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley-Interscience, New York, NY, USA, 1991.

[8] R. S. Crowder III. Predicting the mackey-glass timeseries with cascade-correlation learning. *Proceedings of the 1990 Connectionist Models Summer School*, pages 117–123, 1990.

[9] Carsten O. Daub, Ralf Steuer, Joachim Selbig, and Sebastian Kloska. Estimating mutual information using B-spline functions - an improved similarity measure for analysing gene expression data. *BMC Bioinformatics*, 5(1):118, 2004.

[10] K. Deb. A population-based algorithm-generator for real-parameter optimization. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 9(4):236S–253, 2005.

[11] Andreia Dionisio, Diana A. Mendes, and Rui Menezes. Mutual information: a dependence measure for nonlinear time series. *Physica A: Statistical Mechanics and its Applications*, 344(1-2):326–329, 2003.

[12] C.P. Dolan and M.G. Dyer. Toward the evolution of symbols. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 123–131. L. Erlbaum Associates Inc., 1987.

[13] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems 2*, pages 524–532, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[14] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, August 1991.

[15] J. Michael Fitzpatrick and John J. Grefenstette. Genetic algorithms in noisy environments. *Machine Learning*, 3:101–120, 1988.

[16] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[17] David B. Fogel. An information criterion for optimal neural network selection. *Neural Networks, IEEE Transactions on*, 2(5):490–497, 1991.

[18] CERN-European Organization for Nuclear Research, 1999. Copyright (c).

[19] John E. Freund and Ronald E. Walpole. *Mathematical Statistics*. Longman Higher Education, fourth edition, 1986.

[20] Helyette Geman. *Commodities and Commodity Derivatives: Modelling and Pricing for Agriculturals, Metals, and Energy*. John Wiley & Sons Ltd, 2005.

[21] A. N. Gorban and Donald C. Wunsch II. The general approximation theorem. In *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, volume 2, pages 1271–1274, Anchorage, AK , USA, 1998.

[22] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *Special Interest Group on Knowledge Discovery and Data Mining Explorations*, 11(1), 2009.

[23] Seung-Soo Han and Gary S. May. Optimization of neural network structure and learning parameters using genetic algorithms. In *ICTAI '96: Proceedings of the 8th International Conference on Tools with Artificial Intelligence*, page 200. IEEE Computer Society, 1996.

[24] V. L. Hansen. *Geometry in Nature*. Wellesley, 1993.

[25] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice-Hal, second edition, 1998.

[26] Simon Haykin. *Neural Networks and Learning Machines*. Prentice-Hal, third edition, 2009.

[27] Hanspeter Herzel and Ivo Große. Measuring correlations in symbol sequences. *Physica*, A(216):518–542, 1995.

[28] Hanspeter Herzel, A. O. Schmitt, and W. Ebeling. Finite sample effects in sequence analysis. *Chaos, Solitons & Fractals*, 4(1):97–113, 1994.

[29] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975.

[30] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[31] Norden E. Huang, Zheng Shen, and Steven R. Long. A new view of nonlinear water waves: The hilbert spectrum. *Annual Review of Fluid Mechanics*, 31:417–457, 1999.

[32] Norden E. Huang, Zheng Shen, Steven R. Long, Manli C. Wu, Hsing H. Shih, Quanan Zheng, Nai-Chyuan Yen, Chi Chao Tung, and Henry H. Liu. The empirical mode decomposition and the hilbert spectrum for nonlinear and non-stationary time series analysis. *Proceedings: Mathematical, Physical and Engineering Sciences*, 454(1971):903–995, 1998.

[33] Norden E. Huang, Man-Li Wu, Wendong Qu, Steven R. Long, and Samuel S. P. Shen. Application of hilbert-huang transform to non-stationary financial time series analysis. *Applied Stochastic Model in Business and Industry*, 19:245–268, 2003.

[34] Jiancheng Jia and Hock-Chuan Chua. Solving two-spiral problem through input data representation. In *Proceedings of IEEE International Conference on Neural Networks*, volume 1, pages 132–135, 1995.

[35] Ivan Kojadinovic. On the use of mutual information in data analysis: an overview. In *Proceedings of 11th International Symposium on Applied Stochastic Models and Data Analysis (ASMDA05)*, pages 738–747. Citeseer, 2005.

[36] Richard Lackes, Chris Börgermann, and Matthias Dirkmorfeld. Forecasting the price development of crude oil with artificial neural networks. In *IWANN '09: Proceedings of the 10th International Work-Conference on Artificial Neural Networks*, pages 248–255, Berlin, Heidelberg, 2009. Springer-Verlag.

[37] Kevin J. Lang and Michael J. Witbrock. Learning to tell two spirals apart. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 52–59. Morgan Kaufmann Publishers Inc., 1988.

[38] Wentian Li. Mutual information functions versus correlation functions. *Journal of Statistical Physics*, 60(5-6):823–837, 1990.

[39] P. Merz and B. Freisleben. A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE, 2002.

[40] Olivier Michel and Philippe Collard. Artificial neurogenesis: An application to autonomous robotics. In *ICTAI '96: Proceedings of the 8th International Conference on Tools with Artificial Intelligence*, pages 207–214, 1996.

[41] Young I1 Moon, Balaji Rajagopalan, and Upmanu Lall. Estimation of mutual information using kernel density estimators. *Physical Review E*, 52(3):2318–2321, 1995.

[42] David E. Moriarty and Risto Mikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22(1-3):11–32, 1996.

[43] David E. Moriarty and Risto Mikkulainen. Hierarchical evolution of neural networks. In *Evolutionary Computation Proceedings, IEEE World Congress on Computational Intelligence, The 1998 IEEE International Conference on*, pages 428–433. IEEE Computer Society, 1998.

[44] Ryohei Nakano and Kazumi Saito. Discovering polynomials to fit multivariate data having numeric and nominal variables. In *Progress in Discovery Science, Final Report of the Japanese Discovery Science Project*, pages 482–493, London, UK, 2002. Springer-Verlag.

[45] M. Paliwal and U. A. Kumar. Neural networks and statistical techniques: A review of applications. *Expert Systems with Applications*, 36(1):2–17, 2009.

[46] Dan Pelleg and Andrew W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 727–734, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[47] Parag C. Pendharkar and James A. Rodger. Technical efficiency-based selection of learning cases to improve forecasting accuracy of neural networks under monotonicity assumption. *Decision Support Systems*, 36(1):117–136, 2003.

[48] D. S. Phatak and I. Koren. Connectivity and performance tradeoffs in the cascade correlation learning architecture. *Neural Networks, IEEE Transactions on*, 5(6):930–935, 1994.

[49] Mitchell A. Potter. A genetic cascade-correlation learning algorithm. In *COGANN-92 International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 123–133. IEEE Computer Society Press, 1992.

[50] M. A. Quaddus and M. S. Khan. Evolution of artificial neural networks in business applications: an empirical investigation using a growth model. *International Journal of Management and Decision Making*, 3(1):19–34, 2002.

[51] Gabriel Rilling, Patrick Flandrin, and Paulo Goncalves. On empirical mode decomposition and its algorithms. In *Proceedings of the IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing (NSIP-03)*, volume 2, Grado, Italy, 2003.

[52] Raúl Rojas. *Neural Networks - A Systematic Introduction*. Springer, 1996.

[53] Mark S. Roulston. Estimating the errors on measured entropy and mutual information. *Phys. D*, 125(3-4):285–294, 1999.

[54] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*. MIT Press, Cambridge, MA, USA, 1988.

[55] Kazumi Saito and Ryohei Nakano. Law discovery using neural networks. In *IJCAI'97: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1078–1083, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[56] Kazumi Saito and Ryohei Nakano. Structuring neural networks through bidirectional clustering of weights. In *Discovery Science*, pages 206–219, 2002.

[57] Jorge M. Santos, Joaquim P. Marques De Sá, and Luiís A. Alexandre. Neural networks trained with the eem algorithm: Tuning the smoothing parameter. In *Proceedings: Sixth World Scientific and Engineering Academy and Society, International Conference on Neural Networks*, volume 4 of *4*, pages 295–300, 2005.

[58] Sameer Singh. 2d spiral pattern recognition with possibilistic measures. *Pattern Recognition Letters*, 19(2):141–147, 1998.

[59] Rober Elliot Smith, Max Kun Jiang, Jaume Bacardit, Michael Stout, Natalio Krasnogor, and Jonathan D. Hirst. A learning classifier system with mutual-information-based fitness. *Evolutionary Intelligence*, 3(1):31–50, 2010.

[60] Robert E. Smith and Behzad Behzadan. Mutual information neuro-evolutionary system (MINES). In *CEC'09: Proceedings of the Eleventh Conference on Congress on Evolutionary Computation*, pages 1523–1529, Piscataway, NJ, USA, 2009. IEEE Press.

[61] Robert E. Smith and H. Brown Cribbs. Is a learning classifier system a type of neural network? *Evolutionary Computation*, 2(1):19–36, 1994.

[62] Robert E. Smith and Max Kun Jiang. MILCS: a mutual information learning classifier system. In *GECCO: Genetic and Evolutionary Computation Conference*, page 1877, 2007.

[63] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural network through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[64] Ralph E. Steuer, Jürgen Kurths, Carsten O. Daub, Janko Weise, and Joachim Selbig. The mutual information: Detecting and evaluating dependencies between variables. *Bioinformatics*, 18(90002):231–240, 2002.

[65] L. Su and S.U. Guan. Two-dimensional extensions of cascade correlation networks. In *High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition on*, volume 1, pages 138–141. IEEE, 2002.

[66] R. S. Sutton and C. J. Matheus. Learning polynomial functions by feature construction. In *Proceedings of the Eighth International Machine Learning Workshop*, pages 208–212. Citeseer, 1991.

[67] Yusuke Tanahashi, Kazumi Saito, and Ryohei Nakano. Model selection and weight sharing of multi-layer perceptrons. In *KES '05: Proceedings of the 9th International Conference on Knowledge-Based & Intelligent Information & Engineering Systems*, pages 716–722, 2005.

[68] Evgeniy Voronyuk. Java neural modeling framework new gui, May 2009.

[69] Darrell Whitley, Stephen Dominic, Rajarshi Das, and Charles W. Anderson. Genetic reinforcement learning for neurocontrol problems. *Machine Learning*, 13(2-3):259–284, 1993.

[70] Chad A. Williams. Genetically evolving optimal neural networks. In *Neural Networks and Expert Systems*, 2007.

[71] Stewart W. Wilson. Zcs: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994.

[72] Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[73] Ian Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Fransisco, CA, USA, 2nd edition, 2005.

[74] Q. Xu and Kenji Nakayama. Avoiding weight-illgrowth: Cascade correlation algorithm with local regularization. *IEEE International Conference on Neural Networks*, 3:1954–1959, 1997.

[75] Jingtao Yao and Chew Lim Tan. A case study on using neural networks to perform technical forecasting of forex. *Neurocomputing*, 34(1-4):79–98, 2000.

[76] X. Yao. A review of evolutionary artificial neural networks. *International journal of intelligent systems*, 8(4):539–567, 1993.

[77] Xin Yao and Yong Liu. Evolving neural network ensembles by minimization of mutual information. *International Journal of Hybrid Intelligent Systems*, 1(1-2):12–21, 2004.

[78] Lean Yu, Kin Keung Lai, Shouyang Wang, and Kaijian He. Oil price forecasting with an emd-based multiscale neural network learning paradigm. In *ICCS '07: Proceedings of the 7th International Conference on Computational Science, Part III*, pages 925–932, Berlin, Heidelberg, 2007. Springer-Verlag.

[79] Lean Yu, Shouyang Wang, and Kin Keung Lai. *Foreign-Exchange-Rate Forecasting with Artificial Neural Networks*. Springer Publishing Company, Incorporated, 2007.

[80] Lean Yu, Shouyang Wang, and Kin Keung Lai. An emd-based neural network ensemble learning model for world crude oil spot price forecasting. In B. Prasad, editor, *Soft Computing Applications in Business*, pages 261–271. Springer-Verlag, Berlin, Heidelberg, 2008.

[81] Lean Yu, Shouyang Wang, and Kin Keung Lai. Forecasting crude oil price with an emd-based neural network ensemble learning paradigm. *Energy Economics*, 30(5):2623–2635, September 2008.

[82] Chunkai K. Zhang and Hong Hu. An effective feature selection scheme via genetic algorithm using mutual information. In *Lecture Notes in Computer Science, Fuzzy Systems and Knowledge Discovery*, volume 3614, pages 73–80, 2005.

[83] G. P. Zhang. Neural networks for classification: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 30(4):451–462, 2000.

[84] Tianqing Zhu. Suspicious financial transaction detection based on empirical mode decomposition method. In *APSCC '06: Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing*, pages 300–304. IEEE Computer Society, 2006.