A study of different AI techniques applicability to forecast currency exchange market movements

Final Report for CS39440 Major Project

Author: Piotr Chudzik (pcc9@aber.ac.uk)
Supervisor: Dr Elio Tuci (elt7@aber.ac.uk)

25th April 2012
Version: 1.0 (Release)

This report was submitted as partial fulfilment of a BSc degree in Artificial Intelligence & Robotics (GH7P)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Acknowledgements

I am truly indebted and grateful to my supervisor, Dr Elio Tuci, without whose continual support and guidance this project would not have been possible. I would not have been able to even start this project without his help and support. His recommendations and instructions have enabled me to assemble and finish this dissertation effectively.

Above all,I would like to thank all my lecturers and tutors in current and previous years, who throughout my educational career have supported and encouraged me to believe in my abilities. They have directed me through various situations, allowing me to reach this accomplishment.

Finally, I would like to thank my mother for giving me the dedication and motivation to work to the best of my ability. To her, I am eternally grateful.

Abstract

Financial prediction is a research active area and artificial neural networks have been proposed as one of the most promising methods for such prediction. This report summarizes a study of a number of artificial neural network models designed to forecast a currency exchange rate movements on Forex market. It provides an introduction to the domain of financial time series prediction, emphasizing the issues particularly important with respect to the artificial neural network approach to this task. Three different artificial neural network morphologies and two different training algorithms are designed, implemented and tested to determine if accurate prediction can be achieved. Obtained results indicate superior performance of the proposed models as compared to a traditional statistical analysing tool where the prediction is just a mean of all samples.

CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

”There are only patterns, patterns on top of patterns, patterns that affect other patterns. Patterns hidden by patterns. Patterns within patterns. If you watch close, history does nothing but repeat itself. What we call chaos is just patterns we haven’t recognized. What we call random is just patterns we can’t decipher.”

— Chuck Palahniuk, Survivor

1.1 Context

The global economic crisis that started in 2008 showed us that the world is ruled not by governments but by financial markets. When Lehman Brothers went bankrupt in September 2008 not even the greatest pessimists expected that this will result in a worldwide recession. Although there are many theories about real causes of current economic climate most of experts agree that Lehman Brothers bankruptcy caused a worldwide crisis of trust between banks and governments and, as a result, the flow of money in national economies was abruptly stopped but loans had to be repaid. Currently whole countries face bankruptcy and no one knows for sure when this economic meltdown will come to an end. Learning from their mistakes governments are hastily creating lists of financial institutions that are ”too big to fall”. It is hard to believe that a one bankruptcy started a worldwide snowball effect and directly or indirectly affected every human on the planet. If only people responsible for letting Lehman Brothers to go bankrupt could forecast what effect this will have on financial markets worldwide, they would probably rethink their decision. If only they would have a *blackbox* tool that given Lehman Brothers bankruptcy as input could forecast a global crisis.

Today’s financial markets demand new analytical tools for survival and profit as existing methods of analysis lose their luster. Forecasting can be defined as identifying which variables lead or help to predict other variables interacting with each other. The Lehman Brothers bankruptcy can be described as a variable that affected many other variables in a very complex, non-linear way. In the world of finance everything is quantifiable meaning it can be expressed as a number. The biggest difficulty lies in finding and defining relations between those numbers and as a result creating a model. Modelling real world data like financial data is extremely difficult due to intrinsic noise, high-dimensionality and non-linearities. All financial markets are very volatile - price, spread, exchange rate etc. can move unpredictably and without obvious cause. If we would know the true underlying model generating data observed in financial markets we would be able to obtain very accurate forecasts of what might happen. However in practice the model is too complex or we might come up with many competing models and we don’t know which one is the true model.

So we have to approximate and artificial neural networks(ANNs) are proven to be one of the best tools used for approximation purposes.

ANNs are superior to traditional tools used for technical analysis(e.g. regression models) due to a number of reasons:

1. ANNs can ”learn” from experience meaning that they can automatically and continually improve their performance.
2. ANNs perform exceptionally well on moderately noisy and chaotic data.
3. ANNs excel at capturing non-linearity in the data.
4. ANNs can cope with ”fuzzy” patterns - patterns easily recognized by the eye but difficult to define using formal rules.
5. ANNs can be retrained and thereby adapt to changing market behaviour.
6. Theoretically ANNs given a proper training are able to recognize any pattern that might exist in a data.

Apart from the US Department of Defense, the financial services industry has invested more money in neural network research than any other industry or government body [54]. Currently all major financial institutions are using ANNs for a number of applications. For example, Standard and Poor’s Neural Fair Value (NFV) system used for building portfolios and ranking stocks is based on ANNs.

1.2 Aims

In the early stages of this project, two possibility of how this project might evolve were considered:

- The project will concentrate on building an automated trading platform which would consume web services and have a database back-end. Due to limited time, the AI element would be restricted to minimum and implemented using existing ANN tools/libraries.
- The main goal of the project will be to research AI techniques involving ANNs and genetic algorithms to forecast financial time series data. The main emphasis is put on the AI element and the application will have a basic GUI needed for testing a number of models used for prediction.

Author’s main interests lie in using AI techniques in real world applications. Moreover, due to the fact that the author has spent a year in industry creating applications consuming web services, he was not interested in creating another such application which would not be a challenge. Moreover using existing ANN tools is a great simplification of the AI element and, in fact, even if one is using such tools it does not say that he/she understands what is going on under the hood. As a result the project was decided to be a research project with the main emphasis on AI elements which will be developed from scratch.

The main aim of this project is to investigate the applicability of different neural networks architectures to forecast currency exchange rates movements on Forex market. Multiple neural networks will be trained using historical data and optimized using a number of training algorithms including genetic algorithms.

In order to achieve this aim the author has formulated two problems that will be tackled by neural networks models:

I We have time series i.e. a variable x changing in time $x_t(t = 1, 2 \ldots)$ and we would like to predict the value of x in time $t + h$ (depending on a time frame h can be 1 minute, 1 hour, 1 day).

Neural networks will try to predict the exact value of a given currency pair exchange rate in the future.

II We have time series i.e. a variable x changing in time $x_t(t = 1, 2 \ldots)$ and we would like to predict if x will increase, decrease or stay the same in the future. Moreover we would like to know the magnitude (percentage change) of a change if any.

Neural networks will try to predict the direction(or it absence) and magnitude of a price movement.

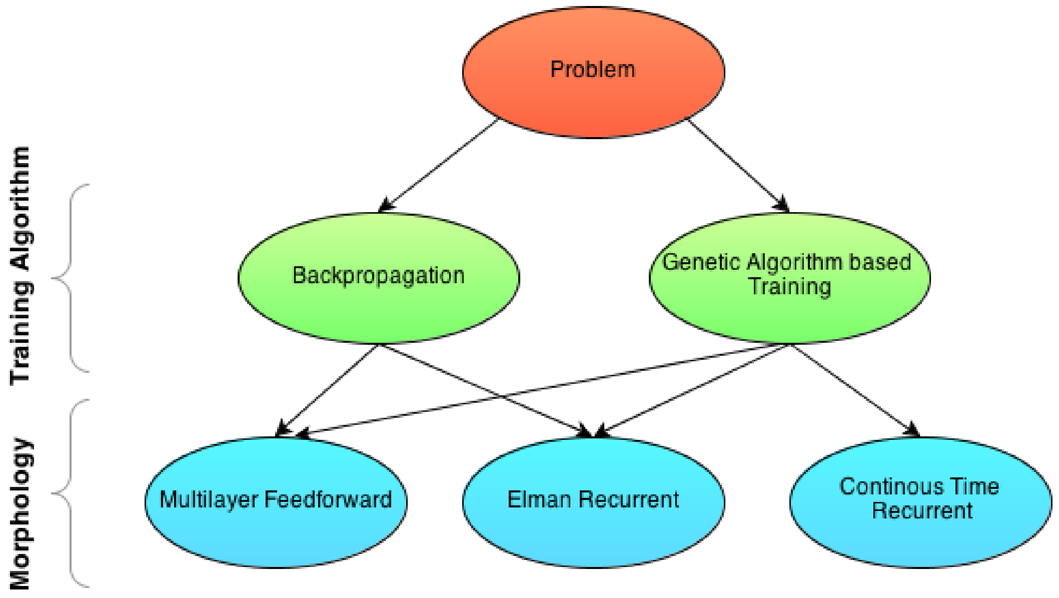


Figure 1.1: ANN models to be implemented.

It is important to note that the main goal of this project is to test applicability of different ANN models for prediction tasks. The ”Neural Network Mentalist” application is just a tool used for testing purposes and is not the main objective of this project. Moreover in context of trading, ANNs created in this project cannot be treated as ultimate tools to trade on Forex or any other financial market, but rather as another element in a larger porfolio of analysing tools.

1.3 Success Evaluation

ANN performance and results will be measured using a number of statistical indicators. The success of a particular model will be measured using three main benchmarks:

I **MSE Final Value and Convergence** - Each training algorithm used will produce MSE (mean squared error, discussed later) results that should converge to some small value. If they do not it means that there is some bug in a training algorithm implementation. MSE is the main measure of ANN performance

II R^2 - coefficient of determination. R^2 measures the quality of prediction by comparing it with a simple benchmark where a prediction is a mean of all samples (discussed later).

III **Time** - One of the main drawback of ANN is a long time needed to train them. Different training algorithm will require different amount of time to produce meaningful results. If two training algorithms produce similar results but the first one requires couple minutes whereas the second one couple days to train, the first one is much more efficient than the second one.

Initially, the author has planned to compare obtained results with results published by researchers in the respective field. Unfortunately most of scientific papers concerning the use of ANNs to predict currency exchange rate does not provide any statistical measures of the results. Moreover they do not provide any details about input data used for training and testing hence it is impossible to compare results.

Since those statistical indicators do not say much to an ordinary user, the evaluation mode will be created. In the evaluation mode the user will be able to visually compare results accomplished by ANN and expected values.

The user will be able to query the ANN with data it did not see before. The expected results might not even exist at the time of a query being made. It is important to note that results of such query do not influence the overall performance of ANN which is measured using testing data only. This mode was created strictly for presentation purposes only.

Chapter 2

Background

To understand is to perceive patterns.

— Isaiah Berlin

2.1 Artificial Neural Networks

This section discusses artificial neural networks - history of their development, basic definition, significance of their structure and training algorithms.

2.1.1 History

Similarly to many concepts in the field of Artificial Intelligence, the history of ANNs can be described in three terms: initial excitement, disappointment and silent adoption.

The history of ANNs starts in 1943 when McCulloch and Pitts came up with a first model of a neuron and a neural network based on neurobiology [43]. This neural network model was very simplistic: it worked on binary data only and it did not incorporated weighting of different inputs. In 1949, Donald Hebb in his book ”The Organization of Behavior” defined so called Hebbs rule - When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that As efficiency, as one of the cells firing B, is increased. [30]. In other words , when two neurons fire together the connection between them is strengthened. Moreover Hebb concluded that the weighting of inputs is necessary for learning process. In 1950s the majority of ANNs researchers were neurobiologists trying to understand how a brain can produce highly complex patterns by using interconnected simple cells.

The breakthrough came with a perceptron model created by Frank Rosenblatt. Rosenblatt using McCulloch's and Pitt's neuron and Hebb's weighting of inputs came up with a neural network model consisting of fully connected input and output layers. Each input was weighted by weights between input and output neurons. Weights could be different for different inputs and by updating those weights a desirable output could be obtained. As a result perceptron was able to recognize simple patterns. Rosenblatt summarised his findings in his famous book ”Principles of Neurodynamics: perceptions and the theory of brain mechanisms” published in 1962 [51]. As a result, the popularity of neural networks flourished during 1960s and a lot of research funding was awarded to this field.

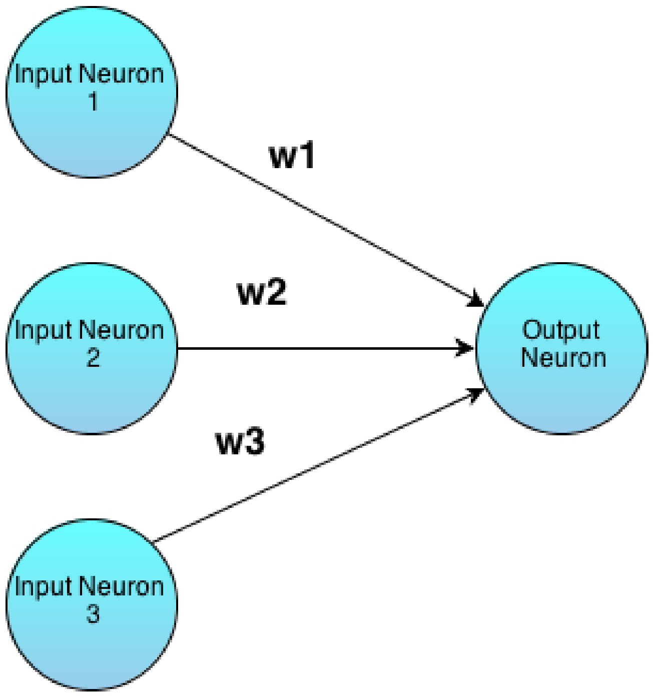


Figure 2.1: Perceptron.

From the perspective of time we can now safely say that some claims Rosenblatt made in his book were wrong, e.g. : ”Given an elementary -perceptron, a stimulus world W, and any classification C(W) for which a solution exists; let all stimuli in W occur in any sequence, provided that each stimulus must reoccur in finite time; then beginning from an arbitrary initial state, an error correction procedure will always yield a solution to C(W) in finite time...” [51]. In other words Rosenblatt claimed that given enough information perceptrons are able to solve any problem which we know is incorrect. Moreover Rosenblatt criticized some more traditional symbol manipulation research what provoked many other AI experts at a time including Marvel Minsky, Rosenblatt’s friend since adolescence. Rosenblatt and Minsky were central figures of a neural network debate followed by whole AI community at a time (very similar to the public debate between Norvig and Chomsky about future of AI nowadays [14]). The growing popularity of neural networks was stopped by the book ”Perceptrons” authored by Marvin Minsky and Seymour Papert in 1969 [45]. Minsky and Papert proved that a perceptron model is able to solve only simple linearly separable problems (e.g. logical AND,OR) and it fails to solve more complex non-linearly separable problems (e.g. logical XOR). They concluded that because of this limitation further research of perceptrons is futile. As a result of this publication neural network research funding was cut and a number of researchers involved in this field drastically dropped down.

Although during 1970s research of ANNs was scarce it is important to note the invention of the backpropagation algorithm by Paul Werbos in 1974 [58]. The potential of the backpropagation as a training technique was not realized until work done by David Rumelhart, Geoffrey Hinton and Ronald Williams on multilayer perceptrons summarised in their article Learning Internal Representations by Error Propagation published in 1986 [52]. They overcame limitations of Rosenblatt's perceptron by adding extra layers (so called *hidden layers*) between input and output layers of the original perceptron. By using Werbos's backpropagation algorithm they were able to solve even non linearly separable problems marking new age in artificial neural networks history.

Since then neural networks were successfully used in a number of real world applications, excelling in classification and pattern recognition. Even nowadays their capabilities are still researched around the world and new successful applications of ANNs are invented every year.

2.1.2 Definition

From the time of first computing machines, people tried to create machines that would think instead of just calculating values. However what is meant by thinking is highly debatable and there are as many definition of "thinking" as there are intelligence definitions. Artificial Neural Networks (ANNs) are one of many approaches to mimic human like thinking process.

"Neural network learning methods provide a robust approach to approximating real-valued, discrete-valued and vector-valued target functions.For certain types of problems, such as learning to interpret complex real-world sensor data, artificial neural networks are among the most effective learning methods currently known." [46, p. 81]. Neural networks originated from neurobiology and are modelled based on a human brain structure. Similarly to a human brain a neural network consists of neurons connected with each other by synapses. Those neurons pass data in a highly structured way and assign a value (weight) to each piece of data. So the importance of a single information is calculated as a weighted sum of all inputs. In order to better understand analogy between a neural network and a human brain T.M. Mitchell provides us with a number of facts from the field of neurobiology concerning a human brain - "The human brain, for example, is estimated to contain a densely interconnected network of approximately 10^11 neurons, each connected, on average, to 10^4 others. Neurons activity is typically excited or inhibited through connections to other neurons. The fastest neuron switching times are known to be on the order of 10^-3 seconds, quite slow compared to computer switching speeds of 10^-10 seconds. Yet humans are able to make surprisingly complex decisions, surprisingly quickly. For example, it requires approximately 10^-1 seconds to visually recognize one's mother. Notice the sequence of neuron firings that can take place during this 10^-1-second interval cannot possibly be longer than a few hundred steps, given that the information-processing abilities of biological neural systems must follow from highly parallel processes operating on representations that are distributed over many neurons." [46, p. 81].

There are two main directions of research of ANNs:

- Computational neuroscience - researching properties of biological systems by using ANNs.
- Approximation of complex functions - using ANNs to approximate complex, often based on real-world data functions.

In scope of this project ANNs are used as approximators (estimators) or real-world data.

2.1.3 Morphology

A Morphology (structure) of an ANN directly affects its ability to approximate data as accurately as possible. There are many types of ANNs morphologies, from relatively simple to fairly complex, just as there are many theories on how biological neural processing works. Different morphologies should be applied to different types of problems and some morphologies are not applicable to some classes of problems. E.g. if the same results can be obtained using two different inputs we need to consider them as two distinctive cases. In order to do that we need some type of memory to keep track of context which in this case are previous inputs. There is a class of ANNs called recurrent ANNs that introduces the concept of memory to the model through a certain type of morphology (see section 4.2.2). The efficiency of ANN is directly affected by a morphology choice as well because more connections between neurons result in more calculations that needs to be performed and more memory needed to keep track of results. The basic morphology of a typical ANN is shown in Figure 2.2

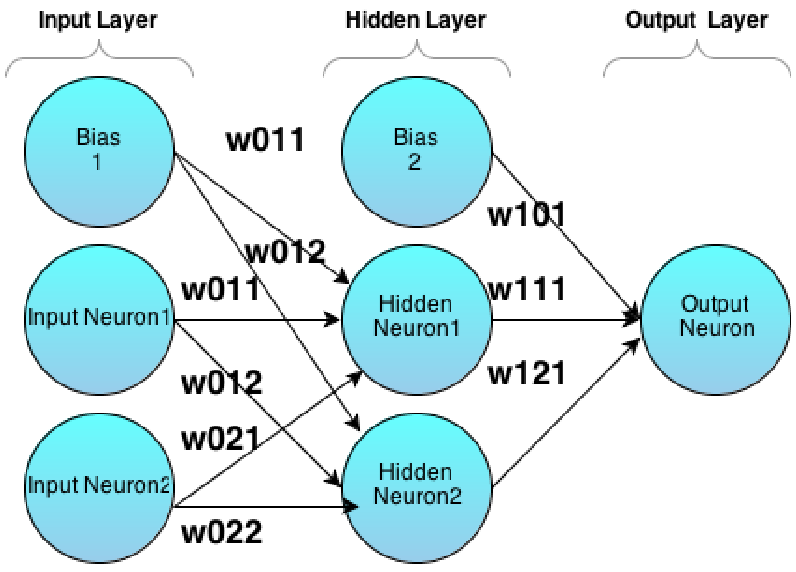


Figure 2.2: Example Artificial Neural Network.

2.1.3.1 Layer

An ANN consists of layers of neurons. Each layer performs independent computations on inputs provided and passes results to the next layer. Traditionally the first layer is always called the *input layer* , the last layer is called the *output layer* and all layers in between are called *hidden layers*. Even a single-layer ANN like e.g. ADALINE(Adaptive Linear Neuron or later Adaptive Linear Element) [1] or Hopfield Neural Network

As a part of this project the author has implemented solution to XOR problem using back-propagation algorithm which will be discussed in Section 4.2.1.2.

It has been proven that ANN with a single hidden layer are able to approximate any function that contains a continuous mapping from one finite space to another [32]. Morphologies with more than one hidden layer are very rare, e.g. Kunihiko Fukushima's Neocognitron used for identifying handwritten characters uses multiple hidden layers [27]. The number of layers

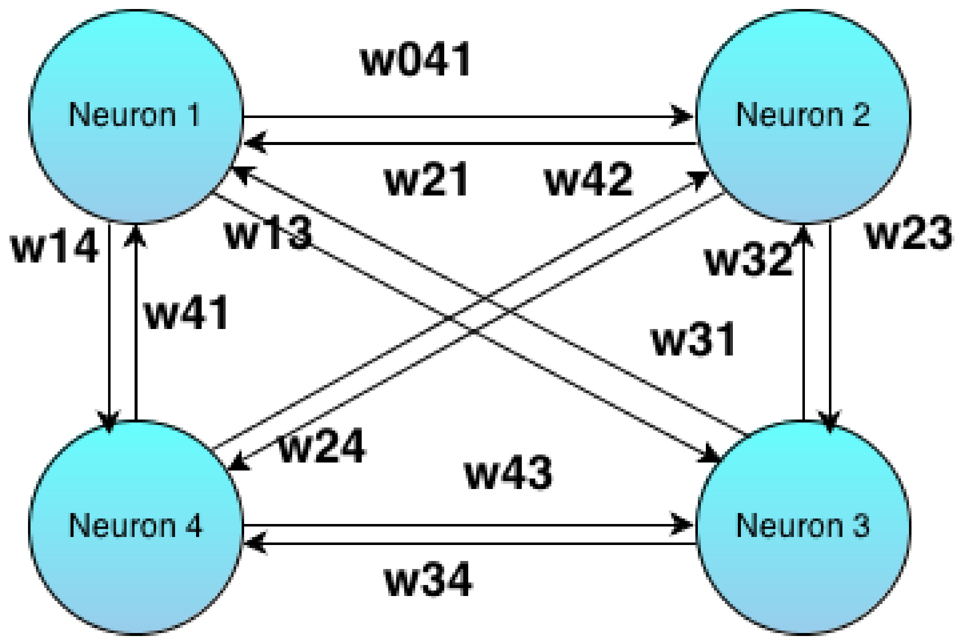


Figure 2.3: Hopfield Neural Network.

as well as neurons in each layer are not known *a priori* because they are dependent on the specific application and must be determined experimentally using adequate fitness measure.

2.1.3.2 Neuron

A neuron (also termed node) is a basic building block of an ANN. The stimulus(weighted input data) is fed into a neuron, processed using activation function and, as result, the output is produced. This output can be treated as input to other neurons an so forth. There can be multiple different weighted input data fed into a neuron but only one output is produced which can be distributed to many other neurons.

In some cases so-called bias neurons are added to an ANN. They always produce output value equal to 1 and they have no input data coming in. Bias neurons allow to shift the activation function when needed. The necessity of bias neurons is easiest to observe with a simple ANN consisting of 1 input neuron and 1 output neuron pictured in the Figure 2.4.

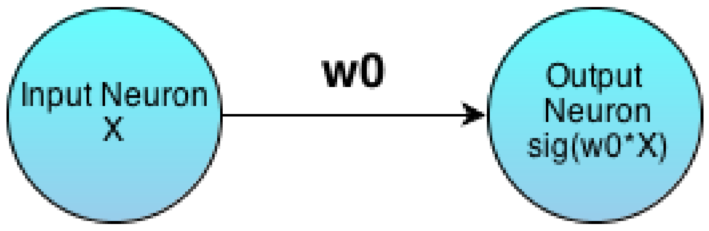


Figure 2.4: ANN consisting of 1 input neuron and 1 output neuron.

Assume that for input value 0 we want to obtain output value 2. Changing the connection weight will only change ”steepness” of the activation function, what we want is to shift the activation function by adding extra bias neuron as in the Figure 2.5.

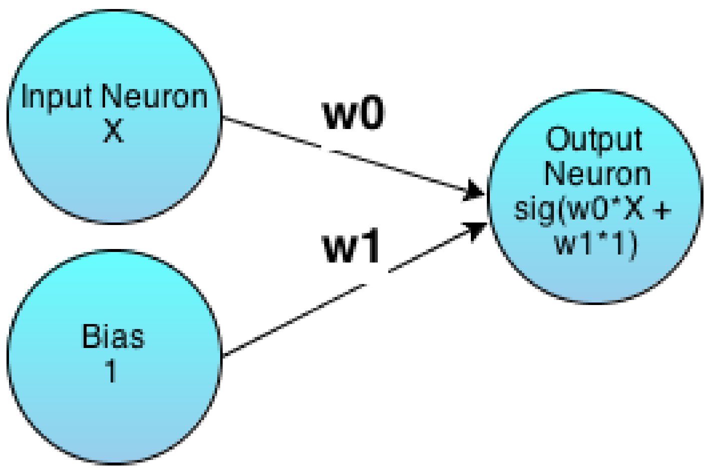


Figure 2.5: ANN consisting of 1 input neuron, 1 bias neuron and 1 output neuron.

The output of this ANN is $sig(w_0 * x + w_1 * 1)$ and for $x = 0$ and $w_0 = 0$ and $w_1 = 2$ we will obtain output value 2.

2.1.3.3 Connections and Weights

Connections are denoted as arrows between neurons.

There are 4 types of connections between neurons:

1. Neurons within one layer are not connected with each other but connected with neurons in the next layer. This is the most common type of connections between neurons which can be observed e.g. in a perceptron.
2. Every neuron is connected to every other neuron in the same layer, e.g. in a Hopfield neural network.
3. In addition to 1. and 2. neurons may have extra connections with each neuron connected to itself, e.g. in a Continuous Time Recurrent Neural Network (CTRNN).
4. In addition to all previous possibilities neurons can be connected to neurons in a previous layer which introduces feedback and short-term memory, e.g. recurrent ANN like Elman neural network.

In general, an ANN learns by updating connections weights during each iteration. Weights indicate not only the strength of the stimulus but also the character of interaction between neurons. Positive weight denotes cooperation and is called *excitation* whereas negative weight denotes competition and is called *inhibition* [56].

2.1.4 Activation Function

Activation function scales output of a neuron into some meaningful range.

Most commonly used activation functions are:

(a) Step Function

$$step(x) = \begin{cases} 1 & if\ x \geq 0 \\ -1 & otherwise \end{cases} \tag{2.1.1}$$

It is most basic activation function which is used when we want to classify argument into one of two groups(binary classification).

(b) Linear Function

$$linear(x) = x \tag{2.1.2}$$

Linear Function is also called a *flow-through mapping* function because it does not modify a sum of weighted inputs. It is like not having any activation function at all hence it is rarely used, mostly for presenting a range of values that inputs can take.

(c) Sigmoid Function

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.1.3}$$

Sigmoid means curved in two directions like the letter "S". Sigmoid function returns value is in range [0,1] meaning it can't return negative values. As a result it can't be used to e.g. forecast percentage movements of time series data. The reason why this function is so commonly used as an activation function lies in the fact that its first derivate (which is used by derive based training algorithms like e.g. backpropagation) is a very neat and simple expression:

$$sigmoid'(x) = sigmoid(x) \cdot (1 - sigmoid(x)) \tag{2.1.4}$$

(d) Hyperbolic Tangent Function

$$hyperbolic(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.1.5}$$

Hyperbolic Tangent function returns value is in range [-1,1] meaning it can be used in ANNs that estimate functions that can return negative values.

In scope of this project for percentage data hyperbolic activation function is used to allow negative values and for other data sigmoid activation function is used.

2.1.5 Training

There are two basic actions possible in ANNs :

- Start with a set of random weights, feed input data to input layer and receive output/s from output layer (Evaluation).
- Start with a set of random weights, feed input data to input layer and receive output/s from output layer, update weights according to some algorithm and feed input data to input layer and receive output/s from output layer. Keep updating weights until satisfactory results will be obtained (Training).

An ANN can be viewed as a *blackbox* tool - data is fed into an ANN and outputs are calculated without knowing the internal mechanism. However obtained results might not be as one would expect. In such cases weights between neurons have to be updated until expected results are obtained. The process of updating weights is called *training*. An *epoch* is a single iteration over all training examples. Training finishes when some predefined conditions are achieved e.g. a number of epochs or the error dropped down to some acceptable level. In other words ANNs learn by updating connections weights between neurons. There are number of standardised training algorithms to be used with ANNs but most popular and effective are Backpropagation, Genetic Training (discussed in next chapters) and Resilient Propagation.

ANN can be subject to supervised or unsupervised learning. In case of supervised learning ANN training algorithm uses expected output (external data) to improve its performance. In other words it uses input data (stimulus) along with expected output to obtain results (ideally the same as expected output) by updating its weights. In case of unsupervised learning which is more complicated that supervised learning, ANNs are provided only with a stimulus. Such ANNs are called *self-organizing maps*(SOMs) and have much more connections between layers and neurons to introduce feedback. SOMs are very efficient in pattern recognising tasks as well as reducing dimensionality of data (e.g. representing multidimensonal data in one dimensional space).

In scope of this project only supervised learning was conducted.

2.1.6 Input Data

An ANN is as good as the input data used to train it. Before input data is fed into neurons it must be pre-processed to equalise the importance of variables. ”Data preprocessing refers to analysing and transforming the input and output variables to minimize noise, highlight important relationships, detect trends, and flatten the distribution of the variable to assist the neural network in learning the relevant patterns” [37]. A pre-processing method should highlight underlying features in training data in order to increase ANN ability to generalize, e.g. derive correct conclusions about the population properties based on training data. Pre-processing methods depend on a problem domain but the most commonly used pre-processing method is normalization.For example, if one input value ranges between 1 and 100, and the next one ranges between 0.1 and 0.001, the ANN will learn to use small input weights for the first input and big input weights for next one to balance signals strengths. However working with such diverse value ranges will decrease network efficiency so it is important to normalize data to small predictable ranges. It is important to remember to denormalize outputs afterwards in order to interpret results correctly.

During this project normalization was chosen as a pre-processing method which is described more in detail in later chapters. For more pre- and post-processing methods please see [50, Chapter 3].

The input data is usually divided into two sets :a *testing* set and a *training* set. A Training set is used to train ANN to approximate desired target function whereas atesting set is used to test its generalization ability. The proportions between those two are determined experimentally however training set is almost always the larger one. If an ANN learns individual properties of the training examples but does not recognizes underlying general features of the data, i.e. performs well for training examples, but fails to produce acceptable results for testing examples, it is said to be subject to *overfitting*.

2.2 Financial Time Series Data

This section discusses financial time series data - its general properties and hypothesis concerning predicting its future values.

2.2.1 Properties of financial time series data

Forecasting financial time series data is a difficult task due to number of reasons [31]:

1. Financial time series are very noisy. Currency exchange rate data is influenced by a number of factors including unemployment rates, interest rate changes, macroeconomic news as well as political news. Those events are random and introduce noise (deviation from standard) in the data.
2. Financial time series are subject to a random walk hypothesis(discussed later) and theoretically correct prediction is impossible.
3. In the long run a forecasting technique becomes invalid. Let's assume that a small group of market participants came up with a revolutionary forecasting technique which yields extraordinary profits. With time this technique becomes widely known and used. The more market participants are using it, the less profitable it becomes. Eventually it will start causing losses and be replaced by another, more profitable technique

2.2.2 EMH: Efficient Market Hypothesis

In 1965 Fama formulated a hypothesis called Efficient Market Hypothesis(EMH) [26]. In essence, EMH states that the current market price is a reflection of all information available. If a new information is added to a system a market price automatically changes to reflect that new addition. Therefore it is impossible for traders to achieve excessive profits by buying undervalued assets or selling overpriced assets because the market price is always fair. Depending on the type of information added to a system, there exist three forms of EMH [31] :

I **Weak Form** - Only past data is considered. A Weak Form implies that a market price follows random walk hypothesis: the more efficient market, the more random the sequence of price changes. In essence, weak form states that any tool based on past data will be no more successful in predicting future values that a coin flip.

II **Semistrong Form** - All available information is considered. This include past data and fundamental data like economic forecasts, company sales forecasts etc.

III **Strong Form** - All public and private information is considered. This part of EMH is hard to test because in practice it is impossible to collect all private and public information influencing a particular market.

The EMH is a controversial hypothesis and a subject of a heated debate in the community. Many researchers, including Tino [49] showed that profits can be achieved by using only past data. Shwaerzel and Rosen [53] proved that random walk hypothesis is not always true and Giles et al. [28] rejects this hypothesis. Research performed in both of those papers includes using ANNs.

2.2.3 Forex: Currency Exchange Rate Market

Forex(foreign exchange market) was chosen as a testing environment because it deals with currencies. Forex is a very liquid, highly volatile market and currencies have a reputation of momentum trading vehicles in which technical analysis has more validity than in other financial markets. Currency exchange rate data is a high dimensional data due to interactions between different countries national economies and consequences of the globalization phenomenon.

Unlike other financial markets, in Forex there is no centralized exchange with just one currency rate value. Forex consists of multiple markets around the world which offer different prices for the same currency pairs. Forex is regarded as a 24/7 financial market because at any given time at least one Forex market is open somewhere.

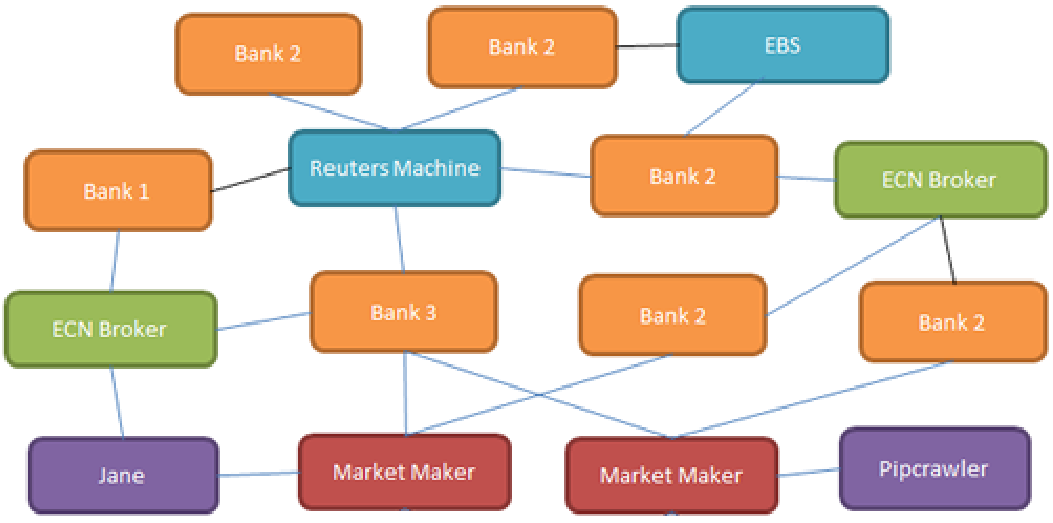


Figure 2.6: Forex structure.

The four biggest Forex markets are located in London, New York, Sydney and Tokio. In scope of this project the historical data is derived from the London market.

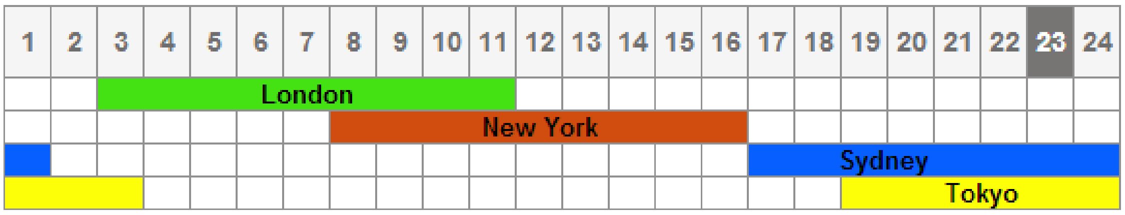


Figure 2.7: Forex markets opening times.

2.2.4 Data used for forecasting financial time series

In general there are three types of data used for forecasting financial time series Hellstorm:

Technical Data Includes close prices, volume , volatility, turning points, technical indicators etc..

Fundamental Data Includes information about current state of a company, industry, national and global economy like inflation, interest rates, unemployment etc...

Derived Entities Combination of two previous ones.

In scope of this project only technical data is used.

2.3 Relevant research

There has been a lot of work done in using ANNs to forecast financial market movements. McCluskey [42] used neural networks to forecast the price increments of S&P500 index. He used a number of different ANN morphologies and amazingly almost all neural networks were able to give a profitable forecast. Moreover he concluded that adding recurrence to ANNs improves their forecasting applicability. Tino et al. [49] confirmed that recurrent neural networks are much more effective than multilayer feedforward networks when it comes to forecasting.

Jingtao Yao et al. [35] used neural networks to forecast currency exchange rates on Forex. They used multilayer feedforward ANNs and trained them using backpropagation with different input data. First model was using raw data of currency exchange rate and the second one was using technical indicators as inputs. Both of them provided satisfactory results.

Castiglione [23] was using neural networks to forecast the sign of price increments on a number of financial market indices including Dow Jones and Nasdaq. He used a number of multi-layered perceptrons with different morphologies to accomplish this task. He concluded that results were a little better than a classic coin toss.

Kondratenko et al. [57] used a recurrent Elman ANN to predict currency exchange rate movements. They concluded that ”Neural networks can predict the increments sign with relatively high probability – approximately 80%...We believe that such quality of forecast is enough for building an automatic profitable trade strategy”. More importantly they provided a number of statistical evidence to support their claim. Kuperin et al. [59] have reached the same conclusion.

As we can see different researchers approach this problem in different ways however all of them agree that using neural networks in forecasting market movements yields better results than a coin toss or linear regression. The quality of results depends on chosen inputs, morphology and training algorithm used.

Chapter 3

Development Process

”Being busy does not always mean real work. The object of all work is production or accomplishment and to either of these ends there must be forethought, system, planning, intelligence, and honest purpose, as well as perspiration. Seeming to do is not doing.”

— Thomas A. Edison

3.1 Life Cycle Model Discussion

A methodology provides an organised approach to work needed to be done by emphasizing most important aspects of every software project. A well chosen methodology could drastically speed up the development process and make the end product more robust and maintainable whereas badly chosen methodology can cause more harm than good. Some methodologies are applicable to a wider range of projects than others ,however there is no ”silver bullet” methodology that can be used for every type of software project. Before the actual work on the project started, the author has investigated a number of software life cycle methodologies that could be used during this project.

3.1.1 Waterfall Model

The author has started his search with the most obvious, a little clichèd nowadays approach - Waterfall Model. The Waterfall Model divides development process in a set of non-overlapping, clearly defined phases:

1. Requirements Specification
2. Software Design
3. Implementation
4. Testing
5. Deployment
6. Maintenance

The Waterfall model provides a sequential, predicable and simple approach. Once all phases are completed, the project is finished and we can move on to new challenges. However this approach is a little bit naive because almost always something unpredictable will happen and revisions of previous, already marked as ”completed” stages, are needed. As mentioned before the author did not have any prior knowledge about ANN and genetic algorithm before the start of this projects hence unexpected perturbations were bound to happen. Moreover, the author felt that the Waterfall approach is too rigid, restrictive and requires too much documentation work to suit this particular project.

3.1.2 Spiral Model

Next methodology considered by the author was the Spiral model. The Spiral model is often characterised as a risk driven approach because it takes for granted that problems will occur and facilitates handling problems. It consist of 4 clearly defined phases:

1. Determine objectives
2. Identify and resolve risks
3. Development and test
4. Plan next iteration

It is an iterative approach - during each iteration all 4 phases must be completed sequentially. The big advantage of the Spiral model over Waterfall model is that if during work on one phase some problems with the previous phase were identified, it can be fixed during next iteration.

One of the disadvantages of this approach is that it is hard to make estimates of how long each iteration can take. In scope of this project where some milestones and deadlines must be reached(e.g. supervisors expect some particular functionality to be ready every week) using Spiral model might not be the best option. Moreover, the author feels that this methodology is more suited for large project where every unexpected change (especially during final iterations) is costly to accommodate.

3.1.3 Iterative and Incremental Development Model

The iterative nature of the Spiral model shifted the author’s attention to the Iterative approach. The most popular iterative approach is called Iterative and Incremental Development(or Iterative Development with Incremental Delivery). In general the project is split into a number of iterations following the same development phases. The working versions of software are released several times, each time with more functionality. Each release is complete, usable and useful to its users. Each release adds more functionality, preferably the most important functionality first.

The author has decided that this methodology is well suited for this project because it supports the idea of ”growing software”. In scope of this project each new ANN morphology or a training algorithm will build on already existing functionality and add more features to the overall product. Although some of those ANN models will provide a completely new functionality they will have to fit into the general framework.For example, all models are using different parameters which must be added to the GUI and underlying structures. As

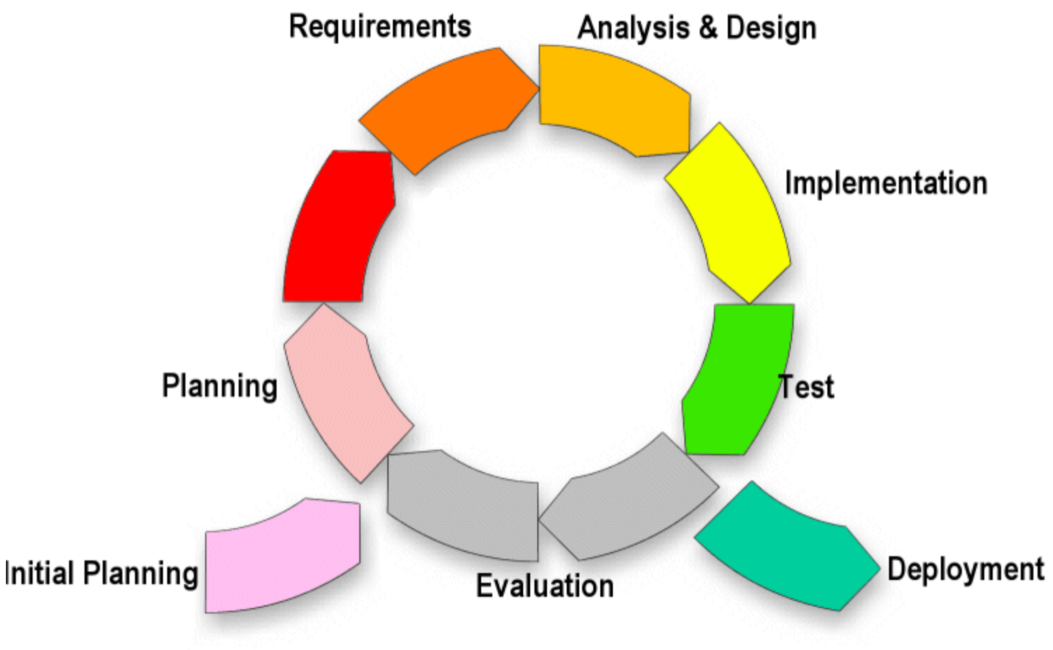


Figure 3.1: Iterative and Incremental Development Lifecycle.

a result with each new model introduced to the existing design will have to be updated and thoroughly tested to make sure that no new bugs were introduced.

3.1.3.1 Initial Planning

During this phase the author has met with Dr Elio Tuci to discuss possible ANN morphologies and training algorithms that can be implemented during this project. Once the list was complete, the author extracted common features of all models and went through a couple iterations of analysis and design. That resulted in the basic design described in section 4.4.1. Moreover, the author has performed research concerning ANNs used for prediction tasks, especially financial time series data. The author considers this stage the most important one because its results had influenced directly or indirectly all following iterations.

3.1.3.2 Iteration 0: Metatrader 5

Metatrader 5 platform was the author's initial choice as implementation environment, however due to various reasons(see Section 5.1.2) it was not used. Nonetheless the knowledge acquired during this iteration proved invaluable when working on data acquisition.

3.1.3.3 Iteration 1: XOR ANN

During this iteration a multilayer feedforward ANN was implemented to solve non-linearly separable XOR problem (see Section 4.2.1). Moreover, the basic version of backpropagation training algorithm was implemented.

3.1.3.4 Iteration 2: Input and Output Selection

One of the most important decision in ANN design is to choose inputs that will produce best outputs. During this iteration, the author has selected a number of technical indicators which were tested to determine which have the highest correlation with the output variable(close price value). The findings of this iteration were summarised in Section 6.2.

3.1.3.5 Iteration 3: Data Aquisition

One of the most important decision in ANN design is to choose inputs that will produce best outputs. During this iteration, the author has selected a number of technical indicators which were tested to determine which have the highest correlation with the output variable(close price value). The findings of this iteration were summarised in Section 6.2.

3.1.3.6 Iteration 4: Data Pre- and Post-Processing

In order to optimize an ANN performance the data fed into ANN must be pre-processed and, as a result, the outputs must be post-processed to return them to original form. The author has used min-max normalization process for data processing(see Section 4.1.3).

3.1.3.7 Iteration 5: Data Partitioning

Data used by ANNs is split into at least two sets: training set used for training and testing set used for testing(evaluation). During this iteration the author implemented the algorithm responsible for data partitioning and tested a number of training/testing ratios((see Section 4.1.4 and section 6.2.5).

3.1.3.8 Iteration 6: Data Timeboxing

During this iteration the author has worked on splitting the input data so that it can be fed into the ANN sequentially. The order of the input data is of high importance and must not be altered because ANNs are looking for patterns in a sequence of data(see Section 4.1.3).

3.1.3.9 Iteration 7: Evaluation Measures

During iteration 1 some simple evaluation measures were implemented to test XOR ANN performance. However with ANN working on time series data more complex measures are needed. During this iteration a number of new evaluation measures were implemented and tested(see Section 4.1.6).

3.1.3.10 Iteration 8: GUI

Due to forthcoming mid-project demonstration the GUI had to be implemented. This work involved implementing charts for results visualization as well. For presentation purposes added new functionality which allowed user to query the ANN without providing expected results.

3.1.3.11 Iteration 8: Adding Recurrence

In order to implement Elman recurrent ANN, the existing design had to be modified. Not only the general data structures had to be revised but the backpropagation algorithm had to be updated to deal with extra neurons. The work needed to complete this task was much shorter than expected so another type of ANN morphology was added as well: Jordan recurrent ANN(see Section 4.2.2).

3.1.3.12 Iteration 9: Adding Genetic Training

During this iteration the genetic algorithm based training was designed, implemented and tested. More changes in existing design were needed to accommodate this training algorithm. A throughout testing of each genetic operator (selection, crossover, mutation) was performed(see Section 4.3.2).

3.1.3.13 Iteration 10: Adding Continous Time Recurrent Neural Network

Adding CTRNN caused many problems because this ANN is different in many fundamental aspects for previously implemented models(see Section 4.2.3 for more information). It took the author a lot of time to understand how this ANN work and due to limited time nor all aspects were investigated as close as they should be.

3.2 Modifications

In general, the author thinks that Iterative and Incremental Development approach worked very well in scope of this project. One of the main advantages of using this methodology was that after each iteration a working software was produced. This allowed the author to present project progress during weekly meetings with supervisor easily.

The author expected that due to his small knowledge of ANNs the design will have to be often changed. Iterative and Incremental Development facilitated this requirement perfectly. The author thinks that this flexibility greatly contributed to the success of this project. Moreover the author had successes and failures of previous iterations in mind when working on new iterations which resulted in faster development and more robust and maintainable software.

During some parts of the project the author was using a prototyping life cycle to speed up the development process. During exploration of Metatrader 5 capabilities, a prototype of a simple ANN was implemented using MQL5 programming language. This prototype was crucial to the future of this project. Looking back, the author feels that it would be impossible to implement all ANN models using MQL5 language in a given time frame because of e.g. missing debugging functionality of Metatrader Editor. When preparing for the mid-project presentation the author has implemented a prototype of GUI which would be updated with more controls and functionality afterwards.

3.3 Development Tools

This project was implemented using Java programming language. Java is a well documented, object oriented language with a huge online community. In order to facilitate rapid development Eclipse IDE was used as a programming environment. Eclipse provides excellent debugging capabilities, a wide range of plugins that expand general functionality, auto completion and many other useful features. In scope of this project when tracking very small or very large values is inevitable, the Eclipse made this task much less daunting and cumbersome.

A free WindowBuilder Eclipse plugin was used to create the initial GUI. Although it speeded up the GUI development in project's early stages it proved to be a nightmare to maintain the GUI afterwards. WindowBuilder provides a very useful visual tool to develop GUI, however the reverse engineering feature does not work so well. Due to design issues the author had to make some alterations in code generated by the WindowBuilder however the plugin was unable to reverse engineer these changes and apply it to the visual interface. As a result, the WindowBuilder became unusable and all future modifications had to be made explicitly by the author.

A free ObjectAid plugin was used to help to create UML class diagram. Although ObjectAid does not generate class diagrams automatically it makes creating them very straighforward and intuitive by dragging and dropping all java classes onto a diagram from Eclipse views. More importantly it automatically updates this diagram if any changes are detected in the corresponding java code.

Analyzing results in a form of a string of numbers is difficult and troublesome for a human, whereas analysing charts is much easier. After some online research,it turned out that there a number of free chart drawing libraries for Java however most praised one was jfreechart. JFreechart stands out because of a well-documented API and a wide range of chart types. From this project point of view, jfreechart provides excellent support for viewing time series data. The only weakness that the author could identify was the lack of 3D charts(which are currently under development). JFreechart is released under GNU Lesser General Public License 2.1 meaning that it is free to use for the purposes of this project.

Other tools used during this project include:

- Apache commons libraries were used to simplify CSV file processing.
- Dropbox was used to store copies of the project in case of a hardware failure or other problems.
- Testing tables were created using Microsoft Office Excel software.

Chapter 4

Design

”Design is not just what it looks like and feels like. Design is how it works.”

— Steve Jobs

4.1 Design Process

Designing an ANN is a complex task requiring a number of important decisions to be made. Each decision have a big impact on the overall success of the project and even a single wrong decision can cause a project to fail.

During work on this project the author has followed a specific design process to ensure that even the smallest choice made is reasonable and justified:

Step 1 Input and output selection.

Step 2 Data acquisition.

Step 3 Data pre- and post-processing.

Step 4 Data partitioning.

Step 5 Data timeboxing.

Step 6 Evaluation.

Step 7 Morphology choice.

Step 8 Training.

Step 9 Implementation.

4.1.1 Step 1: Input And Output Selection

As pointed out before ANN model is as good as data used to train it. Inputs should provide as much relevant information to output as possible in order to reduce data high dimensionality. It is an ANN task to find relationships between different variables , however ANN designer should use a problem domain knowledge to use variables that are somehow related. For example, interest rate will not tell us much about weather next week but it will provide information about a general state of the national economy. Variables that carry little information or this information is already included in other variables should be discarded. There is no algorithm to determine which sets of variables will produce best results, hence a designer have to rely on a problem domain knowledge and knowledge gained through carrying out experiments.

This project utilizes two types of inputs/outputs:

Raw data In a form of a time series data with a currency pair close price value.

Technical indicators Used by real traders in a technical analysis of currency exchange rate data (e.g. Moving Average(AV) or Stochastic RSI Oscillator). The reason behind using technical indicators is that they are very helpful in detecting trends (patterns) in data which helps in predicting future price movements.

4.1.1.1 Outputs

In scope of this project the output variable is always the close price value of particular currency pair. This is the most obvious choice since all trades in Forex are based on buying and selling currency pairs based on their close price. Close price is the price of a last trade executed in a given time window. It is important to note that close prices will differ between brokers and currency exchange markets(e.g. London, Tokio, New York). The author decided that all ANN models will try to forecast only one bar(a single time window) value in the future. The reason behind it is that when trying to predict more than one bar in the future, effectively ANN is using own predictions to create new predictions hence the quality of those predictions is degrading fast.

4.1.1.2 Inputs

Data sets consisting of samples sampled with small frequencies include much more noise that data sets with samples sampled with higher frequency which are more smoothly distributed(standard deviation is much smaller). Author has chosen 15 minutes interval for presentation purposes (it takes much less time to demonstrate than ANN can make a reasonable prediction) and 1 hour and 1 day interval for comparison purposes because they are the most common interval used in other scientific publications.

In scope of this project the author researched a number of currency trading forums including Metatrader 5 forum [12], BabyPips [2], Forex Factory [5], Trade2Win [17] and many others to determine best inputs to use. It is important to note that a close price value of previous bars ($CP_(t), CP_(t-1)...CP(t-n)$) is used as an input to predict the future close price value ($CP_(t+1)$). However the close price value will not be considered in a vacuum and other inputs (technical indicators) will be used to help to detect patterns in data. As described previously the main advantage of technical indicators is that they encompass a lot of information describing the structure of a data in a single number. If only raw data would be used as inputs to the ANN(e.g. price volatility, volume, turning points) the number of inputs would be immense and dimensionality of data very high.

Number of tests were carried out to determine a set of technical indicators that produce most accurate preditions of a currency close price value. As a result following technical indicators were chosen:

MACD ”Developed by Gerald Appel in the late seventies, the Moving Average Convergence-Divergence (MACD) indicator is one of the simplest and most effective momentum indicators available. The MACD turns two trend-following indicators, moving averages, into a momentum oscillator by subtracting the longer moving average from the shorter moving average. As a result, the MACD offers the best of both worlds: trend following and momentum” [11].

Moving Average(MA) ”An indicator frequently used in technical analysis showing the average value of a security’s price over a set period. Moving averages are generally used to measure momentum and define areas of possible support and resistance.” [10].

Relative Strength Index (RSI) ”Developed J. Welles Wilder, the Relative Strength Index (RSI) is a momentum oscillator that measures the speed and change of price movements. RSI oscillates between zero and 100... RSI is an extremely popular momentum indicator that has been featured in a number of articles, interviews and books over the years.” [15]

Stochastic K ”A technical momentum indicator that compares a security’s closing price to its price range over a given time period. The oscillator’s sensitivity to market movements can be reduced by adjusting the time period or by taking a moving average of the result...The theory behind this indicator is that in an upward-trending market, prices tend to close near their high, and during a downward-trending market, prices tend to close near their low.” [16]

Stochastic Slow D ”%D is 3-period moving average of %K... Transaction signals occur when the %K crosses through a three-period moving average called the %D” [16]. To sum up Stochastic K and Slow D indicators should be used together to generate buy/sell signals.

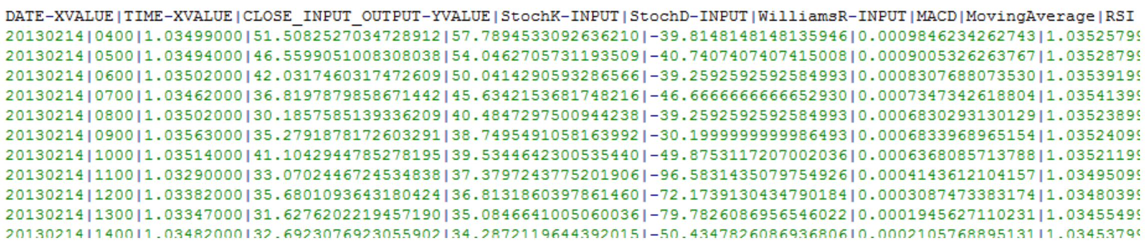
Williams R ”Developed by Larry Williams, Williams %R is a momentum indicator that is the inverse of the Fast Stochastic Oscillator. Also referred to as %R, Williams %R reflects the level of the close relative to the highest high for the look-back period. In contrast, the Stochastic Oscillator reflects the level of the close relative to the lowest low.” [20]

4.1.2 Step 2: Data Acquisition

The author used Metatrader 5 [8] platform to collect historical data. Metatrader 5 allows users to set up a free demo account and trade with virtual money and export historical data for free. Data is imported in a .csv file using a dedicated script written in MQL5 programming language which is discussed in section A. Each file contains information regarding a single currency pair and all inputs and outputs.

The first row defines the order in which data is organised. Values used as inputs to an ANN are marked by the INPUT header, outputs are marked by the OUTPUT header. Additionally data from the CSV file is used to create currency pairs charts in a Cartesian space where x-coordinate is time and y-coordinate is a close price value. Values used as x-coordinates are marked by the XVALUE header (dates and time) and values used as y-coordinates are marked by the YVALUE header. A single information can be used for multiple purposes hence it can have multiple headers.

Through a number of experiments the author determined that a single data set size should be equal to approximately 1000 observations for best results. This number reflects an equilibrium state between data sets with too many observations which take too much time to train



```
DATE-XVALUE|TIME-XVALUE|CLOSE_INPUT_OUTPUT-YVALUE|StochK-INPUT|StochD-INPUT|WilliamsR-INPUT|MACD|MovingAverage|RSI
20130214|0400|1.03499000|51.5082527034728912|57.7894533092636210|-39.8148148148135946|0.0009846234262743|1.0352579
20130214|0500|1.03494000|46.5599051008308038|54.0462705731193509|-40.7407407407415008|0.0009005326263767|1.0352879
20130214|0600|1.03502000|42.0317460317472609|50.0414290593286566|-39.2592592592584993|0.0008307688073530|1.0353919
20130214|0700|1.03462000|36.8197879858671442|45.6342153681748216|-46.6666666666652930|0.0007347342618804|1.0354139
20130214|0800|1.03502000|30.1857585139336209|40.4847297500944238|-39.2592592592584993|0.0006830293130129|1.0352389
20130214|0900|1.03563000|35.2791878172603291|38.7495491058163992|-30.1999999999986493|0.0006833968965154|1.0352409
20130214|1000|1.03514000|41.1042944785278195|39.5344642300535440|-49.8753117207002036|0.0006368085713788|1.0352119
20130214|1100|1.03290000|33.0702446724534838|37.3797243775201906|-96.5831435079754926|0.0004143612104157|1.0349509
20130214|1200|1.03382000|35.6801093643180424|36.8131860397861460|-72.1739130434790184|0.0003087473383174|1.0348039
20130214|1300|1.03347000|31.6276202219457190|35.0846641005060036|-79.7826086956546022|0.0001945627110231|1.0345549
20130214|1400|1.03482000|32.6923076923055902|34.2872119644392015|-50.4347826086936806|0.0002105768895131|1.0345379
```

Figure 4.1: Example Data File.

and data sets with a smaller number of observations which might not be enough to identify patterns in a data set.

Some data sets have visible breaks between observations which denote weekends. Moreover it can be noticed that the last price on late Friday is different than the first price after weekend on late Sunday(when Asian markets are opening and a new week starts). Although Forex is a 24/7 financial market (see section 2.2.3) most of broker companies which in fact are entities who buy/sell currencies in the name of a customer are closed during weekends. Some broker companies trade during weekends but they charge their customers extra for this possibility. In case of live accounts a user can choose a broker company to represent him/her, unfortunately in case of demo accounts the choice is much smaller. In case of Metatrader 5 platform the only broker company available for demo accounts is MIG Capital [9] which does not trade during weekends hence missing observations.Due to the fact that the activity on Forex market during weekends is very small currency prices almost do not change between the end and the beginning of a work week. As a result, from the ANN training point of view missing observations have little or no effect on the quality of predictions.

4.1.3 Step 3: Data Pre- and Post-processing

As described in section 2.1.6 in order to make ANN more effective and efficient a data fed into it must be pre-processed. As a result outputs of ANN must be post-processed to convert data to its original form. Data pre-processing improves ANN performance through optimizing benefits coming from using particular activation functions (discussed in more detail in section 4.1.7.5). The most common process used for data processing and the one used during this project is normalization.

Normalization is a mathematical process that converts numbers into some desired range. Denormalization is a process that converts normalized numbers into their original form. The author has used max-min normalization in this project [50, p. 16]. In order to perform max-min normalization, peripheral data points of a data set must be identified. The normalization process is described by Equation (4.1.1) and the denormalization process is described by Equation (4.1.2).

$$x'_i = (max_{target} - min_{target}) \cdot \left[\frac{x_i - min_{value}}{max_{value} - min_{value}} \right] + min_{target} \tag{4.1.1}$$

$$x_i = \frac{(min_{value} - max_{value}) \cdot x'_i - max_{target} \cdot min_{value} + max_{value} \cdot min_{target}}{min_{target} - max_{target}} \tag{4.1.2}$$

All inputs are normalized to range [0,1]. It is important to note that normalization process neither affects relationships between variables in a data set nor introduces any bias(preference for some values). Underlying distributions of input variables stay intact.

4.1.4 Step 4: Data Partitioning

In order to maximize ANN performance historical data was divided into two non-overlapping data sets :

I **Training Set** - this data set is used by an ANN to learn patterns present in data and, as a result create a model of data. This set is used to train an ANN by adjusting weights(a typical ANN) or changing states of neurons(CTRNN). There is no general algorithm to determine the best size of a training set because it is greatly dependent on a problem domain. In principle this should be the largest data set of all. In scope of this project the training set is equal to 90% of all data set examples by default which was determined experimentally to yield best results.

II **Testing Set** - this data set is used to evaluate ANN generalization ability. In case of ANN that learn by adjusting their connection weights, when ANN processes testing data set examples weights stay fixed. Similarly to training set, there is no general algorithm to determine the best size of a training set because it is greatly dependent on a problem domain. In scope of this project the training set is equal to 10% of all data set examples by default which was determined experimentally to yield best results.

In some applications another subset of data is used called *validation set*. It's main purpose is the same as in case of testing set - to evaluate an ANN performance over some set of data it did not used during training, however it is used after training and before final evaluation. It is used to find the best morphology and parameters of ANN. In other words it is used to choose best ANN model from a number of models. Once best ANN is chosen, it is tested using a testing set. For example, an automated ANN-based trading agent would verify its predictive abilities using a subset of historical data(validation set) however the final testing would be done on live data(testing set). In scope of this project validation and testing sets have the same purpose hence the author decided to use only testing set.

4.1.5 Step 5: Data Timeboxing

Once historical data is extracted from a CSV file and normalized it needs to fed into ANN. In scope of this project a timeboxing technique was used to present data to ANN. Timeboxing creates moving slices of data, it can be imagined as a moving box of data moving forward on a time axis. The only parameters required are the size of the INPUT WINDOW i.e. how many bars (single observations in a unit of time) will be used (how big a moving box will be) for a prediction, and the size of the OUTPUT WINDOW i.e. how many bars are going to be predicted. In scope of this project only 1 bar is always predicted (see section 4.1.1.1) hence the value of OUTPUT WINDOW is always 1.

For example, let's assume that we have 2 different input time series and one output time series. The order of observations is determined by the time in which they were taken so the first observation of each time series input was made in t_0 and the nth observation was made in t_n. For example, if the INPUT WINDOW value is equal to 3 that means that 3 observations

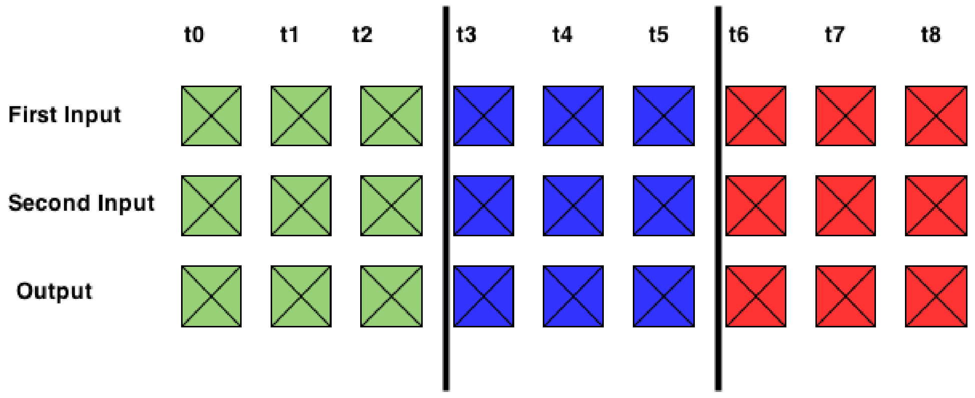


Figure 4.2: Timeboxing.

of each time series will be selected beginning from the first observation to predict the value of the 4th observation of the output time series. During next epoch a box will move one bar forward so input observations will be made in t_1, t_2, t_3 to predict the 5th observation of the output time series. The process will continue until the end of the output time series is reached.

4.1.6 Step 6: Evaluation

ANNs evaluation will be performed using statistical methods and time needed to train an ANN. As noticed by Jao and Tan [35] there are no dedicated tools to measure ANN performance, hence we have to rely on statistical tools measuring the goodness of fit. Statististical methods used in this project were taken from [57]:

Mean Squared Error (MSE) - the mean of differences between actual and predicted values. This is probably the most commonly used statistical method to evaluate the quality of estimation or prediction.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2, \tag{4.1.3}$$

where y_i id the actual value, $\hat{y}$ is the predicted value. In the scope of this project MSE is used for 3 purposes:

- ANN performance evaluation.
- Backpropagation termination criteria (see section **??**).
- Genetic Algorithm based training fitness measure (see section **??**).

Min Absolute Error Predicted values over all patterns subtracted from the minimum of the actual value.

Max Absolute Error Predicted values over all patterns subtracted from the maximum of the actual value.

R^2 - The coefficient of multiple determination. It compares the ANN predictions to the predictions of a simple benchmark model - the mean of all samples. If ANN predictions are worse than predictions that could be obtained using the mean of all samples, R^2 value is less than 0. R^2 value of 1 means a perfect fit and near 1 means a very good fit.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2} \tag{4.1.4}$$

where y_i id the actual value, $\hat{y}$ is the predicted value of y and $\bar{y}$ is the mean of all y_i values.

As Kondratenko et al. [57] noticed the most representative statistical estimate of forecast quality among of all mentioned above is R^2 which gives information on both accuracy and correlation.

It is important to note that both MSE and R squared are calculated on data in its original (denormalized) form. They are calculated separately for training data and testing data to differentiate between ANN performance accomplished during training and testing(evaluation) phase.

4.1.7 Step 7: Morphology choice

The general ANN structure - how neurons are connected, is imposed by the models chosen by the author to be implemented (discussed in Section 1.2). However those models do not define all information needed to build a successful ANN. The number of layers, number of neurons in each layer or activation functions are different for different problem domains and have to be determined experimentally. As discussed in 2.1.3.1, in scope of this project an ANN with 3 layers (input, hidden,output) is sufficient to produce accurate predictions. Nonetheless the amount of neurons in each layer or activation function choice is a much more complex decision.

4.1.7.1 Input Neurons

Number of input neurons should be equal to number of input variables presented to an ANN. In case of this project the number of input neurons equal to the product of the timeboxing's input window parameter and number of different input types used(4.1.5).

$$input\ neurons = input\ window \cdot inputs \tag{4.1.5}$$

For example, if there are 4 different types of input used and input window is equal to 4 bars, 16 input neurons will be needed in an input layer of an ANN.

4.1.7.2 Hidden Neurons

Hidden neurons are the most important neurons in an ANN especially in a 3 layer ANN where almost all calculations are based on connections coming to and from hidden neurons. Too few hidden neurons results in a phenomenon called *underfitting* - there are not enough neurons to adequately detect patterns in a data set. Too many hidden neurons can lead to *overfitting* (see section 2.1.6 for explanation). Moreover it will increase a computation overhead considerably and, as a result the time needed to train an ANN.

There is no universal formula for selecting optimal number of hidden neurons, therefore researchers fall back to trial and error. Nevertheless, some rules of thumb exist:

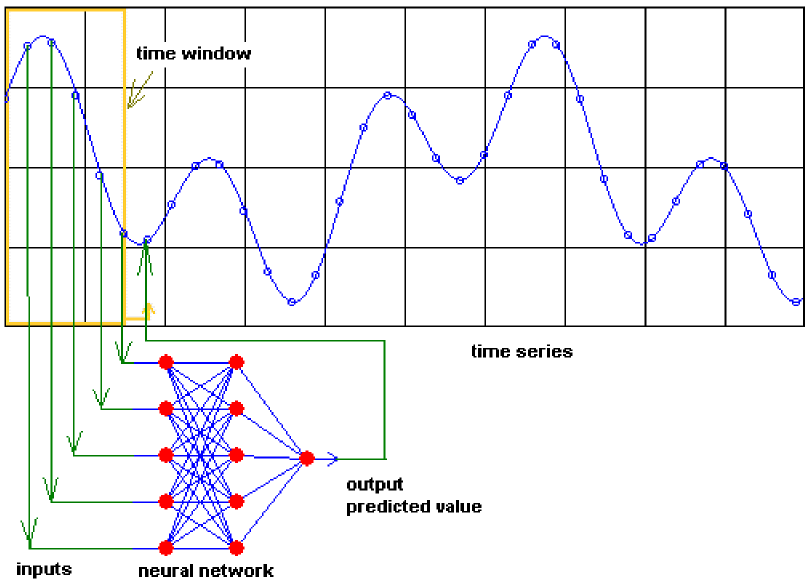


Figure 4.3: Predicting Time Series Data

- In a 3 layer ANN number of hidden neurons should be equal to 75% of the number of input neurons [22].
- There should be at least 5 times as many training examples as connection weights, which sets an upper limit on the number of both hidden and input neurons [39].
- Geometric pyramid rule: in case of a 3 layer ANN with k input neurons and l output neurons, the number of hidden neurons should be equal to $\sqrt{k \cdot l}$ [41].
- Number of hidden neurons should be between 50% and 300% of number of input neurons [38].
- Number of hidden neurons should be doubled until ANN performance decreases during testing [25].

In scope of this project the author has used above rules of thumbs as a starting point in determining optimal number of hidden neurons.

4.1.7.3 Output Neurons

As discussed in section 4.1.5 ANNs will always try to predict 1 bar (a single numeric value), hence only one output neuron is needed. In general, a single output neuron is used in ANNs used for regression problems whereas more output neurons are utilised in ANNs performing classification tasks.

4.1.7.4 Initial Weights

At the beginning of each training connections weights are initialized to random values. Training algorithm will adjust those values during each epoch multiple times(with each training example). In theory, eventually weights should obtain values that will estimate the target function without error. In practice training finishes when the results are good enough. The starting connections weights values affect the speed at which ANN obtains good enough results. Weights values are uniformly drawn from a range of numbers calculated by formula (4.1.6).

$$weight = ((random(0,1) \cdot x) + y), \tag{4.1.6}$$

where x is a scaling factor and y is a shifting factor.

After some experimentation and consultation with Dr Tuci, the author has decided that initial connections weights values will be randomly(uniform distribution) selected from range (-0.25, 0.25). The reason behind those values is that the data set is fairly big (1000 examples) and contains a big number of patterns which must be identified by the same ANN. Moreover the inputs and outputs are normalized to range (0,1) for non-percentage data meaning that connections weights will be of similar magnitude.

4.1.7.5 Activation Function

Section 2.1.4 described the most common activation functions used in ANN development.In scope of this project two activation functions were used:

- Sigmoid Function - because it is applicable to non-linear data like financial time series and is differentiable which is important for some training algorithms, e.g. backpropagation. Most of ANN models in this project are using sigmoid activation function.
- Hyperbolic Tangent Function - because it can return negative values(compared to a sigmoid function which returns only positive values) and is differentiable. In scope of this project ANN models which work on percentage data (can be negative) are using hyperbolic tangent activation function.

Both sigmoid and hyperbolic tangent functions have so-called *active range*- the area of the function curve where the derivative of the function is clearly non-zero. For example, active range of a sigmoid function is $(-\sqrt{3}, \sqrt{3})$ - in this range changes in input values result in relatively large changes in output values. Any input value change outside of this active range (near the asymptotic ends of the sigmoid function) will have small impact on output value. Moreover the derivatives of values outside sigmoid active range are almost zero. This is important for training algorithms using derivates, e.g. backpropagation where derivates close to zero result in very small changes in connection weights (very slow or no learning). That is why normalization of inputs improves ANN performance - inputs normalized to a subset of an activation function's active range will result in larger updates of connection weights which will speed up a training process.

4.1.8 Step 8: Training

Once input data is chosen, collected, processed and ANN morphology is chosen, it is time to start training. The main goal of training is to find ANN parameters that will maximize an ANN generalization ability. An ANN is trained using training data (*in-sample* data) but evaluated using testing data(*out-of-sample* data).

The biggest possible problem of all training algorithms is the overfitting phenomenon(see section 2.1.6). The solution to overfitting is to reduce the number of hidden neurons and/or increase the size of training data set [37].

There are number of training algorithm available, however due to limited time the author has decided to implement two most popular and acclaimed algorithms: Backpropagation and Genetic Training. Both of them are described in more detail in section 4.3.

4.1.9 Step 9: Implementation

Although the implementation is listed as the last step, in fact during each design step the author has considered how to implement necessary functionality. Starting from technologies to be used through statistical tools to the nitty-gritties of training algorithms, the implementation must be the realization of design.

Apart from implementing ANN models, an application used for testing these models have to be implemented. The application (named ”Neural Network Mentalist” by the author) will work in two main modes: training and evaluation. It is important to note that during evaluation phase an ANN does not learn, it just calculates outputs, so the weights are not updated. The user will have the possibility to query ANN with input of his own choosing as well.

4.2 ANN Morphologies

This section discusses ANN morphologies designed, implemented and tested for the purpose of this project.

4.2.1 Multilayer Feedforward Neural Network

4.2.1.1 Basic Definition

Multilayer feedforward ANNs (MFANNs) are the simplest and most commonly used ANNs. The word ”multilayer” comes from additional hidden layers located between input and output layer. Those ANNs are called feedforward because the output of any layer is always passed forward to the next layer(apart the output layer). There are neither backward connections nor connections never skip a layer. Generally the layers are fully connected, meaning that all neurons in one layer are connected with all neurons in the next layer.

4.2.1.2 XOR Problem

Predicting XOR logical function values using a MFANN is sort of a ”Hello World!” application in the world of ANNs.

Unlike other boolean functions like OR,AND, NOR or NAND which can be represented by a perceptron model(no hidden layers), the XOR requires extra hidden layer to return correct results. A simple perceptron cannot evaluate XOR function because the data set examples are not linearly separable. This can be easily proven by trying to design a perceptron that can approximate XOR function.

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Figure 4.4: Logical XOR Truth Table.

There are 2 inputs and 1 output in XOR function hence 2 input neurons and 1 output neuron are needed. As a result there would have to be two connections weights w_1 and w_2 and a threshold value σ. The weighted sum of all inputs should exceed the threshold value to ”fire”. The ”fire” term comes from the neurobiology - if the cell receives enough input from other neurons, it fires and sends an electrical signal down its axon which can in turn induce other neurons to fire. The table 4.1 shows what conditions need to be satisfied by weights and the threshold in order to estimate XOR function.

Table 4.1: Perceptron and XOR function

Input	Weighted sum of all inputs	Output	Condition
0,0	0	0	$0 < \theta$
1,0	w_1	1	$w_1 > \theta$
0,1	w_2	1	$w_2 > \theta$
1,1	$w_1 + w_2$	0	$w_1 + w_2 < \theta$

In order to receive the output value 0 for two inputs equal to 0 the following condition must be fulfilled: $0 < \theta$. In other words the output neuron will ”fire” (return 1) only for a weighted sum of all inputs larger than 0. It is easy to work out next three conditions. Based on the first three conditions it can be deduced that $w_1 + w_2 > \theta$ which is in contradiction with the last condition: $w_1 + w_2 < \theta$. This proves that a perceptron is unable to approximate XOR logical function.

The author designed following MFANN to solve XOR problem:

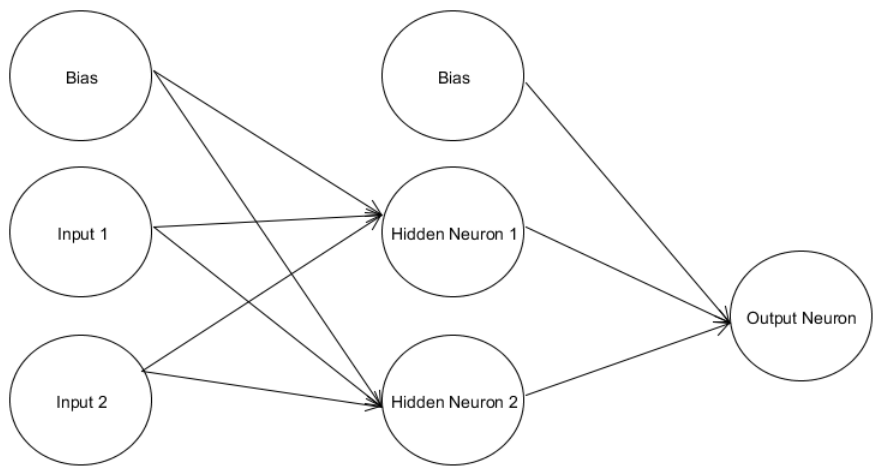


Figure 4.5: XOR Neural Network.

Adding extra hidden layer introduces a number of new connections weights between inputs neurons and the output neuron. More connections weights (degrees of freedom) increase the ”plasticity” of the model meaning that it can learn more patterns.

While working on this simple model the author has created the general design of a multi-layer feedforward ANN and implemented the basic version of the backpropagation training algorithm.

4.2.2 Recurrent Neural Networks

4.2.2.1 Basic Definition

The fundamental feature of Recurrent ANNs (RANNs) is that they introduce feedback into the system. Compared with MFANNs where only forward connections between two layers are possible, in RANNs any neuron can be connected with every other neuron and even with itself(self-recurrent). The neurons in higher layers feed neurons in lower layers which using so-called feedback connections. This is more biologically plausible than in case of MFANNs because it resembles the way in which human brain neurons are connected.

In general, the connections weights in ANNs create a long-term memory - their values are modified based on experience during multiple epochs. RNNs use output values of neurons in future epochs through backward connections. These backward connections create short-term memory. The RANN is able to recall the outputs of neurons from previous epochs and use it as inputs during the current epoch. Addition of a short-term memory gives RANNs a big advantage over MFANNs: they are able to recognize its current input within the context of previous input given and, as a result, identify more complex patterns in the data.

4.2.2.2 Elman Recurrent Network

The first RANN implemented in scope of this project was Elman RANN developed by Jeffrey L. Elman in 1990. Apart from standard 3 layers of neurons it uses 1 extra layer of neurons called context layer. The number of neurons in the context layer is equal to the number of neurons in the hidden layer. Neurons in the context layer are fed outputs from the hidden layer without weighting. During the next epoch the Elman network will use context layer output as input to the hidden layer but with weighted connections. This creates a short-term memory.

4.2.2.3 Jordan Recurrent Network

Jordan RANN was developed by Michael I. Jordan in 1986. Similarly to Elman RANN, Jordan RANN adds extra context layer to the structure of a network. The number of neurons in the context layer is equal to the number of output neurons. Jordan neurons are fed outputs from the output layer without weighting. During next epoch, jordan neurons feed hidden neurons using weighted connections which introduces short-term memory to the system.

Initially the author did not plan to use this particular ANN topology ,however once Elman RANN was implemented adding Jordan RANN was very straightforward. More importantly, Elman and Jordan RANNs can be used together which results in more powerful short-term memory.

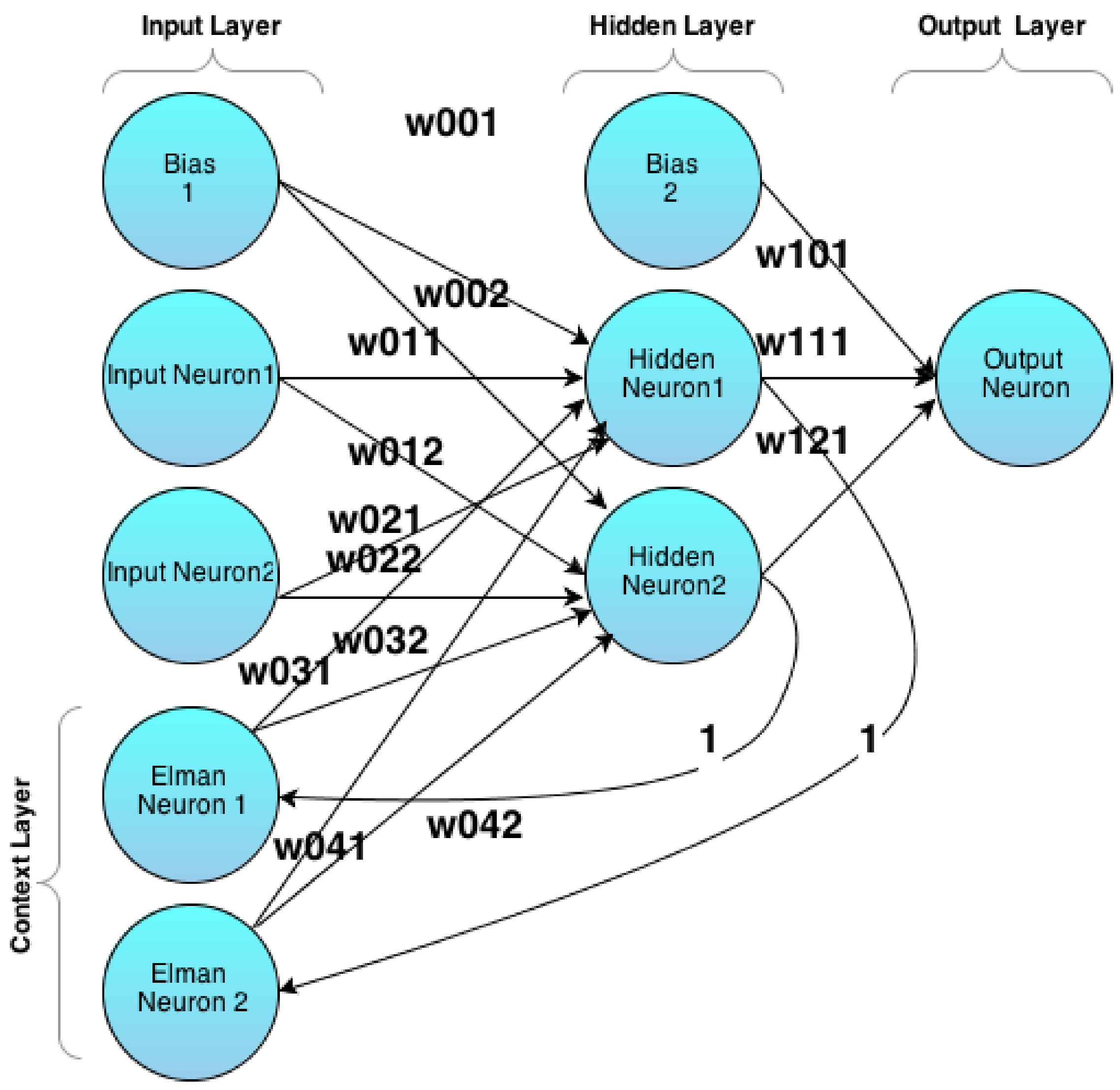


Figure 4.6: Example Elman Neural Network.

4.2.3 Continous Time Recurrent Neural Network

A human nervous system and hands are physical systems with positions and values interacting with each other smoothly in a continuous real time. This is a good example of a dynamical system - a system in which arbitrary number of variables vary over time in a lawful manner, depending on the current values of these variables. It is impossible to model such systems using static models like e.g. MFANNs because the state of such system is continuously changing. CTRNN is a dynamical system itself that is able to model other dynamical systems. In fact, it was proven by Funahashi and Nakamura [33] that a CTRNN is able to approximate any dynamical system.

As opposed to other ANNs, weights in a CTRNN are fixed and the learning process is based on changing neurons' states. As a result, it is hard to think about CTRNN as another ANN morphology, in fact, it is a little bit misleading.In a CTRNN each neuron is just a variable of the system. For example if CTRNN would be used to model another ANN then some of the

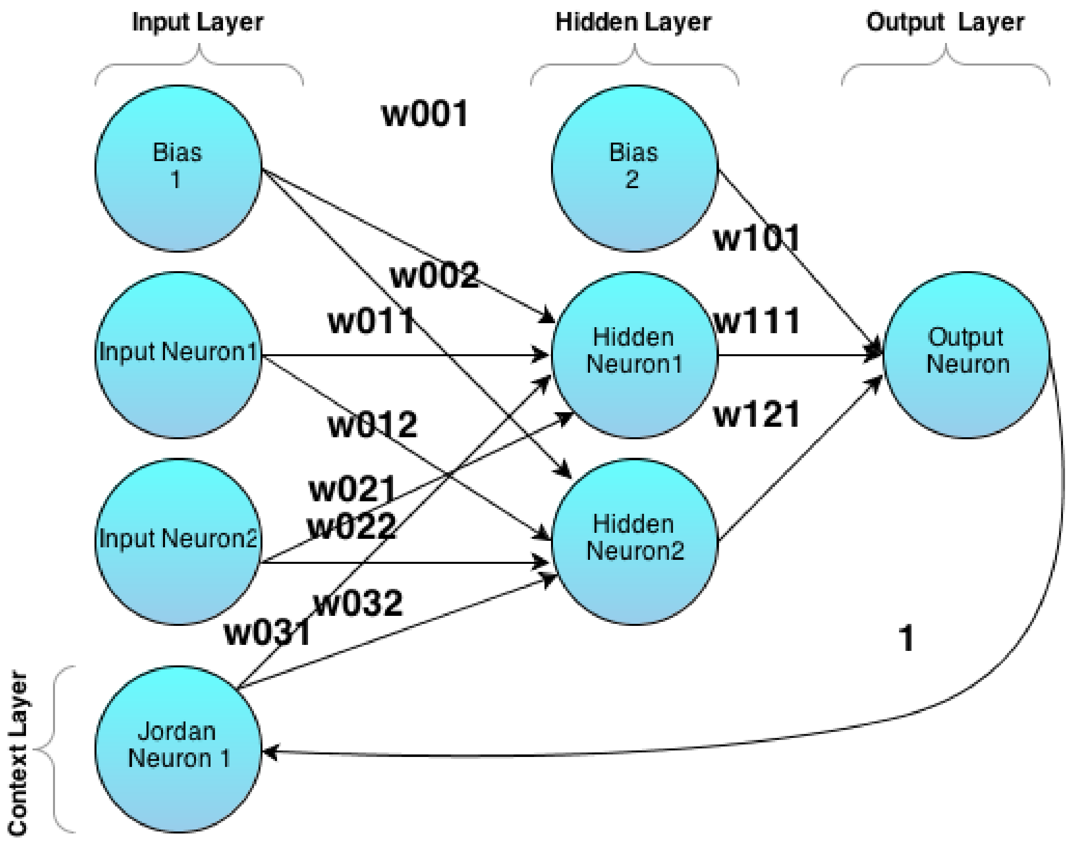


Figure 4.7: Example Jordan Neural Network.

neurons would represent weights whereas other activations. In CTRNNs neurons not weights play the most vital part.

For a neuron i in a CTRNN the rate of change of activation is given by Formula (4.2.3):

$$\tau_i \frac{\mathrm{d}y_i}{\mathrm{d}t} = -y_i + \sigma(\sum_{j=1}^{n} w_{ji} y_j - \Theta_j) + I_i(t) \tag{4.2.1}$$

where y_i i s the activation value of node i,τ_i is the time constant,$w_j i$ is a connection weight from node j to node i,$\sigma(x)$ is the value of sigmoid function for argument x, Θ_j is a bias of node j and $I_i(t)$ is possible sensory input.

In scope of this project, due to the fact that CTRNNs are much different than other ANNs (fixed weights) the existing design had to be heavily modified. It is important to note that since CTRNNs are dynamical systems every time they are used, they learn. This is important for the sake of evaluation. Other ANN do not learn during evaluation, they just calculate output values for given inputs. In case of CTRNNs the learning can't be switched off during evaluation meaning that every time evaluation will result different(hopefully better) results.

The author could not find any scientific literature concerning using CTRNNs to predict time series data. As such implementing this model was more of an experiment and the main goal of the author was to obtain some meaningful results rather than results comparable with other models. Luckily for the author, Dr Tuci is an expert in the field of CTRNNs and his help was invaluable. The parameters' ranges used in Equation (4.2.1) were provided by Dr Tuci based on a number of CTRNNs' applications he worked in the past. The exact initial values are unimportant because they will be determined using GA.

4.3 ANN Training Algorithms

4.3.1 Backpropagation

Backpropagation is one of the oldest and most popular training algorithm used. Backpropagation is a supervised learning algorithm - apart from inputs, it requires expected outputs as well. These expected outputs are compared against the actual results produced by the ANN to calculate the error used to update the connections weights. Backpropagation can be used with a number of different ANN morphologies however because this algorithm is based on using gradient descent, the activation function must be differentiable(e.g. a step function can't be used with backpropagation). The main goal of this algorithm is to minimize the error between expected and actual outputs by modifying connections weights between neurons.

As the name suggest backpropagation works backwards - the difference between expected and actual outputs is propagated to previous layers.In each layer backpropagation performs two actions:

(a) Calculate Error Contributions - backpropagation determines how much each neuron (according to its connections weights) has contributed to the error value. The neurons with higher weights contributed more hence they will have to be adjusted more compared with other neurons.

(b) Adjust Connections Weights.

Backpropagation looks for the minimum value of the error function in a weight space using gradient descent as in Figure 4.8. Connections weights that minimize the error function is then considered to be a solution to the learning problem.

During each epoch backpropagation performs following actions(In scope of this project all ANNs have 3 layers hence the backpropagation will be explained using a 3 layer ANN):

(a) Propagate the input - initialize ANN with random weights, feed input data into input neurons and calculate the output value of each neuron.

(b) Calculate error δ:

$$\delta = (desiredoutput) - (actualoutput) \tag{4.3.1}$$

(c) Propagate error backward - starting from the output layer calculate deltas(required changes in incoming connections weights values) for output and hidden neurons (no need to calculate them for input neurons since they have no connections coming in). Deltas are calculated using *Delta Rule* described in Equation 4.3.2.

$$\Delta w_i = \eta x_i \delta_i \quad where\ \delta_i = f'(net_i)(d_i - y_i) \tag{4.3.2}$$

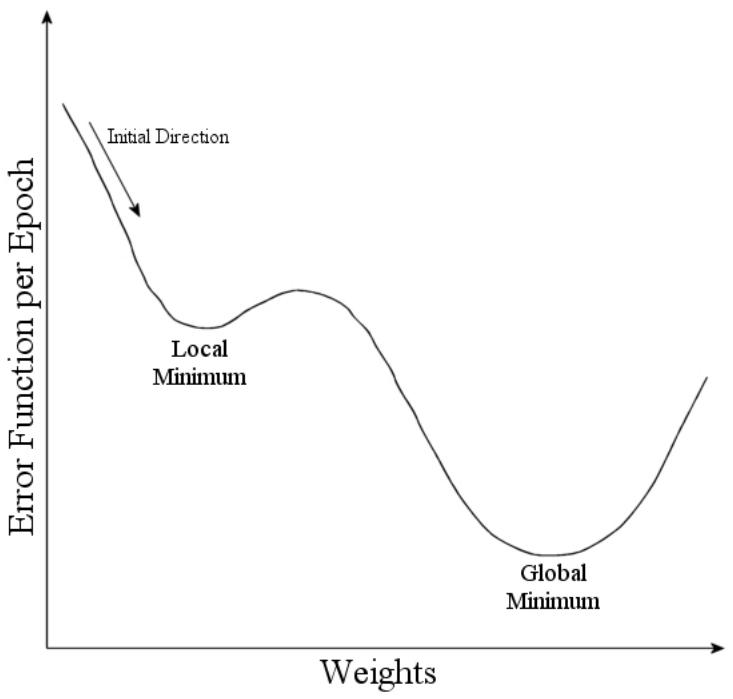


Figure 4.8: Backpropagation search over space of all possible weights values.

w is the array of weights, x is the array of inputs, and η is the learning rate, $f'(net_i)$ id the value of the derivate of the activation function for the weighted sum of all inputs coming into this particular neuron, y_i and d_i are the actual and desired outputs, respectively.
For example, if a sigmoid activation function is used(which derivate is given by the formula $f'(x) = x(1 - x)$) the δ_i for the output layer would look as in Equation 4.3.3.

$$\delta_i = y_i(1 - y_i)(d_i - y_i) \tag{4.3.3}$$

δ_i for neurons in the hidden layer is a calculated in a more complex way because when changing weights of incoming connections to a hidden neuron, not only the effect on the hidden neuron output value must be considered but also the effect on the output neurons. In order to do that the delta values of the output neurons are needed hence backpropagation - the error is propagated backwards from the output layer to the hidden layer. In order to keep this description simple, the author will not go into the mathematical derivation of the delta rule for the hidden layer. It has been proven [36] that for a neuron q in hidden layer p, δ_i is:

$$\delta_p(q) = x_p(q)[1 - x_p(q)] \sum w_{p+1}(q, i)\delta_{p+1}(i) \tag{4.3.4}$$

(d) Adjust connections weights by using delta rule described in Equation 4.3.2.

The main disadvantage of the backpropagation is that it is possible to become stuck in a local minimum. Although some methods exist to decrease the possibility of reaching local minimum (e.g. adding extra factor called *momentum* [48]) neither guarantees that the local minima will be avoided in 100% of cases.

4.3.2 Genetic Algorithm based Training

Natural selection is a key component of evolution. Organisms better suited to their environment are more likely to survive whereas organisms poorly suited to their environment are more like to die. The surviving organisms produce offspring that inherit many of parents' qualitites including qualities that decided about parents superiority over other organisms. As a result, those children are more likely to survive and produce own children. This process was described by Darwin in his famous "survival of the fittest" theory - an ongoing process of evolution in which life continues to improve over time. Exactly the same concepts are used in genetic algorithms.

Genetic Algorithms(GA) are the main paradigm of Evolutionary Computing. They solve problems by mimicking processes observed in nature: selection, crossover and mutation(genetic operators). These operators are responsible for maintaining genetic diversity across the population. They are adaptive search algorithms that are very good at searching large, nonlinear search spaces. In a nonlinear search space an optimal solution cannot be found using conventional iterative methods due to an enormous number of potential solutions. For example, a famous "traveling salesman" problem is too complex for traditional methods however GA is able to find the optimal or at least good enough solution in a reasonable time frame. A "traveling salesman" problem is an example of a NP-hard problem which cannot be solved in a polynomial time.

Genetic Algorithm based training is not as efficient as backpropagation. Backpropagation training is based on gradient descent and always tries to minimise the error. In other words it performs a directed search using heuristic. GA performs a directed random search by starting from random points and thereby it converges slowly. Nevertheless the global search ensures that the algorithm will not be stuck in local minima as is the case with the backpropagation. Therefore if a training time is not an issue, the GA based training is a better choice that the backpropagation.

Before going more into GA details it is necessary to explain following terms:

Chromosome (Individual)Contains a solution in form of genes.

Gene Contains a part of solution in a numerical form.

Population Number of chromosomes with the same length.

Selection] Selecting chromosomes for crossover based on their fitness.

Crossover Taking two fit chromosomes and mix their genetic material to create new chromosomes.

Mutation Randomly changing a gene.

Fitness The value assigned to a chromosome based on how far or close it is from the solution.

Fitness Function A problem specific function that assigns fitness values to chromosomes.

During the work on this project the author has split the GA based training into number of implementation steps:

Step 1 Create initial population of chromosomes.

Step 2 Rank chromosomes according to their fitness value.

Step 3 Select chromosomes for crossover.

Step 4 Crossover selected chromosomes and produce offspring.

Step 5 Randomly mutate chromosome's genes.

Step 6 Repeat steps 3 to 5 until new population is created.

Step 7 Terminate training process.

4.3.2.1 Step 1: Create initial population of chromosomes

Before a GA is put to work, a method is needed to encode all possible solutions in a form that is understandable to a computer. In scope of this project the author decided that a single gene will represent a single connection weight. As a result a single chromosome represents a single ANN. It is important that chromosomes can be easily converted into ANN and vice versa.

4.3.2.2 Step 2: Rank chromosomes according to their fitness value

In order to choose chromosomes for mating (crossover) a fitness function is needed. The author decided that a fitness of a chromosome will be denoted by a final MSE value of the ANN that was used to create that chromosome. In order to calculate a MSE value of a chromosome it has to be converted into an ANN.

4.3.2.3 Step 3: Select chromosomes for crossover

According to Darwin's evolution theory only the fittest should survive and produce offspring. There are a number of possible selection algorithms that can be used, however the author has decided to implement a Roulette Wheel Selection(RWS).

In a RWS the chance of an individual's being selected is proportional to its fitness. Individuals with higher fitness value has bigger chance of being selected than less fit individual. This mimics the nature more closely than simply choosing randomly from a set of fittest individuals. Conceptually, this can be thought as a game of roulette.

4.3.2.4 Step 4: Crossover selected chromosomes and produce offspring

The idea behind the crossover(mating) is that the offspring may be better(have higher fitness value) than both parents by combining the best features of both. The author has decided to use a two-point crossover as a crossover technique.

Two-point crossover randomly selects two crossover points with a chromosome then interchanges the two parent chromosomes between these points to produce two new offsprings.

4.3.2.5 Step 5: Randomly mutate chromosome's genes

Mutation is used to introduce new genetic material into a population. It is unknown whether or not mutation will produce an attribute that will improve chromosome's overall fitness, however it does not matter. The fate of mutated chromosome will be determined by the natural selection. Mutation is an important part of the genetic search because it prevents the search from falling into a local optimum.

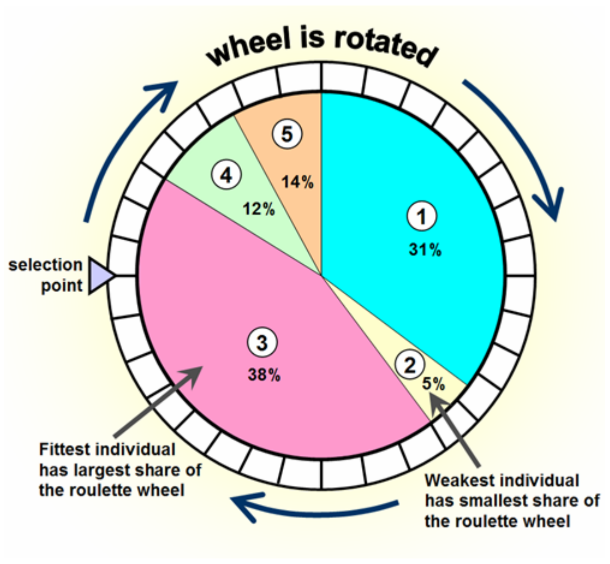

wheel is rotated
selection point
1
31%
2
5%
3
38%
4
12%
5
14%
Fittest individual has largest share of the roulette wheel
Weakest individual has smallest share of the roulette wheel

Figure 4.9: Roulette Wheel Selection.

In scope of this project mutation is done at two levels.

(a) Chromosome Mutation - only a percentage of all offspring are chosen to be mutated.

(b) Gene Mutation - only a percentage of all genes within a chromosome are chosen to mutate.

In case of a chromosome mutation, the mutation operator uses a simple random uniform distribution to decide if a chromosome should be mutated. If this random value is smaller than some predefined value e.g. 0.10, the offspring is chosen for mutation process.

In case of gene mutation, the mutation operator adds a unit Gaussian distributed random number to the chosen gene. If a new value falls outside a predefined range it is clipped. Random number is drawn from a Gaussian Distribution (famous *bell curve*) with mean 0 meaning that the most of increments/decrements are fairly small and on average much smaller than if uniform distribution was used.

4.3.2.6 Step 6: Repeat steps 3 to 5 until new population is created

New generation can either consist of only offspring or offspring and fittest individuals from previous generation. This strategy is called *elitism* - the fittest individuals are allowed to survive unaltered in the next generation of chromosomes.

4.3.2.7 Step 7: Terminate training process

Typically GA based training is terminated if the best solution does not change for a number of predefined generations. In scope of this project, the author has decided to add an alternative termination criteria: terminate the training after a number of predefined generations. The reason behind it is that the GA training process takes a lot of time and sometimes trade-offs between the training time and best solutions must be made.

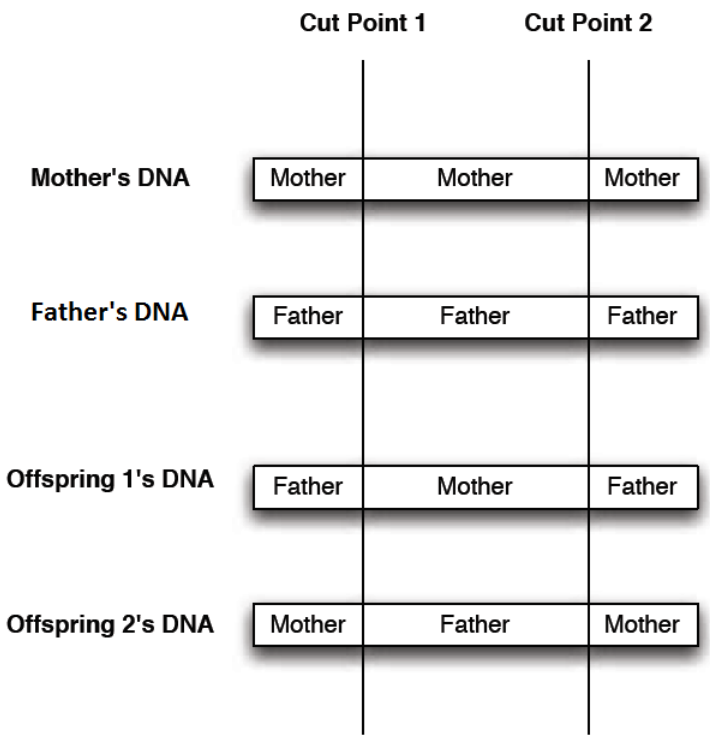


Figure 4.10: Two Point Crossover.

4.4 Overall Architecture

This section describes the overall architecture of the application used for testing ANNs models.

4.4.1 Initial Architecture

The initial architecture was designed while working on the XOR problem.

This is the simplest architecture that allows to implement an ANN. This design was improved with every development iteration.

4.4.2 Final Architecture

The final architecture is shown in the Figure 4.12.

Due to a high number of classes within packages each package will be discussed separately. Please not that class diagrams does not contain private operations because they only serve the purpose of being implementation details and are just auxiliary to other operations. Business, exception and utils packages are used by most of other package but in order to maintain diagram readability the author did not add any connections between those packages.

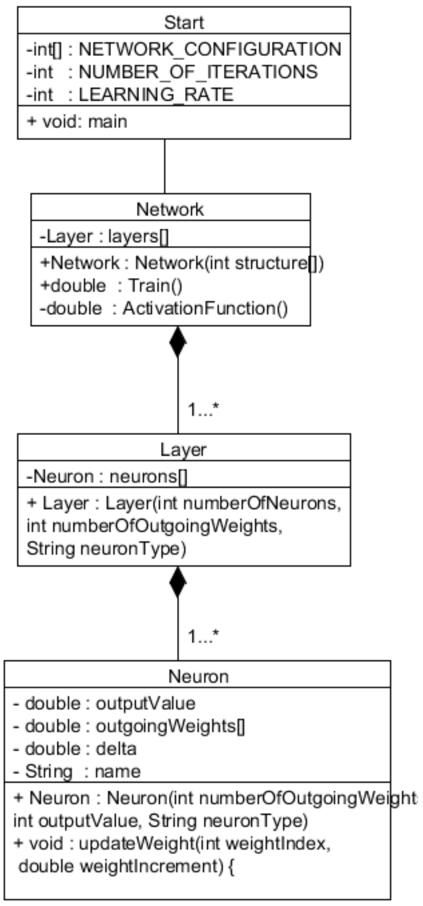

Start
-int[] : NETWORK_CONFIGURATION
-int : NUMBER_OF_ITERATIONS
-int : LEARNING_RATE
+ void: main
Network
-Layer : layers[]
+Network : Network(int structure[])
+double : Train()
-double : ActivationFunction()
1...*
Layer
-Neuron : neurons[]
+ Layer : Layer(int numberOfNeurons, int numberOfOutgoingWeights, String neuronType)
1...*
Neuron
- double : outputValue
- double : outgoingWeights[]
- double : delta
- String : name
+ Neuron : Neuron(int numberOfOutgoingWeight int outputValue, String neuronType)
+ void : updateWeight(int weightIndex, double weightIncrement) {

Figure 4.11: Initial architecture of an ANN.

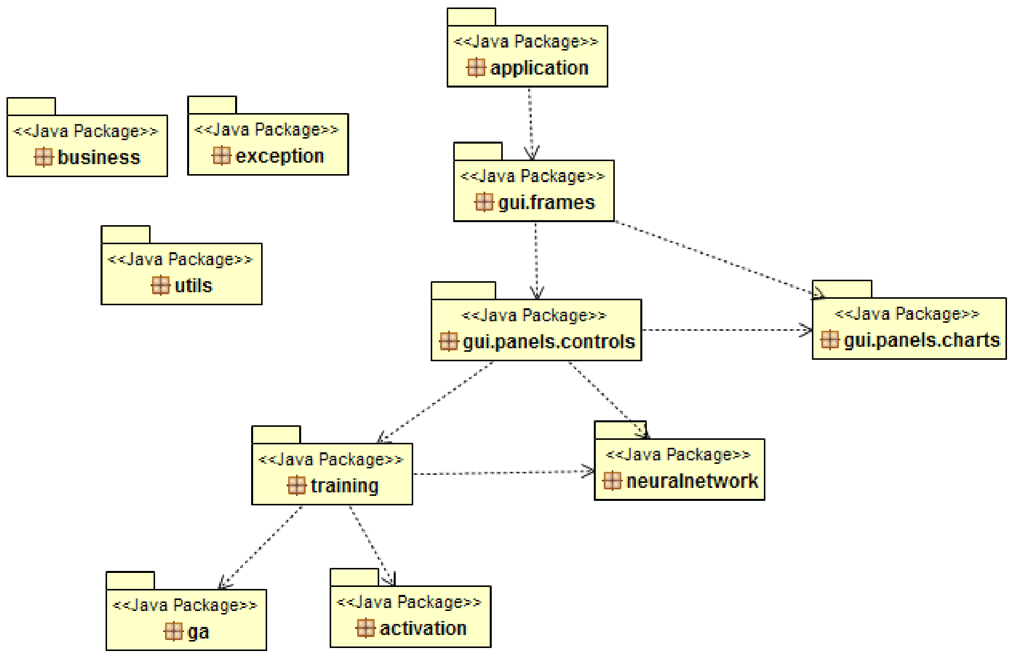


Figure 4.12: Final architecture of the application.

4.4.2.1 Application package

The application package contains only one class called "Run". This is class responsible for starting the application by creating necessary windows(frames) and initialising objects.

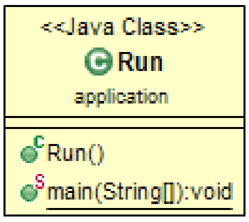


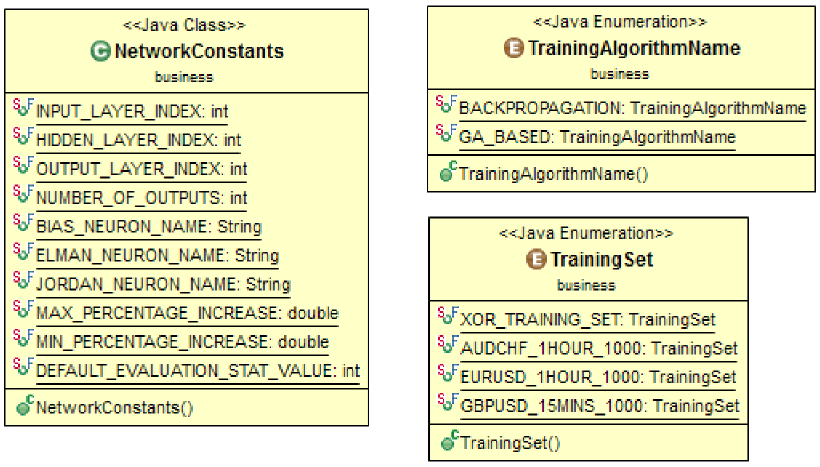
Figure 4.13: Application Package.

4.4.2.2 Business package

The business package(Figure 4.14) is designed to store constants that are used across multiple packages. The advantage of this approach is that changing constant value in one place result in changing this constant in many other places as well.

4.4.2.3 Utils package

The utils package(Figure 4.15) consist of classes with methods that are used across many packages.



<<Java Class>>
NetworkConstants
business
INPUT_LAYER_INDEX: int
HIDDEN_LAYER_INDEX: int
OUTPUT_LAYER_INDEX: int
NUMBER_OF_OUTPUTS: int
BIAS_NEURON_NAME: String
ELMAN_NEURON_NAME: String
JORDAN_NEURON_NAME: String
MAX_PERCENTAGE_INCREASE: double
MIN_PERCENTAGE_INCREASE: double
DEFAULT_EVALUATION_STAT_VALUE: int
NetworkConstants()
<<Java Enumeration>>
TrainingAlgorithmName
business
BACKPROPAGATION: TrainingAlgorithmName
GA_BASED: TrainingAlgorithmName
TrainingAlgorithmName()
<<Java Enumeration>>
TrainingSet
business
XOR_TRAINING_SET: TrainingSet
AUDCHF_1HOUR_1000: TrainingSet
EURUSD_1HOUR_1000: TrainingSet
GBPUSD_15MINS_1000: TrainingSet
TrainingSet()

Figure 4.14: Business Package.

Tuple A tuple is a data structure consisting of a pair of objects. Using tuples can simplify the implementation a lot. Unfortunately java API does not support tuples (compared to e.g. Haskell) and the author had to create it himself.

NumberUtils Stores methods used for number manipulation. Most of those methods are responsible for handling conversion between BigDecimal and double type numbers.

StatisticalUtils Stored methods used for calculating statistical indicators (MSE, R^2, absolute max and min errors).

ListUtils Stored methods used for list manipulation.

DataUtils Stored methods used for data pre- and post-processing.

DateUtils Stored methods used for date manipulation.

FileUtils Stored methods used for data saving and loading.

4.4.2.4 Exception package

The exception package(Figure 4.16) consists of classes implementing custom exceptions. Custom exceptions are used to identify situations when the application's business logic is breached. Standard java exceptions are often too broad in meaning whereas custom exception help to limit possible causes of a failure.

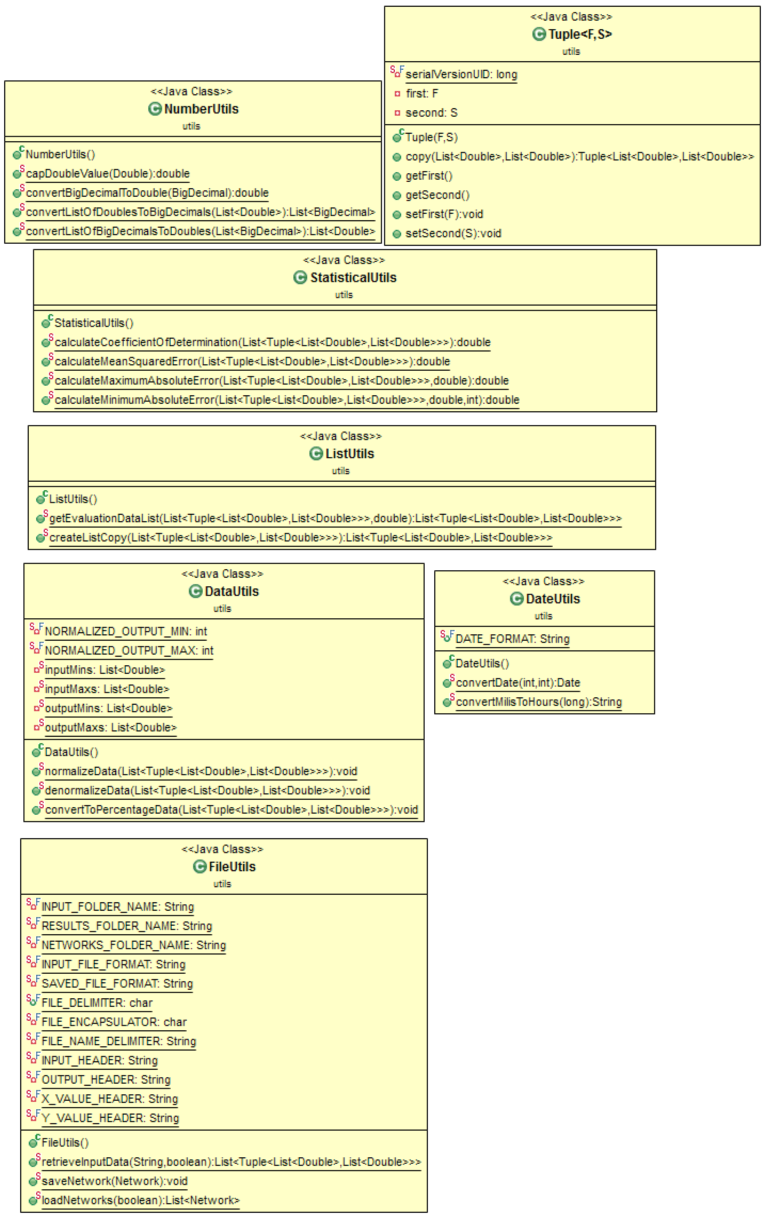
NeuralNetworkException Used for general problems related to ANNs.

TrainingException Used for problems with training process of ANNs.

4.4.2.5 Frames package

The frames packages(Figure 4.17) consists of classes creating frames(GUI windows).

MainFrame Draws the main window of the application titled ”Neural Network Mentalist”. It controls the creation of all other frames.



<<Java Class>>
Tuple<F,S>
utils
serialVersionUID: long
first: F
second: S
Tuple(F,S)
copy(List<Double>,List<Double>):Tuple<List<Double>,List<Double>>
getFirst()
getSecond()
setFirst(F):void
setSecond(S):void
<<Java Class>>
NumberUtils
utils
NumberUtils()
capDoubleValue(Double):double
convertBigDecimalToDouble(BigDecimal):double
convertListOfDoublesToBigDecimals(List<Double>):List<BigDecimal>
convertListOfBigDecimalsToDoubles(List<BigDecimal>):List<Double>
<<Java Class>>
StatisticalUtils
utils
StatisticalUtils()
calculateCoefficientOfDetermination(List<Tuple<List<Double>,List<Double>>>):double
calculateMeanSquaredError(List<Tuple<List<Double>,List<Double>>>):double
calculateMaximumAbsoluteError(List<Tuple<List<Double>,List<Double>>>,double):double
calculateMinimumAbsoluteError(List<Tuple<List<Double>,List<Double>>>,double,int):double
<<Java Class>>
ListUtils
utils
ListUtils()
getEvaluationDataList(List<Tuple<List<Double>,List<Double>>>,double):List<Tuple<List<Double>,List<Double>>>
createListCopy(List<Tuple<List<Double>,List<Double>>>):List<Tuple<List<Double>,List<Double>>>
<<Java Class>>
DataUtils
utils
NORMALIZED_OUTPUT_MIN: int
NORMALIZED_OUTPUT_MAX: int
inputMins: List<Double>
inputMaxs: List<Double>
outputMins: List<Double>
outputMaxs: List<Double>
DataUtils()
normalizeData(List<Tuple<List<Double>,List<Double>>>):void
denormalizeData(List<Tuple<List<Double>,List<Double>>>):void
convertToPercentageData(List<Tuple<List<Double>,List<Double>>>):void
<<Java Class>>
DateUtils
utils
DATE_FORMAT: String
DateUtils()
convertDate(int,int):Date
convertMilisToHours(long):String
<<Java Class>>
FileUtils
utils
INPUT_FOLDER_NAME: String
RESULTS_FOLDER_NAME: String
NETWORKS_FOLDER_NAME: String
INPUT_FILE_FORMAT: String
SAVED_FILE_FORMAT: String
FILE_DELIMITER: char
FILE_ENCAPSULATOR: char
FILE_NAME_DELIMITER: String
INPUT_HEADER: String
OUTPUT_HEADER: String
X_VALUE_HEADER: String
Y_VALUE_HEADER: String
FileUtils()
retrieveInputData(String,boolean):List<Tuple<List<Double>,List<Double>>>
saveNetwork(Network):void
loadNetworks(boolean):List<Network>

Figure 4.15: Utils Package.

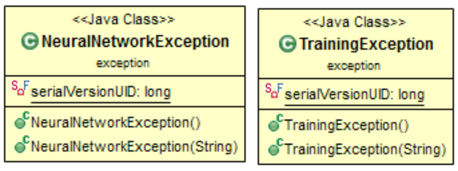


Figure 4.16: Exception Package.

ErrorFrame Draws the window used for presenting MSE convergence titled ”Mean Squared Error”.

WeightsFrame Draws the window used for presenting ANN’s weights convergence titled ”Weights and Error Correlation”.

MainChartFrame Draws the window used for presenting a data set used as ANN’s input titled ”Time Series”.

EvaluationFrame Draws the window used for presenting evaluation results titled ”Evaluation”.

TrainingComparisonFrame Draws the window used for presenting comparison of MSE convergence across multiple ANNs titled ”MSE Results Comparison”.

EvaluationComparisonFrame Draws the window used for presenting comparison of evaluation results across multiple ANNs titled ”Evaluation Results Comparison”.

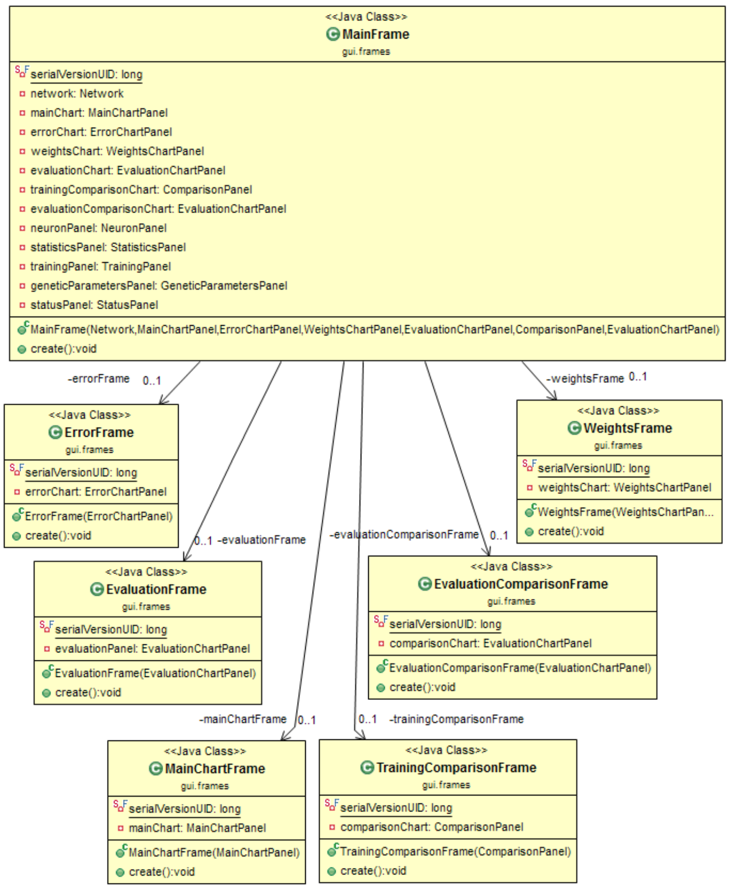

<<Java Class>>
MainFrame
gui.frames
serialVersionUID: long
network: Network
mainChart: MainChartPanel
errorChart: ErrorChartPanel
weightsChart: WeightsChartPanel
evaluationChart: EvaluationChartPanel
trainingComparisonChart: ComparisonPanel
evaluationComparisonChart: EvaluationChartPanel
neuronPanel: NeuronPanel
statisticsPanel: StatisticsPanel
trainingPanel: TrainingPanel
geneticParametersPanel: GeneticParametersPanel
statusPanel: StatusPanel
MainFrame(Network,MainChartPanel,ErrorChartPanel,WeightsChartPanel,EvaluationChartPanel,ComparisonPanel,EvaluationChartPanel)
create():void
-errorFrame 0..1
-weightsFrame 0..1
<<Java Class>>
ErrorFrame
gui.frames
serialVersionUID: long
errorChart: ErrorChartPanel
ErrorFrame(ErrorChartPanel)
create():void
<<Java Class>>
WeightsFrame
gui.frames
serialVersionUID: long
weightsChart: WeightsChartPanel
WeightsFrame(WeightsChartPan...
create():void
0..1 -evaluationFrame
-evaluationComparisonFrame 0..1
<<Java Class>>
EvaluationFrame
gui.frames
serialVersionUID: long
evaluationPanel: EvaluationChartPanel
EvaluationFrame(EvaluationChartPanel)
create():void
<<Java Class>>
EvaluationComparisonFrame
gui.frames
serialVersionUID: long
comparisonChart: EvaluationChartPanel
EvaluationComparisonFrame(EvaluationChartPanel)
create():void
-mainChartFrame 0..1
0..1 -trainingComparisonFrame
<<Java Class>>
MainChartFrame
gui.frames
serialVersionUID: long
mainChart: MainChartPanel
MainChartFrame(MainChartPanel)
create():void
<<Java Class>>
TrainingComparisonFrame
gui.frames
serialVersionUID: long
comparisonChart: ComparisonPanel
TrainingComparisonFrame(ComparisonPanel)
create():void

Figure 4.17: Frames Package.

4.4.2.6 Charts package

The charts package(Figure 4.18) consists of classes used to draw charts (chart types are described in Section 4.6.6).

ChartPanel A main class (also superclass) storing common methods used across other chart implementing classes.

EvaluationChartPanel Draws the chart used for presenting evaluation results titled ”Evaluation Chart”.

WeightsChartPanel Draws the chart used for presenting ANN’s weights convergence titled ”Weights and Error Correlation”.

ErrorChartPanel Draws the chart used for presenting MSE convergence titled ”Error Chart”.

MainChartPanel Draws the chart used for presenting a data set used as ANN’s input titled ”Main Chart”.

MainChartPanel Draws the chart used for presenting evaluation (”Evaluation Comparison Chart”) and training (”Training Comparison Chart”) comparison results.

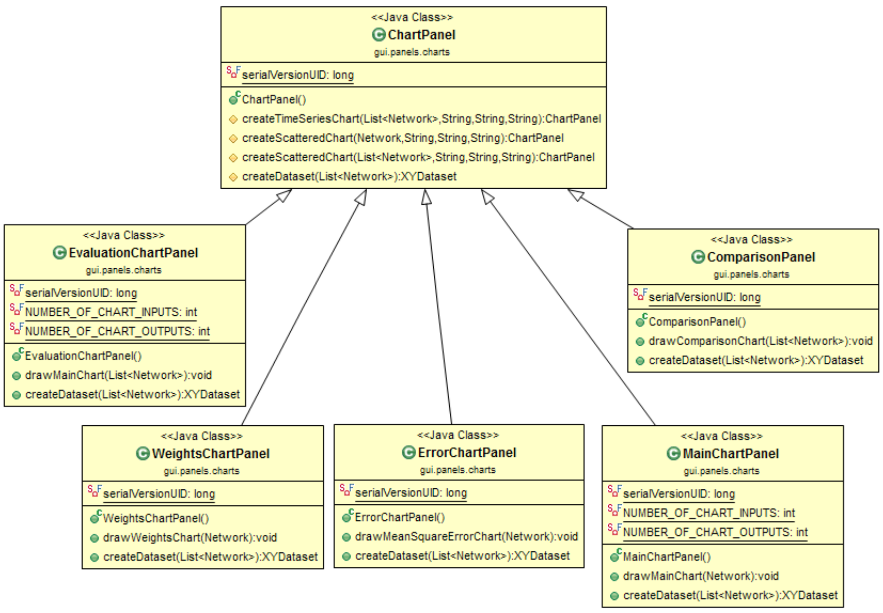

<<Java Class>>
ChartPanel
gui.panels.charts
serialVersionUID: long
ChartPanel()
createTimeSeriesChart(List<Network>,String,String,String):ChartPanel
createScatteredChart(Network,String,String,String):ChartPanel
createScatteredChart(List<Network>,String,String,String):ChartPanel
createDataset(List<Network>):XYDataset
<<Java Class>>
EvaluationChartPanel
gui.panels.charts
serialVersionUID: long
NUMBER_OF_CHART_INPUTS: int
NUMBER_OF_CHART_OUTPUTS: int
EvaluationChartPanel()
drawMainChart(List<Network>):void
createDataset(List<Network>):XYDataset
<<Java Class>>
ComparisonPanel
gui.panels.charts
serialVersionUID: long
ComparisonPanel()
drawComparisonChart(List<Network>):void
createDataset(List<Network>):XYDataset
<<Java Class>>
WeightsChartPanel
gui.panels.charts
serialVersionUID: long
WeightsChartPanel()
drawWeightsChart(Network):void
createDataset(List<Network>):XYDataset
<<Java Class>>
ErrorChartPanel
gui.panels.charts
serialVersionUID: long
ErrorChartPanel()
drawMeanSquareErrorChart(Network):void
createDataset(List<Network>):XYDataset
<<Java Class>>
MainChartPanel
gui.panels.charts
serialVersionUID: long
NUMBER_OF_CHART_INPUTS: int
NUMBER_OF_CHART_OUTPUTS: int
MainChartPanel()
drawMainChart(Network):void
createDataset(List<Network>):XYDataset

Figure 4.18: Charts Package.

4.4.2.7 Controls package

The controls package(Figure 4.19) consists of classes used to draw control panels (panels used to input ANN’s parameters and output results).

TrainingPanel The main panel used to control other panels. Contains most important parameters including the choice of input data set, training algorithm and evaluation controls.

GeneticParametersPanel Contains parameters of GA based training.

NeuronPanel Contains parameters of ANN’s morphology.

StatisticsPanel Contains values of statictical indicators.

StatusPanel Contains textual information about current application state.

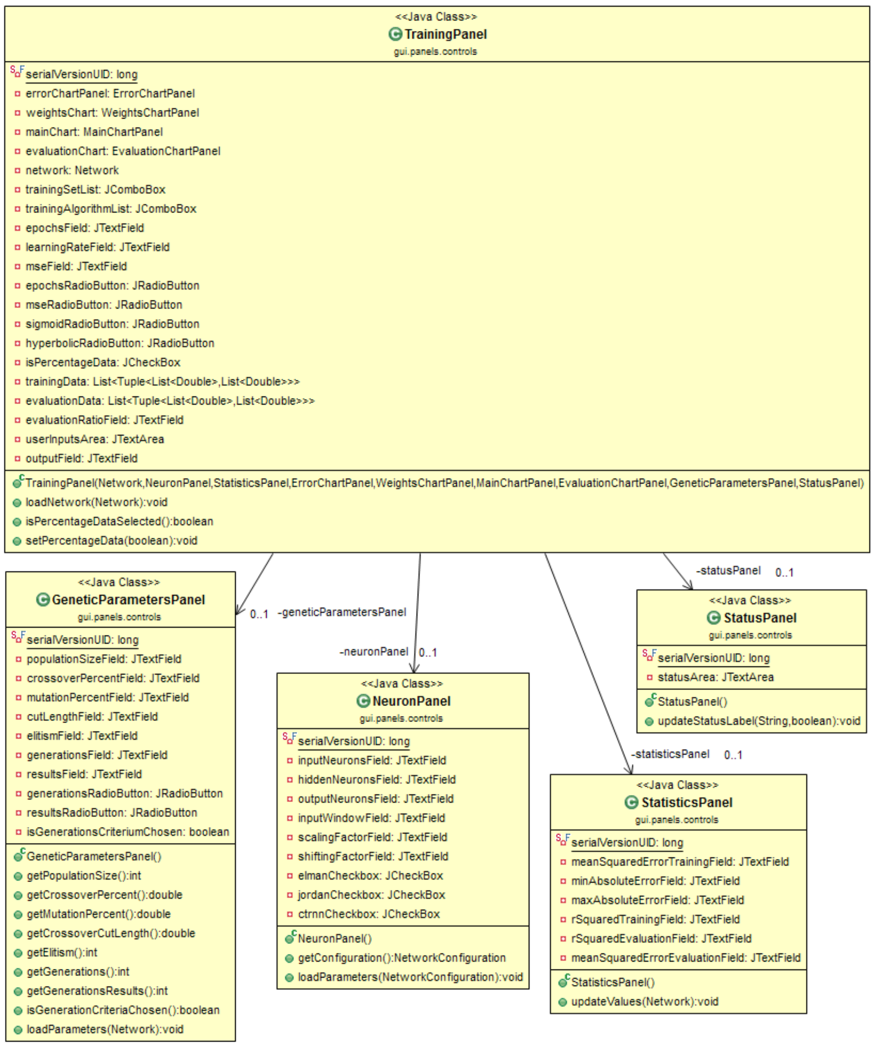

<<Java Class>>
TrainingPanel
gui.panels.controls
serialVersionUID: long
errorChartPanel: ErrorChartPanel
weightsChart: WeightsChartPanel
mainChart: MainChartPanel
evaluationChart: EvaluationChartPanel
network: Network
trainingSetList: JComboBox
trainingAlgorithmList: JComboBox
epochsField: JTextField
learningRateField: JTextField
mseField: JTextField
epochsRadioButton: JRadioButton
mseRadioButton: JRadioButton
sigmoidRadioButton: JRadioButton
hyperbolicRadioButton: JRadioButton
isPercentageData: JCheckBox
trainingData: List<Tuple<List<Double>,List<Double>>>
evaluationData: List<Tuple<List<Double>,List<Double>>>
evaluationRatioField: JTextField
userInputsArea: JTextArea
outputField: JTextField
TrainingPanel(Network,NeuronPanel,StatisticsPanel,ErrorChartPanel,WeightsChartPanel,MainChartPanel,EvaluationChartPanel,GeneticParametersPanel,StatusPanel)
loadNetwork(Network):void
isPercentageDataSelected():boolean
setPercentageData(boolean):void
0..1 -geneticParametersPanel
-neuronPanel 0..1
-statusPanel 0..1
-statisticsPanel 0..1
<<Java Class>>
GeneticParametersPanel
gui.panels.controls
serialVersionUID: long
populationSizeField: JTextField
crossoverPercentField: JTextField
mutationPercentField: JTextField
cutLengthField: JTextField
elitismField: JTextField
generationsField: JTextField
resultsField: JTextField
generationsRadioButton: JRadioButton
resultsRadioButton: JRadioButton
isGenerationsCriteriumChosen: boolean
GeneticParametersPanel()
getPopulationSize():int
getCrossoverPercent():double
getMutationPercent():double
getCrossoverCutLength():double
getElitism():int
getGenerations():int
getGenerationsResults():int
isGenerationCriteriaChosen():boolean
loadParameters(Network):void
<<Java Class>>
NeuronPanel
gui.panels.controls
serialVersionUID: long
inputNeuronsField: JTextField
hiddenNeuronsField: JTextField
outputNeuronsField: JTextField
inputWindowField: JTextField
scalingFactorField: JTextField
shiftingFactorField: JTextField
elmanCheckbox: JCheckBox
jordanCheckbox: JCheckBox
ctrnnCheckbox: JCheckBox
NeuronPanel()
getConfiguration():NetworkConfiguration
loadParameters(NetworkConfiguration):void
<<Java Class>>
StatusPanel
gui.panels.controls
serialVersionUID: long
statusArea: JTextArea
StatusPanel()
updateStatusLabel(String,boolean):void
<<Java Class>>
StatisticsPanel
gui.panels.controls
serialVersionUID: long
meanSquaredErrorTrainingField: JTextField
minAbsoluteErrorField: JTextField
maxAbsoluteErrorField: JTextField
rSquaredTrainingField: JTextField
rSquaredEvaluationField: JTextField
meanSquaredErrorEvaluationField: JTextField
StatisticsPanel()
updateValues(Network):void

Figure 4.19: Controls Package.

4.4.2.8 Neural Network package

The neural network package(Figure 4.20) consists of classes that model an ANN structure.

Network The main class of this package that models an ANN.

Layer Models an ANN's layer.

Neuron Models an ANN's neuron.

Synapse Models a connection between to neurons.

NetworkConfiguration Class storing most important information about ANN's structure.

CTRNeuralNetwork Models a CTRNN. The Network class could not be used to model a CTRNN due to differences between CTRNN and other ANNs are too big (e.g. CTRNN has fixed connections weights values).

CTRLayer Models a layer in a CTRNN.

CTRNeuron Models a neuron in a CTRNN.

4.4.2.9 Training package

The training package(Figure 4.21) consists of classes implementing training algorithms.

Training Interface enforcing methods that needs to be implemented by every training algorithm.

TrainingAlgorithm Superclass storing methods common for all training algorithms.

Backpropagation Contains the implementation of the backpropagation training algorithm.

GeneticAlgorithmBasedTraining Contains the implementation of the GA based training algorithm.

4.4.2.10 GA package

The ga package(Figure 4.22) contains classes modelling basic GA structures

Chromosome Models a chromosome as a list of weights with a fitness value.

FitnessComparator Used to compare fitness values of two chromosomes.

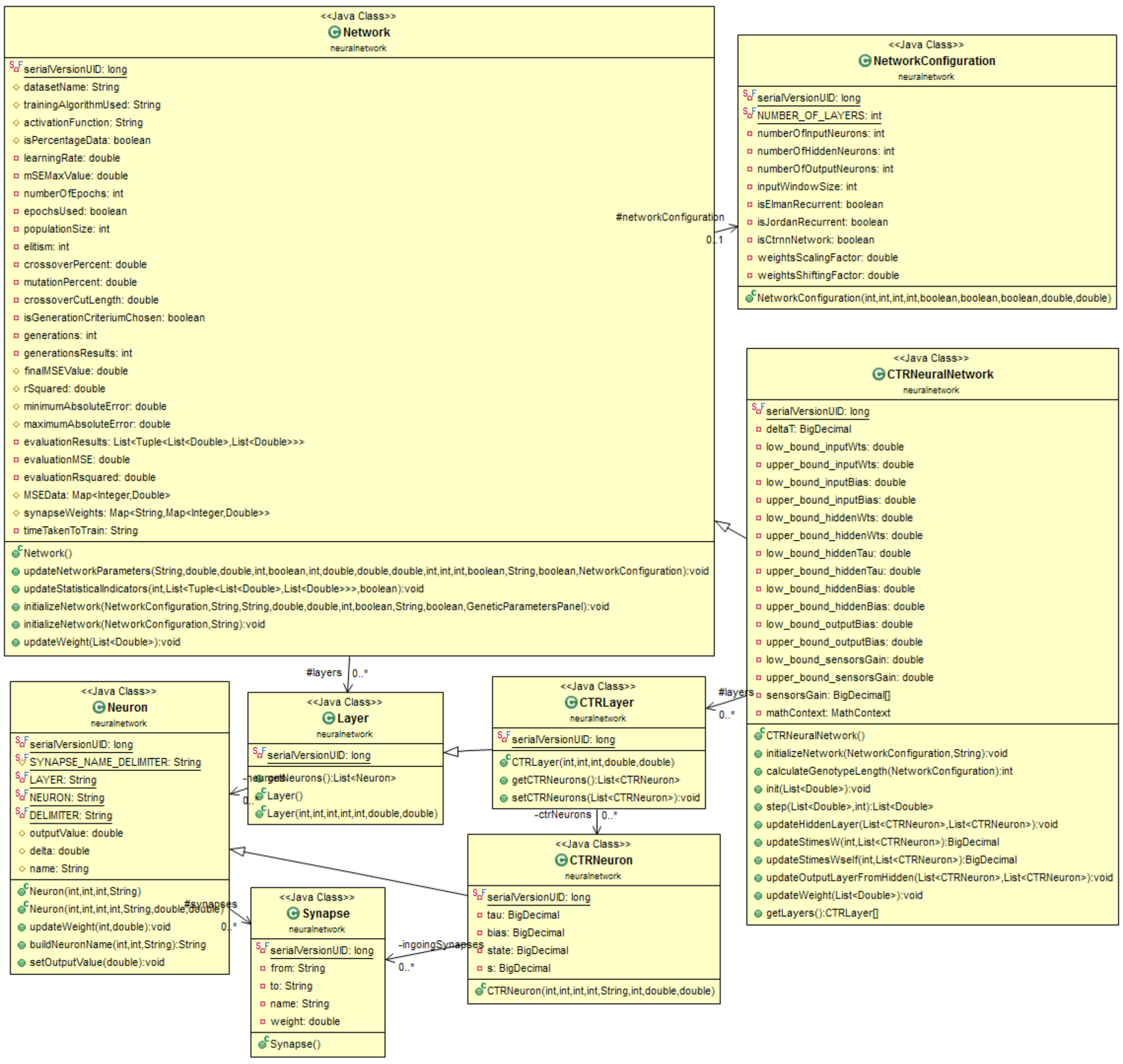
4.4.2.11 Activation package

The activation package(Figure 4.23) consists of classes implementing activation functions.

ActivationFunction Interface enforcing methods that needs to be implemented by every activation function.

SigmoidFunction Implements a sigmoid activation function.

HyperbolicFunction Implements a hyperbolic activation function.



<<Java Class>>
Network
neuralnetwork
serialVersionUID: long
datasetName: String
trainingAlgorithmUsed: String
activationFunction: String
isPercentageData: boolean
learningRate: double
mSEMaxValue: double
numberOfEpochs: int
epochsUsed: boolean
populationSize: int
elitism: int
crossoverPercent: double
mutationPercent: double
crossoverCutLength: double
isGenerationCriteriumChosen: boolean
generations: int
generationsResults: int
finalMSEValue: double
rSquared: double
minimumAbsoluteError: double
maximumAbsoluteError: double
evaluationResults: List<Tuple<List<Double>,List<Double>>>
evaluationMSE: double
evaluationRsquared: double
MSEData: Map<Integer,Double>
synapseWeights: Map<String,Map<Integer,Double>>
timeTakenToTrain: String
Network()
updateNetworkParameters(String,double,double,int,boolean,int,double,double,double,int,int,int,boolean,String,boolean,NetworkConfiguration):void
updateStatisticalIndicators(int,List<Tuple<List<Double>,List<Double>>>,boolean):void
initializeNetwork(NetworkConfiguration,String,String,double,double,int,boolean,String,boolean,GeneticParametersPanel):void
initializeNetwork(NetworkConfiguration,String):void
updateWeight(List<Double>):void
#networkConfiguration
0..1
<<Java Class>>
NetworkConfiguration
neuralnetwork
serialVersionUID: long
NUMBER_OF_LAYERS: int
numberOfInputNeurons: int
numberOfHiddenNeurons: int
numberOfOutputNeurons: int
inputWindowSize: int
isElmanRecurrent: boolean
isJordanRecurrent: boolean
isCtrnnNetwork: boolean
weightsScalingFactor: double
weightsShiftingFactor: double
NetworkConfiguration(int,int,int,int,boolean,boolean,boolean,double,double)
<<Java Class>>
CTRNeuralNetwork
neuralnetwork
serialVersionUID: long
deltaT: BigDecimal
low_bound_inputWts: double
upper_bound_inputWts: double
low_bound_inputBias: double
upper_bound_inputBias: double
low_bound_hiddenWts: double
upper_bound_hiddenWts: double
low_bound_hiddenTau: double
upper_bound_hiddenTau: double
low_bound_hiddenBias: double
upper_bound_hiddenBias: double
low_bound_outputBias: double
upper_bound_outputBias: double
low_bound_sensorsGain: double
upper_bound_sensorsGain: double
sensorsGain: BigDecimal[]
mathContext: MathContext
CTRNeuralNetwork()
initializeNetwork(NetworkConfiguration,String):void
calculateGenotypeLength(NetworkConfiguration):int
init(List<Double>):void
step(List<Double>,int):List<Double>
updateHiddenLayer(List<CTRNeuron>,List<CTRNeuron>):void
updateStimesW(int,List<CTRNeuron>):BigDecimal
updateStimesWself(int,List<CTRNeuron>):BigDecimal
updateOutputLayerFromHidden(List<CTRNeuron>,List<CTRNeuron>):void
updateWeight(List<Double>):void
getLayers():CTRLayer[]
#layers
0..*
<<Java Class>>
Neuron
neuralnetwork
serialVersionUID: long
SYNAPSE_NAME_DELIMITER: String
LAYER: String
NEURON: String
DELIMITER: String
outputValue: double
delta: double
name: String
Neuron(int,int,int,String)
Neuron(int,int,int,int,String,double,double)
updateWeight(int,double):void
buildNeuronName(int,int,String):String
setOutputValue(double):void
<<Java Class>>
Layer
neuralnetwork
serialVersionUID: long
getNeurons():List<Neuron>
Layer()
Layer(int,int,int,int,int,double,double)
-neurons
0..*
<<Java Class>>
CTRLayer
neuralnetwork
serialVersionUID: long
CTRLayer(int,int,int,double,double)
getCTRNeurons():List<CTRNeuron>
setCTRNeurons(List<CTRNeuron>):void
#layers
0..*
-ctrNeurons
0..*
<<Java Class>>
CTRNeuron
neuralnetwork
serialVersionUID: long
tau: BigDecimal
bias: BigDecimal
state: BigDecimal
s: BigDecimal
CTRNeuron(int,int,int,int,String,int,double,double)
#synapses
0..*
<<Java Class>>
Synapse
neuralnetwork
serialVersionUID: long
from: String
to: String
name: String
weight: double
Synapse()
-ingoingSynapses
0..*

Figure 4.20: Neural Network Package.

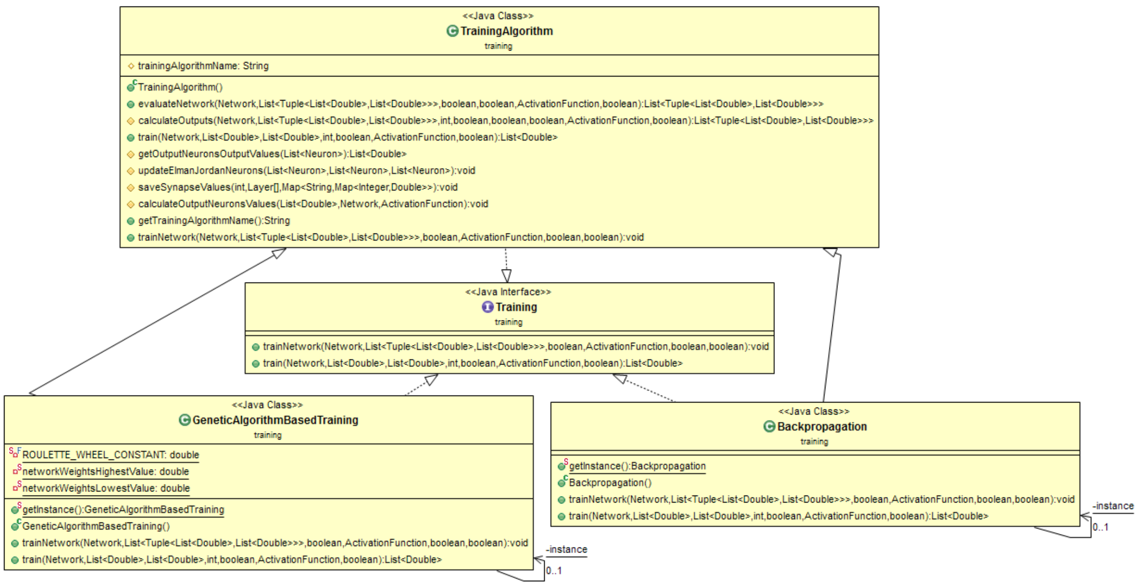

<<Java Class>>
TrainingAlgorithm
training
trainingAlgorithmName: String
TrainingAlgorithm()
evaluateNetwork(Network,List<Tuple<List<Double>,List<Double>>>,boolean,boolean,ActivationFunction,boolean):List<Tuple<List<Double>,List<Double>>>
calculateOutputs(Network,List<Tuple<List<Double>,List<Double>>>,int,boolean,boolean,boolean,ActivationFunction,boolean):List<Tuple<List<Double>,List<Double>>>
train(Network,List<Double>,List<Double>,int,boolean,ActivationFunction,boolean):List<Double>
getOutputNeuronsOutputValues(List<Neuron>):List<Double>
updateElmanJordanNeurons(List<Neuron>,List<Neuron>,List<Neuron>):void
saveSynapseValues(int,Layer[],Map<String,Map<Integer,Double>>):void
calculateOutputNeuronsValues(List<Double>,Network,ActivationFunction):void
getTrainingAlgorithmName():String
trainNetwork(Network,List<Tuple<List<Double>,List<Double>>>,boolean,ActivationFunction,boolean,boolean):void
<<Java Interface>>
Training
training
trainNetwork(Network,List<Tuple<List<Double>,List<Double>>>,boolean,ActivationFunction,boolean,boolean):void
train(Network,List<Double>,List<Double>,int,boolean,ActivationFunction,boolean):List<Double>
<<Java Class>>
GeneticAlgorithmBasedTraining
training
ROULETTE_WHEEL_CONSTANT: double
networkWeightsHighestValue: double
networkWeightsLowestValue: double
getInstance():GeneticAlgorithmBasedTraining
GeneticAlgorithmBasedTraining()
trainNetwork(Network,List<Tuple<List<Double>,List<Double>>>,boolean,ActivationFunction,boolean,boolean):void
train(Network,List<Double>,List<Double>,int,boolean,ActivationFunction,boolean):List<Double>
-instance
0..1
<<Java Class>>
Backpropagation
training
getInstance():Backpropagation
Backpropagation()
trainNetwork(Network,List<Tuple<List<Double>,List<Double>>>,boolean,ActivationFunction,boolean,boolean):void
train(Network,List<Double>,List<Double>,int,boolean,ActivationFunction,boolean):List<Double>
-instance
0..1

Figure 4.21: Training Package.

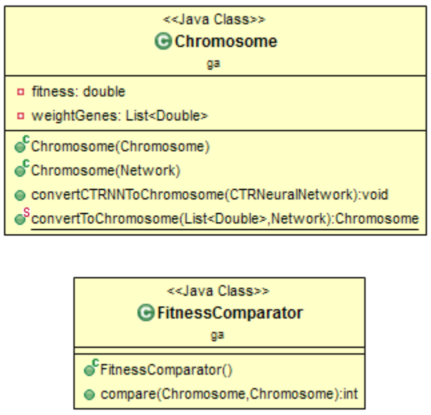

<<Java Class>>
Chromosome
ga
fitness: double
weightGenes: List<Double>
Chromosome(Chromosome)
Chromosome(Network)
convertCTRNNToChromosome(CTRNeuralNetwork):void
convertToChromosome(List<Double>,Network):Chromosome
<<Java Class>>
FitnessComparator
ga
FitnessComparator()
compare(Chromosome,Chromosome):int

Figure 4.22: GA Package.

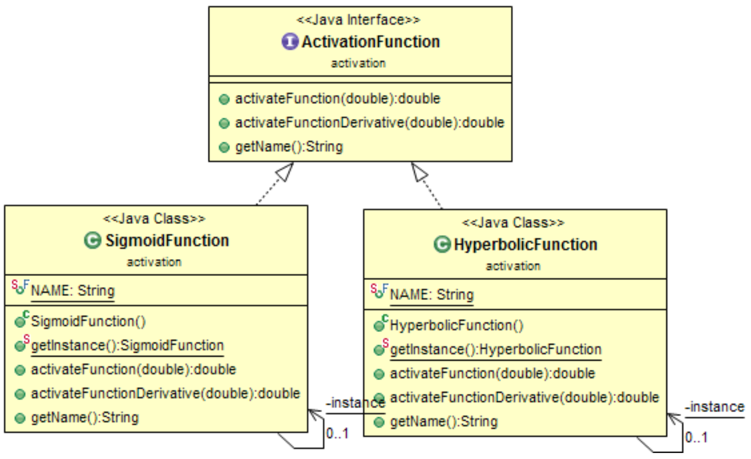

<<Java Interface>>
ActivationFunction
activation
activateFunction(double):double
activateFunctionDerivative(double):double
getName():String
<<Java Class>>
SigmoidFunction
activation
NAME: String
SigmoidFunction()
getInstance():SigmoidFunction
activateFunction(double):double
activateFunctionDerivative(double):double
getName():String
-instance
0..1
<<Java Class>>
HyperbolicFunction
activation
NAME: String
HyperbolicFunction()
getInstance():HyperbolicFunction
activateFunction(double):double
activateFunctionDerivative(double):double
getName():String
-instance
0..1

Figure 4.23: Activation Package.

4.5 Use Case Diagram

The use case diagram is shown in Figure 4.24.

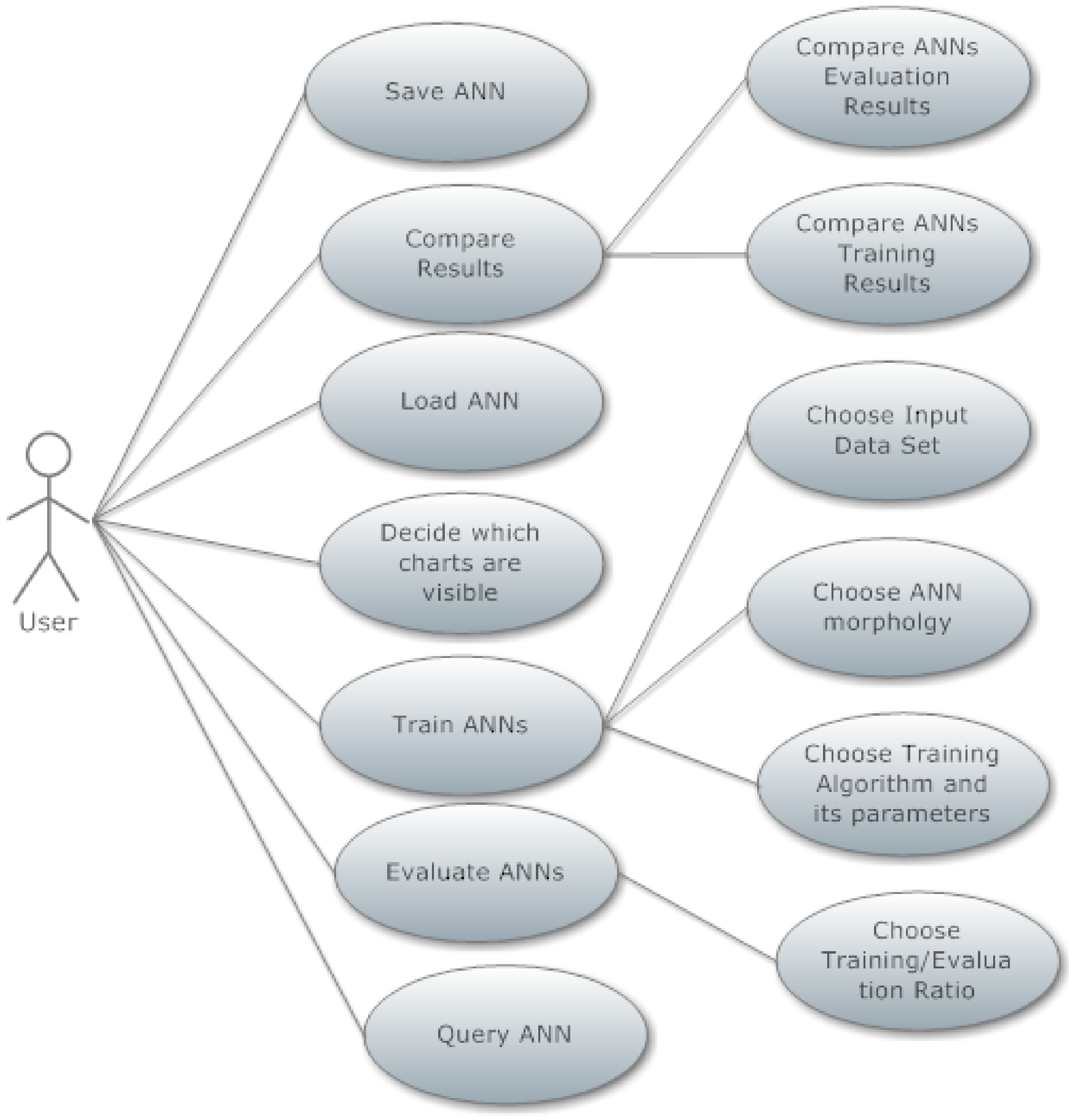
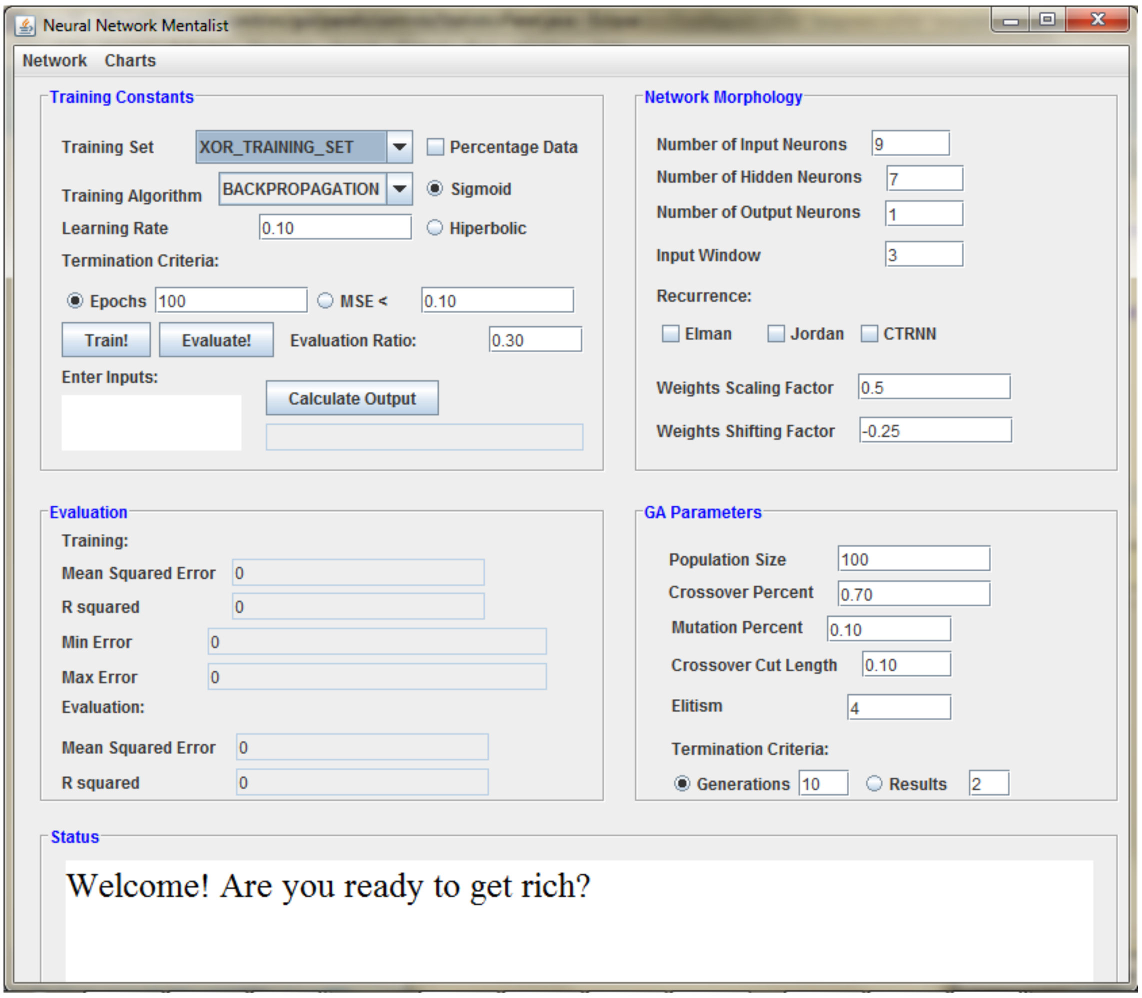

Save ANN
Compare ANNs Evaluation Results
Compare Results
Compare ANNs Training Results
Load ANN
Choose Input Data Set
User
Decide which charts are visible
Choose ANN morpholgy
Train ANNs
Choose Training Algorithm and its parameters
Evaluate ANNs
Choose Training/Evalua tion Ratio
Query ANN

Figure 4.24: Use Case Diagram.

4.6 User Interface

The GUI of the ”Neural Network Mentalist” testing application is shown in Figure 4.25. The interface is divided into 5 main sections:

- Training Constants



Training Constants

Training Set	XOR_TRAINING_SET
Percentage Data	☐
Training Algorithm	BACKPROPAGATION
[✓ Sigmoid]	Hiperbolic
Learning Rate	0.10

Termination Criteria:

[✓ Epochs]	100
MSE <	0.10
Evaluation Ratio:	0.30

Enter Inputs:

Network Morphology

Number of Input Neurons	9
Number of Hidden Neurons	7
Number of Output Neurons	1
Input Window	3

Recurrence:

☐ Elman ☐ Jordan ☐ CTRNN

Weights Scaling Factor	0.5
Weights Shifting Factor	-0.25

Evaluation

Training:

Mean Squared Error	0
R squared	0
Min Error	0
Max Error	0

Evaluation:

Mean Squared Error	0
R squared	0

GA Parameters

Population Size	100
Crossover Percent	0.70
Mutation Percent	0.10
Crossover Cut Length	0.10
Elitism	4

Termination Criteria:

[✓ Generations]	10
Results	2

Status

Welcome! Are you ready to get rich?

Figure 4.25: Neural Network Mentalist's GUI.

- Network Morphology
- Evaluation
- GA Parameters
- Status

4.6.1 Training Constants

This is the application's main panel containing parameters controlling application's modes, input data sets, training algorithms and more.

Training Set List of data sets available.

Percentage Data If selected all time series data will be converted to a percentage changes format.

Training Algorithm List of training algorithms available.

Sigmoid\Hyperbolic Activation functions available.

Learning Rate Main parameter of a backpropagation training determining speed of the learning process.

Termination Criteria Backpropagation can be stopped by either setting a number of epochs(iterations) for a training or by setting a minimum MSE an ANN should achieve.

Evaluation Ratio Denotes the ratio between training and testing(evaluation) data sets e.g. 0.10 means that 10% of all training examples will be used as the evaluation data set and 90% will be used as the training data set.

Enter Inputs and Calculate Output The user can query the ANN by providing an input of his/her own choosing and clicking "Calculate Output" button. It is imperative that the format of the input data must be exactly the same as the one used to train an ANN. For example, if an ANN was trained using an input window equal to 3 it means that 3 observations must be provided in order to obtain results.

Train! Button used for starting training mode.

Evaluate! Button used for starting evaluation mode.

4.6.2 Network Morphology

This panel containing parameters controlling ANN morphology details, recurrence and weights values.

Number of Input Neurons Denotes the number of neurons in the input layer.

Number of Hidden Neurons Denotes the number of neurons in the hidden layer.

Number of Output Neurons Denotes the number of neurons in the output layer.

Input Window Denotes the number of bars(observations) used to predict a variable.

Recurrence The user can add Elman and/or Jordan recurrence to the ANN or use a CTRNN instead. Please note that CTRNN can't be used alongside Elman/Jordan recurrence and due to the nature of CTRNN only GA based training can be used.

Weights Scaling Factor Scales the range of initial random weights(see Equation 4.1.5).

Weights Shifting Factor Shifts the range of initial random weights(see Equation 4.1.5).

4.6.3 Evaluation

The evaluation panel contains values of statistical indicators.

Mean Squared Error Denotes the final value of MSE after training or evaluation finishes. For evaluation the default MSE value is -1000. Please note that MSE value is different for training and evaluation because is calculated using different data sets.

R squared Denotes the R^2 value of training and evaluation results. For evaluation the default MSE value is -1000. Please note that MSE value is different for training and evaluation because is calculated using different data sets.

Min Error Denotes the minimum absolute error during training. It is only calculated for backpropagation to check if the range of the error is not too excessive that could suggest some problems in backpropagation implementation. For GA based training its value is always 0.

Max Error Denotes the maximum absolute error during training. It is only calculated for backpropagation to check if the range of the error is not too excessive that could suggest some problems in backpropagation implementation. For GA based training its value is always 0.

4.6.4 GA Parameters

In scope of this project following parameters can be used to optimize GA based training:

Population size Denotes the number of individuals in a population. If population size is too small, GA will have a few possibilities to perform crossover and as a result only a small part of a search space will be explored. Too many individuals in a population will slow down the GA.

Crossover percent Denotes the percentage of top individuals that will have the privilege to mate.

Mutation percent Denotes the percentage of offspring that will be mutated. Too high level of mutation will result in performing a random search(sue to the lack of adaptation).

Crossover Cut Length Denotes the distance between the first and the second cut point during crossover.

Elitism Number of the fittest individuals that will be added to the next generation.

Termination Criteria Training process is finished after either a predefined number of generations or if the best solution will not change for a number of predefined iterations.

4.6.5 Status

This panel shows information about the current status of the application. All errors are displayed using Status panel.

4.6.6 Charts

In order to evaluate results more easily, the author has decided to present most important data using charts. Free Jfreechart [19] library was chosen to implement charts(see section **??**).

4.6.6.1 Main Chart

Main chart(Figure 4.26) presents data set used as input for ANN. Y-values represent close price value whereas x-values represent time in the format ”minutes:hours:day:month:year”.

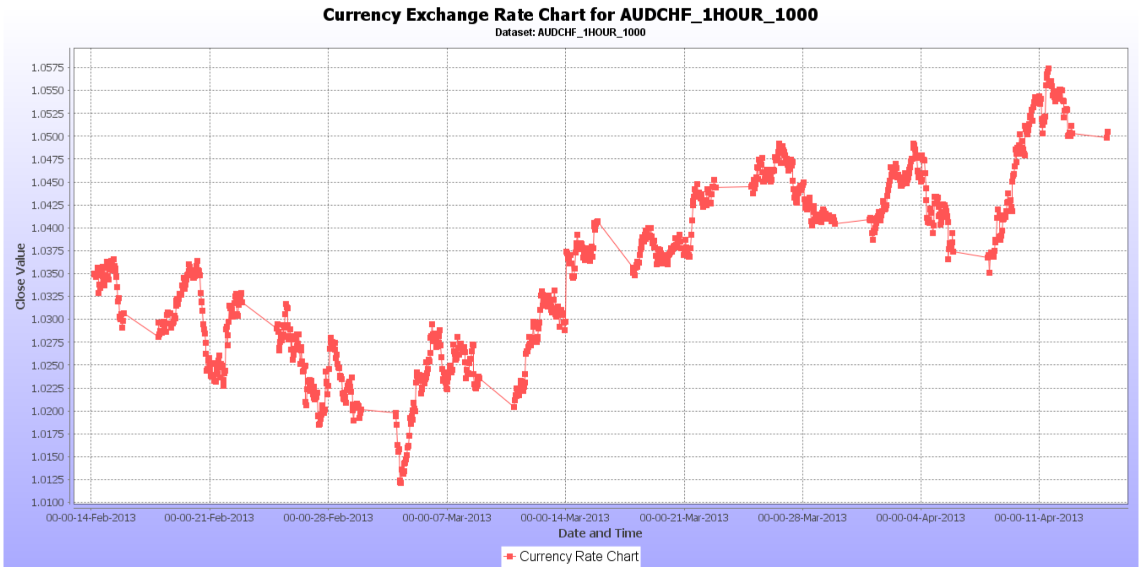


Figure 4.26: AUD/CHF 1 HOUR 1000 Observations Chart.

Visible ”holes” in data correspond to weekends. They have little or no effect on the quality of prediction(discussed in section 4.1.2).

4.6.6.2 Error Chart

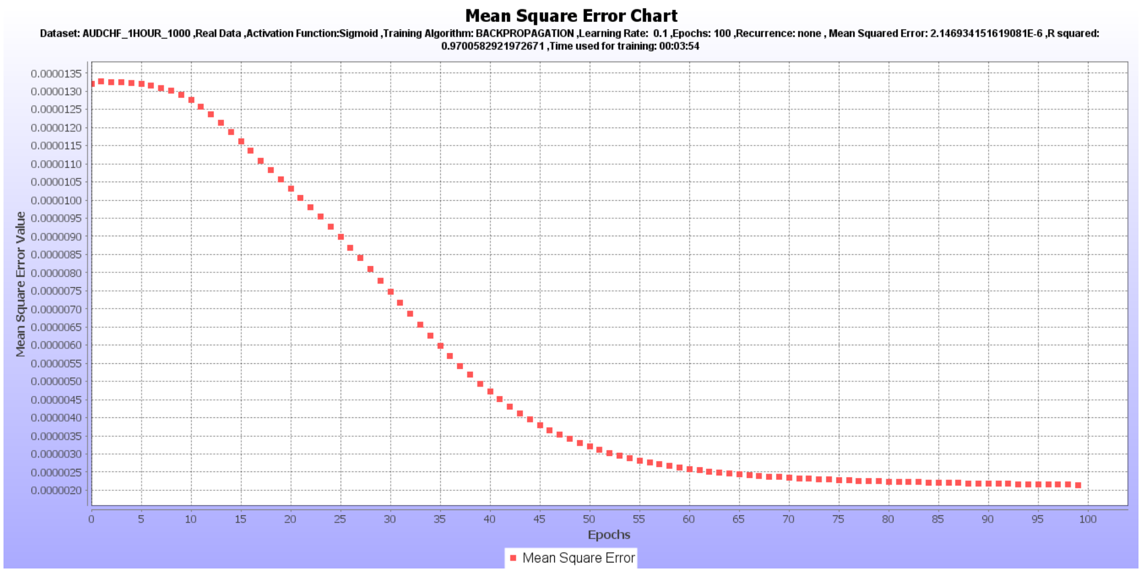
Error chart(Figure 4.27) presents how MSE changes over time. Ideally it should always converge to some small value, otherwise there is some problem with the training algorithm. In case of genetic algorithm based training the chart will be more stepwise because the MSE value corresponds to the MSE value of the fittest genotype which might be the same for a number of generations if elitism is enabled(see section

Y-values represent MSE values whereas x-values represent iterations(epochs). The top of the chart contains most important information about the ANN model.

It is important to note that this chart is created based on MSE calculated using training (not testing) data.This is the most important chart because it contains information about how well/bad ANN model has performed during training.

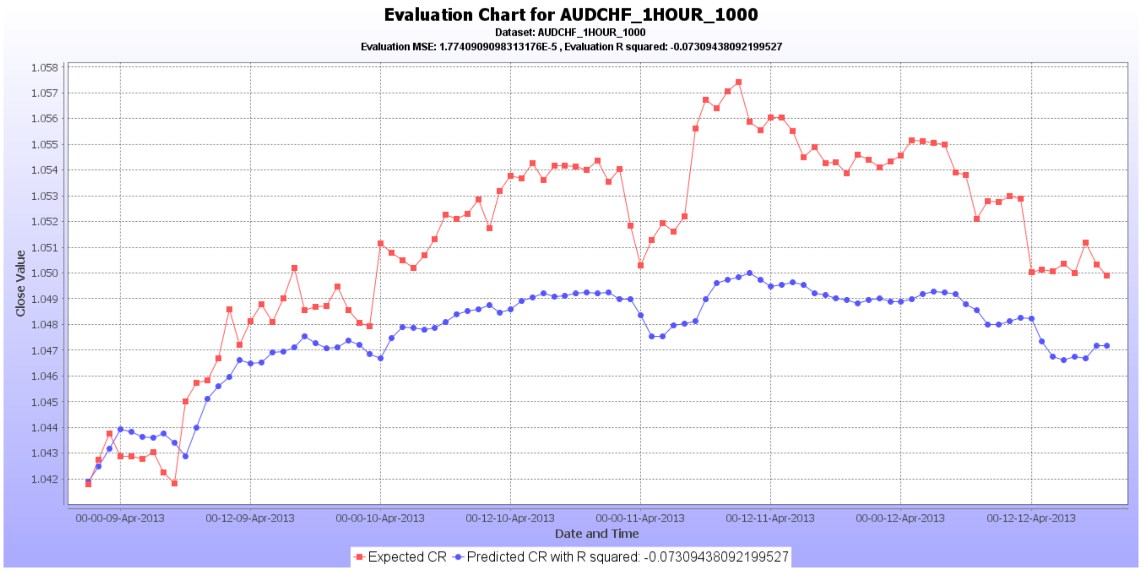
4.6.6.3 Evaluation Chart

This chart(Figure 4.28) presents predictions made by the ANN model compared with expected(perfect results). Predictions are calculated using testing (not training) data with ANN trained using training data. The top of the chart contains information about R^2 and MSE calculated using testing data. Similarly to main chart, y-values represent close price value whereas x-values represent time in the format ”minutes:hours:day:month:year”.



Dataset: AUDCHF_1HOUR_1000 ,Real Data ,Activation Function:Sigmoid ,Training Algorithm: BACKPROPAGATION ,Learning Rate: 0.1 ,Epochs: 100 ,Recurrence: none , Mean Squared Error: 2.14693415161908 1E-6 ,R squared: 0.9700582921972671 ,Time used for training: 00:03:54

Figure 4.27: Example Error Chart.



Dataset: AUDCHF_1HOUR_1000
Evaluation MSE: 1.7740909098313176E-5 , Evaluation R squared: -0.07309438092199527

Figure 4.28: Example Evaluation Chart.

4.6.6.4 Weights Chart

This chart(Figure 4.29) shows how connections weights are changing in correspondence to MSE changing. This information is useful when e.g. checking if backpropagation was not stuck in some local minimum. Y-values represent mean squared error value whereas x-values represent connections weights values.

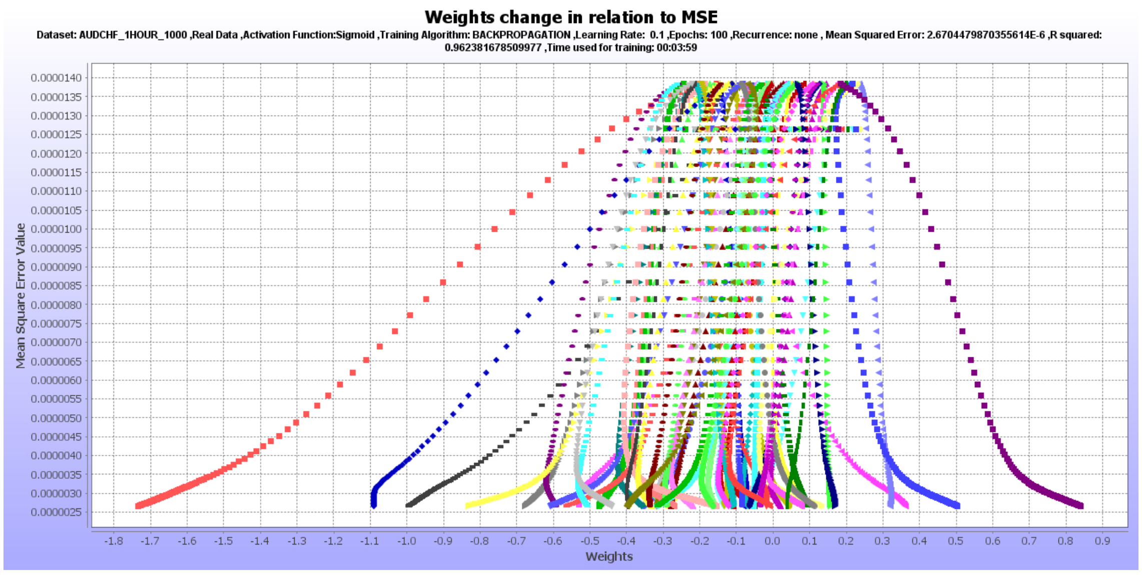


Figure 4.29: Example Weights Chart.

This chart was extensively used by the author in the early stages of the project when the backpropagation performance was tested using simple models. Unfortunately when ANN models became more sophisticated and the number of connections weights increased, the chart became unreadable. The chart legend was removed because it was so big that the chart itself was not visible.

4.6.6.5 Training Comparison Chart

This chart(Figure 4.30) shows how MSE was converging for a number of ANNs. Y-values represent MSE values whereas x-values represent iterations(epochs).

This chart is used for the purpose of comparing the speed of convergence of different ANNs as well as the final MSE values they manage to obtain.

4.6.6.6 Evaluation Comparison Chart

This chart(Figure 4.31) shows how successful ANNs are in predicting close price values. Y-values represent MSE values whereas x-values represent iterations(epochs).

This chart is used for the purpose of comparing the speed of convergence of different ANNs as well as the final MSE values they manage to obtain.

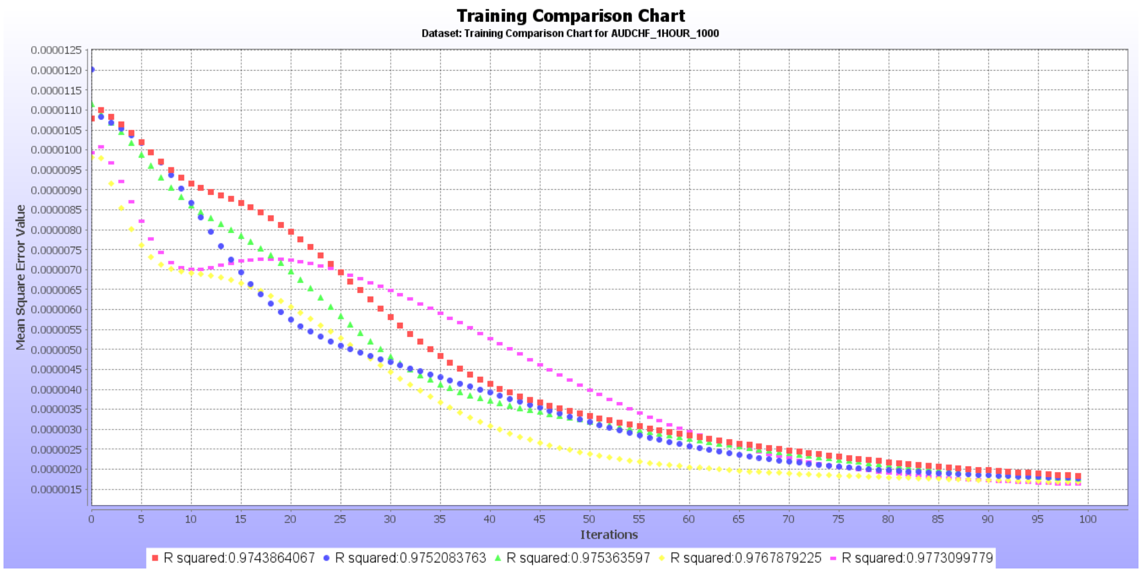


Figure 4.30: Example Training Comparison Chart.

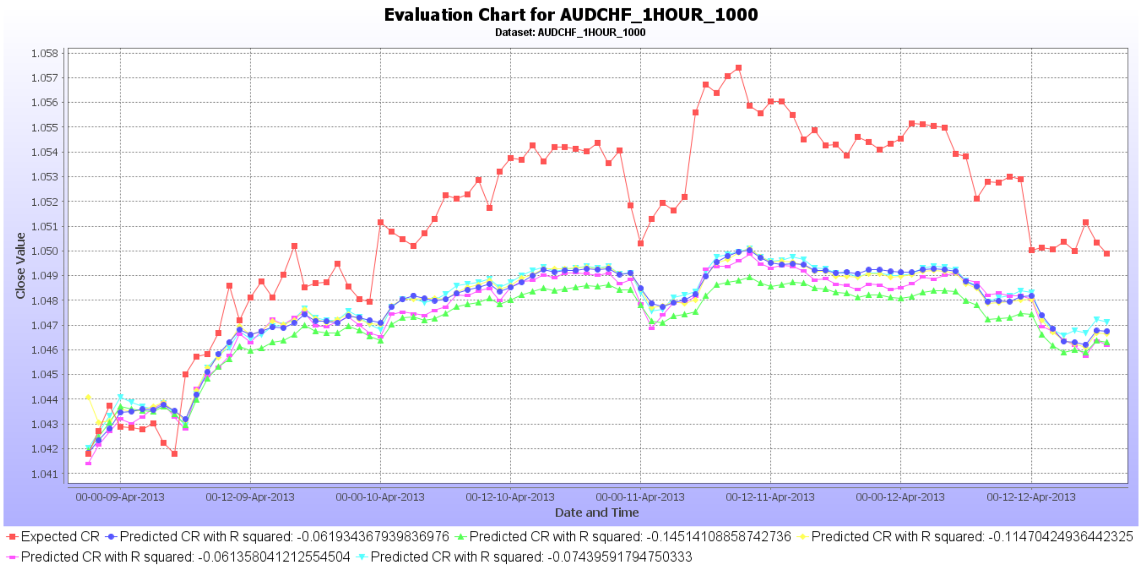


Figure 4.31: Example Evaluation Comparison Chart.

Chapter 5

Implementation

”Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.”

— Brian Kernighan

5.1 Implementation Options and Choices

This section discusses implementation choices made by the author during this project.

5.1.1 Morphologies and Training Algorithms

Due to the fact that the author had no prior knowledge about ANNs, models chosen to be implemented are derived from a number of scientific publications and consultations with Dr Elio Tuci. McCluskey [42], Tino et al. [49] and Kondratenko et al. [57] suggested that recurrent Elman and Jordan ANNs are one of the most successful models for prediction tasks. Jingtao Yao et al. [35] and Castiglione [23] proved that using multilayer feedforward perceptron trained using backpropagation algorithm can be very efficient and effective. Both backpropagation and multilayer feedforward perceptron were implemented in this project.Choosing correct training algorithm is as much important as ANN’s morphology. Riley et al. [34] and Nag et al. [21] showed that genetic algorithms can be used to train neural networks and provide better results than backpropagation. The author used their findings in this project by implementing Genetic Algorithm based training.

Dr Elio Tuci suggested using Continous Time Recurrent Neural Network (CTRNN) which was implemented as well. CTRNN are used to approximate dynamical systems and, according to author’s best knowledge, were never used to predict financial time series data. There is no scientific proof that CTRNNs would actually work and produce valid results, that was a proverbial ”cherry” of this project - working on something that was never used (according to scientic literature) before.

5.1.2 Metatrader 5

During research on using different AI techniques in financial markets the author came across the MetaTrader 5 platform [8]. MetaTrader 5 enables users to program an automated Trading Agent which will buy and sell currencies according to implemented strategy. It uses MQL5 programming language which is loosely based on C++. It has a simple IDE called MetaEditor as well. The author has spent a considerable amount of time exploring MetaTrader 5 capabilities and learning MQL5 programming language. The author considered implementing the whole project using this platform so some prototyping was carried out to test Metatrader 5 and MQL5 capabilities. The main goal of the prototyping process was to implement a simple ANN that could be initialized with random connection weights, calculate the net (weighted sums) values of each neuron and activate them using sigmoid function.

While working on this mini project the author came across a number of issues with MetaTrader 5 platform:

1. MQL5 has poor support for mathematical operations and a programmer must implementing a lot of mathematical functions himself. ANN are heavily based on mathematical models so a mathematical support is essential.
2. Debugging functionality is almost non-existing.
3. The MQL5 community is small. Issues raised by community members are waiting for a long time to be explained or resolved. The lack of support might be problematic in the future when e.g. I would came across some MQL5 issue and needed some advice.

The author has concluded that MetaTrader 5 platform is not mature enough for the purpose of this project. Nevertheless time invested in exploring MetaTrader 5 capabilities was not fruitless. Anyone can set up a free demo account using Metatrader 5 platform and as a result have access to historical data. The author has implemented a MQL5 script to export historical data in a CSV format which is used as input data for ANNs.

Initially the author wanted to increase project difficulty by implementing ANN models using a programming language that he was not familiar with, however due to time spent on exploring MetaTrader 5 and MQL5 capabilities, he had to verify his plans and fall back to a programming language he already used before - Java. Java provides a very good support for mathematical operations, especially those requiring high precision(e.g. BigDecimal class described in section 5.2.1), a simple to use GUI toolkit called Swing and is a well established object-oriented language with a huge active community.

5.1.3 Existing ANN technologies

As discussed in section 1.2 one of the main objectives of this project is to design, implement and test ANNs from scratch without using any external libraries or tools. The reason behind it is that the author wants to thoroughly study ANNs to very nitty-gritty parts of them, whereas existing ANN tools are more of a ”magic box” solutions - supply data, enter parameters and enjoy results. Nonetheless the author feels that the discussion about ANNs would be poorer without at least mentioning some of the most successful ANN tools/libraries which might be of interest to a reader:

WEKA Open source software that provides a number of AI algorithms implementations including ANNs [18]. Moreover it provides tool for data pre-processing and visualization as well.

FANN Open source neural network library originally implemented in C, but bindings to more than 15 programming language are available [4]. Similarly to WEKA it provides tools for data preprocesing and visualization. Currently only one training algorithm is provided which is backpropagation.

ENCOG Free programming framework that has a number of AI algorithms ready to be used once given right data and parameters [3]. Compared to FANN and WEKA it provides the biggest choice of ANN morpologies and training algorithms to be used(it misses CTRNN though).

It is important to note that existing ANN libraries are based on calculus and matrix calculations which result in a very efficient implementations. Due to the fact that this project was created based on a number of books and scientific publications, it's an educative project for the author and the author is not familiar with most of those mathematical calculations, ANNs implementations in this project are much more transparent and easier to understand even for people that never used ANN before. The mathematics was limited to absolute minimum and replaced by less efficient but easier to follow implementations.

5.2 Implementation Problems Encountered

This section discusses implementation problems encountered during this project.

5.2.1 BigDecimal vs. Double

During testing of some ANN models statistical indicators returned value NaN (not a number). This happens in Java when a division by 0 or infinity occurs. After some investigation the author identified the issue - some numbers were too small or too big for a double primitive type to represent and were returning infinity which was triggering NaN value.

Initially all numbers were represented by primitive double type. Double is the biggest primitive type to store a numerical value - it uses 8 bytes and covers a range from $4.940656458412e-324d$ to $1.797693134862e+308d$ (positive or negative) [29]. However some ANN calculations were falling outside of that range and causing problems. If double type can't represent a given number it classifies it as +infinity of -infinity.

Author's first solution was to represent all numbers using java BigDecimal class. BigDecimal was designed to provide higher precision calculations and transparent rounding behaviour that can be specified by a user [7]. Because it is immutable (after object's construction its state cannot be change) it is a little bit troublesome to use because it is easy to make a mistake, especially with running totals. BigDecimal allows precise representation of any real number that can be represented precisely in decimal notation hence there is no danger of obtaining infinity values and NaN as a result.

Unfortunately using BigDecimal to represent all numbers in ANN greatly decreased ANN efficiency to a point where e.g. performing a single epoch training using a backpropagation was taking more than an hour compared to about a minute previously. BigDecimal numbers requires much more computational power than double type numbers which was expected, however such enormous degradation of performance was not anticipated.

As a result the author decided to use BigDecimals only in the most important and risky calculations by converting double type numbers to BigDecimals and converting them back to doubles after calculations. If a BigDecimal number can't be converted to a double number because a double type is unable to represent it, the number is capped to some acceptable value. Of course this results in a lost of precision however when dealing with such small of big numbers (smaller than $10^{(} - 307)$ or bigger than $10^{(}307))$ capping them has a very small effect on the ANN performance.

5.2.2 Roulette Wheel Optimization

Roulette Wheel Selection (RWS) assigns the biggest part of a roulette wheel to the fittest individual. Standard RWS assigns a larger part of a roulette wheel to a chromosome with higher fitness. In scope of this project fitness of each chromosome is measured by the MSE value: smaller MSE, higher fitness value. This introduces a challenge because RWS works based on the maximum optimization whereas in this particular application the minimum optimization is desired.

In order to solve this problem the author has come up with a simple algorithm:

(a) Get a chromosome with the highest fitness (HF) value from a set of chromosomes chosen to mate.

(b) Subtract all other chromosomes' fitness values from HF.

As a result chromosomes with smallest fitness values become chromosomes with biggest fitness values and get biggest part of the roulette wheel. Of course this process is temporary and chromosomes' fitness values are not permanently overwritten.

A special case that needs to be considered is when we subtract a highest fitness value from itself which will result in 0. That means that the least fit chromosome in a set of all chromosomes chosen to mate will have much smaller chance to mate than others which is undesired. In order to prevent this a very small constant number is added to the minuend so a fitness value can never be equal to 0.

Chapter 6

Testing

”Program testing can be used to show the presence of bugs, but never to show their absence!”
— Edsger Dijkstra

6.1 Overall Approach to Testing

The main goal of the testing phase is to show that ANN can produce good predictions of currency exchange rate values. The secondary objective of testing is to extract general rules concerning ANN’s morphologies and training process that could help in determining the optimal parameters values in the future.

The ANN performance depends on the input data provided. Different data sets will require different parameters to obtain optimal results.By using a particular input data set the author will perform an analysis of all morphologies and training algorithms. The author has chosen the AUD \CHF currency pair with 1 hour time frame because it is a ”cross pair”. ”Cross Pairs” do not include USD and are less popular than ”major pairs” (include USD) meaning that their volatility is smaller and data distribution smoother. As a result it should be easier to detect patterns in AUD \CHF and produce more accurate predictions.

All tests were performed using the same testing computer with following parameters:

- Processor: Intel Core 2 Duo T6500 2.10GHz
- Memory : 4GB (2x2GB)
- OS : Windows 7 64-bit Professional

The initial parameters for the first test are as follows:

1. Dataset: AUD/CHF, 1000 observations every 1 hour between 04:00 14/02/2013 and 00:00 15/04/2013.
2. Morphology: multilayer feedforward ANN with 1 output neuron; the number of hidden neurons equal to 75% of the number of input neurons; number of input neurons depends on the number of inputs and input window size (see equation (4.1.5)).
3. Training : backpropagation using 0.1 learning rate. The Training finishes after 100 epochs.

4. Input Window : 4 bars(observations).
5. Output : Close Price value 1 hour in the future.
6. Training \Evaluation Ratio : 90% \10%

Each following test will try to determine best parameters values in order to come up with an ANN producing most accurate predictions.

6.2 Common Parameters Testing

6.2.1 Test 1: Single Input Testing

As discussed in section 4.1.1.2, the very first decision in designing an ANN is the choice of input variables. In scope of this project there are only two types of input variables to consider:

- Raw Data - represented by close price value. Technical Indicators - 6 technical indicators were chosen to be considered as inputs : MACD, MA, RSI, StochK, StochD, WilliamsR.

The goal of the first test was to determine which input variable has the strongest relationship with the output variable and, as a result, will produce best predictions. In other words, the author was looking for the highest correlation between the close price value(output) and input variables.

Technical Indicators Testing nr 1	Training Final MSE	Training R squared	Evaluation Final MSE	Evaluation R squared	Training Time Taken
Close Price	2.45E-06	0.965706155	2.63E-05	-0.071343352	00:01:21
Moving Average (MA)	5.16E-06	0.924993649	4.03E-05	-0.084633259	00:01:20
MACD	1.064316471	-7.04E-05	1.100427402	-1.37E-05	00:01:24
Relative Strength Index(RSI)	188.8525433	-4.37E-07	188.3506834	-7.57E-08	00:01:26
Stochastic K	14.48776796	-7.46E-06	14.31375735	4.59E-07	00:01:26
Stochastic Slow D	33.16342861	-2.92E-06	32.92047761	-4.15E-08	00:01:26
Williams R	3.52352025	5.71E-06	3.536884373	-1.23E-05	00:01:24

Figure 6.1: Testing Inputs Results 1

As expected using historical values of close price to predict its future price are producing the best results. MA and MACD seems to have the weakest relation with the output value.

Evaluation results are confirming previous findings. The worst prediction(green color) was produced by RSI input whereas the best predictions (light blue-MA and light pink-Close Price) are not visible because they overlap with the expected/perfect results (red) and zooming in is needed to actually see them:

6.2.2 Test 2: Two Input Testing

Previous test has proven that using past close price values returns the best predictions. The main goal of this test is to check if using additional input will improve prediction results. Since the number of inputs increased to 2, the number of input neurons must be increased to

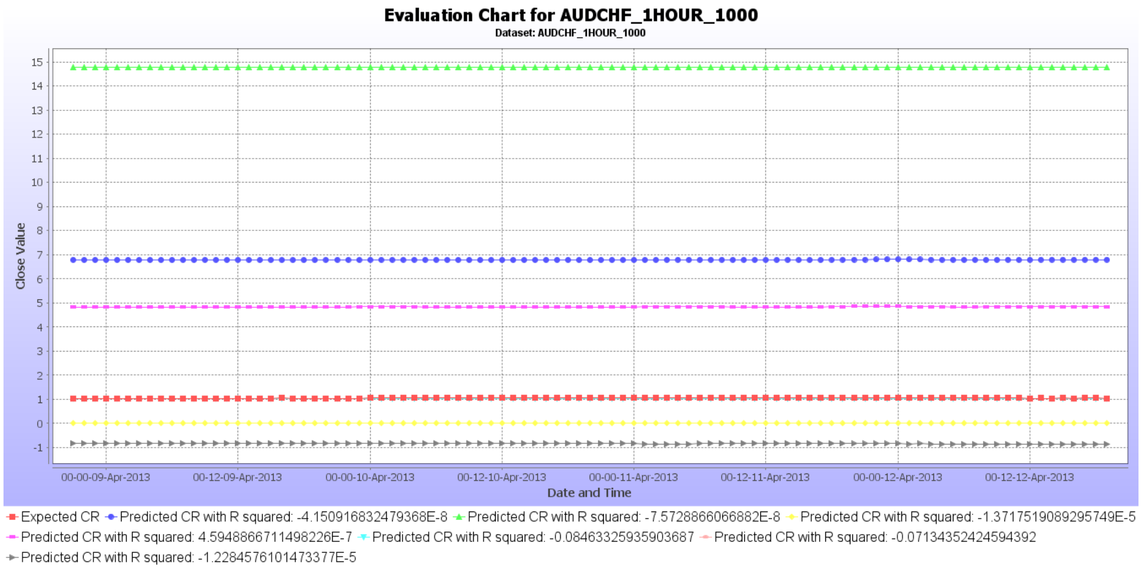


Figure 6.2: Inputs Evaluation Results 1

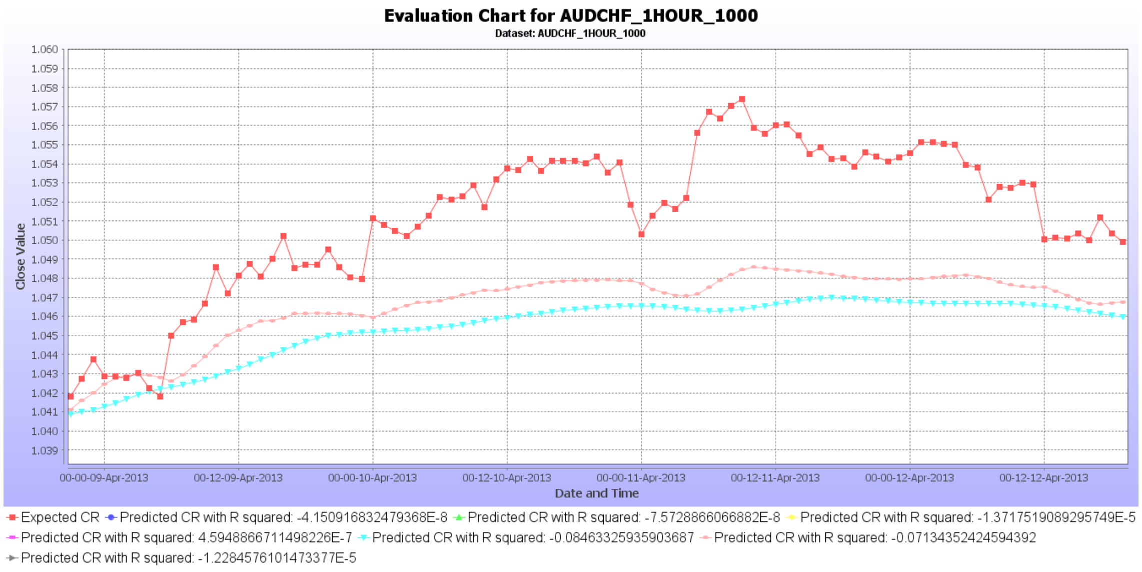


Figure 6.3: Inputs Evaluation Results 1 Zoomed In.

Technical Indicators Testing nr 2	Training Final MSE	Training R squared	Evaluation Final MSE	Evaluation R squared	Training Time Taken
Close Price+MA	2.67E-06	0.962381679	2.63E-05	-0.041519539	00:03:59
Close Price+MACD	2.22E-06	0.968879224	2.08E-05	-0.087051879	00:04:01
Close Price+RSI	1.72E-06	0.976018226	1.89E-05	-0.04036615	00:03:56
Close Price+StochK	2.15E-06	0.970058292	2.45E-05	-0.058955315	00:03:54
Close Price + StochasticD	2.26E-06	0.968415328	2.49E-05	-0.072126127	00:03:55
Close Price + WilliamsR	1.76E-06	0.975464714	2.30E-05	-0.081313345	00:03:55
Average	2.13E-06	9.70E-01	2.31E-05	-6.36E-02	00:03:57

Figure 6.4: Testing Inputs Results 2

8 (input window=4 multiplied by number of input types=2) and the number of hidden neurons to 6 (75% of 8).

Table 6.4 shows that Close Price + MA combo produce the worst results whereas the Close Price + RSI combo produce the best results. Those are interesting results because Test 1 showed that RSI on its own produces worst results of all inputs whereas paired with Close Price returns best results. More importantly those results are better than using Close Price on its own. It is worth mentioning that the average time of training increased approximately 2.5 times due to the increase of number of input and hidden neurons and, as a result, the number of connections weights to be updated.

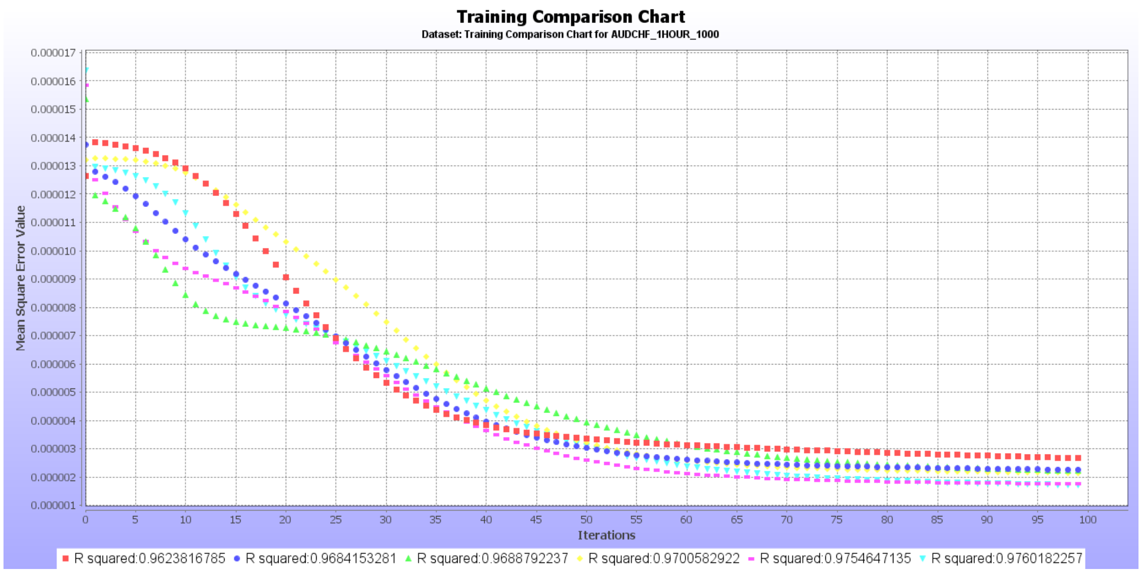


Figure 6.5: Inputs MSE Comparison 2

The MSE convergence comparison shows that two best results (pink-Close Price + WilliamsR and light blue-Close Price + RSI) are converging in the fastest pace (their lines are ”flatter” and, in general, on average closer to 0 during all epochs than any other combo).

Figure 6.6 is confirming previous results, Close Price+RSI (pink) is the closest line to the expected results(red). In general all results are better or comparable with the best results of Test 1. This proves that using raw data enriched with technical indicators values improves ANN generalization ability.

6.2.3 Test 3: Three Input Testing

If using two inputs improves ANN performance let’s add yet another input variable. The main goal of this test is to check if using 3 input types will improve ANN generalization ability. Since the number of inputs increased to 3, the number of input neurons must be increased to 12 (input window=4 multiplied by number of input types=3) and the number of hidden neurons to 9 (75% of 12).

Adding third input increased general performance during training (the average MSE is lower and the average R^2 is higher) however the evaluation’s results are more ambiguous. Although the best MSE result is better than the best result obtained using two input types, the best

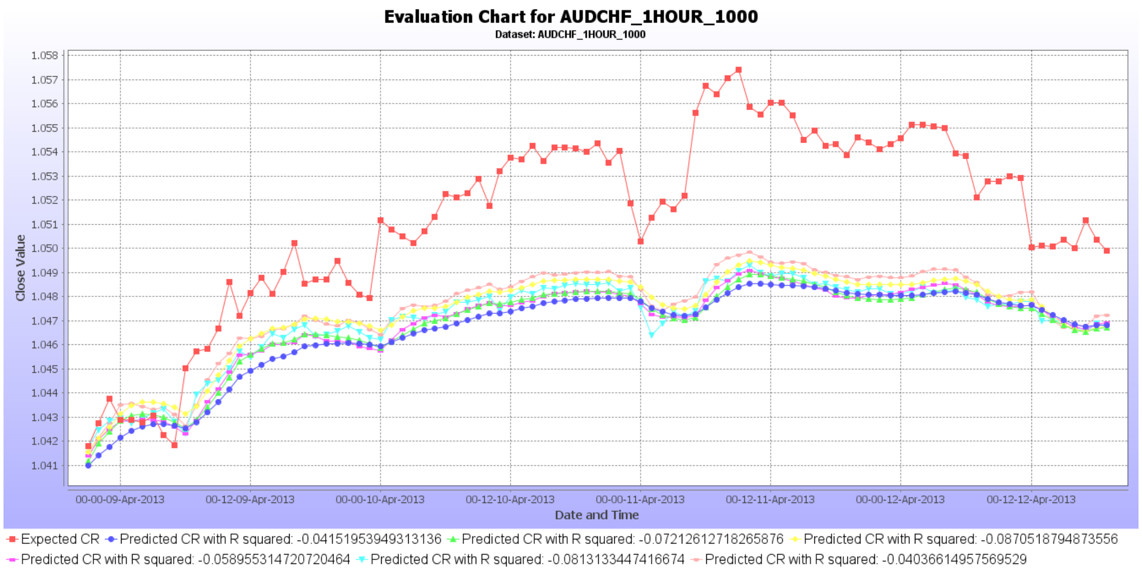
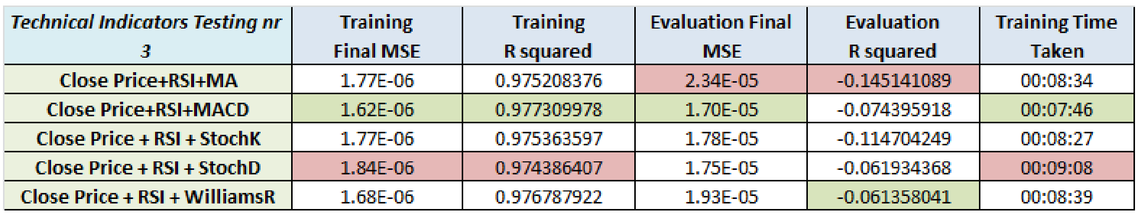


Figure 6.6: Inputs Evaluation Comparison 2



Technical Indicators Testing nr 3	Training Final MSE	Training R squared	Evaluation Final MSE	Evaluation R squared	Training Time Taken
Close Price+RSI+MA	1.77E-06	0.975208376	2.34E-05	-0.145141089	00:08:34
Close Price+RSI+MACD	1.62E-06	0.977309978	1.70E-05	-0.074395918	00:07:46
Close Price + RSI + StochK	1.77E-06	0.975363597	1.78E-05	-0.114704249	00:08:27
Close Price + RSI + StochD	1.84E-06	0.974386407	1.75E-05	-0.061934368	00:09:08
Close Price + RSI + WilliamsR	1.68E-06	0.976787922	1.93E-05	-0.061358041	00:08:39

Figure 6.7: Testing Inputs Results 3

R^2 result(-0.0613580412125545) is worse than the best result when using two input types(-0.0403661495756952). The average time needed to train a network using 3 different inputs increased approximately 2 times.

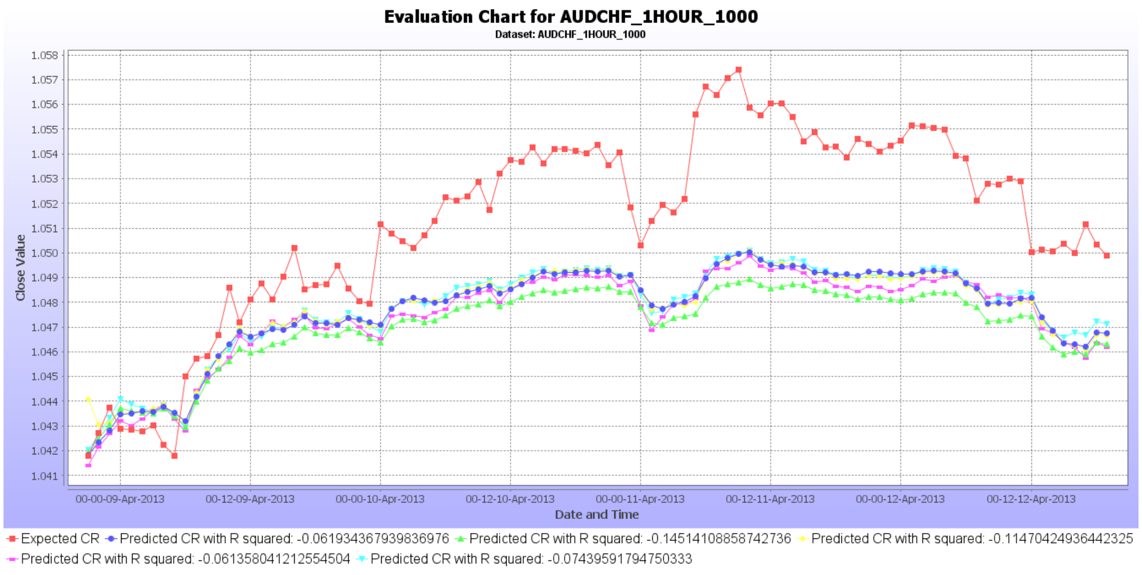


Figure 6.8: Input Evaluation Comparison 3

Figure 6.8 shows that Close Price+RSI+MACD combo (light blue) produced the best results (closest to the red, expected results). That confirms that this combo is the best of all 3 input combos even though its R^2 is not the best one. In general Close Price+RSI+MACD combo achieved 3 out of 4 best possible result(across all 3 tests), graph closest to expected close price value during evaluation and one of the best R^2 value hence the author has chosen these inputs to be used in future tests as ones producing best results.

It is interesting to note that some of the models using 3 input types produced worse predictions than models using 2 input types. This proves that adding more input types to an ANN does not always result in better performance and, in fact, it can make ANN results worse by introducing noise to the dataset(overfitting).

The author suspects that using 4 inputs would improve results a little bit however the time needed to train a network would increase at least 2 times to around 15 minuts. Having in mind that this is the simplest ANN model and adding recurrence and/or using genetic algorithm based training will drastically increase training time(more neurons, multiple ANN generations consisting of dozens and hundreds ANNs each), the author decided to use only 3 inputs in all future testing so the time of training will not take more than couple hours for most sophisticated models.

6.2.4 Test 4: Input Window Testing

Input window size determines how many bars(observations) of each input type an ANN will use as input data. It is important parameter because it directly affects the number of input neurons and indirectly the number of hidden neurons and training time.

As per Figure 6.9 Using 1 bar and using 5 bars as inputs returns worst results both during training as well as evaluation. This is understandable because using 1 bar is a classic example

Input Window Testing	Training Final MSE	Training R squared	Evaluation Final MSE	Evaluation R squared	Training Time Taken
1	1.81E-06	0.974351216	1.93E-05	-0.11592111	00:00:50
2	1.66E-06	0.976789633	1.85E-05	-0.036669146	00:02:14
3	1.65E-06	0.976794555	1.71E-05	-0.049365014	00:05:00
4	1.65E-06	0.97702892	1.77E-05	-0.073094381	00:07:43
5	1.82E-06	0.97446485	1.86E-05	-0.133868134	00:12:26

Figure 6.9: Input Window Testing Results

of *underfitting*. With 1 bar and 3 input types, the ANN has 3 input neurons and 2 hidden neurons which is not sufficient to determine complex, more than 1 bar long patterns in a data set. Using input window equal to 5 bars results in *overfitting*. With 5 bars and 3 input types, ANN has 15 input neurons and 12 hidden neurons and is able to fit not only data patterns but noise as well.

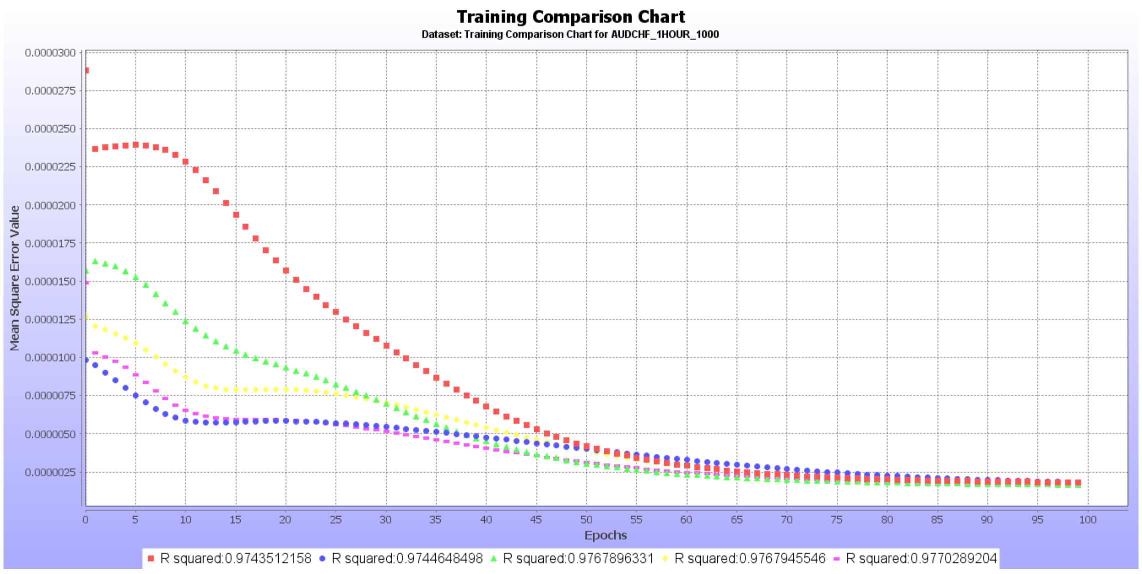


Figure 6.10: Input Window MSE Convergence Comparison

As per Figure 6.10 input window size equal to 5(dark blue), 4(pink) and 3(yellow) are converging faster than other two meaning that they provide better results in the same training time.

All evaluation results are comparable(Figure 6.11.

Taking all into consideration, the ANN using input window size equal to 3 provides best results. Although input window equal to 4 provides similar results it performs worse during evaluation phase signalling possible *overfitting* condition. If input window is equal to 3 and 3 input types are used, as a result ANN will have 9 input neurons, 7 hidden neurons (approximately 75%) and 1 output neuron.

6.2.5 Test 5: Training/Testing Ratio Testing

There is no general algorithm to find ratio between training and testing data sets that will produce the best results. Moreover the training/testing ratio is very problem domain specific,

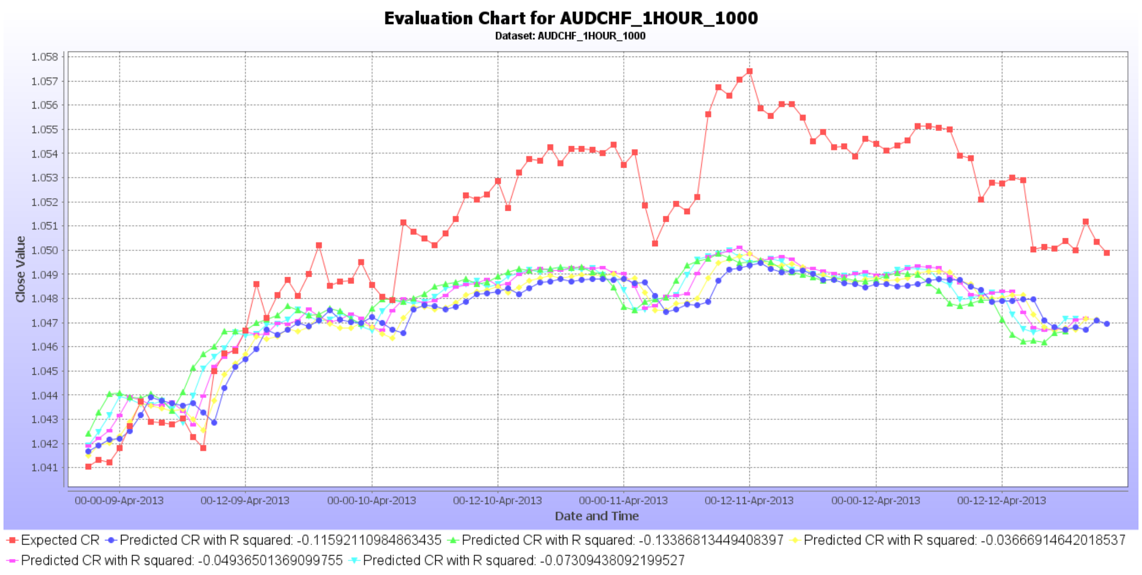


Figure 6.11: Input Window Evaluation Comparison.

hence trial and error approach is often used. The rule of thumb is that the training data set should be much larger than the testing data set. This rule was applied by the author when deciding which test cases to use.

Training/Testing Ratio Testing	Training Final MSE	Training R squared	Evaluation Final MSE	Evaluation R squared	Training Time Taken
70/30	1.94E-06	0.968908237	6.21E-06	0.328530726	00:03:56
80/20	1.92E-06	0.971505534	1.14E-05	0.259464855	00:04:25
90/10	1.82E-06	0.97422851	1.90E-05	-0.175982285	00:05:01
95/5	1.79E-06	0.977031532	1.33E-05	-0.137918769	00:05:13

Figure 6.12: Training\Testing ratio results.

Figure 6.12 shows that the best results during training were achieved using 95/5 ratio whereas the best evaluation results were achieved using 70/30 ratio.

Figure 6.13 and figure 6.14 show that 70/30 ratio produces much better results. Moreover since training set is smaller the training time is much smaller as well. As a result, the author has decided to use this ratio in future testing.

6.2.6 Test 6: Hidden Neurons Testing

So far the number of hidden neurons was chosen using one of the rules of thumb described in section 4.1.7.2, namely the number of hidden neurons is equal to 3/4 of the number of input neurons. Too many hidden neurons can cause overfitting and too long training times, too few hidden neurons can cause underfitting. Because of that the number of hidden neurons is one of the most important parameters of an ANN.

Figure 6.15 shows that using 6 hidden neurons provide better results during training however using 7 hidden neurons provide better results during evaluation.

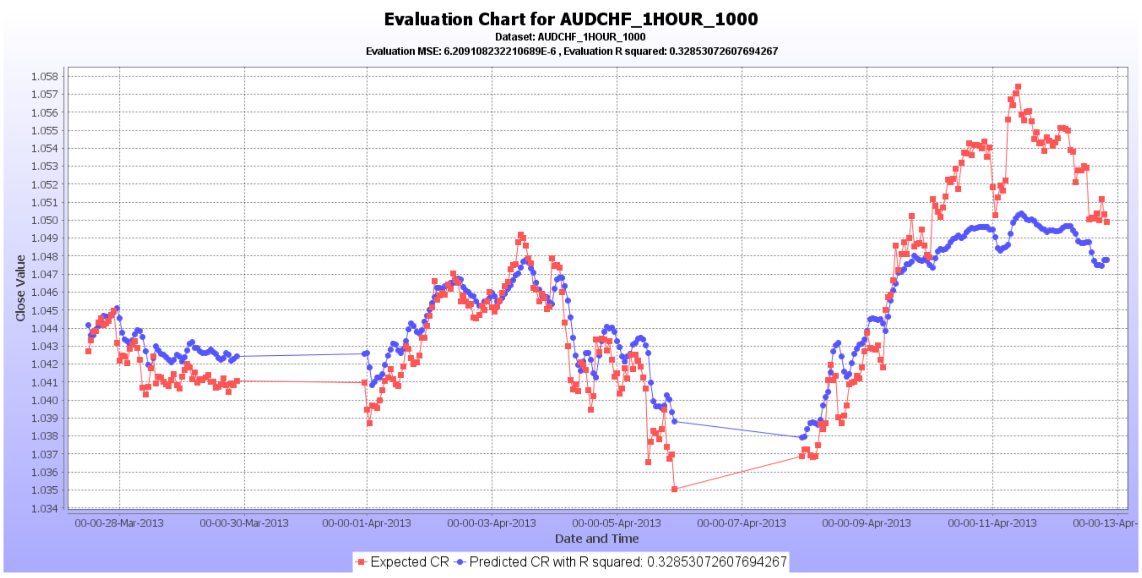


Figure 6.13: 70\30 Training\Testing ratio results.

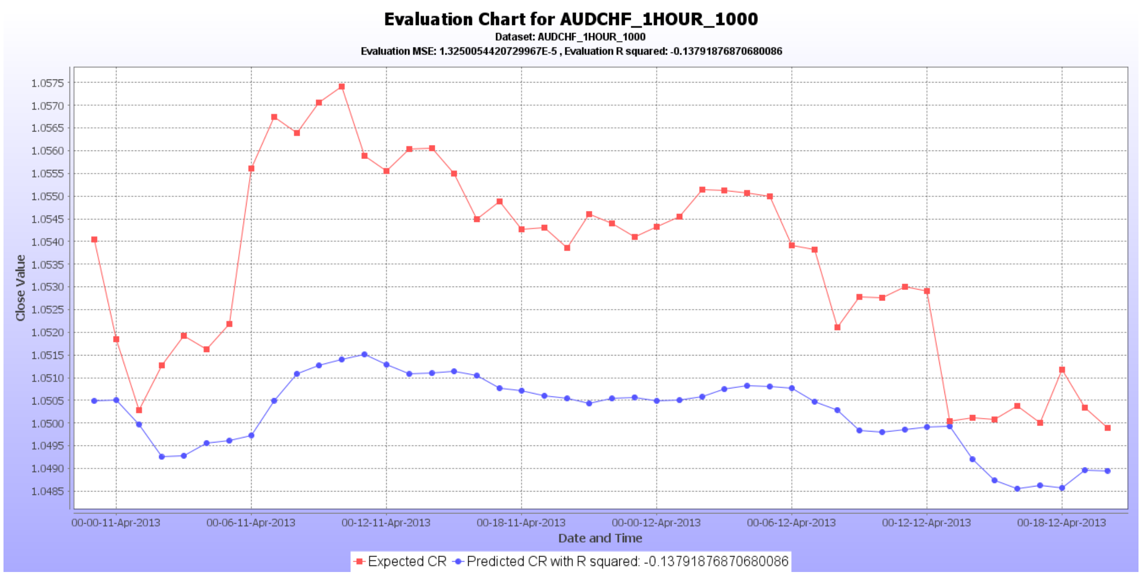
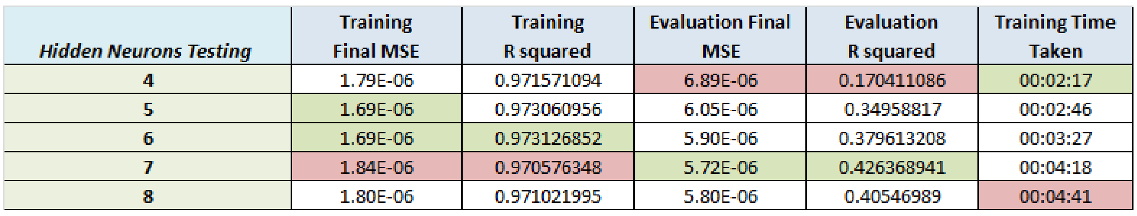


Figure 6.14: 95\05 Training\Testing ratio results.



Hidden Neurons Testing	Training Final MSE	Training R squared	Evaluation Final MSE	Evaluation R squared	Training Time Taken
4	1.79E-06	0.971571094	6.89E-06	0.170411086	00:02:17
5	1.69E-06	0.973060956	6.05E-06	0.34958817	00:02:46
6	1.69E-06	0.973126852	5.90E-06	0.379613208	00:03:27
7	1.84E-06	0.970576348	5.72E-06	0.426368941	00:04:18
8	1.80E-06	0.971021995	5.80E-06	0.40546989	00:04:41

Figure 6.15: Hidden neurons testing results.

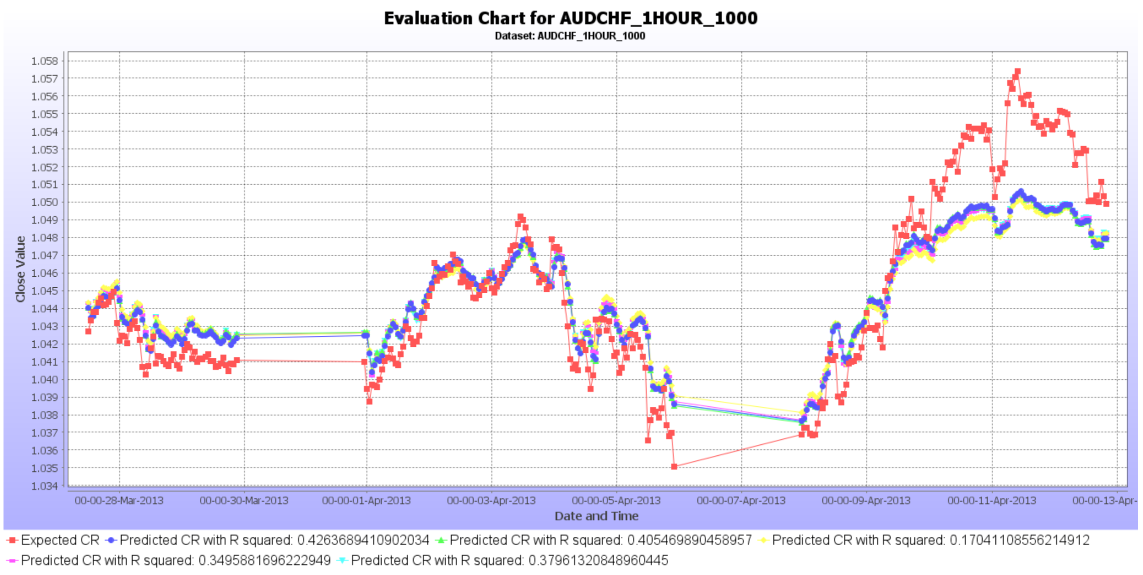


Figure 6.16: Hidden neurons evaluation results.

Figure 6.16 shows that all ANN models produce comparable results. All models are very good in predicting first and second part of the graph(separated by weekends), however they are failing in the third part.

The author decided to use ANN with 7 hidden neurons because it produces best predictions using testing data.

There are algorithms that help to determine the correct number of hidden neurons like e.g. Selective Pruning [13] unfortunately the author did not have enough time to implement it.

6.3 Backpropagation Testing

The backpropagation requires only learning rate parameter to perform training. Too small learning rate will slow down the learning process whereas if the learning rate is too high the algorithm can oscillate and become unstable. The optimal learning rate changes during the training process, as the algorithm moves across the performance surface. Ideally the learning rate should be updated during the training process. There are a number of algorithms that can be use to dynamically optimize learning rate during training (see [47] for more info) however the author was not able to implement them due to limited time.

Backpropagation Learning Rate Testing	**Training Final MSE**	**Training R squared**	**Evaluation Final MSE**	**Evaluation R squared**	**Training Time Taken**
0.1	1.78E-06	0.971776598	5.99E-06	0.371657413	00:04:18
0.2	1.57E-06	0.975481307	7.01E-06	0.109622446	00:04:00
0.5	1.45E-06	0.977593915	1.08E-05	-1.013812269	00:04:53
0.7	1.42E-06	0.978078126	1.25E-05	-1.620904145	00:03:54
1	1.38E-06	0.97880372	1.32E-05	-2.100474893	00:03:54
2	1.40E-06	0.978502436	1.90E-05	-5.380150043	00:03:59
5	1.50E-06	0.977096652	2.75E-05	-6.638406901	00:04:04

Figure 6.17: Learning Rate Testing Results.

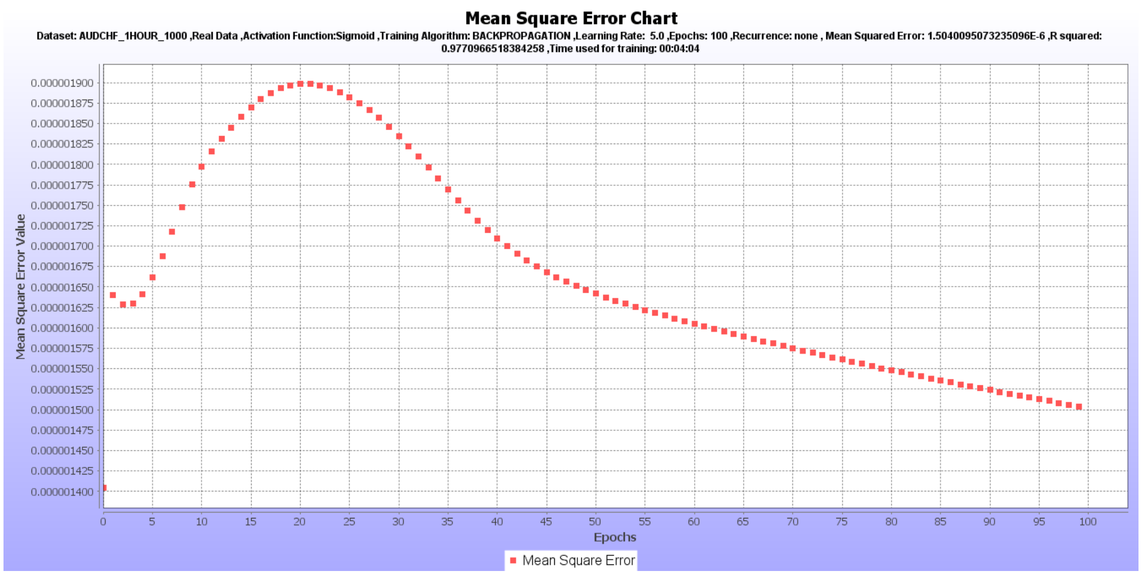


Figure 6.18: MSE growing during early stages due to too high learning rate.

Figure 6.17 shows that both training and evaluation results were getting worse for learning rates higher than 0.7. Figure 6.18 shows that in the case of the ANN with learning rate equal to 5 the MSE was actually growing with each epoch at the beginning. Too high learning rate causes the gradient descent to overshoot and oscillate which results in higher MSE. This is a clear sign that the learning rate should be reduced.

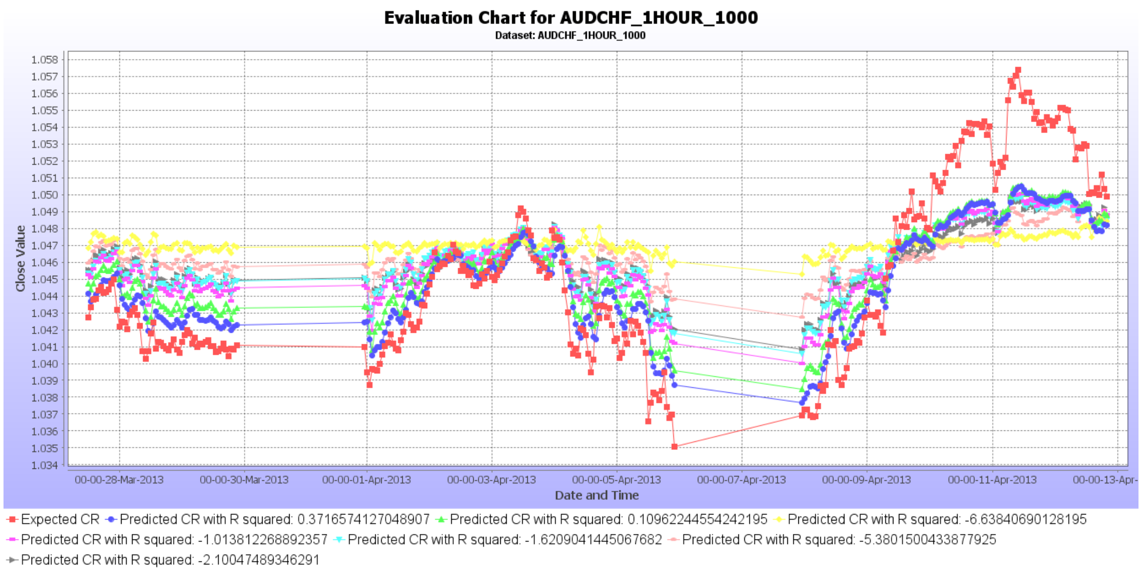
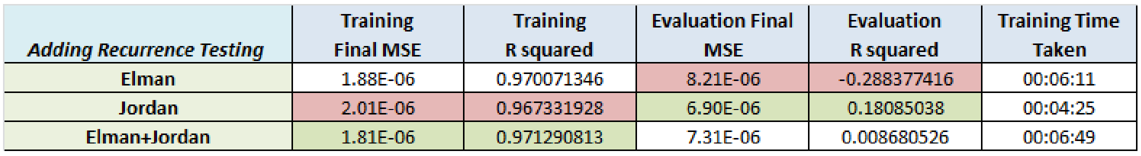


Figure 6.19: Learning Rate Testing Evaluation Results.

Figure 6.19 confirms that the best evaluation results are produced with the smallest learning rate. In the scope of this project the convergence speed is not the top priority hence small learning rates are acceptable. As a result the author has decided that the learning rate will be equal to 0.1 in all relevant future cases.

6.4 Recurrence Testing

Many researchers that added recurrence to the ANN agree that it improves the quality of predictions [42] [49] [57].

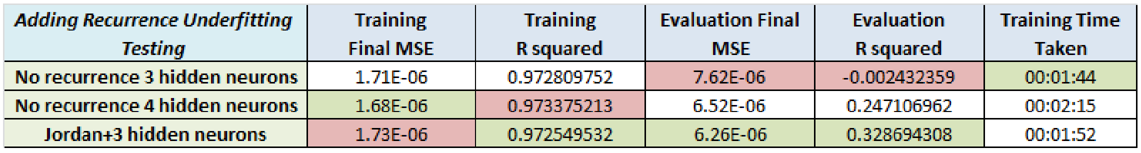


Adding Recurrence Testing	Training Final MSE	Training R squared	Evaluation Final MSE	Evaluation R squared	Training Time Taken
Elman	1.88E-06	0.970071346	8.21E-06	-0.288377416	00:06:11
Jordan	2.01E-06	0.967331928	6.90E-06	0.18085038	00:04:25
Elman+Jordan	1.81E-06	0.971290813	7.31E-06	0.008680526	00:06:49

Figure 6.20: Recurrence Testing Results.

As expected using both Elman and Jordan provides best training results and very good results evaluation results. Interestingly enough the evaluation results obtained by recurrent ANNs are worse than the results obtained by best non-recurrent MFANNs (see Figure 6.17). The author suspected that there is a problem in the implementation however no bug was identified. One hypothesis that could explain this phenomenon is that adding recurrence to the ANN causes overfitting of the data. For this particular data set MFANNs best results are very close to perfect (perfect results are when the MSE value is equal to 0 and the R^2 value is equal to 1) and extra Elman and Jordan neurons, which are redundant since the current ANN morphology is able to estimate the target function very closely, make the ANN to learn noise which results in worse results.

In order to test the hypothesis that adding recurrence improves the ANN generalization ability , the author has set up a small test suite. The number of hidden neurons in the best MFANN was reduced to 3 to introduce underfitting- this is the benchmark model. Adding recurrence to this model should improve results more significantly than just increasing the number of hidden neurons. In both cases we are increasing the number of neurons in the ANN however adding recurrent neurons should increase the ANN's performance more than adding the same amount of hidden neurons. It is known that increasing the number of hidden neurons in MFANN which is underfitting data will improve its performance. If adding recurrence to such model would improve the performance more than adding hidden neurons it would mean that adding recurrence can improve ANN generalization ability.



Adding Recurrence Underfitting Testing	Training Final MSE	Training R squared	Evaluation Final MSE	Evaluation R squared	Training Time Taken
No recurrence 3 hidden neurons	1.71E-06	0.972809752	7.62E-06	-0.002432359	00:01:44
No recurrence 4 hidden neurons	1.68E-06	0.973375213	6.52E-06	0.247106962	00:02:15
Jordan+3 hidden neurons	1.73E-06	0.972549532	6.26E-06	0.328694308	00:01:52

Figure 6.21: Comparing recurrence and hidden neurons addition effects.

Figure 6.21 shows that adding a single Jordan recurrent neuron to the benchmark model improved the ANN generalization performance more than adding a single hidden neuron. Thereby adding recurrence to the ANN can improve the quality of predictions.

To summarise, for this particular data set and particular ANN adding recurrence does not improve the ANN generalization ability because the ANN is already producing very good results and adding recurrence most probably makes the ANN overfit the data. In general adding recurrence can improve the quality of predictions produced by an ANN.

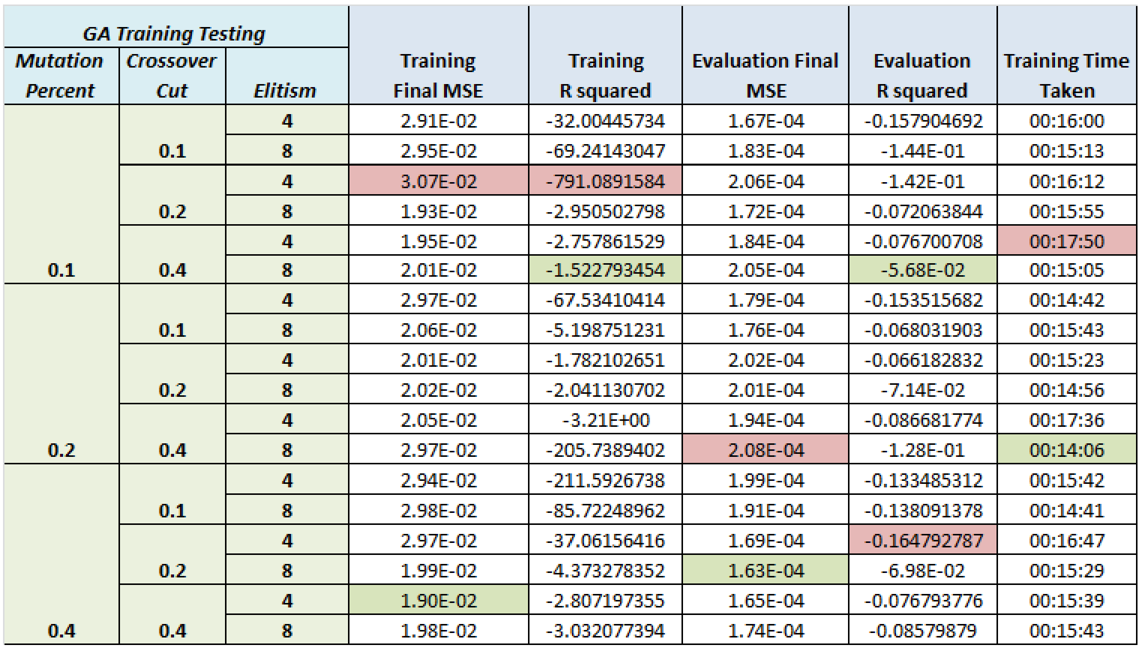
6.5 GA Testing

GA based training parameters determine the level of genetic variety across the population. All of them are very closely related hence trying to deduce their optimal values for a particular problem domain using an iterative testing approach(testing one parameter at a time) is meaningless since e.g. increasing one parameter and decreasing another parameter at the same time may result in no effect at all.

The initial values for GA parameters were chosen in consulation with Dr Elio Tuci who has used GA for many years:

- Population size = 100
- Crossover percent = 0.70
- Mutation percent = 0.10
- Crossover Cut Length = 0.10
- Elitizm = 4

In order to create a next generation of ANNs with parameters which were chosen during testing so far, the computer used for testing requires about 90 seconds. This is due to the fact that the fitness of each individual must be measured over all training examples. Backpropagation uses all training examples exactly once and always in the same order during a single epoch whereas GA based training in order to create a new generation uses all training example for each individual meaning that each training example is used the number of times equal to the population size. That is why GA is much slower than backpropagation. In order to keep the training time reasonable the author has decided to create 10 generations before terminating.



GA Training Testing							
Mutation Percent	*Crossover Cut*	*Elitism*	Training Final MSE	Training R squared	Evaluation Final MSE	Evaluation R squared	Training Time Taken
0.1	0.1	4	2.91E-02	-32.00445734	1.67E-04	-0.157904692	00:16:00
		8	2.95E-02	-69.24143047	1.83E-04	-1.44E-01	00:15:13
	0.2	4	3.07E-02	-791.0891584	2.06E-04	-1.42E-01	00:16:12
		8	1.93E-02	-2.950502798	1.72E-04	-0.072063844	00:15:55
	0.4	4	1.95E-02	-2.757861529	1.84E-04	-0.076700708	00:17:50
		8	2.01E-02	-1.522793454	2.05E-04	-5.68E-02	00:15:05
0.2	0.1	4	2.97E-02	-67.53410414	1.79E-04	-0.153515682	00:14:42
		8	2.06E-02	-5.198751231	1.76E-04	-0.068031903	00:15:43
	0.2	4	2.01E-02	-1.782102651	2.02E-04	-0.066182832	00:15:23
		8	2.02E-02	-2.041130702	2.01E-04	-7.14E-02	00:14:56
	0.4	4	2.05E-02	-3.21E+00	1.94E-04	-0.086681774	00:17:36
		8	2.97E-02	-205.7389402	2.08E-04	-1.28E-01	00:14:06
0.4	0.1	4	2.94E-02	-211.5926738	1.99E-04	-0.133485312	00:15:42
		8	2.98E-02	-85.72248962	1.91E-04	-0.138091378	00:14:41
	0.2	4	2.97E-02	-37.06156416	1.69E-04	-0.164792787	00:16:47
		8	1.99E-02	-4.373278352	1.63E-04	-6.98E-02	00:15:29
	0.4	4	1.90E-02	-2.807197355	1.65E-04	-0.076793776	00:15:39
		8	1.98E-02	-3.032077394	1.74E-04	-0.08579879	00:15:43

Figure 6.22: GA Testing Results

Figure 6.22 shows the results of testing by simultaneously changing values of selected parameters. The author decided that there is no reason to modify population size and crossover percent since those parameters have smaller effect on genetic variety than mutation percent, crossover cut length or elitism. All results are very similar and it is hard to choose the model that produced much better results than the rest.

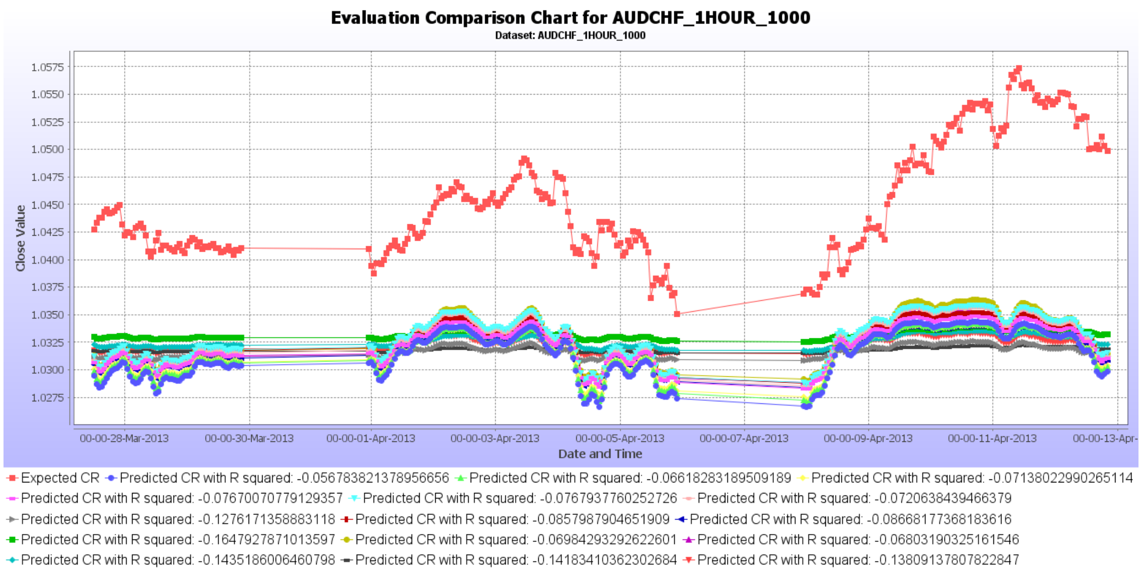


Figure 6.23: GA Evaluation Comparison Results.

Figure 6.23 confirms that all models produced comparable results. More importantly none of tested models produced positive R^2 value for evaluation data. The author suspects that the training time was simply too short to provide better results.

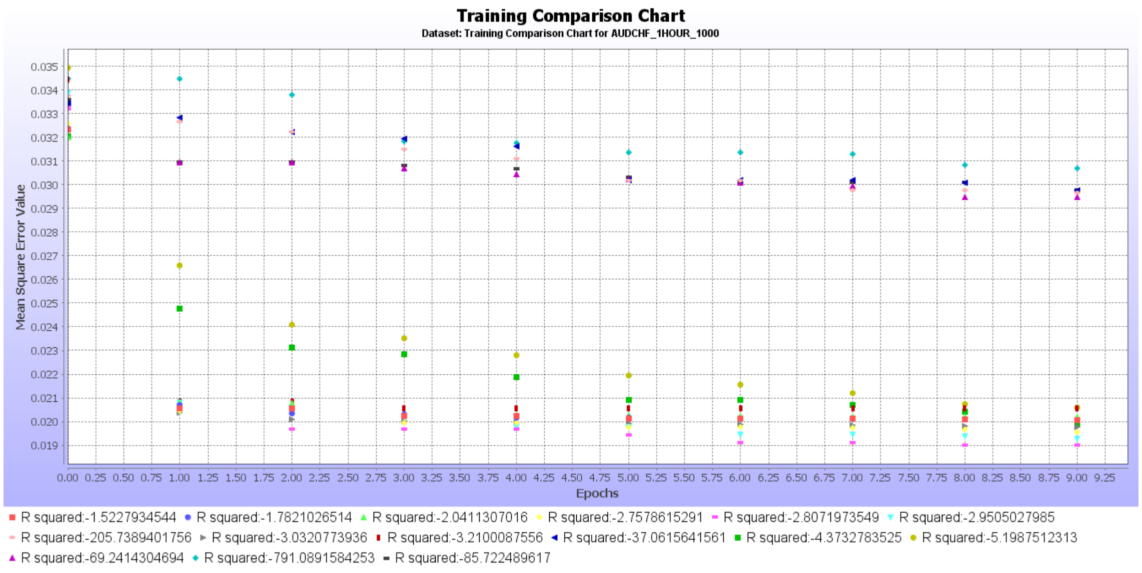


Figure 6.24: GA Training Comparison Results.

Figure 6.24 shows that in general the MSE value was converging much slower than in the case of backpropagation. The author suspects that this is due to the large amount of genes in a chromosome. All weights influence each other meaning that even a single weight change can change ANN performance from one of the best to one of the worst. The bigger the amount of genes, the harder it is to find optimal solution.

6.6 CTRNN Testing

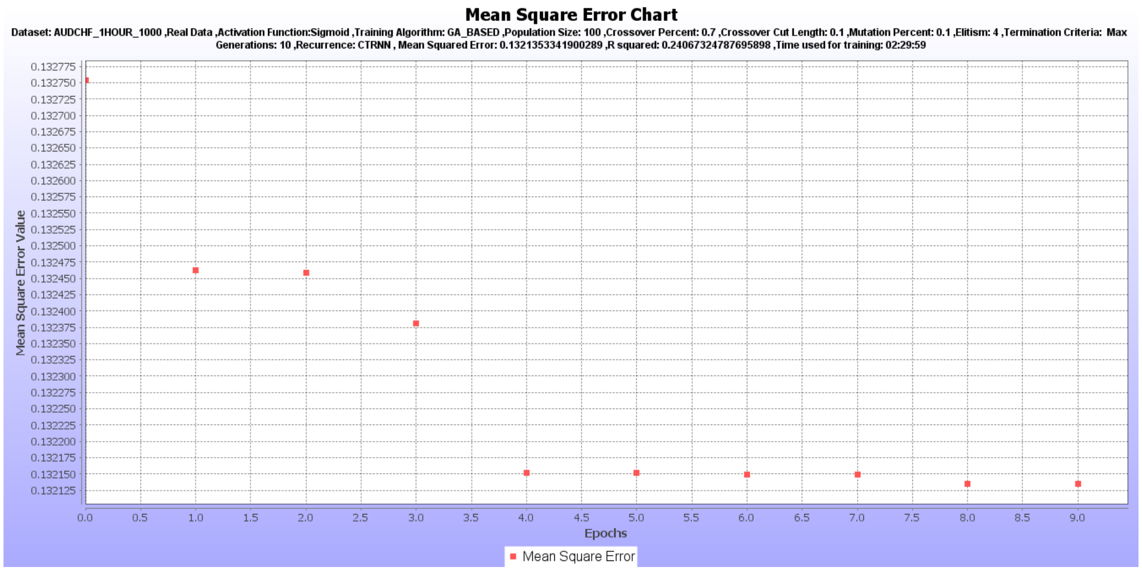
Testing dynamical systems such as a CTRNN is difficult because their state is continuously changing. That means that evaluation results will be different every time even with the same inputs.

The amount of calculations performed by CTRNNs is much larger than in case of any other model. This results in a very long training times. In order to create a population of 100 CTRNNs and evaluate their fitness around 15 minutes is needed using current testing box. the author decided to keep the training restricted to creating 10 generations.

CTRNN Training	Training Final MSE	Training R squared	Evaluation Final MSE	Evaluation R squared	Training Time Taken
Basic Model	1.32E-01	0.240673248	1.57E-04	-0.217204399	02:29:59

Figure 6.25: CTRNN Testing Results.

Figure 6.25 shows the results of testing. THe CTRNN was able to produce very good training results and good evaluation results. The evaluation and training MSE values are comparable with the best results achieved so far.



Dataset: AUDCHF_1HOUR_1000 ,Real Data ,Activation Function:Sigmoid ,Training Algorithm: GA_BASED ,Population Size: 100 ,Crossover Percent: 0.7 ,Crossover Cut Length: 0.1 ,Mutation Percent: 0.1 ,Elitism: 4 ,Termination Criteria: Max Generations: 10 ,Recurrence: CTRNN , Mean Squared Error: 0.1321353341900289 ,R squared: 0.24067324787695898 ,Time used for training: 02:29:59

Figure 6.26: CTRNN First Model MSE convergence.

Figure 6.26 shows that the first population provided a very good initial MSE value. Interestingly enough, this value is more than two times smaller/better than any initial value produced by GA training before (Figure 6.24).

Figure 6.27 shows even more interesting evaluation results. It looks like the CTRNN learned global optimums and oscillated between them. This explains the low value of R^2 which uses a mean of all samples as a benchmark model.

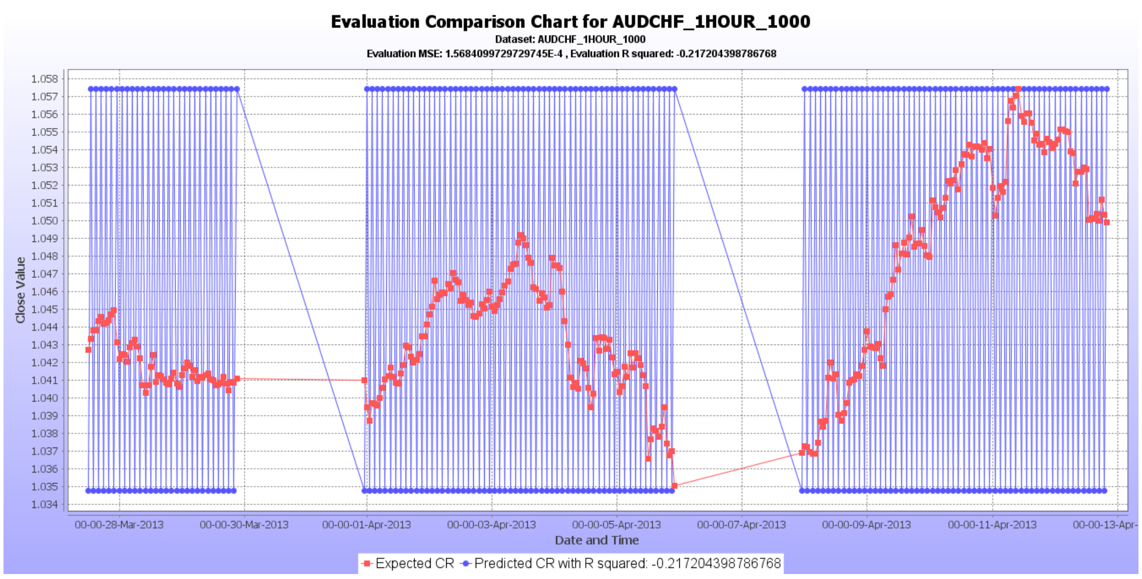
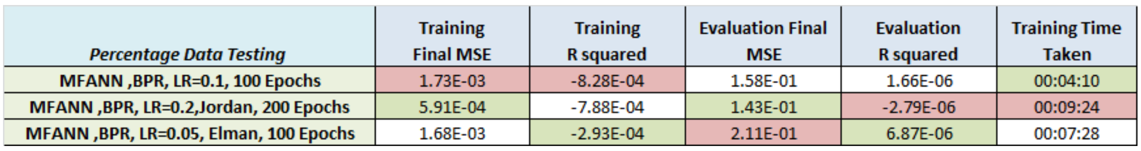


Figure 6.27: CTRNN First Model evaluation results.

The CTRNN is definitely an interesting model however the author feels that this could be a topic for a completely new project. The number of parameters and conditions that needs to be considered is enormous and requires a lot of research and domain knowledge.

6.7 Percentage Data Testing

Percentage data is created by converting every training example into a percentage change in respect to the previous value. During this test best ANN models from previous tests will be used to predict percentage data movements. Due to the fact that a percentage change can be negative it is important to remember to use hyperbolic tangent activation function.



Percentage Data Testing	Training Final MSE	Training R squared	Evaluation Final MSE	Evaluation R squared	Training Time Taken
MFANN ,BPR, LR=0.1, 100 Epochs	1.73E-03	-8.28E-04	1.58E-01	1.66E-06	00:04:10
MFANN ,BPR, LR=0.2,Jordan, 200 Epochs	5.91E-04	-7.88E-04	1.43E-01	-2.79E-06	00:09:24
MFANN ,BPR, LR=0.05, Elman, 100 Epochs	1.68E-03	-2.93E-04	2.11E-01	6.87E-06	00:07:28

Figure 6.28: Percentage Data Testing Results.

Figure 6.28 shows the results of testing. All evaluation results rae very close each other. For two models the evaluation R^2 is positive meaning that the results are better than ones produced by a mean of all samples. However the results are much worse that ones obtained using the same models with non percentage data. This confirms that ANN models are input data specific meaning that for different inputs the produced results can vary greatly.

6.8 Conclusions

During testing following phenomena were observed:

(a) When using only a single type of input, nothing provides best results than past values of output variable(Section 6.2.1).

(b) Using technical indicators as extra inputs increases ANN generalization ability(Section 6.2.2).

(c) Adding more input types does not always result in better results. In fact, it may introduce noise to the dataset and make results worse(Section 6.2.3).

(d) Too big input window results in overfitting, too small in underfitting(Section 6.2.4).

(e) Adding recurrence to the ANN can improve the quality of predictions(Section 6.4).

(f) The bigger the amount of genes, the harder it is to find optimal solution.(Section 6.5).

All ANNs created during this testing phase are saved in ”Networks” folder on the disk provided. They can be loaded and examined if needed.

In general many ANNs achieved very good results with R^2 very close to 1 during training and higher than 0 during the evaluation phase. This proves that ANNs can produce predictions better than one can achieve using a simple regression model. As a result the main objective of this project was accomplished - ANNs can forecast currency exchange market movements with a very good accuracy.

Chapter 7

Evaluation

"Experience: that most brutal of teachers. But you learn, my God do you learn."

— C. S. Lewis

7.1 Achievement of Original Goal

The project was successful in its main objective of creating ANNs that are able to forecast a currency exchange rate movements with a really impressive accuracy. All ANN morphologies and training methods that were described in the initial project's requirements(Section 1.2) were implemented. Many ANNs models achieved nearly perfect results (R^2 value was close to 1) which were much better than one could achieve using a regression model. More importantly good enough results can be obtained within a small time frame using a relatively modest computational power meaning that they are feasible for real world applications.

As per other objectives, the ANN models were able to produce good results for both problems (percentage and non-percentage) stated in Section 1.2. Initial goals of the project included using multiple currency pairs with multiple time frames in testing phase. The author did not anticipated that ANNs are so susceptible to the input data and ANN training and morphology parameters are so input data specific. In the early stages of this project, the author assumed that there are algorithms to determine the best values of these parameters. Now is clear to the author that the parameters values are problem domain specific and only trial and error approach can be used to determine good enough values. As a result during the testing phase only single currency exchange rate(AUD\CHF) and single time frame (1 Hour) was thoroughly tested to present the testing approach that can be used with other data sets. The author regrets that he did not have enough time to research and customize CTRNN which produces rather interesting testing results.

The "Neural Network Mentalist" application used for testing purposes as it stands is perfectly usable, however in order to obtain a really good results a long training time is needed. The author did not anticipate that the training time will be so long. The author regrets that he did not have more time to use a profiling tool to identify bottlenecks in the implementation. Moreover, a choice of data structures used in the implementation could be definitely much better. If some of those models would be used to help traders make better investments, a couple hours long training time would be unacceptable since the data (and patterns within the data) are changing very rapidly meaning that the ANN must be retrained very often. With such a long training time when the ANN training is over, the model is already outdated.

From the author's point of view, the main goal of design was to create a ”plug and play” architecture where addition of new training algorithms and ANN morphologies would be really simple. The author thinks that the current design achieved this goal. With hindsight it is clear that the author should pay more attention to the efficiency issues. For example, adding multithreading capabilities to the application could drastically shorten training times.

The author thinks that the choice of tools/technologies for this project was generally correct. Using SVN client instead of Dropbox would be definitely an advantage. Every commit to the SVN repository can include a short description of what was added/changed/removed which would prove very useful when working on this report. With hindsight, using MQL5 language as the main implementation language instead of Java would probably result in the overall failure of this project. Debugging was one of the most time consuming activities during this project and the only IDE that supports MQL5 currently is MetaEditor which has very limited debugging capabilities.

7.2 Development Plan Consideration

The author thinks that the Iterative Development approach was the right choice for this project. The project was divided into a number of iterations that helped to control project's progress and react accordingly. With hindsight it is clear that the required time assigned to each iteration was mostly either too generous or too limited. Adding recurrence to the project has taken much less time than initially anticipated however implementing GA based training and CTRNN took much more time than expected. As a result, the author has run out of time to properly test and customize the CTRNN model which proved to be most complex of all models. The order of the iterations had to be moved around as well in order to meet project's deadlines.For example, the GUI and statistical indicators had to be implemented much sooner than planned due to mid-project presentation.

7.3 Future Improvements

This is one those projects that can never be finished. There are plenty of other ANN morphologies and training algorithms that can be added to this project in the future. Fully recurrent NNs, Resilient Propagation training or Microbial training algorithms to name a few. Moreover already implemented models can be enhanced in many ways. For example, there exist many modifications of the backpropagation to minimize the effects of local optimum problem like e.g. selective pruning.

As mentioned before adding multithreading capabilities and using more efficient data structures could improve the efficiency of the training significantly. Using a profiler tool to identify efficiency bottlenecks would improve general application performance as well. Finally implementing automated testing that would be used to determine the optimal training parameters values and number of neurons in each layer would invaluable.

The author plans to port most successful models implemented during this project to the Metatrader5 platform and use it as a backbone of automated trading agent. They will be used alongside other technical analysis tools to determine if an autonomous trading agent can ”beat the system” and learn a profitable trading strategy.

7.4 Reflection

In the beginning I was very sceptical about the capabilities of ANNs. I regarded them as another AI algorithm that works great for simple, textbook problems, however which is not scalable or robust enough to work with the real world data. I expected results that will be a little bit better than using a model based on a variable mean calculation. To my astonishment the evaluation results of the first MFANN model were very close to the real data! I spent the next day checking the code, hunting for bugs because I thought that it is probably a mistake but it was not.

I learned a great deal thanks to this project, in fact I think that I am an ”ANN-believer” now. If I could start this project again, I would started much earlier. Some people think that studying makes your time management skills better, in my opinion it just teaches you how important deadlines are. As longest deadlines are far away, they are not important. If I would start sooner I could implement many other morphologies and training algorithms or maybe Hidden Markov Models which are praised as prediction tools.

As a many years’ chess player I know that the ability to identify patterns is invaluable and, in some sense, enables you to make accurate predictions about the future. In my opinion you can’t overestimate the capabilities of ANNs which seem unlimited. There is a lot of discussion about what intelligence is and it is my strong belief that intelligence is the ability to find patterns. The evolutionary competitive advantage in being able to see patterns that others cannot is obvious. For example, identifying patterns in seasons allowed our ancestors to start planting their own crops at the right time of year. Detecting patterns in our own bodies allowed the development of medicine. Some people define genius as the ability to identify subtle patterns that other people cannot. IQ tests, the accepted norm by which we measure intelligence test our ability to identify patterns in numbers, words and shapes. I think that given proper input data we could train an artificial neural network to excel at any IQ test ever created. If IQ tests are the measure of human intelligence and an ANN can score a high result, does it mean that it is intelligent? I leave this question for the reader.

Appendices

Appendix A

MQL5 Script

```
//+------------------------------------------------------------------+
//|                                              DownloadData.mq5 |
//|                                                 Piotr Chudzik |
//|                                             http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Piotr Chudzik"
#property link      "http://www.mql5.com"
#property version   "1.00"
//+------------------------------------------------------------------+
//| Script program start function
|
//+------------------------------------------------------------------+

// Export Indicator values for NN training by ENCOG
extern string IndExportFileName = "AUDCHF_1H_1000.csv";
extern int  trainSize = 1000;

MqlRates closePrices[];
double StochasticK[], StochasticD[], WilliamsR[], MACD[]
        , MovingAverage[], RSI[];

void OnStart()
  {
//---
   ArraySetAsSeries(closePrices, true);
   ArraySetAsSeries(StochasticK, true);
   ArraySetAsSeries(StochasticD, true);
   ArraySetAsSeries(WilliamsR, true);
   ArraySetAsSeries(MACD, true);
   ArraySetAsSeries(MovingAverage, true);
   ArraySetAsSeries(RSI, true);

   int copied = CopyRates(Symbol(), PERIOD_H1, 0,
    trainSize, closePrices);
```

```
if (copied!=trainSize) { Print("Not enough data for " + Symbol());
 return; }

int hStochastic = iStochastic(Symbol(), PERIOD_H1, 8, 5, 5,
 MODE_EMA, STO_LOWHIGH);
int hWilliamsR = iWPR(Symbol(), PERIOD_H1, 21);
int hMACD = iMACD(Symbol(), PERIOD_H1, 12,
     26,9,PRICE_CLOSE);
int hMovingAverage = iMA(Symbol(), PERIOD_H1,10,0,
     MODE_SMA,PRICE_CLOSE);
int hRSI = iRSI(Symbol(),PERIOD_H1,14,PRICE_CLOSE);


CopyBuffer(hStochastic, 0, 0, trainSize, StochasticK);
CopyBuffer(hStochastic, 1, 0, trainSize, StochasticD);
CopyBuffer(hWilliamsR, 0, 0, trainSize, WilliamsR);
CopyBuffer(hMACD, 0, 0, trainSize, MACD);
CopyBuffer(hMovingAverage, 0, 0, trainSize, MovingAverage);
CopyBuffer(hRSI, 0, 0, trainSize, RSI);

int hFile = FileOpen(IndExportFileName,
 FILE_CSV | FILE_ANSI | FILE_WRITE | FILE_REWRITE, "|",
 CP_ACP);

FileWriteString(hFile,
   "DATE-XVALUE|TIME-XVALUE|CLOSE_INPUT_OUTPUT-YVALUE|StochK-INPUT|"+
   "StochD-INPUT|WilliamsR-INPUT|MACD|MovingAverage|RSI\n");

Print("Exporting indicator data to " + IndExportFileName);

for (int i=trainSize-1; i>=0; i--)
   {
      string candleDate = TimeToString(closePrices[i].time,
       TIME_DATE);
      StringReplace(candleDate,".","");
      string candleTime = TimeToString(closePrices[i].time,
       TIME_MINUTES);
      StringReplace(candleTime,":","");
      FileWrite(hFile, candleDate, candleTime,

            DoubleToString(closePrices[i].close),
            DoubleToString(StochasticK[i], 16),
         DoubleToString(StochasticD[i], 16),
            DoubleToString(WilliamsR[i], 16),
         DoubleToString(MACD[i], 16),
         DoubleToString(MovingAverage[i], 16),
         DoubleToString(RSI[i], 16)
```

```
            );
        }

    FileClose(hFile);

    Print("Indicator data exported.");
  }
//+------------------------------------------------------------------+
```

Annotated Bibliography

[1] "ADALINE," http://en.wikipedia.org/wiki/ADALINE/, accessed April 2013.

Information about ADALINE neural network.

[2] "BabyPips," http://www.babypips.com/, accessed April 2013.

This website contains a lot of information about Forex, especially good for beginners.

[3] "ENCOG library," http://www.heatonresearch.com/encog, accessed April 2013.

This website contains information about ENCOG neural network library.

[4] "FANN library," http://leenissen.dk/fann/wp/, accessed April 2013.

This website contains information about FANN neural network library.

[5] "ForexFactory," http://www.forexfactory.com/, accessed April 2013.

This website contains a lot of information about Forex.

[6] "Hopfield Network," http://en.wikipedia.org/wiki/Hopfield_network/, accessed April 2013.

Information about Hopfield neural network.

[7] "How to Use Java BigDecimal: A Tutorial," http://www.opentaps.org/docs/index.php/How_to_Use_Java_BigDecimal:_A_Tutorial, accessed April 2013.

This website contains information java BigDecimal class.

[8] "Metatrader 5," http://www.metatrader5.com/, accessed April 2013.

This website contains information about Metatrader 5 platform.

[9] "Mig Capital," http://www.migcapital.co.uk/, accessed April 2013.

This website contains information about Mig Capital broker company.

[10] "Moving Average - MA," http://www.investopedia.com/terms/m/movingaverage.asp, accessed April 2013.

I used this website to find more information about Moving Average(MA) technical indicator.

[11] "Moving Average Convergence-Divergence (MACD)," http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:moving_average_conve, accessed April 2013.

I used this website to find more information about MACD technical indicator.

[12] "MQL Forum," http://www.mql5.com/, accessed April 2013.

This is a website of the MQL5 community where one can find all information about MQL5 and MetaTrader 5 platform.

[13] "NN Session 9: Pruning Neural Networks," http://www.heatonresearch.com/wiki/NN_Session_9:_Pruning_Neural_Networks, accessed April 2013.

I used this website to find more information about determining the correct amount of hidden neurons.

[14] "Norvig vs. Chomsky and the Fight for the Future of AI," http://www.tor.com/blogs/2011/06/norvig-vs-chomsky-and-the-fight-for-the-future-of-ai/, accessed April 2013.

This article describes the dispute between Peter Norvig and Noam Chomsky about future of AI.

[15] "Relative Strength Index (RSI)," http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:relative_strength_index_rsi, accessed April 2013.

I used this website to find more information about Relative Strength Index(RSI) technical indicator.

[16] "Stochastic Oscillator," http://www.investopedia.com/terms/s/stochasticoscillator.asp, accessed April 2013.

I used this website to find more information about Stochastic K and Stochastic D technical indicators.

[17] "TradeToWin," http://www.trade2win.com/, accessed April 2013.

This website contains a lot of information about Forex.

[18] "WEKA 3," http://www.cs.waikato.ac.nz/ml/weka/, accessed April 2013.

This website contains information about WEKA 3 platform.

[19] "Welcome To JFreeChart!" http://www.jfree.org/jfreechart/, accessed April 2013.

This website contains information about jfreechart library used for drawing charts.

[20] "William http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:williams_r, accessed April 2013.

I used this website to find more information about WilliamsR technical indicators.

[21] A. M. A. Nag, " Forecasting Daily Foreign Exchange Rates Using Genetically Optimized Neural Networks." Wiley InterScience, 2002.

This article presents an example of genetic algorithms used to train Neural Networks.

[22] D. Baily and D. Thomson, "Developing neural network applications," in *AI Expert*, 1990, pp. 33–41.

This article discusses the issue of how many hidden neurons should be in a hidden layer.

[23] F. Castiglione, " Forecasting Price Increments Using an Artificial Neural Network." Hermes-Oxford, 2001.

This article presents examples of using Multilayer Perceptrons for forecasting financial market moevements.

[24] P. V. Cauwenberghe, "Iterative development with incremental delivery," http://www.methodsandtools.com/archive/archive.php?id=14/, accessed April 2013.

This is a website describing Iterative Development with Incremental Delivery.

[25] O. Ersoy, "Tutorial at Hawaii International Conference on Systems Sciences," 1990.

This article discusses the issue of how many hidden neurons should be in a hidden layer.

[26] E. F. Fama, "The Behavior of Stock-Market Prices," *The Journal of Business*, vol. 38, no. 1, pp. 34–105, 1965.

This article describes Efficient Market Hypothesis.

[27] K. Fukushima, "Self-organization of a neural network which gives position-invariant response," in *Proceedings of the 6th international joint conference on Artificial intelligence - Volume 1*, ser. IJCAI'79. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1979, pp. 291–293.

This article describes a neural newtork with multiple hidden layers.

[28] C. Giles, S. Lawrence, and A.-C. Tsoi, "Rule inference for financial prediction using recurrent neural networks," in *Computational Intelligence for Financial Engineering (CIFEr), 1997., Proceedings of the IEEE/IAFE 1997*, 1997, pp. 253–259.

This article shows that recurrent artificial neural networks can produce results better than a method based on a random walk hypothesis.

[29] E. R. Harold, "Java's Primitive Data Types," http://www.cafeaulait.org/course/week2/02.html, accessed April 2013.

This website contains information java primitive types.

[30] D. O. Hebb, *The Organization of Behavior*. New York: Wiley, 1949.

This book describes Hebb's Rule at page 62.

[31] T. Hellstrm and K. Holmstrm, "Predicting the stock market," Department of Mathematics and Physics, Malardalen University, S-721 23 Vasteras, Sweden, Tech. Rep. IMa-TOM-1997-07, 1998.

This report describes properties of financial time series and data types used for forecasting.

[32] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, July 1989.

This article proves that artificial neural networks with a single hidden layer can approximate any function that contains a continuous mapping from one finite space to another.

[33] K. ichi Funahashi and Y. Nakamura, "Approximation of dynamical systems by continuous time recurrent neural networks," *Neural Networks*, vol. 6, no. 6, pp. 801–806, 1993.

This article provides a description of CTRNN and shows that they can approximate any dynamical system.

[34] V. C. J. Riley, " An Evolutionary Approach to Training Feed-Forward and Recurrent Neural Networks." Proceedings Of The Second International Conference On Knowledge Based Intelligent Electronic Systems, 1998.

This article presents an example of genetic algorithms to train Neural Networks.

[35] C. L. T. Jingtao Yao, “ A case study on using neural networks to perform technical forecasting of forex,” in *Neurocomputing 34*. Elsevier, 2000, pp. 79–98.

This article presents examples of using Multilayer Perceptrons for forecasting financial market moevements.

[36] T. Jones, *Artificial Intelligence: A Systems Approach*, ser. Computer Science. Jones & Bartlett Learning, 2008. [Online]. Available: http://books.google.co.uk/books?id=ekUHwvRP7nUC 9780763773373.

I used this book as a reference book for the backpropagation training algorithm.

[37] I. Kaastra and M. S. Boyd, “Designing a neural network for forecasting financial and economic time series,” *Neurocomputing*, vol. 10, no. 3, pp. 215–236, 1996.

This article provides general tips about how to design an artificial neural for forecasting of economic time series , e.g. how to pre-process input data.

[38] J. Katz, “Developing neural network forecasters for trading,” in *Technical Analysis of Stocks and Commodities*, 1992, pp. 58–70.

This article discusses the issue of how many hidden neurons should be in a hidden layer.

[39] C. Klimasauskas, “Applying Neural Networks,” in *Neural Networks in Finance and Investing: Using Artificial Intelligence to Improve Real World Performance*, 1993, pp. 64–65.

This article discusses the issue of how many hidden neurons should be in a hidden layer.

[40] R. Lawrence, “ Using Neural Networks to Forecast Stock Market Prices.” University of Manitoba, 1997.

This article presents an ovierview of a number of neural network morphologies used for forecasting market movements.

[41] T. Masters, *Practical Neural Network Recipes in C++*. Academic Press, Inc., 1993, 0-12-479040-2.

I read chapters 1-7 and 10-11 of this book to familiarize myself with neural networks concepts and time-series prediction.

[42] P. McCluskey, “ Feedforward and Recurrent Neural Networks and Genetic Programs for Stock Market and Time Series Forecasting.” Brown University, 1993.

This article presents a number of neural network morphologies and training algorithms used for forecasting financial market moevements.. It concludes that recurrent neural networks are much more effective than feedforward neural networks.

[43] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity. 1943.” *Bulletin of Mathematical Biology*, vol. 52, no. 1-2, pp. 99–115; discussion 73–97, 1990. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/2185863

This article presents the very first model of an artificial neural network.

[44] P. McNelis, *Neural Networks in Finance: Gaining Predictive Edge in the Market*. Elsevier, 2005, 0-12-485967-4.

I read chapters 1-2 of this book to familiarize myself with neural networks and their applicability to finance problems.

[45] M. L. Minsky and S. Papert, *Perceptrons: An introduction to computational geometry*. MIT press Cambridge, Mass, 1988.

This book discusses limitations of a perceptron neural model.

[46] T. Mitchell, *Machine Learning*. McGraw-Hill, 1997, 0-07-115467-1.

I read chapter 4 of this book to familiarize myself with neural networks in general and the backpropagation algorithm. I cited passages from this book (pages 81-82) to provide a basic definition of neural networks as well.

[47] M. Moreira and E. Fiesler, "Neural Networks with Adaptive Learning Rate and Momentum Terms," IDIAP, Martigny, Switzerland, Idiap-RR Idiap-RR-04-1995, 1995.

This report provides a good summary of algorithms used to optimize backpropagation's learning rate.

[48] N. Nikolaev, "Practical Aspects of Backpropagation," http://homepages.gold.ac.uk/nikolaev/311practbp.htm, accessed April 2013.

I used this website to find more information about backpropagation, especially momentum parameter.

[49] G. D. P. Tino, Ch. Schittenkopt, " Financial Volatility Trading using Recurrent Neural Networks." IEEE Press Piscataway, 2001.

This article presents an example of using Elman Recurrent Neural Network for forecasting financial market moevements.

[50] K. Priddy and P. Keller, *Artificial neural networks*, ser. Tutorial TExts in Optical Engineering. SPIE Press, 2005. [Online]. Available: http://books.google.co.uk/books?id=BrnHR7esWmkC 9780819459879.

This book provides general description of ANNs. Normalization used in this project was based on chapter 3 of this book.

[51] F. Rosenblatt, *Principles of neurodynamics: perceptions and the theory of brain mechanisms*. Washington, DC: Spartan, 1962.

This book discusses a perceptron neural model - first neural network to use weighted inputs.

[52] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Mit Press Computational Models Of Cognition And Perception Series*, pp. 318–362, 1986.

This article describes first multilayer perceptron ever trained by backpropagation.

[53] R. Schwaerzel and B. Rosen, "Improving the accuracy of financial time series prediction using ensemble networks and high order statistics," in *Neural Networks,1997., International Conference on*, vol. 4, 1997, pp. 2045–2050 vol.4.

This article proves that random walk hyphothesis is not always valid.

[54] R. Trippi and E. Turban, *Neural networks in finance and investing: using artificial intelligence to improve real-world performance*. Irwin Professional Pub., 1996, 9781557389190.

This book contains a number of examples of how ANNs are used in a financial world.

[55] E. Tuci, I. Harvey, and M. Quinn, "Evolving Integrated Controllers for Autonomous Learning Robots Using Dynamic Neural Networks," in *In*. MIT press, 2002, pp. 4–9.

This article provides a good introduction to the world of CTRNNs.

[56] B. R. Valluru, *C++ Neural Networks and Fuzzy Logic*. IDG Books Worldwide, 1995, 1-55-851552-6.

I read chapters 1-7 of this book to familiarize myself with neural networks concepts and backpropagation training algorithm.

[57] Y. A. K. V.V. Kondratenko, " Using Recurrent Neural Networks To Forecasting of Forex." St. Petersburg State University, 2003.

This article presents an example of using Elman Recurrent Network and statistical tools for measuring network performance.

[58] P. J. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*, ser. Adaptive and Learning Systems for Signal Processing, Communications and Control Series. Wiley-Interscience, 1994, 0471598976.

This books describes backpropagation algorithm used to train artificial neural networks.

[59] I. S. Y. Kuperin, L. Dimitrieva, " Neural Networks in financial markets dynamical studies: Working paper series of the Center for Management and Institutional Studies." St. Petersburg State University, 2001.

This article presents an example of using Recurrent Neural Network for forecasting financial market moevements.