The London School of Economics and Political Science

Topological Optimisation of Artificial Neural Networks for Financial Asset Forecasting

Shiye (Shane) He

A thesis submitted to the Department of Management of the London School of Economics for the degree of Doctor of Philosophy.

April 2015, London

Declaration

Abstract

The classical Artificial Neural Network (ANN) has a complete feed-forward topology, which is useful in some contexts but is not suited to applications where both the inputs and targets have very low signal-to-noise ratios, e.g. financial forecasting problems. This is because this topology implies a very large number of parameters (i.e. the model contains too many degrees of freedom) that leads to over fitting of both signals and noise. This results in the ANN having very good in-sample performance on the data used for its training but poor performance out-of-sample for forecasting.

The main contribution of my research is to develop a new heuristic method called "ANN reduction" for optimising the topological structure of a feed-forward ANN in order to improve its out-of-sample performance (using an RMS measure). The research concentrated on the topological optimization of the graph representing an ANN, which reduces the effective degrees of freedom of the ANN whilst still maintaining its feed-forward (but incomplete) topology. Such reductions in the number of parameters have been attempted before in the literature, but our procedure is of a different (graph theoretic) nature and (in extremis) optimal for small-size ANNs.

Two applications of the ANN reduction are also implemented and programmed for empirical simulations. For this purpose, two datasets generated from deterministic functions and three datasets derived from foreign exchange market prices are used for evaluating the ANN reduction applications. These applications generate new ANN topologies with some clear performance advantages over those obtained by the best complete ANNs, improving the generalization (out-of-sample) performance by up to 27.6% compared to the complete ANN on the function generated datasets and up to 14.1% on the financial forecasting problem for the FX data.

Acknowledgements

Foremost, I would like to thank my advisors, Prof. Nicos Christofides, for guiding me through the whole PhD life. His continuous encouragement, invaluable guidance and constant support helped me greatly in pursuing my goals in life.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Nigel Meade and Dr. Cormac Lucas for their encouragement, insightful comments, and hard questions.

I owe special thanks to my mom, dad and my fiancé, for their continuous encouragement, care and support during my study. This dissertation is simply impossible without them.

Contents

Chapter 1

Introduction

1.1 The Financial Forecasting Problem (FFP)

Forecasting is a process of making statements about events, in which actual outcomes have not yet been observed (output) often by employing time series analysis on historical data. The basic idea of forecasting assumes that future occurrences are partially based on present and past observations, e.g. a forecast of variable X_t is $E(X_{t+h}|W_t)$ where W_t is the information set used and available at time t, and the forecast is for h periods ahead. The success of a forecasting model E relies on the accuracy of finding the relationships between particular input and output sets. Mathematical modeling and trend forecasting for stochastic financial time series has long been a popular research topic. Most modern methodologies have involved statistical models such as time series analysis. These models usually use real market data as inputs such as unexpected growth in GDP, changing industry structure or more specifically corporate fundamentals i.e. price earnings ratios, etc.

The FFP involves making statements about a financial asset's future value based on the observed information and assumptions that reflect existing conditions in the financial market. The term "forecasting" is reserved for estimating quantitative values at specific times in the future, thus the FFP is associated with quantitative methods and quantified information.

A commonplace application of the FFP is to estimate the holding period return of a portfolio in the next period. The return is forecasted based on a careful selection of prior information such as portfolio current value, recent share price volatility or closing exchange rate and various macroeconomic conditions.

Ultimately, if one can forecast a future event, one can exploit this informatioin to the advantage. The FFP is a means of evaluating the profitability of an investment opportunity and to form a basis on prioritising investment decisions. It can also assist investors to provide for their future investment needs and is invaluable to traders to translate the forecast into profits. Accurately identifying investment opportunities and cost is one of the most important elements in a financial market which leads to success.

Despite an enormous amount of research already carried out on this subject, the predictability of a financial market is still under debate [2, 3] especially at the turning of economic cycles. There are mainly two famous hypotheses that assert pessimism for the FFP. These are the Random Walk Hypothesis (RWH) and the Efficient Market Hypothesis (EMH).

The RWH claims that a stock price does not follow any pattern, so it is not possible to forecast future prices from past data[5]. According to this hypothesis, a trader investing money in the market is no different than throwing dice in a casino.

In an informationally efficient market, public news is promptly delivered to investors without any delay. The EMH claims that prices on traded assets already reflect all past publicly available information so any excess investment returns cannot be earned from investment strategies based on historical share prices or other historical data. Therefore, the EMH states that the FFP will not be able to reliably forecast future returns because it is based on observed information that is useless in an (information) efficient market[4, 6].

The two hypotheses have the same judgement regarding past information to future outcomes and render the FFP useless. Both have been continuously criticised from researchers. Lo and MacKinlay argue that prices do not follow a random walk[7]. Strong negative correlations have been found between past price earnings ratio and following stock returns[8], which is in conflict with the RWH. Most financial prediction models have demonstrated limited abilities to generate excess returns under most circumstances. Several studies[9, 10, 11, 30, 16, 32, 33, 35] provide some evidence of predictability emanating from short term horizon strategies although many models have failed to obtain a reliable forecasting function.

The main reason for this is that the relationship functions are usually weak, with very low signal to noise ratio, so it is necessary to construct an approximate representation for the function of interest by choosing an appropriate approximation. The approximation function is a particular function involving a number of free parameters. These parameters are then to be "tuned" (calibrated) and refined through a process called training or learning in order to minimise the difference between the approximation and the original signal.

The movement of some asset prices such as stock prices are stochastic in nature. Ultimately it argues for the existence of a stochastic mapping relationship between the historical and future values. In some commonly considered situations this data exhibits complicated statistical correlations. However, Timmerman [12] comments "the stronger (i.e. easier to detect) the evidence of past return predictability, the greater the expected decline in future predictability, as predictability patterns get more rapidly incorporated into current market prices." Further more, he invistigated predictablility of stock returns from their past values and found some modest evidence of local predictability in some short window over a long time horizon, but none of the forecasting models appear able to predict returns sustainablely [12].

1.2 Traditional forecasting methodologies

Many financial market practitioners and academic researchers have spent tremendous effort working on the FFP. Different methods have been developed for the specific needs of financial applications. The selection of the method depends on the objectives and the condition of the particular FFP. Traditionally there are two broad categories of methods for forecasting problems and they are differentiated through their handling of input and output data. The two categories are time series methods and causal methods. Recent developments of machine learning techniques have led to a new approach in the FFP and it is categorised as the third method. Table 1.2.0 lists quantitative forecasting methods that are relevant to many forecasting situations. A brief overview for the two traditional categories is introduced in this section following the machine learning methods in Section 1.3.

Forecasting Methods	Examples
Time series methods	Weighted Moving Average; Auto-Regressive Moving Average; etc.
Causal methods	Leading Indicator Analysis; etc.
Machine learning methods	Artificial Neural Networks; Support Vector Machines; etc.

Table 1.2.0 Examples of forecasting methods

1.2.1 Time series methods

In time series methods, past data are used as the basis of estimating future outcomes. The difficulty is to extract features including patterns, changes or disturbances in the past time series and project them with current data to forecast the future. Two time series methods will be discussed here, Moving Average(MA) and Auto-Regressive Moving Average (ARMA):

MA

MA is perhaps the most common type of time series forecasting method. A moving average MA_t of data at time t is the average of the values from time $t-\Delta$ to t, where Δ is a fixed time window. If those values are $x_{t-\Delta+1}, x_{t-\Delta+2}, ..., x_t$, then $MA_t = \frac{x_{t-\Delta+1}+x_{t-\Delta+2}+...+x_t}{\Delta}$. The idea of MA is to form a smooth version of the time series from consecutive past periods. One possible forecast for the next period is then $x_{t+1} = MA_t + \varepsilon_t$ where ε_t is white noise. MA_t is often adjusted by multiplying the time series values with a "weight" w_t (with $\sum w_t = 1$) , varying with t so that $MA_t = \frac{w_{t-\Delta+1} * x_{t-\Delta+1} + w_{t-\Delta+2} * x_{t-\Delta+2} + ... + w_t * x_t}{\Delta}$, e.g. giving greater weights to recent observations (with respect to time t) and less weight to older observations. This adjusted version is called Weighted Moving Average[1]. A specific example is the Exponential MA where the weights "die" exponentially backwards from t to $t-\Delta$, e.g. $w_t = 1 - exp(\frac{(t)-(t-1)}{\Delta})$.

MA is a very simple method but has slow reactions to a fast-changing market in that its value always lags the original time series, i.e. it introduces a phase-shift in the input data stream, which limits its usefulness when applying it to the FFP.

ARMA

ARMA models, sometimes referred to as Box-Jenkins, are typically applied to auto-correlated time series data[13]. ARMA uses a variable's past behavior to select the best forecasting model from a general class of models for forecasting. As its name states, the model consists of a MA component and an Auto-Regressive (AR) component.

The MA part or more precisely the Weighted Moving Average, computes a weighted linear combination of past errors: $MA(q) = c + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i+1} + \varepsilon_t$, where c is a constant, θ_i are parameters and ε is white noise and q is an integer defining the order of the MA process.

Similarly,the AR part is written $AR(p) = c + \sum_{i=1}^{p} w_i X_{t-i+1} + \varepsilon_t$, where c is a constant, w_i are parameters and ε is again white noise.

An ARMA(p,q) model with p autoregressive terms and q moving-average terms is therefore, $X_{t+1} = c + \sum_{i=1}^{p} w_i X_{t-i+1} + \sum_{i=1}^{p} \theta_i \varepsilon_{t-i+1} + \varepsilon_t$. To apply ARMA, there are three stages : 1, model identification and model selection; 2, parameter estimation; and 3, model checking. The first step is to ensure that the time series data conforms to the specifications of a stationary univariate process. This is done by checking the residuals which should be independent of each other and constant in mean and variance over time. Sometimes there exists seasonality in the time series and therefore seasonal differencing is necessary. Using plots for the autocorrelation and partial autocorrelation functions of the time series to decide which (AR, MA or both) component should be used in the model. The second step is to apply regression analysis to estimate the value of the best-fit parameters. The final step is to back-test the data from the estimated model. If the estimation is inadequate or the stationarity condition of the data series is not satisfied, another approach has to be used.[14]

1.2.2 Causal methods

Causal methods concentrate on finding the cause-result relationship between observable factors and outcomes. These methods assume that it is possible to have underlying factors that might influence the variable that is being forecast (the "target"). For example, including information about a recent number of passengers who booked flights may well improve the ability of a model to forecast the cost of fuel for an airline company in the coming period. [15] In general, the premise is that the cause of changes in the value of a target variable is closely associated with changes in one or more of the currently observable variable(s). Therefore, if future values of these "inputs" can be estimated, they can be used to forecast the value of the target. An important causal method is leading indicator analysis:

Leading Indicator Analysis (LIA)

Forecasting using financial and economic statistics in order to predict the value or direction of the target is called LIA. The term "leading" means that the independent variables x_t at time t, have a consequential influence on the dependent variable y_{t+1} at time $t+1$. For example, an analyst may find that a rise in oil price in the previous period indicates (leads) that the price of an asset will increase in future periods.

In general it is very difficult to identify leading indicators in any market. Often the only technique available is to list as "possible leading indicator" every conceivable variable which (under some "knowledge" or rationale) could be a possible candidate. One can then test (by linear regression, for example) whether a variable in this list is a suitable leading indicator or not, and select a small subset that are then proposed as leading indicator variables (inputs). Note that although a variable in this list is rejected by linear regression, it may be an important leading indicator but one whose effects on the target are nonlinear in nature and cannot be identified by linear regression, for example.

1.2.3 Practical discussion

Several studies examine the practical relationship between stock returns and fundamental stock variables. Variables such as earnings yield, cash flow yield, book-to-market ratio, and size are shown to have some power in predicting stock returns. Another input is provided by independent credit rating agencies such as Fitch, Moodys and Standard & Poor's who employ expert knowledge in making subjective credit analysis and rating decisions. Expert knowledge incorporates market factors and non-quantifiable influences to produce forecasts of possible credit events.

Fama and French [36] forecast the asset price in terms of three common risk factors: an overall market factor, a "size" factor relating to firm size, a "value" factor the book-to-market equity.

At the same time other researchers find that macroeconomic variables such as short-term interest rates, expected inflation, dividend yields, yield spreads between long and short-term government bonds, yield spreads between low grade and high grade bonds, lagged price-earnings ratios, and lagged returns have some power to predict stock returns. These studies lead to econometric forecasting approaches.

The major difference between time series methods and casual methods is that the former forecasting techniques use past trends of a particular variable to be the basis of forecasting the future of that variable; while the latter method uses past time series on many relevant variables to produce the forecast for the variable of interest. Note that applications combine both of the previous two methods. For general forecasting issues, one of the key problems encountered is to find the most appropriate forecasting method. When applying the above methods to the FFP, the problem is to understand the different forecasting methods and their relative merits to be able to choose which method to apply in a particular situation. However, given the large space of available information and complexity of the financial market, it is usually difficult to find an adequate answer.

1.3 Machine learning methods

Machine learning is a branch of artificial intelligence which is concerned with the design of algorithms that allows computers to learn based on empirical data. Next we will introduce the two important classes of learning: supervised learning and unsupervised learning, followed with a discussion of their applications in the FFP. Some examples are given in Subsection 1.3.2.

1.3.1 Machine learning classes

Supervised learning

Supervised learning generates a function that maps inputs to a desired target. The word 'supervise' means that the generated mapping function is inferred from supervised (or labelled) training data samples. Each training sample is considered to have a vector of inputs and a vector of a desired mapping target. The input-target pair is also called the supervisory signal as they are often provided by human experts labelling the training examples. A supervised learning algorithm analyses the training samples and computationally produces an approximate mapping function from the training process.

For example, given $(x_1, y_1), ..., (x_N, y_N)$ for a set of N training examples, x_i is the i-th input example and y_i is its label, a learning algorithm seeks the function $g : X \mapsto Y$, where X is the input space and Y is the output space.

The mapping function is also called a classifier in discrete cases or a regression function if the data is continuous. The objective of the exercise is to produce the correct output $g(x)$ given a new training input x through a learning algorithm, which is a process of rewarding or penalising the system if it gives the correct solution or not. The performance is indicated through a mismatch between output $g(x)$ and the training target y, which contains prior knowledge about the problem domain. If the learning algorithm is well trained and the system itself is predictable, the inferred approximation function should predict the correct output value for any valid input x' for out-of-sample cases $\bar{y'} \approx y' = g(x')$. This allows the learning algorithm to generalise correct mapping from the training data to other unseen instances and to predict outputs given inputs that it has not encountered. Tasks that fall within the paradigm of supervised learning are pattern recognition (also known as classification) and regression (also known as function approximation). The supervised learning paradigm is also applicable to sequential data, e.g. speech and gesture recognition. This can be thought of as learning with a 'teacher' in the form of a function that provides continuous feedback on the quality of solutions obtained thus far.

Unsupervised learning

Unsupervised learning algorithms study how to represent patterns from particular inputs, so that the system can reflect the statistical structure of the overall input collection. This type of algorithm is given no explicit target outputs nor needs any evaluations associated with inputs and state/environment. Different to supervised learning, the unsupervised learner "brings to bear prior biases as to what aspects of the structure of the input should be captured in the output"[29]. One method which has been suggested for unsupervised learning is density estimation techniques explicitly for building statistical models (such as Bayesian probabilities) of how underlying causes can create observed unlabelled inputs. A priori biases include implicit properties of the model, the parameters and the observed variables. Tasks that fall within the paradigm of unsupervised learning are in general estimation problems; the applications include clustering, the estimation of statistical distributions, compression and filtering.

When applying the FFP, for the need of forecasting in complicated market conditions and for different objectives, variations on the machine learning application have been developed. Next we will illustrate some implementations presented in the literature. There are other classes associated with machine learning which are largely extended or combined from the above basic three, so we will not discuss them explicitly.

1.3.2 Machine learning models

The well-known developed machine learning models are Artificial Neural Network (ANN), K-Nearest Neighbours (KNN), Bayesian Statistics (BS) and Support Vector Machines (SVM). This section focuses on ANN as it is used throughout the thesis.

- A large volume of literature has used ANN for financial and economic forecasting [18, 19, 20, 21]. The design of ANN was first inspired by the functional structure of biological neurons in the 1940s[17] and it is perhaps the oldest machine learning application. Through decades of development, ANN has become an information processing paradigm that consists of a number of interconnected processing neurons. Modern ANNs, like the human brain, learn by training in specific applications such as pattern recognition or financial data approximation. A detailed introduction of structure and implementation is presented in Chapter 2.

- KNN is a method for classifying objects based on the K closest training examples in the featured space. For regression analysis, it assigns the function value for the object to be equal to the weighted sum values of its k nearest neighbours. KNN is a type of instance-based learning, which means it approximates a function locally[23]. Therefore KNN is sensitive to the local structure of the data, resulting in uneven performance for time series prediction[54]

- The Bayesian framework for machine learning works in a similar manner to Baye's Rule $P(D) = P(D|M)P(M)/P(M|D)$. First, by examining each possible model to gain a prior probability $P(M)$; second, by examining each training data to evaluate the conditional probability of the data given in the model $P(D|M)$; finally, it enumerates through all training data samples and reasonable models to gain knowledge of the posterior probability of the model $P(M|D)$. In short, it learns probabilities about how evidence supports outcomes, and then predicts new evidence outcomes. For real world applications with large space of models, approximate Bayesian methods can be used[24].

- SVM constructs one or more hyper-planes to separate classes in the sample space, which can be used for classification. Based on the complexity of the separation, the original finite-dimensional space may be mapped into a higher dimensional space using a kernel function to ensure that data becomes linearly separable. A version of SVM for regression is proposed by Vapnik[25]; for time series prediction SVM has also produced some attractive results[26]. There are many corresponding evidence however, in general, SVM has not demonstrated any improvement over what can be achieved using ANNs.

In the supervised learning problem, one is usually presented with a time series sample of input-output pairs. The goal is to infer the mapping implied by the sample inputs and outputs. The reward/penalty function in ANN is called the cost function. The cost function is set in relation to the difference between the network's output (given an "input") and the actual data (which is sometimes called the "target"). A commonly used cost function is the mean-squared error, which aims to minimise the averaged squared error between the network's output and the sample target over all the time series. For this type of problem ANN is usually structured in multilayer perceptrons and the cost function is optimised using gradient descent. The multilayer perceptrons structure and the gradient descent optimisation process will be discussed in Chapter 2.

1.3.3 Machine learning in the FFP

The idea of using machine learning for the FFP has been popularised since the 1990s but its effect remains inconclusive . Although no financial institution has reported significant profit or loss resulting from particular machine learning models, machine learning technology remains widely employed as a tool in the financial investment sector. In recent years the presence of machine learning in both the academic literature and financial and economic industries has increased substantially, providing a new perspective to the agenda of finance and economics by their ability to handle large amounts of financial data and simulate complex models. An important application of computational finance is in forecasting asset prices in finance. Some of the currently used methods employ learning methods such as Support Vector Machines, Bayesian approaches and ANN.

In a modelling system, regardless of the type of problem and the method chosen, the objective criterion is always of central importance to assess the effectiveness of forecasting. While estimating prediction risk is important in its own right for providing a means of estimating the expected error inherent in the prediction output from a prediction process, it is also an important tool for assessing method selection, data preprossessing and improving the performance of the selected method.

A successful machine learning system for financial data forecasting is an ensemble of three crucial components: 1, data pre-processing; 2, forecasting algorithms; and 3, evaluation, tuning and forecasting. Each component itself is a challenge and can stand alone as a research topic. Since there are few proven prior general solutions, extensive experimentations are necessary, but even experience of failure can aid the development of the subject. However, notable in regards to the types of machine learning technique employed for the FFP, a recent survey shows there seems to be an increasing number of applications using existing ANN models. The development of ANN in the FFP focused on enhancing new training algorithms [31].

1.4 ANN and applications in the FFP

ANN is a type of machine learning system that was inspired from the structure of interconnectivity of neurons in the human brain as illustrated earlier. The learning process of ANN is also similar to that of human learning processes such that the nature of a relationship between incidents can be learned provided there is enough sample information for training. The actual performance of training and simulating is determined by the ANN's architecture, which is the interconnection between the processing elements (neurons) and their parameters. A classical type of ANN structure is the feed forward network, which is described in detail in later chapters. ANNs have been widely used for quantitative data modelling, such as financial stock price prediction because of its superior abilities to discover non-linear relationships in multivariate analyses. They have also been successfully used for low-level cognitive tasks such as speech recognition and character recognition and are now being explored for knowledge induction and decision support.

Referring to a previously mentioned example of credit rating companies we can use expert knowledge of finding methods for credit rating forecasting. Moody and Utan [32] find that when applying ANN for forecasting bond credit ratings, the results from ANN are more accurate compared to conventional results from linear regression. Earlier work from Dutta and Shekhar [33] also demonstrates a similar result by developing ANN to determine whether a correct credit rating is given. On the same training set ANN gave the best performance of 92.4% whereas a regression model only classified 64.7% correctly. Odom and Sharda [10] have also shown that ANN maybe more appropriate in predicting bankruptcy compared to traditional approaches.

Although there is an increasing amount of research which supports applying ANNs in a real business situation, opinions regarding their value are mixed. In particualr, Callen [34] used ANN forecasting of quarterly accounting earnings but the study revealed that neural-network models are not superior to linear time series models when the data are financial, seasonal and non-linear; this is partly because the data used for training contains a significant amount of noise and exhibits non-stationary characteristics. A study by Timmermann [12] also shows that in forecasting stock returns standard ANN does not produce better results compared to some other models including AR. Furthermore, Timmermann's emperical findings suggest that most of the time stock returns are not predictable but there exists only modest evidence of local predictability within a limited timeframe [12].

The research objective of this thesis is to explore some new possibilities to improve performance of ANNs, particularly in applications to the financial industry. In particular we consider the problems arising from over-training and the effects of too many degrees of freedom inherent in the ANN because of the excessive numbers of parameters to be estimated [73, 74, 75]. This exploration is to be modelled, simulated and compared based on applications of financial index forecasting using C++. Before proceeding to the research details, a complete background of ANN will be introduced in the following chapter.

Chapter 2

Background

2.1 Topology of ANN

Neural networks are assembled from interconnected neurons, thus it is important to understand the functionalities of these neurons. In the early days the development of artificial neural networks was attempted as a tool to understand the generic biological neuron in the bodies of mammals, such as the one displayed in Figure 2.1.0a, by simulating computational algorithms for them to use.

Biological neurons are cells. On the input end of the cell are a number of fine transmission lines namely *dendrites* with a tree shape resemblance. There are many varieties in the number of dendrites and shapes of neurons which reflect the information processes performed in neurons. The cell body is also referred to as the *soma* and contains the nucleus which has related cellular metabolic apparatus. Neurons are interconnected with each other through a long thin item called an *axon*. The axon is the transmission line for the signals produced by a neuron. For example, a spinal motor neuron in the backbone can have meter long axons running to the toes. On the end of the axon the transmission lines branch again in what is called a *terminal arborisation*. On the end of the axonal branches are synapses where the output of the neuron is produced. To portrait a standard procedure of information flow in the neuron, dendrites

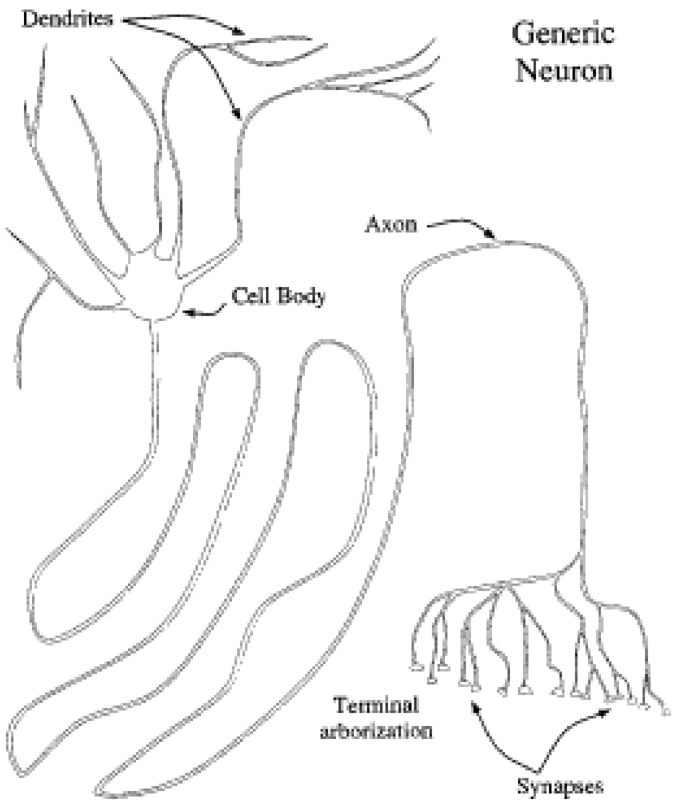
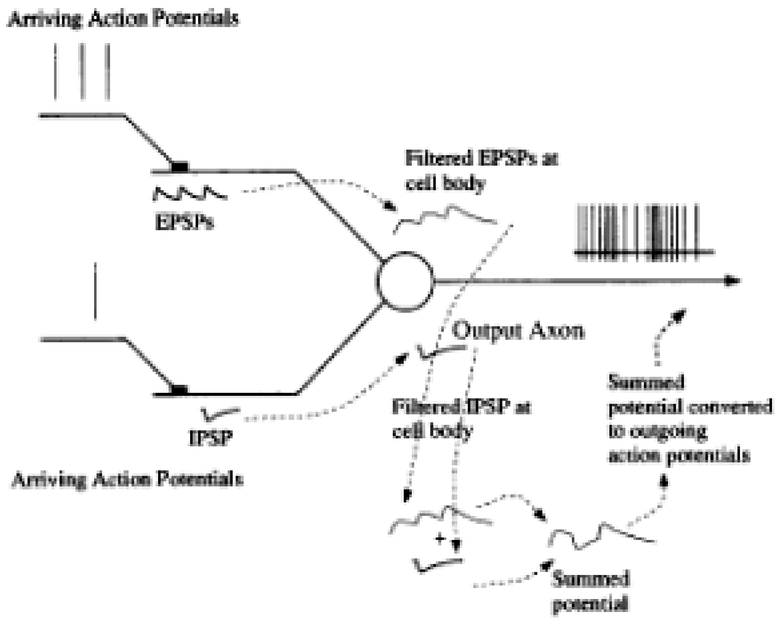


Figure 2.1.0a Generic biological neuron [39].

on top of the backbone in a human body receives input signals from brain cells, the soma then processes and integrates the inputs, and information is transmitted along the axon to the synapses and output to other neurons or to effector organs such as hand muscles in order to perform various movements [39].

Investigations have shown that the extent of dendritic spread allows significant spatial integration, which means multiple signals from different locations (or output from synapses) can be added. This signal is created in a form of electronic voltage difference built by ions in the body of cells and has two general types, namely *excitatory post-synaptic potential* (EPSP) and *Inhibitory post-synaptic potential* (IPSP). The EPSP helps to depolarise the ions in the cells to build up the voltage and IPES does the opposite. The soma is transforming electronic voltage difference to a firing pulse and is sent down the axon. This behaviour of soma is effectively like a voltage to frequency convert and was first theorised by Stevens Hillyard [98] in 1967. Figure

2.1.0b suggests the way EPSP and IPSP signals are received, low passed and integrated at cell body, the processed potential is then transmitted through the axon to other neurons.



Arriving Action Potentials
Filtered EPSPs at cell body
EPSPs
Output Axon
Summed potential converted to outgoing action potentials
Filtered IPSP at cell body
IPSP
Arriving Action Potentials
+
Summed potential

Figure 2.1.0b Steven's neuron [39].

Through decades of studying, research has developed many formal neural models described in the neural network literature. Among them, perhaps the most historically important, is the *McCulloch-Pitts neuron* [38]., first proposed in 1943. Twenty years earlier than Steven's neuron, McCulloch-Pitts neuron is a binary threshold logic unit as is known today. The neuron has a fixed threshold and responds to its two binary synapse inputs, which represent two excitatory signals passed through two dendrites to a nerve cell. Given different thresholds the neurons have behaviours similar to an electronic AND or OR logic gate. As illustrated in Figure 2.1.0c and Table 2.1.0 for example, with a threshold of 1, the neuron output is 1 if any of the inputs is 1; with threshold of 2, the output is 1 only if both inputs are 1.

Although McCulloch's approximations of real biological neurons is not close to what we found as the behaviour of neurons in the body today, the impact of the McCulloch and Pitts work was not among neuroscientists, but among computer scientists. It served as a foundation in developments of most modern generic neurons and complex neural networks[39].

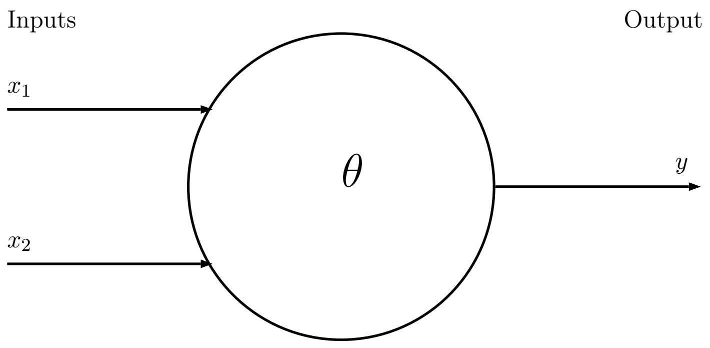


Figure 2.1.0c McCulloch-Pitts' neuron, a binary threshold logic unit

Input at time t		Output at time t+1
x_1	x_2	y
$\theta = 1$ (inclusive OR)		
0	0	0
1	0	1
0	1	1
1	1	1
$\theta = 2$ (AND)		
0	0	0
1	0	0
0	1	0
1	1	1

Table 2.1.0 Input and output table for McCulloch-Pitts neuron with thresholds 1 and 2

2.1.1 Feed-forward ANN

A *feed-forward neural network*, as illustrated in Figure 2.1.1a, is an ANN where connections between the neurons form a directed acyclic graph. The information flow is only in one direction: forward from an *input layer*, shown in yellow, through the green layers and to the *output layer* in blue. No matter how many neurons and how neurons in each layer are connected, there are no cycles in this network.

The above figure has two (green) layers with hidden neurons, which collectively are called the *hidden layer*. We will discuss this multi-hidden-layer ANN in Subsection 2.1.1. Before that, let us start with the simplest kind of feed-forward ANN: a *single neuron network*.

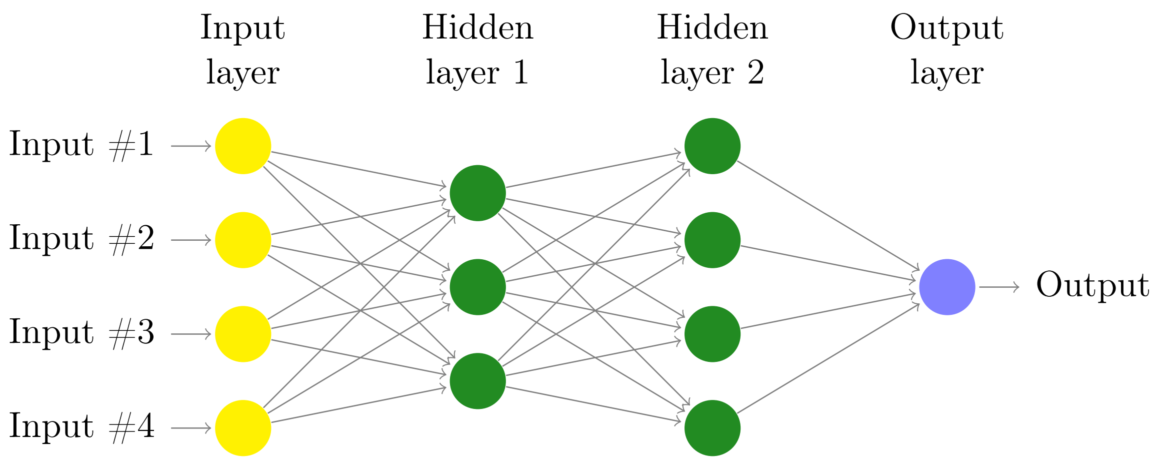


Figure 2.1.1a In a feed-forward ANN, information only flow in one direction

A single neuron network

Referring to Figure 2.1.1b The artificial neuron is an algorithm for supervised learning of combinations of inputs into possible outputs. The first step is a simple weighted sum of the synaptic inputs. The second step is a nonlinear activation function that operates on the node result in a nonlinear way to produce an output. Figure 2.1.1b shows a single neuron, which consists of n inputs and a bias input b, an assembler output u, and an activation function h. The inputs maybe rescaled by weights w based on the importance of the connections.

The summation step is a linear combination of dot products between the input vector $\{x_1, x_2, .., x_i, ..., x_{\hat{i}}\}$ and input weights vector $\{w_1, w_2, ...w_i, ..., w_{\hat{i}}\}$. In addition to the weighted sum, a constant bias b is added. The output of the node is u.

$$u = b + \sum_{i=1}^{\hat{i}} w_i x_i$$

The activation step consists of an activation function $h(\cdot)$ that defines the output signal y from the output neuron in terms of its net input signal.

$$y = h(u) = h(b + \sum_{i=1}^{\hat{i}} w_i x_i)$$

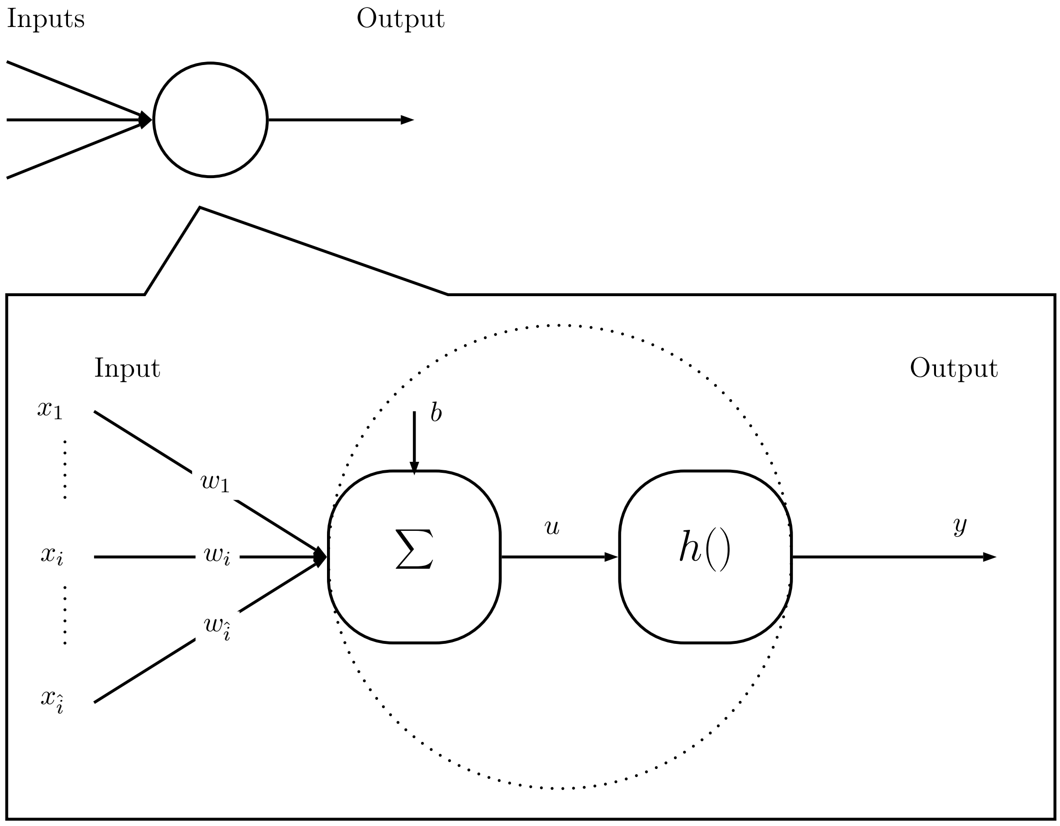


Figure 2.1.1b A hidden neuron with hyperbolic tangent activation function

Generally there are three types of activation function: binary (also referred to as a threshold, or step), linear and non-linear. For example, the threshold in McCulloch-Pitts neurons effectively acts as a binary activation function. Due to the fact that the weighted sum of the inputs is potentially unbounded, one reason for the incorporation of an activation function $h(u)$ is to dampen or magnify the weighted summation depending on the knowledge of classification problems to produce an output within an acceptable range. We briefly introduce one of the non-linear activation functions here but more detailed background in provided in Subsection 2.1.2.

To choose an activation function that matches the need for an ANN in application to a particular problem is not a simple task, but we have several conditions to help with the decision. The compression of the range of values of u from $[-\infty, \infty]$ to a limited range is also referred to in McClelland and Rumelhart as a *squashing function*[40]. If such an activation function is differentiable, monotonically increasing, and has the property that $-\infty < \lim_{u\to-\infty} h(u) < \lim_{u\to\infty} h(u) < \infty$, (which has theoretical significance which was pointed out in the 1970s[41]), the realised function is an s-shaped non-linear function curve. It is also an advantage to have an activation function which produces output in the range from 0 to +1 (a sigmoid, for example i.e. $\frac{1}{1+e^{-x}}$) or from -1 to $+1$ (a $\tanh(\cdot)$ function, discussed next). The hyperbolic tangent function shown in Figure 2.1.1c is chosen as the activation function for the ANN in much of this research to meet the needs of general approximation problems.

$$h(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

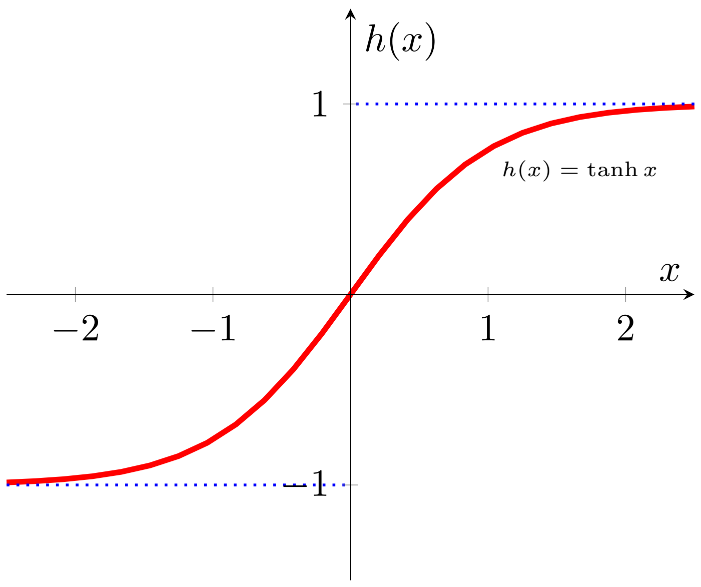


Figure 2.1.1c Hyperbolic tangent function

Multi-neuron network

A more complex feed-forward ANN is the *single-hidden-layer ANN*, which consists of a single hidden layer with a multiple number of artificial neurons. The neurons are interconnected into a feed-forward topology where each neuron in any layer has directed connections to the neurons of the subsequent layer (consider an input layer, a hidden layer and an output layer). Such single-hidden-layer ANN can be described using a three step model as in Figure 2.1.1d: a linear weighted summation step between the inputs and the next part, a non-linear step that does non-linear transformations and another linear summation step between the hidden layer to the outputs[42] (in this case a single output). Note that the function of output neuron is to sum the weighted output from hidden neurons and the output-neuron does not necessarily have to have an activation function.

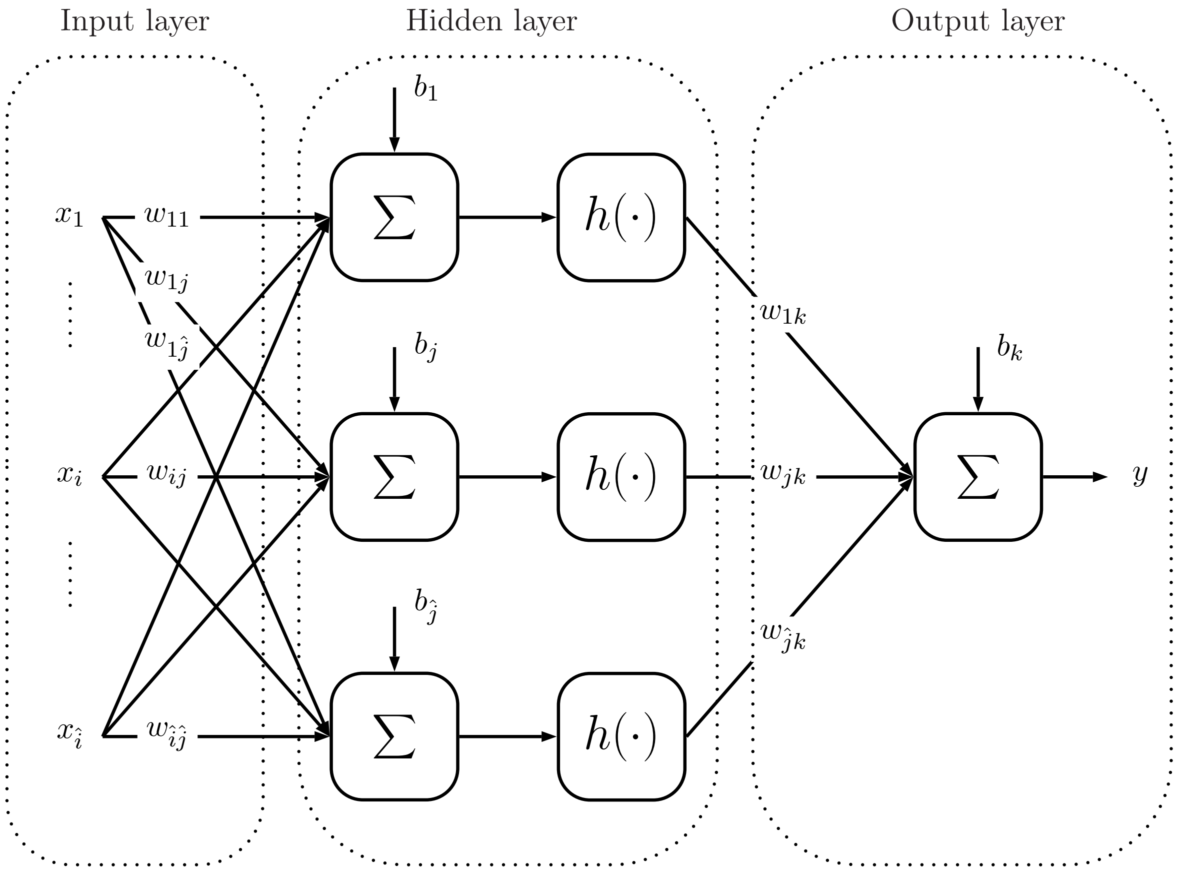


Figure 2.1.1d A multi-neuron network with a single-hidden-layer, where $h(\cdot)$ is the activation function

Suppose the index of nonlinear hidden neurons is j, and the index of output neurons is k. With input vector $[x_1, x_2, ..., x_i, ..., x_{\hat{i}}]$, the output of the jth summation neurons of the hidden layer is

$$b_j + \sum_i w_{ij} x_i$$

The outputs of the jth nonlinear neuron of the hidden layer is

$$h(b_j + \sum_i w_{ij}x_i)$$

The final output of the kth neuron at the output layer is:

$$y = b_k + \sum_j [w_{jk}h(b_j + \sum_i w_{ij}x_i)] \qquad (2.1)$$

For a *multi-hidden-layer ANN*, the feed-forward topology is simply cascading the output of one hidden layer to the input of another hidden layer. For example, a two-hidden-layer ANN is illustrated in Figure 2.1.1e.

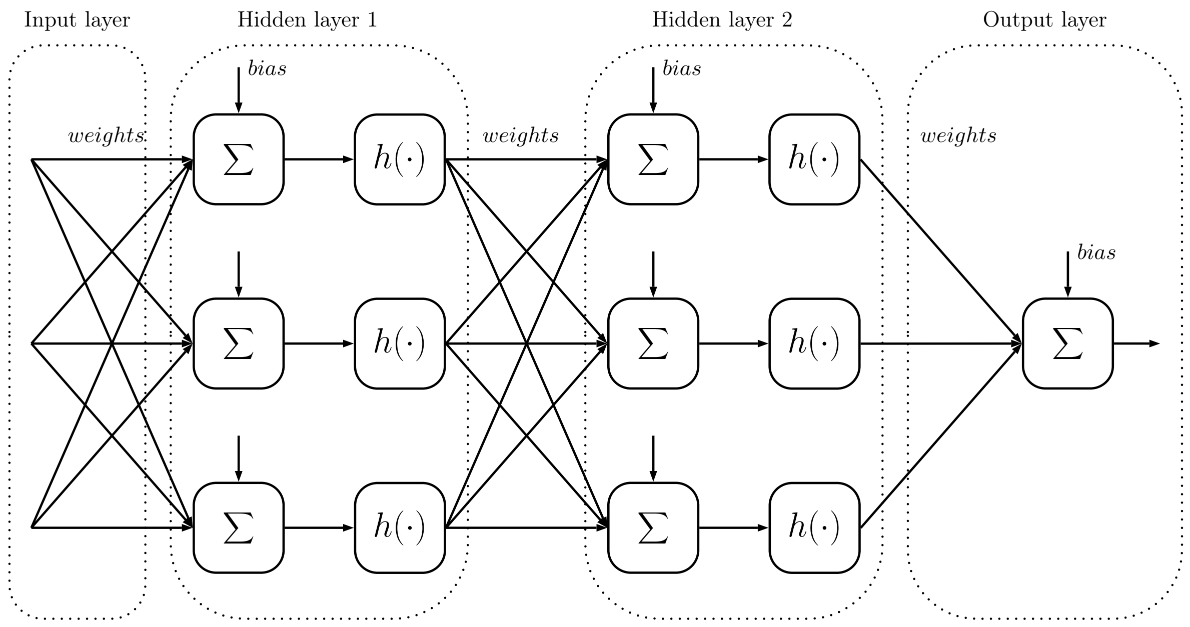


Figure 2.1.1e A multi-hidden-layer network

We have seen a multi-hidden layer ANN network can also be seen as a directed graph of neurons described in the beginning of this section. Figure 2.1.1f displays the graphical interpretation of a single-hidden-layer ANN. Arcs are "weighted" which means that the output signal from an arc is the input signal multiplied by this weight. Note that neurons on the output layer do nothing other than sum its inputs. This is because the use of non-linear neuron at the outputs would limit the range of possible outputs to a range attainable by the activation function, which would be undesirable in many cases.

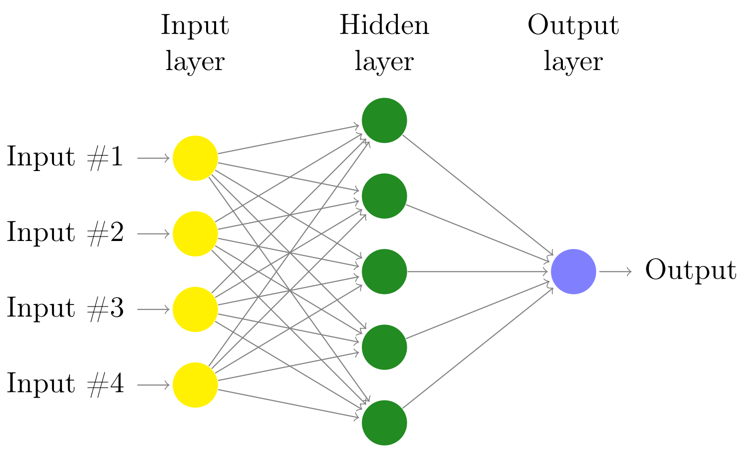


Figure 2.1.1f Illustration of a 4-5-1 ANN

The single-hidden-layer ANN in Figure 2.1.1f has four neurons in the input layer, one hidden layer with five neurons and one neuron in the output layer. Therefore, it is also called a 4-5-1 feed-forward ANN. Similarly, a 4-5-5-1 topology means that there are two hidden layers with 5 neurons each in the ANN. This definition allows us to clearly describe a feed-forward ANN topology completely.

2.1.2 Approximation using ANN

Suppose there is a mapping J of set X, which contains input x, to a real set $\Re$. The mapping function is defined as $J : X \to \Re$. An approximation problem can be described in the following way:

Given the topology of an ANN defined by the graph G (together with the activation of the vertices), and given a set w of weights of the arcs and biases b of the vertices, the output of ANN for a specific input vector X_n (n indicates the sample index) is y_n. The observed "target" of the mapping system when the system input is X_n is $\bar{y}_n$ so $\bar{y}_n = J(X_n)$. The error for the input is some measure of the difference between $\bar{y}_n$ and y_n.

In this section we focus on the problem of approximating the topology G. The process of finding suitable parameters w and b is through a training process, which is discussed in Section 2.1.

At this stage we want to state a well-known theorem which justifies the approach described above, namely "Universal Approximation theorem" states[45] that a single-hidden-layer feed-forward ANN with a finite number of hidden neurons can approximate a continuous function on compact subsets of $\Re^{\hat{\imath}}$, under mild assumptions on the activation function.

The formal definition (which we will not discuss further) is:

Theorem 1 (Universal Approximation theorem)**.** *Let σ be any continuous discriminatory function. Let $I_{\hat{\imath}}$ denote the $\hat{\imath}$-dimensional unit hypercube $[0,1]^{\hat{\imath}}$. The space of continuous functions on $I_{\hat{\imath}}$ is denoted by $C(I_{\hat{\imath}})$. Then given any arbitrary function $J \in C(I_{\hat{\imath}})$ and $\epsilon > 0$, there exist a finite sum $y = S(x) = \sum_{i=1}^{\hat{\imath}} \alpha_i \sigma(w_i^T x + \beta_i)$, $\forall \alpha_i \in \Re$, $\forall \beta_i \in \Re$ and $\forall w \in \Re^{\hat{\imath}}$. The functions $S(x)$ are dense in $C(I_{\hat{\imath}})$ so that S is an approximation of the function J and $|S(x) - J(x)| < \epsilon$ $\forall x \in I_{\hat{\imath}}$*

The discriminatory function σ Cybenko applied to prove the above theorem is a sigmoid, which becomes the justification of using some continuous, bounded and monotonically increasing functions as activation function. In this subsection, we will first discusses applications with simpler activation function such as linear and binary, then we will look into some intuitive proof of using the hyperbolic tangent as activation functions.

Linear activation function

If the activation functions are linear, then for any such network one can always find an equivalent network without implementing any hidden layers. This follows from the fact that the composition of successive linear transformations is itself a linear transformation. However, if the number of hidden neurons is smaller than either the number of input or output neurons, then the linear transformation which the network generates is not the most general possibility since information is lost in the dimensionality reduction in the hidden neurons.

In practice there are few approximation architectures using purely linear functions. Thus, in general there is little interest in multilayer linear networks, and we shall therefore mainly consider networks where the activation functions are non-linear.

Binary activation function

In this section we describe the kind of decision problems that can be represented by ANN with binary activation functions. In order to visualise the described we will limit the discussion to a 2-dimensional input vector.

A binary activation function h(x) takes the value 0 or 1 depending on whether $x < 0$ and $x \geq 0$

Considering a ANN with a single hidden layer (see Figure 2.1.2a). One can show that y can distinguish between input (x_1, x_2) that lie in a region as shown in Figure 2.1.2d(a) i.e. a separating hyperplane $\mathbb{P}$ of the Euclidean plane

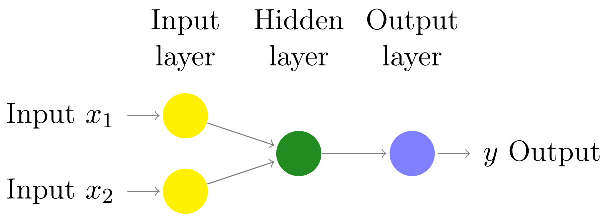


Figure 2.1.2a A 2-2-1 ANN with binary activation functions

Considering a ANN with two weight layers (see Figure 2.1.2b), we can show that y can distinguish points (x_1, x_2) in a convex polyhedron $\mathbb{P}$ from points outside this polyhedron as shown in Figure 2.1.2d(b).

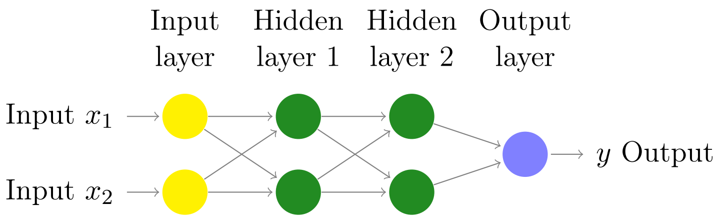


Figure 2.1.2b A 2-2-1-1 ANN with binary activation functions

Considering a ANN with three weight layers (see Figure 2.1.2c), we can also show that y can distinguish points (x_1, x_2) in an arbitrary subset of the Euclidean plane, including a collection on disconnected 2-D region that may or may not be contiguous as demonstrated in Figure 2.1.2d(c).

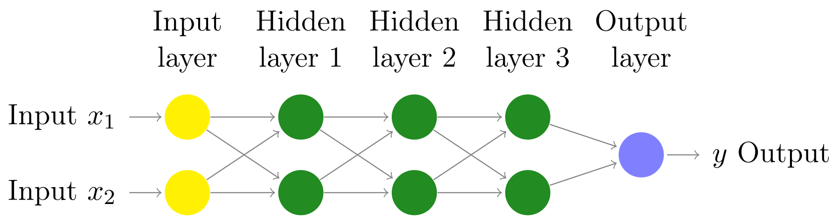


Figure 2.1.2c A 2-2-2-1-1 ANN with binary activation functions

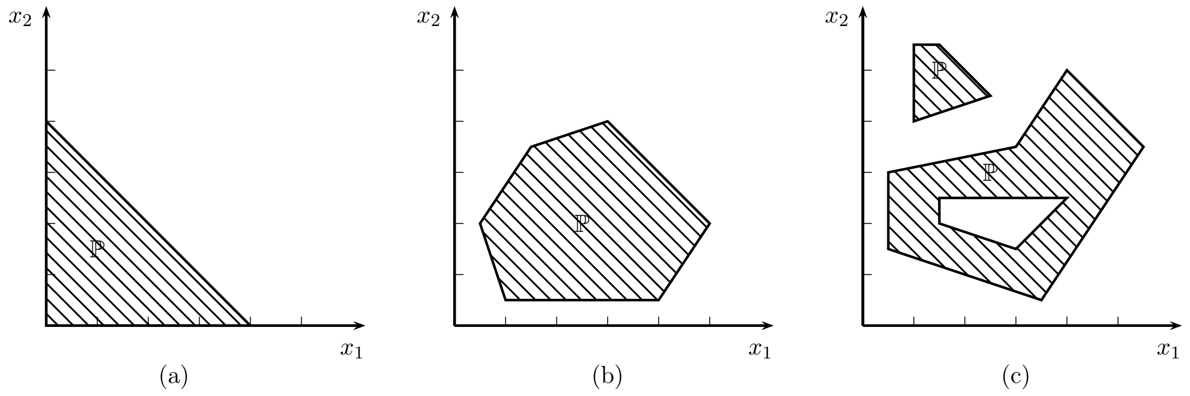


Figure 2.1.2d(a-c) Decision boundary produced from different two inputs ANN topologies with binary activation functions.

Non-linear activation function

It is shown previously in Figure 2.1.1c that a hyperbolic tangent function has the characteristics of being continuously differentiable, monotonically increasing and has the bounded property that

$$-\infty < \lim_{u\to-\infty} h(u) < \lim_{u\to\infty} h(u) < \infty$$

These characteristics become important properties in determining the topology and training the ANNs. Details about training processes are introduced in Section 2.2.

Using a non-linear function such as the hyperbolic tangent to be the activation function satisfies the general need of flexibility in approximation problems. It not only provides monotonic and continuous differentiable nonlinearities, but also approximates any linear activation function arbitrarily accurately. This can be achieved by rearranging the weights on the connections feeding into each neuron, including the bias, to be small so that the summed input lies on the linear part of the hyperbolic tangent curve around the center. Then the weights on the outputs of the neuron leading to the subsequent layer of neurons can be made in large sizes to re-scale the activation functions (with a suitable offset biases if it is necessary). Similarly, a hyperbolic tangent function can be made to approximate a step function by setting the weights and the bias feeding into a neuron to be very large in absolute values, so that the scaled input signal lies close to the two limits of the hyperbolic tangent curve.

The multi-hidden-layer ANN with hyperbolic tangent activation function has an important property relating to the approximation capability. Much research in the late 1980s has concluded that a multilayer feed-forward ANN with finite number of non-linear neurons in the hidden layers can approximate arbitrarily accurately any function f [42] that is $f : S \to \Re$ is continuous over closed and bounded sets S. Here we demonstrate an intuitive heuristic proof developed by Bishop [43] and graphically shows that a two-hidden-layer ANN is sufficient to approximate any smooth multivariate mapping with arbitrary accuracy.

Theorem 2. *A two-hidden-layer network with hyperbolic tangent activation functions could represent an arbitrary decision boundary to arbitrary accuracy.*

Consider a two-hidden-layer network with hyperbolic tangent activation functions in a two dimensional input space. Figure 2.1.2e(a) shows the output from hyperbolic tangent neurons in the first nonlinear layer, which is

$$h(b_j + \sum_i w_{ij} x_i)$$

The orientation of the S-shaped hyperbolic tangent surface is determined by the direction of vector w, its location is determined by bias b, and the steepness of the S-shape surface is determined by $\|w\|$.

The linear layer following the first nonlinear layer forms weighted linear combinations of these S-shaped surfaces. Consider the combination of two such surfaces in which we choose the second hyperbolic tangent surface to have the mirrored orientation of the first surface but displace it by a short distance. A ridge-like function as shown in Figure 2.1.2e(b) can be obtained.

Repeat the above process to construct two or more of the ridge-like functions with orthogonal orientations and combine them to produce a bump-like structure as shown in Figure 2.1.2e(c).

The bump-like structure has a central peak that presents many other ridges that stretch out to infinity. A second nonlinear layer can effectively act as a filter to remove the excess ridges. So feeding the bump-like output from the linear layer into a new layer of hyperbolic tangent neurons to isolate the central bump, is shown in Figure 2.1.2e(d).

Bishop[43]'s work highlights that any reasonable function can be approximated to arbitrary accuracy by a linear superposition of a sufficiently large number of localised bump-like structures, provided the coefficients in the linear combination are appropriately chosen. This superposition is performed by the output neuron, which has a linear activation function. This serves as a foundation to the architecture of feed-forward ANNs used throughout this thesis.

Further to the effort towards multi-hidden-layer ANN, as introduced in the begining of this section, the universal approximation theorem derived by Cybenko [45] states that all functions in Borel space can be approximated with arbitrary accuracy by a single-hidden-layer ANN with sigmoid activation functions The complete mathematical proof for this foundation theorem is discussed in Cybenko(1989)[45].

Here we outline an intuitive proof for the universal property derived later by Jones[99] to demonstrate the approximation ability using hyperbolic tangent functions.

Consider a two input function $J(x_1, x_2)$, using Fourier decomposition with coefficient number n_1 and n_2, this function can be approximated in linear summation within any given sum-of-squares error. Applying decomposition to the function in the variable x_2 results

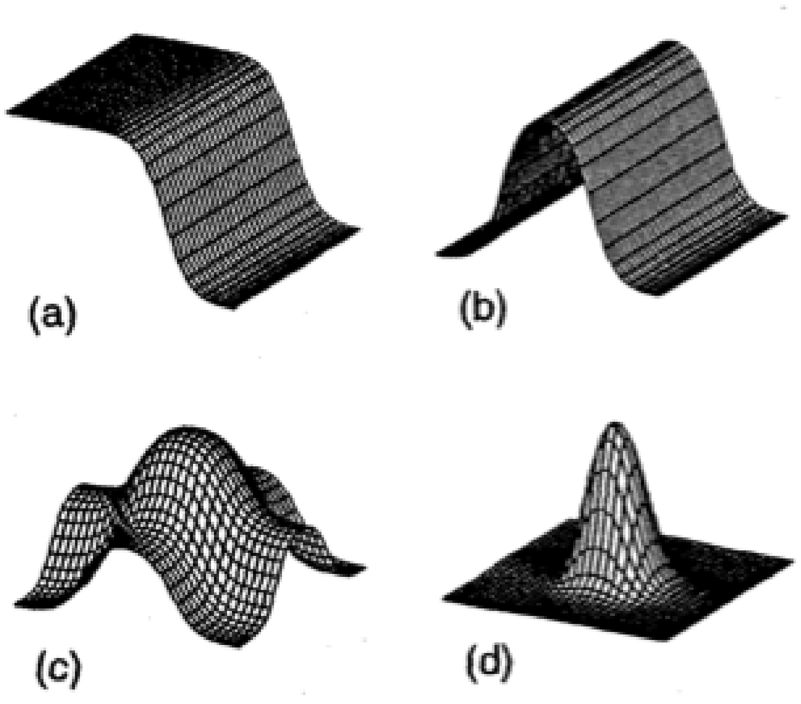


Figure 2.1.2e(a-d) Graphical illustration of two hidden layer ANN topologies for approximating smooth multivariate mapping to arbitrary accuracy [43].

$$J(x_1, x_2) \simeq \sum_{n_2} A_{n_2}(x_1)\cos(n_2 x_2)$$

where $A_{n_2}(x_1)$ means coefficients A_{n_2} are functions of x_1. Applying decomposition again on the coefficients leads to

$$J(x_1, x_2) \simeq \sum_{n_2}\sum_{n_1} A_{n_2 n_1}\cos(n_1 x_1)\cos(n_2 x_2)$$

Using the trigonometric identity $\cos(n_1 x_1)\cos(n_2 x_2) = \frac{1}{2}\cos(n_1 x_1 + n_2 x_2) + \frac{1}{2}\cos(n_1 x_1 - n_2 x_2)$

$$J(x_1, x_2) \simeq \sum_{n_2}\sum_{n_1} A_{n_2 n_1}[\frac{1}{2}\cos(n_1 x_1 + n_2 x_2) + \frac{1}{2}\cos(n_1 x_1 - n_2 x_2)]$$

It is notable that a cosine function can be approximated to arbitrary accuracy by a linear combination of a step function. A summation of cosine function can also be approximated to arbitrary accuracy. The case of a two input function $J(x_1, x_2)$ can also extend to any number of inputs. Recall that in the beginning of the section we described that a hyperbolic tangent neuron can approximate a linear or a step function. Thus, it forms the basis of a simple proof that a single-hidden-layer ANN with hyperbolic tangent activation function can approximate any continuous function to an arbitrary accuracy.

2.2 Training of ANN

Once a successive approximation paradigm is established, we can turn the focus to the tuning of function parameters for achieving the best approximation. These tuning processes are performed based on training the approximation architecture with a given set of actual input and target data pairs. A set of outputs can be realised from projections of the actual inputs through the approximation architecture. These realised outputs are then to be compared with the actual targets to evaluate performance of the approximation. If the set of realised results is not quite close to targets, the weights and biases need to be adjusted until a satisfactory evaluation for the approximation architecture is produced.

Solving an approximation problem consists of three steps as illustrated in figure 2.2.0. The first step is to choose a suitable deterministic function space in which the solution of the problem is to be approximated. The elements of these function spaces are those encompassed by the multilayer ANN. In the second step the variation problem is formulated by selecting an appropriate objective function defined in terms of the function space chosen before. The third step is to solve the reduced function optimisation problem.

In order to construct approximation architectures using ANN, the objective function must be defined as in Subsection 2.2.2. Training an ANN is an optimisation problem where the value of connection weights and biases on neurons is to be determined. Details of the solving process are presented in Subsection 2.2.3. Because the feed-forward ANN with hyperbolic

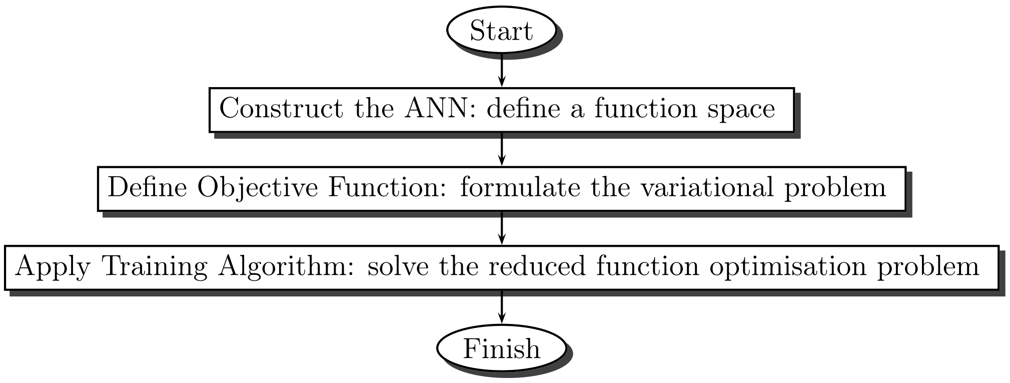

Start
Construct the ANN: define a function space
Define Objective Function: formulate the variational problem
Apply Training Algorithm: solve the reduced function optimisation problem
Finish

Figure 2.2.0 Solving an approximation problem

tangent function has a non-linear architecture, the objective function must be optimised using non-linear optimisation methods. Various methods are discussed in Subsection 2.2.4. First, In order to describe these procedures completely, we need to look at an example of input-target data pairs.

2.2.1 Sampled input-target data pairs

The input-target data set allows the approximation architecture to be evaluated and improved systematically. A good data set contains limited redundant data and significant information for feature selections. Referring to the FFP described in Chapter 1, for an ANN to approximate a desired function of financial asset price movement; a careful selection of input-target data set must be defined for the learning environment. The training data maybe obtained from various sources such as raw trading information or derived data. Raw information includes volume, price or daily price change. Derived data includes technical indicators, e.g. moving average, trend-line indicators; and fundamental indicators (e.g. intrinsic share value, economic environment, etc.)

Consider a collection of sample pairs as shown in Figure 2.2.1:

The first column is the index of samples and the next four columns include data of the market return, the asset return, the Libor rate and the yield curve over the corresponding sampling period. Some market research has concluded that all of these four time series are

Sample Index	Market Return in last period	Asset Return in last period	Libor Rate Change in last period	Yield Curve Slope in last period	Actual Asset Return in the next sample period
1	-0.006386	0.0033	-0.004542	-0.081465	-0.007278
2	-0.006639	0.019629	-0.005692	-0.096378	0.004855
3	-0.010802	0.023265	0.002947	-0.045364	-0.025915
4	0.008009	0.010157	-0.017075	-0.029022	0.103779
5	0.006414	-0.002706	-0.01802	-0.034067	0.115777
⋮	⋮	⋮	⋮	⋮	⋮
497	-0.001118	-0.006639	-0.000263	-0.064543	0.074455
498	-0.031406	-0.010802	0.015641	-0.057588	0.010986
499	-0.036141	0.008009	0.021997	-0.07632	-0.016566
500	-0.021018	0.006414	0.014133	0.03528	0.027425

Table 2.2.1 Example of financial time series samples

the factors in forecasting the return of the asset over the future period, as shown in the last column. For the FFP discribed in the above example, the construction of approximation ANN topology is therefore required to have 4 inputs and 1 output; the 500 input-target sample pairs are then used for training and evaluation of the ANN performance.

2.2.2 Objective function

Typically, the optimisation problem is of the least squares form

$$\begin{array}{ll} \text{minimise} & \dfrac{1}{N}\displaystyle\sum_{n=1}^{N} \|e(x_n)\|^2 \\ \text{subject to} & x \in \Re^n \end{array}$$

where $e(x_n)$ is the difference function between the target and the ANN output on sample n.

In the context of the approximation problem discussed in the beginning of Subsection 2.1.2, suppose the observed target $\bar{y}$ is approximated by an ANN with a vector of inputs X, the input-target data set consist of N pairs of Xs and $\bar{y}$s.

Suppose n is the index of a sample pair, the corresponding output of ANN from X_n is $y_n = S(X_n, W)$ where S is the function of ANN model and W is a vector of the weight and the bias parameters for the topology of the ANN. The difference between the target sample $\bar{y}_n$ and the ANN output y_n is also called the error function $e(X_n, W) = \bar{y}_n - y_n$, and the sum of error over all N samples is

$$E(W) = \sum_{n=1}^{N} \|e(X_n, W)\|^2 = \sum_{n=1}^{N} (\bar{y}_n - y_n)^2 \qquad (2.2)$$

The optimisation problem is then the problem of finding weights and biases W so that the objective function $E(w)$ is minimised.

2.2.3 Back-propagation

In the case of approximating a single target using a single hidden neuron as discussed in Section 2.1, if the nonlinear activation function $h(\cdot)$ is continuously reversible, we can transfer (from the output layer towards the input layer) the objective function by applying the reverse of the activation function on the target sample

$$\underset{w}{\text{minimise}} \qquad \frac{1}{N} \sum_{n=1}^{N} \left\| [h^{-1}(\bar{y}_n - b)] - \sum_i w_{ij} x_{in} \right\|^2$$

We can substitute the components in the square bracket with a constant number. As a result, solving this minimisation problem can be solved by the ordinary least squares (OLS) method where weight and bias of the ANN can be estimated from linear regression.

However in a multilayer ANN case, the above OLS approach cannot be used to determine the weights and biases other than the ones on the final linear summation layer. This is known as the credit assignment problem. Bertsekas [42] describes this problem thus: If an output neuron produces an incorrect response when the network is presented with an input vector, we have no way to determine which of the hidden neurons is responsible for generating the error; Therefore, there is no way of determining which weight to adjust or by how much. In other words, in order to adjust the weight parameter of a connection, the derivative of the weight with respect to the error function must be calculated.

Because of the credit assignment problem, research in ANN had stalled during the 1960-80s. It was not until 1974 and later when back-propagation was applied in the context of ANN, discovered by Werbos in his PhD thesis and discussed in detail by Rumelhart, Hinton and Williams[60], which led to a renaissance in the field of ANN research.

To explain the procedures of back-propagation, we use a network with three non-linear hidden layers as an example. Illustrated in Figure 2.2.3, considering a particular (middle hidden) layer with nonlinear activation neurons indexed with j, where on the adjacent layer close to the inputs are activation neurons indexed by i and neurons on the adjacent layer close to the outputs indexed by k.

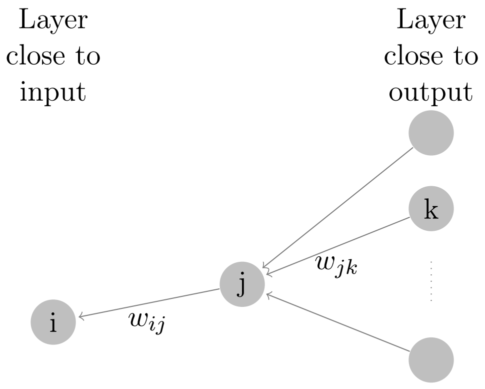


Figure 2.2.3 The direction of arrows illustrate how error is propagated backward from the neurons near outputs to the neurons near inputs in a feed-forward network.

For the purpose of simplification, all the bias terms are considered as extra units or inputs with fixed activation output of +1, so that the biases are not dealt with explicitly. Recall in Subsection 2.1.1 of this chapter we described u as the summed signal before the nonlinear transformation of a hidden neuron and let us also denote z as the post nonlinearly transformed signal,

$$u_j = \sum_i w_{ij} z_i \qquad (2.3)$$

where z_i is the output from a neuron i on the layer close to the inputs, which sends a signal via a connection to neuron j with an associated weight w_{ij}. The summed single u_j is then transformed by a nonlinear activation function in neuron j, so that

$$z_j = h(u_j) \qquad (2.4)$$

For the neuron in the subsequent layer, we have

$$u_k = \sum_j w_{jk} z_j$$

$$z_k = h(u_k)$$

Recall that the objective function is to minimise the sum of the error function (Equation 2.2)

$$E(X, w) = \bar{y} - S(X, w)$$

For an ANN with a fixed set of weights and biases, $E(X, w)$ can be calculated via forward propagation since it can be regarded as a forward flow of information through the network. The error function has a corresponding input vector, so that the summed error E is affected by a particular connection with weight w_{ij} only via the summed input u_j to activation unit z_k. The partial derivatives of error function to the respect weight is therefore:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial u_j} \frac{\partial u_j}{\partial w_{ij}} \qquad (2.5)$$

From Equation 2.3 we know that

$$\frac{\partial u_j}{\partial w_{ij}} = z_i \qquad (2.6)$$

Define

$$\delta_j = \frac{\partial E}{\partial u_j} \qquad (2.7)$$

Substituting Equation 2.6 and Equation 2.7 into Equation 2.5 we can obtain

$$\frac{\partial E}{\partial w_{ij}} = \delta_j z_i \qquad (2.8)$$

Equation 2.8 shows that the required derivative with respect to a connection weight w_{ij} can be obtained simply by multiplying the value of delta δ_j for the neuron at the output side of the connections by the value of z_i for the neuron at the input side of the connections ($z = 1$ if it is a bias).

For neuron k, we can also see that from Equation 2.3 and Equation 2.6:

$$\delta_k = \frac{\partial E}{\partial u_k} = h'(u_k)\frac{\partial E}{\partial z_k} \qquad (2.9)$$

Thus, to evaluate the δ for each hidden neuron we can apply the chain rule:

$$\delta_j = \frac{\partial E}{\partial u_j} = \sum_k \frac{\partial E}{\partial u_k}\frac{\partial u_k}{\partial u_j} \qquad (2.10)$$

where the summation is for all neurons on the layer of k from which neuron j sends signals, refers to Figure 2.2.3.

From Equation 2.3 and Equation 2.4 it is known that

$$\frac{\partial u_k}{\partial u_j} = \frac{\partial u_k}{\partial z_j}\frac{\partial z_j}{\partial u_j} = h'(u_j)\sum_k w_{jk} \qquad (2.11)$$

Substituting Equation 2.7 and Equation 2.11 into Equation 2.9, the formula of back-propagation can be obtained

$$\delta_j = h'(u_j)\sum_k w_{jk}\delta_k \qquad (2.12)$$

The reason of the name back-propagation is that from Equation 2.12 the value of δ for a particular hidden neuron on any layer can be obtained by propagating the δ's backwards from neurons on the layer close to the output as illustrated in Figure 2.2.3. Since the value of the δ on the output layer is known, all the deltas for the hidden neurons in a feed-forward network can be evaluated recursively by applying Equation 2.12.

As stated in Bishop[43], the back-propagation procedure can be summarised in the following five steps. For evaluating the derivatives of the error function with respect to the weights:

1. Initialise all weights and apply an input vector X to the ANN
2. Forward-propagate through the network using Equation 2.3 and Equation 2.4 to find the activations of all the hidden and output neurons.
3. Evaluate the delta δ_k for all the output neurons using Equation 2.9.
4. Back-propagate the deltas δ_k using Equation 2.12 to obtain δ_j for each hidden neuron in the network.
5. Use Equation 2.5 to obtain the required derivatives of the error function with respect to the weight for a connection.

Note that in this thesis, it is assumed that all activation functions are the same. However, in real applications they may be different. In those cases the output of each individual activation function is simply kept track of.

Once the derivatives (sensitivity) of error functions with respect to all weights and biases are obtained through back-propagation, the next step is to adjust the weight parameters which could reduce the error by the maximal amount. This adjustment is a steepest descent and the method is repeated many times until the error minimisation criteria are satisfied. The ANN's parameters are effectively 'trained', so that the total error between the target samples and the output of the model is minimised.

2.2.4 Conjugate gradient methods

Introduced in the previous chapter, the back-propagation training method was popularised and contributed to much research development of ANNs throughout 1980s to early 1990s, but soon after it was replaced by the conjugate gradient method. This is because a pure multidimensional steepest descent method produces a very large number of small steps that zig-zag in the space of variables, leading to a very large number of iterations. The conjugate gradient method improves the situation.

Conjugate Gradient

A typical unconstrained optimisation problem is formulated in the form of $min\left\{f\left(X\right):X\subset\Re^n\right\}$, where $f:\Re^n\mapsto\Re$ is continuously differentiable. The domain X of the function f is called the searching space or the choice set, while the elements of X, set x, are called candidate solutions or feasible solutions. The goal of the optimisation problem is to find a solution $\{x_0, x_1\cdots\}$ that minimises the objective function $f(X)$.

Note that in the case of training of ANN models, the optimisation objective is the error function $E(w)$ and the search space is the weights and bias set W, but the conventional symbols $f(X)$ with $x\in X$ are used throughout the following section for the convenience of expression.

There are many approaches for non-linear optimisation problems; perhaps one of the most popular methods is Conjugate Gradient (CG) searching.

CG methods comprise a class of unconstrained optimisation algorithms which are characterised by low memory requirements and strong local and global convergence properties. Although there is continuous development in algorithms, they commonly have the same structure expressed by the iterative form

$$x_{k+1} = x_k + \alpha_k d_k,$$

where d is the descent direction of the objective function $f(x)$ and α is a positive step size obtained from line search. The descent direction d is further generated iteratively from

$$d_{k+1} = -g_{k+1} + \beta_k d_k$$

where g_k is the initial descent direction when k=0 or the gradient of the objective function $f(x_k)$ when $k > 0$

$$d_0 = -g_0, \qquad g_k = \nabla f(x_k)^{\mathrm{T}}.$$

Here β_k is the direction parameter and is calculated differently in various algorithms.

The first non-linear CG method was proposed by Fletcher and Reeves [50] in 1964

$$\beta_k^{FR} = \frac{\|g_{k+1}\|^2}{\|g_k\|^2}$$

Polak and Ribiere [51] proposed another CG method in 1969

$$\beta_k^{PR} = \frac{g_{k+1}^{\top} y_k}{\|g_k\|^2}$$

while $y_k = g_{k+1} - g_k$

Once a search direction is completed as a conjugate gradient, a step size must be computed in order to move along this direction, a number of alternatives exists:

1) <u>*Golden Section:*</u>

Define the initial bracket interval $[a, b]$ (Figure 2.2.4a) that contains only one local minimum.

x_1 and x_2 are selected by the golden ratio rule $R = \frac{\sqrt{5}-1}{2}$, $x_1 = b - R(b-a)$, $x_2 = a + R(b-a)$. Compare $f(x_1)$ $f(x_2)$ and choose the larger, in this case $f(x_2) > f(x_1)$ so $x_2$is the new upper bracket. The new interval is between $a'$and b'. $x_1'$and x_2' are then calculated and compared, and iterations are repeated until $\|x_2 - x_1\|$ is small.

2) <u>*Brent's method:*</u>

The Van Wijngaarden–Dekker–Brent method combines root bracketing, bisection and

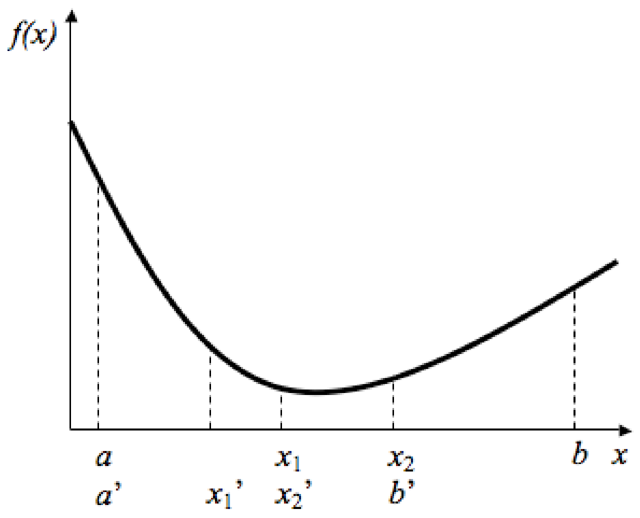


Figure 2.2.4a Illustration of Golden Section method

quadratic interpolation. Because any quadratic function $f(x)$ can be interpolated inversely using three prior points x_1,x_2,x_3; Brent suggests that we should search for a value at $f'(x) = 0$ which is then taken as the next estimation of the root x. Figure 2.2.4b explains this graphically. A parabola (dashed line) is drawn through the three original points ①, ②, ③ representing x_1,x_2,$x_3$on the given function $f(x)$ (solid line). The function is evaluated at the parabola's minimum $f'(x) = 0$, ④, representing x, which replaces point ③. A new parabola (dotted line) is drawn through points ①, ④, ②. The minimum of this parabola is at point ⑤, which is close to the minimum of the original function.

Let

$$
\begin{aligned}
v_1 &= \frac{[f(x) - f(x_1)][f(x) - f(x_2)]x_3}{[f(x_3) - f(x_1)][f(x_3) - f(x_2)} \\
v_2 &= \frac{[f(x) - f(x_2)][f(x) - f(x_3)]x_1}{[f(x_1) - f(x_2)][f(x_1) - f(x_3)} \\
v_3 &= \frac{[f(x) - f(x_1)][f(x) - f(x_3)]x_2}{[f(x_2) - f(x_1)][f(x_2) - f(x_3)}
\end{aligned}
$$

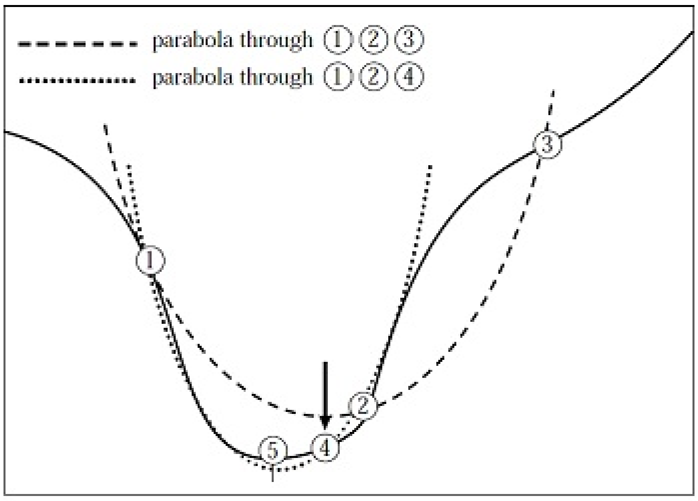


Figure 2.2.4b Illustration of Brent's method

The inverse quadratic interpolation is

$$x = v_1 + v_2 + v_3$$

Setting $f'(x) = 0$ gives an estimated result for the minimum root, which can be written as

$$x = x_2 + \frac{\frac{f(x_2)}{f(x_1)}[\frac{f(x_1)}{f(x_3)}(\frac{f(x_2)-f(x_1)}{f(x_3)})(x_3 - x_2) - (1 - \frac{f(x_2)}{f(x_3)})(x_2 - x_1)]}{(\frac{f(x_2)}{f(x_1)} - 1)(\frac{f(x_2)}{f(x_3)} - 1)(\frac{f(x_1)}{f(x_3)} - 1)}$$

Next, replace x_2 with x_1 or x_3 (the one closer to x). If, however, x is not between x_1 and x_3 then bisection steps are used instead. The iteration repeats until $\|x_2 - x\|$ is small.

3) Hanger-Zhang *and line search based on Wolfe conditions:*

In 2005 William Hager and Hongchao Zhang[52] proposed a nonlinear conjugate gradient method and an associated implementation based on an inexact line search. Its direction parameter is updated as:

$$\beta_k^{HZ} = (y_k - 2d_k \frac{\|y_k\|^2}{d_k^\top y_k})^\top \frac{g_{k+1}}{d_k^\top y_k}$$

where $y_k = g_{k+1} - g_k$.

The inexact line search described in their implementation is the improved version of Wolfe conditions combined with bisection and secant methods. Consider a search interval $[a, b]$

– L0. Terminate the line search if the Wolfe condition is satisfied.

– L1. $[a_{k+1}, b_{k+1}] = secant^2(a_k, b_k)$.

– L2. if $b_{k+1} - a_{k+1} > \gamma(b_k - a_k)$, then $c = (a_{k+1} + b_{k+1})/2$ and $[a_{k+1}, b_{k+1}] = update(a_{k+1}, b_{k+1}, c_{k+1})$.

– L3. Repeat L0.

The following is the list of $update(a, b, c)$, $secant^2(a, b)$ and the parameter used for this project

1) Interval update $[\bar{a}, \bar{b}] = update(a, b, c)$

– U0. If $c \notin (a, b)$, then $\bar{a} = a$, $\bar{b} = b$, and return.

– U1. If $f'(c) \geq 0$, then $\bar{a} = a$, $\bar{b} = b$, and return.

– U2. If $f'(c) < 0$ and $f(c) \leq f(0) + \epsilon_k$, then $\bar{a} = a$, $\bar{b} = b$, and return.

– U3. If $f'(c) < 0$ and $f(c) > f(0) + \epsilon_k$, then set$\hat{a} = a$, $\hat{b} = b$, and do the following

  ∗ a. Set $d = (1 - \theta)\hat{a} + \theta\hat{b}$; if $f'(c) \geq 0$, then set $\bar{a} = \hat{a}$, $\bar{b} = d$, and return.

  ∗ b. If $f'(c) < 0$ and $f(c) \leq f(0) + \epsilon_k$, then set $\bar{a} = d$, and go to step a.

  ∗ c. If $f'(c) < 0$ and $f(c) > f(0) + \epsilon_k$, then set $\hat{b} = d$, and go to step a.

2) Double Secant Step $[\bar{a}, \bar{b}] = secant^2(a, b)$, $secant(a, b) = \frac{af'(b) - bf'(a)}{f'(b) - f'(a)}$.

– S1. $c = secant(a, b)$ and $[A, B] = update(a, b, c)$.

– S2. If $c = B$, then $\bar{c} = secant(b, B)$.

– S3. If $c = A$, then $\bar{c} = secant(a, A)$.

– S4. If $c = A$ or $c = B$, then $[\bar{a}, \bar{b}] = update(A, B, \bar{c})$. Otherwise $[\bar{a}, \bar{b}] = [A, B]$.

3) Line Search Parameters

– $c_1 \in (0, 0.5)$, used in the Wolfe conditions (Armijo rule)

– $c_2 \in [c_1, 1)$, used in the Wolfe conditions (Curvature condition)

– $\epsilon \in [0, \infty)$, used in the approximate Wolfe termination (U2)

– $\theta \in (0, 1)$, used in the update rules when the potential intervals $[a, c]$ or $[c, b]$ violate the opposite slope condition contained (U3)

– $\gamma \in (0, 1)$, determines when a bisection step is performed (L2)

The Wolfe Condition consists of two rules: i) The Armijo rule and ii) The Curvature condition: i) ensures that the step length α_k decreases $f(x)$ sufficiently, and ii) ensures that the slope has been reduced sufficiently.

The Armijo rule holds if the following inequality is satisfied

$$f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k d_k^\top \nabla f(x_k)$$

with $0 < c_1 < 1$.

It is illustrated in Figure 2.2.4c. The solid curve is $f(x)$, the dashed line is $\nabla f(x_k)$, and the dotted line is determined by $f(x_k) + c_1 \alpha_k d_k^\top \nabla f(x_k)$ with $c_1 = 0.5$. In order for step length α to satisfy the Armijo rule, $f(x_k + \alpha_k d_k)$ has to lie in the interval between the dashed and dotted line, which prevents the line search from taking too long steps.

The Curvature condition holds if another inequality is satisfied

$$d_k^\top \nabla f(x_k + \alpha_k d_k) \geq c_2 d_k^\top \nabla f(x_k)$$

with $0 < c_1 < c_2 < 1$.

However, the above condition can result in a value for the step length that is not close to a minimiser of $f(x)$. To force α_k to lie close to a critical point, the modified version is applied

$$\mid d_k^\top \nabla f(x_k + \alpha_k d_k) \mid \geq c_2 \mid d_k^\top \nabla f(x_k) \mid$$

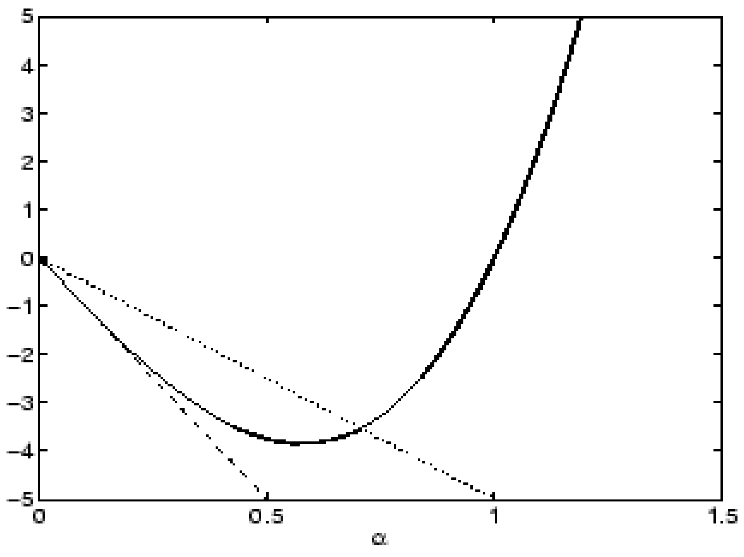


Figure 2.2.4c Illustration of Armijo Rule

Together with the Armijo rule and the modified curvature condition, we have this so-called strong Wolfe condition. The details of its implementation are described in Subsection 2.3.2.

Combining the direction methods and root-finding algorithms, the computational method of CG can be described in six steps:

- Step 1 Calculate the steepest direction: $\Delta x_n = -\nabla_x f(x_n)$,
- Step 2 Compute β according to one of the formulas,
- Step 3 Update the conjugate direction: $\Lambda x_n = \triangle x_n + \beta \Lambda x_{n-1}$,
- Step 4 Perform a line search: optimise α_n, $\arg\min_{\alpha_n} f(x_n + \alpha_n \Lambda x_n)$,
- Step 5 Update the position: $x_{n+1} = x_n + \alpha_n \Lambda x_n$,
- Step 6 Repeat step1 to 5 until stopping criteria is reached.

In the following content, three examples, a linear CG problem and two non-linear CG problems are presented. The line search algorithms are the Golden section and the Brent Method while the Fletcher–Reeves and Polak–Ribire methods are chosen to update the β parameter. These are compared later to Hager and Zhang, a method with a search based on Wolfe conditions in Subsection 2.3.2.

Example of Linear Conjugate Gradient Search

Considering the linear system $Ax = b$, the conjugate gradient method can be performed directly to solving the linear equation $A^\top Ax = A^\top b$ with iterative steps for programming purposes.

Given that $Ax = \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ with the initial guess $x = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$. Table 2.2.4a is the result generated from the minimisation code.

Iteration	x_1	x_2
0	2	1
1	0.23565	0.338369
2	0.0909091	0.636364

Table 2.2.4a Result of CG on a trivial linear problem

The results indicate that with the Fletcher–Reeves method, it takes only 2 iterations to achieve the stopping criteria $\|g_n\| < 0.001$. The linear problem is simple to compute.

Comparative study of Non-Linear Conjugate Gradient Methods

Whereas linear conjugate gradient methods can be applied directly, the non-linear conjugate gradient method is generally using its gradient $\nabla f(x)$ alone, under the condition that the function is twice differentiable at its minimum.

Consider a square polynomial $f(x) = (x_1 + 2x_2 - 7)^2 - (2x_1 + x_2 - 5)^2$. The contour of $f(x)$ in Figure 2.2.4d indicates that the function has a single minimum at $f(1, 3) = 0$.

Start with an initial guess $x = \begin{bmatrix} 10 \\ 10 \end{bmatrix}$, Table 2.2.4b-2.2.4c is the result generated from iteration of the following methods respectively.

Again, the β uploading method is Fletcher–Reeves; the Golden section method is used for line search. It takes 12 iterations to achieve the stopping criteria $\|g_n\| < 0.001$.

However, with the Polak–Ribire and Brent methods it takes 40 iterations (Table 2.2.4c).

This result indicate that for this square polynomial function, the Fletcher–Reeves and Golden Section methods result in a better performance.

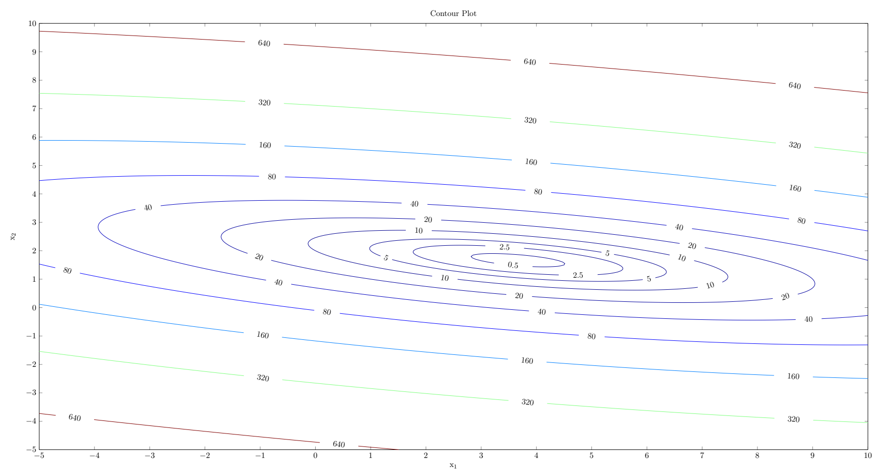


Figure 2.2.4d Contour of the cubic polynomial $f(x)$

Iteration	x_1	x_2
0	10	10
1	1.88198	2.10439
2	1.55093	2.47514
4	1.10952	2.92999
7	1.0049	2.99528
12	0.999749	3.00022

Table 2.2.4b Result of CG with Fletcher–Reeves method

Iteration	x_1	x_2
0	10	10
1	3.21678	3.40262
2	2.36927	2.67623
4	1.92212	2.39415
7	1.45018	2.57007
11	1.18439	2.8239
20	1.02364	2.97525
40	1.00027	2.99971

Table 2.2.4c Result of CG with Polak–Ribire methods

Now consider an objective function of five variables:

$$f(x) = 100(x_2-x_1^2)^2+100(x_3-x_2^2)^2+100(x_4-x_3^2)^2+100(x_5-x_4^2)^2+(1-x_2)^2+(1-x_3)^2+(1-x_4)^2+(1-x_5)^2$$

From intuition the minimum of this function should be at $x_1 = x_2 = x_3 = x_4 = x_5 = 1$

resulting $f(x) = 0$. It is first simulated with the Fletcher–Reeves and Golden section methods. The initial guess is $x_1 = x_2 = x_3 = x_4 = x_5 = 2$. Executing the programme shows that it takes 1099 iterations to reach the final result $x_1 = 0.998593$; $x_2 = 1.00251$; $x_3 = 0.997066$; $x_4 = 1.00035$; $x_5 = 0.998386$. But the Hanger-Zhang method takes only 76 iterations to reach the final result $x_1 = 1$; $x_2 = 1$; $x_3 = 1$; $x_4 = 1$; $x_5 = 1$. Based on the result of these comparisons, the Hanger-Zhang method is chosen as the optimisation method for ANN design in this thesis.

2.2.5 Local minima and global minima

In mathematics, the maxima and minima of a function, known collectively as extrema, are the largest and smallest value that the function takes at a point either within a given neighbourhood (local extremum) or on the function domain in its entirety (global extremum). For example, as illustrated in Figure 2.2.5, the local maxima and minima, and the global maxima and minima is shown for $\cos(3\pi x)/x$, between $0.1 \leq x \leq 1.1$.

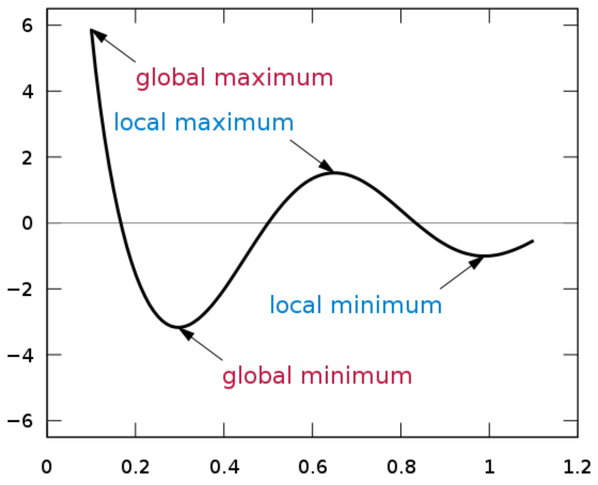


Figure 2.2.5 Local minima and global minima

In the case of ANN the target function is an error function, the error-surface is formed by a combination of all values of weights and bias defined by the ANN topology. To minimise the error function described in Subsection 2.2.2 is the goal of training.

However, a significant drawback of the training procedure is that the error-surface may contains local minimum. The result of the gradient descent method is dependent on the initial stating position defined by the usually randomised initial weight and bias at the beginning of training. Such a method, however, does not guarantee the finding of a global minimum.

2.3 ANN for forecasting

2.3.1 Over-fitting problem

In machine learning, over-fitting occurs when an approximating model learns to describe random error or noise instead of just the underlying relationship. Over-fitting generally occurs when a model is over parameterised, such as for an ANN that has too many weight and bias parameters relative to the actual required number for the given observations. A model which has been overfitted will generally have good approximation performance to the training data pairs (in-sample) but have poor predictive (out-of-sample) performance to unknown data by exaggerating minor fluctuations caused by noise contained in the data.

A model is typically trained by maximising its performance on some sets of training data, i.e. to minimise the difference of in-sample estimating and actual data. However, the efficacy of the model should be determined not by its performance on the training data but by its ability to perform well on unseen data, i.e. to minimise the difference of out-of-sample estimation and actual data. Over-fitting occurs when a model begins to memorise training data rather than learning to generalise from the features of the training set. As an extreme example, if the number of parameters is the same as or greater than the number of observations, a simple model can learn to perfectly predict the in-sample training data simply by memorising the training data entirely. Such a model will typically fail drastically on unseen out-of-sample data, as it has not learned to generalise at all.

In training of an ANN, the error function is assumed to reach a state where it will also be able to predict the correct output for other unknown data, thus generalising to situations which were not presented during training . However, especially in cases where learning is performed over too long a period or where training examples are rare, the ANN may adjust to very specific random features of the training data, which have no relation to the target function. In this process of over-fitting, the performance on the training examples still increases while the performance on unseen data becomes worse, i.e. the RMS error is reduced when applying it to training data instead of all data.

The potential of over-fitting depends not only on the number of parameters and data but also the conformability of the approximating model with the data shape, and the magnitude of model error compared to the expected level of noise or error in the data.

Even when the fitted model does not have an excessive number of parameters, it is to be expected that the fitted relationship will appear to perform less well on a new data set than on the data set used for fitting[53]. In particular, the value of the coefficient of determination will shrink relative to the original training data. In statistics, the coefficient of determination R^2 is used in the context of statistical models whose main purpose is the prediction of future outcomes on the basis of other related information. It is the proportion of variability in a data set that is accounted for by the statistical model. It provides a measure of how well future outcomes are likely to be predicted by the model.

In order to avoid over-fitting, it is necessary to use techniques such as cross-validation or early stopping.

1. Cross-validation: when designing an ANN, cross-validation can be implemented by separating the sampled data set into two, one for training and one for validation. For every iteration step of the training, the validation data set is applied as an out-of-sample test. As illustrated in Figure 2.3.1, where training error is shown in the blue line, validation error is in the red line and both are as functions of the number of training cycles. If the validation error increases while the training error steadily decreases, where it is marked by an exclamation, this indicates that further training does not result in better generalisation. Such a situation concludes that a over-fitting may have occurred. The best

predictive and fitted model would be where the validation error has its global minimum, i.e. at $x*$.

2. Early stopping: setting a maximum number of training iterations is effective when the convexity error function becomes very small and inefficient as well as preventing over-fitting from over-training.

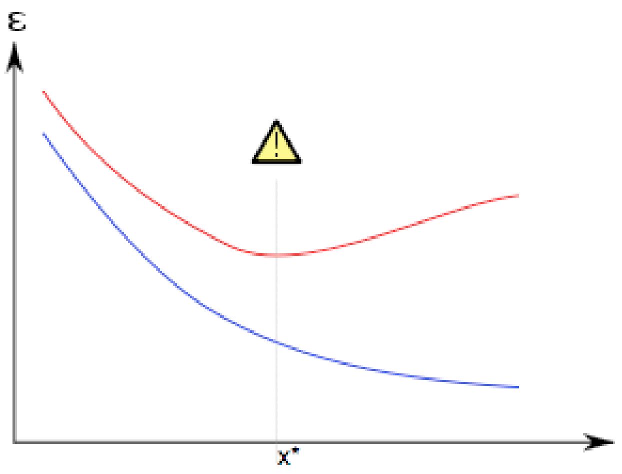


Figure 2.3.1 Training error and validation error against training iteration

Generally speaking, the basis of these techniques are either (1) to explicitly penalise overly complex models, or (2) to test the model's ability in generalising by evaluating its performance on an out-of-sample set of data, which is assumed to be in a similar form of some typical data that the model will encounter in its applications.

2.4 Inputs to ANN

Traditional ANN models are limited by the dimensionality of the problem space because the complexity of an ANN approximation model is dependent on the topology as well as the number of inputs. The number of parameters needed to describe the connections for a single-hidden-layer ANN with $\hat{\imath}$ inputs, $\hat{\jmath}$ hidden neurons and $\hat{k}$ outputs is $\hat{\imath} \times \hat{\jmath} + \hat{\jmath} + \hat{\jmath} \times \hat{k} + \hat{k}$. For example, the 4-5-1 ANN introduced in Figure 2.1.1f of Section 2.1 uses a total of 25 parameters to specify its connectivities between the input and hidden layer and 6 parameters between the hidden and output layer; if one more input is assigned to the network then the number of parameters has to increase by 6. In the other words, even with the smallest increase of the input dimensions, it gives rise to large regions in terms of complexity of calculation for the model. Also, following the analysis by Scott[97], increasing the dimension of input requires an exponential increase in the number of samples required in order to map a given function over the model parameter space with sufficient confidence.

A problem that cannot be ignored is that although such a black box has powerful ability in approximating a complex function deterministically, training samples are also one of the critical factors in successfully performing the mapping. In real life applications, ANN models may be supplied with insufficient or uninformative input-output sample pairs, which is called under-specified; or more inputs than is strictly necessary, which is called over-specified; or noise from sampling superfluous; or even weakly informative data. In either one or more cases, the performance of the model will be poor simply because some of the behavior of the output remains unexplained or biased by noise from the selected input sample. In most situations, it is reasonable to argue that some expert knowledge of the system can be put into consideration, so a reasonable set of candidate input-output sample pairs are selected based on this knowledge when building the model.

Therefore, there are two key considerations when treating the sample pairs for training an ANN: dimension of data and relevance of data. A priori assumptions in the development of ANNs is that at least one or more of the available candidate sample inputs is capable of describing some, if not all, of the output behavior. There are techniques in helping to determine the nature and relative strength of these relationships, which in return provides a base for selecting inputs as well as data dimension reduction.

Limited by the scope of this research, we will not focus on discussing the problems of data dimensionality and the related techniques for data reduction, but the importance of these topics cannot be ignored in the ANN applications.

2.5 Survey of ANN topological optimisation

Despite the progress of development in ANN, there are many issues that remain to be solved for achieving a general applicability in real-world problems. One of the most important tasks is to develop an approach for determining the right ANN network topology to solve a specific problem.

Formal learning theories have been used to support the construction of machine learning systems by estimating the necessary network size. Most work is based on the pioneering Vapnik–Chervonenkis theory[75] which contributes to the study of the relationships between the complexity of a machine learning system and the number of training samples available[73, 74]. Under the assumption that the network is capable of exactly representing (with zero error) an arbitrary training set consisting of t different patterns, Yu's theoretical work proves that such an exact representation using a single hidden layer requires a maximum of t-1 neurons in that layer. It is of course, obvious that this upper bound is of no practical significance.

Studies of previous ANN implementations has shown that the dilemma of network design stems from the fact that both large and small networks exhibit a number of benefits[56]: for a network with an excess number of free parameters, i.e. too many neurons or weight connections generate a complex decision surface[62], although it is arguable to say that such a complex surface can benefit the generalisation ability[64]. Baum and Haussler have shown theoretically that networks with few free parameters exhibit a better generalisation performance[65]. This view is also supported by some experimental work[63, 66, 67, 68, 69]. Moreover, a smaller trained network means a simpler interpretation of embedded knowledge and thus the rule extraction procedure [69, 70].

Optimal network architecture is a compromise of two things: (*i*) the ANN is large enough to learn the problem efficiently and (*ii*) small enough to generalise the application well.

To determine a suitable ANN architecture, one straightforward approach is to try random networks with different sizes. This process is computational infeasible for real world applications. As a result, more systematic approaches are needed.

In order to facilitate complexity optimisation in ANNs, several algorithms are developed, which are described in the following subsections. From the procedures carried out in each one, these algorithms can mainly be divided into three groups: growing methods, pruning methods, and a combination of the two[72]. The details in algorithmic procedure for some of the methods are presented in the next chapter.

2.5.1 Growing methods

As its name indicates, growing methods search for an optimum architecture by increasing the network's size. It is also called constructive methods (or additive), usually starting with a minimal network and adding one or more new neurons and connections during its optimisation process until the subsequent training process satisfies the constraints. Kwok[81] in his survey paper categorised the single-valued growing methods by differentiating their training processes, as demonstrated in Table 2.3.2a. The representative algorithms for individual categories are introduced as follows.

Training process	Representative algorithms
Training with memorisation	Resource-Allocating Network
Training the whole network	Dynamic Node Creation
Training only the new hidden neurons	Projection Pursuit Regression

Table 2.3.2a Categories of single-valued state transition mapping

- Platt describes the Resource-Allocating Network (RAN) as a single-hidden-layer network of locally tuned hidden neurons whose responses are linearly combined to form an output response[79]. It is essentially a summing radial basis function that interpolates the continuous function on a compact interval with arbitrary accuracy[80, 77]. The RAN starts with zero hidden neurons and grows by allocating hidden neurons subject to the present observed output satisfying one or more growth criteria. The changes between each consecutive observation can then be adapted as sequence learning problems such as sequential recognition and on-line prediction. This is an important advantage of RAN. A detection method may also be defined, whereby the process memorises recent significant differences from state transitions and applies them to learning. The objective of learning is to gradually approach the appropriate complexity of the network that is sufficient to provide an approximation to the underlying mapping function. Much of the work that followed is derived from the principle of the resource allocating network (RAN) or its variations[76, 77]. A drawback of these algorithms is that their convergence properties are unknown[81].

- Dynamic Node Creation (DNC), which was first introduced by Ash, is another type of growing method[82]. In a single hidden layer case, the state transition proceeds by adding one additional sigmoid hidden neuron at one time. The process always applies on the same hidden layer for multilayer networks. Different to RAN, after every neuron addition, the network is retrained completely for the DNC network. Some similar approaches are produced with a slight alternation in the neuron addition rules[78, 83]. These algorithms are simple to implement, and the convergence to the target function follows directly from the universal approximation property of the underlying architecture. However, the major difficulty lies in the tremendous increase in computational requirements in complex problems when the network becomes large.

- The third common type of growing method is inspired from a statistical technique called Projection Pursuit Regression (PPR) developed by Friedman[84]. PPR overcomes the curse of dimensionality by projecting the input vector onto a one-dimensional hyperplane. It then applies a nonlinear transformation of the input variables and adds them in a linear fashion. The main difference is that the functions fitted in PPR can be different for each combination of input variables and is estimated one at a time and then updated with the weights, although these are all specified upfront and estimated simultaneously in the ANN. So when applying PPR as a growing method, the hidden neurons to be added are in complicated transfer functional forms rather than just sigmoidal, and the training is only applied upon the hidden neuron. For nonparametric problems such as financial data forecasting, smoothing splines are used to obtain transfer functions of the hidden neurons. Although using nonparametric hidden neurons is a computationally intensive process, strong convergence to the target function for the network sequence produced by these algorithms has been proved[85].

In general, growing methods have several advantages. One is that the initial network for a constructive problem is simple and straightforward to implement. The characteristic of growth ensures that the methods search for smaller solutions which meet the objectives. There are also some disadvantages to these methods. Many constructive algorithms employ a greedy approach to neuron addition, which may be trapped in local minima in most cases. This suggests that the growing method is sensitive to the initial condition.

2.5.2 Pruning methods

The approach taken by the pruning algorithms described in this section is to train a network that is larger than necessary and then remove the neurons or connections that are the least relevant to the problem. A trimmed system also favours improved generalisation while its complexity is reduced simultaneously. The processes of removal are mainly divided into five broad groups as proposed in Reed's survey paper[64], which is listed with respective characteristics in Table 2.3.2b.

Pruning algorithms	Characteristics
Sensitivity calculation methods	Modify on trained network and prune upon significance in changes.
Penalty-term methods	Modify the objective function to penalise unnecessary neuron.
Interactive pruning	Remove hidden neurons with linear correlated or constant outputs.
Genetic Algorithms	Remove or retain neurons through mutation from a pool of networks.

Table 2.3.2b Brief introduction of pruning optimisation process

- One of the most commonly developed pruning algorithms is the sensitivity calculation method. It first estimates the sensitivity of the error function with respect to the temporary removal of one or more network connections, then pruning is applied based on the significance of changes resulting from the modification. Karnin developed an approximation to the weight sensitivity calculation, so the expression for connection pruning is easy to calculate and does not need for a separate sensitivity-calculation phase[86]. Le Cun deduced another iterative pruning procedure called optimal brain damage, which measures the salience of weights by expanding the error function using a Taylor series, then reducing the expression to a second order derivative of error function with respect to the weight[88]. A neurons pruning method called 'skeletonisation' algorithm, which was introduced by Mozer, estimates the sensitivity of input and hidden neurons by calculating the derivative of the error function with respect to an introduced gating neuron[87]. Based on Mozer's contribution, Segee further found that summing variance of weights of connectors into a neuron is a good indicator of the neuron's relevance and that the output error is a monotonic function of the relevance of this neuron[91]. The drawback of sensitivity calculations is that it requires a large amount of computational effort. Approximations of derivative calculations are introduced to simplify the process but with additional noise too.

- Penalty-term methods are some of the other common approaches used in network pruning. These methods modify the error function by adding a penalty term so that the training process effectively prunes a network by driving one or a group of weights of the connections to zero. The design of penalty-terms varies based on the needs or criteria from problem sets, but most work is derived from two fundamental terms: weight decay and sensitivity measure. The weight decay term, first introduced by Plaut et al., is a summation of the square of weights multiplied by a decay function, thus it is differentiable with respect to the weights. By adding it to the objective function, weights which are not essential to the solution decay to zero during the back-propagation learning[58]. Instead of using the function of weights as a decay term, Chauvin proposed a sensitivity measure in order to prune hidden neurons. The term adds a summation of monotonic functions of hidden neurons to the training objective, which effectively measures how much the hidden neuron's activity varies over the training patterns. The output's variance is an indication of the importance of the hidden neurons: if a neuron's output changes significantly, it is more sensitive to inputs and probably encodes important information; in contrast, neurons which are less important can be removed by driving all output connections to zero[90]. The problem of the penalty-term is that this approach may eliminate small weights that are indeed effective to the overall architecture of the networks and may introduce additional local minima during training on the error surface.

- Interactive methods require the designer to explicitly inspect the target network's elements one by one. It removes hidden neurons whose outputs are linear correlated or constants. Sietsma used a heuristic approach to exam the network. Reed, citing Sietsma states 'although for a simple demonstrated network, the method can find a smaller network that solves the problem and has good performance in out-of-sample tests, it is unable to learn the problem reliably when a random network is encountered'[64].

- Combining evolution strategies and genetic algorithms produces the concept of Breeder Genetic procedures which select an optimal network topology[94, 93]. These procedures generally involve four steps: generate initial pool, mating, retraining and evaluation. The basic idea behind the genetic algorithms is that the optimal solution will be found in areas of the solution space that contain good solutions and that these areas can be identified through robust sampling. A typical procedure starts from a completely trained network. Random disconnection is performed on the network while creating a population of pruned versions. Each version is then fed with the same data set for evaluation. The top performing pruned networks are randomly mated and produce some further pruned networks which retain only the parent mutual connections. These networks mutate with none or some additional connections. Last, the mutated networks can be trained and evaluated for performance or proceed for further mating and mutation process until a satisfactory solution is found. The advantages of genetic algorithms are that they are less likely to be trapped in local minima. It also allows other methods described above to be applied.

An important question for a pruning algorithm is when to stop the pruning. Separate validation data sets may be required to answer the question. Alternatively, sensitivity methods may stop pruning automatically when the variation of sensitivity jumps above a threshold. Dynamic control factor may be introduced before the training in the penalty-term methods to balance the scaling factor of the additional penalty-terms.

For the purpose of research, I have reviewed many growing and pruning algorithms based on different techniques and also briefly discussed each algorithm with its own advantages and limitations. The latest developments in ANN topological optimisation can be traced with references from these techniques[95, 72]. In the implementation process one can use more than a single technique in searching for optimal network architecture. Combinatorial algorithms of growing and pruning have also been developed in the hope of capturing the benefit from both algorithms' advantages[72]. More sophisticated methods sometimes reach better solutions, but the accuracy is usually compensated by the disproportional increase in the computation time and space.

Chapter 3

Theoretical framework

3.1 Representing ANN topological structures as graphs

The network architecture for an ANN is the critical factor in determining the model's overall performance on both the in-sample training result and the generalisation ability. The connectivity and the transfer function on each neuron in the network are pre-specified before training the parameters. As an example, a complete feed-forward ANN has many arcs to connect every neuron from one layer to every neuron in the adjacent layers; a random non-zero initial weight is attributed to each arc and is subject for training. The complete ANN can be represented as a complete tripartite graph and this representation is introduced in Section 3.1.

It is mentioned in Chapter 1 that the major problem in using ANN in the FFP is that the model usually has too high a degree of freedom, which leads to a good result for fitting in-sample training data but a poor result when applying it to forecasting on out-of-sample data. Therefore, once a complete ANN is fully trained, finding an optimal network connectivity with the improved generalisation performance compared to the original ANN is called the ANN topology optimisation problem. More detail about the problem is discussed in Subsection 3.1.2.

As introduced in the previous chapter, there are existing techniques for automatically improving the architectures of an ANN. The majority of these existing techniques can be categorised in two: additive methods and pruning methods. Despite the increasing number of techniques developed, however, little progress has been made towards the success of systemically constructing a near-optimal architecture because many techniques produce output which is trapped at local optima [101, 102]. Designing the optimal architecture for an ANN can be formulated as the topology optimisation problem: given some performance (optimality) criterion, the performance level of all architectures forms a discrete solution space. Miller et al. [103] stated several characteristics of this space:

1) The space is infinitely large since the number of possible neurons and connections is unbounded;

2) The space is non-differentiable since changes in the number of neurons or connections are discrete and can have a discontinuous effect on the ANN's performance;

3) The space is complex and noisy since the mapping from architecture to its performance is complex and indirect, and dependent on the evaluation method used;

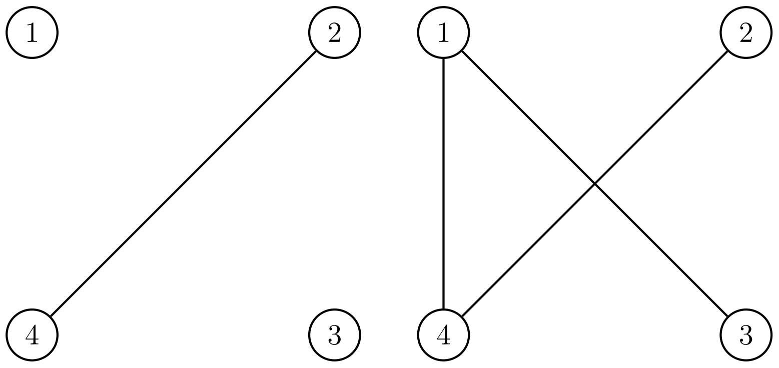
4) The space is multi-modal since different architectures may have similar performance.

These characteristics inspired the research in this thesis where a heuristic technique is applied for searching for the optimal architecture of an ANN without altering the number of neuron but changing the set of connections. As a result, a heuristic algorithm named ANN-reduction is proposed for systematically optimising existing sub-optimal networks. The ANN-reduction algorithm is developed to repetitively generate modifications on the topology until the output "can no longer be improved". This framework is discussed in Subsection 3.2.1.

Two procedures have been designed as the practical implementations of ANN-reduction algorithm. These are: 1) Enhanced ANN-Reduction Procedure (ERP) and 2) Cascaded Enhanced ANN-Reduction Procedure (CERP). These implementations are introduced in Subsection 3.2.2 and 3.2.3.

3.1.1 Graph and ANN network architecture

A *graph* G is a finite set V of elements called *vertices* together with a set A of pairs of vertices called *arcs.* Below is an example of a simple graph: vertex set $V = \{1, 2, 3, 4\}$ and arc set $A = \{(2, 4)\}$, which is shown in the left of Figure 3.1.1a where circles are vertices and arcs are lines that connect the vertices. In some cases there are weights or distances associated to the arcs, this is called a *weighted graph.* If there is a path from every vertex to every other vertex, the graph is called a *connected graph*; otherwise, the graph is a *disconneted graph.* The left of Figure 3.1.1a is a disconnected graph and the right figure is a connected graph.



Vertices set $V = 1, 2, 3, 4$ with arc set $A = (2, 4)$

Vertices set $V = 1, 2, 3, 4$ with arc set $A = (1, 3), (1, 4), (2, 4)$

Figure 3.1.1a Two graphs with vertices set V and arcs set A.

A *directed graph* contains directed arcs which are vertex pairs with an initial vertex and a terminal vertex. A *path* in a directed graph consists of a sequence of directed arcs which connects a pair of vertices on the path. If the initial vertex and terminal vertex is the same for one path, such a path is called a *cycle* or *circuit.*

A *bipartite graph* is a graph G whereby the vertex set Vcan be partitioned into two non-empty sets V_1 and V_2 in a such way that every arc of G goes from a vertex in V_1 to a vertex in V_2. When the bipartite graph is complete, it can be written as $G_{|V_1|,|V_2|}$ where $|V_1|, |V_2|$ is the number of vertices in V_1 and V_2 respectively. Similarly, a *tripartite graph* is a graph where the vertex set is partitioned into three non empty sets of vertices V_1, V_2 and V_3.

Recall the interpretation of a multi-layer perceptron network which is shown in Figure 2.1.1f of Chapter 2. Each vertex represents a neuron, each arc represents a connection and the cost of an arc is the weight of the connection. Ignoring the biases, a single hidden layer ANN can be effectively represented as a tripartite graph where $\hat{\imath}$ is the number of vertices on the input layer or the number of inputs to the ANN; $\hat{\jmath}$ is the number of vertices on the hidden layer or the number of hidden neurons; $\hat{k}$ is the number of vertices on the output layer or the number of outputs. For feed-forward ANNs, the acyclic graph is directed from inputs to outputs so that all vertices on the input layer are initial vertices while all vertices on theq output layer are terminal vertices.

Definitions 1.1 A *maximal path* in a graph G from vertex V_i to vertex V_j is a path of maximum cardinality between V_i and V_j. If G is k-partite, the largest possible maximal path is of $k-1$. A k-partite graph where every vertex is contained in a maximal path of cardinality $k-1$ is an ANN. If the removal of an arc from an ANN does not result in another ANN then the ANN is called a *basic ANN*

A basic ANN may or may not be a tree rooted at the output vertex. If a tree exists that is a basic ANN, then the tree is called a *basic-ANN-tree*, and is a graph of minimum degrees of freedom as measured by the number of arcs.

Note, consider a tripartite graph G with an "input" set of vertices V_1, an "intermediate" (hidden) set of vertices V_2 and an "output" set of vertices V_3, which in our case will be a single vertex, i.e. $|V_3|$=1. If $|V_1| \geq |V_2|$ then a basic ANN tree always exists. If $|V_1| < |V_2|$ then only a basic ANN exists and a basic-ANN-tree does not exist.

On the left of Figure 3.1.1c illustrates a tripartite graph representation of a complete ANN with $|V_1| = 4$, $|V_2| = 3$ and $|V_3| = 1$; On the right of Figure 3.1.1c illustrate a basic-ANN-tree with the same sets of vertices.

3.1.2 ANN topological optimisation problem

By transforming the topology of an ANN and its network connectivities into a graph, the topological optimisation problem can be explained using graphical approaches. Starting from a

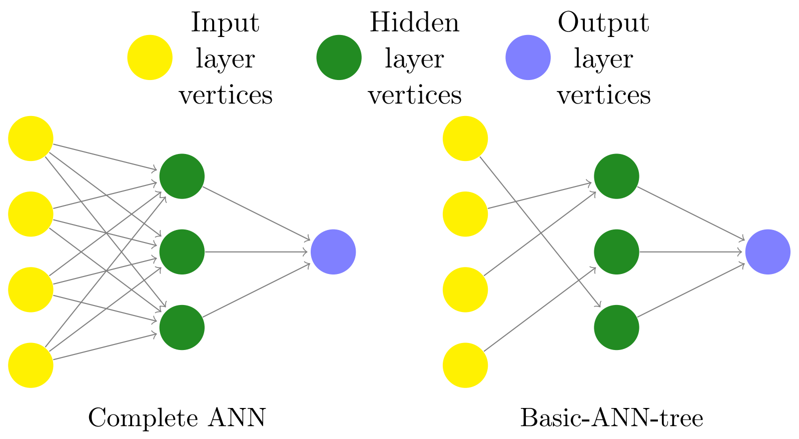


Figure 3.1.1c Illustration of a complete ANN and a basic ANN tree with the same set of vertices.

fully trained ANN which is a complete tripartite graph, two ideas were considered in searching for the optimal graph:

Static vertex optimisation

Consider a tripartite graph where $\hat{i} = |V_1|$, $\hat{j} = |V_2|$ and $\hat{k} = |V_3|$, we call the complete tripartite graph $G_{\hat{i},\hat{j},\hat{k}}$. Let E_G ($\bar{E}_G$) be the in-sample (out-of-sample) RMSE of the ANN given by G.

Definitions 1.2 From the vertex set V_1, V_2, V_3 where $v_i \in V_1$, $v_j \in V_2$, $v_k \in V_3$, find a set of arcs A to construct an ANN $G(\hat{a})$ where $|A| = \hat{a}$ is given, so that $G(\hat{a})$ produces the "best" generalisation performance in terms of the out of sample RMSE $\bar{E}_{G(\hat{a})}$ compared with the equivalent performance $\bar{E}_{G_{\hat{i},\hat{j},\hat{k}}}$. By best here we mean set A which maximises $\bar{E}_{G(\hat{a})} - \bar{E}_{G_{\hat{i},\hat{j},\hat{k}}}$

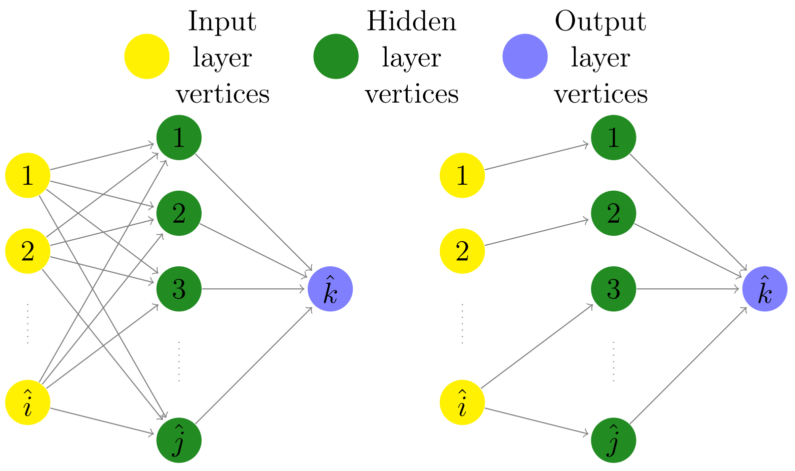
This definition of the static vertex optimisation applies to optimising the complete tripartite graph, but this can be easily generalised to apply to m-partite graphs or any non-complete graphs.

Ignoring the vertex biases (note that these were converted by the addition of extra vertices and arcs), the upper bound and the lower bound on $\hat{a}$ are as follows:

The total number of arcs in $G_{\hat{i},\hat{j},\hat{k}}$ is $\hat{i} \times \hat{j} + \hat{j} \times \hat{k}$ and the total number of arcs in a corresponding basic ANN is $\max\left(\hat{i}, \hat{j}\right) + \hat{j} \times \hat{k}$. As the result $G(\hat{a})$ shows in Figure 3.1.2a and

Figure 3.1.2b, the range of $\hat{a}$ is limited by:

$$\left(\max\left(\hat{\imath}, \hat{\jmath}\right) + \hat{\jmath} \times \hat{k}\right) \leq \hat{a} < \left(\hat{\imath} \times \hat{\jmath} + \hat{\jmath} \times \hat{k}\right)$$



Example of a static vertex optimised graph with the maximum number of arcs $\hat{a}$

Example of a static vertex optimised stucture with the minimal number of arcs $\hat{a}$

Figure 3.1.2a Left, The total number of arcs in a complete ANN $G_{\hat{\imath},\hat{\jmath},\hat{k}}$ is $\hat{\imath} \times \hat{\jmath} + \hat{\jmath} \times \hat{k}$; Right, The number of arcs in the corresponding basic ANN is $\max\left(\hat{\imath}, \hat{\jmath}\right) + \hat{\jmath} \times \hat{k}$

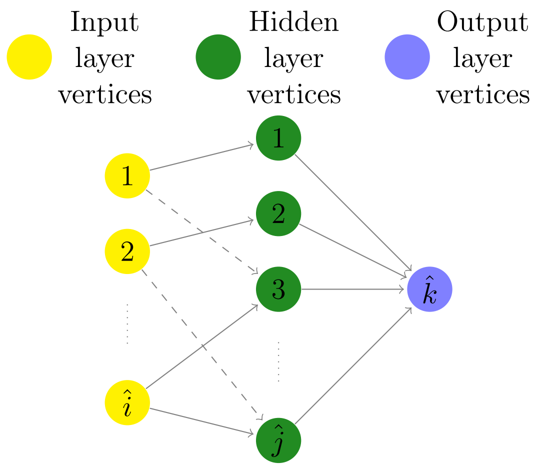


Figure 3.1.2b Based on the constraint, the number of arcs $\hat{a}$ must be chosen within a specific range.

Note that if $\hat{\imath} \geq \hat{\jmath}$ and $\hat{k} = 1$ a basic-ANN-tree is possible and the lower bound on $\hat{a}$ is $\hat{\imath} + \hat{\jmath}$ (which is one less than the number of vertices, as of course it should be)

Dynamic vertex optimisation

The definition of dynamic vertex optimisation is more general then that of static vertex optimisation. In dynamic vertex optimisation the number of vertices $|V_2|$ is not given. Therefore, with a given number of arcs $\hat{a}$ many different basic ANNs may be possible and the problem then becomes one of also optimising all such graphs. Note that if for a given number $|V_2|$, a basic-ANN-tree exists and $\hat{a}$ defines such a tree (i.e. takes its minimal value), then the problem is somewhat simplified in that the additional optimisation of the arc sets A are over all such trees.

Based on an initial complete ANN $G_{\hat{i},\hat{j},\hat{k}}$, Figure 3.1.2c shows the difference between a possible solution from static vertex optimisation and a possible solution from dynamic vertex optimisation.

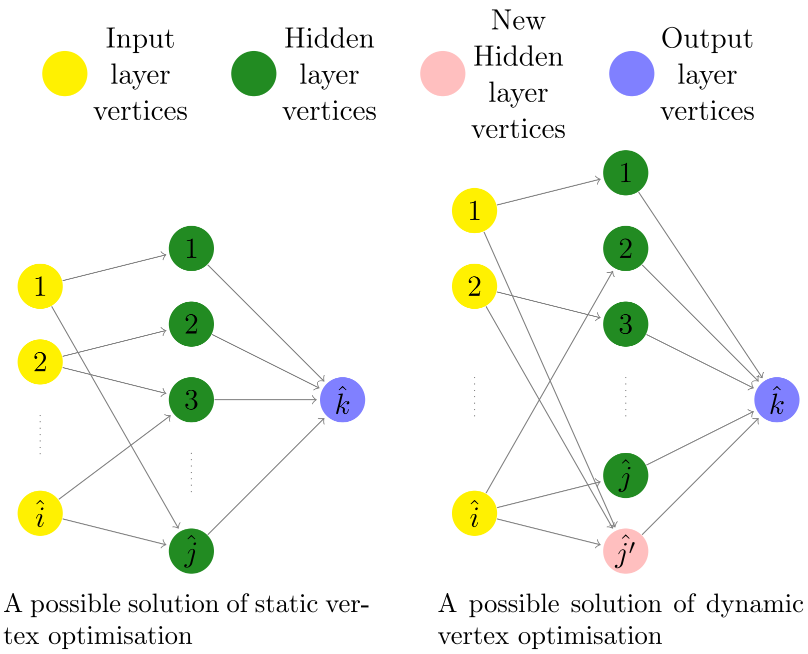


Figure 3.1.2c Applying static vertex optimisation and dynamic vertex optimisation on the same network.

Consider applying the static vertex optimisation approach on a tripartitie graph with $\hat{i}$ input vertices, $\hat{j}$ hidden vertices and $\hat{k}$ output vertices. In order to find the suitable set of connections A using this approach, the search space is potentially large because the number of possible connectivities with a specified number of arcs $\hat{a}$ is

$$\binom{\hat{\imath} \times \hat{\jmath} + \hat{\jmath} \times \hat{k}}{\hat{a}}$$

although, of course, not all choices of the $\hat{a}$ arcs are feasible.

For example, let a 19-4-1 complete ANN be the subject to be optimised using the static vertex optimisation approach, so that the tripartite graph $G_{\hat{\imath},\hat{\jmath},\hat{k}}$ has $\hat{\imath} = 19$, $\hat{\jmath} = 4$, $\hat{k} = 1$. In this case

$$\max\left(\hat{\imath}, \hat{\jmath}\right) + \hat{\jmath} \times \hat{k} = 23 \leq \hat{a} \leq \hat{\imath} \times \hat{\jmath} + \hat{\jmath} \times \hat{k} = 80$$

Specify $\hat{a} = 50$, the searching space for such ANN is

$$\binom{80}{\hat{a}} = 8.8714125e + 21$$

Although one can use greedy algorithms to locate the optimial solution by searching though all possible combinations of A, a better way is to use an algorithm inspired from heuristic search, which is discussed in the next section.

3.2 Heuristic implementation of static vertex optimisation

The designation of a 'heuristic' algorithm is an algorithm that produces a feasible solution to a problem but one which is not necessarily optimal.

One classical application of a heuristic search is in connection with the travelling salesman problem (TSP). A weighted complete undirected graph consists of vertices (cities) and weighted arcs (arcs with costs); The objective of a TSP is to find a *Hamiltonian cycle* with a minimum sum of weights. A Hamiltonian cycle is a cycle which passes through all the vertices once and once only. Because there is no known polynomial algorithm for solving the TSP exactly, a heuristic method is desired for obtaining a reasonably good (but not necessarily optimal) solution[104] to large-scale problems. One approach is to find any Hamiltonian cycle C first, then search for a modification on C to produce another Hamiltonian cycle C' with less weight; the process can be repeated until no improvement is found. For example, a simple heuristic procedure can be as follows:

On a complete undirected graph with n vertices, there is a total number of $(n-1)!$ Hamiltonian cycles. For indices i, j and k $i < j < k$ and $j > i+1$, $k > j+1$, let

$$C = v_1 v_2 \dots v_i v_{i+1} \dots v_j v_{j+1} \dots v_k v_1$$

be a Hamilton cycle which may be a suboptimal solution to the TSP.

As illustrated in Figure 3.2.0, a new Hamilton cycle can be obtained by removing the arcs (v_i, v_{i+1}) and (v_j, v_{j+1}) and adding the arcs (v_i, v_j) and (v_{i+1}, v_{j+1}), so that

$$C' = v_1 v_2 \dots v_i v_j v_{j-1} \dots v_{i+1} v_{j+1} v_{j+2} \dots v_k v_1$$

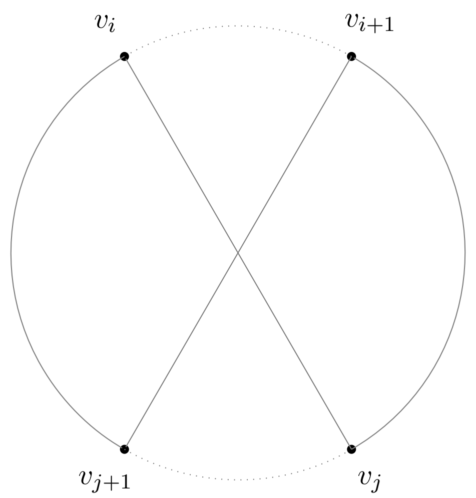


Figure 3.2.0 The modified Hamilton cycle C'

Comparing C' to C, if the summed weight of the newly added arcs is less than that of the removed arcs, i.e.

$$w(v_i, v_j) + w(v_{i+1}, v_{j+1}) < w(v_i, v_{i+1}) + w(v_j, v_{j+1})$$

the cycle C' is an improvement on C.

In the above example, only two arcs are replaced in a modification but other numbers are also possible. Define λ as the number of arcs to be swapped. If a solution is reached such that the removal of any set of λ arcs and its replacement with another set of λ arcs does not lead to an improved cycle, then the solution is called λ-*optimal* [105]. This heuristic approach is called a λ-*optimal* algorithm (or simply λ-*opt*). A λ-optimal algorithm can be applied to most combinatorial optimisation problems, not just the TSP.

Repetitive application of a λ-optimal algorithm, starting from different random initial cycles result in a high probability of finding the global optimal or near optimal solution among many local optimal solutions. The probability of obtaining an optimal solution using a 3-opt ($\lambda = 3$) algorithm is significantly higher than that of 2-opt ($\lambda = 2$) algorithms. Experimental results also indicated that the implementation of 4-opt ($\lambda = 4$) algorithms cost more computational time while not noticeably increasing the probability that it is optimum. It was concluded that the 3-opt is a good application for solving the TSP [105].

3.2.1 λ-optimal on ANN static vertex optimisation

Recall the objective of an ANN topology optimisation is to improve the ANN generalisation performance on out of sample data by removing some arcs between vertices on the input layer to vertices in the adjacent layers (and within hidden neural layers for multi-layer cases) while retaining the ANN property.

A new procedure called *ANN-reduction*, inspired by the λ-optimal algorithm in TSP is developed and implemented to automatically improve the topological graph of an ANN. Similarly, in attempting to find a graph with the optimal performance subject to an ANN network, the ANN-reduction algorithm involves disconnection and reconnection of a number of arcs to form different new network graphs. Introduced in the definition of the static vertex optimisation, $\hat{a}$ is defined as the total number of arcs required in the network. The idea behind ANN-reduction is that the heuristic optimisation algorithm effectively maps an initial starting solution into some local optimal solutions and attempts to search for the one near global optimal among all of those solutions.

In order to have a clear interpretation of ANN-reduction implementation, the parameters vector of an ANN graph can be recorded as a vector of real number representations for the arcs weights and biases inputs. An ANN with $\hat{\imath}$ input vertices, $\hat{\jmath}$ hidden layer vertices and $\hat{k}$ output vertices has a parameters vector W of size $(\hat{\imath}+1)\times\hat{\jmath}+(\hat{\jmath}+1)\times\hat{k}$ (which also includes $\hat{\jmath}+\hat{k}$ numbers of biases). Suppose b_{j_1} is the value of bias on the first hidden layer vertex, b_{k_1} is the value of bias on the first output vertex and so on. The parameter vector is stored as

$$
\begin{aligned}
W = \Big[& b_{j_1}, w(v_{i_1}, v_{j_1}), w(v_{i_2}, v_{j_1}), && ..., w(v_{\hat{\imath}}, v_{j_1}), \\
& b_{j_2}, w(v_{i_1}, v_{j_2}), w(v_{i_2}, v_{j_2}), && ..., w(v_{\hat{\imath}}, v_{j_2}), \\
& ..., && ..., w(v_{\hat{\imath}}, v_{\hat{\jmath}}), \\
& b_{k_1}, w(v_{j_1}, v_{k_1}), w(v_{j_2}, v_{k_1}), && ..., w(v_{(\hat{\jmath}}, v_{k_1}), \\
& ..., && ..., w(v_{\hat{\jmath}}, v_{\hat{k}}) \Big]
\end{aligned}
$$

For example, a 19-3-1 ANN can be represented by a parameter vector W with size of 64, where $W[0]$ is the value of bias on the first vertex, $W[1]...W[19]$ are the value of weights from the inputs to the first hidden vertex, as illustrated in Figure 3.2.1a, $W[0] = b_{j_1}$,$W[1] = w(v_{i_1}, v_{j_1})$, $W[21] = w(v_{i_1}, v_{j_2})$ etc. respectively.

The following is an implementation of ANN-reduction on ANN static vertex optimisation which is starting from a complete ANN $G_{\hat{\imath},\hat{\jmath},\hat{k}}$, with its set of vertices as defined in the definition A.1.

Step 1: Construct the fully trained ANN, its topology being represented by a tripartite graph $G_{\hat{\imath},\hat{\jmath},\hat{k}}$. The number of vertices $\hat{\imath}, \hat{\jmath}, \hat{k}$ on the input, hidden and output layer respectively

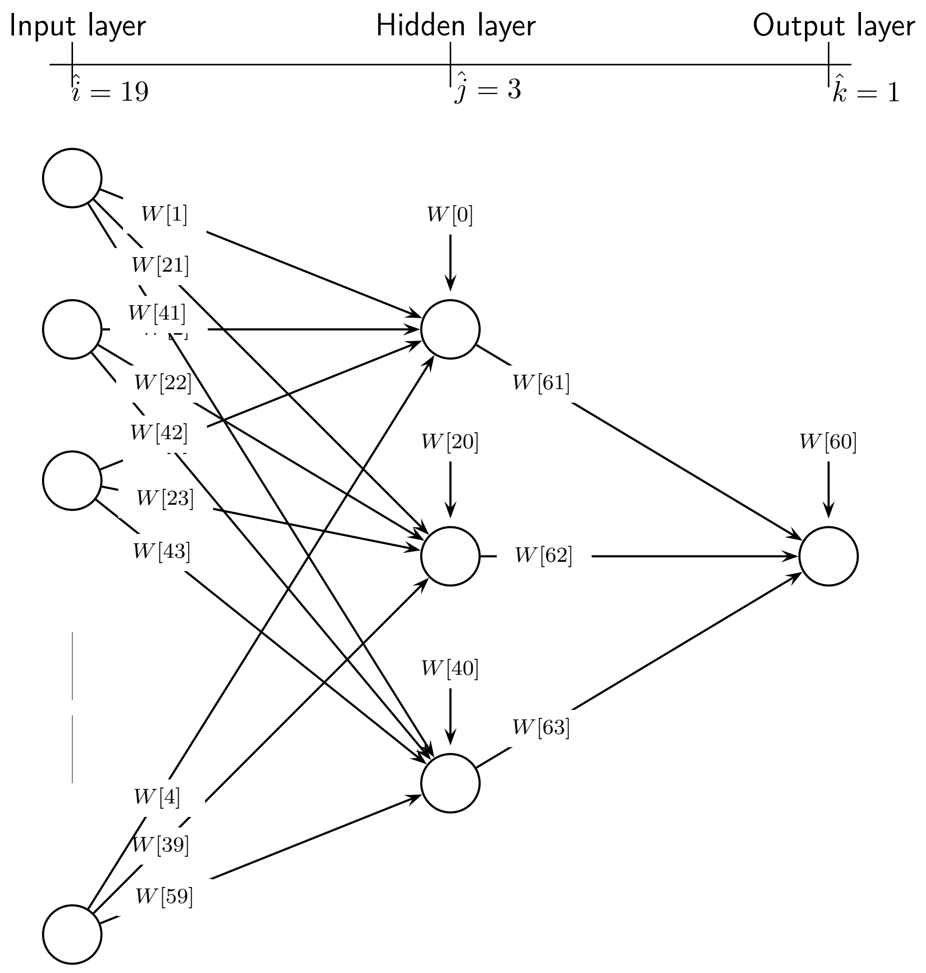


Figure 3.2.1a Vector representation of ANN's weights and biases parameters

and consists of three disjoint vertices sets V_1, V_2, V_3 where $v_i \in V_1$,$v_j \in V_2$ and $v_k \in V_3$. For the trained complete ANN, the number of connections is $\hat{i} * \hat{j} + \hat{j} * \hat{k}$ and the weight parameter vector is $W_{G_{\hat{i},\hat{j},\hat{k}}}$. A testing data sample is applied to compute the out-of-sample RMSE as $\bar{E}_{G_{\hat{i},\hat{j},\hat{k}}}$.

Step 2: From the disjoint vertices sets V_1, V_2, V_3, construct a number $\hat{m}$ of basic ANNs, represented by H^m for $m \in M = \{1, 2....\hat{m}\}$. Discussed in Section 3.1.2, the number of arcs on each basic ANN is $\max\left(\hat{i}, \hat{j}\right) + \hat{j} \times \hat{k}$. The weights parameter vector W_{H^m}. The weight of the arcs of H^m are taken from the complete ANN:

$$w_{H^m}(v_i, v_j) = w_{G_{\hat{i},\hat{j},\hat{k}}}(v'_i, v'_j)$$

where arc (v_i, v_j) of graph H^m corresponds to arc (v'_i, v'_j) of graph $G_{\hat{i},\hat{j},\hat{k}}$

Step 3: Feed the training samples to train all $\hat{m}$ of constructed basic ANNs H^m to $H^{'m}$ ($H^{'m}$ is topological identical to H^m but with different arc weights). Compute the out-of-sample RMSE for each of the basic ANN $\bar{E}_{H'^m}$.

Step 4: Select the graph H^* from the graphs $H^{'m}$, $m = 1...\hat{m}$ with the minimum value of $\bar{E}_{H'^m}$ as the basic ANN for the next stage.

Step 5: Based on the selected basic ANN H^* and corresponding arc-weight vector W_{H^*} , add $\hat{a} - \left\{\max\left(\hat{i},\hat{j}\right) + \hat{j}\times\hat{k}\right\}$ number of arcs to create $G(\hat{a})$. Recall from Section 3.1.2 that

$$\max\left(\hat{i},\hat{j}\right) + \hat{j}\times\hat{k} \le \hat{a} < \hat{i}\times\hat{j} + \hat{j}\times\hat{k}$$

so with the specified total number of arcs $\hat{a}$, the above inequality must be satisfied.

Step 6: For a specified $\hat{a}$ repeat Step 5 $\hat{n}$ times. For $n \in N = \{0, 1, 2....\hat{n}\}$ produce a set of $G^n(\hat{a})$ which contains all possible combinations of the added arcs. Feed the in-sample training data to evaluate every $G^n(\hat{a})$ and record the one with the minimal in-sample RMSE as $G^*(\hat{a})$; Use the training sample to retrain the $W_{G^*(\hat{a})}$ to become $W_{G'^*(\hat{a})}$, and evaluate its out-of-sample RMSE: $\bar{E}_{G'^*(\hat{a})}$.

Step 7: Repeat Step 5 and 6 with some different $\hat{a}$. Choose the $\hat{a}$ and the n for which $G^{'*}(\hat{a})$ has the least out-of-sample RMSE. Denote this as G^{**} and $\bar{E}_{G^{**}}$ respectively.

Finally, compare the $\bar{E}_{G^{**}}$ to the performance of the complete ANN $\bar{E}_{G_{\hat{i},\hat{j},\hat{k}}}$. If $\bar{E}_{G^{**}} \le \bar{E}_{G_{\hat{i},\hat{j},\hat{k}}}$ then G^{**} is the solution of ANN-reduction and we call it a *'topologically-optimised ANN.*

As the weights of arcs are represented directly by those of non-zero real numbers, setting the value of a weight to be zero is equivalent to removing the arc from the graph G. Note that

1) The procedure effectively reduces the number of connections $\hat{i} * \hat{j} + \hat{j} * \hat{k}$ in the $G_{\hat{i},\hat{j},\hat{k}}$ to $\hat{a}$ in the $G_{\hat{a}}$ with a chosen value of added arcs: $\hat{a} - \left\{\max\left(\hat{i},\hat{j}\right) + \hat{j}\times\hat{k}\right\}$;

2) The size of parameter vector is fixed: $\dim(W_{G_{\hat{a}}}) = \dim(W_{G_{\hat{\imath},\hat{\jmath},\hat{k}}}) = (\hat{\imath}+1)\times\hat{\jmath}+(\hat{\jmath}+1)\times\hat{k}$, so $W_{G_{\hat{\imath},\hat{\jmath},\hat{k}}}$ is fitted with all non-zero values while $W_{G_{\hat{a}}}$ must contain $(\hat{\imath}+1)\times\hat{\jmath}+(\hat{\jmath}+1)\times\hat{k}-\hat{a}$ number of zeros.

The flow chart representation of the pseudo code for basic ANN construction (Step 1 to Step 4) of the ANN-reduction algorithm is displayed in Figure 3.2.1b; Arcs addition and Simplified ANN construction (Step 5 to Step 7) of the ANN-reduction algorithm is displayed in Figure 2.1c

3.2.2 Enhanced ANN-Reduction Procedure (ERP)

Two major issues of the ANN-reduction procedure appear in the implementation described previously. First, the process requires us to find an optimal performing basic ANN as a basis for the following arc re-connection; accordingly, all m possible graphs of the basic ANN are to be constructed, trained, tested and compared (see Figure 3.2.1b). Second, provide a basic ANN and a specified number of arcs to be added, the process required for building the simplified ANN topology by adding the arc connections. All n possible combinations from the arcs addition are assessed and the new graphs are then tested for further process. For both of the issues, the problem is the huge search space for some even small sizes of ANN.

The question of calculating the exact number of possible basic ANNs from a specified size of input and hidden vertices is a problem of calculating a Stirling partition number. The problem counts the number of ways to partition a set of labeled items into a number of unlabeled nonempty subsets. In the words of single hidden layer ANN, given vertices $\hat{\imath},\hat{\jmath},\hat{k}$, the 'labeled items' are the vertices, which is the larger of input vertices size and hidden vertices size $a = \max\left(\hat{\imath},\hat{\jmath}\right)$; The 'unlabeled subsets' are the vertices to be varied by the possible basic ANNs $b = \min\left(\hat{\imath},\hat{\jmath}\right)$. The total number of possible basic ANN graphs is Stirling $S\binom{a}{b}$ can be calculated by Sharp's algorithm[106]. Consider the first issue on finding a basic ANN for a 5-3-1 ANN, the number of possible basic ANN is $m = 25$ for $G_{5,3,1}$. These basic ANN are shown in Appendix A. However, for a 10-5-1 ANN $G_{10,5,1}$, such number increases to $m = 34105$.

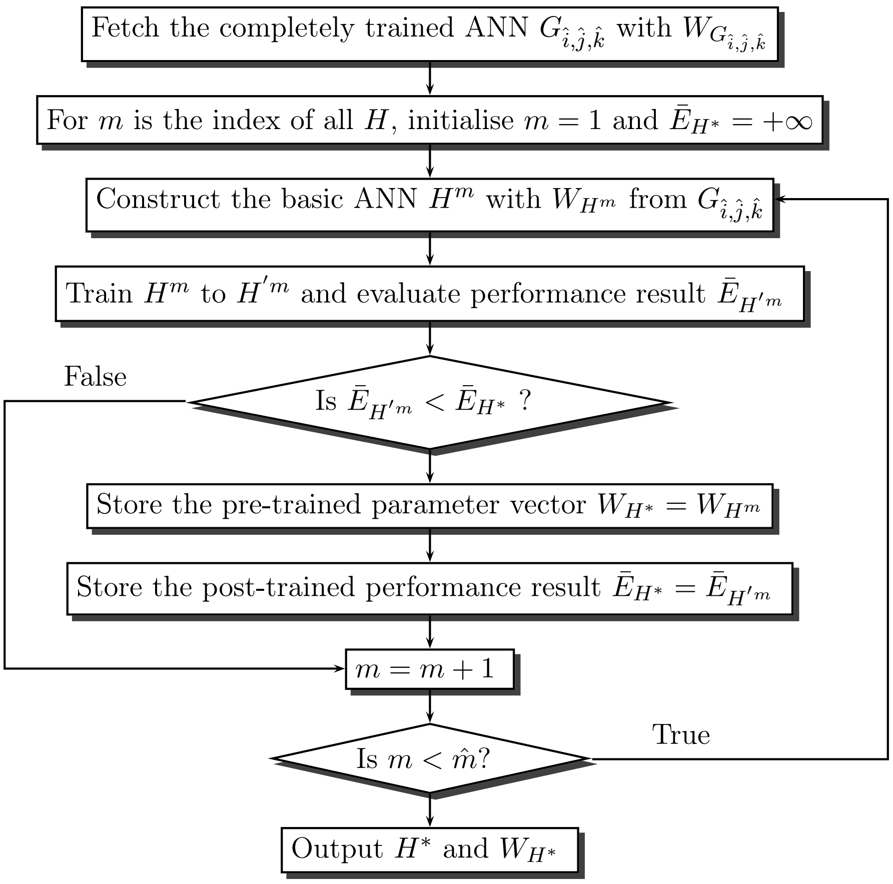
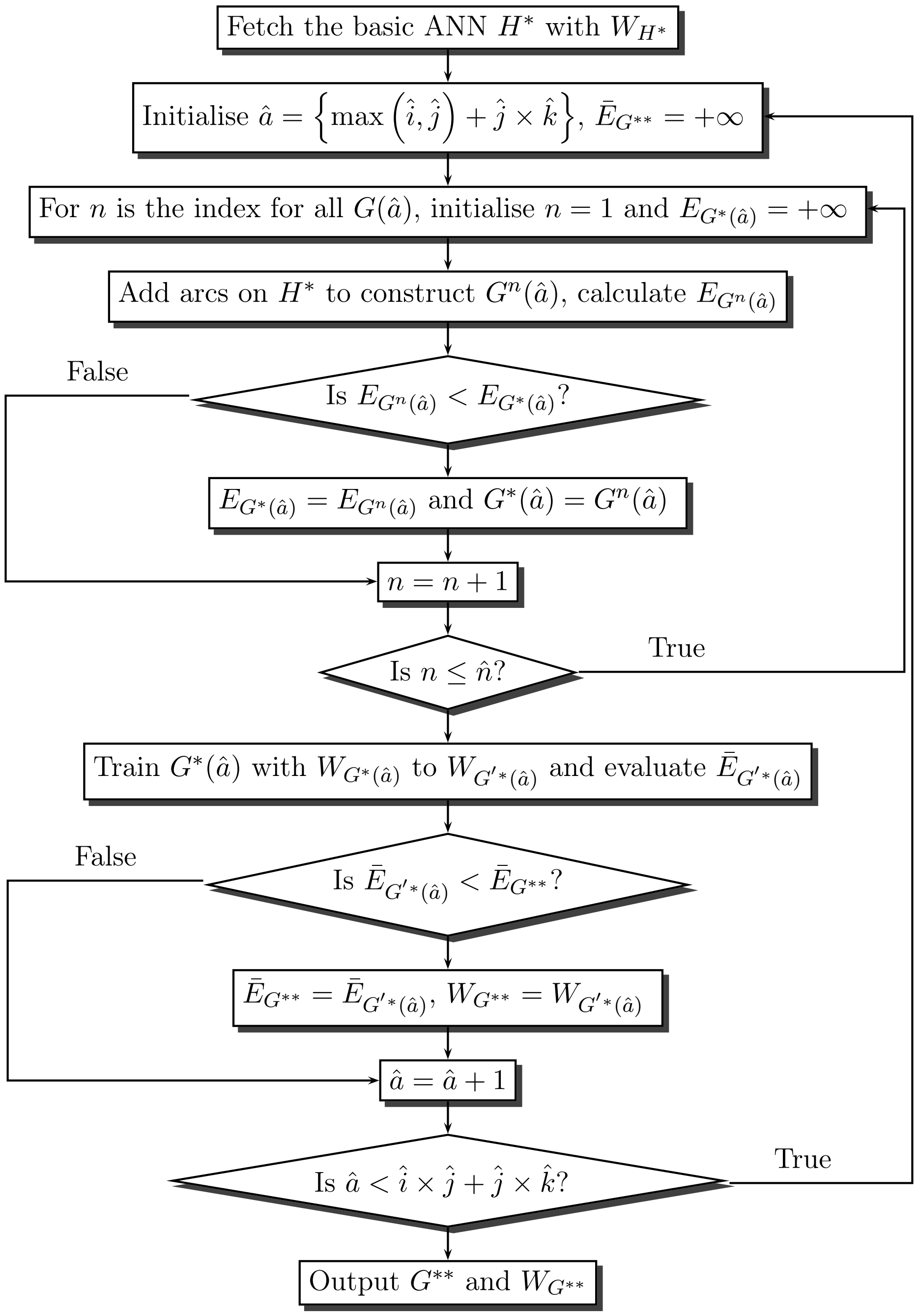

Fetch the completely trained ANN $G_{\hat{\imath},\hat{\jmath},\hat{k}}$ with $W_{G_{\hat{\imath},\hat{\jmath},\hat{k}}}$
For m is the index of all H, initialise $m = 1$ and $\bar{E}_{H^*} = +\infty$
Construct the basic ANN H^m with W_{H^m} from $G_{\hat{\imath},\hat{\jmath},\hat{k}}$
Train H^m to H'^m and evaluate performance result $\bar{E}_{H'^m}$
False
Is $\bar{E}_{H'^m} < \bar{E}_{H^*}$?
Store the pre-trained parameter vector $W_{H^*} = W_{H^m}$
Store the post-trained performance result $\bar{E}_{H^*} = \bar{E}_{H'^m}$
$m = m + 1$
True
Is $m < \hat{m}$?
Output H^* and W_{H^*}

Figure 3.2.1b Flowchart of ANN-reduction algorithm Step 1-4



Fetch the basic ANN H^* with W_{H^*}
Initialise $\hat{a} = \left\{\max\left(\hat{i}, \hat{j}\right) + \hat{j} \times \hat{k}\right\}$, $\bar{E}_{G^{**}} = +\infty$
For n is the index for all $G(\hat{a})$, initialise $n = 1$ and $E_{G^*(\hat{a})} = +\infty$
Add arcs on H^* to construct $G^n(\hat{a})$, calculate $E_{G^n(\hat{a})}$
Is $E_{G^n(\hat{a})} < E_{G^*(\hat{a})}$?
False
$E_{G^*(\hat{a})} = E_{G^n(\hat{a})}$ and $G^*(\hat{a}) = G^n(\hat{a})$
$n = n + 1$
Is $n \leq \hat{n}$?
True
Train $G^*(\hat{a})$ with $W_{G^*(\hat{a})}$ to $W_{G'^*(\hat{a})}$ and evaluate $\bar{E}_{G'^*(\hat{a})}$
Is $\bar{E}_{G'^*(\hat{a})} < \bar{E}_{G^{**}}$?
False
$\bar{E}_{G^{**}} = \bar{E}_{G'^*(\hat{a})}$, $W_{G^{**}} = W_{G'^*(\hat{a})}$
$\hat{a} = \hat{a} + 1$
Is $\hat{a} < \hat{i} \times \hat{j} + \hat{j} \times \hat{k}$?
True
Output G^{**} and $W_{G^{**}}$

Figure 3.2.1c Flowchart of ANN-reduction algorithm Step 5-7

Indicated by the large difference in the above examples, it could be very expensive to iterate through all candidate basic graphs for training and evaluation. To overcome this first issue of ANN-reduction, a substitution is developed in ERP to efficiently build a basic ANN: Inspired from sensitivity pruning methods, the modification enhances the ANN-reduction by strictly searching for a suitable basic ANN from a completely trained ANN network by disconnecting a path with effectively least aggregate weights out of the graph in each iteration. This is different to the formal algorithm where searching for the basic ANN is performed randomly among combinations of a given minimal number of edges.

This modification is based on the following assumption, which is related to the amount of an approximation error from a well-trained ANN. Suppose each input of the ANN v_i pass through a directed path of connections with parameters w_{v_i,v_j} and w_{v_j,v_k} to output v_k contributes some useful information while carrying an amount of training error (noise). Disconnecting such a connection effectively removes the contribution as well as the noise transmitted through that path. If such a contribution is less than the noise, the overall performance of the ANN should be improved after the parameter path is disconnected.

If the absolute value of the weight parameter for a connection between input v_i to hidden layer neuron v_j $\left|w_{v_i,v_j}\right|$ is relatively small compared to the absolute value from all other inputs $\bar{v_i}$ to the hidden vertices v_j, this suggests that the input v_i contributes less information compared to the rest of inputs on the paths through the hidden neuron v_j. Similarly, a small value of weight $|w_{v_i,v_k}|$ has a similar implication whereby the signal from neuron v_j has less of a contribution to the output v_k. To compare the weight between the paths, the relative value of a path from an input vertex v_i through a hidden vertex v_j to the output vertex v_k can be calculated by multiplying the weight parameters of the two connections $p_{i,j,k} = w_{v_i,v_j} \times w_{v_j,v_k}$. For a complete ANN $G_{\hat{i},\hat{j},\hat{k}}$, the total number of paths is $\hat{i} \times \hat{j} \times \hat{k}$.

For example, a complete ANN $G_{3,2,1}$ with the parameter vector in Table 3.2.2 has its graph displayed in Figure 3.2.2a. The dashed path has a relative value of $0.1 \times 0.1 = 0.01$; the dotted path has a relative value of $0.4 \times 0.2 = 0.08$

The implication of relative value for paths provide a foundation for the following modifica-

Input to hidden layer	Bias	Input 1	Input 2	Input 3
Neuron 1	1	0.7	0.3	0.4
Neuron 2	1	0.1	0.5	0.2

Hidden to output layer	Bias	Neuron 1	Neuron 2
Output Neuron	1	0.2	0.1

Table 3.2.2 Parameter vector of $G_{3,2,1}$

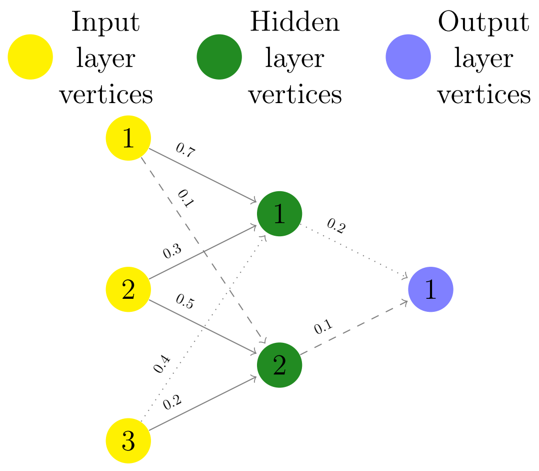


Figure 3.2.2a Graph of $G_{3,2,1}$, the dotted path has a relative value of 0.08 and the dashed path has a relative value of 0.01.

tion on the ANN-reduction process of constructing a basic ANN.

Start from a list of paths $P_{G_{\hat{i},\hat{j},\hat{k}}}$ which contains all paths in $G_{\hat{i},\hat{j},\hat{k}}$ with values calculated from the parameters vector $W_{G_{\hat{i},\hat{j},\hat{k}}}$. The basic ANN can be constructed from the following steps:

Step 1: Sort the paths $p_{i,j,k}$ in $P_{G_{\hat{i},\hat{j},\hat{k}}}$ in ascending order according to the absolute value. Initialise an empty list $P_{removed}$ and initialise iteration counter $iteration = 1$;

Step 2: Check the path $p_{i',j',k'}$ of list $P_{\hat{i},\hat{j},\hat{k}}$ proceed to next step if $d^{out}_{i'} > 1$ and $d^{in}_{j'} > 1$ is true, which means the path is not the only path connecting the input vertices $v_{i'}$, or the only path connecting the hidden layer vertices $v_{j'}$; otherwise if $d^{out}_{i'} = 1$ or $d^{in}_{j'} = 1$ then go to step 4;

Step 3: Add $p_{i',j',k'}$ to $P_{removed}$, disconnect the path by setting the corresponding arc's weight

$w_{v_{i',j'}} = 0$ in the parameter vector W ;

Step 4: Remove $p_{i',j',k'}$ from $P_{G_{\hat{i},\hat{j},\hat{k}}}$;

Step 5: Stop the process if the list is empty. Otherwise increment the iteration counter $iteration = iteration + 1$ and goto Step 2,

When the list of paths $P_{G_{\hat{i},\hat{j},\hat{k}}}$ in Step 5 is empty, a basic ANN is constructed and its parameter vector is stored in $W_{G_{\hat{i},\hat{j},\hat{k}}}$. Note that the basic ANN produced by the ERP may not have the best out-of-sample RMSE compared to all possible basic ANNs assessed in the ANN-reduction. However because the pruning process is based on the discussed assumption that the connections with small weights contribute less useful information and introduce noise, by disconnecting the light weighted paths and retaining only the most heavy weighted connections, a basic ANN is constructed for the rest of the optimisation process.

As an example, Figure 3.2.2b is the basic ANN constructed from the previous example of $G_{3,2,1}$ with parameter vector listed in Table 3.2.2.

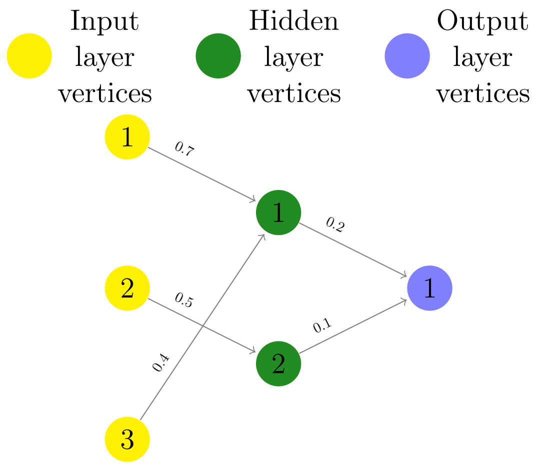


Figure 3.2.2b Graph of basic ANN constructed from $G_{3,2,1}$

On the basic ANN created from $G_{\hat{i},\hat{j},\hat{k}}$, the number of arcs can be added is fixed by the total number of connections $\hat{a}$ specified in the simplified graph $G(\hat{a})$, which is $\hat{a} - \left\{\max\left(\hat{i},\hat{j}\right) + \hat{j} \times \hat{k}\right\}$. The number of arcs removed in the basic ANN is $\hat{i} \times \hat{j} - \max\left(\hat{i},\hat{j}\right)$, so the number $\hat{n}$ of possible sets of arcs to be added is therefore $\hat{n} = \begin{pmatrix} \hat{\imath} \times \hat{\jmath} - \max\left(\hat{\imath}, \hat{\jmath}\right) \\ \hat{a} - \left\{\max\left(\hat{\imath}, \hat{\jmath}\right) + \hat{\jmath} \times \hat{k}\right\} \end{pmatrix}$. For example, suppose 3 arcs are to be added on a basic ANN with topology $(5, 3, 1)$, there are $\hat{n} = \binom{10}{3} = 120$; for 3 arcs are to be added on a basic ANN with topology $(10, 5, 1)$, $\hat{n} = \binom{40}{3} = 9880$.

Again the large difference in the example above indicates that the cost to construct all n graphs for a specified $\hat{a}$ could be very high; Furthermore, by default ANN-reduction requires to search through an entire range of $\hat{a}$: $\left[\max\left(\hat{\imath}, \hat{\jmath}\right) + \hat{\jmath} \times \hat{k} \leq \hat{a} < \hat{\imath} \times \hat{\jmath} + \hat{\jmath} \times \hat{k}\right]$ to obtain a global optimal solution and the range changes accordingly with the size of input and hidden vertices.

A complete implementation of ERP is represented in the following flow charts. Figure 3.2.2c is the flow chart representation of ERP for steps of creating the basic ANN; Figure 3.2.2d is the flow chart representation of ERP for the steps of arcs addition process and the subsequent searches for the optimal solution.

The use of random arc re-connection has a drawback. The performance of ERP depends on an appropriate number of arc-addition attempts (trials), say $n*$ which we specify. Using a 10-5-1 topology as an example, (ignoring the biases) the basic ANN has 15 arcs and the complete ANN has 55 arcs. If $\hat{a} = 17$, which means $17 - 15 = 2$ arcs are to be added on the basic ANN, the number of possible combinations for adding the 2 arcs are $\hat{n} = \begin{pmatrix} 40 \\ 2 \end{pmatrix} = 780$ where 40 is the number of possible vertex pairs where arcs may be added. For an $\hat{a}$ that is close to the size of the complete ANN or close to the size of the basic ANN, the total number n of the possible ways for adding the arcs is relatively small so most of the graphs $G_{\hat{a}}$ can be assessed in ERP by setting a sufficiently large trial number $n*$ for the random arc re-connection or even the original greedy approach can be used in this case. On the contrary, for $\hat{a}$ that is near the mid point of the range, the number of possible arc-set addition can be too large for ERP to be computationally feasible.

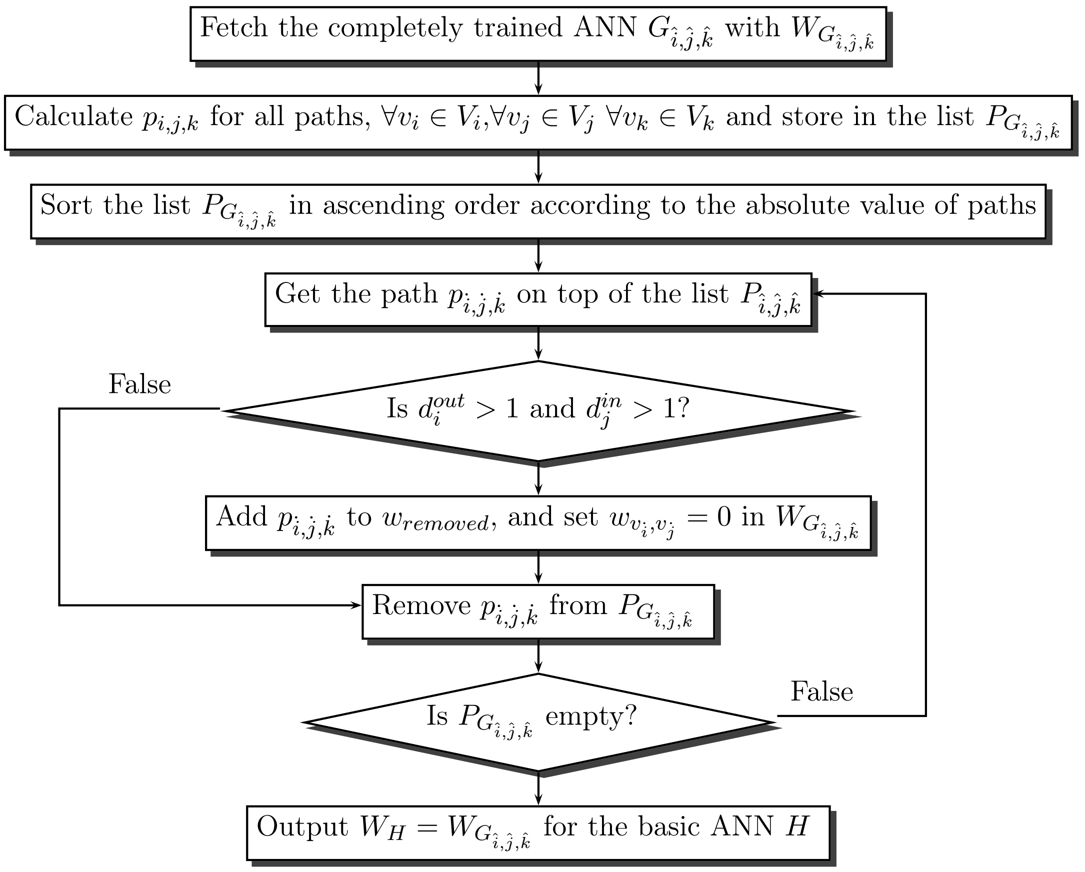
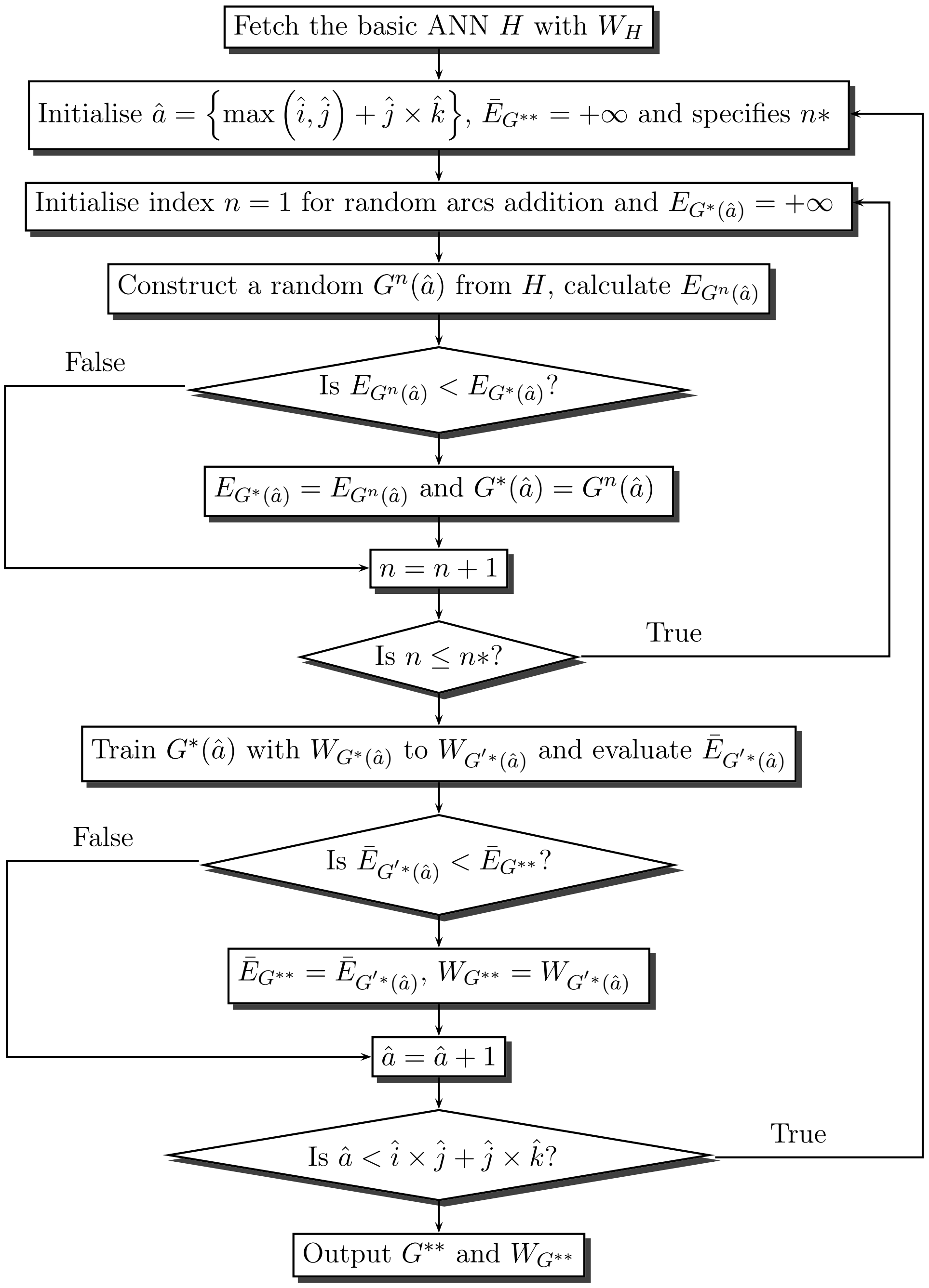

Fetch the completely trained ANN $G_{\hat{\imath},\hat{\jmath},\hat{k}}$ with $W_{G_{\hat{\imath},\hat{\jmath},\hat{k}}}$
Calculate $p_{i,j,k}$ for all paths, $\forall v_i \in V_i, \forall v_j \in V_j\ \forall v_k \in V_k$ and store in the list $P_{G_{\hat{\imath},\hat{\jmath},\hat{k}}}$
Sort the list $P_{G_{\hat{\imath},\hat{\jmath},\hat{k}}}$ in ascending order according to the absolute value of paths
Get the path $p_{\dot{i},\dot{j},\dot{k}}$ on top of the list $P_{\hat{\imath},\hat{\jmath},\hat{k}}$
False
Is $d_{\dot{i}}^{out} > 1$ and $d_{\dot{j}}^{in} > 1$?
Add $p_{\dot{i},\dot{j},\dot{k}}$ to $w_{removed}$, and set $w_{v_{\dot{i}},v_{\dot{j}}} = 0$ in $W_{G_{\hat{\imath},\hat{\jmath},\hat{k}}}$
Remove $p_{\dot{i},\dot{j},\dot{k}}$ from $P_{G_{\hat{\imath},\hat{\jmath},\hat{k}}}$
False
Is $P_{G_{\hat{\imath},\hat{\jmath},\hat{k}}}$ empty?
Output $W_H = W_{G_{\hat{\imath},\hat{\jmath},\hat{k}}}$ for the basic ANN H

Figure 3.2.2c Flow chart representation of ERP for steps of creating the basic ANN



Fetch the basic ANN H with W_H
Initialise $\hat{a} = \left\{\max\left(\hat{i}, \hat{j}\right) + \hat{j} \times \hat{k}\right\}$, $\bar{E}_{G^{**}} = +\infty$ and specifies $n*$
Initialise index $n = 1$ for random arcs addition and $E_{G^*(\hat{a})} = +\infty$
Construct a random $G^n(\hat{a})$ from H, calculate $E_{G^n(\hat{a})}$
Is $E_{G^n(\hat{a})} < E_{G^*(\hat{a})}$?
False
$E_{G^*(\hat{a})} = E_{G^n(\hat{a})}$ and $G^*(\hat{a}) = G^n(\hat{a})$
$n = n + 1$
Is $n \leq n*$?
True
Train $G^*(\hat{a})$ with $W_{G^*(\hat{a})}$ to $W_{G'^*(\hat{a})}$ and evaluate $\bar{E}_{G'^*(\hat{a})}$
Is $\bar{E}_{G'^*(\hat{a})} < \bar{E}_{G^{**}}$?
False
$\bar{E}_{G^{**}} = \bar{E}_{G'^*(\hat{a})}$, $W_{G^{**}} = W_{G'^*(\hat{a})}$
$\hat{a} = \hat{a} + 1$
Is $\hat{a} < \hat{i} \times \hat{j} + \hat{j} \times \hat{k}$?
True
Output G^{**} and $W_{G^{**}}$

Figure 3.2.2d Flow chart representation of ERP for the steps of arcs addition process.

3.2.3 Cascaded Enhanced ANN-Reduction Procedure (CERP)

An alternative implementation, namely CERP is designed by cascading a series of arc reconnection processes following the construction of the basic ANN in the ERP. Each arc reconnection process is only performed within a limited range of small $\hat{a}$ and in each process the arcs are added on the output graph produced from its predecessor; the cascaded process terminates when it fails to obtain a strucure with an improvment on the out-of-sample performance after retraining.

Consider an implementation of CERP for optimisng the complete ANN $G_{\hat{i},\hat{j},\hat{k}}$, where $\forall v_i \in V_i$,$\forall v_j \in V_j$ and $\forall v_k \in V_k$. A basic ANN H is constructed in the first place following the steps of ERP described in Section 3.2.2; the performance evaluation of the basic ANN is denoted by $\bar{E}_H$. Suppose all of the cascaded arc re-connection processes are configured to add a single arc only in turn:

Step 1: Initialise the starting reference ANN graph for the cascaded arc re-connection process, $G_{temp} = H$, $\bar{E}_{G_{temp}} = \bar{E}_H$ and $W_{G_{temp}} = W_H$;

Step 2: Perform arc re-connection by adding a single arc (v_i, v_j) to G_{temp} to form $G_{temp} \cup (v_i, v_j)$;

Step 3: Retrain $G_{temp} \cup (v_i, v_j)$ to $G'_{temp} \cup (v_i, v_j)$ and obtain the out-of-sample performance $\bar{E}_{G'_{temp} \cup (v_i, v_j)}$;

Step 4: Find the arc (v'_i, v'_j) for which the out-of-sample performance improvement of the newly constructed graph $G_{improved} = G'_{temp} \cup (v'_i, v'_j)$ is larger than all other possible arcs, so that

$$\Delta E_{G_{improved}} = \bar{E}_{G'_{temp} \cup (v'_i, v'_j)} - \bar{E}_{G_{temp}}$$

is maximised. If the arc (v'_i, v'_j) does exist and $E_{G_{improved}} > 0$, goto Step 5; If no such arc exists to satisfy $E_{G_{improved}} > 0$, the output graph is assigned to $G^{**} = G_{temp}$ and goto Step 6;

Step 5: Reassign reference ANN graph $G_{temp} = G_{improved}$, $\bar{E}_{G_{temp}} = \bar{E}_{G_{improved}}$ and $W_{G_{temp}} =$

$W_{G_{improved}}$ and goto Step 2.

Step 6: Compare the $\bar{E}_{G^{**}}$ to the performance of complete ANN $\bar{E}_{G_{\hat{i},\hat{j},\hat{k}}}$, if $\bar{E}_{G^{**}} \leq \bar{E}_{G_{\hat{i},\hat{j},\hat{k}}}$ then G^{**} is the topologically-optimised graph.

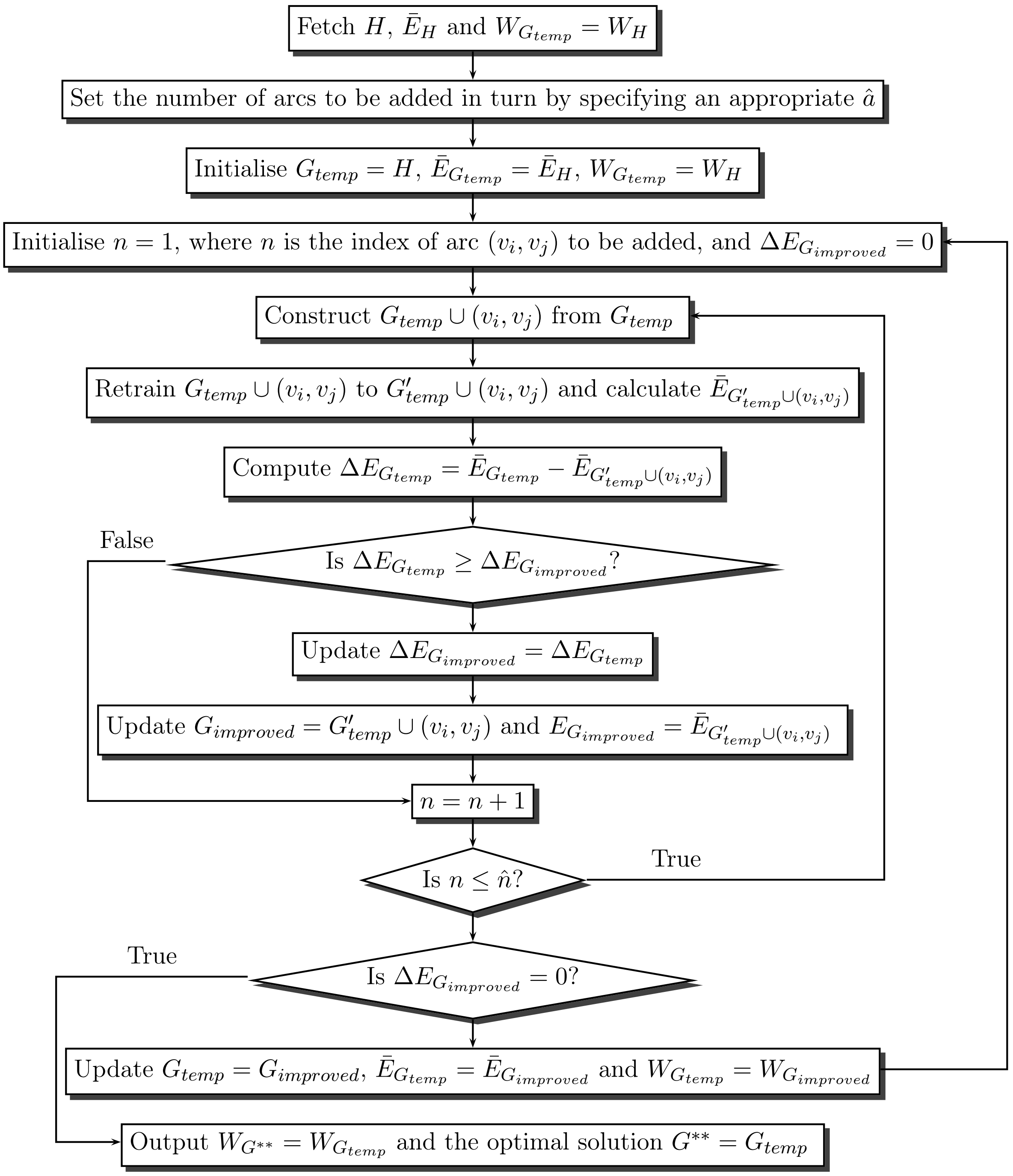
Because the CERP steps for creating the basic ANN from a complete graph are the same as ERP, the steps are not discussed explicitly here. The following Figure 3.2.3 is a flow chart representation of CERP steps for the cascaded arcs addition with a specific number of arcs in turn.

The improvement of CERP from ERP is in two folds: 1) Each cascaded arcs addition in CERP attempts for all combinations of the arcs for a given $\hat{a}$, whereby in ERP only a portion of combination maybe tested. 2) CERP outputs a result immediately at the point that the cascading process is terminated whenever the condition of performance improvement is not satisfied, whereas in ERP the selected graphs from all ranges of $\hat{a}$ are compared.

3.3 Conclusion of the chapter

In this chapter we developed a framework of techniques for searching the optimal architecture of an ANN and we introduced a graphical topology representation. The heuristic algorithm namely ANN-reduction is proposed for systematically optimising existing sub-optimal networks by changing the set of connections but without altering the number of neurons. The ANN-reduction algorithm is developed to repetitively generate modifications on an initial complete topology until the out of sample performance can no longer be improved. ANN-reduction can be applied to improve the problems associated with over-training and the effects of too many degrees of freedom inherent in the ANN application because of the excessive numbers of parameters to be estimated in the complete topology.

The aim of this research is to improve performance of ANNs particularly in applications to the FFP. For this purpose, two implementations are designed following the framework of ANN-reduction algorithm: 1) ERP and 2) CERP. Both implementation have a common phase of



Fetch H, $\bar{E}_H$ and $W_{G_{temp}} = W_H$
Set the number of arcs to be added in turn by specifying an appropriate $\hat{a}$
Initialise $G_{temp} = H$, $\bar{E}_{G_{temp}} = \bar{E}_H$, $W_{G_{temp}} = W_H$
Initialise $n = 1$, where n is the index of arc (v_i, v_j) to be added, and $\Delta E_{G_{improved}} = 0$
Construct $G_{temp} \cup (v_i, v_j)$ from G_{temp}
Retrain $G_{temp} \cup (v_i, v_j)$ to $G'_{temp} \cup (v_i, v_j)$ and calculate $\bar{E}_{G'_{temp} \cup (v_i, v_j)}$
Compute $\Delta E_{G_{temp}} = \bar{E}_{G_{temp}} - \bar{E}_{G'_{temp} \cup (v_i, v_j)}$
False
Is $\Delta E_{G_{temp}} \geq \Delta E_{G_{improved}}$?
Update $\Delta E_{G_{improved}} = \Delta E_{G_{temp}}$
Update $G_{improved} = G'_{temp} \cup (v_i, v_j)$ and $E_{G_{improved}} = \bar{E}_{G'_{temp} \cup (v_i, v_j)}$
$n = n + 1$
True
Is $n \leq \hat{n}$?
True
Is $\Delta E_{G_{improved}} = 0$?
Update $G_{temp} = G_{improved}$, $\bar{E}_{G_{temp}} = \bar{E}_{G_{improved}}$ and $W_{G_{temp}} = W_{G_{improved}}$
Output $W_{G^{**}} = W_{G_{temp}}$ and the optimal solution $G^{**} = G_{temp}$

Figure 3.2.3 Flow chart representation of CERP steps for the cascaded arcs addition with a specific number of arcs in turn.

constructing the basic ANN but have different procedures of executing the arc addition phase. Because of this difference, the two implementations may yield different optimised topologies when applied to the same problem. We will discuss with simulation examples in the next chapter.

Chapter 4

Empirical simulation on function approximation

4.1 Introduction

In Chapter 3 we have proposed a theoretical framework for static vertex optimisation of feed-forward ANNs, namely the ANN reduction process. The process is developed using a heuristic searching method which consist of two stages:

(i) Basic ANN construction (see Figure 3.2.1b) and

(ii) Arc re-connection (see Figure 3.2.1c).

In the first stage, the trained complete ANN is processed to construct a set of basic ANNs; After retraining, the best performing basic ANN is selected as a candidate solution with a minimal number of arcs, and its out-of-sample performance is recorded for the next stage.

In the second stage, a number of arcs are added to the basic ANN to construct some new ANNs, which are then retrained and compared for further selection of the solution. From all newly constructed ANNs , the output of the ANN reduction process is the one with the best improvement on out-of-sample performance compared to the initial complete ANN. Because all

ANNs produced during the process have the same number of neurons, this distinguishes ANN reduction process from the traditional applications listed in Chapter 2.

We have also drawn attention to some critical steps of the ANN reduction where a huge search space may be produced in practice, preventing the process from performing efficiently. In order to address the issue, two implementations of the ANN reduction process are developed. The procedures are called Enhanced ANN-Reduction Procedure (ERP) and Cascaded Enhanced ANN-Reduction Optimisation Procedure (CERP).

Compared to the ANN reduction process, both ERP and CERP introduce a systematic pruning implementation in the first stage to build the basic ANN instead of constructing a pool of random constructed basic ANN. Furthermore, the two procedures perform arc re-connection differently in the second stage. Because of these modifications, the search spaces of ERP and CERP are much reduced to the original ANN reduction process.

However, these reductions in the search space imply that when the original ANN reduction, ERP and CERP are applied to the same topology optimisation problem, the former process should find an optimised ANN with at least an equal or better generalasation performance compared to the latter two. In other words, we can prove the effectiveness of the original ANN reduction process (which, however, is impractical) by testing the out-of-sample performance of ERP and CERP using simulations.

In this chapter, empirical results are collected from simulations with ERP and CERP on improving the topology of ANNs for approximating two deterministic functions. Two bespoke functions are designed for applications and each one generates a sample dataset for the simulation.[1].

The two deterministic functions are described separately in the beginning of Sections 4.2 and 4.3 followed by four subsections : Subsection 1 presents the settings of simulation for ANN training and testing in respect to each sample dataset; Subsection 2 records the approximation

[1]The simulations are tested in two platforms: x64 Windows OS with Visual C++ compiler and x64 Mac OS with LLVM compiler. There is a minor rounding difference between the results from the two platform and the former results are displayed in this thesis

results from some complete ANNs; Subsection 3 shows the ERP results from optimisng the top performing complete ANN and some other selected complete ANN listed in Subsection 2; Subsection 4 shows the CERP result from optimising the top performing complete ANN. The analytical discussion are in-line with the summaries concluded in the end of the sections.

4.2 Simulation with deterministic function one

Consider the following function designed for the first approximation problem:

$$\begin{aligned} y &= \frac{1}{200}x_1^3 + 10(x_2^3 + x_3^3)^{\frac{1}{3}} + 100(x_4 + x_5) + 100x_6^{\frac{1}{3}} \\ &\quad + x_6 \log(\exp(x_7) + \exp(x_8)) + x_9 x_{10} \end{aligned}$$

This ten variable function is complex as it consists of a series of non-linear terms including cubic, exponential and logarithm terms. Note that the function is continuously and monotonically increasing with respect to any variable; the constant parameters are carefully adjusted so that within a bounded input space, the corresponding output calculated from the function is not dominated by any input.

The inputs $x_1, ..., x_{10}$ are all integers generated randomly within a range of $[-100, 100]$. From each set of input samples, an output y is calculated accordingly and forms a set of input-target sample pairs.

Using an ANN to approximate the above function in a 10-dimension space requires a sufficiently large size of samples for training and testing, so a total of 1200 input-target sample pairs are generated for this purpose. Specifically, 600 out of the 1200 sample pairs are randomly allocated for training, 200 sample pairs are randomly allocated for validation, another 200 sample pairs are randomly allocated for testing and the rest is allocated for out-of-sample approximation (forecasting).

4.2.1 Optimisation problem settings

Given the sample pairs for the approximation models, we are now ready to construct some complete ANNs to be optimised by the topology optimisation applications. Recall the universal approximation theorem discussed in Section 2.1.2 states that all functions in a broad space can be approximated with arbitrary accuracy using a single-hidden-layer ANN with hyperbolic tangent activation functions; for this reason, all ANNs are constructed using a single hidden layer topology in all simulations.

In addition, there are two configurations needed to be pre-determined on the complete ANN:

(i) The number of hidden neurons
In order to determine the optimal number of neurons on the hidden layer, complete networks with the different number of hidden neurons must be trained and compared. For this reason, different networks are constructed with 10-k-1 topology where $k = 2, 3, ..., 12$ for the simulation.

(ii) The initial search position
Each topology is tested with 20 random initial values of arc weights and biases.

Furthermore, as discussed in Chapter 2, the Hanger-Zhange conjugate gradient search algorithm is applied throughout the training process. In order to ensure that a network is sufficiently and efficiently trained, two conditions governing the algorithm of conjugate gradient search must be specified in advance:

(i) The maximum number of conjugate gradient iterations is capped to 2000 so that the conjugate gradient process has sufficient number of attempts to find the optimal solution;

(ii) The gradient convergence tolerance is limited to a threshold of 0.05 on the norm of search gradients. The search stops if the search gradient becomes less than 0.05.

For the purpose of this simulation, a good test of the topology optimisation problem should start from the complete ANN which has the best testing-sample RMSE. Therefore, a complete

ANN is explicitly selected among the pool of ANNs constructed following the initial training. In this case, a total of $220(20 * 11)$ networks (20 initial start points $\times$ 11 ANN topologies) have been constructed and the best performing one is selected as the first target ANN to be optimisied in this simulation.

4.2.2 Result from complete ANN

Once all 220 complete ANNs are constructed after training, validation and testing, the performance results are compared for selecting the best complete ANN for the topology optimisation simulation.

Table 4.2.2a displays the out-of-sample performance from the 220 complete ANNs, the table rows are indexed by the number of hidden neurons k; and the table columns are indexed by the seed ID which applied for generating the initial parameter vector before the initial training.

	Initial parameter seed ID									
Topology	1	2	3	4	5	6	7	8	9	10
2	0.407551	0.51171	0.436613	0.511952	0.557178	0.434353	0.407475	0.547115	0.568491	0.511836
3	0.567149	0.366283	0.360564	0.339117	0.36829	0.337115	0.337214	0.311993	0.432864	0.363071
4	0.305674	0.310351	0.310072	0.362756	0.308915	0.521407	0.273821	0.283356	0.273346	0.372609
5	0.265458	0.180008	0.178147	0.691958	0.173111	0.304711	0.179155	0.163922	0.179155	0.167922
6	0.16845	0.610069	0.36268	0.156081	0.331456	0.286098	0.147104	0.331549	0.151033	0.318756
7	0.619525	0.179925	0.311941	0.171597	0.118739	0.176791	0.16116	0.1598	0.156958	0.18246
8	0.325656	0.18522	0.183263	0.145587	0.156752	0.185056	0.128355	0.181614	0.156211	0.155771
9	0.169109	0.202688	0.16042	0.127723	0.225309	0.0712184	0.166227	0.117885	0.141904	0.184965
10	0.178173	0.1609	0.190761	0.227093	0.127421	0.163434	0.118882	0.123601	0.165073	0.124715
11	0.121898	0.138827	0.254509	0.134853	0.171568	0.655521	0.157162	0.661449	0.13032	0.195021
12	0.229843	0.226787	0.135845	0.190723	0.152399	0.214244	0.19763	0.509312	0.127189	0.211545

	Initial parameter seed ID									
Topology	11	12	13	14	15	16	17	18	19	20
2	0.434891	0.434679	0.407386	0.436612	0.64104	0.436612	0.511925	0.436619	0.511643	0.511593
3	0.363151	0.508926	0.51176	0.337174	0.337518	0.337344	0.363707	0.530194	0.337101	0.363506
4	0.282214	0.315465	0.346399	0.36277	0.31068	0.379958	0.250061	0.300251	0.238224	0.307901
5	0.283797	0.380361	0.178021	0.16921	0.172659	0.176814	0.324746	0.177408	0.222555	0.305615
6	0.182098	0.403302	0.184603	0.210478	0.219599	0.315518	0.175311	0.185944	0.249338	0.324182
7	0.200147	0.154457	0.184453	0.184689	0.189304	0.216241	0.33225	0.178394	0.325029	0.324056
8	0.144895	0.155169	0.299157	0.54607	0.183398	0.154015	0.142512	0.16257	0.330266	0.123197
9	0.151325	0.336093	0.307371	0.124043	0.155609	0.295608	0.161766	0.187075	0.33999	0.206395
10	0.120321	0.145696	0.213005	0.164722	0.184223	0.127638	0.122117	0.201204	0.177389	0.142277
11	0.125982	0.158079	0.547352	0.155892	0.162057	0.17036	0.126067	0.203779	0.623782	0.172392
12	0.15725	0.156833	0.179241	0.167167	0.137754	0.62059	0.12775	0.190972	0.082723	0.219955

Table 4.2.2a Performance measured as testing-sample RMSE from all 220 constructed complete ANNs. The best result is highlighted in red.

As it is indicated in the table, the top performing complete ANN, which is to be applied for both simulation of ERP and CERP, is obtained from the 10-9-1 topology and it is trained using an initial search position generated from seed ID 6. For the simulation of ERP, in addition to optimising the top performing complete network, the second to the sixth best performing complete ANN are also selected to be optimised by the application. The topology of these chosen complete ANN networks and their seed ID are listed in Table 4.2.2b.

Note that according to the flowchart of ERP described in Chapter 3, the parameter vector of the optimised ANN is obtained after receiving two rounds of training in total (the first round is the initial training on the complete ANN and the second round is for re-training the topologically optimised ANN). Therefore, any comparison on the output performance between the complete ANN and the topologically optimised ANN should be performed on the parameters produced after the same depth of training to get the topologically optimisied ANN from the fully trained complete ANN.

For the above reason, two testing-sample performance result on each of the selected complete networks are recorded in Table 4.2.2b: Column 3 is the performance produced from a maximum training iteration of 2000; Column 4 is the performance produced from a maximum training iteration of 4000. In some cases, if the training condition of gradient convergence tolerance is satisfied and the training procedure is terminated before the 2000 iteration, the increment on the maximum iteration setting does not affect any of the outcome and the two columns can have an identical value. Figure 4.2.2 show the graphical representation of the performance result listed in the table.

Rank	Topology	Random Seed ID	Performance of the complete ANN trained up to 2000 iterations	Performance of the complete ANN trained up to 4000 iterations
1	10-9-1	6	0.0712184	0.05652
2	10-12-1	19	0.082723	0.07658
3	10-9-1	8	0.117885	0.11477
4	10-7-1	5	0.118739	0.11343
5	10-10-1	7	0.118882	0.11157
6	10-10-1	11	0.12032	0.12032

Table 4.2.2b Performance (testing-sample RMSE) of the selected complete ANNs at different level of maximum training iteration

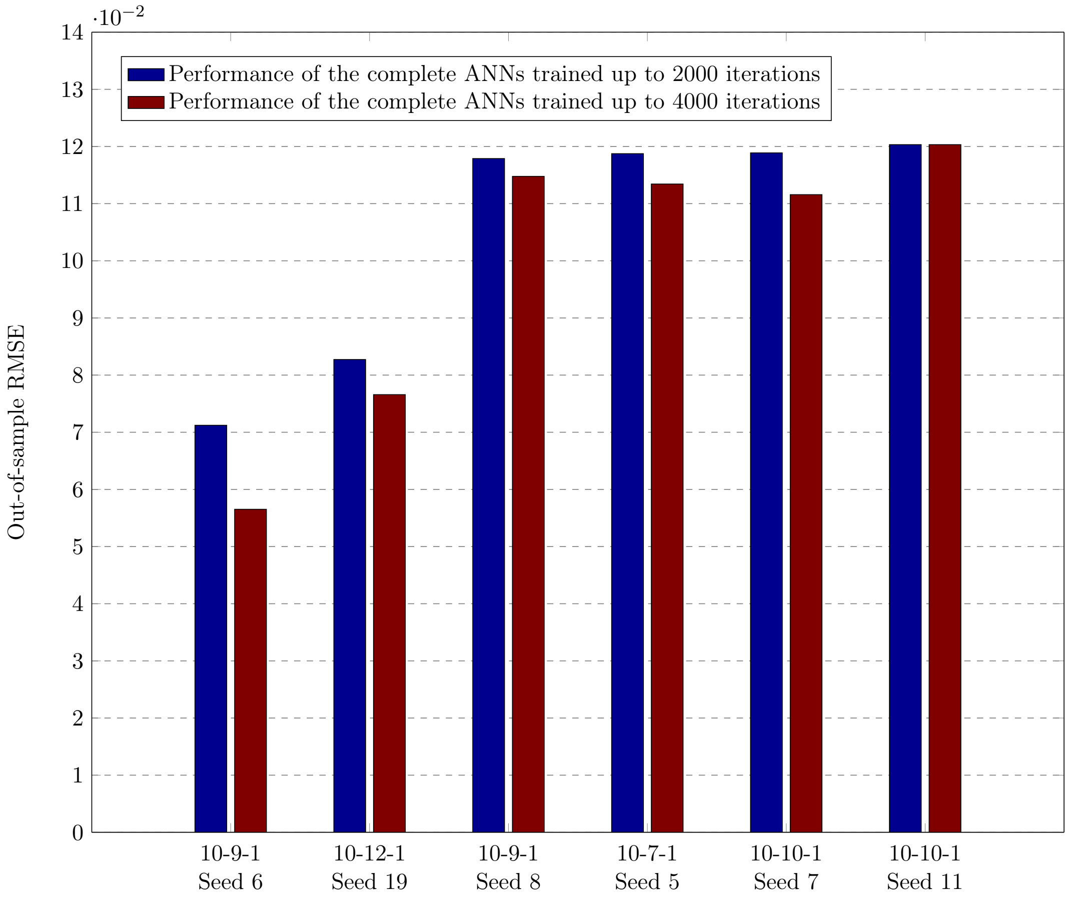


Figure 4.2.2 Graphical representation of testing-sample performance from the complete ANNs (with seed ID) subject to topology optimisation

The following Table 4.2.2c shows the parameter vector of the top performing complete 10-9-1 ANN, obtained from the initial parameter vector generated from random generator using seed ID of 6.

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10
Neuron 1	0.815287	-0.0149804	-0.0006692	0.00077773	-0.018028	-0.0130862	0.0159306	-0.0136298	-0.0256615	0.302132	0.262766
Neuron 2	-0.0015882	-0.004493	0.0119685	-0.0081415	0.00549122	-0.0157849	0.162285	-0.523953	0.507484	-0.0085196	-0.0002807
Neuron 3	0.738853	0.00123026	0.00083283	-0.0051976	0.00321306	0.00111096	0.253726	-0.168008	-0.157757	-0.0640853	0.10079
Neuron 4	0.00750358	-0.0090472	0.0128761	-0.0103685	0.0049237	-0.0116615	-0.161937	-0.614361	0.607345	-0.0065514	-0.0024366
Neuron 5	-0.760023	0.00472197	-0.0043851	0.00567956	0.00195972	0.00370975	0.239502	0.165494	0.175495	-0.0372796	0.067102
Neuron 6	0.812022	0.00999102	0.00229457	0.0023436	0.00948603	0.00942313	0.014975	-0.0771023	-0.0752904	0.28919	-0.247271
Neuron 7	-0.006682	-1.66994	0.00977476	0.00367253	-0.015478	-0.0004025	0.00791051	0.00988735	0.00563888	0.0255108	-0.0017587
Neuron 8	-1.13737	0.185726	0.0140008	-0.0068822	0.258468	0.187348	-0.156419	-0.0904841	0.0272234	0.233192	0.310606
Neuron 9	0.232437	0.0629526	0.00791475	0.00836278	0.0821993	0.0809821	-0.0497592	-0.0057086	0.0290013	0.0137922	0.0925956
Hidden to output layer	Bias	Neuron 1	Neuron 2	Neuron 3	Neuron 4	Neuron 5	Neuron 6	Neuron 7	Neuron 8	Neuron 9	
Output neuron	-0.880781	-2.90057	-1.77148	2.76931	1.28485	3.08203	3.25075	0.343027	0.618533	6.00623	

Table 4.2.2c Trained parameter vector of the complete 10-9-1 complete ANN

4.2.3 Result of topologically-optimising complete ANN for function one using ERP

Given a complete ANN to be optimised, ERP first prunes some arcs to construct a basic ANN. In each iteration of the systematic pruning process, a path from any input to the output with the smallest weight is found and the arc between the input vertex and the hidden vertex on the path is disconnected. As a result of the pruning, weight parameters on the remaining arcs are the same as the trained complete ANN, while weight parameters for the disconnected arcs are replaced with 0.

From the top performing ANN with seed ID of 6 listed in Table 4.2.2c, the parameter vector of the corresponding basic ANN constructed is listed in Table 4.2.3a and the corresponding graph is shown in Figure 4.2.3a.

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10
Neuron 1	0.815287	0	0	0	0	0	0	0	0	0	0.262766
Neuron 2	-0.0015882	0	0	0	0	0	0	-0.523953	0.507484	0	0
Neuron 3	0.738853	0	0	-0.0051976	0	0	0	0	0	0	0
Neuron 4	0.00750358	0	0.0128761	0	0	0	0	0	0	0	0
Neuron 5	-0.760023	0	0	0	0	0	0.239502	0	0	0	0
Neuron 6	0.812022	0	0	0	0	0	0	0	0	0.28919	0
Neuron 7	-0.006682	-1.66994	0	0	0	0	0	0	0	0	0
Neuron 8	-1.13737	0	0	0	0.258468	0	0	0	0	0	0
Neuron 9	0.232437	0	0	0	0	0.0809821	0	0	0	0	0

Hidden to output layer	Bias	Neuron 1	Neuron 2	Neuron 3	Neuron 4	Neuron 5	Neuron 6	Neuron 7	Neuron 8	Neuron 9
Output neuron	-0.880781	-2.90057	-1.77148	2.76931	1.28485	3.08203	3.25075	0.343027	0.618533	6.00623

Table 4.2.3a Basic ANN constructed from the 10-9-1 network

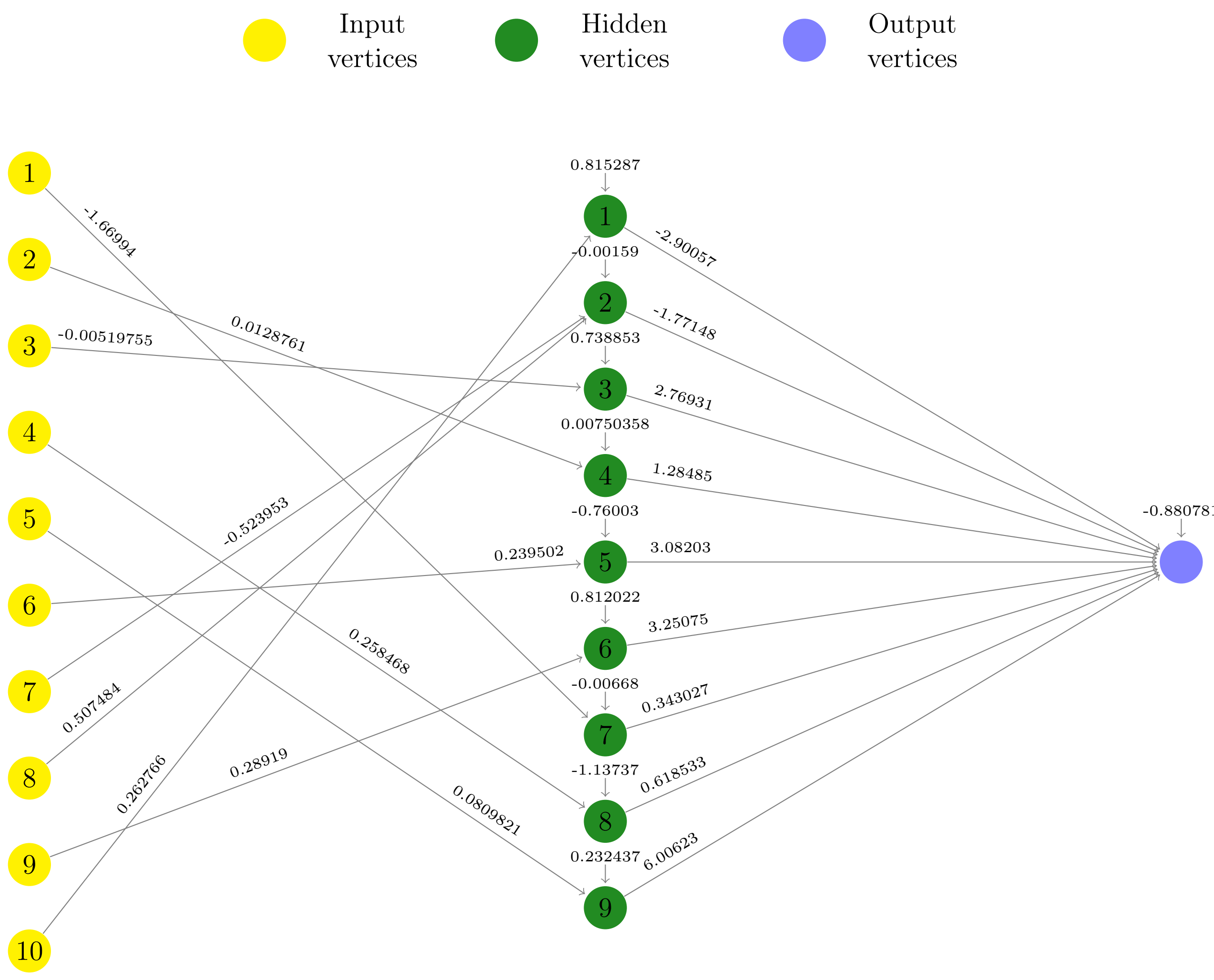


Figure 4.2.3a Corresponding graph of the network in Table 4.2.3a

The next stage of ERP produces some new ANNs by adding a number of arcs on the basic ANN. Following the algorithm described in Chapter 3 and for this simulation:

(i) A specified number of arcs are repetitively and randomly added on the basic ANN to form 50 ANNs with a total of $\hat{a}$ arcs in each;

(ii) Among all the constructed ANNs, the one with the lowest training-sample RMSE is selected for retraining of up to 2000 iterations.

In order to search for the ANN with an optimal number of arcs, it is necessary to repeat the arc re-connection process with different value of $\hat{a}$. For the simulation, this is done on all possible number of arcs that can be added on the basic ANN. Then an ANN with the best testing-sample performance is selected for the output of the ERP.

Continuing the simulation on the 10-9-1 basic ANN with the parameter vector presented in Table 4.2.3a, given a number of arcs specified to be added in the second stage, the arcs are randomly added on the basic ANN and this process is repeated for 50 times in the simulations so 50 new ANNs are constructed. The re-training is applied on the best training-sample performing ANN and the testing-sample RMSE is calculated. Table 4.2.3b shows the performance result of the selected ANN given each number of arcs being added $0, 1, ..., 79$. Note that the basic ANN has 19 arcs, so the corresponding values of $\hat{a}$ is $19, 20, ..., 98$.

Num. Arc added	0	1	2	3	4	5	6	7	8	9
Out-of-Sample	0.55763	0.556229	0.55256	0.388283	0.56316	0.555014	0.473678	0.371905	0.3776	0.565976
Num. Arc added	10	11	12	13	14	15	16	17	18	19
Out-of-Sample	0.550643	0.540459	0.399693	0.516637	0.541297	0.480672	0.516487	0.407611	0.387844	0.231651
Num. Arc added	20	21	22	23	24	25	26	27	28	29
Out-of-Sample	0.299936	0.217275	0.314385	0.166259	0.17033	0.175397	0.173021	0.207397	0.119497	0.388971
Num. Arc added	30	31	32	33	34	35	36	37	38	39
Out-of-Sample	0.298098	0.204808	0.119494	0.120723	0.183095	0.202116	0.117857	0.177541	0.325603	0.164346
Num. Arc added	40	41	42	43	44	45	46	47	48	49
Out-of-Sample	0.161753	0.116935	0.180557	0.122268	0.121969	0.249241	0.128503	0.168778	0.128884	0.12005
Num. Arc added	50	51	52	53	54	55	56	57	58	59
Out-of-Sample	0.116665	0.121963	0.122563	0.132488	0.124668	0.176828	0.0836928	0.174734	0.0652948	0.0659341
Num. Arc added	60	61	62	63	64	65	66	67	68	69
Out-of-Sample	0.0740358	0.153482	0.0664806	0.0626142	0.169649	0.0620185	0.068024	0.0562831	0.0612434	0.0558611
Num. Arc added	70	71	72	73	74	75	76	77	78	79
Out-of-Sample	0.0571836	0.056823	0.0568077	0.057147	0.0564708	0.0564332	0.0568085	0.0567244	0.056856	0.0563009

Table 4.2.3b Testing-sample performance of the best performing ANN produced with indicated number of arcs being added on the basic ANN (starting graph 10-9-1, seed6)

The result in Table 4.2.3c has shown that the best testing-sample performing ANN is produced from adding 69 arcs on the 10-9-1 basic ANN. With an out-of-sample standardised RMSE of 0.0558611 from the ANN, this is an 1.16% improvement from the complete ANN which has a testing-sample standardised RMSE of 0.05652. Therefore, we conclude a topologically-optimised ANN has been found by ERP and the parameter vector of the network is listed in the Table 4.2.3d (pre-retraining) and Table 4.2.3e (post-retraining)

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10
Neuron 1	0.815287	-0.0149804	0	0	-0.018028	0	0.0159306	-0.0136298	-0.0256615	0.302132	0.262766
Neuron 2	-0.0015882	-0.004493	0.0119685	-0.0081415	0.00549122	-0.0157849	0.162285	-0.523953	0.507484	-0.0085196	-0.0002807
Neuron 3	0.738853	0.00123026	0.00083283	0	0	0.00111096	0.253726	-0.168008	-0.157757	-0.0640853	0.10079
Neuron 4	0.00750358	-0.0090472	0.0128761	-0.0103685	0.0049237	-0.0116615	-0.161937	-0.614361	0.607345	-0.0065514	-0.0024366
Neuron 5	-0.760023	0.00472197	-0.0043851	0.00567956	0.00195972	0	0.239502	0.165494	0.175495	-0.0372796	0.067102
Neuron 6	0.812022	0	0.00229457	0	0.00948603	0	0	-0.0771023	-0.0752904	0.28919	-0.247271
Neuron 7	-0.006682	-1.66994	0.00977476	0.00367253	0	-0.0004025	0.00791051	0.00988735	0.00563888	0.0255108	-0.0017587
Neuron 8	-1.13737	0.185726	0.0140008	-0.0068822	0.258468	0.187348	-0.156419	-0.0904841	0.0272234	0.233192	0.310606
Neuron 9	0.232437	0.0629526	0.00791475	0.00836278	0.0821993	0.0809821	-0.0497592	-0.0057086	0.0290013	0.0137922	0.0925956
Hidden to output layer	Bias	Neuron 1	Neuron 2	Neuron 3	Neuron 4	Neuron 5	Neuron 6	Neuron 7	Neuron 8	Neuron 9	
Output vertex	-0.880781	-2.90057	-1.77148	2.76931	1.28485	3.08203	3.25075	0.343027	0.618533	6.00623	

Table 4.2.3d Parameter of the topologically-optimised 10-9-1 ANN from ERP (pre-retraining)

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10
Neuron 1	0.763097	-0.0073454	0	0	6.02E-05	0	0.0112408	-0.0104286	-0.0140099	0.263305	0.235631
Neuron 2	0.00689082	-0.0020497	0.0115126	-0.0048973	0.00537161	-0.0064154	0.136069	-0.540696	0.52815	-0.0068527	1.98E-05
Neuron 3	0.741616	-0.0026557	0.00190684	0	0	0.00030411	0.236602	-0.154717	-0.147468	-0.0414116	0.0658725
Neuron 4	0.0149547	-0.0018397	0.0107743	-0.007176	0.00257174	-0.0073326	-0.135505	-0.560548	0.556747	-0.0073115	-0.0009837
Neuron 5	-0.756364	0.00080349	-0.0013835	0.00608684	-0.005117	0	0.228757	0.154486	0.159676	-0.0505272	0.0515318
Neuron 6	0.774157	0	0.00048555	0	0.00171301	0	0	-0.0559592	-0.0586401	0.250008	-0.226613
Neuron 7	0.00576105	-0.686811	0.00487147	0.00258587	0	-0.0009362	0.00665927	-0.0030159	0.00209987	0.00370649	0.00868127
Neuron 8	-1.184	0.308459	0.0109257	0.00529496	0.180629	0.162509	-0.116828	0.01261	0.0399712	0.0605006	0.272171
Neuron 9	0.386146	0.142264	0.00581273	0.00607273	0.0868309	0.0831904	-0.0554277	0.00630236	0.0151294	0.0259587	0.125863
Hidden to output layer	Bias	Neuron 1	Neuron 2	Neuron 3	Neuron 4	Neuron 5	Neuron 6	Neuron 7	Neuron 8	Neuron 9	
Output vertex	-1.05684	-3.87272	-1.89626	3.27111	1.73053	3.50441	4.1855	1.4639	1.6913	6.4779	

Table 4.2.3e Parameter of the topologically-optimised 10-9-1 ANN from ERP (post-retraining)

Table 4.2.3f summarises the results of ERP on optimising all six chosen networks listed in Table 4.2.2b using the same setting described above. The standardised testing-sample performance for the complete ANNs in column 4 is compared to the performance of the optimised ANNs in columns 6. The number of arcs in the compete ANNs and the topologically-optimised ANN is compared between column 5 and 7. The percentage of performance improvement for each case is listed in column 8. These results show that ERP has successfully produced a topologically-optimised ANN for each of the 6 target complete ANNs. In addition, a graphical performance comparison of the complete ANNs and the corresponding topologically-optimised ANNs produced from ERP is shown in Figure 4.2.3b.

Performance rank of Complete ANN	Topology	Random Seed ID	Performance of Complete ANN	Num. of arcs in the Complete ANN	Performance of Topology optimised ANN	Num. of arcs in the Optimised ANN	Performance improvement
1	10-9-1	6	0.05652	99	0.0558611	88	1.16%
2	10-12-1	19	0.07658	132	0.06607	114	13.73%
3	10-9-1	8	0.11477	99	0.111028	81	3.26%
4	10-7-1	5	0.11343	77	0.109412	55	3.54%
5	10-10-1	7	0.11157	110	0.108516	69	2.73%
6	10-10-1	11	0.12032	110	0.045507	95	62.18%

Table 4.2.3f Testing-sample result of ERP: topology and performance comparison for the selected networks

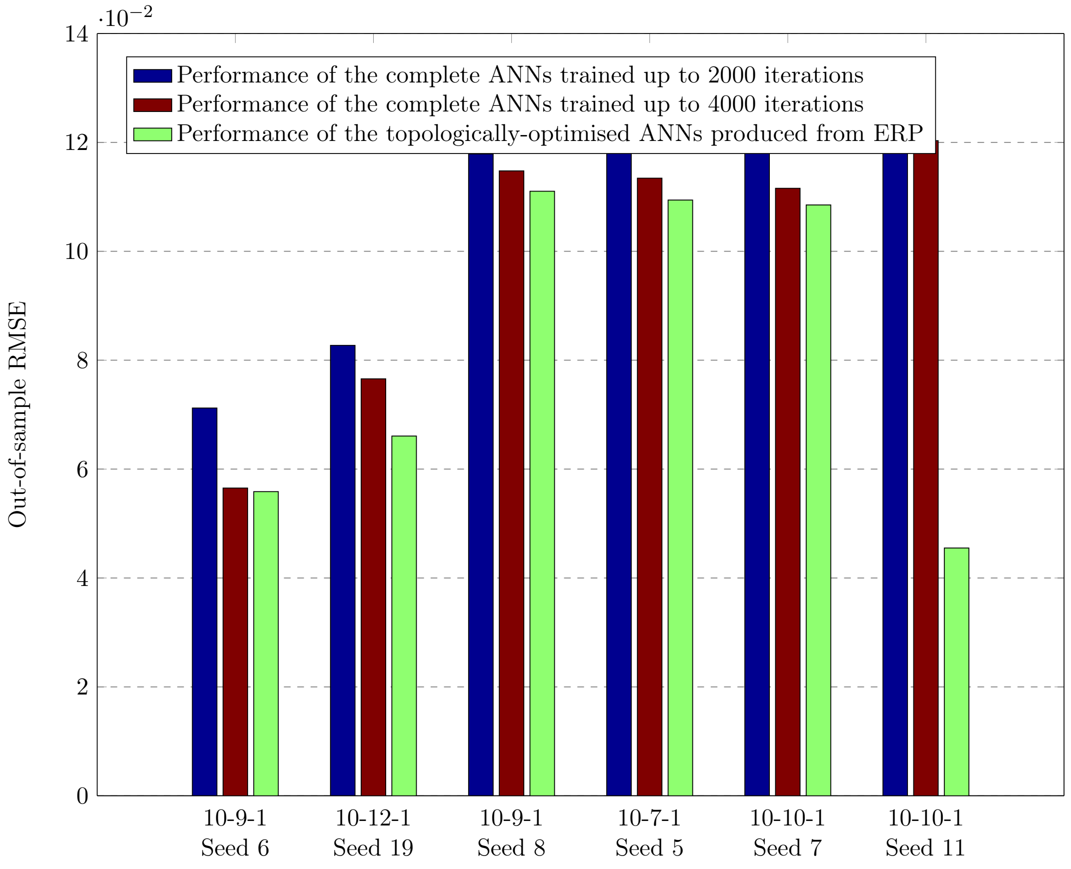


Figure 4.2.3b Graphical performance (measured as testing-sample RMSE) comparison of the complete ANNs and the corresponding topologically-optimised ANNs produced from ERP

From Table 4.2.3f and Figure 4.2.3b it is important to note that the overall best ANN is obtained not from the best performing complete and fully trained starting topology, but from the starting topology ranked in 6th place (with 110 arcs). For this 10-10-1 complete ANN the algorithm removed 15 arcs (reducing the number to 95) and obtained a topologically-optimised ANN with an RMSE of 0.045507 instead of 0.0558611, an improvement of 18.54%.

Next, the forecasting sample pairs are applied on the above listed topologies for out-of-sample performance evaluation. The result is shown in Table 4.2.3g and the graphical result is presented in Figure 4.2.3c.

Performance rank of Complete ANN	Topology	Random Seed ID	Performance of Complete ANN	Num. of arcs in the Complete ANN	Performance of Topology optimised ANN	Num. of arcs in the Optimised ANN	Performance improvement
1	10-9-1	6	0.0573485	99	0.0549979	88	4.27%
2	10-12-1	19	0.0812	132	0.07203	114	12.73%
3	10-9-1	8	0.12201	99	0.118311	81	3.13%
4	10-7-1	5	0.13153	77	0.12957	55	1.51%
5	10-10-1	7	0.13089	110	0.13016	69	0.56%
6	10-10-1	11	0.122753	110	0.106092	95	15.70%

Table 4.2.3g forecasting-sample result of ERP: topology and performance comparison for the selected networks

4.2.4 Result of optimising complete ANN for function one using CERP

This subsection demonstrates the simulation of CERP on the top performing complete ANN with 10-9-1 topology displayed in Table 4.2.2c. We have shown in Chapter 3 that both ERP and CERP follow the same procedures in the basic ANN construction stage, so the intermediate result from the first stage is not explicitly written again. All the training settings are the same as in Subsection 4.2.2.

In the second stage of CERP, the basic ANN in Table 4.2.3a is retrained and forms a temporary reference for the following iterations of arc re-connection. In each iteration, a single arc is added to the reference ANN and the addition is repeated for all disconnected pairs of vertices between the input layer and the hidden layer to construct new ANNs. The output from each iteration is the ANN with the best performance improvement (based on the testing-sample RMSE) compared to the reference ANN, which is then replaced as a new reference for the next iteration; otherwise the iteration is terminated if there is no improvement. Table

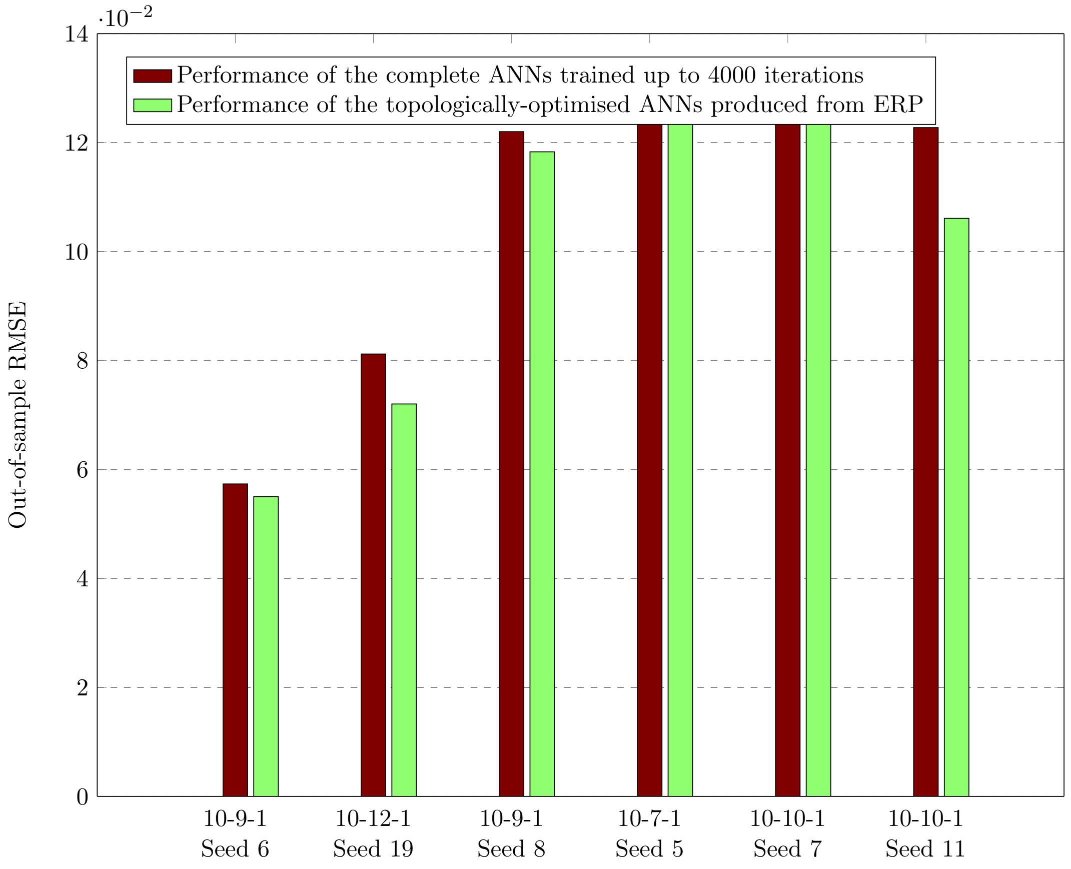


Figure 4.2.3c Graphical performance (measured as forecasting-sample RMSE) comparison of the complete ANNs and the corresponding topologically-optimised ANNs produced from ERP

4.2.4a records the testing-sample performance of the reference ANN in each iteration until the process is terminated.

The above table shows that, on the problem of optimising the complete 10-9-1 ANN, CERP algorithm is terminated after adding 33 arcs on to the basic ANN. However, according to the out-of-sample performance of the output network, a standardised RMSE of 0.110326 is higher than the initial complete ANN, which is 0.05652. In other words, CERP fails to find the optimised network for this example.

Figure 4.2.4 shows the graphical results of the performance from the ANNs produced during CERP arc re-connection, iterations in Table 4.2.4a as well as the performance of initial complete ANN in Table 4.2.2b and the corresponding performance of the topologically-optimised ANN produced by ERP in Table 4.2.3f.

Iteration Number	0	1	2	3	4	5
Testing Sample RMSE	0.557630	0.467387	0.381763	0.283653	0.240733	0.172604
Iteration Number	6	7	8	9	10	11
Testing Sample RMSE	0.167410	0.165066	0.162711	0.162054	0.161218	0.160449
Iteration Number	12	13	14	15	16	17
Testing Sample RMSE	0.159641	0.158723	0.158466	0.157970	0.156414	0.155577
Iteration Number	18	19	20	21	22	23
Testing Sample RMSE	0.137594	0.123019	0.119561	0.117362	0.116146	0.115045
Iteration Number	24	25	26	27	28	29
Testing Sample RMSE	0.113626	0.112955	0.112111	0.111736	0.111403	0.111053
Iteration Number	30	31	32	33		
Testing Sample RMSE	0.110900	0.110527	0.110510	0.110326		

Table 4.2.4a Iteration number to the performance on testing samples results from CERP intermediate steps

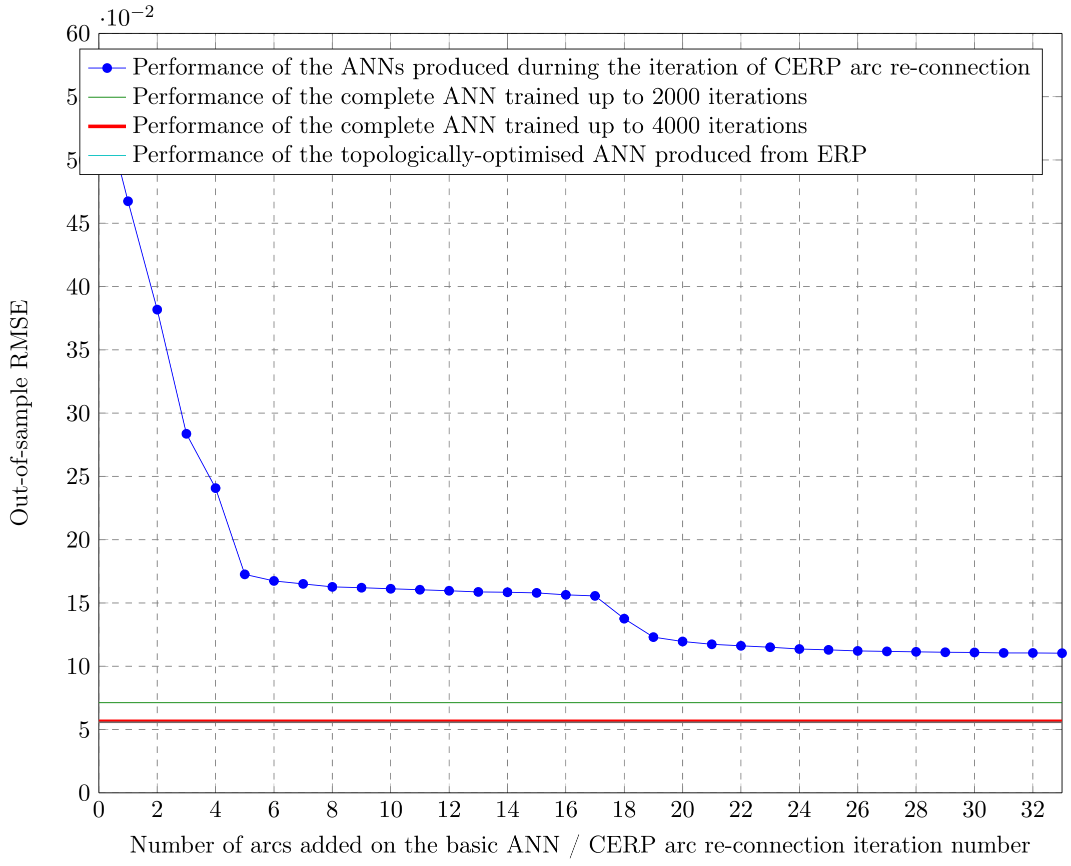


Figure 4.2.4 Testing-sample performance comparison between the ANNs produced during the iteration of CERP, the complete ANN and the ERP optimised ANN

From Figure 4.2.4 and Table 4.2.4b we note the following:

(i) The improvement in the RMSE is not smooth with respect to the number of arcs added.

(ii) As a result of (*i*) it is impossible to predict whether continuing the single arc re-connection (when the previous steps produced negligible improvements) is worthwhile or not. For example, at the time that 17 arcs were added, the previous 4 or 5 arc re-connection produced negligible improvement but when 18 and 19 were added there was a very significant improvement in the error.

(iii) Observation (*ii*) above is typical of combinatorial optimisation problems where local optimisation seems to produce a convergent solution but more "extended" local optimisation (for example adding 2,3 more arcs simultaneously) can produce significantly better results.

The above comments explain why the process of CERP arc re-connection iteration terminated with 33 arcs with no improvement possible by adding any other single arc, but clearly (considering that the complete connected graph has better RMSE) an improvement is possible if a set of arcs (more than one arc) is added during the iterations.

Finally as a summary of the simulations for both ERP and CERP applications on the function one's datasets, Table 4.2.4b lists the performance comparison between the best performing complete ANN and the corresponding topologically-optimised ANNs produced from ERP and CERP.

Topology	Standardised testing-sample RMSE	Standardised forecasting-sample RMSE	Number of connections	Testing-sample performance improvement	Forecasting-sample performance improvement
Complete ANN	0.05652	0.0573485	99		
ERP optimised ANN	0.0558611	0.0549979	88	1.16%	1.92%
CERP optimised ANN	0.110326	0.134012	52	-	-

Table 4.2.4b Performance improvement from ERP and CERP to the best complete ANN for approximating the function one.

4.3 Simulate with sample data from deterministic function two

Consider the following function proposed for generating the training and testing sample pairs in the simulations:

$$\begin{aligned} y &= 100*(x_2-x_1)^2+100*(x_3-x_2)^2+100*(x_4-x_3)^2 \\ &\quad +100*(x_5-x_4)^2+100*(x_1-x_5)^2+(1-x_1)^2 \\ &\quad +(1-x_2)^2+(1-x_3)^2+(1-x_4)^2+(1-x_5)^2 \end{aligned}$$

This function is constructed using five independent variables. To differentiate from the function one used in the previous simulations, the independent variables of function two are structured in a symmetrical form. All independent variables $x_1, ..., x_5$ are randomly generated integers between [-100,100] and the dependent variable y is calculated accordingly to form an input-output sample pair.

A total of 1200 sample pairs is generated. From which 600 samples are randomly allocated for training and the rest are divided equally and randomly allocated for validation, testing and forecasting.

4.3.1 Problem settings

For the same reason as stated in Subsection 4.2.1, we want to select the best performing complete ANNs as the subject of topology optimisation simulation. There are 5 independent variables and 1 dependent variable for each sample pair so the 5-k-1 complete ANNs are constructed accordingly, where $k = 2, 3, ..., 10$. For each value of k, 20 sets of weight parameters are generated randomly for training. As a result, a total of $200(20*10)$ ANNs are constructed.

The training settings have remained the same as in the previous simulations:

(i) The number of training iterations is capped to 2000;

(ii) The gradient convergence tolerance (in norm value) is limited to 0.05.

4.3.2 Result from complete ANN

After training the complete ANNs, the testing samples are fed into the ANNs for the testing-sample performance calculation. Table 4.3.2a shows the result of the performance measurement on all 200 ANNs, where the rows are indexed by the number of hidden neuron k and the columns are indexed by the seed ID for generating the initial weight parameters.

	Initial Search ID									
Number of Hidden Neuron	1	2	3	4	5	6	7	8	9	10
2	0.4555941	0.60757303	0.58196519	0.5355216	0.58196519	0.51153372	0.45559127	0.51645591	0.50256729	0.50258145
3	0.24082471	0.50525496	0.44041688	0.43959274	0.44080488	0.59627295	0.50820034	0.5226752	0.52019994	0.44033758
4	0.23627834	0.23757261	0.23726277	0.23813478	0.51022529	0.23540209	0.23699061	0.54321924	0.2359838	0.55611662
5	0.24640111	0.23299112	0.2327739	0.21459097	0.19157847	0.18873136	0.25876464	0.40068253	0.23119557	0.23148303
6	0.18560331	0.24573103	0.19439159	0.24390914	0.18092921	0.05536895	0.18782141	0.05772356	0.08960336	0.25396904
7	0.05694898	0.10838726	0.04657387	0.04865291	0.07128818	0.56463841	0.20000226	0.04985259	0.07867401	0.04692901
8	0.04502584	0.10372534	0.58789277	0.07063935	0.24035911	0.04804344	0.04578541	0.55137002	0.04544528	0.07031649
9	0.03122248	0.04456902	0.10249026	0.04437078	0.22673558	0.05293703	0.11368357	0.07015591	0.0318141	0.04695932
10	0.04385449	0.11752474	0.04428213	0.05564112	0.33504014	0.35167589	0.0235972	0.02959175	0.47117914	0.03216217

	Initial Search ID									
Number of Hidden Neuron	11	12	13	14	15	16	17	18	19	20
2	0.58196519	0.58196519	0.58196519	0.58196519	0.58196519	0.58196519	0.58196519	0.58196519	0.58196519	0.58196519
3	0.50716379	0.59502683	0.44043671	0.53999915	0.59505798	0.50729973	0.54983219	0.5072091	0.44015066	0.50374261
4	0.54424446	0.23572891	0.23639445	0.51067276	0.23617383	0.23769665	0.23939025	0.5140996	0.47851711	0.59168778
5	0.26783132	0.23166541	0.23380053	0.24139821	0.23004998	0.26038828	0.19267251	0.46006315	0.28100709	0.25706
6	0.20275138	0.19188405	0.18962205	0.25138702	0.25112816	0.05390957	0.18105977	0.05287897	0.59279797	0.05478497
7	0.08434246	0.04938048	0.32048315	0.04772285	0.05034084	0.07658874	0.56777354	0.08640196	0.53343717	0.17599683
8	0.04979283	0.0438644	0.04143389	0.08521673	0.0583959	0.04673388	0.07784562	0.0575525	0.04889959	0.0973265
9	0.04579391	0.04328014	0.12927767	0.0341469	0.03940243	0.04576191	0.08980926	0.05051898	0.0473122	0.0435203
10	0.04538552	0.06352426	0.03067871	0.04009289	0.14322887	0.06369391	0.04918308	0.07505544	0.0508257	0.08720231

Table 4.3.2a Performance measured as testing-sample RMSE from all 200 complete ANN constructed. The best result is highlighted in red.

The above table indicates that the top performing complete ANN has a 5-10-1 topology trained from a set of the initial parameters obtained by random number generator using seed ID of 7. This network is selected as the initial ANN to be optimised by ERP.

In the simulations with data generated from function one, we optimised all 6 best perfoming ANNs using ERP. Similarly in this simulation for function two, we want to also test the effectiveness of the optimsiation process on different networks. Because the ten best performing complete topologies are all produced from topologies with either 9 or 10 hidden neurons (but with different weight parameters), we are selecting the second and the third best performing complete ANN together with another three networks to be processed in the following simulations by ERP. The performance of the six selected complete ANNs is listed in Table 4.3.2b. Figure 4.3.2 shows the graphical representation of the performance result listed in that table.

Rank	Topology	Random Seed ID	Performance of the complete ANN trained up to 2000 iterations	Performance of the complete ANN trained up to 4000 iterations
1	5-10-1	7	0.0235972	0.0188145
2	5-10-1	8	0.02959175	0.02712
3	5-9-1	1	0.03122248	0.02522
-	5-6-1	6	0.05536895	0.05537
-	5-5-1	6	0.18873136	0.18361
-	5-3-1	1	0.24082471	0.24082

Table 4.3.2b Performance (testing-sample RMSE) of the selected complete ANNs at different levels of maximum training iteration

Table 4.3.2c shows the parameter vector of the top performing 5-10-1 complete ANN.

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5
Neuron 1	0.197664	0.0385426	0.0939453	0.0585473	0.157778	0.165421
Neuron 2	2.17103	0.0502977	0.10233	0.0811365	0.234934	0.255555
Neuron 3	-1.63444	0.596404	-0.278238	0.0284927	0.0919592	-0.346744
Neuron 4	0.775261	-0.0715918	-0.0229444	-0.013542	0.475701	-0.0529839
Neuron 5	1.28857	0.468474	-0.210515	0.0202325	0.071339	-0.268478
Neuron 6	0.675667	-0.071445	0.943846	0.203166	0.395814	0.13755
Neuron 7	-1.66532	-0.0327849	-0.281185	0.651712	-0.287607	0.0701861
Neuron 8	0.802541	-0.0397389	0.447003	-0.00236289	-0.0392217	-0.103334
Neuron 9	0.788407	0.0285906	-0.0117311	-0.0252995	-0.0982598	0.446585
Neuron 10	1.36006	-0.0163542	-0.22312	0.478412	-0.225713	0.0582374

Hidden to output layer	Bias	Neuron 1	Neuron 2	Neuron 3	Neuron 4	Neuron 5	Neuron 6	Neuron 7	Neuron 8	Neuron 9	Neuron 10
Output vertex	7.87373	3.78169	7.65505	2.13229	-3.97619	-3.88677	-0.065674	1.9114	-2.80242	-4.22502	-3.82434

Table 4.3.2c Parameter vector of the 5-10-1 complete ANN

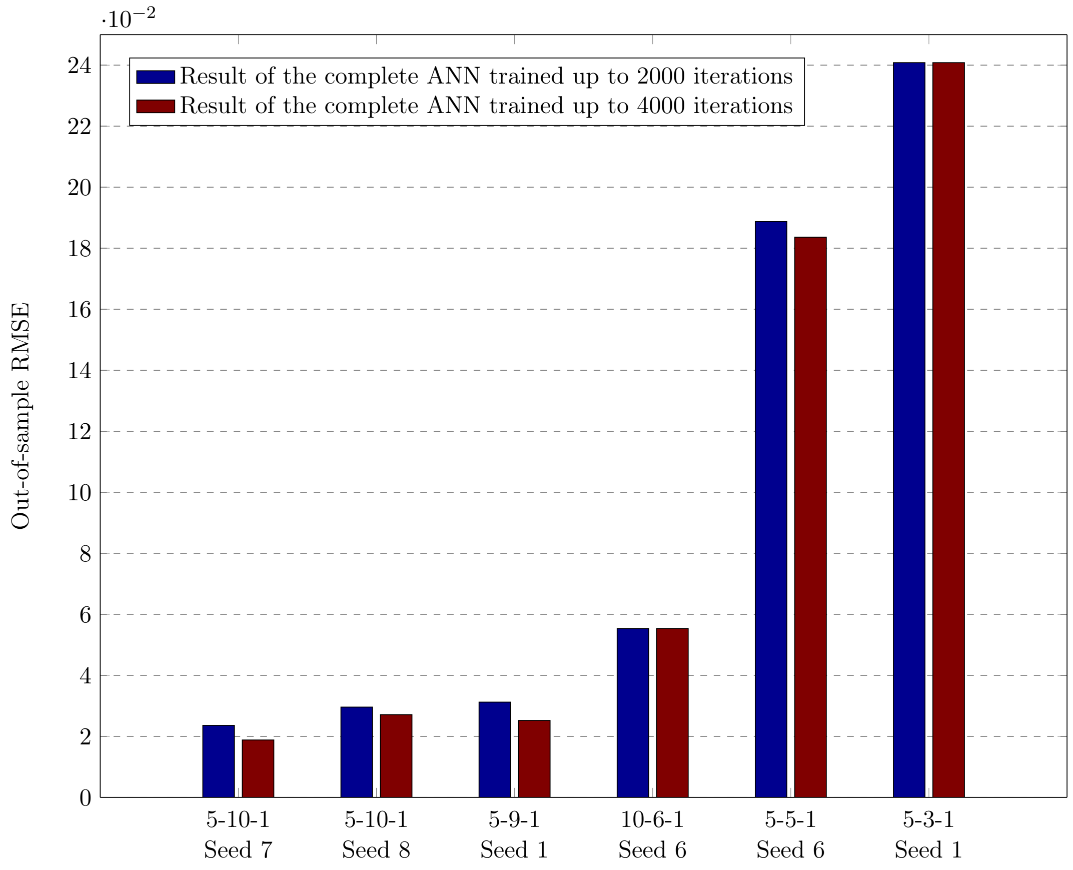


Figure 4.3.2 Graphical representation of testing-sample performance for the complete ANNs subject to topology optimisation

4.3.3 Results of ERP

From the top performing 5-10-1 complete ANN listed in Table 4.3.2c, the first stage of ERP prunes the network into a basic ANN. This produces the parameter vector recorded in Table 4.3.3a; the corresponding graph is shown in Figure 4.3.3a

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5
Neuron 1	0.197664	0	0	0	0	0.165421
Neuron 2	2.17103	0	0	0	0	0.255555
Neuron 3	-1.63444	0.596404	0	0	0	0
Neuron 4	0.775261	0	0	0	0.475701	0
Neuron 5	1.28857	0.468474	0	0	0	0
Neuron 6	0.675667	0	0.943846	0	0	0
Neuron 7	-1.66532	0	0	0.651712	0	0
Neuron 8	0.802541	0	0.447003	0	0	0
Neuron 9	0.788407	0	0	0	0	0.446585
Neuron 10	1.36006	0	0	0.478412	0	0

Hidden to output layer	Bias	Neuron 1	Neuron 2	Neuron 3	Neuron 4	Neuron 5	Neuron 6	Neuron 7	Neuron 8	Neuron 9	Neuron 10
Output vertex	7.50794	3.27796	7.28459	1.99891	-3.68492	-3.735	-0.0823348	1.79094	-2.6395	-3.92952	-3.67265

Table 4.3.3a Basic ANN constructed from the 5-10-1 complete ANN

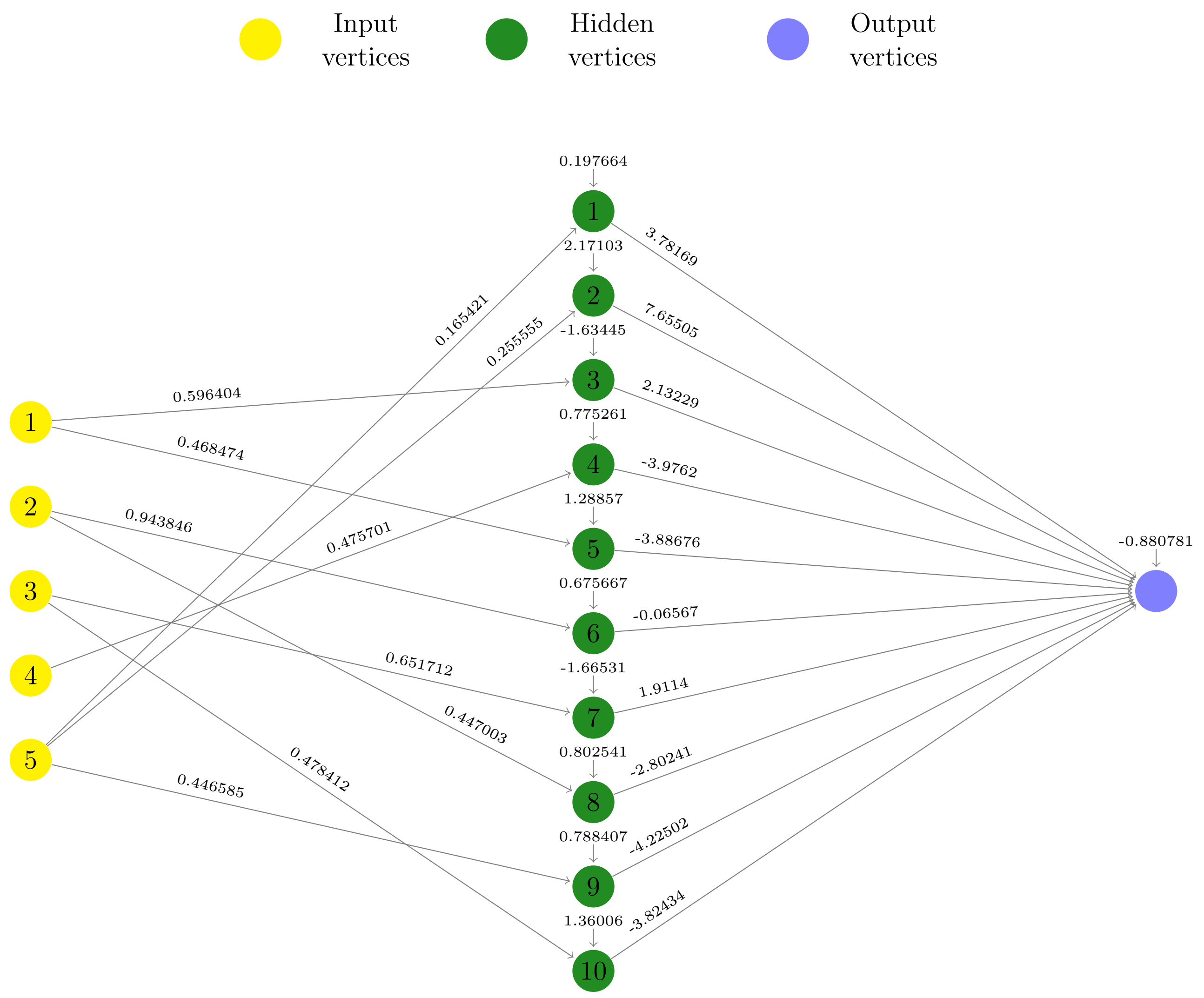


Figure 4.3.3a Corresponding graph of the network in Table 4.3.3a

On the second stage of ERP, $\hat{a}$ numbers of connections are randomly added on to the 5-10-1 basic ANN to construct some new ANNs. Particularly for this simulation, the number of arcs to be added is $0, 1, ..., 39$ and the corresponding $\hat{a} = 20, 21..., 59$. For each $\hat{a}$, the random process is repeated for 50 times and ERP selects the best in-sample performing ANN out of the 50. The selected ANN is further trained for up to 2000 iterations before the performance calculation. Table 4.3.3b shows the out-of-sample RMSE of the selected ANN from every $\hat{a}$ assigned in the second stage of this simulation.

Num. Arc added	0	1	2	3	4	5	6	7	8	9
Out-of-Sample	0.464777	0.381158	0.33781	0.330319	0.333859	0.285402	0.286506	0.216204	0.198554	0.0506598
Num. Arc added	10	11	12	13	14	15	16	17	18	19
Out-of-Sample	0.204352	0.0532992	0.0756511	0.0539306	0.0300784	0.0304495	0.0546929	0.0240544	0.0251505	0.0301257
Num. Arc added	20	21	22	23	24	25	26	27	28	29
Out-of-Sample	0.0222016	0.0345123	0.0229365	0.0217007	0.0277345	0.0195861	0.0298685	0.0265805	0.0323089	0.0311179
Num. Arc added	30	31	32	33	34	35	36	37	38	39
Out-of-Sample	0.0350045	0.0201318	0.022037	0.0411805	0.0223831	0.0199765	0.0190606	0.0218507	0.021023	0.0187567

Table 4.3.3b Performance (measured as testing-sample RMSE) of topologically-optimised 5-10-1 networks with indicated number of arcs added

According to Table 4.3.2b, the testing-sample RMSE of the top performing 5-10-1 complete ANN is 0.0188145 after the training of up to 4000 iterations. Compare this with Table 4.3.3b, the out-of-sample performance of ERP optimised ANN is improved from the initial complete ANN when $\hat{a} = 59$, which has a standardised out-of-sample RMSE of 0.0187567, representing a small improvement of 0.31%. Table 4.3.3c is the ERP intermediate result from selecting the best in-sample performing ANN out of the 50 ANNs constructed by adding 39 arcs; the corresponding parameter of the final ANN after retraining is in recorded Table 4.3.3d.

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5
Neuron 1	0.197664	0.0385426	0.0939453	0.0585473	0.157778	0.165421
Neuron 2	2.17103	0.0502977	0.10233	0.0811365	0.234934	0.255555
Neuron 3	-1.63444	0.596404	-0.278238	0.0284927	0.0919592	-0.346744
Neuron 4	0.775261	-0.0715918	-0.0229444	-0.013542	0.475701	-0.0529839
Neuron 5	1.28857	0.468474	-0.210515	0.0202325	0.071339	-0.268478
Neuron 6	0.675667	0	0.943846	0.203166	0.395814	0.13755
Neuron 7	-1.66532	-0.0327849	-0.281185	0.651712	-0.287607	0.0701861
Neuron 8	0.802541	-0.0397389	0.447003	-0.00236289	-0.0392217	-0.103334
Neuron 9	0.788407	0.0285906	-0.0117311	-0.0252995	-0.0982598	0.446585
Neuron 10	1.36006	-0.0163542	-0.22312	0.478412	-0.225713	0.0582374

Hidden to output layer	Bias	Neuron 1	Neuron 2	Neuron 3	Neuron 4	Neuron 5	Neuron 6	Neuron 7	Neuron 8	Neuron 9	Neuron 10
Output vertex	7.87373	3.78169	7.65505	2.13229	-3.97619	-3.88677	-0.065674	1.9114	-2.80242	-4.22502	-3.82434

Table 4.3.3c Parameter vector of the selected ANN obtained after adding 39 arcs (pre-retraining)

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5					
Neuron 1	0.218093	0.0314945	0.0829806	0.0492129	0.143559	0.146167					
Neuron 2	2.0207	0.040161	0.0914836	0.065396	0.210328	0.220941					
Neuron 3	-1.55633	0.539046	-0.252159	0.0206251	0.0809547	-0.312221					
Neuron 4	0.748251	-0.0567616	-0.0195598	-0.00619662	0.430715	-0.046008					
Neuron 5	1.27489	0.437119	-0.19803	0.0155267	0.0628049	-0.249188					
Neuron 6	0.590192	0	0.759866	0.233843	0.335064	0.101587					
Neuron 7	-1.58512	-0.0211412	-0.26473	0.586639	-0.263714	0.0649551					
Neuron 8	0.774346	-0.0282234	0.402387	0.00869792	-0.0345434	-0.0920231					
Neuron 9	0.757358	0.0279468	-0.0069586	-0.0216032	-0.0776221	0.404296					
Neuron 10	1.333	-0.0105932	-0.21294	0.442563	-0.209924	0.0538297					
hidden to output layer	Bias	Neuron 1	Neuron 2	Neuron 3	Neuron 4	Neuron 5	Neuron 6	Neuron 7	Neuron 8	Neuron 9	Neuron 10
Output vertex	9.038	4.97524	8.82318	2.51404	-4.84453	-4.30947	-0.0714315	2.22326	-3.29003	-5.10089	-4.28055

Table 4.3.3d Final (post retraining) parameter vector of the optimised ANN

Table 4.3.3e summarises the results of ERP of optimising all six selected complete ANNs for approximating function two evaluated from the testing samples. It demonstrates that ERP again has produced a topologically-optimised ANN with an improved performance from each complete topologies listed in Table 4.3.2b. In addition, Figure 4.3.3b shows the graphical performance comparison of the complete ANNs and the corresponding topologically-optimised ANNs produced from ERP.

Performance rank of Complete ANN	Topology	Random Seed ID	Performance of Complete ANN	Num. of arcs in the Complete ANN	Performance of Topology optimised ANN	Num. of arcs in the Optimised ANN	Performance improvement
1	5-10-1	7	0.0188145	60	0.0187567	59	0.31%
2	5-10-1	8	0.02712	60	0.02414141	55	10.98%
3	5-9-1	1	0.02522	54	0.025048	44	0.66%
-	5-6-1	6	0.05537	36	0.03678132	25	33.57%
-	5-5-1	6	0.18361	30	0.18095668	24	1.44%
-	5-3-1	1	0.24082	18	0.23509169	17	2.38%

Table 4.3.3e Testing-sample results of ERP: topology and performance comparison for the selected networks

Last, the forecasting sample pairs are applied on the above topologies for evaluation of the out-of-sample performance. This result is shown in Table 4.2.3f and the bar chart is presented in Figure 4.2.3c.

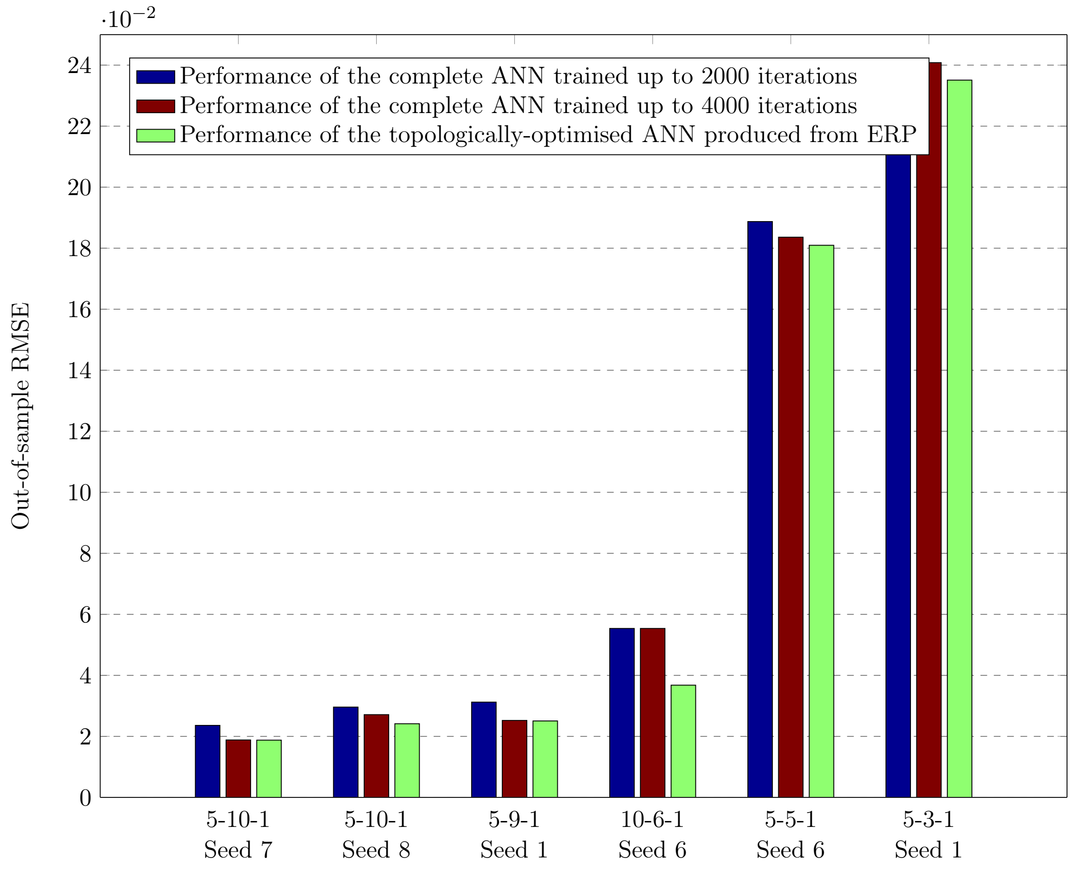


Figure 4.3.3b Graphical performance (testing-sample RMSE) comparison of the complete ANNs and the corresponding topologically-optimised ANNs produced from ERP

Performance rank of Complete ANN	Topology	Random Seed ID	Performance of Complete ANN	Num. of arcs in the Complete ANN	Performance of Topology optimised ANN	Num. of arcs in the Optimised ANN	Performance improvement
1	5-10-1	7	0.0158266	60	0.0153268	59	3.16%
2	5-10-1	8	0.03021	60	0.0257561	55	14.74%
3	5-9-1	1	0.028374	54	0.027329	44	3.68%
-	5-6-1	6	0.05834	36	0.04221	25	27.65%
-	5-5-1	6	0.16261	30	0.16046	24	1.34%
-	5-3-1	1	0.240245	18	0.238872	17	0.57%

Table 4.3.3f Forecasting-sample results of ERP: topology and performance comparison for the selected networks

4.3.4 Results of CERP

In the next simulation, we apply CERP to optimise the top performing 5-10-1 complete ANN listed in Table 4.3.2c. The same basic ANN produced from the systematic pruning process in

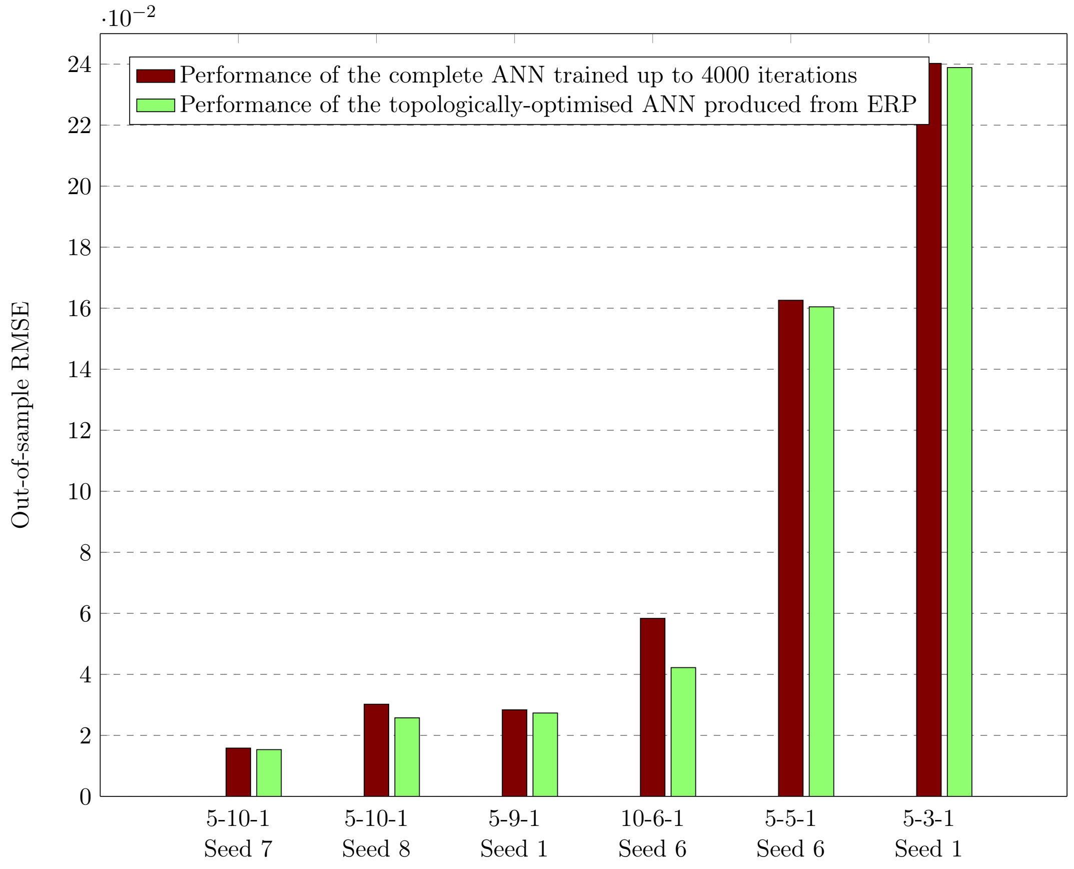


Figure 4.3.3c Graphical performance (forecasting-sample RMSE) comparison of the complete ANNs and the corresponding topologically-optimised ANNs produced from ERP

Table 4.3.3a is created and it is initialised to be the reference ANN. Described in Subsection 3.3.2 of Chapter 3, a single arc on the reference ANN is added on the basic ANN of the iteration to construct new ANNs; the new ANNs are then compared for selection of the reference ANN in the following iteration. Table 4.3.4a records the out-of-sample performance of the best ANN produced from each iteration until the process is terminated.

Figure 4.3.4a shows the graphical results of the testing-sample performance from the ANNs produced during iterations of CERP arc re-connection. For comparison, the figure has also shown the performance of initial complete ANN in Table 4.3.2b and the corresponding topologically-optimised ANN produced by ERP in Table 4.3.3e. Figure 4.3.4b is the zoomed in version of the Figure 4.3.4a to explicitly differentiate the final performance results between CERP and ERP.

Iteration Number	0	1	2	3	4	5
Testing Sample RMSE	0.464776	0.378892	0.324366	0.275250	0.198983	0.036399
Iteration Number	6	7	8	9	10	11
Testing Sample RMSE	0.026521	0.023733	0.022478	0.020504	0.019152	0.018343
Iteration Number	12	13	14	15	16	17
Testing Sample RMSE	0.017544	0.017030	0.016134	0.015791	0.015335	0.014933
Iteration Number	18	19	20	21	22	23
Testing Sample RMSE	0.014684	0.014583	0.014292	0.014167	0.014066	0.013959
Iteration Number	24	25	26			
Testing Sample RMSE	0.013920	0.013871	0.013840			

Table 4.3.4a Iteration number (number of added arcs) and the testing-sample RMSE from the CERP intermediate steps.

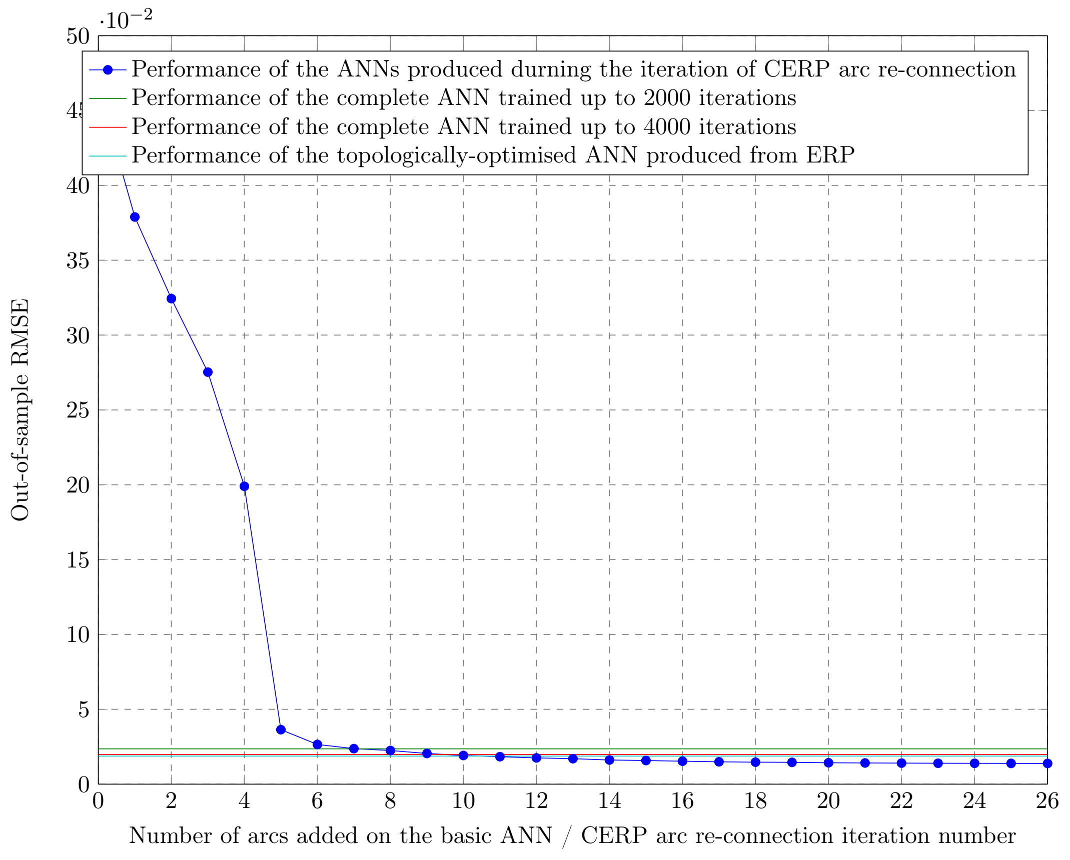


Figure 4.3.4a Performance (measured as testing-sample RMSE) comparison between the ANNs produced during the iteration of CERP, the complete ANN and the ERP optimised ANN

As it is indicated in Table 4.3.4a and Figure 4.3.4a, CERP terminates after adding a total of 26 arcs on the basic ANN, producing a topologically-optimised ANN with the parameter

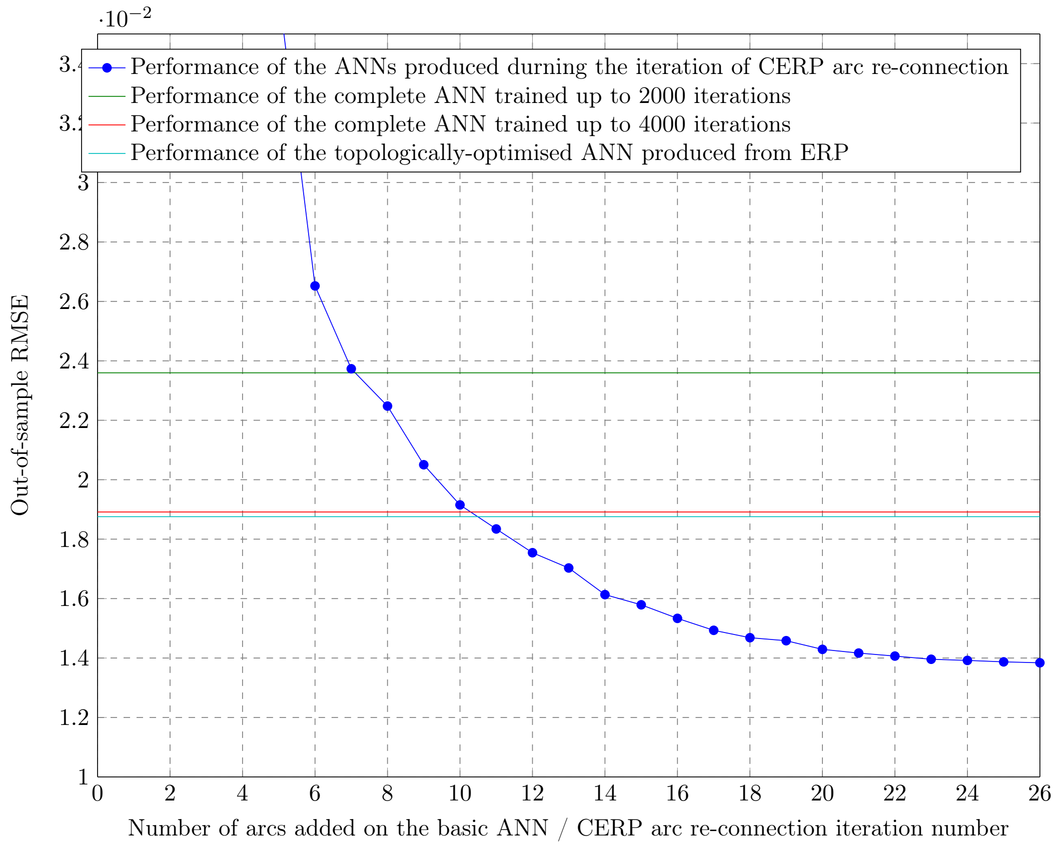


Figure 4.3.4b Zoomed in version of Figure 4.3.4a on the testing-sample RMSE scaled between 10^{-2} and 3.5×10^{-2}

vector display in Table 4.3.4b. Furthermore, we have showin in Figure 4.3.4a the significant performance improvement from the reference ANNs produced throughout the CERP arc re-connection iterations, which surpassed the performance of both the complete ANN and the ERP optimisied ANN after adding 10 arcs to the basic ANN.

To summarise the results of simulations on topology optimising the ANNs for approximating the function two, in Table 4.3.4c we list the performance and topology of the Complete ANN, the optimised ANN from ERP and the result from CERP. Comparing the perfromance difference between the complete ANN, the ERP optimised ANN and the CERP optimised ANN, we can conclude that for this specific approximation problem, while the topology produce by CERP performs significantly better than the output from ERP, both ERP and CERP have successfully produced topologically-optimised ANNs from the complete topology with an improved out-of-

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5
Neuron 1	-0.688142	0.0188687	-0.0018256	-0.0017033	-0.165142	0.177981
Neuron 2	4.6491	0.176328	0.246609	0.600174	-0.0171675	-0.0065245
Neuron 3	1.89584	-0.0465279	-0.112507	0	0	0
Neuron 4	0.728946	0	0	0.15284	-0.191379	0.00275895
Neuron 5	0.720748	0.237589	-0.126816	0	0.0156305	-0.0164924
Neuron 6	-0.718039	0.0741848	0.183718	0	0	0
Neuron 7	-0.709378	0	-0.124725	0.207736	-0.013124	-0.0012451
Neuron 8	0.797542	0.0252118	-0.793093	0.134183	-0.0064763	0
Neuron 9	0.692942	-0.172598	-0.0047923	0	-0.0059953	0.178266
Neuron 10	0.722869	0.01295	-0.551886	0.0777391	0	0

Hidden to output layer	Bias	Neuron 1	Neuron 2	Neuron 3	Neuron 4	Neuron 5	Neuron 6	Neuron 7	Neuron 8	Neuron 9	Neuron 10
Output	14.8875	15.1953	14.571	5.55913	-14.2376	-8.91718	-11.2244	12.1684	2.45861	-15.5168	-7.7474

Table 4.3.4b Parameter vector of topologically-optimised ANN produced by CERP based on the samples generated from Function 2

sample performance.

Topology	Standardised testing-sample RMSE	Standardised forecasting-sample RMSE	Number of connections	Testing-sample performance improvement	Forecasting-sample performance improvement
Complete ANN	0.0188145	0.0158266	60		
ERP optimised ANN	0.0187567	0.0153268	59	0.31%	3.16%
CERP optimised ANN	0.01384	0.0138546	41	26.4%	12.46%

Table 4.3.4c Performance improvement from ERP and CERP to the best complete ANN for approximating the function two.

4.4 Summary of result

Throughout this chapter, the empirical simulations are performed to test ERP and CERP on ANNs for approximating deterministic functions. Two non-trivial functions are designed to generate sample data and the samples are separated into training set, validation set, testing set and forecasting set for the simulation. On each of the approximation problems, a large number of ANN with different complete topologies are compared. The best 6 performing complete ANNs are then selected for topology optimisation using ERP and the top performing complete ANN is applied to be optimised by CERP.

For both of the two approximating problems, all topology-optimised ANNs produced from ERP have improved performance from the initial complete topology. CERP did not find a better performing topology than the complete ANN on approximating the function 1. However, the performance improvement throughout the iterative arc-addition process on the problem has provided an example of the combinatorial effect of adding multiple arcs simultaneously as discussed in the Chapter 3. Finally, The topology-optimised ANN produced from CERP outperforms the one produced from ERP for function 2, but this is clearly not generally (see next chapter).

In the coming chapter we will perform further empirical simulations on noisy data. With data obtained from financial market, the performance of ERP and CERP are tested on ANN for FFP.

Chapter 5

Empirical simulation on foreign exchange price forecasting

5.1 Introduction

We have demonstrated in Chapter 4 that the topology optimisation process can improve generalisation (out-of-sample) performance of ANNs on approximating problems for two non-trivial functions with input-target data pairs generated without the addition of sample noise. However, as discussed in Chapter 1, many practical ANN applications such as for the FFP, encounter input data that usually contain large amounts of noise. Therefore in this Chapter, we examine the application of the topology optimisation process on ANNs for the FFP using historical financial data obtained from the Foreign Exchange (FX) market.

The content of this Chapter is organised in the following way: Section 5.2 introduces the raw data obtained from the FX market and the construction of the three FFPs used throughout this Chapter for the simulations. Section 5.3 describes two designs of ANN systems using ERP and CERP respectively to be applied for the ANN training, optimisation and forecasting. Section 5.4 shows the performance result from the complete ANN for each FFP. From the complete ANN, topology optimisation is applied and Section 5.5 and Section 5.6 show the intermediate results as well as the final output produced using ERP and CERP respectively.

5.2 Data construction for the ANN forecasting problem

In order to construct a FFP for ANN, we need to first process the raw data to form input and target datasets for ANNs to be trained with. The raw data is a collection of direct quoted foreign exchange rates for the G10 currencies and Chinese Yuan to US dollar (excluding US dollar to itself). The G10 currencies are the 10 most influential and actively traded currencies in the world and include 1) US Dollar (USD); 2) Canadian Dollar (CAD); 3) Japanese Yen (JPY); 4) Australian Dollar (AUD); 5) New Zealand Dollar (NZD); 6) British Pound (GBP); 7) Euro (EUR); 8) Swiss Franc (CHF); 9) Swedish Krona (SEK); 10) Norwegian Krone (NOK). To limit the scope of the experiment, all FX rates are sampled at the daily New York trading closing time.

From the FX raw data, a single forecasting target is defined for the FFP in this research: suppose on day t, the 1 day holding period return on Euro to US dollar exchange price is $r_t^1(\texttt{EUR/USD})$, which can be calculated from the proportion of the price change from day $t-1$ to day t: $r_t^1(\texttt{EUR/USD}) = \frac{p_t(\texttt{EUR/USD}) - p_{t-1}(\texttt{EUR/USD})}{p_{t-1}(\texttt{EUR/USD})}$. This is also known as momentum in the word of technical analysis and it can be compared with the trends regressed from a price series to provide information on the sustainability of an assets price in a short future. In the case of this research, the target of forecasting problem is set to estimate the 5 day return of EUR/US FX rate on the future trading day $t+1$

Target: Eur/USD 5 Day Return on day $t+1$, denoted as: $r_{t+1}^1(\texttt{EUR/USD})$

To construct the appropriate inputs for the forecasting target defined above, there are many technical analysis tools that are commonly applied in market analysis and related the FFP which can be used to guide the construction of the dataset for ANN training. In particular, three technical analysis tools are most relevant and important to the forecasting problem on the FX market. These are Bollinger Bands, Long-Short Trend Cross-over and Absolute/Relative

Currency Strength.

Bollinger Bands, developed by John Bollinger [107], provide a relative measurement within a period of time for higher band and lower band of an asset price. The bands are useful in comparing price dynamic from time to time to support rigorous price pattern recognition and forms market indicators for trading decisions. Bollinger Bands are usually calculated from: 1) A 20 day period moving average of the asset price; 2) A 20 day period volatility of the asset price; The upper band is the sum of the moving average and a multiple of volatility; and the lower band is the moving average minus a multiple of volatility. Figure 5.2.0a shows an example of Bollinger Bands for the EUR/USD pair from Dec 2013 to May 2014, the bands consist of the 20 day period moving average and the 20 day period volatility multiplied by a factor of 0.2.

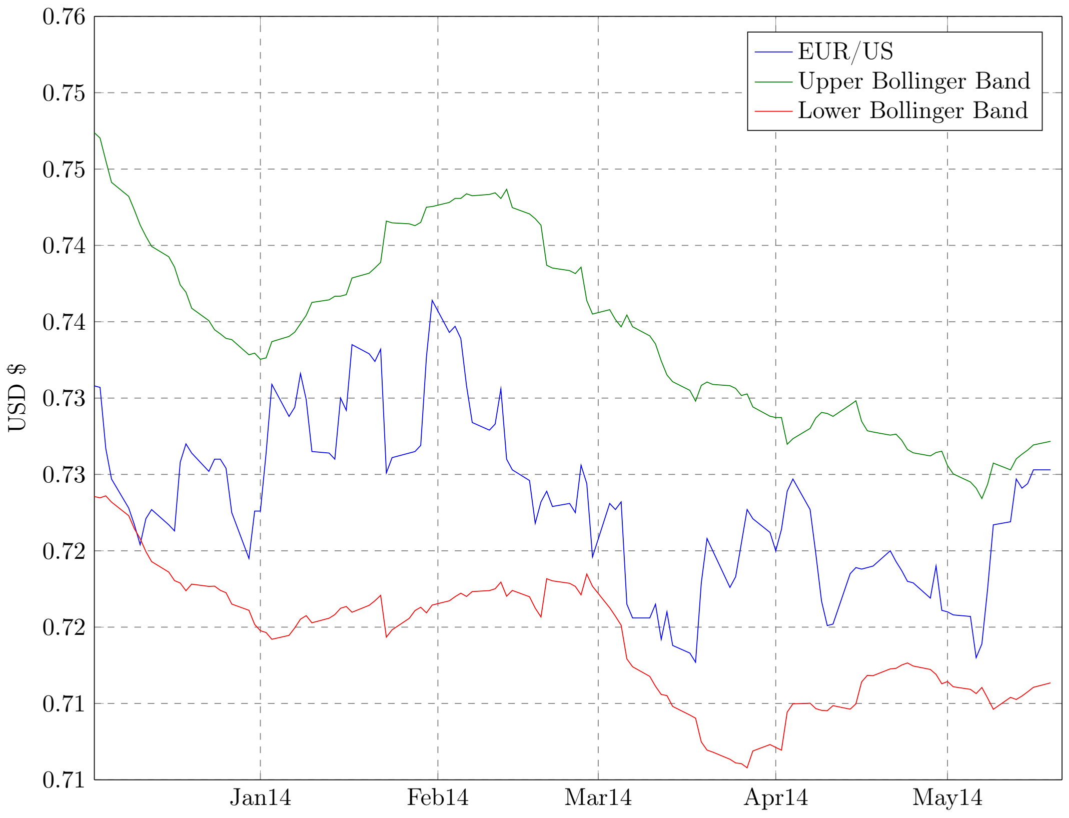


Figure 5.2.0a Illustration of Bollinger Bands for the EUR/USD price from Dec 2013 to May 2014

Long-Short Trend Cross-over, which is applied to signal the trigger point of buy/sale in machine trading, consists of two trends each based on different length of regression period from a time series. [1] As its name shows, the long trend is a regression gradient of an asset prices over a large window (e.g. 120 days); the short trend is from regression gradient over a small window (e.g. 20 days). While the short trend may be more reactive to daily price differences, the long trend encapsulates market trend over a longer time. Figure 5.2.0b shows a 6 month trend and a 1 month trend for the EUR/USD pair on the day of 19th May 2014

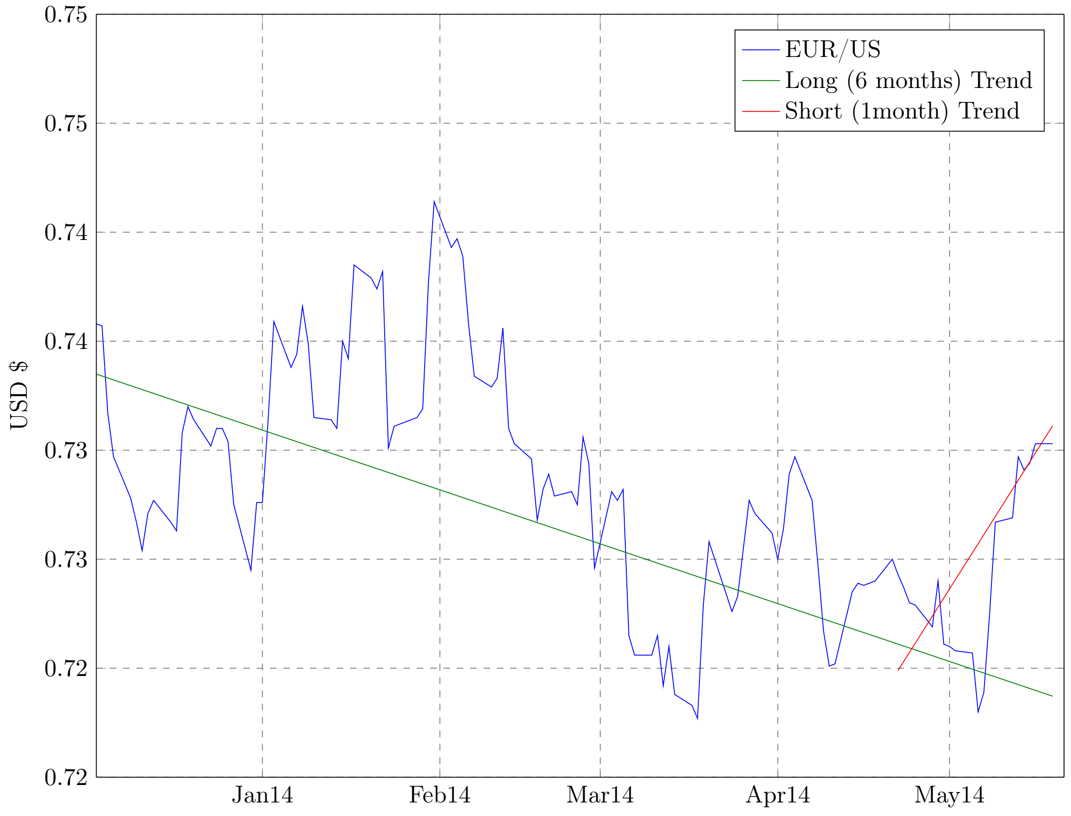


Figure 5.2.0b Illustration of a 6 month trend and a 1 month trend for the EUR/USD price on the day of 19th May 2014

Absolute/Relative Currency Strength are two technical analysis tools particularly used in the analysis of the foreign exchange market. The term absolute or relative only differ from the reference point in the calculations, otherwise they are the same. The basic idea is to show that if the exchange rate of a currency pair is up-trending, whether it is due to the strength of domestic currency or the weakness of foreign currency. For example, if the exchange rate of

[1]The trends in Long-Short Trend Cross-over model commonly known to be based on moving average. A trend based on linear regression, however, identifies the change of direction faster than the ones based on moving average given the same period of window [108].

Euro/US is rising, the currency strength indicator can tell if it is because of a strengthening Euro, or weakening US dollar or both.

Inspired from the elements of the above technical analysis tools, 10 inputs are fixed for the forecasting problems:

Input1: Eur/USD 1 Day Return on day t, denoted as: $r_t^1(\texttt{EUR/USD})$

Input2: Eur/USD 20 Day Return on day t: $r_t^{20}(\texttt{EUR/USD})$

Input3: Eur/USD Daily Return Volatility over past 20 Days (annualized) on day t: $\sigma_t^{20}(\texttt{EUR/USD})$

Input4: Eur/USD Daily Price 20 Day Trend on day t: $\beta_t^{20}(\texttt{EUR/USD})$

Input5: Eur/USD Daily Price 120 Day Trend on day t: $\beta_t^{120}(\texttt{EUR/USD})$

Input6: Average of the other 9 currencies' 1 Day Return on day t: $\overline{r_t^1(\texttt{EUR/USD})}$, where

$$\begin{aligned}\overline{r_t^1(\texttt{EUR/USD})} = \frac{1}{9}\Big[& r_t^1(\texttt{CAD/USD}) + r_t^1(\texttt{JPY/USD}) + r_t^1(\texttt{AUD/USD}) \\ &+ r_t^1(\texttt{NZD/USD}) + r_t^1(\texttt{GBP/USD}) + r_t^1(\texttt{CHF/USD}) \\ &+ r_t^1(\texttt{SEK/USD}) + r_t^1(\texttt{NOK/USD}) + r_t^1(\texttt{CNY/USD})\Big]\end{aligned}$$

Input7: Average of the other 9 currencies' 20 Day Return on day t: $\overline{r_t^{20}(\texttt{EUR/USD})}$

Input8: Average of the other 9 currencies' Daily Return Volatility over the past 20 Days (annualized) on day t: $\overline{\sigma_t^{20}(\texttt{EUR/USD})}$

Input9: Average of the other 9 currencies' Daily Price 20 Day Trend on day t: $\overline{\beta_t^{20}(\texttt{EUR/USD})}$

Input10: Average of the other 9 currencies' Daily Price 120 Day Trend on day t: $\overline{\beta_t^{120}(\texttt{EUR/USD})}$

These inputs are selected because:

1. Both input 1 and 2 are momentum indicators over the two time windows. In particular, input 1 is also the 1 day lagged result from the output and as discussed, the difference

between input 2 and 5 provide information on how long the FX rate may be sustainable at that level.

2. Together with input 3, 5 and a constant parameter, the Bollinger Bands analysis can be applied.

3. Input 4 and 5 can be used to form Long-Short Trend Cross-over indicator.

4. The relative strength of the foreign currency in a FX pair can be found through the change in averaging the exchange price of all other than domestic currencies in the world to the foreign currency. In the case of this research, the relative strength of US dollar is found through averaging the technical indicators of the rest world major currencies, which lead to input 6-10.

Following the discussion above, the input-target data pairs for ANN forecasting are processed from the raw data obtained from The Wall Street Journal for the period between Jan 1998 to May 2014. The daily FX rate for currency pair has more than 4000 samples in the raw data and this number is slightly reduced by the end-effect of the above period.

In the simulations of this chapter, data for training ANN in the FFP is divided into four sets: Training, Validation, Testing and Forecasting. This is different to the simulations in the previous chapter which use only three sets of data: Training, Validation, and Testing. In the simulation with the datasets generated from the deterministic functions, the Testing dataset is subject to the topology optimisation process and is also the only out-of-sample data for the performance evaluation. In the current simulation for the FFP with datasets obtained from the market, however, the testing dataset is used for ERP or CERP to optimise the graph of ANN; and the forecasting dataset is to be used solely for performance evaluation and comparison.

For the purpose of this research, the size of both in and out-of-sample datasets are large enough for the results to be statistically representative. A procedure we applied to setup the four datasets is: We use 1000 input-target sample pairs ordered on date from day 1 to 1000, which covers approximately a 4 year horizon, the training set and validation set are randomly allocated from the first 700 samples in proportion of 80:20 with 560 samples for training and

140 samples for validation; then the next 100 samples from day 701 to 800 is allocated for testing; and the last 200 samples from day 801 to 1000 is for forecasting.

Guided from the above discussion, three forecasting problems can be setup by partitioning the input-target sample pairs produced in Section 5.2:

1. The first problem covers 1000 samples dated from 1st July 2010 to 30th April 2014

   - 700 samples from 1st July 2010 to 6th March 2013 for training and validation, in which 560 samples are randomly drawn for training and the rest 140 for validation;
   - 100 samples from 7th March 2013 to 24th July 2013 for testing;
   - 200 samples from 25th July 2013 to 30th April 2014 for forecasting;

   Note that the target samples for testing and forecasting is continuously in time, as depicted in Figure 5.2.0c, the timeseries of $r^1_{t+1}(\texttt{EUR/USD})$ is shown between 7th March 2013 to 30th April 2014. The first 100 sample in the shaded area are for the testing and the rest are for the forecasting.

2. The second problem covers 1000 samples dated from 1st September 2006 to 01st July 2010

   - 700 samples from 1st September 2006 to 7th May 2009 for training and validation;
   - 100 samples from 8th May 2009 to 24th September 2009 for testing;
   - 200 samples from 25th September 2009 to 1st July 2010 for forecasting;

   Figure 5.2.0d shows the time series of $r^1_{t+1}(\texttt{EUR/USD})$ between 7th April 2009 to 31st May 2010 for the problem 2. The shaded area includes the testing target samples only.

3. The third problem covers 1000 samples dated from 1st October 2002 to 31st July 2006

   - 700 samples from 1st October 2002 to 6th June 2005 for training and validation;
   - 100 samples from 7th June 2005 to 24th October 2005 for testing;
   - 200 samples from 25th October 2005 to 31st July 2006 for forecasting;

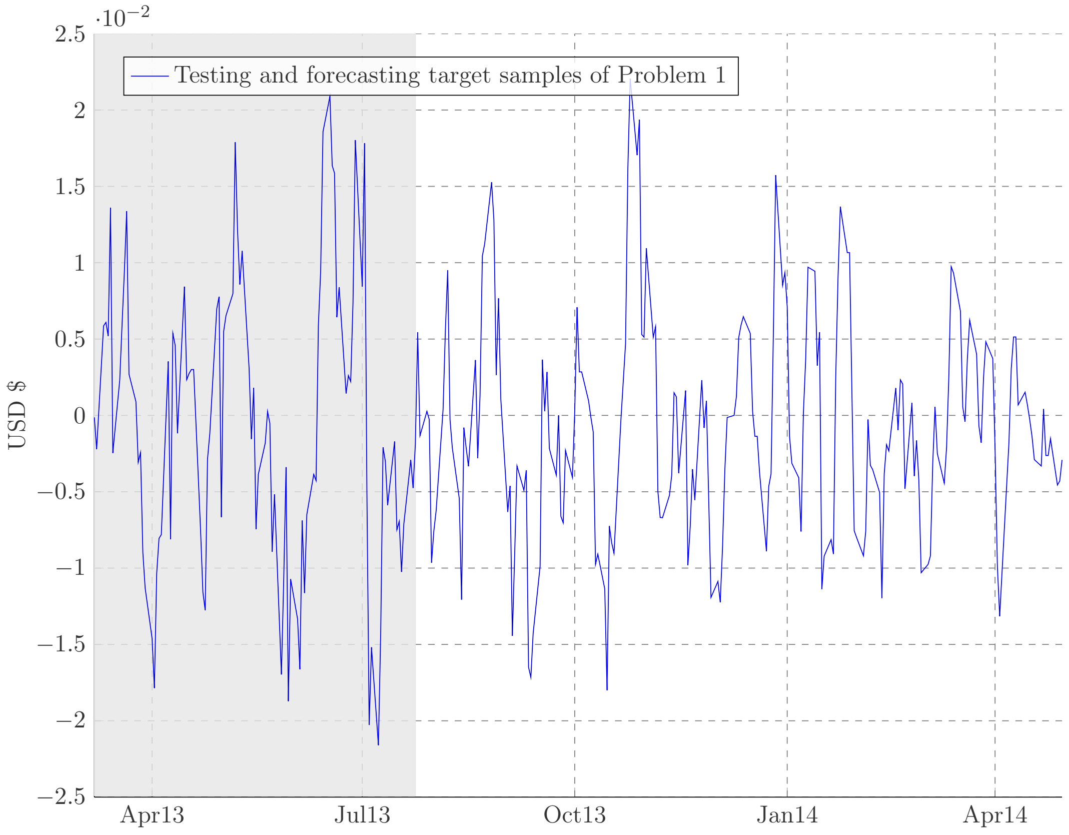


Figure 5.2.0c $r^1_{t+1}(\texttt{EUR/USD})$ between 7th March 2013 to 30th April 2014

Figure 5.2.0e shows the timeseries of $r^1_{t+1}(\texttt{EUR/USD})$ between 7th June 2005 to 31st July 2006 for the problem 3. The shaded area includes the testing target samples only.

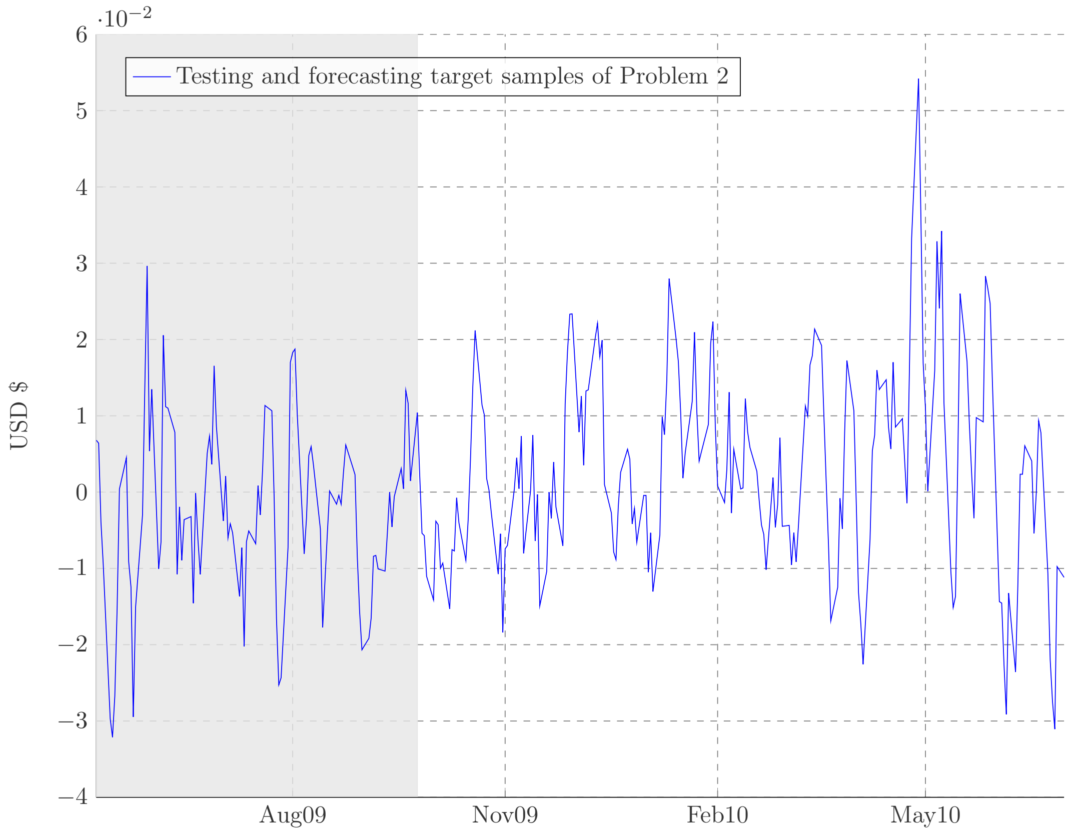


Figure 5.2.0d $r^1_{t+1}(\texttt{EUR/USD})$ between 7th April 2009 to 31st May 2010

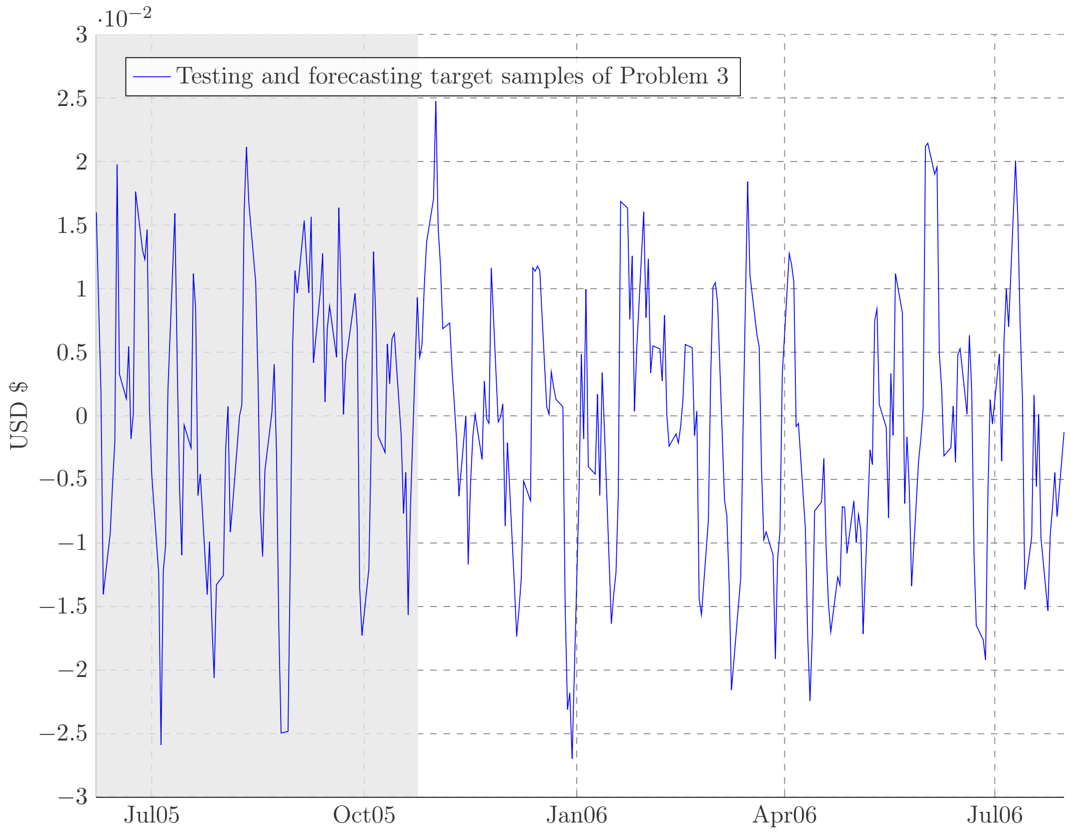


Figure 5.2.0e r^1_{t+1}(`EUR/USD`) between 7th June 2005 to 31st July 2006

5.3 Design and setting of the ANN forecasting System

Once the data structure for the forecasting problems is fixed, a system of simulation including complete ANN training and selection, topology optimisation and forecasting can be constructed. The flow chart in Figure 5.3.0 is an overview of the system assembled from implementations which has been introduced in the previous chapters. The detailed configuration of the system is described in the steps following the figure.

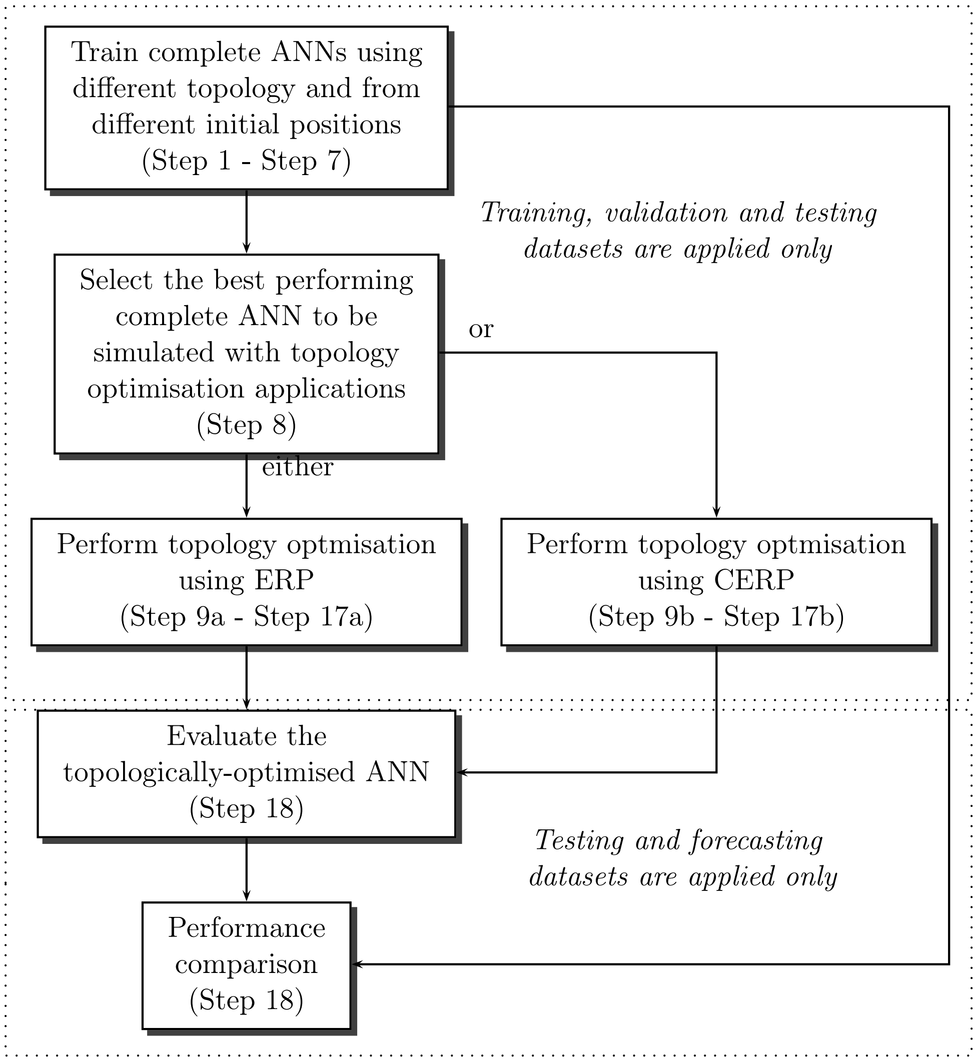

Train complete ANNs using different topology and from different initial positions (Step 1 - Step 7)
Training, validation and testing datasets are applied only
Select the best performing complete ANN to be simulated with topology optimisation applications (Step 8)
or
either
Perform topology optmisation using ERP (Step 9a - Step 17a)
Perform topology optmisation using CERP (Step 9b - Step 17b)
Evaluate the topologically-optimised ANN (Step 18)
Testing and forecasting datasets are applied only
Performance comparison (Step 18)

Figure 5.3.0 Flow chart representation of ANN forecasting system for the FFP of FX data

Step 1: Given the datasets of an FX forecasting problem, construct a complete ANN denoted by $G_{10,\hat{j},1}$, so the topology has 10 inputs, 1 output and $\hat{j}$ number of hidden neurons which is initialised to $\hat{j} = 2$.

Step 2: Initialise the parameter vector of the complete ANN with a random number generator using seed ID $S = 1$;

Step 3: Train the complete ANN, measure and store the performance measured from the testing samples;

Step 4: Increase the SeedID S by 1;

Step 5: If $S \leq 20$, Goto Step 3, so that 20 initial search positions are applied on each complete ANN topology ;

Step 6: Increase the number of hidden neuron $\hat{j}$ by 1;

Step 7: If $\hat{j} \leq 12$, Goto Step 2, so a total of 11 ANN complete topology are tested

Step 8: The trained complete ANN with the best performance in record is selected to proceed, denote as $G_{10,\hat{j}',1}$;

$- - - - -$ Start of ERP sub-Routine $- - - - -$

Step 9a: Apply the basic ANN construction stage of ERP on the selected $G_{10,\hat{j}',1}$ to construct the basic ANN, denoted as H;

Step 10a: Initalise the number of arcs to be added is 0 so that $\hat{a} - \left\{\max\left(10, \hat{j}'\right) + \hat{j}'\right\} = 0$ and the trial ID $n = 1$

Step 11a: Randomly adding the number of arcs to the basic ANN so a total of $\hat{a}$ arcs in the ANN; test and record the performance of the network from the training set (in-sample) data;

Step 12a: Increase the trial ID $n = n + 1$;

Step 13a: If $n \leq 200$ trials, go to Step 11a so that the arcs are added randomly for 200 times

Step 14a: Evaluate and select the best performing network found based on the training samples, retrain the ANN from the existing parameter and record the result as the candidate output ANN of ERP;

Step 15a: Increase the number of arcs to be added by 1;

Step 16a: Repeat from Step 10a until $\hat{a}$ reaches the maximum.

Step 17a: Selected the $\hat{a}$ which specifies the best performing candidate output ANN on the testing set, which is the topologically-optimised ANN, denote as $G_{\hat{a}}$.

– – – – – End of ERP sub-Routine – – – – –

– – – – – Start of CERP sub-Routine – – – – –

Step 9b: Apply the basic ANN construction stage of CERP on $G_{10,\hat{j}',1}$ to construct the basic ANN, denoted as H and set $G_{temp} = H$;

Step 10b: Train H and evaluate the out-of-sample performance

Step 11b: Initialise the trial ID to $n = 1$

Step 12b: For the nth element with a value of 0 in the parameter vector of G_{temp}, where the arc (v_i, v_j) under indication is effectively disconnected, reconnect the indicated arc to form $G_{temp} \cup (v_i, v_j)$; train and test the result network to $G'_{temp} \cup (v_i, v_j)$ and record the testing sample (out-of-sample) performance;

Step 13b: Increase the trial ID $n = n + 1$;

Step 14b: Repeat from Step 12b until the last element with 0 value in the parameter vector of G_{temp} has been indicated;

Step 15b: Select the best performing ANN $G'_{temp} \cup (v'_i, v'_j)$ in the record, if the out-of-sample performance is better than the G_{basic} go to Step 16b, otherwise go to Step 17b;

Step 16b: Assign the best performing network to G_{temp}, go to Step 11b;

Step 17b: Assign the selected network to $G_{\hat{a}}$.

– – – – – End of CERP sub-Routine – – – – –

Step 18: Apply the topologically-optimised ANN $G_{\hat{a}}$ on the forecasting dataset and compare to the forecasting performance of complete ANN $G_{10,\hat{j}',1}$ using the value of standardised RMSE between the output values and the target samples.

Similar to the previous simulations discussed in Subsection 4.2.1 of Chapter 4, there are two conditions governing the training and learning process of the ANN need to be set. For the simulation of the FFP, the maximum number of training iterations is set to 2000 and the stopping limit of gradient convergence tolerance is set to 10^{-4}.

5.4 Forecasting performance of complete ANN

This section is divided into three subsections each one presenting the results of the best complete ANN obtained on a dataset constructed in Section 5.2. The complete ANN is constructed following Step 1 to Step 8 of the simulation systems designed in Section 5.3.

Note that in this chapter, the performance measurement is shown in the form of standardised value of RMSE between the ANN forecasted result and the sample target. Introduced in Chapter 4: the RMSE obtained from the approximation model for a particular sample set is standardised by dividing the RMSE with the average of the target sample absolute values. From the standardised RMSE, it is clearer to interpret the performance of a system across different data sets with different size and value of samples.

In addition to the standardised RMSE, some other measurements are taken for supplemental studies on the performance. These include the number of correct estimation on the directional sign of the target sample as well as the percentage of RMSE improvement. Note that all these supplemental measurement are evaluated only for a relative comparison between different topologies, i.e. complete ANN, ERP optimised ANN and CERP optimised ANN constructed during the simulations.

5.4.1 Forecasting Problem 1

From the 560 samples recorded in Section 5.2 for training and the 140 samples for validation constructed for Problem 1, a complete ANN $G_{10,2,1}$ is produced. The complete network consists of $\hat{j}' = 2$ hidden neurons and is trained with initial parameters generated from $seedID = 10$; The training condition is met and the iteration is terminated after 17 training cycles. The parameter vector of this complete ANN $G_{10,2,1}$ is displayed in Table 5.4.1a:

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10
Neuron 1	0.718879	0.575939	0.340596	0.164531	0.117442	0.254044	0.957463	0.218548	0.130085	0.916669	0.333643
Neuron 2	0.291268	0.619459	0.0222096	0.703529	0.999962	0.756335	0.0305982	0.329984	0.0859183	0.542532	0.329601
Hidden to output layer	Bias	Neuron 1	Neuron 2								
Output	-0.00157161	0.112988	-0.125494								

Table 5.4.1a Parameters of the 10-11-1 complete ANN for Problem 1

Feed the 100 testing samples on the above complete ANN $G_{10,2,1}$ produces results with performance indicators recorded on the upper row of Table 5.4.1b; Apply the same network on the 200 forecasting samples producing results with performance indicators recorded in the lower row of the table:

Testing Sample RMSE	Directional correctness	Number of Samples
1.20372	56	100
Forecasting Sample RMSE	Directional correctness	Number of Samples
1.33289	83	200

Table 5.4.1b Performance indicators of $G_{10,2,1}$ on the testing samples and forecasting sample for Problem 1

5.4.2 Forecasting Problem 2

Similarly, for the training and validation sample set of Problem 2, the resulting complete ANN with topology $G_{10,2,1}$ has $\hat{j}' = 8$ on the hidden layer and is trained with $seedID = 8$ for 11 iterations before the training conditions are met. Table 5.4.2a shows the parameter vector of the $G_{10,2,1}$ and Table 5.4.2b is its corresponding results with performance indicators on the testing sample sets and the forecasting sample sets.

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10
Neuron 1	0.217297	0.533894	-0.0363056	0.20332	0.977062	0.772806	0.712451	0.951504	0.583963	0.162424	0.767263
Neuron 2	0.642077	0.482657	0.194613	0.933769	0.167704	0.482409	0.938685	0.209894	0.619177	0.359694	0.186694
Hidden to output layer	Bias	Neuron 1	Neuron 2								
Output neuron	0.0292679	0.171265	-0.0204738								

Table 5.4.2a Complete ANN $G_{10,2,1}$ for Problem 2

Testing Sample RMSE	Directional correctness	Number of Samples
1.25454	56	100
Forecasting Sample RMSE	**Directional correctness**	**Number of Samples**
1.26749	110	200

Table 5.4.2b Performance result of $G_{10,2,1}$ for Problem 2 on the testing samples

5.4.3 Forecasting Problem 3

Table 5.4.3a is the parameter vector of the complete ANN produced following Step 1 to Step 8 on the training and validation sample set of Problem 3. The complete ANN $G_{10,7,1}$ has $\hat{j}' = 7$ and is trained from $seedID = 3$ after 23 training iterations. The corresponding results with performance indicators on the testing sample is listed in the upper row of Table 5.4.3b and the results on forecasting sample is in the lower row of the table.

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10
Neuron 1	0.190398	0.443337	0.582247	0.579185	0.146256	0.211177	0.385638	0.569971	0.48278	1.01295	0.0328951
Neuron 2	0.878388	0.65805	0.607833	0.881124	0.489164	0.185209	0.539137	0.338801	0.881392	0.246838	0.0965114
Neuron 3	-0.242191	0.136376	1.03228	1.21114	1.03644	0.481833	1.11992	0.487381	-0.00875052	0.424771	0.418579
Neuron 4	0.704599	0.960257	0.864852	0.345339	0.49042	0.0894278	0.121483	0.0432097	0.118666	0.916029	0.600254
Neuron 5	0.227071	1.06085	0.719177	0.161046	0.509035	0.722609	0.956368	0.550305	0.457381	0.0665518	0.434911
Neuron 6	0.426958	0.889725	0.482792	0.0696612	0.588499	0.454945	0.753524	0.254704	0.260308	0.743121	0.668761
Neuron 7	0.687403	0.219279	0.713396	0.577511	0.124302	1.05727	0.280918	0.224055	0.861022	0.621636	0.166156
Hidden to output layer	Bias	Neuron 1	Neuron 2	Neuron 3	Neuron 4	Neuron 5	Neuron 6	Neuron 7			
Output neuron	-0.0885371	0.536204	0.387812	-0.809672	-0.113546	-0.00443031	0.0380259	0.110771			

Table 5.4.3a Parameter of the best trained complete ANN $G_{10,2,1}$ for Problem 3

Testing Smaple RMSE	Directional correctness	Number of Samples
1.18548	51	100
Forecasting Sample RMSE	Directional correctness	Number of Samples
1.24101	107	200

Table 5.4.3b Performance result of $G_{10,7,1}$ for Problem 3 on the testing samples

5.5 ERP results

Continue with the topology optimisation system described in Section 5.2 to optimise the above complete ANNs, this section records the results produced by taking the ERP subroutine (Step 9a to 17a) . Each subsection presents the outputs from a problem, including the basic ANN constructed in Step 9 and the final optimised ANN in Step 17 with the performance indicators.

5.5.1 Forecasting Problem 1

Continue the steps applying ERP on the complete ANN $G_{10,2,1}$ listed in Table 5.4.1a, the basic ANN with minimal network G_{basic} is produced from Step 9. Table 5.5.1a contains the parameter vector of G_{basic}.

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10
Neuron 1	0.718879	0	0.340596	0	0	0	0.957463	0	0.130085	0.916669	0
Neuron 2	0.291268	0.619459	0	0.703529	0.999962	0.756335	0	0.329984	0	0	0.329601

Hidden to output layer	Bias	Neuron 1	Neuron 2
Output	-0.00157161	0.112988	-0.125494

Table 5.5.1a Parameters of the basic ANN G_{basic} for Problem 1

Proceed from Step 10 to Step 17 to construct numbers of topology simplified ANN, the best performing one is selected to be the topologically-optimised solution $G_{\hat{a}}$ from the system. For the problem 1,$G_{\hat{a}}$ with parameter vector listed in Table 5.5.1b is produced by adding $\hat{a} = 3$.

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10
Neuron 1	0.717228	0	0.34413	0.174587	0	0	0.958394	0.222858	0.144583	0.91059	0
Neuron 2	0.297778	0.621271	0	0.689794	1.00436	0.755906	0	0.326906	0	0.550684	0.326861

Hidden to output layer	Bias	Neuron 1	Neuron 2
Output	-0.0387877	0.0949776	-0.0806881

Table 5.5.1b Parameters of the topology optimised ANN $G_{\hat{a}}$ for Problem 1

The testing-sample performance of the selected $G_{\hat{a}}$ is recorded in the top row of Table 5.5.1c. This result outperforms other candidate simplified ANNs constructed from the different number of arcs added. Finally, in Step 18 of the forecasting system, the topologically-optimised ANN listed above is applied to the performance evaluation on the forecasting samples . The bottom row of Table 5.5.1c shows the result from the topologically-optimised ANN.

Testing Sample RMSE	Directional correctness	Number of Samples
1.20107	57	100
Forecasting Sample RMSE	**Directional correctness**	**Number of Samples**
1.28416	104	200

Table 5.5.1c Performance result of $G_{\hat{a}}$ from ERP for Problem 1 on the testing samples (top) and forecasting samples (bottom)

While the ERP topology optimisied ANN makes a slightly 0.22% improvement on the testing samples compared to results from the complete ANN, the performance difference on the forecasting samples is improved by 3.66% comparing the standardised out-of-sample RMSE in Table 5.4.1b and Table 5.5.1c. As a result, the forecasting performance is generally improved over the 200 samples with the topologically-optimised ANN.

5.5.2 Forecasting Problem 2

From Step 9, the parameter vector of the basic ANN $G_{10,2,1}$ obtained by ERP is displayed in the Table 5.5.2a.

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10
Neuron 1	0.217297	0.533894	0	0.20332	0.977062	0.772806	0.712451	0.951504	0.583963	0.162424	0.767263
Neuron 2	0.642077	0	0.194613	0	0	0	0	0	0	0	0

Hidden to output layer	Bias	Neuron 1	Neuron 2
Output neuron	0.0292679	0.171265	-0.0204738

Table 5.5.2a Basic ANN G_{basic} for Problem 2

Then from Step 10 to Step 17, the topologically-optimised ANN $G_{\hat{a}}$ is found with $\hat{a} = 9$. The $G_{\hat{a}}$ is with the parameter vector in the Table 5.5.2b and the testing sample performance results in the top row of Table 5.5.2c

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10
Neuron 1	0.112558	0.520133	-0.157704	0.278555	1.09974	0.645556	0.697776	0.916615	0.65221	0.177816	0.717335
Neuron 2	0.643738	0.457252	0.198654	0	0.162332	0.505124	0.944115	0.218986	0.595817	0.336425	0.248246

Hidden to output layer	Bias	Neuron 1	Neuron 2
Output neuron	0.136787	0.498721	-0.40633

Table 5.5.2b Topology optimised ANN $G_{\hat{a}}$ for Problem 2

In Step 18, the performance of the topologically-optimised ANN $G_{\hat{a}}$ on the forecasting samples for the Problem 2 is evaluated, which is recorded in the bottom row of Table 5.5.2c.

Testing Sample RMSE	Directional correctness	Number of Samples
1.18774	49	100
Forecasting Sample RMSE	**Directional correctness**	**Number of Samples**
1.28228	111	200

Table 5.5.2c Performance result of $G_{\hat{a}}$ for Problem 2 on the testing samples (top) and on the forecasting samples (bottom)

Comparing the staderdised out-of-sample RMSE between the output of the complete ANN, the ERP topologically-optimised ANN and target samples in Table 5.4.2b and Table 5.5.2c, we notice the big improvement on the performance of topologically-optimised ANN when applied to the testing data, which is 5.33%. However, the performance is not improved when the ERP optimised ANN is applied to the forecasting data.

5.5.3 Forecasting Problem 3

Subsequently the basic ANN G_T with parameter vector in Table 5.5.3a is produced from Step 9.

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10
Neuron 1	0.190398	0	0	0	0	0	0	0	0	1.01295	0
Neuron 2	0.878388	0	0	0	0	0	0	0	0.881392	0	0
Neuron 3	-0.242191	0	1.03228	1.21114	1.03644	0	1.11992	0	0	0	0
Neuron 4	0.704599	0.960257	0	0	0	0	0	0	0	0	0
Neuron 5	0.227071	0	0	0	0	0	0	0.550305	0	0	0
Neuron 6	0.426958	0	0	0	0	0	0	0	0	0	0.668761
Neuron 7	0.687403	0	0	0	0	1.05727	0	0	0	0	0
Hidden to output layer	Bias	Neuron 1	Neuron 2	Neuron 3	Neuron 4	Neuron 5	Neuron 6	Neuron 7			
Output neuron	-0.0885371	0.536204	0.387812	-0.809672	-0.113546	-0.00443031	0.0380259	0.110771			

Table 5.5.3a Parameters of the basic ANN G_{basic} for Problem 3

Next from Step 10 to Step 17 of the system, the topologically-optimised ANN $G_{\hat{a}}$ is found for $\hat{a} = 46$. The optimised network has parameter vector listed in Table 5.5.3b and its corresponding testing sample performance indicators is recorded in the top row of Table 5.5.3c.

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10
Neuron 1	0.0754125	0.490469	0.349621	0.454131	0.294076	0.303452	0.376267	0.411164	0.528946	1.25533	-0.166286
Neuron 2	0.994978	0.508483	0.662756	0.98752	0.475662	0	0.545469	0.661029	0.980859	0.529205	0.139557
Neuron 3	-0.50002	-0.0672461	0.922691	1.15628	1.15691	0.0937933	1.00352	0.601595	-0.035826	0.332981	0.535932
Neuron 4	0.766139	0.984672	0.945408	0.487787	0	0.10434	-0.0303194	-0.0929022	0.20281	0.793751	0
Neuron 5	0.221132	1.06622	0.710441	0	0.517782	0.72652	0	0.551614	0.466201	0.0830251	0
Neuron 6	0.331632	0.988325	0	0	0	0.457856	0	0.290079	0.2343	0.81104	0.655909
Neuron 7	0.635789	0	0.789786	0.480149	0.224333	1.17285	0	0.333422	0.861482	0	0
Hidden to output layer	Bias	Neuron 1	Neuron 2	Neuron 3	Neuron 4	Neuron 5	Neuron 6	Neuron 7			
Output neuron	-0.121599	0.62743	0.539339	-0.966876	-0.604769	0.0118163	0.334202	0.211392			

Table 5.5.3b Parameters of the topology optimised ANN $G_{\hat{a}}$ for Problem 3

Last in Step 18, the performance of $G_{\hat{a}}$ on the forecasting samples is calculated and shown in the bottom row of the Table 5.5.3c.

The improvement of the ERP topologically-optimised ANN from the complete ANN, according to the standardised out-of-sample RMSE in Table 5.4.3b and Table 5.5.3c, is approximately 0.83% on the testing data and 0.5% on the forecasting data.

Testing Sample RMSE	Directional correctness	Number of Samples
1.17569	51	100
Forecasting Sample RMSE	**Directional correctness**	**Number of Samples**
1.2348	112	200

Table 5.5.3c Performance result of $G_{\hat{a}}$ for problem 3 on the testing samples (top row) and the forecasting samples

As the performance results indicated in Table 5.5.1c for problem 1, Table 5.5.2c for problem 2 and Table 5.5.3c for problem 3, in all three simulations the ERP topologically-optimised ANNs have outperformed the complete ANN. These empirical results conclude that ERP can improve the generalisation performance of a complete ANN by constructing the topologically-optimised ANN with reduced number of aces between the input and the hidden layer.

5.6 CERP results

This section records the output produced by taking the CERP sub-routine (Step 9a to 17a) of the topology optimisation system to optimise the complete ANNs in Section 5.3. Each subsection presents the final optimised topology in Step 17 with the performance indicators. Recall that both ERP and CERP construct a unique basic ANN from the samples so it is not shown repetitively in the following subsections.

5.6.1 Forecasting Problem 1

Applying CERP on the complete ANN listed in Subsection 5.4.1 for Problem 1, the process first constructs the basic ANN as displayed in Table 5.5.1a. The subsequent arc re-connection follows the loop described in Step 15b of the system takes only 1 iteration to reach a solution. Table 5.6.1a shows the testing sample performance along with the iteration number:

Iteration Number	0	1
Testing Sample RMSE	0.930974	0.908024

Table 5.6.1a Iteration number to the performance on testing samples from the corresponding topology produced during the loop of CERP.

The configuration of CERP subroutine in the system has set that for each iteration, the CERP adds 1 arc on the optimal ANN chosen from the previous iteration. The loop of CERP terminates after 1 iteration means the result ANN has 1 arc added onto the basic ANN. The parameter vector of the result is shown in Table 5.6.1b and the performance indicators on testing sample is in Table 5.6.1c

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10
Neuron 1	0.717715	0	0.345766	0	0	0	0.959179	0	0.146889	0.909022	0
Neuron 2	0.298289	0.620139	0	0.691102	1.00907	0.753851	0	0.325043	0	0	0.321553
Hidden to output layer	Bias	Neuron 1	Neuron 2								
Output neuron	-0.0493728	0.0557513	-0.0314627								

Table 5.6.1b Parameter vector of $G_{\hat{a}}$ from CERP for Problem 1

The bottom row of Table 5.6.1c shows the performance results from applying the $G_{\hat{a}}$ on the forecasting sample which is to be compared with performance results from the complete ANN $G_{10,2,1}$ in Table 5.4.3

standardised Testing Sample RMSE	Directional correctness	Number of Samples
1.20133	59	100
Forecasting Sample RMSE	Directional correctness	Number of Samples
1.27887	105	200

Table 5.6.1c Performance of $G_{\hat{a}}$ produced from CERP on the testing samples (top row) and the forecasting samples (bottom row) for Problem 1

Finally, we can meausre the performce difference of the complete ANN and the CERP optimised ANN from the value of standardised out-of-sample RMSE. For the CERP optimised ANN, the performance is improved 0.20% on the testing data and 4.05% on the forecasting data.

5.6.2 Forecasting Problem 2

From the sample training sets of problem 2, the basic ANN described in Table 5.5.2a is built, from which the CERP takes only one iteration before the arc re-connection iteration is terminated, as it is recorded in Table 5.6.2a:

Iteration Number	0	1
Testing Sample RMSE	1.256835428	1.21739

Table 5.6.2a Iteration number to the performance on testing samples from CERP intermediate steps

The corresponding parameter vector of the result $G_{\hat{a}}$ from CERP is recorded in Table 5.6.2b

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10
Neuron 1	0.189768	0.528682	0	0.223925	1.00127	0.752253	0.705395	0.946737	0.602595	0.162614	0.759973
Neuron 2	0.641108	0	0.206542	0	0	0	0.941407	0	0	0	0
Hidden to output layer	Bias	Neuron 1	Neuron 2								
Output neuron	0.0472913	0.209364	-0.217389								

Table 5.6.2b Parameter vector of $G_{\hat{a}}$ from CERP for Problem 2

From the $G_{\hat{a}}$, the performance on the testing sample indicators are listed in the top row of Table 5.6.1c; the performance on the forecasting samples are listed in the bottom tow of Table 5.6.2c

standardised Testing Smaple RMSE	Directional correctness	Number of Samples
1.21739	56	100
Forecasting Sample RMSE	Directional correctness	Number of Samples
1.28969	110	200

Table 5.6.2c Performance of $G_{\hat{a}}$ produced from CERP on the testing samples (top row) and the forecasting samples (bottom row) for Problem 2

According to the change between the standardised out-of-sample RMSE listed in Table 5.4.2b and Table 5.6.2c, the performance of the CERP optimised ANN on the testing data is significantly improved by 2.96% compared as the complete ANN but no improvement on the forecasting data in terms of RMSE.

5.6.3 Forecasting Problem 3

For the basic ANN constructed from datasets of Problem 3, CERP takes 7 iterations to produce the optimised ANN. The detail with iteration are shown in Table 5.6.3a

Iteration Number	0	1	2	3	4	5	6	7
Testing Sample RMSE	1.476095	1.377691	1.296116	1.274964	1.253709	1.219632	1.219012	1.20646

Table 5.6.3a Iteration number to the performance on testing samples from CERP intermediate steps

The parameter vector of the topologically-optimised ANN $G_{\hat{a}}$ produced from CERP is recorded in Table 5.6.3b.

Input to hidden layer	Bias	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10
Neuron 1	1.98156	0	0	0.137113	0	-1.54246	0	-0.114671	0	0.195466	0
Neuron 2	-0.156039	0	0	0	0	0	0	0	0.13805	0	0
Neuron 3	-3.20723	0	1.38972	2.50967	1.0019	0	-0.324854	0	0	0	0
Neuron 4	3.03371	1.97453	0	0	0	0	0	0	0	0	0
Neuron 5	2.78733	0	0	0	1.4579	0	0	0	0	0	2.54096
Neuron 6	-2.08238	0	0	0	0	1.74176	0	0	0	0	0
Neuron 7	1.89047	0	0.115946	0	0	0	0	1.38926	-0.104086	0.355344	0

Hidden to output layer	Bias	Neuron 1	Neuron 2	Neuron 3	Neuron 4	Neuron 5	Neuron 6	Neuron 7
Output neuron	0.320631	4.15035	3.15644	-0.786137	-0.165435	-0.336006	3.88299	0.607364

Table 5.6.3b Parameter vector of $G_{\hat{a}}$ from CERP for Problem 3

The corresponding performance of the $G_{\hat{a}}$ on the testing samples is listed in the top row of Table 5.6.3c; the performance of the $G_{\hat{a}}$ on the forecasting samples is listed in the bottom tow of Table 5.6.3c

standardised Testing Smaple RMSE	Directional correctness	Number of Samples
1.20646	-	100
Forecasting Sample RMSE	**Directional correctness**	**Number of Samples**
1.31697	-	200

Table 5.6.3c Performance of $G_{\hat{a}}$ produced from CERP on the testing samples (top row) and the forecasting samples (bottom row) for Problem 2

According to the value of standardised out-of-sample RMSE listed in Table 5.4.2b and Table 5.6.2c, for problem 3, the CERP with a single-arc addition implementation fails to optimise the complete ANN as the CERP iterations terminate on an ANN with a testing data RMSE worse than the complete ANN. Therefore, the network produced in Table 5.6.3b is not topologically optimised.

5.7 Robustness test

From the previous simulations, we have obtained 3 different ANNs from each of the three FX forecasting problems: a complete ANN, an ERP optimised ANN and a CERP optimised ANN. An immediate question emerges: how robust the ANN is - can the ANN be performed consistently well on data from a different period of time?

With the parameter vectors recorded in Table 5.4.1a to Table 5.6.3b, we can gauge the robustness of the ANNs by applying each topology on the out-samples of all three problems. Table 5.7 shows the results of this.

	Problem 1	Problem 2	Problem 3
Problem 1			
Complete ANN	1.33289	1.29396	1.25116
ERP-opt ANN	1.28416	1.29772	1.24685
CERP-opt ANN	1.27887	1.29862	1.2481
Problem 2			
Complete ANN	1.34585	1.26749	1.27778
ERP-opt ANN	1.48942	1.28228	1.28987
CERP-opt ANN	1.39189	1.28969	1.29857
Problem 3			
Complete ANN	1.60497	1.29961	1.24101
ERP-opt ANN	1.71889	1.38116	1.2348
CERP-opt ANN	1.56151	1.55825	1.31697

Table 5.7 Performance comparison on the robustness test

From Table 5.7, we can conclude that in terms of the forecasting RMSE, the topologies obtained from its original problem has consistently (highlighted in red) outperformed the other topologies. This suggests that if the underlying sampling period for the FX forecasting problems has changed, the ANN must be promptly retrained as well.

5.8 Comparative study of ANN and AR model

Timmerman's results [12] have shown that on forecasting US stock returns, the complete ANNs deliver a superior out-of-sample forecasting performance than some naive forecasting methods including the AR models. In our research it is also interesting to compare the performance of the complete ANN and the topology optimised ANN with the AR models on the problems of FX forecasting.

We constructed three sets of 1000 input-target sample pairs for the tests on ANNs and AR models. The inputs are $(R^1_{t-4}, R^1_{t-3}, R^1_{t-2}, R^1_{t-1}, R^1_t)$, which are the Eur/USD 1 Day Returns on day $t-4, t-3, t-2, t-1, t$ respectively, and the corresponding target is R^1_{t+1}. The data pairs are sampled for the same periods of times as the three forecasting problems listed in Section 5.2.

For the comparative study, the same system designed in Section 5.3 is applied for building the 5 input ANNs. The training, validation, testing and forecasting datasets are divided in the same proportion as the problems in Section 5.2.

An $AR(5)$ model is built: $R^1_{t+1} = c + \sum_{i=0}^{4} w_i R^1_{t-i} + \varepsilon$, where R_t is Eur/USD 1 Day Return on day t, c is a constant, w_i are the parameters and ε is the white noise. The rolling window approach is applied for AR(5), which uses a fixed-length window of the most recent 120 data samples to estimate the parameters (c, w) of the model and then predicts the target conditional on those parameter estimates.

Given the data pairs constructed above, the AR(5) is applied and the standerdised RMSE measurement between the the predictions and the targets is compared with the corresponding ANN forecasting results. Table 5.8 summrise the out-of-sample performance from the ANN topologies and the AR(5) on all three problems.

Table 5.8 indicates that for the data pairs 1 and the data pairs 2, all ANN topology has superior out-of-sample RMSE performance compare to the AR(5) models. For the data pairs 3, the system fails to find a good complete ANN, however, both ERP and CERP can optimise the

	Problem 1	Problem 2	Problem 3
Complete ANN	1.29006	1.31223	1.29897
ERP optimised ANN	1.28378	1.30483	1.26925
CERP optimised ANN	1.30398	1.30158	1.27583
AR(5)	1.31355	1.32534	1.28591

Table 5.8 Out-of-sample RMSE of ANN and AR(5)

	Problem 1	Problem 2	Problem 3
Complete ANN	110	106	105
ERP optimised ANN	116	112	117
CERP optimised ANN	104	105	107
AR(5)	105	110	105

Table 5.9 Out-of-sample directional correctness (out of 200 samples) of ANN and AR(5)

ANN to produce a better performing topology compare to AR(5) in terms of the out-of-sample forecasting performance

5.9 Summary and comparison of simulation results

In this chapter we have built a system for constructing, training, selecting and topology optimising ANN for the FFP with FX market data. The system is tested on three FX forecasting problems. For each of the problem sets, an optimally performing complete ANN is constructed and trained, which becomes the subject of topology optimisation simulation. The two ANN reduction applications: ERP and CERP are applied in turn, producing topologically-optimised ANNs with a reduced number of network connectivities. As a conclusion to the simulation, we summarise all the performance results from the complete ANN and from the topologically-optimised ANNs produced by ERP and CERP given each problem dataset, which is listed in Table 5.10.

Because ANNs are huristic (approximatly local optimal) and not not necessarily optimal, it is possible to impose a constraint (e.g. ERP and CERP) and the results to improve. Clearly, this would not have been a possible outcome if the complete ANNs were globally optimal and not local optimal.

According to Table 5.10, comparing the out-of-sample RMSE of the topologically-optimised ANNs obtained through ERP and CERP, a mixed result can be found. As the amount of improvement in RMSE evaluation for both testing sample or forecasting sample is varied, although in many cases ERP performs better in terms of the RMSE, CERP has also had some good results. This mixed comparison has verified our discussion at the end of Section 3.2 in Chapter 3, while the implementation of CERP has allowed for more number of training iterations than ERP, CERP does not account for the combined effect of the arcs as much as ERP does.

While there is no significant line to conclude whether ERP or CERP is better than the other, for all three problems, ERP produces a topologically-optimised ANN with improved performance compared to the selected optimally performing complete ANNs in terms of the testing samples; for problem 1 and problem 2, CERP produces a topologically-optimised ANN with improved performance. Note although it is indicated from the result of problem 2, a topology with improved testing sample performance may not produce a better forecasting performance, in more general cases (including the results from robustness test in Section 5.7 and comparative tests in Section 5.8) the forecasting performance is positively correlated to the testing sample performance. These evidently shows the outcome of this research that the proposed ANN reduction and its application can improve the generalisation performance of a complete ANN by optimising the topological graph subject to a reduced number of arc connectivities and the degree of freedom.

Further more, we have tested the robustness of the ANN topologies and compared the ANNs with an AR(5) models. The results shows that any ANN topology must be promptly retrained if the underlying data is changed and all ANN topology delivers superior out-of-sample performance than AR(5).

Testing Data Samples (In-sample)				
Problem	Topology	Standardised RMSE	% improvement (RMSE)	Directional correctness (out of 100 samples)
1	Complete	1.20372		56
	ERP	1.20107	0.22%	57
	CERP	1.20133	0.20%	59
2	Complete	1.25454		56
	ERP	1.18774	5.33%	49
	CERP	1.21739	2.96%	56
3	Complete	1.18548		51
	ERP	1.17569	0.83%	51
	CERP	1.20646[1]	-	-
Forecasting Data Samples (Out-of-Sample)				
Problem	Topology	Standardised RMSE	% improvement (RMSE)	Directional correctness(out of 200 samples)
1	Complete	1.33289		83
	ERP	1.28416	3.66%	104
	CERP	1.27887	4.05%	105
2	Complete	1.26749		110
	ERP	1.28228	-1.17%	111
	CERP	1.28969	-1.75%	110
3	Complete	1.24101		107
	ERP	1.2348	0.50%	112
	CERP	1.31697[1]	-	-

Table 5.10 Summary of performance evaluation for the selected complete ANN, and the topologically-optimised ANNs produce from ERP and CERP.

[1]This network is not topologically optimised from CERP with the single-arc addition implementation.

Chapter 6

Conclusion

6.1 Introduction

Using machine learning algorithms for financial data forecasting has long been an intriguing topic. In this thesis we are particularly focused on discussing the work related to a category of these learning algorithms, namely the ANN model. Inspired from the interconnectivity of neurons in the human brain, the ANN model is constructed in ways similar to that of human learning processes with supervision. The actual performance of training and forecasting is determined by the ANN's topology, which is the interconnection between the processing elements (activation functions of hidden neurons) and their parameters (weights of connections).

We discussed in the work of Cybenko and Bishop, who both proved that any function can be approximated with arbitrary accuracy by a feed-forward ANN with single-hidden-layer and hyperbolic tangent activation function on the hidden neurons. We also compared several learning processes for training the ANN system and selected a nonlinear conjugate gradient method with an associated implementation based on an inexact line search developed by Hanger and Zhang. These become the basis for all implementation of ANN topology, training and simulation throughout the research phase of this thesis.

A key issue remains to define the capability of ANN in solving the FFP, in view of the fact that the models usually train well in-sample but perform poorly in generalisation (out-of-sample). This is because the network's parameters often over-fit the in-sample data by the training process, but the model is insufficiently fitted to the underlying signal and performs poorly out-of-sample in forecasting applications.

The cause of this problem is that in many practical forecasting situations, an exact relationship between the input sample and the target data is not possible to be determined. Especially in the case of the FFP, training samples involve financial data which contains market noise which significantly affect any input-target relation. The stochastic nature of financial data also means the relationship is changing overtime. Therefore, in order to construct a good approximation model from a given training dataset, the ANN topology is often configured with more than sufficient degree of freedom by over-specifying the number of parameters rather than trying to find the necessary size, so that the model can be trained well by the information enclosed within the training set, which means that the extra degrees of freedom are trained on random noise. While the noise-trained parameters are fitted for approximating in-sample data, they produce a large error on the out-of-sample data.

The objective of this thesis is to explore a new approach to improve the generalisation performance of ANN models through optimising the ANN graph theoretic topological structure. As a result, a concept of a heuristic approach is proposed for optimising a complete ANN by altering its arc connectivities within a suitable range. This is in contrast to the conventional approaches of reducing the model's degrees of freedom. Albeit the approaches in the literature have similar objectives, none has produced an effective algorithm for optimising ANN applications in the financial field. Therefore, a more effective process on ANN topology optimisation needed to be found to fill-in this gap, particularly in applications to the FFP.

6.2 Theoretical framework and implementation

The design of the new topology optimisation algorithm, ANN reduction, is inspired from the class of λ-optimal algorithms for solving the TSP. Similar to the applications of λ-optimal in

TSP, the ANN reduction algorithm involves both arc disconnection and arc reconnection on the graph of the target complete ANN. The algorithm heuristically improves the out-of-sample performance of the ANN by altering a number of connections and the topology.

In the thesis, the ANN reduction algorithm is discussed along with a constraint that the vertex/layer topology of the target ANN is fixed. The constraint is in place so that all hidden neurons are kept connected with at least one input and output during the optimisation process. This implies that the number of neurons in the post-optimised topology is not changed from the pre-specified complete ANN. A network topology with a minimal number of connections which satisfy the above constraint is defined as a basic ANN.

An abstract design of ANN reduction algorithm is developed in two stages: 1) Basic ANN construction phase and 2) Arc re-connection phase

Basic ANN construction stage . Finds the optimal network with a minimum degree of freedom by performing arc disconnection on the complete ANN. While the disconnection is performed, a number of network structures that satisfy the static vertex constraints is produced. The best performing one is recorded as the basic ANN and also is a candidate solution of the topology optimisation called simplified ANN with the minimal degree of freedom.

Arc re-connection stage . Constructs new networks by reconnecting a specific number of previously disconnected arcs on the base ANN. An optimal structure is selected from many networks constructed this way and also becomes the simplified ANN with the specified degree of freedom. This phase is repeated with different numbers of arcs in the reconnection and is terminated when all possible numbers have been considered. The output of the topology optimisation is selected from all simplified ANN constructed, and the one with the best out-of-sample performance is selected, where the "out-of-sample" performance is measured on a subset of the reserved out-of-sample instances.

Following the abstract design of the ANN reduction algorithm, two applications are implemented, namely ERP and CERP. While both of the two applications use the same procedure for building the basic ANN, the ERP executes the arc re-connection phase similarly to what was described in the abstract design. CERP alters the abstract design by cascading a series of single arc re-connection processes to perform progressive optimisation.

For the purpose of practical implementation, in designing the two applications we have discussed several simplifications. The most important simplification that has been applied is at the basic ANN construction stage. This stage, which applied to both ERP and CERP, is a systematic process of pruning connections from the trained complete ANN, retaining only connections associated with heavy weighted parameters. The objective of the pruning process is to search for a network which has the minimal number of arcs to satisfy the static vertex constraint and has the maximum sum of weights from all possible arc combinations. The search result is retrained to become the basic ANN for the arc re-connection phase and the simplified ANN for performance comparison.

ERP executes arc re-connection by reconnecting a specified number of arcs simultaneously and randomly from the basic ANN. Because there is a huge number of potential combinations, as a simplification, a large but limited number of networks are constructed from the random process. These networks are accessed so the best performing ones are retrained and become a simplified ANN. The arc re-connection is then repeated with a different number of arcs specified for reconnection. All simplified ANN are compared to select the best as the output of ERP.

CERP executes arc re-connection by reconnecting only a small number of arcs initially on the basic ANN so that networks with all combination can be constructed, retrained and compared. If the best performing construction shows a performance improvement from the basic ANN, the best performing network is applied with further arc re-connection (with a small number of arcs again) until no further improvement can be found. The progressively improved network is a unique solution and is the output of CERP.

Both applications aim for the same objective of finding the ANN graph with an optimal degree of freedom to produce a better generalisation performance compared to the complete ANN with the same vertex/layer topology. Depending on the complexity of the forecasting problem, the algorithm find some near-optimal solutions to it. Because the two applications use different approaches, the output graphs from the two may be different.

6.3 Empirical evidence

We demonstrated the effectiveness of the ANN reduction algorithm through some simulations. For this purpose, a total of five datasets collected from various forecasting problems have been used for testing the two topology optimisation applications. Among the five datasets, two are generated from non-trivial "designed" functions; The other three datasets are obtained from the FX market for different time periods.

For the data samples generated from the designed functions, two functions are designed in the ways that they cannot be approximated easily by the ANNs. One dataset is produced from each of the functions. Within every input-target sample pair, the inputs are randomly generated values and the target is a corresponding output calculated according to the function given the inputs. Each dataset is then randomly distributed into three subsets: training, validating and testing for the simulation process.

For the data samples obtained from the FX market, the raw data are daily exchange prices across major currency countries over three non-overlapping periods. To build the forecasting problems, some raw data is transformed into technical indicators which are widely accepted in the industry for analysing and forecasting future returns. The input-target sample pairs consist of those technical indicators as the inputs and the actual exchange return at that time is the target. To mimic a realistic FFP with noisy and stochastic time series, each dataset are segmented into four subsets: training, validating, testing and forecasting, according to the sample order in time.

Throughout the simulations, the generalisation performance of the network structures is evaluated by the RMS difference between the outputs of ANN and the corresponding target samples of the testing set.

The simulation is initiated by constructing a complete ANN model from each data with the number of inputs and the number of outputs fixed by the training sample pairs. The topology of the complete ANN is determined by testing using different numbers of neurons in the hidden layer and is trained with many random initial training positions in order to produce a complete graph with the best performance. ERP and CERP are then applied in turn on the complete ANN to perform topology optimisation on the best performing complete ANN produced.

The simulation results on all five problems (both function generated data and FX data) have shown that ERP produce topologically-optimised ANNs which have better generalisation performance compared to the complete ANN. Although CERP with a single-arc addition implementation has only successfully produced topology optimised network in three cases, a possible course to improve this is to apply a revision on the arc addition process, which involves restarting the process by altering one or more previously added arcs.

We have analysed the performance plot and discussed the reason in terms of the combinatorial effects. Furthermore, on the three FX datasets, the complete ANN and the topologically-optimised ANNs are tested with the forecasting samples to evaluate their performance in solving the FFP. The RMSE evaluation on the test results has shown that all topologically-optimised ANNs produce some level of improvement in the forecasting performance compared to the complete ANNs. These results provide evidence to support our research objective in development of ANN reduction and the related implementations of ERP and CERP.

6.4 Limitation and future research

This research was conducted following a comprehensive study of ANN background including the structural design, training methodologies, generalisation issues, and existing methodologies for topology optimisations. Then the research focused on developing the theoretical framework for a heuristic algorithm: ANN reduction, for optimising single hidden layer ANNs with a static vertex/layer feed-forward topology and this was followed by implementations of the algorithms, ERP and CERP. Finally, the algorithms were evaluated on financial data forecasting problems to demonstrate the performance of the purposed algorithm. As a consequence of the research methodology, scope and implementation, the study has encountered a number of limitations, which can be considered in future research:

1. Further work towards dynamic topology optimisation: In this regard it may just be possible using a piecewise linearised approximation to the transfer function, to formulate small ANN-design problem as mixed integer programs and solve these optimally for small sizes. Much can be learned from such an approach, especially in providing an absolute comparison with heuristic techniques.
2. Extend from single hidden layer feed-forward topology: ANN reduction is discussed on the basis of optimising single hidden layer feed-forward structures. Although the globe approximation theorem has supported the use of such structure for approximating problems, it is highly likely that the optimal ANN for a given number of vertices and arcs is not a single hidden-layer structure, and the exact (small-size) solution mentioned above may lead to more exotic but also more effective graph topologies.
3. Effect of noise: it is axiomatic that noise reduces the effectiveness of any forecasting system. There are very few studies in the literature that consider this issue by performing carefully controlled injection of noise in a simulated environment and noting the different ANN structures that result for the same problem but at different noise levels. This is particularly important if ANNs are to be used more successfully in financial forecasting since, perhaps more important than the forecast itself, is the robustness of the forecasts and how this robustness is impacted by noise.

Bibliography

[1] Chou, Y. (1975). Statistical Analysis. Holt International.

[2] B. G. Malkei (1999). A Random Walk Down Wall Street, 7th ed.

[3] E. Maasoumi and J. Racine (2002). Entropy and Predictability of Stock Market Returns, Journal of Econometrics.

[4] E. Fama (1970). Efficient Capital Markets: A Review of Theory and Empirical Work, Journal of Finance 25 (2): 383–417.

[5] E. Fama (1965). The Behavior of Stock Market Prices, Journal of Business 38: 34–105.

[6] P. Samuelson (1965). Proof That Properly Anticipated Prices Fluctuate Randomly, Industrial Management Review 6: 41–49.

[7] Lo and MacKinlay (1999). A Non-Random Walk Down Wall Street.

[8] Dreman and Berry (1995). Overreaction, Underreaction, and the Low-P/E Effect, Financial Analysts Journal 51 (4): 21–30.

[9] E Gately (1995). Neural networks for financial forecasting.

[10] Odom and Sharda (1990). A neural network model for bankruptcy prediction. IJCNN International Joint Conference on Neural Networks 163 - 168 vol.2.

[11] Cao and Tay (2001). Financial Forecasting Using Support Vector Machines. Neural Computing & Applications Volume 10, Number 2, 184-192.

[12] Timmermann, A. (2008). Elusive return predictability. International Journal of Forecasting, 24(1), 1-18.

[13] Box, G. E. ., Jenkins, G. M., & Reinsel, G. C. (1976). Time series analysis. Holden-day San Francisco.

[14] Pankratz, A. (1983). Forecasting with univariate Box-Jenkins models (Vol. 3). Wiley Online Library.

[15] Nahmias, S., & Nahmias, S. (2004). Production and Operations Analysis with Student CD (5th ed.). McGraw-Hill/Irwin.

[16] Mitchell, T. (1997). Machine Learning. New York: McGraw-Hill.

[17] McCulloch, W. S., & Pitts, W. (1990). A logical calculus of the ideas immanent in nervous activity. Bulletin of mathematical biology, 52(1-2), 99-115.

[18] Cheng, W., Wagner, L., & Lin, C. H. (1996). Forecasting the 30-year US treasury bond with a system of neural networks. Journal of Computational Intelligence in Finance, 4(1), 10–16.

[19] Kutsurelis, J. (1998). Forecasting financial markets using neural networks: An analysis of methods and accuracy. Naval postgraduate school monterey ca.

[20] Moody, J. (1995). Economic forecasting: Challenges and neural network solutions. Proceedings of the International Symposium on Artificial Neural Networks.

[21] Yao, J., & Tan, C. L. (2000). A case study on using neural networks to perform technical forecasting of forex. Neurocomputing, 34(1), 79–98.

[22] Chitra, A., & Uma, S. (2010). An Ensemble Model of Multiple Classifiers for Time Series Prediction. International Journal of Computer Theory and Engineering, 2, 454–458.

[23] Shakhnarovich, G. (2005). Nearest-neighbor methods in learning and vision theory and practice. Cambridge, Mass, MIT Press,.

[24] Toni, T.; Welch, D.; Strelkowa, N.; Ipsen, A.; Stumpf, M.P.H., & Toni, T.; W. (2009). Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. Journal of the Royal Society Interface, 6(31)

[25] Drucker, H., Burges, C., Kaufman, L., Smola, A., & Vapnik, V. (1997). Support vector regression machines, advances in neural information processing systems, 9, 156-161.

[26] Muller, K. R., Smola, A., Ratsch, G., Scholkopf, B., Kohlmorgen, J., & Vapnik, V. (1999). Using support vector machines for time series prediction. Advances in kernel methods: Support vector learning, 253.

[27] Samuel, A. L. (2000). Some studies in machine learning using the game of checkers. IBM Journal of Research and Development, 44(1.2), 206 –226.

[28] Ghahramani, Z. (2004). Unsupervised learning. Advanced Lectures on Machine Learning, 72–112.

[29] Dayan, P. (1999). Unsupervised learning. The MIT Encyclopedia of the Cognitive Science. MIT.

[30] Cooper, M. (1999). Filter rules based on price and volume in individual security overreaction. Review of Financial Studies, 12(4), 901–935.

[31] Krollner, B., Vanstone, B., & Finnie, G. (2010). Financial time series forecasting with machine learning techniques: A survey. European Symposium on Artificial Neural Networks: Computational and Machine Learning.

[32] Moody, J., & Utans, J. (1994). Architecture selection strategies for neural networks: Application to corporate bond rating prediction. Neural networks in the capital markets (pp. 277–300).

[33] Dutta, S., & Shekhar, S. (1988). Bond rating: A nonconservative application of neural networks. Neural Networks, 1988., IEEE International Conference on (pp. 443–450).

[34] Callen, J. L., Kwan, C. C. ., Yip, P. C. ., & Yuan, Y. (1996). Neural network forecasting of quarterly accounting earnings. International Journal of Forecasting, 12(4), 475–482.

[35] Kim, K., & Han, I. (2000). Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. Expert Systems with Applications, 19(2), 125–132.

[36] Fama, E. F., & French, K. R. (1993). Common risk factors in the returns on stocks and bonds. Journal of financial economics, 33(1), 3–56.

[37] Officer, R. R. (1972). The Distribution of Stock Returns. Journal of the American Statistical Association, 67(340), 807-812. doi:10.2307/2284641

[38] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. Bulletin of mathematical biology, 5(4), 115–133.

[39] Anderson, J. A., & Davis, J. (1995). An introduction to neural networks (Vol. 1). MIT Press.

[40] Rumelhart, D. E., & McClelland, J. L. (1986). Parallel distributed processing: Psychological and biological models (Vol. 2). The MIT press.

[41] Wilson, H. R., & Cowan, J. D. (1972). Excitatory and inhibitory interactions in localized populations of model neurons. Biophysical journal, 12(1), 1–24.

[42] Bertsekas, D. P., & Tsitsiklis, J. N. (1996). Neuro-Dynamic Programming (1st ed.). Athena Scientific.

[43] Bishop, C. M. (1995). Neural networks for pattern recognition.

[44] Gibson, G. J., & Cowan, C. F. N. (1990). On the decision regions of multilayer perceptrons. Proceedings of the IEEE, 78(10), 1590–1594.

[45] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems (MCSS), 2(4), 303–314.

[46] Irie, B., & Miyake, S. (1988). Capabilities of three-layered perceptrons. Neural Networks, 1988., IEEE International Conference on (pp. 641–648).

[47] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. Proceedings of the national academy of sciences, 79(8), 2554.

[48] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735–1780.

[49] Rumelhart, D. E., Hintont, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533–536.

[50] Fletcher, R., & Reeves, C. (1964). Function minimization by conjugate gradients. The computer journal, 7(2), 149–154.

[51] Polak, E., & Ribiere, G. (1969). Note sur la convergence de directions conjugates, Rev. Franataise Informat. Recherche Opertionelle, 3e annate, 16, 35–43.

[52] A New conjugate gradient method with guaranteed descent and an efficient line search - WW Hager, H Zhang (2006)

[53] Everitt, B. S., & Skrondal, A. (2002). The Cambridge dictionary of statistics. Cambridge: Cambridge.

[54] Chitra, A., & Uma, S. (2010). An Ensemble Model of Multiple Classifiers for Time Series Prediction. International Journal of Computer Theory and Engineering, 2, 454–458.

[55] Rumelhart, D. E. (1985). Learning internal representations by error propagation. DTIC Document.

[56] Castellano, G., Fanelli, A. M., & Pelillo, M. (1997). An iterative pruning algorithm for feedforward neural networks. Neural Networks, IEEE Transactions on, 8(3), 519–531.

[57] Kung, S., & Hwang, J. (1988). An algebraic projection analysis for optimal hidden units size and learning rates in back-propagation learning. Neural Networks, 1988., IEEE International Conference on (pp. 363–370).

[58] Plaut, D. C., & Hinton, G. E. (1987). Learning sets of filters using back-propagation. Computer Speech & Language, 2(1), 35–61.

[59] Burr, D. J. (1988). Experiments on neural net recognition of spoken and written text. Acoustics, Speech and Signal Processing, IEEE Transactions on, 36(7), 1162–1168.

[60] Rumelhart, D. E., Hintont, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533–536.

[61] Yu, X.-H. (1992). Can backpropagation error surface not have local minima. IEEE Transactions on Neural Networks, 3(6), 1019-1021. doi:10.1109/72.165604

[62] LippmaANN, R. P. (1987). An introduction to computing with neural nets. ARIEL, 209, 115–245.

[63] Emmerson, M. D., & Damper, R. (1993). Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application. Neural Networks, IEEE Transactions on, 4(5), 788–793.

[64] Reed, R. (1993). Pruning algorithms-a survey. Neural Networks, IEEE Transactions on, 4(5), 740–747.

[65] Baum, E. B., & Haussler, D. (1989). What size net gives valid generalization? Neural computation, 1(1), 151–160.

[66] Denker, J., Schwartz, D., Wittner, B., Solla, S., Howard, R., Jackel, L., & Hopfield, J. (1987). Large automatic learning, rule extraction, and generalization. Complex systems, 1(5), 877–922.

[67] Chauvin, Y. (1990). Generalization performance of overtrained back-propagation networks. Neural Networks, 45–55.

[68] Towell, G. G., Craven, M. W., & Shavlik, J. W. (1991). Constructive induction in knowledge-based neural networks. Proc. Eighth Intl. Workshop Machine Learning (pp. 213–217).

[69] LeCun, Y., & others. (1989). Generalization and network design strategies. CoANNectionism in perspective, 143–155.

[70] Towell, G., & Shavlik, J. W. (1992). Interpretation of Artificial Neural Networks: Mapping knowledge-based Neural Networks into Rules. Advances in Neural Information Processing Systems, pp. 977-984, Denver, CO. Morgan KaufmaANN.

[71] Xu, J., & Ho, D. W. C. (2006). A new training and pruning algorithm based on node dependence and Jacobian rank deficiency. Neurocomputing, 70(1-3), 544–558.

[72] Narasimha, P. L., Delashmit, W. H., Manry, M. T., Li, J., & Maldonado, F. (2008). An integrated growing-pruning method for feedforward network training. Neurocomputing, 71(13-15), 2831–2847.

[73] Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1989). Learnability and the Vapnik-Chervonenkis dimension. Journal of the ACM (JACM), 36(4), 929–965.

[74] Ehrenfeucht, A., Haussler, D., Kearns, M., & Valiant, L. (1989). A general lower bound on the number of examples needed for learning. Information and Computation, 82(3), 247–261.

[75] Vapnik, V. N., & Chervonenkis, A. Y. (1971). On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. Theory of Probability and its Applications, 16(2), 264. doi:10.1137/1116025

[76] Zhang, J., & Morris, A. (1998). A sequential learning approach for single hidden layer neural networks. Neural networks, 11(1), 65–80.

[77] Kadirkamanathan, V., & Niranjan, M. (1993). A function estimation approach to sequential learning with neural networks. Neural Computation, 5(6), 954–975.

[78] Chung, F., & Lee, T. (1995). Network-growth approach to design of feedforward neural networks. Control Theory and Applications, IEE Proceedings- (Vol. 142, pp. 486–492).

[79] Platt, J. (1991). A resource-allocating network for function interpolation. Neural computation, 3(2), 213–225.

[80] Buhmann, & Martin D. (2003). Radial Basis Functions: Theory and Implementations. Cambridge University Press.

[81] Tin-Yau Kwok, & Dit-Yan Yeung. (1997). Constructive algorithms for structure learning in feedforward neural networks for regression problems. IEEE Transactions on Neural Networks, 8(3), 630-645.

[82] Ash, T. (1989). Dynamic Node Creation in Backpropagation Networks. Connection Science, 1(4), 365-375.

[83] Azimi-Sadjadi, M. R., Sheedvash, S., & Trujillo, F. O. (1993). Recursive dynamic node creation in multilayer neural networks. IEEE Transactions on Neural Networks, 4(2), 242-256.

[84] Friedman, J. H., & Stuetzle, W. (1981). Projection Pursuit Regression. Journal of the American Statistical Association, 76(376), 817-823.

[85] Jones, L. K. (1987). On a Conjecture of Huber Concerning the Convergence of Projection Pursuit Regression. The Annals of Statistics, 15(2), 880-882.

[86] Karnin, E. D. (1990). A simple procedure for pruning back-propagation trained neural networks. Neural Networks, IEEE Transactions on, 1(2), 239–242.

[87] Mozer, M. C., & Smolensky, P. (1989). Skeletonization: A technique for trimming the fat from a network via relevance assessment. Morgan Kaufmann Publishers Inc.

[88] Le Cun, Y., Denker, J. S., Solla, S. A., Howard, R., & Jackel, L. (1990). Optimal brain damage. Advances in neural information processing systems, 2(1), 1990.

[89] Plaut, D. C., Nowlan, S. J., & Hinton, G. E. (1986). Experiments on learning by back propagation.

[90] Chauvin, Y. (1989). A back-propagation algorithm with optimal use of hidden units. Advances in neural information processing systems 1 (pp. 519–526).

[91] Segee, B. E., & Carter, M. J. (1991). Fault tolerance of pruned multilayer networks. , IJCNN-91-Seattle International Joint Conference on Neural Networks, 1991 (Vol. ii, pp. 447-452 vol.2).

[92] Kruschke, J. (1988). Creating local and distributed bottlenecks in hidden layers of back-propagation networks. Proceedings of the 1988 Connectionist Models Summer School (pp. 120–126).

[93] Whitley, D., Starkweather, T., & Bogart, C. (1990). Genetic algorithms and neural networks: optimizing connections and connectivity. Parallel Computing, 14(3), 347-361.

[94] Zhang, B. T., & Muhlenbein, H. (1993). Evolving optimal neural networks using genetic algorithms with Occam's razor. Complex Systems, 7(3), 199–220.

[95] Augasta, M. G., & Kathirvalavakumar, T. (2011). A Novel Pruning Algorithm for Optimizing Feedforward Neural Network of Classification Problems. Neural Processing Letters, 1–18.

[96] Bellman, R. E. (1961). Adaptive control processes: a guided tour. Princeton University Press.

[97] Scott, D. W. (1992). Multivariate Density Estimation: Theory, Practice, and Visualization. Wiley-Blackwell.

[98] Hillyard, S. and Galambos, R. (1967). Effects of stimulus and response contingencies on a surface negative slow potential shift in man. Electroencephalography and Clinical Neurophysiology, 22(4), pp.297-304.

[99] Jones, L. (1990). Constructive approximations for neural networks by sigmoidal functions. Proc. IEEE, 78(10), pp.1586-1589.

[100] Newell, A., & Simon, H. A. (1976). Computer science as empirical inquiry: Symbols and search. Communications of the ACM, 19(3), 113–126.

[101] Xin Yao. (1999). Evolving artificial neural networks. Proceedings of the IEEE, 87(9), 1423–1447. doi:10.1109/5.784219

[102] Angeline, P. J., Saunders, G. M., & Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. IEEE Transactions on Neural Networks, 5(1), 54–65. doi:10.1109/72.265960

[103] G. F. Miller, P. M. Todd, and S. U. (1989). Hegde, Designing neural networks using genetic algorithms, in Proc. 3rd Int. Conf. Genetic Algorithms and Their Applications, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 379-384.

[104] Bondy, J. A., & Murty, U. S. R. (1976). Graph Theory With Applications. Elsevier Science Ltd/North-Holland. 65

[105] Lin, S. & Kernighan, B. W. (1973). An Effective Heuristic Algorithm for the Traveling-Salesman Problem. Operations Research 21 (2): 498-516.

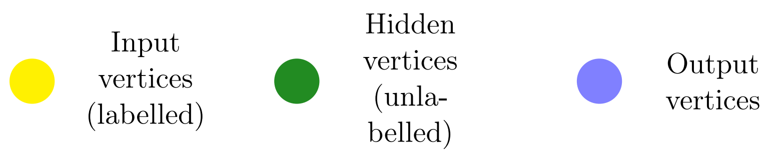
[106] Sharp Henry (1968), Cardinality of finite topologies, J. Combinatorial Theory 5: 82-86.

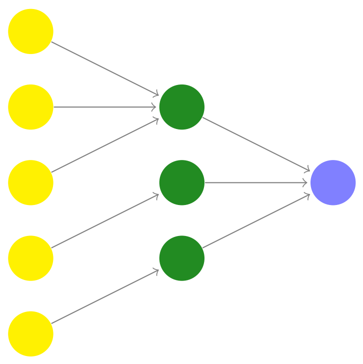
[107] Bollinger, John. Bollinger on Bollinger Bands. McGraw Hill, 2002.

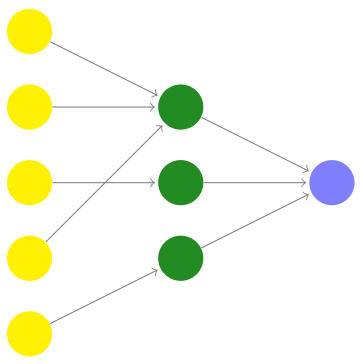
[108] Kaufman, Perry J. New Trading Systems And Methods. 5th ed. Hoboken, N.J.: John Wiley & Sons, 2005. 273-275.

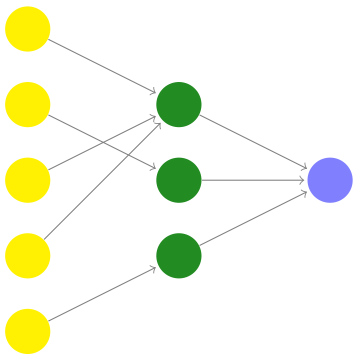
Appendix A

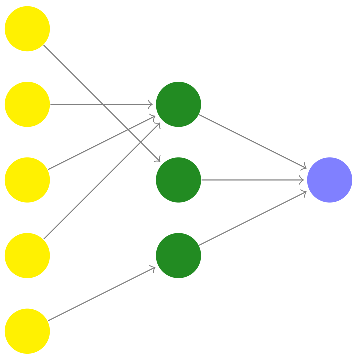
25 Possible topology of 5-3-1 Basic ANN

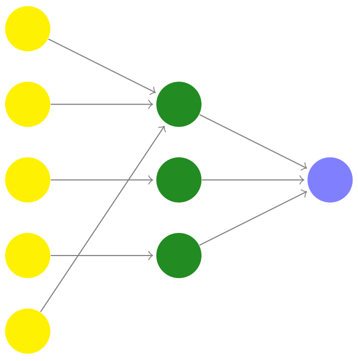


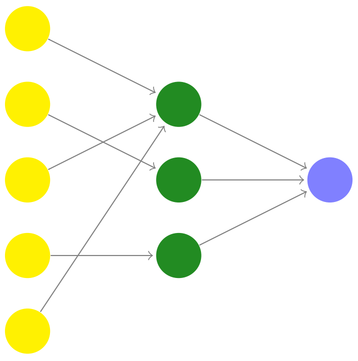

1

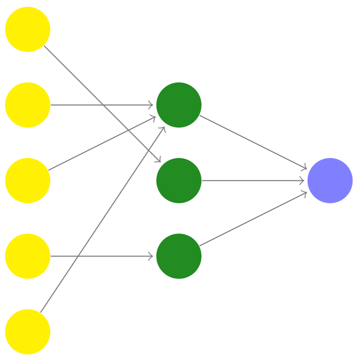

2

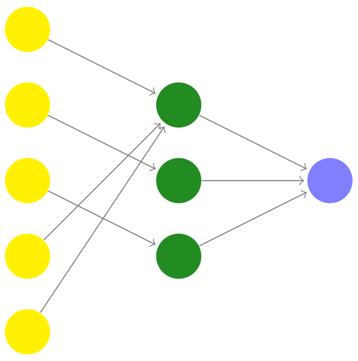

3

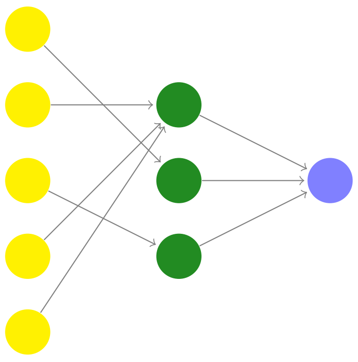

4

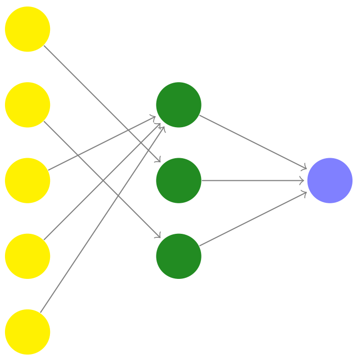

5

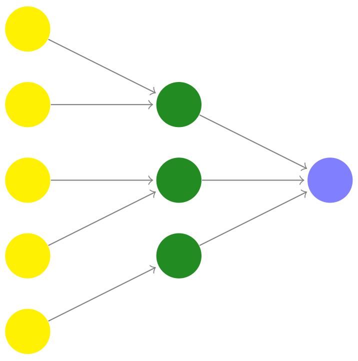

6

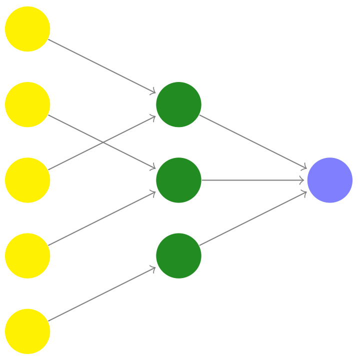

7

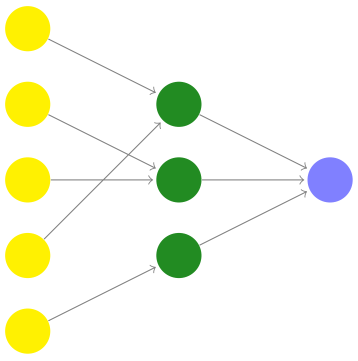

8

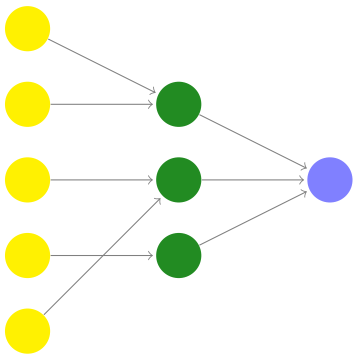

9

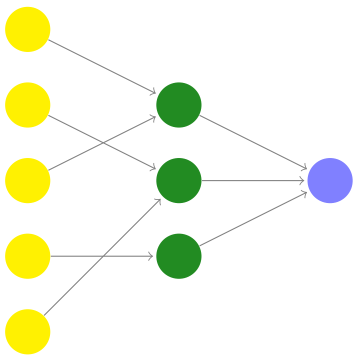

10

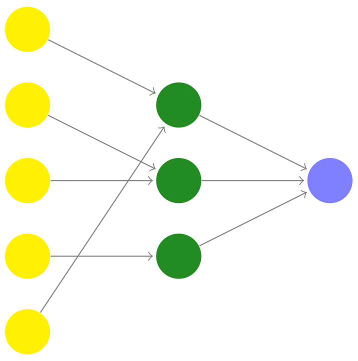

11

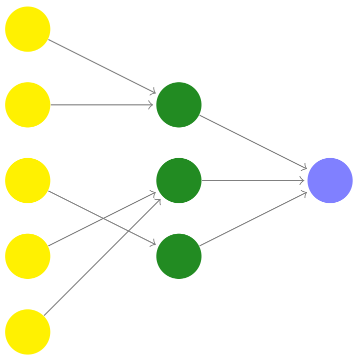

12

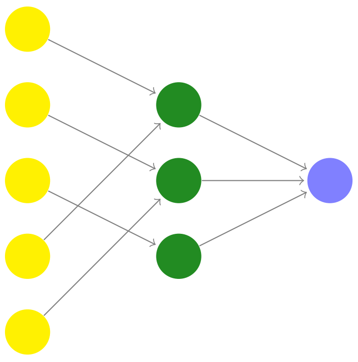

13

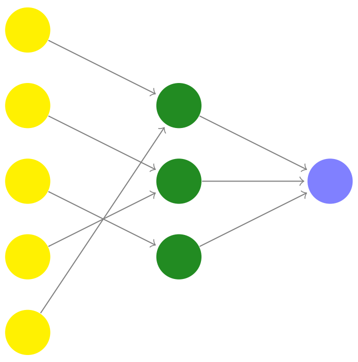

14


15

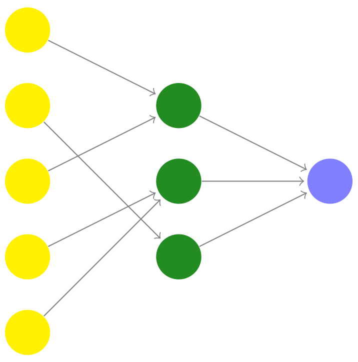

16


17

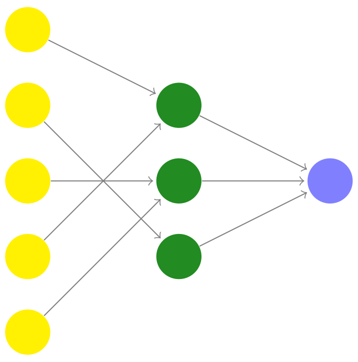

18



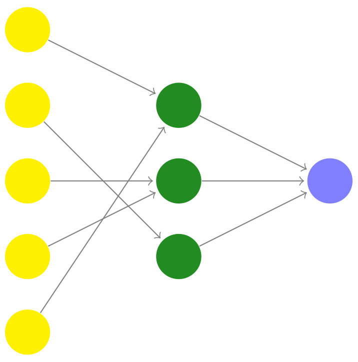
19



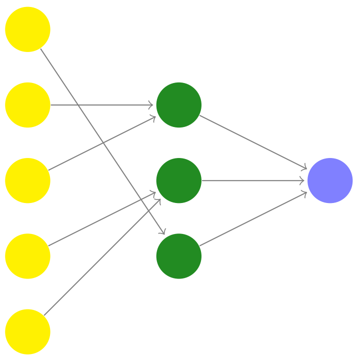
20



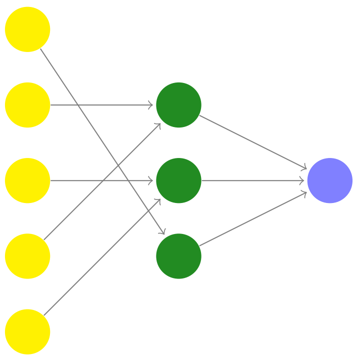
21



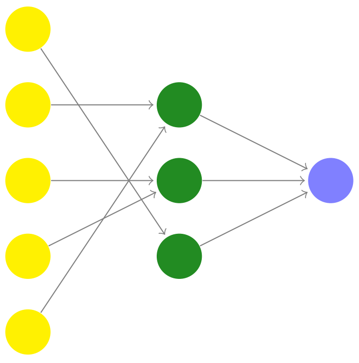
22



23



24



25