NTNU
Det skapende universitet

Trading "in-play" betting Exchange Markets with Artificial Neural Networks

Øyvind Norstein Øvregård

Master i datateknikk
Oppgaven levert: Juni 2008
Hovedveileder: Keith Downing, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Oppgavetekst

Betting exchanges such as Betfair can be seen as bearing a strong similarity to that of the stock exchange. The main difference is that commodity being traded is a bet rather than a stock, or a futures contract. Technical analysis is the study of market action, for the purpose of forecasting future price movements.

Betfair offers "in play" tennis markets. This means that traders are allowed to trade while the tennis match is being played, with the odds fluctuating accordingly to the events of a match. E.g. if a player suddenly gets a break opportunity, the odds of this player winning the match will decrease. The main characteristic of in-play tennis odds markets is its volatility.

This project investigates the potential profitability of Artificial Neural Networks for technical analysis of "in play" tennis markets on Betfair. Several types of neural networks are trained and tested on historical "in play" tennis markets. The various Neural Networks is specifically developed to generate trading signals rather than make price predictions. A custom designed cost function are identified to suit the underlying problem, and are implemented directly into the training process. Results of training neural networks with the custom cost function are compared to training with standard cost functions in respect to training error and achieved profit.

Oppgaven gitt: 21. januar 2008
Hovedveileder: Keith Downing, IDI

Abstract

Betting exchanges such as Betfair can be seen as bearing a strong similarity to that of the stock exchange. The main difference is that commodity being traded is a bet rather than a stock, or a futures contract. Technical analysis is the study of market action, for the purpose of forecasting future price movements.

Betfair offers "in play" tennis markets. This means that traders are allowed to trade while the tennis match is being played, with the odds fluctuating accordingly to the events of a match. E.g. if a player suddenly gets a break opportunity, the odds of this player winning the match will decrease. The main characteristic of in-play tennis odds markets is its volatility.

This project investigates the potential profitability of Artificial Neural Networks for technical analysis of "in play" tennis markets on Betfair. Several types of neural networks are trained and tested on historical "in play" tennis markets. The various Neural Networks is specifically developed to generate trading signals rather than make price predictions. A custom designed cost function are identified to suit the underlying problem, and are implemented directly into the training process. Results of training neural networks with the custom cost function are compared to training with standard cost functions in respect to training error and achieved profit.

Preface

This is a master thesis from the Department of Computer and Information Science at the Norwegian University of Science and Technology. The work was conducted by Øyvind Norstein Øvregård and lasted from January to June 2008. The thesis is based on the results and findings of "Predicting Betting Exchange Markets using Neural Networks and Genetic Programming" [17].

The field of Artificial Intelligence, and Artificial Neural Networks in particular, has always been very intriguing to me. In addition, the field of odds trading and tennis in general are interests of mine, and the idea for a problem definition came as a consequence of this. Although the work has had periods of intensive research, long hours of coding and debugging, and periods of intensive writing, the work on this thesis has been very rewarding.

I would like to thank my teaching supervisor Keith Downing for guidance and sharing of expertise during this project. I would also like to thank my former partner from the project which made this thesis possible, Henrik Vatnar. He has helped me a lot, not only by assisting in the implementation process, but also as a good friend and work place companion.

Øyvind Norstein Øvregård

Contents

Chapter 1

Introduction

1.1 General

This master thesis is a continuation of the work and research that was conducted in the Intelligent Systems Specialization Project, autumn 2007 [17]. For a general introduction to Artificial Neural Networks (ANNs), Betfair, the Betfair API, Tennis match-odds markets and trading, reference to this project is made.

1.2 Problem Description

Since the introduction of betting exchanges, and particularly Betfair [17] in 2000, new and previously undiscovered opportunities have presented itself to punters and small time traders. But the betting exchanges has also drawn a new kind of crowd; the full-time, high-volume traders who buy and sell odds on the exchanges in the same way as financial traders buy and sell stocks or other commodities in the financial markets. In fact, there are many former financial traders operating on Betfair markets and many of these professional traders uses trading techniques and strategies that originate from financial trading.

Investing, or trading in the stock markets, is a form of gambling. E.g. when buying stocks, you gamble that the price will increase in the future - so that the stocks can be sold with profits. However, trading differs from pure gambling in many ways. When trading, although the gamble is that the price should move in a certain direction, the risks are substantially lower. Gambling means to bet on a certain outcome; if the outcome does not occur, the whole stake is lost. With trading, if the price goes against your position, the possibility of cutting losses and exiting the market is present. The whole "stake" is normally not lost. Financial investors and traders do not gamble - they take "calculated risks". When gambling, the odds are normally stacked against you. When trading in odds, punters have the opportunity to both "buy" and "sell" odds. This is

what determines the price, there is no bookmaker involved. With the opportunity for everybody to play both sides of the spread, the prices represent a more fair market.

Besides being a fascinating sport, tennis or tennis odds markets are suitable for ANN training for a number of reasons. Many of these were mentioned in "Predicting Betting Exchange Markets using Neural Networks and Genetic Programming" [17], but the most important are repeated here. Tennis are among the most traded sports on Betfair, and with over 69 ATP tournaments each year there is an abundance of available tennis markets to trade. Tennis is among the markets with highest liquidity on Betfair. The outcome of a tennis match is binary, and with stats on every match since 1991 freely available to the general public through the ATPtennis.com website, relevant ANN training data is accessible. The fact that this kind of data, or any other kind of data [1], can directly be linked up to the performance (or anticipated performance) of each of the tennis players, makes the modeling task easier than in e.g. soccer or any other team sport. In soccer, any relevant statistics regarding how the team is performing are composed of stats on how the individuals comprising the team are performing. This makes the matter harder, and it is further complicated by the fact that terms as e.g. "team spirit" and "team-chemistry" not easily can be quantified.

In-play, or live betting/trading is wagering on an event as it happens. On Betfair, once an event starts, the market is turned "in-play". In-play markets differ much from traditional markets, or "pre-match" markets with the main difference being the level of risk involved. In-play markets fluctuate at a much higher rate and with a greater impact than the more stable pre-match markets as it closely reflects the unfolding actions of the match. Section 2.2 describes this in more detail. Many traders are specializing in in-play trading, with their advantage being having faster streams of information from the event being traded than the general public. In fact, at tennis matches, people have been spotted sitting court-side with a laptop trading/gambling on the match in front of them. Traders doing this, obviously have a clear advantage over e.g. traders watching the match "live" on television. Any event broadcasted "live" on TV usually is 3 to 10 seconds (or even more) delayed. However, people trading tennis matches court side has been known to be banished by ATP officials, following the Davydenko scandal [20][2] and the suspicion of illegal gambling activities on tennis matches [12]. Although of little relevance to this project, it is mentioned as an example of how people proceed to gain an advantage, or an "edge" over other traders.

In order to trade any market profitably in the long run, such an advantage over the rest of the market must be present. This project seeks to produce such an advantage, by using artificial neural networks. Based on data from the past markets, is it possible to predict future odds movements during live tennis matches? Is the potential profitability of such a net enhanced by using well known, basic trading strategies from financial

[1] Experiments in [17] showed that fundamental data was of little importance to the prediction task. This project mainly focuses on technical data.

trading? And by incorporating these strategies into the actual training of neural nets?

1.3 Previous work

There has been several experiments using ANNs to predict the winner in various sporting events, e.g. greyhound racing [4], rugby [14] [1] or basketball [1]. However, few have made attempts using ANNs to predict future odds fluctuations on betting exchanges. The predecessor to this project [17] made an attempt to predict future odds movement in pre-match tennis odds markets on Betfair. "In play" or "live" markets differ much from the more static pre-match markets on betting exchanges. Any information/documentation on prediction by the use of ANNs of these highly volatile in play markets are not currently available/non-existent.

There are vast similarities between financial markets and betting odds markets. Consequently, previous research on predicting financial markets by the use of ANNs is used as an underlying foundation for this project. Pissarenko [18] give an overview and serves as an introduction to financial time series prediction using ANNs.

1.4 Motivation

Artificial Neural Networks learning methods provide a robust approach to approximation of functions. Financial time series prediction, or stock prediction is popular applications of artificial neural networks. Betting exchange markets share many of the properties of financial markets, but few efforts has been made to explore the profitability of using neural nets for prediction of odds fluctuation on betting exchanges. In "Predicting Betting Exchange Markets using Neural Networks and Genetic Programming" [17] experiments were conducted on pre-match odds markets. Although the results held some promise, the testing was on an initial stage and needs to be further investigated. One of the observations in [17] was the fact that pre-match odds markets not necessarily fluctuate to the extent where it becomes of interest for the ANN prediction task. Consequently, the focus shifts slightly in this work, when the attention is moved from pre-match markets to in-play markets. The motivation for this master thesis is to see if it is possible to use artificial neural networks, trained with problem specific cost functions, and in combination with well known trading strategies and methods, to profitably trade the in-play tennis odds markets on Betfair.

1.5 Goals

The thesis has three focus areas which combined forms the overall goal of the entire thesis.

Time Series Prediction with Artificial Neural Networks is a popular area of application for this learning method. Predicting betting exchange time-series,

in the specific case of in-play tennis match odds markets, is the overall goal of this thesis.

Various Neural Network training algorithms, structures and data representation are experimental factors that must be identified in order to find the optimal solution of any underlying problem. Finding an optimal composition regarding these factors for the problem of in-play tennis odds trading is a goal of this thesis.

Problem Specific Training Cost Functions are designed to "fit" the underlying problem structure, rather than using quadratic and least square predictors, thereby implying a symmetric and quadratic cost relationship. Identifying a suitable cost function for the problem domain of trading, and using this for ANN training is a goal of this thesis.

The goal of this thesis is to explore how different composed artificial neural networks, can be used to predict in-play tennis odds market time series on Betfair. The thesis seek to identify a relevant problem specific cost function which can be used for ANN training, and analyze how implementing the problem specific properties directly into the training process compares to standard statistical cost functions with regard to training error and potential trading profitability.

1.6 Organization

Chapter 1 - Introduction

Introduces the problem and the motivation behind it. Gives an overview over relevant work and states the goal of the thesis.

Chapter 2 - Background

Explains the general concept of odds trading, including the characteristics of in-play markets and some of the different ways to define sporting odds. While the general introduction to artificial neural nets was given in [17], this chapter introduces the background ANN theory relevant to this project.

Chapter 3 - Problem

Describes some of the challenges related to the underlying prediction problem and introduces ways to handle these. Identifies properties that should be implemented in the training and testing.

Chapter 4 - Methods and Implementations

Presents the chosen implementation and the methods used. Introduces the custom designed cost function used for part of the training.

Chapter 5 - ANN Testing and Results

Presents the different Tests and the results of these. The results are only presented; evaluation and discussion is left to Chapter 6.

Chapter 6 - Discussion

Evaluation of the results achieved from the testing. Explains how the tests are evaluated, and discusses all the tests of Chapter 5 in detail.

Chapter 7 - Conclusion and Future Work

Concludes the main achievements of the thesis, and suggests issues that could be subject for future work.

Chapter 2

Background

This chapter gives a general introduction to some of the relevant concepts of this thesis.

2.1 Tennis match odds trading

Section 2.1 also appeared in [17], but is included here for the sake of readability of this report.

On betting exchanges, traders can fill the role of both the traditional punter and bookmaker. That is, traders can both offer odds to other traders in the same way as bookmakers, and take odds offered by other traders in the same way as punters. The former is known in betting exchange terminology as "Laying" and the latter is known as "Backing".

The trader is not necessarily interested in the outcome of the particular event. The trader is interested in the movement, or fluctuation in the odds. Similar to stock market trading, the trader seeks to buy (lay) when the odds is low, and sell (back) when the price is higher, or the two in reversed orders. In this way, trading differs from ordinary betting.

Figure 2.1 shows a typical tennis match odds market on Betfair. The match is Janko Tipsarevic vs. Rafael Nadal, with current odds levels around 4.1 for Tipsarevic and 1.31 for Nadal on the backing side. The corresponding book % is 100.7 %. The graph shows the development of price and volume over time. To illustrate how trading on Betfair works, consider the following example from Figure 2.1. Shortly after the match odds market opened, odds of 1.15 were available on the lay side on Rafael Nadal. Speculating in these odds, a trader predicts that the odds would eventually stabilize at a higher level. The trader then lays 1.15 for, say, 2000 NOK. At this stage, this would involve a risk of losing (2000*1.15)-2000 = 300 NOK if Nadal went and lost the match. However, not long after the trader took this opening position in the market, the odds increased to a level of around 1.3. The trader is then successful in his prediction, and now chooses to exit the market for a guaranteed profit before the actual tennis match has even begun.

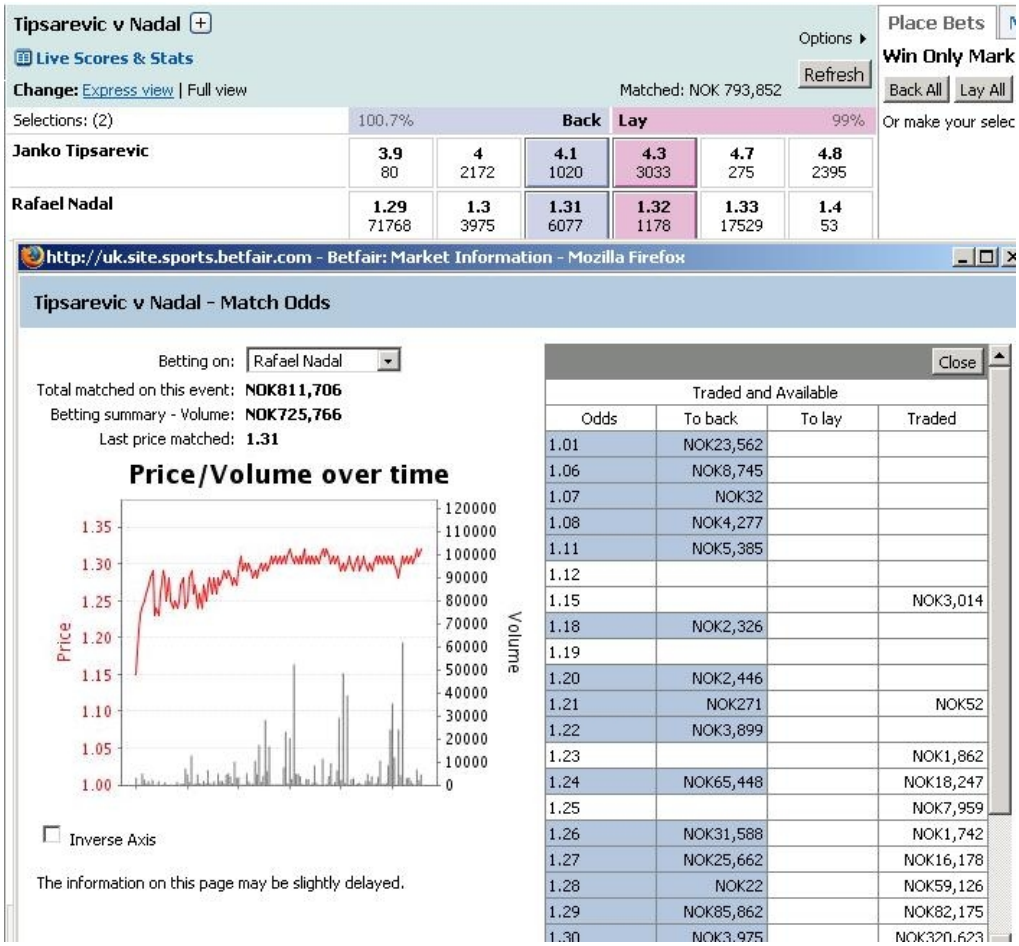


Figure 2.1: Example of Price/Volume over time

There are several ways of distributing the profit when making the exit trade. In this example, if the trader would like to get the same amount of profit no matter what the outcome of the game he can now back Rafael Nadal with a stake of 1769.23 NOK. The guaranteed profit would then be distributed over the two outcomes of the game with ((1769.23*1.3) NOK - 300 NOK) = 230.77 NOK if Nadal wins the game, and with (2000 NOK - 1769.23) = 230.77 NOK if Tipsarevic wins the game. This is just an example, there are many ways to distribute trading profits, and the same goes for trading losses. Another example may be if the trader fancy Tipsarevic to win the game, he/she can distribute the profit so that he/she gets a "risk free bet" on this. The profit would in this case be (2000 NOK - 1000 NOK) = 1000 NOK if the bet went in, and (300 NOK - 300 NOK) = 0 NOK (hence risk free) if the bet is lost.

2.2 In-play odds markets

In-play betting, in-running betting or live betting [1] is to wager on an event as it happens, with the prices being updated constantly. The prices move accordingly to what takes place in the event. Bookmakers who offer in-play betting, constantly have to compile new odds as the event unfolds. A result of this is that the price offered to the customers is often lower than the appurtenant probability. On the betting exchanges however, the odds are being offered by other traders, and is thus much closer to the semi strong form of an efficient market [17].

Figure 2.2: Example of an in-play match odds graph. The match is between Paul Henri Mathieu and James Blake

Figure 2.2 shows an example of the price development in a tennis match between Paul Henri Mathieu and James Blake in New Haven, August 2007. The graph shows how the price of James Blake evolves throughout the game. Blake won the match 6-4 3-6 7-6(2). In the beginning (A) of the match, the graph does not move considerably and the price is stable of around 1.58. This is the pre-match section of the graph. Once the match is under way (B), you can see the price fluctuate as the game swing back and forth. Next, Blake gets the upper hand and wins the first set 6-4 (C). The price falls and stabilizes on a level about 1.2. Mathieu comes back and wins the second set 6-3, and the price rises back to just under the pre-match level (D). The third set is very tight, with Blake narrowly being on top. In the eight game of the third set, Blake broke his opponent to go up 5-3. The price came crashing down to 1.09. Blake now served for the match and had two match points as he went up 40-15 in the ninth game. The price came down to 1.02, indicating a 98% probability that Blake would win the match (E). However, Mathieu saved the two match points and broke his opponent back. The price increased back to previous levels as the match went into a tie-break (E). In the end, Blake then

[1] Bookmakers use all three terms, but as Betfair uses the term "in-play", so will this report.

won the tie-break and the match, as can be seen by the price graph.

The example above illustrates how in-play tennis odds markets fluctuate on Betfair. Volatility is often viewed as a negative in that it represents uncertainty and risk. The Betfair in-play markets are highly volatile and there is a lot more risk involved in trading markets like this, compared to the more stable pre-match markets. However, the conclusions of [17] showed that if you concentrate on short periods of time in the pre-match markets, most of the time the price will not move at all, thus making it uninteresting for any prediction method. If you compare the tennis match odds markets to financial markets, match odds can be described as short term markets. Long term odds markets on Betfair are the ones that exists over several months e.g. Premiership - Winner 2007/08. The pre-match tennis markets are not volatile, but at the same time short term - short term trading strategies fail because of the minimal price movements.

In-play tennis odds markets have some attractive characteristics. Anyone that has seen a tennis match knows that the momentum in the game often swings back and forth between the two players, constituting very volatile markets. Volatile markets, although seen as risky in the financial world, can have advantages in the odds trading world. Tennis has some properties which make them more suitable for trading than many other sports. To explain, consider the example of an in-play soccer match. When a goal is scored, the odds move in an "irreversible" step-wise fashion. Figure 2.3 shows the in-play odds of an international friendly match between Poland and Denmark. The figure shows how the odds of Poland to win the match develop over time. Some time into the game, suddenly the odds increases dramatically, clearly this is the result of Denmark scoring a goal. The impact of goals in soccer matches is so large, that it almost reduces the trading on it to pure betting. If you were to have the market position of backing Poland, before Denmark scored, after the goal you have two options. You can either get out of the trade, i.e. lay Poland, to lose approximately 60 % of the initial stake. Or you can hope that Poland equalize. Looking at the Figure 2.3, they did, and so the odds stabilizes at approximately the same level as before the first goal. Goals have a profound effect on the market, sitting "hoping" for them is betting, not trading. When goals occur, the market changes dramatically, in periods without goals the markets are very stable. Tennis markets do not behave in a similar fashion. The odds of a tennis match changes on a point-to-point basis, meaning that the odds fluctuates all the time. The odds are not only volatile inside each game, but also has the larger fluctuations within each set, meaning that there are more trading possibilities, suitable to different trading strategies.

2.3 Price Format

Odds can be expressed or presented in different ways. The most common of these include fractional odds, moneyline odds and decimal odds.

Figure 2.3: Example of price - time graph from an in-play soccer match between Poland and Denmark

2.3.1 Fractional Odds

Fractional odds are favored by the bookmakers in the United Kingdom. A fractional odds of 3 to 1 (written as 3-1, 3:1, or 3/1) means that if you bet 1 unit at these odds and the event occurred, you would get 3 units in addition to getting the 1 unit initially staked back. In the same way, a winning bet at the odds of 1:3 would win 1 unit in addition to getting your 3 unit stake back.

2.3.2 Moneyline Odds

Moneyline odds are favored by American bookmakers (thus often called "American odds"). With moneyline odds there are two possibilities, the figured quote can be either positive or negative. The moneyline format of the example above (fractional odds of 3:1) would be quoted as 300$. The quote shows how much money can be won on a 100$ wager. The quote is positive because the odds are better than evens. If the odds are below evens, the quote changes become negative. It now shows how much money must be wagered to win 100%. An example of this is the fractional odds of 1:3 which in moneyline odds would be quoted as -300$.

2.3.3 Decimal Odds

Decimal odds are favored in continental Europe, and are the easiest to work with for trading. Betfair uses decimal odds. The decimal odds are the inverse of the probability of the event, and thus often easiest for people to understand in probabilistic terms. An example of decimal odds and the corresponding probability is given in Figure 2.4. The decimal odds of the above examples are 4. Compared to the fractional odds, this is the fractional odds divided (3/1) plus 1.

Figure 2.4: The odds and the relating implied chance of the event happening.

2.3.4 Binary Prices

If you ask a person on the street what they think the chances are of a certain event happening, it is not likely they are to say "It is a 1.4 shot" or "I recon it's around 2/5". Most people would say "I think it has around 70% chance of happening". Thinking in terms of probabilities is the most natural and intuitive way of expressing odds. The "binary price" is a value between 0 and 100 which is simply the percentage chance of the event occurring. As the event becomes more likely the binary price will move towards 100; as it becomes less likely the price moves towards 0. The bet will "expire" at a level of 100 if the event occurred; or at a level of 0 if it did not occur. Binary prices suits "binary markets" such as tennis match odds markets, and are explained in Section 3.2.

2.4 Artificial Neural Networks

For a general introduction to Artificial Neural Networks reference to [17] is made. In this section, background material of subjects relevant to this project is introduced.

2.4.1 Time Series Processing with Neural Networks

Neural networks, being developed primarily for the purpose of pattern recognition (classification), are not well suited for modeling time series because the original applications of neural networks were concerned with detection of patterns in arrays of measurements which do not change inn time [6]. Neural networks used for time series prediction must possess memory in one way or the other due to the dynamic nature of spatio-temporal data as time series are. Sliding windows are a technique that, together with the Multi-Layer Perceptron (MLP) [18], can be used to supply the neural network with memory, and are introduced in the following section. Other approaches to time-series processing includes the recurrent neural networks such as Jordan networks [3], Elman networks [18] and Hopfield networks [3].

Sliding Window

The challenge for neural networks in a time-series environment is the representation of temporal organization inherent in the input data. With the sliding window technique, the networks are not trained on the whole data set, but rather a small subset of the data corresponding to a certain period in time before the point to be predicted. Then, the window is shifted one step forward in time, and the network is trained on the subsequent data. This process is repeated through all the training data.

2.4.2 Cost Functions in Feed-Forward Neural Networks

Supervised learning is a technique for learning a function given a set of example pairs of input and output. Feed-forward multilayered neural networks are trained with the class of algorithms belonging to supervised learning, with the most common being the back-propagation algorithm [19]. The general idea behind this class of algorithms is the minimization of an error function by iteratively updating the weights of the network. This is done by using the derivatives of the error, using gradient descent on a single pattern error. The error e_j (j-th neuron) of units in hidden layers is computed using the error of following layers, thus propagating the errors backwards through the network unto the input layer. The mean square error (MSE) is the most commonly used cost function. However, it has been suggested that it is not the best suited function for all function approximation problems [13] [21]. In financial applications of neural networks, the MSE may have nothing to do with the system objectives. Consider the example of stock price prediction, where a neural net trained with the MSE error function, can correctly predict most of the many small price changes, but fail to predict the few big ones. In a profitable system, it is more desirable to do the opposite, predict the few big

price changes and fail to predict the many small changes. In the latter case, the MSE of the net would be larger than the in the first case, but would still generate more profit. It would be better to train a neural network for this application directly on the profit function, instead of the MSE [10]. Below follows a brief introduction to some of the cost functions commonly used in neural networks.

Mean Square Error (MSE)

As mentioned above, the MSE is the most common cost function and is considered to be the standard. It is used extensively in statistics and is defined as:

$$e = (d - y)^2 \tag{2.1}$$

with d the estimator and y being the estimated parameter. The derivative is

$$\frac{\partial e}{\partial y} = d - y \tag{2.2}$$

Some of the properties which make the MSE attractive are that its derivative is continuous and analogous to the sign and magnitude of the sign and magnitude that the algorithm tries to minimize. As a result of the squaring, it gives more emphasis to reducing the larger errors as compared to the smaller errors. One of the negatives of this cost function is that it sums all the errors for all input patterns, meaning that if a training pattern is not well represented and happens to have small errors, it may be ignored by the learning algorithm.

Mean Absolute Error (MAE)

The Mean Absolute Error (MAE) cost function is similar to the MSE, but does not give emphasis to larger errors:

$$e = \sqrt{(d - y)^2} \tag{2.3}$$

where the derivative is

$$\frac{\partial e}{\partial y} = \frac{d - y}{\sqrt{(d - y)^2}} \tag{2.4}$$

One of the drawbacks with MAE is that the derivative is not analogous to the degree of error, hence making it difficult to minimize.

Cross Entropy

In many classification problems, the cross entropy (maximum likelihood) function has been reported as more appropriate than e.g. the MSE function [21]. It is defined as follows:

$$e = md * log(\frac{md}{my}) + (1 - md) * log(\frac{1 - md}{1 - my}) \tag{2.5}$$

where the factors my and md is defined as in equations 2.6 and 2.7

$$md = \frac{d+1}{2} \tag{2.6}$$

$$my = \frac{y+1}{2} \tag{2.7}$$

As the expected and actual neuron outputs must be transformed from [-1:1] to [0:1] to consider them as probabilities[2]. The derivative of this cross entropy function is:

$$\frac{\partial e}{\partial y} = 0.5 * (\frac{1+d}{1+y} - \frac{1-md}{1-my}) \tag{2.8}$$

Cross entropy tends to allow errors to change weights even when nodes saturate, as their derivatives are asymptotically close to 0.

Asymmetric Cost Functions

Asymmetric cost functions have the property of taking the sign of the error into account when calculating the error. Generally, the cost will generally increase with the numerical magnitude of the error, but by a factor that is different for negative and positive errors. Consider a general asymmetric linear cost function of the form:

$$e(d,y) = \begin{cases} a|d-y| & \text{for d>y} \\ 0 & \text{for d = y} \\ b|d-y| & \text{for d<y} \end{cases}$$

where a and b are scalars selected to fit the underlying problem. For a $\neq$ b this cost function is non-symmetric about 0. The advantage with cost functions of this type is that it more correctly can evaluate problems where the real world cost is non-symmetric. As the gradient descent algorithms used for training uses the error's derivatives, the cost functions needs to be fully differentiable to allow calculations of these.

The function determining the size of the error e should reflect the significance of the difference in desired and actual output depending on the underlying learning problem. Traditional neural network theory focuses on quadratic error functions, thus implying a quadratic cost relationship [5]. However, the real world underlying problem may not always have this property.

2.4.3 Cascade-Correlation Architecture

The Cascade-correlation architecture is a supervised learning algorithm for artificial neural networks. Instead of adjusting the weights in a network of fixed topology, Cascade-correlation begins with a minimal network, then automatically trains and adds new hidden units one by one, creating a multi-layer structure while it trains the network [7].

[2]This transformation is not needed when using asymmetric activation functions in the output layer

The number of candidate units is trained separately from the main network, and the unit most "fit" is inserted into the neural network. This units' input-side weights are then frozen, and the unit becomes a permanent feature-detector in the network.

The idea behind the Cascade-correlation learning algorithm was to overcome the limitations of the backprop algorithm. These limitations mainly includes the slow pace at which backprop learns from examples, and Fahlman and Lebiere [7] identifies the "the step-size-problem" and "the moving target problem" as the major contributors to the slowness.

The Step-Size Problem

The standard backprop algorithm only computes $\partial E/ \partial w$, the partial first derivative of the overall error function with respect to each weight in the network. With these derivatives, a gradient descent in weight space can be performed. By taking infinitesimal steps down the gradient vector, running new training epochs will re-compute the gradient, and reduce the error with each step. However for fast learning, we do not want to take infinitesimal steps, we want to take the largest steps possible. The problem with too big steps on the other hand, is that the network will not reliably converge to a good solution.

The Moving Target Problem

Each of the hidden units of an ANN is trying to evolve into some feature detector that will benefit the networks' overall computation. But as all the other units are also changing at the same time, this task is highly complicated one. With no communication directly between the units of a hidden layer, each unit only sees the error signal backpropagated to it from the networks' output. This signal defines the problem, but changes constantly, with the result that it takes very long time before all the units settle down. Backprop learning time increases as the number of hidden layers in a network is increased. Fahlman and Lebiere [7] believes this, among other factors, is due to the moving target effect; units in the hidden layers see a constantly changing picture, and this makes it impossible to converge decisively to a good solution.

The Cascade-correlation algorithm tries to overcome both these problems. In order to choose a reasonable step size, we need to know not just the slope of the error function, but something about its higher-order derivatives - its curvature - in the vicinity of the current point in weight space. This information is not available in the standard back-propagation algorithm. With the cascade architecture, the hidden units are added to the network one at a time and its input weights are essentially frozen, only the output connections are trained repeatedly. This combats the moving target and the step-size problems.

Chapter 3

Problem

This chapter describes some of the challenges related to the underlying prediction problem and introduces ways to handle these. Identifies properties that should be implemented in the training and testing.

3.1 Time Series Prediction with Artificial Neural Networks

In the following we assume a feed-forward multilayer perceptron (MLP) of arbitrary topology.

When modeling the time series that is used for ANN training data, we look to well known methods in statistics. In analogy to a non-linear autoregressive AR(n) [18], a variable y_t is autoregressive of order n if it is a function of its past values. At a point in time t (t=1,...,T), a one step ahead forecast y_{t+1} is computed using observations y_t, y_{t-1},...,y_{t-n}, with n being the number of inputs. The model can be expressed as:

$$y_{t+1} = f(y_t, y_{t-1}, ..., y_{t-n}) \tag{3.1}$$

Figure 3.1 shows the application of the AR(n) model. The net is a MLP with four input neurons for observation in time t, t-1, t-2 and t-3, four hidden units and one output neuron for time t+1. The net is fully connected, i.e. 20 trainable weights. The figure illustrates how the relevant time-series data extracted from Betfair is mapped to the input layers of the neural net to be trained.

The autoregression-like model above is not sufficient for representing multiple input vectors. A more appropriate model would be the multiple regression model. When dealing with more than one variable x_1,x_2,...,x_k observed at times t=1,t=2,...,T, the time series can be expressed as:

$$y_{t+1} = f(x_{1,t}, x_{2,t}, ..., x_{1,t-1}, x_{2,t-1}, ..., x_{k-1,t-n}, x_{k,t-n}) \tag{3.2}$$

As the data extracted from Betfair consists of multiple data types, sampled simultaneously, this model can be used to represent the ANN data.

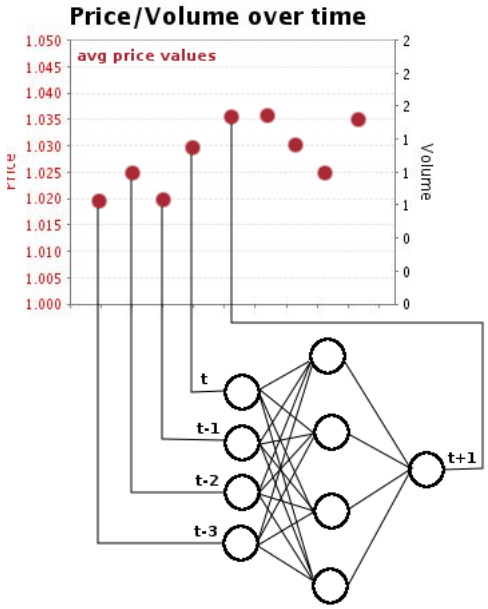


Figure 3.1: Example of neural network application to time-series forecasting

3.2 Binary markets and binary prices

3.2.1 Binary markets

Tennis match odds betting is a form of a binary market. The market has only two outcomes, either player A wins the match, or player B wins the match. However, on the betting exchanges you can both back and lay the same selection, thus presenting four different options to the trader[1]. Figure 3.2 shows these four options.

In a binary market, laying one selection technically means the same as backing the

Selections: (2)	100.9%		Back	Lay		99.6%
Robin Soderling	1.18 39492	1.19 22578	1.2 11641	1.21 10347	1.22 19846	1.23 14225
Stefan Koubek	5.4 22599	5.5 76	5.7 2268	5.9 64	6 2087	6.2 31

Figure 3.2: Example of a binary market with four different options available to the trader

opposite selection, and vice versa. Thus, to always get the best possible price, you have to investigate which of these two reciprocal options to use. Let us use Figure 3.2 as an

[1]Throughout this report, people involved in Betfair markets are referred to as traders. However, many people operating on betting exchanges only make wagers, hence making them gamblers or "punters"

example. if you think Robin Soderling is going to win the match, you can either back Soderling at a price of 1.2 or you can lay his Stefan Koubek at the price of 5.9. In this case, laying Koubek would yield the best price. Converting the lay price to back price with Equation 3.3 :

$$Backprice = \frac{1}{1 - \frac{1}{layprice}} \tag{3.3}$$

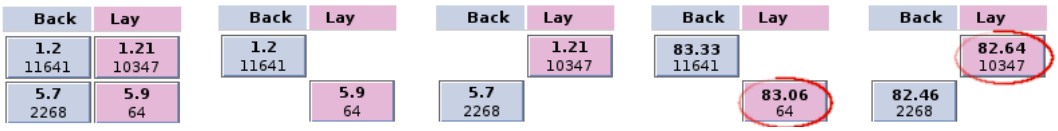
gives an effective back price on Robert Soderling of 1.204, which is better than the 1.2 currently available to back. Conversely, converting a back price into a corresponding lay price is done with Equation 3.4.

$$Layprice = \frac{1}{1 - \frac{1}{Backprice}} \tag{3.4}$$

3.2.2 Combining two selections onto one ladder for binary markets

In an effective binary market, the prices of the two selections will behave directly contrary of each other. E.g. in a tennis match, if one of the players break his opponent, the odds of the player will decrease while the odds of the opponent will increase correspondingly. However, in this project, we are not concerned with specific selections (tennis players). Which selection wins the match is irrelevant; it is the market, or the match as a whole that are interesting. Consequently, we need one price which defines the market as one.

A relatively easy solution to this is to define each market as the binary price of player A (the player that appears first). E.g. in Figure 3.2, it is Robin Soderling that defines the market. The market price will be in a binary form on weather Robin Soderling will win the match or not. It is the highest price/lowest probability currently available for this selection which defines the current market quota for the back quote, and the lowest price/highest probability that defines the lay quote. The solution is illustrated in Figure 3.3. The market in the example currently has back price of 83.06 and a lay price of



(a) Market odds (b) Back/The ask (c) Lay/The bid (d) Binary - Back (e) Binary - Lay

Figure 3.3: The binary prices defining the match odds market

82.64. The back and the lay bears strong resemblance to "The ask" and "The bid" of

the stock market. Also similar to the stock market, the difference between these two prices are called the spread [17]

There are several advantages in defining the tennis match odds markets in this way. Firstly, as mentioned earlier, it is intuitively easier for humans to comprehend the probability for events happening in terms of percentages rather than the various odds forms 2.3. Secondly, it is easy to see the benefits of having one global price defining a market rather than two inversely coupled ones. This also makes further comparisons to financial markets, including financial trading strategies/methods, more feasible. But the main reason to define the markets in this way is due to the manner in play tennis markets are traded on Betfair. In play markets particularly differ from pre-match markets in terms of speed of fluctuations, and liquidity. During a live tennis match, prices changes very rapidly. A consequence is most often that it is only one of the selections that are being traded on. The liquidity on the market as whole is healthy (i.e. trading activity is high), but only one of the players are being traded, while the other is not. Usually, the player currently favorite (decimal odds $\leq$ 2.0/binary price $\geq$ 50), is the one being traded. The other selection may sometimes not have any price available.

If an Artificial Neural Network is trained with inaccurate or inadequate data, it can not generalize the problem or present a solution to it. If the ANNs are trained with two different prices representing the same market, or indeed if it only concentrated on trading one fixed selection, there would gaps in the price input patterns. Thus, ANNs would become inaccurate due to non-existent or inadequate data. By having one price to represent the market, guarantees (given that the market is liquid [17]) that at any given time, there will be price data describing the current market situation.

3.3 Trading Strategies

A trading strategy is a clear set of predefined rules (or trading formulas) to apply to the trading. The rules do not deviate based on anything other than market action, emotional bias [9] in the trading is thus eliminated.

3.3.1 Stop Loss

Stop loss rules are used to protect the capital invested, or the funds being locked up in a market position by automatically placing counter orders (bets) once the price drops below/rise above a certain threshold set by the trader. E.g. if a odds trader utilizing a stop loss strategy at +-10% has backed Roger Federer to win a match against Rafael Nadal at a binary price of 75, the trader will get out of the position (lay - and take the loss) if the price hits 67.5. There are two types of stop loss strategies:

- **Fixed Price** stop loss. Creates a counter bet- when a fixed price level is crossed by the current ruling price.

- **Trailing Price** stop loss. Creates a counter bet - when a variable price level is crossed by the current ruling price. The variable price level can e.g. be determined by fixed amount or fixed percentage of the high/low of the ruling price.

3.3.2 Stop Profit

A Profit Stop is, in the same way as stop loss, the close of a trade at a specified level, with the difference being that the trade is a profitable trade and the close is executed in order to lock in the profit.

3.3.3 Money management - Staking

There exists an abundance of staking systems or staking strategies with the best known including:

- **Fixed stakes** - staking the same amount on each selection.
- **Fixed profits** - staked varied based on odds to ensure same profit for each winning selection
- **Kelly** - uses formula f = $\frac{bp-q}{b}$, where f is the fraction of the current bankroll to wager, b is the odds, p is probability of winning and q is the probability of losing (1-p).
- **Martingale** - involves doubling the stake after each loss.

A staking system is important to any betting or trading strategy as it defines how much risk you are taking on each bet.

3.3.4 Balancing trade profit/loss

Trading binary markets on Betfair differs somewhat from trading financial markets in regards to locking in profits/losses. The commodity being traded is a bet rather than a stock or a futures contract. Trading bets means trading probabilities. When exiting a position in e.g. the stock market, the profit/loss is instantly locked in; basically the difference between the prices when entering the market and exiting the market defines the acquired profit or loss. However, when trading probabilities, the profit or loss is directly attached to the actual probability being traded. With a flat staking strategy[2], once exiting a trade, the potential profit/loss is only realized if the coherent event of the probability happen. For instance, if you were to back a selection at a price of 1.7 for 1000 NOK, and then later laying the same selection at a price of 1.5 for 1000 you have made successful, profitable trade. The profit is (1.7-1.5)*1000 NOK = 200 NOK, but unfortunately only on this selection. This means that this selection win, a profit of 200 NOK is realized - if the other selection win, 0 NOK is realized (Nothing won, nothing lost). With the exit odds of 1.5, the probability of realizing the profit is 75%. One might

[2]Flat staking strategy: Backing and Laying for the same amount of money

say that the result of the trade was a "risk free" bet, at odds of 1.2.

As mentioned in 2.1, one may balance the profit or loss by adjusting the "exiting" stake proportionately to the "entering" stake. By doing this, the profits or losses are spread over the two selections and not dependent on one of them winning the match. If the profit or loss are spread evenly across the selections, the result of the actual trade is no longer a bet, but realized p/l. Equation 3.5 are used to distribute the return of the trades evenly among the selections.

$$ExitStake = \frac{EnteringOdds}{CurrentOdds} \times EnteringStake \tag{3.5}$$

Equations 3.6 and 3.7 shows the resulting profit/loss after the exit stake has been calculated with 3.5. The equations differ in regards to what the initial position in the market was. If the initial position was back, equation 3.6 is used. Conversely, if the initial position was lay, equation 3.7 is used for calculating the return of the trade.

$$Profit/Loss = ExitLayStake - EnterBackStake \tag{3.6}$$

$$Profit/Loss = EnterLayStake - ExitBackStake \tag{3.7}$$

To illustrate how this works, consider the example above. You backed at odds of 1.7 with a stake of 1000 NOK. The odds then drop to 1.5. You decide to trade out of your position, but this time you want the resulting trade to lock in a profit that is independent of the outcome of the match. To do this, equation 3.5 is used to calculate how much the exiting lay stake should be: 1.7/1.5 * 1000 NOK = 1133.33 NOK. The resulting profit of the trade is then obtained by using equation 3.6 : 1133.33 NOK - 1000 NOK = 133.33 NOK.

3.4 Training with Trading Strategies

Function approximation problems can be split into two classes [16];

- **Classification problems** are problems where the objective is to identify (classify) whether the input belongs to one or more groups. The output is discrete, e.g. odds-market direction.
- **Regression problems** are problems where the output is real-valued, e.g. fixed-odds prediction.

Training a neural net to the classification problem of whether the market price will go up or down in the next time slice does not indicate the potential profitability of the net [17]. Even if the net has a very high hit rate, i.e. classifies market direction significantly more correct than incorrect, the net return of the trades can be negative. Consider the situation where a trained neural net predicts over 90% correct market movements.

However, each of these correctly predicted trades only generate small profits, while the remaining 10% which is incorrect, amounts to big losses - bigger than the combined small profits. The hit rate consequently is a poor evaluation metric for the networks.

In this work, the task of trading an odds market is approached as a regression problem. Instead of predicting whether or not the "market will go up or down", a new and more refined objective is required. Having the trained ANNs produce a real valued output could potentially have beneficial aspects. For instance, a real-valued output could hold information on not only the market direction (up or down), but also say something about the magnitude of this direction, i.e. how much the market will go up or down. This magnitude could potentially be exploited in form of the "degree of confidence" in each ANN produced trading signal. This magnitude could e.g. be incorporated in the trading strategies' staking plans by influencing how much risk is to be attached in each trade.

3.5 Problem Specific Cost Functions

The overall objective of the prediction task in this project is to generate maximum profits. Unless it is perfectly trained, a neural networks output differs from the desired outputs. The network learns the underlying relationship through minimization of this difference. However, the real significance of the difference depends on the area of application. The function evaluating the significance of the output error should thus be specific to the problem of odds trading, and reflect the actual profit or loss resulting from each trade.

Assume the desired outputs of the time series being real values ranging from 0 to 1, with 0.5 being the midpoint separating the two different trading options. Output values >0.5 indicates back decisions, and output values <0.5 indicates lay decisions. The difference between the output value and the midpoint, describes the magnitude of the projected odds fluctuation. Consider the case where a trained network runs a previously unseen pattern and produces an output of e.g. 0.52, a back signal. Imagine two different scenarios. First, let's say that what actually happened to the odds was that it decreased, and the produced trading signal to back was a correct decision. However, the actual output should have been 0.56, i.e. the odds decreased more than was predicted with a difference of 0.04. Conversely, consider the situation where the actual output was 0.48, the "correct" option was to lay, and the produced back signal was wrong. Also this time the difference between real and produced output is 0.04, but with a different sign. In general, prediction theory focuses on quadratic error functions and least-square predictors, implying a symmetric and quadratic cost relationship [5]. If the network above was trained with a symmetric, quadratic cost function, e.g. with the MSE function described in Section 2.4.2, the costs of the two different situations would be the same. However, if we study real world impact of the scenarios, they are not. The negative effect of doing the opposite of what actually happen must neces-

sarily be greater than the negative effect of producing the "correct" signal, but with wrong magnitude. A cost function that generates identical errors in the two scenarios does not reflect the real world cost. Errors identical in magnitude, cause different costs and the cost function must reflect this. Producing an output of 0.52 when the real output was 0.56 should still cause some cost, however far from the same cost as if the real output was 0.48. The objective cost function should thus penalize incorrect trading decisions more than correct ones, even if the error magnitude of the correct one is bigger.

Each error measure should be selected in accordance to the underlying problem structure, representing approximations of the objective function [5]. The problem of trading a live tennis market should have a cost function that considers the following properties of odds trading, identified by the author:

- More important to predict few big fluctuations, compared to many small
- Wrongly predicted trading signals more penalized than correct ones, indifferent to magnitude
- Prioritize predicting correct type of signal, then magnitude
- Sign of error plays a critical role
- Error not necessarily non-quadratic

The above properties must be taken into consideration when implementing a cost function suitable for the underlying problem. Assuming the effective market hypothesis to be true [17] and that the time series behave like "random-walk" process, in the long run, it can be claimed that on average, anybody or anything will achieve a 50 % hit rate on predicting whether the market will go up or down. A hit rate of 50 % however, results in loss over time, due to the overhead costs of trading. When training the neural nets, the hit rate is not a sufficient evaluation metric. A low hit rate, even much lower than 50 % is acceptable, as long as the trained network is able to predict the large trades, and are able to capitalize these predictions to yield a overall profitable result.

One thing to keep in mind when constructing specialized cost functions, is that it has to be fully differentiable to allow the calculation of the error's derivatives for training with the gradient descent algorithm. [5]

3.6 The generalization of tennis odds markets

The generalization ability is the measure for all learning algorithms of the ability to give accurate predictions when presented with unseen data [17]. Under the assumption that there exists a function relating the correct outputs to inputs for all tennis odds markets, and that this function is equal for all tennis odds markets, one could potentially train a neural network with data from every tennis market. A neural network could be trained

with data from e.g. 100 random tennis odds markets from last year and make valid predictions on tennis odds markets taking place today. This mindset corresponds to the idea that the underlying data of all the tennis odds markets not only holds the same function(s), but each market holds also complementary properties which contribute to the generalization ability.

In the contrary case, one may argue that every tennis odds market is different. There exists no applicable underlying function in the data among different markets, because there are different "actors" in every market. The liquidity of each market is different, and hence also the market efficiency. If the same yardstick was applied to every tennis odds market, the opportunity to discover and potentially utilize individual market characteristics might be missed. This mindset corresponds to the idea that the underlying data of every tennis odds market holds its own distinct function, and that this function can not generalized across different markets.

Chapter 4

Methods and implementations

This chapter describes the methods used and the implementation chosen.

4.1 Overall Method

Based on the findings in [17], the general idea of this thesis was to continue the experimentation and identify new approaches towards the prediction task. The implementation in [17] provided a basis on which to build future tests and experiments. Neural network training and testing is an experimental process, with a high degree of trial and error before conclusive results are achieved. The overall method of this thesis has been to test different neural network structures, with different input parameters both regarding the training algorithms and input/output data. One of the main conclusions of [17] was that the overall potential profitability of a neural net could most likely be increased by using training functions which to a higher degree inhibited the properties of odds trading. Attempts have been made to identify such a function. The construction of a custom cost function (4.1.1) has been a result of analyzing not only the underlying data and its representation, but also the nature of trading odds in general.

4.1.1 Custom Cost Function

The proposed cost function tries to reflect the properties identified in Section 3.4 and Section 3.5. Function focuses on profitability rather than merely the distance between desired an actual output. It is defined as follows:

$$e(d,y) = \begin{cases} (d-y)^{2e^z} & \text{for d>M, y>M, (d - y)>0} \\ -((d-y)^{2e^z}) & \text{for d>M, y>M, (d - y)<0} \\ (d-y)^{2e^{-z}} & \text{for d>M, y<M} \\ (d-y)^{2e^z} & \text{for d<M, y<M, (d - y)>0} \\ -((d-y)^{2e^z}) & \text{for d<M, y<M, (d - y)<0} \\ -((d-y)^{2e^{-z}}) & \text{for d<M, y>M} \end{cases}$$

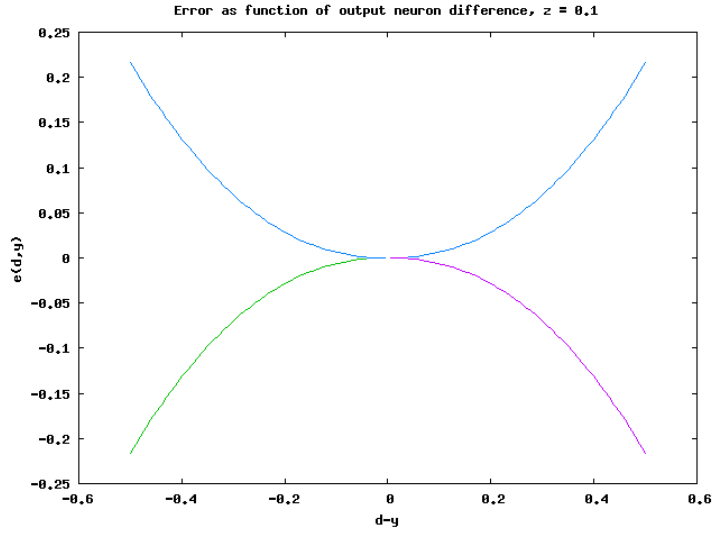


Figure 4.1: The base of the cost function almost reduces to the squared error of the neuron output.

Where d = desired output, y = actual output, z = y - M and M = Midpoint (E.g. 0.5 as the example from 3.5). The four different graphs corresponds to the two different desired trading signals, back and lay, each with different options in relation to the two actual output signals.

The proposed cost function has some defining properties. It has two differentiable components. The base of the equation is the difference between the desired and actual output, just as the squared error cost function. The exponent is the exponential function of z. The graph of the base of the cost function is shown in Figure 4.1. In the example, the z is held constant at 0.1. The graph then reduces to a standard cost function, much similar to the Squared Error function. Figure 4.2 shows the exponent of the cost function, with the neuron difference, d-y, held constant at 0.3. It illustrates how the cost function behaves at large neuron difference values. The slopes of the graphs are not very steep, and behave almost linear. Figure 4.3 shows the exponent, with the neuron difference, d-y, held constant at 0.03. At smaller neuron difference values, the cost functions reward correct big fluctuation predictions more aggressively, and conversely, penalizes wrongly predicted large movements more aggressively.

4.2 Overall Implementation

Parts of the overall implementation of [17] forms the basis of the implementation in this work. The Fast Artificial Neural Network (FANN) 4.2.1 library is used for ANN training, testing and network configuration. The application for building data patterns,

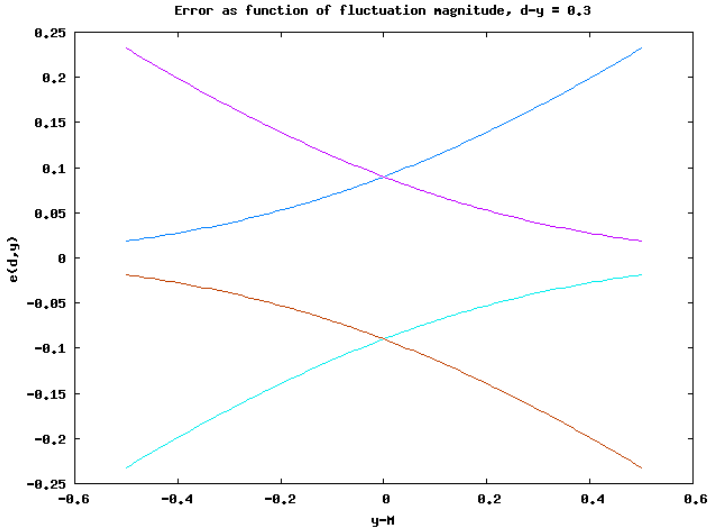


Figure 4.2: The error as function of fluctuation magnitude. The neuron difference, is 0.3.

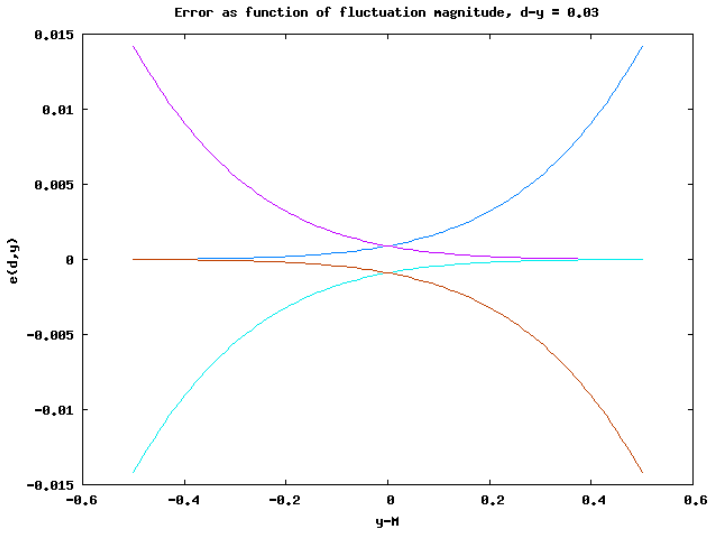


Figure 4.3: The error as function of fluctuation magnitude. The neuron difference, is 0.03.

generating tests and analyzing ANN input and output was implemented in C++ 4.2.2 and the QT framework 4.2.3. The application for extracting data through the Betfair API was developed in Java 4.3.3 and has only been exposed to minor modifications throughout this project.

4.2.1 FANN

FANN [1] is a free open source neural network library, which implements multilayer artificial neural networks in C with support for both fully connected and sparsely connected networks. It includes bindings for C++. FANN offers back propagation training with the R-prop, Quick-prop, batch and incremental training algorithms. In order to train with custom cost functions, these was implemented directly in the library code. The reason for this is that the library only supports callback functionality to the net after each epoch, and not after each training pattern.

4.2.2 C++

As mentioned above, the main structure of the application was implemented in [17]. However, new functionality, and modification of old features was necessary. This included:

- Logic for building patterns with binary prices
- Exception handling for rejecting non-valid patterns
- Functionality for evaluating tests
- Framework for running trading simulations, during and after training
- Callback functionality for doing custom things during training
- Modification of code to fit new evaluation metrics.

The application has constantly undergone changes throughout the entire project, in order to accommodate the constantly evolving experiments.

4.2.3 QT

In order to visualize many of the processes, and specially the Betfair extracted data and the ANN input patterns, the QT GUI development framework was utilized. An example of the application GUI can be seen in Figure 4.4

4.3 ANN Data

The performance of a neural network is critically dependent on the training data. The training data must be representative of the task to be learnt [3]. One of the biggest challenges in [17] was the collection, filtering and preparation of data. This task proved to be very time consuming, and occupied valuable time that could have been used development and testing.

[1] http://leenissen.dk/fann

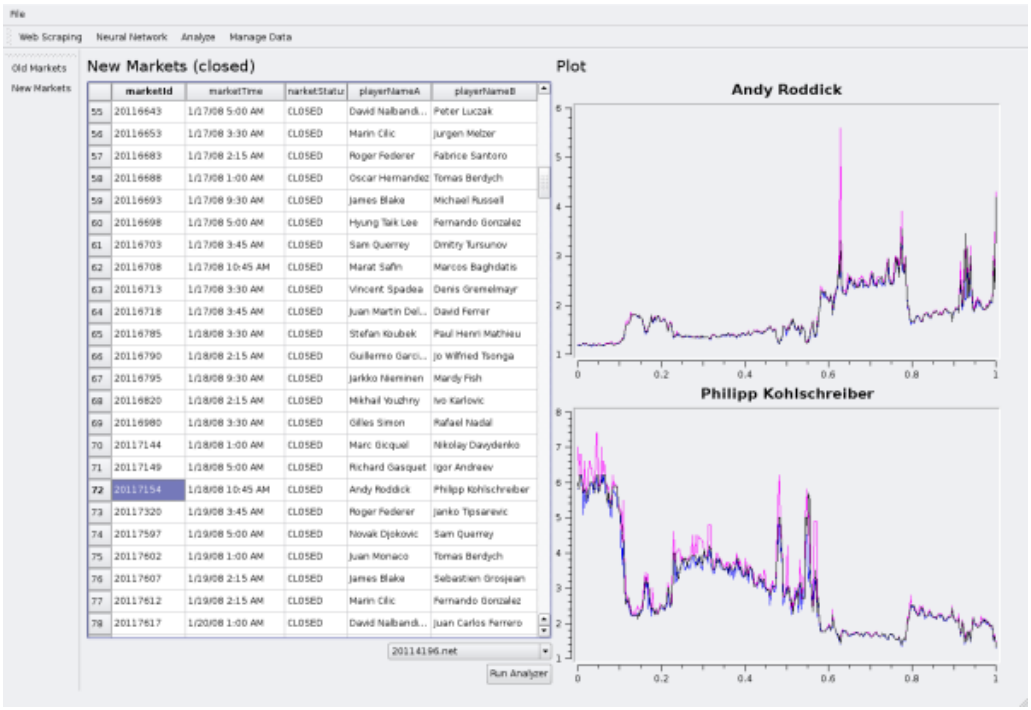


Figure 4.4: The figure shows the Betfair extracted price data of the two players of a match. The blue line indicates the back price available, the pink lines shows the lay price available, while the black lines shows at which level the prices are matched.

4.3.1 Temporal data

With origin in stock market literature, [17] defined the available data as *technical data* and *fundamental data*. The fundamental data proved to be of little significance to the prediction rate. One of the problems with the fundamental data, is its static nature. As the data source [2] only is updated once a week, this kind of data is not suitable for dynamic time series prediction. Moving the focus from pre-match markets which ranges over several days, to in-play markets which only lasts a few hours, the data is required to be even more dynamic. Consequently, the fundamental data from [17] is rejected. However, a new fundamental data candidate emerges; the in-play tennis statistics (e.g. first serve percentage, unforced errors etc. of the players during the match). Unfortunately, it is almost impossible to get hold of, and utilize this kind of data by means of an automatic process in time for it to be useful. Unlike Betfair, which offers an API, there currently is no web site or service that can provide this kind of fundamental data "as it happens". Atptennis.com has a live scoreboard, which contains many of these stats, but it is updated at a far too slow and unstable rate for it to be considered in this project.

[2] http://www.atptennis.com

The ANNs of this work is consequently exclusively being trained with technical data. The data is extracted directly from Betfair 4.3.2 with an automated agent described in Section 4.3.3. The data constantly changes over time, and are sampled at a rate of minimum once per 30 seconds.

4.3.2 Betfair API

Betfair offers various API products to their customers. This makes it possible to build custom applications with direct access to the data and services normally given through Betfairs' web interface. The different API products differs in terms of calls allowed to the web service per minute, and are priced accordingly. Due to the limited budget of this project, the Free Access API was chosen. The limitation of the Free Access API is described in [17].

4.3.3 Betfair Data Extractor Agent

In order to extract the required data needed for ANN training, and later on, automatically placing bets according to the trading signals produced by the neural nets, a custom java application was implemented. During the duration of this project, this application constantly ran while ATP tennis markets were offered by Betfair. The class diagram for application is depicted in Figure 4.5. All the data acquired from Betfair through the Betfair Data Extractor Agent is dumped in MySQL database. This database runs on a private server. In order to reliably provide consistent and meaningful data, the data agent had to be functional at all times. This robustness requirement meant that extra attention was given to the exception- and error handling of the agent.

4.4 ANN Input

4.4.1 Sliding Windows Technique

In order to accommodate the dynamic nature of the temporal data that time series are, a sliding window technique was applied. The general sliding window technique is described in [17], and is illustrated in Figure 4.6. The implemented program builds the ANN input patterns in the following way. A sliding window with a user defined number of slices is specified. The sample rate of the data extracted from Betfair is variable, so in order to ensure that the time frame, or time distance between each data point is constant throughout different markets, averages are calculated for each time slice. As the Betfair Data extractor agent guarantees to collect at least two data points per minute, a time frame of 30 seconds is the smallest definition permitted by the data. The average of each distinct type of data points are calculated within the defined time slice. This is done for each slice defined in the window. After all averages are calculated for all types of data-points, for each slice in the window, an input pattern is returned. The sliding window then shifts one time slice to the right, i.e. if the defined time slice length was 30 seconds, the start and end of the window is forwarded 30 seconds. The process proceeds

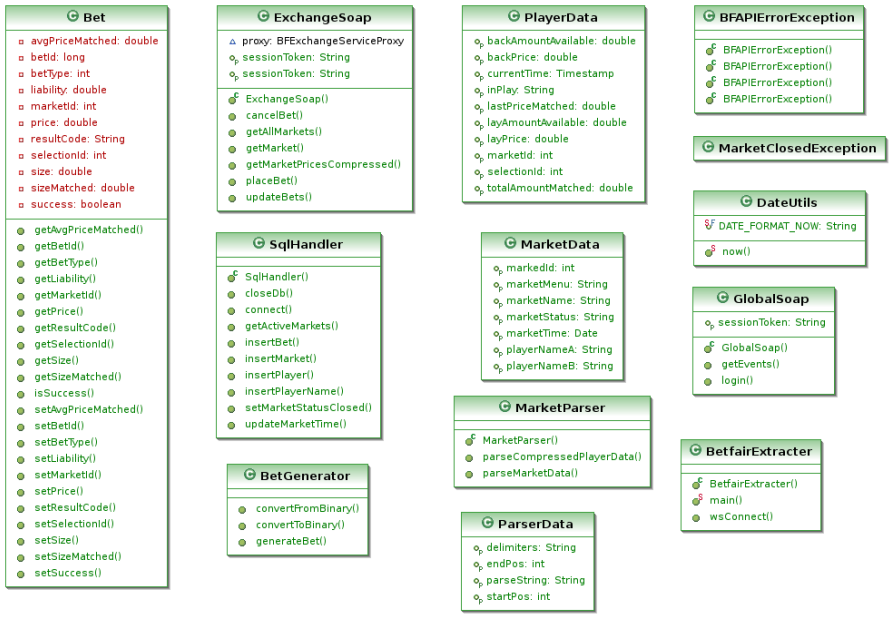


Figure 4.5: Betfair Data Extractor Agent class diagram

until the end of the market data is reached. The building of the ANN input patterns are summarized below:

for all Sliding Windows W **do**
  for Slices S[i], $i = 0$ to n **do** {n, user defined}
    for all Technical Data T **do**
      Average Data(W, S, T) = SUM(data points) / NUM(data points)
    end for
    $i \leftarrow i + 1$
  end for
end for

The task of building ANN input patterns are, due to large volumes of data, a time consuming process. Each market may consist of several thousand records of data, and each of these requires queries to the database. Functionality for saving and loading retrieved data from the DB by means of serialization was implemented in order to make the ANN training and testing more time efficient.

4.4.2 From odds to binary prices

The prices extracted from Betfair are in the form of decimal prices, and are also saved in the database in this format. When the data is retrieved from the database, and built into ANN input patterns they are converted to binary prices. As discussed in Section

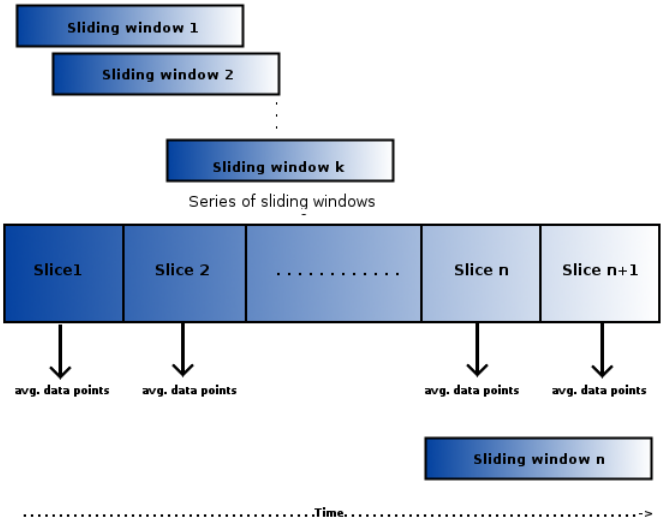


Figure 4.6: The sliding window used in the initial tests.

3.2, the four betting options are reduced into two, and with that defining the market as the binary prices of player A. This is done by using equations 3.3 and 3.4 to extract the "best" price for the corresponding trade.

4.4.3 Scaling

All the ANN input and output data are scaled. This is to prevent individual features to dominate others as a result of having more influence on the input to an ANN unit. Without scaling, for instance the "amount available" input vectors, which could have values ranging from 1 to several millions, would heavily dominate the other smaller inputs. To ensure such "unfair" bias to individual features, all the values are scaled between 0.1 and 0.9. Several of the activation functions, including the Sigmoid and the Gaussian functions has a target range from 0 to 1. The min and max limits are set to 0.1 and 0.9 to help prevent the network from grinding to a halt through running at the extreme limits of its operating range [3].

4.5 ANN Output

As described in 3.4, the prediction task is approached as a regression problem. The proposed prediction output looks at the difference between slice n and slice n+1 in the sliding window.

4.5.1 Log Difference

The pure numerical difference in price between slice n and slice n+1 in the sliding window technique, does not correctly represent the trading returns in the underlying fluctuation. To illustrate this, consider the two trading situations in which a) you backed at a binary price of 25 and layed at a price of 50, and b) you backed at 50 and layed at 75. In both scenarios the difference in price is 25. However, the two cases do not yield the same return. Using the balancing profit method of 3.3.4, case a) would yield a return of 150 % while case b) would return 50 %. As the difference between the prices in a) and b) are the same, but the trading return in the two cases is different, the pure numerical difference does not suffice.

Using the logarithmic difference of the prices in the two cases gives a more accurate situation. The logarithmic differences are $\log(\frac{50}{25}) \neq \log(\frac{75}{50})$ and with $\log(\frac{50}{25}) < \log(\frac{75}{50})$, and therefore models the underlying return potential more correctly. The logarithmic difference is approximately the same as the percentage change. The practice of using logarithmic differences is common in financial analysis. Generally, the output of the trained ANNs is defined by Equation 4.1:

$$\log(\frac{binaryprice_{n+1}}{binaryprice_n}) \tag{4.1}$$

with positive outputs indicating increase and negative outputs indicating a decrease in market price.

Chapter 5

ANN Testing and Results

This chapter presents the results of training and testing the various ANNs. The results are presented as they are; no discussion or conclusions are made, but are left to Chapter 6. All the graphs presented in this chapter can also be studied in greater detail in the Appendix A and Appendix B.

5.1 General Configuration

Training and testing ANNs often appears to be an *ad hoc* process in that most problems require much experimentation before acceptable results are attained [3]. The results presented in this report only represent a small fraction of the actual training and testing conducted. Experimentation with different network topology and configurations, training parameters and features - classifications/outputs were conducted. The most promising tests and results are documented in this report. Figure 5.1 shows the sliding window technique and features/output employed in these tests. Each time slice is one minute, and contains data for :

- The amount available to back on player A.
- The amount available to lay on player A
- The amount available to back on player B.
- The amount available to lay on player B
- The back price [1]
- The lay price [1]

The output of the network is the logarithmic difference between the mean of the back and lay price of slice n+1, and the mean of back/lay price of slice n as described in Section 4.5.

[1]The binary market back/lay price as described in 3.2.2

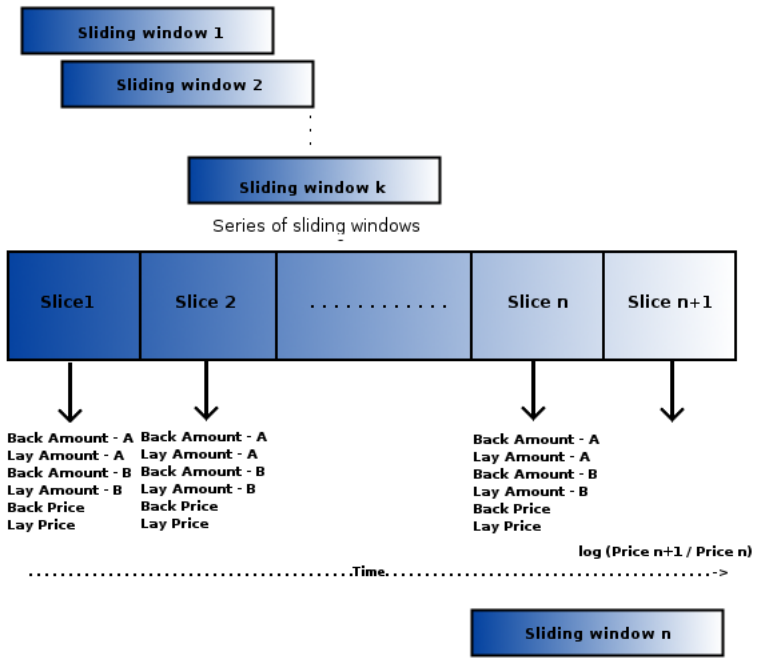


Figure 5.1: The sliding window technique used in the tests.

5.2 Initial Testing

The initial tests are meant to be a basis for later testing, but are also subject for comparison. No particular trading strategy is implemented. The networks are trained to produce trading signals to indicate which direction the market is presumed to fluctuate in the following time slice. In Test 1-1 5.2.1 the neural nets are trained on a fixed portion of a match, and consequently tested on the remainder of the same match. Test 1-2 5.2.2, 1-3 5.2.3, 1-4 5.2.4, 1-5 5.2.5, 1-6 5.2.6 are trained on a set of matches, and are then tested on a completely different set of matches. The latter nets are trained with separate training methods, network structures and parameters in order to identify the most beneficial and suitable methods/configurations for the data at hand. All the initial tests are trained with the MSE cost function.

Test 1-1	
ANN parameters	
Training Method:	CASCADE
Max no neurons: Desired error: Training Algorithm: Cost function: Bit Fail limit: Learning Rate: Momentum:	100 0.0000000001 RPROP Linear 0.35 0.7 0.0
ANN Data	
Output Features	1 8 * 6 = 48
Training set: Test set:	60(min) * 10 = 600 patterns Remainder of match * 10 (variable)

Table 5.1: The training parameters and data for Test 1-1

5.2.1 Test 1-1: Cascade Correlation - Separate Markets

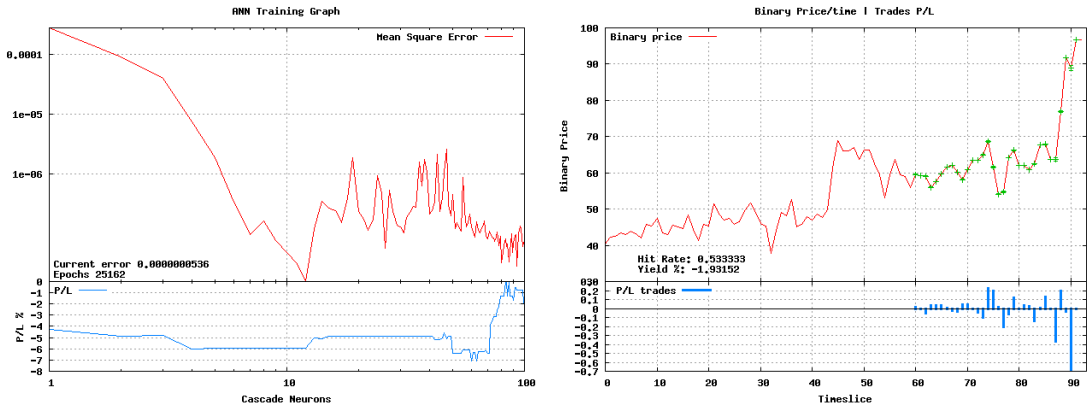
The tests are trained with the Cascade correlation 2 algorithm 2.4.3. The training data consists of 10 unique tennis matches, each trained separately as proposed in Section 3.6. The networks are trained with data up until a fixed time-limit in each market, and are then tested on the remaining data of the match. Table 5.1 summarizes the training parameters and data for the initial tests. The 10 tennis matches constituting the data sets were randomly selected. The only criteria was that the matches were played best of 3 sets [2]. The pool of matches is displayed in table 5.2. All the matches took place during the first months of 2008, and are from the ATP tour. The actual results of the match is unimportant, but are listed as a curiosity.

The graphs below show the results of the initial training and testing. The graphs to the left (a), displays the Mean Square Error (MSE) of the neural net as it is trained with the cascade correlation algorithm with a maximum of 100 neurons. The sub-graph below the MSE, exhibits the development of the total Profit/Loss on the test set as the network is being trained with the training set. After each neuron is added to the net, a trading simulation on the test set is run, and the P/L% calculated. The graphs to the right (b) displays the details of the trading on the test set, after the network is finished training (100 neurons added, desired error never reached). The upper graph shows the development of the binary market price throughout the match. The green crosses on the graph marks where trades are entered; they are exited in the following time slice. The

[2]Grand Slam tournaments, some Davis cup and Masters series matches are played best of 5 sets and consequently not considered when selecting the data for the initial tests.

Date	Match	Result
3/4/08	Amer Delic v Jurgen Melzer	7-5 7-6(2)
2/28/08	Andy Roddick v Gilles Muller	6-4 7-6(4)
2/13/08	Marin Cilic v Mikhail Youzhny	4-6 6(3)-7
2/14/08	Andy Murray v Stanislas Wawrinka	3-6 7-6(5) 6-1
3/4/08	Philip Kohlschreiber v Rafael Nadal	6-3 1-6 4-6
3/5/08	Fabrice Santoro v Novak Djokovic	3-6 6(3)-7
2/29/08	Ivan Ljubicic v Mario Ancic	7-6(2) 6-4
2/27/08	Stefan Koubek v Olivier Rochus	2-6 4-6
3/4/08	Janko Tipsarevic v Feliciano Lopez	3-6 6-4 4-6
1/31/08	Carlos Berlocq v Luis Horna	7-6(3) 3-6 7-6(3)

Table 5.2: The tennis matches/markets totaling the data set



(a) Training Data

(b) Trading Data

Figure 5.2: Test 1-1: Amer Delic v Jurgen Melzer

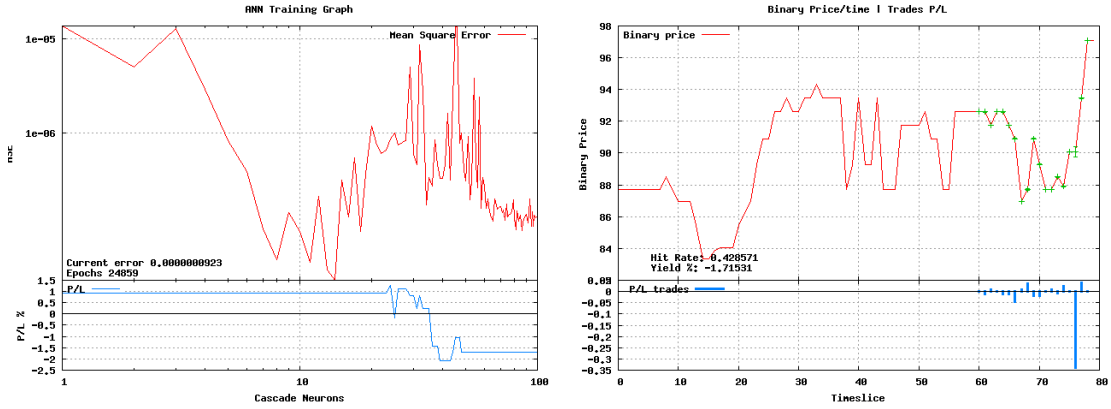
sub-graphs show the returning P/L of each trade.

5.2.2 Test 1-2: Cascade Correlation Combined Markets

Test 1-2 are trained with the same credentials as in Test 1-1, but this time uses all the 10 matches in Table 5.2 as training data. The trained net is then traded on a pool of 378 matches. The trading results of the trained network can be seen in Table 5.7. The training method is the Cascade correlation algorithm.

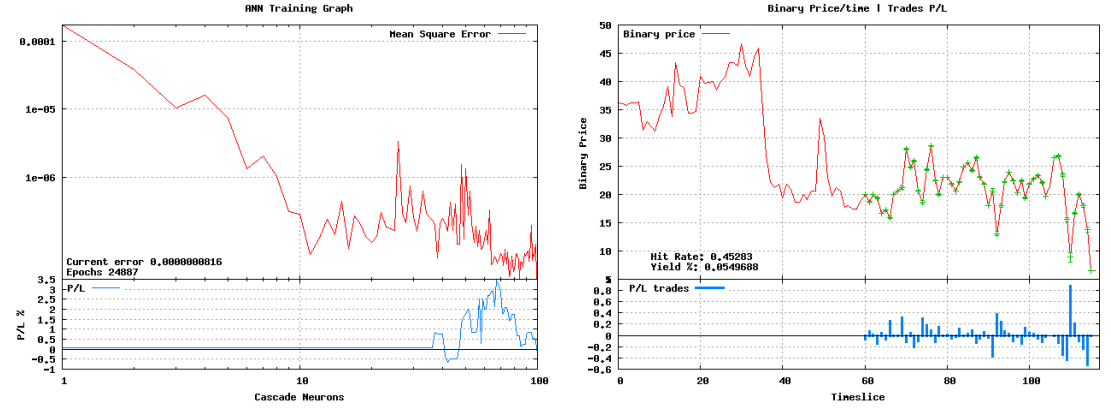
5.2.3 Test 1-3: Incremental

Test 1-3 is, as the former test, trained with all the 10 matches in Table 5.2. The training algorithm used is the standard back propagation algorithm, where the weights are updated after each training pattern (incremental). The activation function used is



(a) Training Data (b) Trading Data

Figure 5.3: Test 1-1: Andy Roddick v Gilles Muller



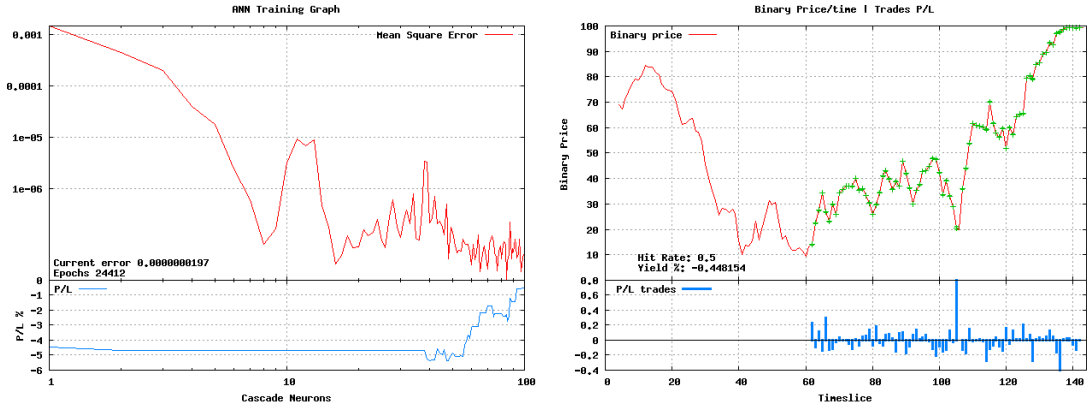
(a) Training Data (b) Trading Data

Figure 5.4: Test 1-1: Marin Cilic v Mikhail Youzhny

the sigmoid function. The training parameters of Test 1-3 is summarized in Table 5.3. The trading results of the trained network can be seen in Table 5.7.

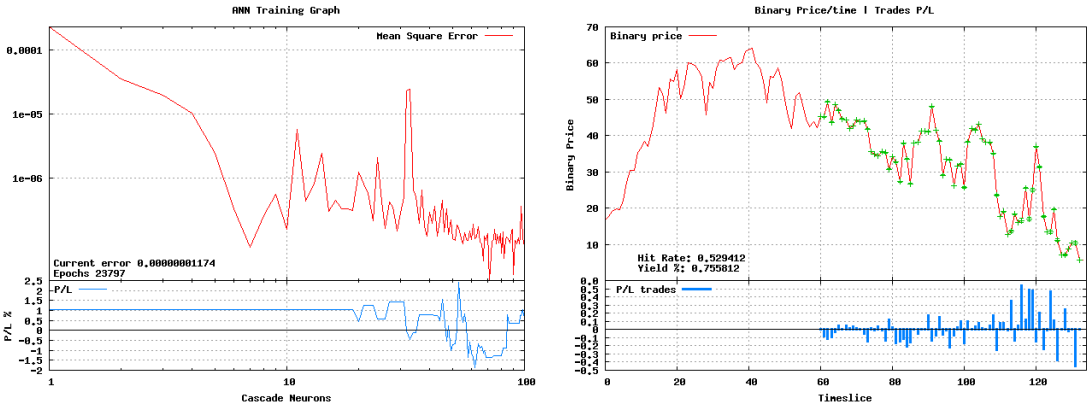
5.2.4 Test 1-4: Batch

In Test 1-4, the training method is the standard back propagation algorithm, this time with weight updating after calculating the mean square for the whole training set (batch). The training data is the same as in the above tests. However, some adjustments have been done to the net structure, parameters and activation functions, this is summarized in Table 5.4. The learning rate parameter is lowered to 0.6, and the net is structured with 3 hidden layers. The trading results of the trained network can be seen in Table 5.7.



(a) Training Data (b) Trading Data

Figure 5.5: Test 1-1: Andy Murray v Stanislas Wawrinka



(a) Training Data (b) Trading Data

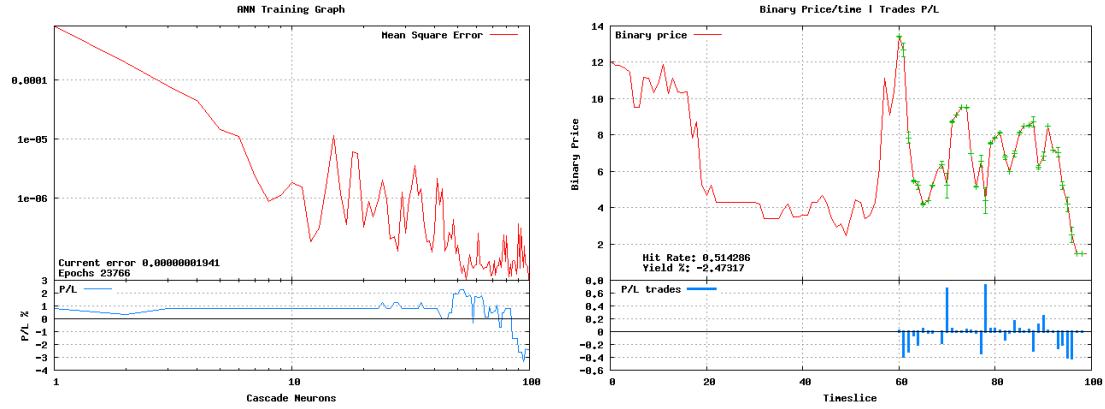
Figure 5.6: Test 1-1: Philip Kohlschreiber v Rafael Nadal

Test 1-3		
ANN parameters		
Training Method: Hidden layers: Output layer:	FANN TRAIN INCREMENTAL FANN SIGMOID FANN SIGMOID	
Epochs: 10000	Learning Rate: 0.7	Momentum: 0.0
No Hidden Layers: 1	Neurons: 48–64–1	

Table 5.3: Configurations of Test 1-3

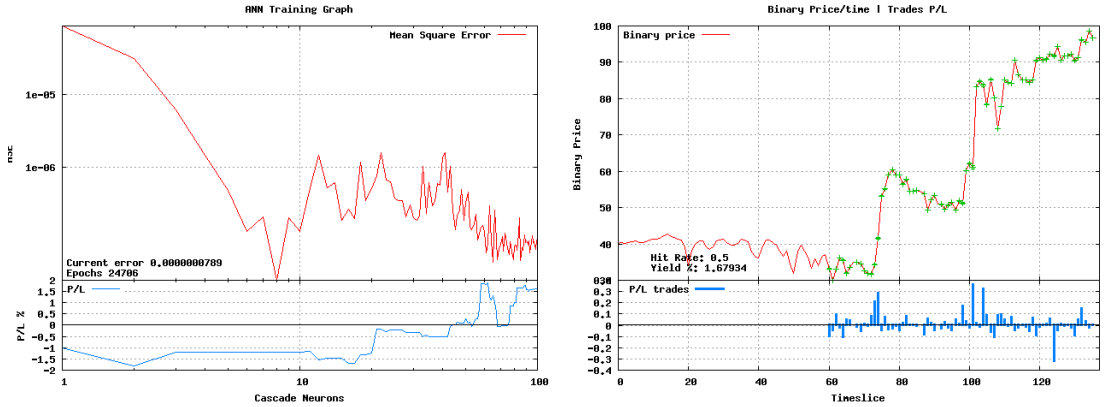
5.2.5 Test 1-5: R-prop

A version of the Resilient Propagation method (RPROP) training algorithm is used in Test 1-5, the iRPROP [11] algorithm. RPROP is a more advanced batch training



(a) Training Data (b) Trading Data

Figure 5.7: Test 1-1: Fabrice Santoro v Novak Djokovic



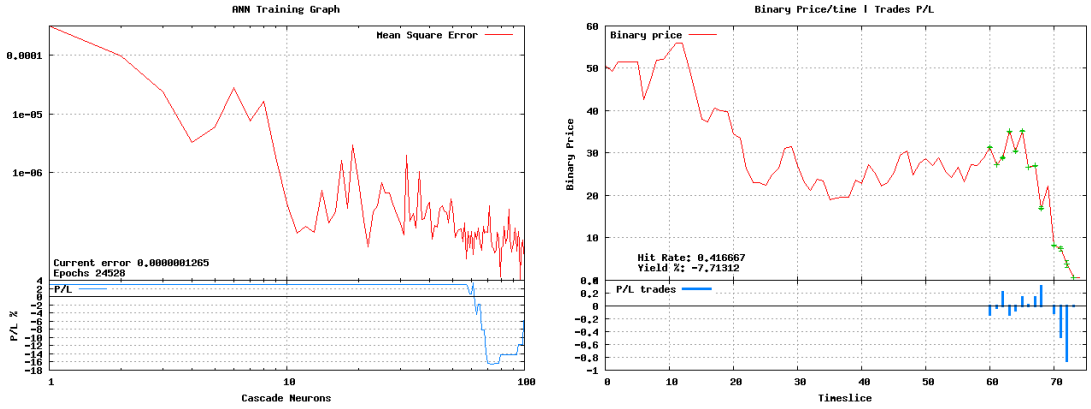
(a) Training Data (b) Trading Data

Figure 5.8: Test 1-1: Ivan Ljubicic v Mario Ancic

Test 1-4		
ANN parameters		
Training Method: Hidden layers: Output layer:	FANN TRAIN BATCH FANN SIGMOID FANN LINEAR	
Epochs: 10000	Learning Rate: 0.6	Momentum: 0.0
No Hidden Layers: 3	Neurons: 48–48-64-16–1	

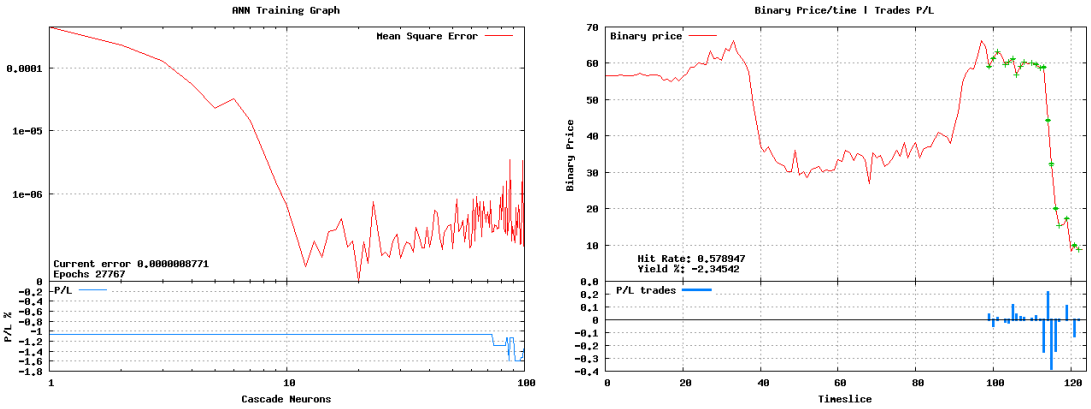
Table 5.4: Configurations of Test 1-4

algorithm. It is an adaptive algorithm, and does not use the learning rate parameter. The Gaussian activation function is used in the hidden layers, while the Sigmoid function



(a) Training Data (b) Trading Data

Figure 5.9: Test 1-1: Stefan Koubek v Olivier Rochus



(a) Training Data (b) Trading Data

Figure 5.10: Test 1-1: Janko Tipsarevic v Feliciano Lopez

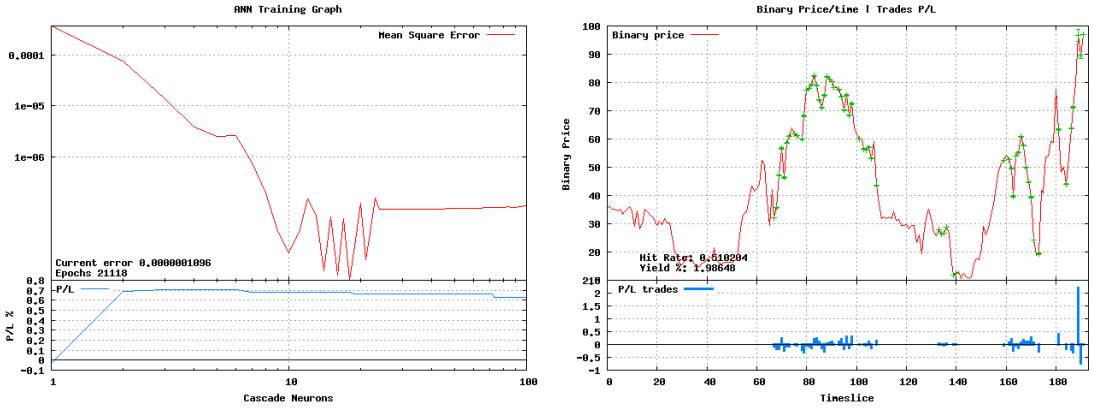
is used in the output layer. The net has two hidden layers, with 128 and 96 neurons respectively. 5.5. The trading results of the trained network can be seen in Table 5.7.

5.2.6 Test 1-6: Quick-prop

Test 1-6 uses the Quick-prop algorithm [8]. The trading results of the trained network can be seen in Figure 5.7 The training MSE on the various nets are shown in Figure 5.12

5.2.7 Test 1-7: Full Scale Incremental

Test 1-7 was trained with the same network structure and parameters as in Test 1-3, listed in Table 5.3. The difference was that it was trained for 15,000 epochs and the training



(a) Training Data (b) Trading Data

Figure 5.11: Test 1-1: Carlos Berlocq v Luis Horna

Test 1-5		
ANN parameters		
Training Method: Hidden layers: Output layer:	FANN TRAIN PROP FANN GAUSSIAN FANN SIGMOID	
Epochs: 10000	Learning Rate: NA	Momentum: 0.0
No Hidden Layers: 2	Neurons: 48–128-96–1	

Table 5.5: Configurations of Test 1-5

Test 1-6		
ANN parameters		
Training Method: Hidden layers: Output layer:	FANN TRAIN QUICKPROP FANN SIGMOID FANN SIGMOID	
Epochs: 10000	Learning Rate: 0.7	Momentum: 0.0
No Hidden Layers: 3	Neurons: 48–128-96-16-1	

Table 5.6: Configurations of Test 1-6

data consisted of the full data set; 320 markets for training and 68 markets for testing by trading simulations. This was approximately a 80-20% training-test distribution. While training the net, it was constantly tested on the 68 markets, to see the evolving profit/loss. The training graph, and the P/L graph are shown in Figure 5.13

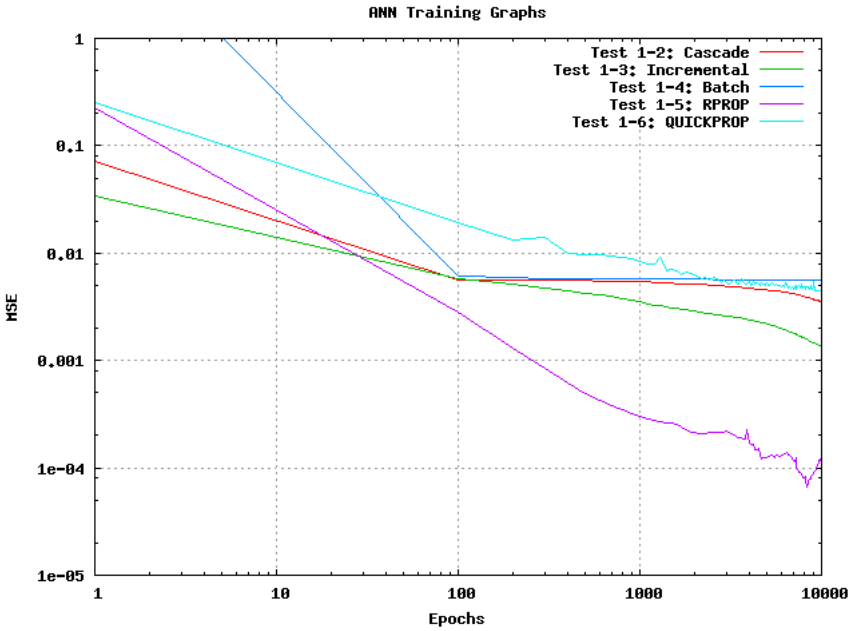


Figure 5.12: The MSE of the trained networks. Test 1-2 Cascade was trained with 100 neurons, but appear in the graph for comparison reasons. The x-axis is logarithmically scaled.

Results of trading				
1-2 Yield %	**1-3 Yield %**	**1-4 Yield %**	**1-5 Yield %**	**1-6 Yield %**
0.243899	**0.506836**	**0.223963**	**0.158223**	**0.363955**

Table 5.7: The results of trading with the trained networks of Tests 1-2, 1-3, 1-4, 1-5, 1-6. The trading is conducted on a pool of 378 matches, totaling 45336 patterns

5.3 Custom Cost Function Testing

The ANNs of this section has been trained with the specialized cost function proposed in Section 4.1.1. The networks are trained to produced not only signals as to indicate which direction the market are presumed to fluctuate in the following time slice, but also the magnitude of this fluctuation. This is further exploited when the profitability of the nets are evaluated by the objective error function which simulates the trades. The error function uses the predicted fluctuation magnitude as a staking parameter of each trade. This yields a dynamic staking strategy.

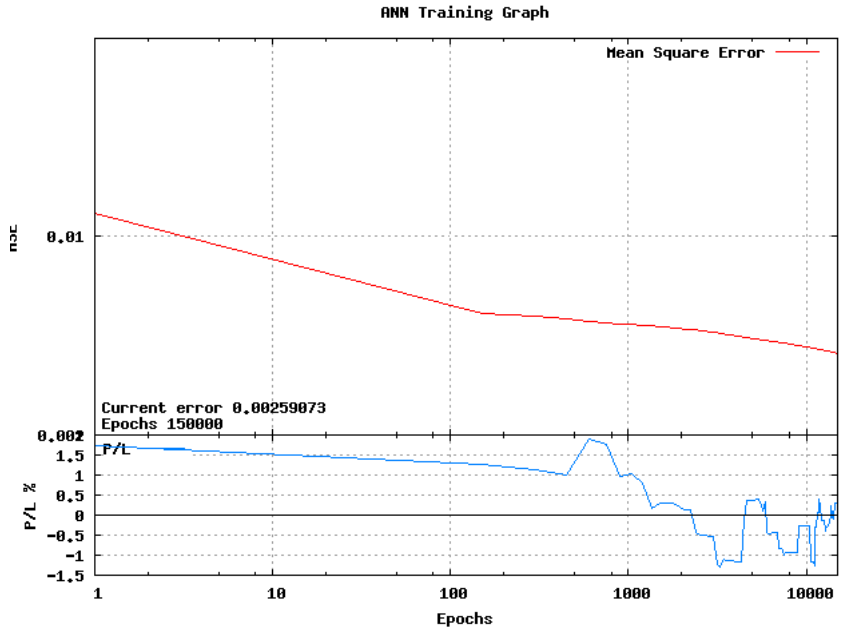


Figure 5.13: The error of the network trained in Test 1-7. The sub-graph shows how the profit/loss of the test set evolves as the network are being trained. The x-axis is logarithmically scaled.

5.3.1 Test 2-1: Cascade Correlation - Seperate Markets

Test 2-1 consists of the same pool of matches as were trained in Test 1-1 5.2.1, shown in Table 5.2. The same training method is used, the training are conducted on a fixed portion of a match, and subsequently tested on the remainder of the same match. Cascade correlation training is used, for a maximum of 100 neurons. The parameters of the training are summarized in Table 5.8

5.3.2 Test 2-2: Cascade Correlation - Combined Markets

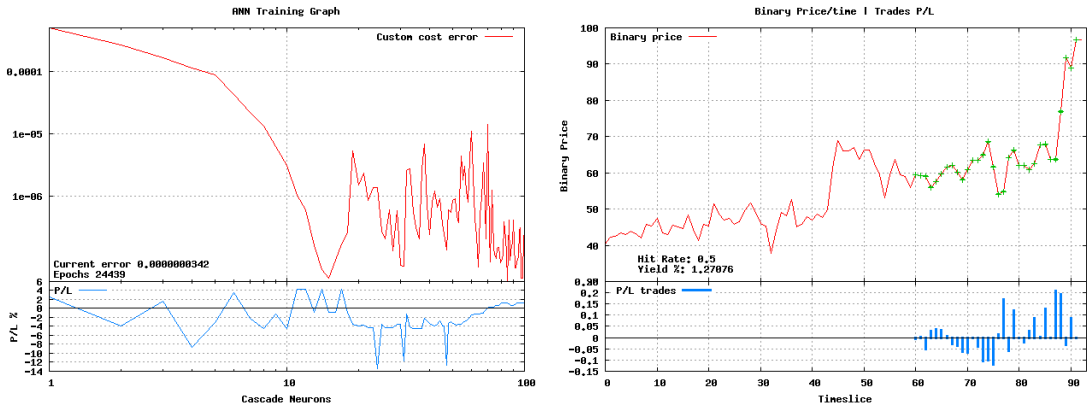
Test 2-2 is trained with the specialized cost function and the cascade correlation training algorithm, the remaining credentials are identical to that of Test 1-2 5.2.2. The training graph is displayed in figure 5.24, and the trading results are given in Table 5.9.

5.3.3 Test 2-3: Incremental

Test 2-3 is trained with the specialized cost function and the incremental training algorithm, the remaining credentials are identical to that of Test 1-3 5.2.3. The training graph is displayed in figure 5.24, and the trading results are given in Table 5.9.

Test 2-1	
ANN parameters	
Training Method:	CASCADE
Max no neurons: Desired error: Training Algorithm: Cost function: Bit Fail limit: Learning Rate: Momentum:	100 0.0000000001 RPROP Custom Cost Function 0.35 0.7 0.0
ANN Data	
Output Features	1 8 * 6 = 48
Training set: Test set:	60(min) * 10 = 600 patterns Remainder of match * 10 (variable)

Table 5.8: The training parameters and data for the custom cost function tests



(a) Training Data

(b) Trading Data

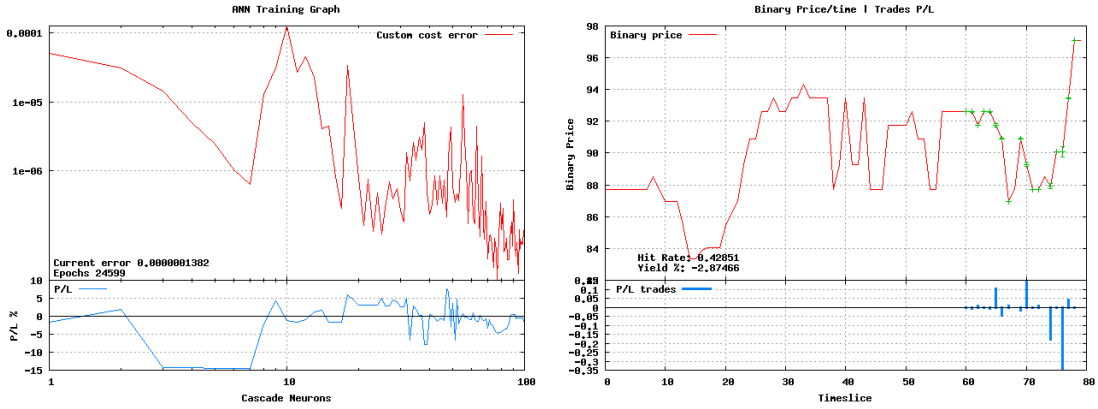
Figure 5.14: Test 2-1: Amer Delic v Jurgen Melzer

5.3.4 Test 2-4: Batch

Test 2-4 is trained with the specialized cost function and the batch training algorithm, the remaining credentials are identical to that of Test 1-4 5.2.4. The training graph is displayed in figure 5.24, and the trading results are given in Table 5.9.

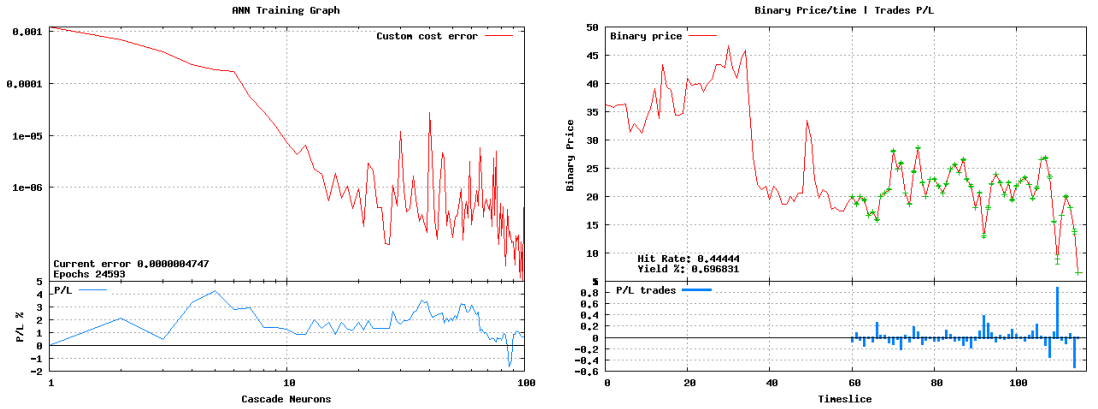
5.3.5 Test 2-5: R-prop

Test 2-5 is trained with the specialized cost function and the R-prop training algorithm, the remaining credentials are identical to that of Test 1-5 5.2.5. The training graph is



(a) Training Data (b) Trading Data

Figure 5.15: Test 2-1: Andy Roddick v Gilles Muller



(a) Training Data (b) Trading Data

Figure 5.16: Test 2-1: Marin Cilic v Mikhail Youzhny

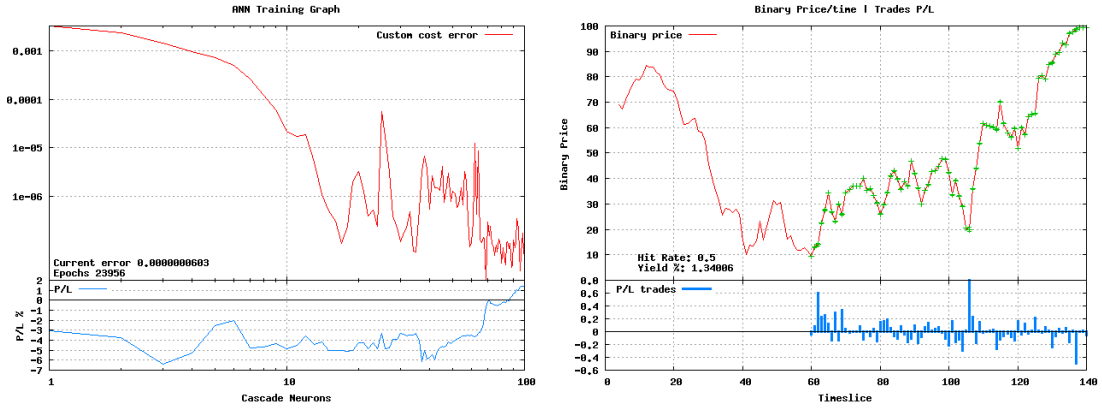
displayed in figure 5.24, and the trading results are given in Table 5.9.

5.3.6 Test 2-6: Quick-prop

Test 2-6 is trained with the specialized cost function and the Quick-Prop training algorithm, the remaining credentials are identical to that of Test 1-6 5.2.6. The training graph is displayed in figure 5.24, and the trading results are given in Table 5.9.

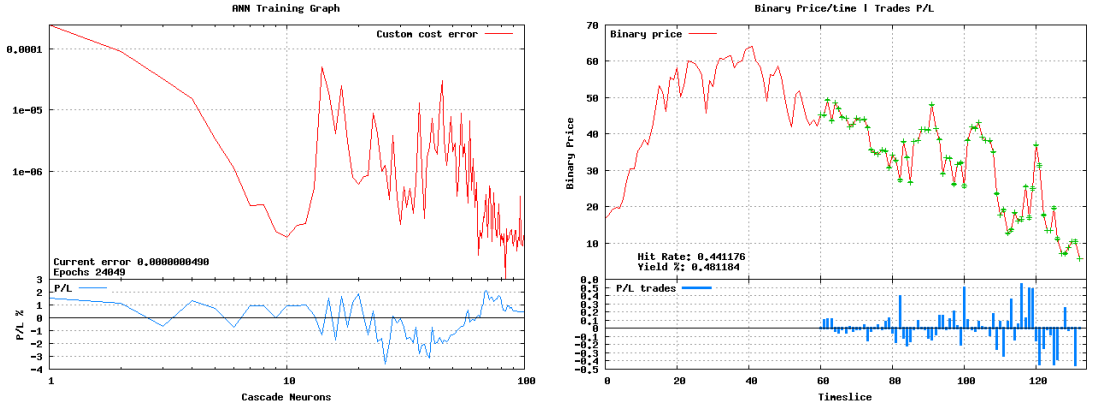
5.3.7 Test 2-7: Full Scale Batch

This test was trained and tested on the full data set, with an 80-20% train-test distribution. The training input parameters to the net are identical to the one in Test 1-4, listed in Table 5.4. The training graph, and the p/l graph are shown in Figure 5.25



(a) Training Data (b) Trading Data

Figure 5.17: Test 2-1: Andy Murray v Stanislas Wawrinka



(a) Training Data (b) Trading Data

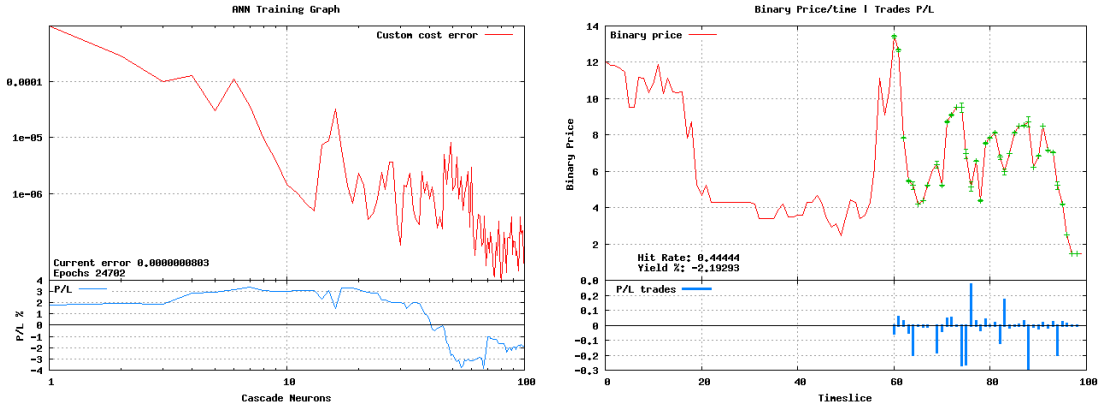
Figure 5.18: Test 2-1: Philip Kohlschreiber v Rafael Nadal

Results of trading				
2-2 Yield %	**2-3 Yield %**	**2-4 Yield %**	**2-5 Yield %**	**2-6 Yield %**
0.378753	**1.8436**	**3.17309**	**0.815817**	**1.84849**

Table 5.9: The results of trading with the trained networks of Tests 2-2, 2-3, 2-4, 2-5, 2-6. The trading is conducted on a pool of 378 matches, totaling 45336 patterns

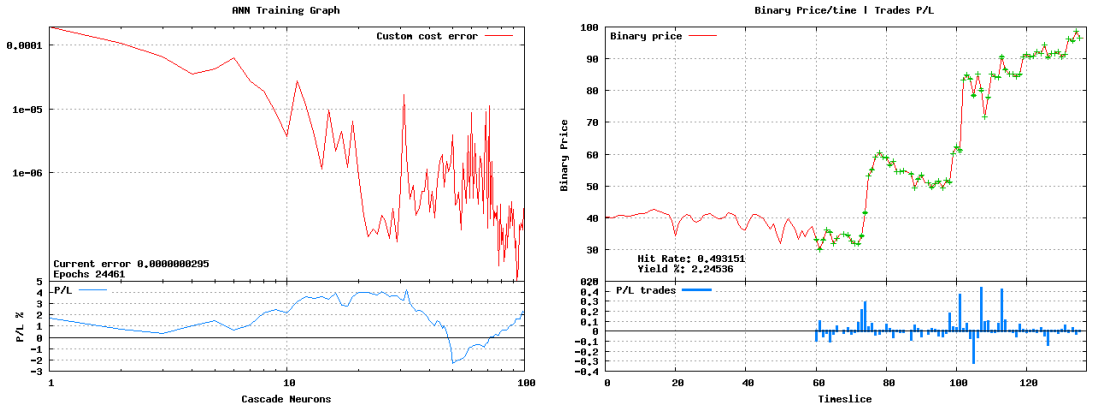
5.3.8 Test 2-8: Full Scale Cascade Correlation - Seperate Markets

Test 2-8 is the full scale version of Test 2-1. 5.3.1 The test were set up to train and trade on all the matches, except the ones in Test 2-1, individually. The test used a version of the Kelly staking strategy introduced in Section 3.3.3. The initial bank was set to 1000.



(a) Training Data (b) Trading Data

Figure 5.19: Test 2-1: Fabrice Santoro v Novak Djokovic



(a) Training Data (b) Trading Data

Figure 5.20: Test 2-1: Ivan Ljubicic v Mario Ancic

How the bank evolves during the training and trading of the different markets can be seen in Figure 5.26

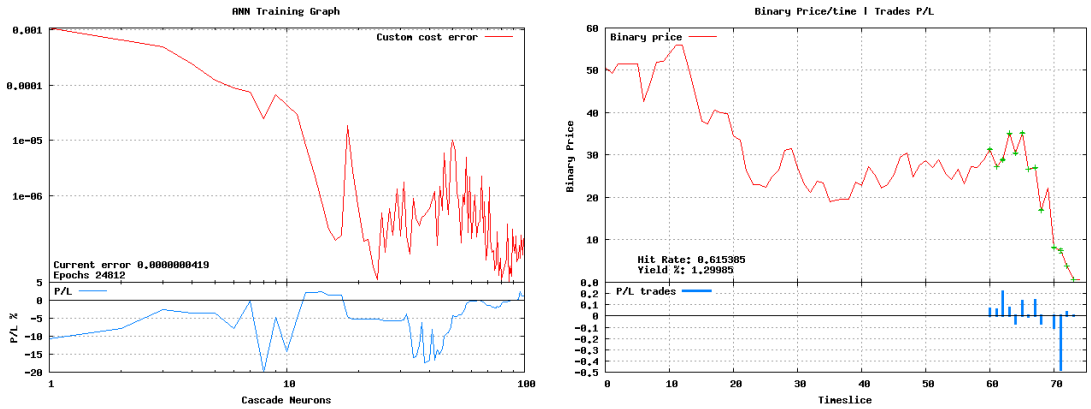
5.4 Comparison

5.4.1 Test 1-1 vs. Test 2-1

Comparison of the results from Test 1-1 5.2.1 and Test 2-1 5.3.1 are listed in Table 5.10

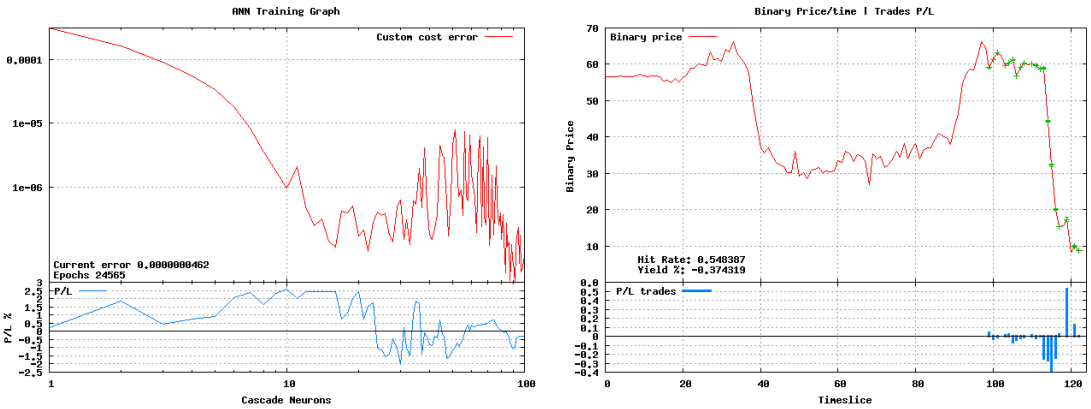
5.4.2 Test 1-(2-6) vs. Test 2-(2-6)

Comparison of the trading yields from Test 1-(2-6) 5.2.2 and Test 2-(2-6) 5.3.1 are listed in Table 5.11



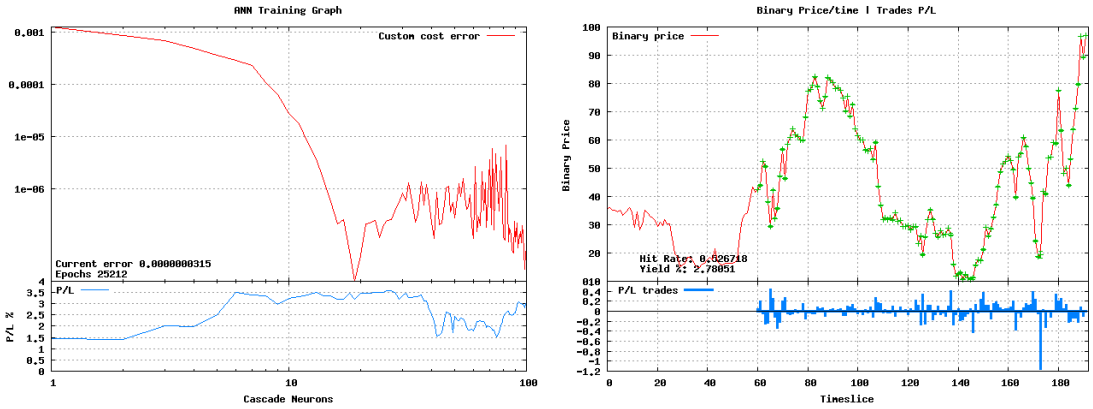
(a) Training Data (b) Trading Data

Figure 5.21: Test 2-1: Stefan Koubek v Olivier Rochus



(a) Training Data (b) Trading Data

Figure 5.22: Test 2-1: Janko Tipsarevic v Feliciano Lopez



(a) Training Data (b) Trading Data

Figure 5.23: Test 2-1: Carlos Berlocq v Luis Horna

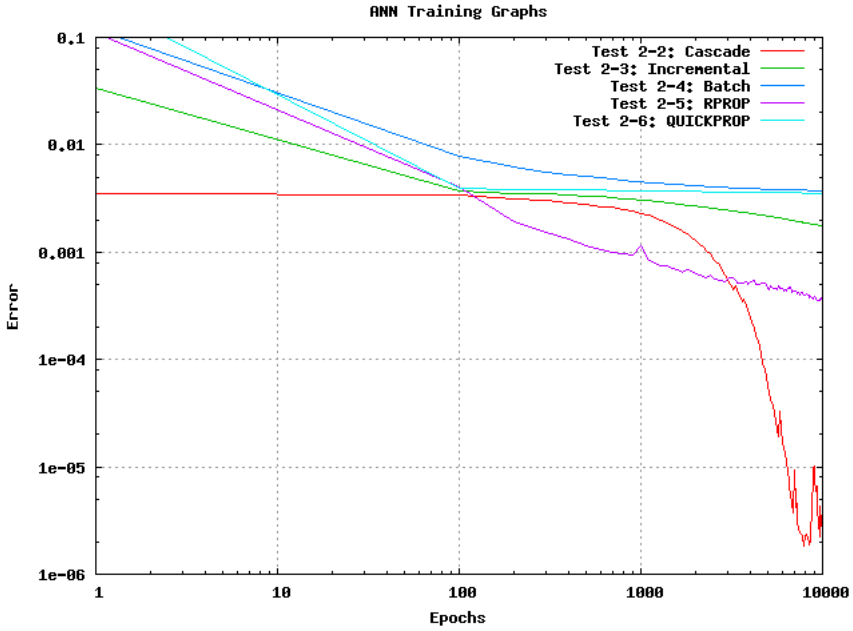


Figure 5.24: The error of the trained networks. Test 2-2 Cascade was trained with 100 neurons, but appears in the graph for comparison reasons. The x-axis is logarithmically scaled.

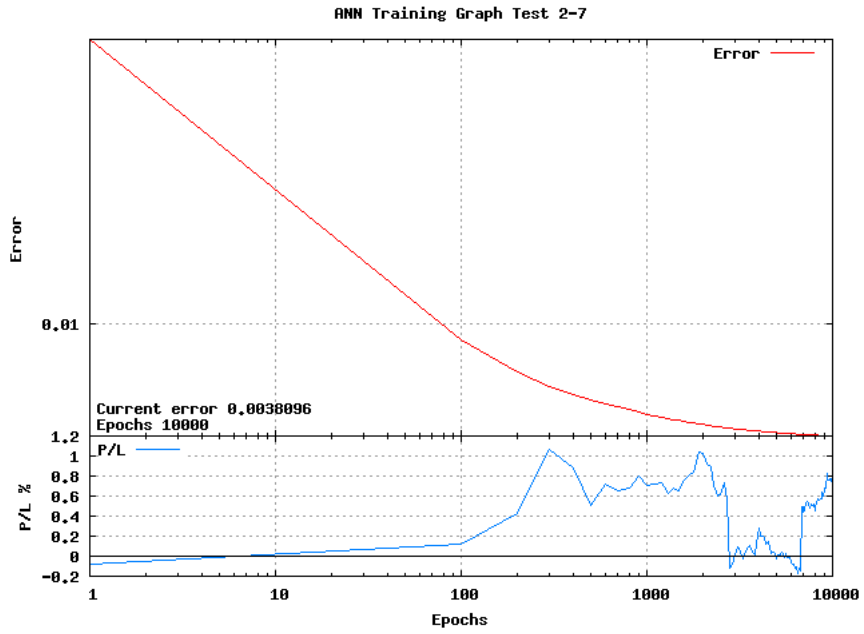


Figure 5.25: The error of the network trained in Test 2-7. The sub graph shows how the profit/loss of the test set evolves as the network are being trained. The x-axis is logarithmically scaled.

Match	**Test 1-1**		**Test 2-1**	
	Error	PL%	Error	PL%
Amer Delic v Jurgen Melzer	5.36e-8	-1.93152	3.42e-8	1.27076
Andy Roddick v Gilles Muller	9.23e-8	-1.71531	1.382e-7	-2.87466
Marin Cilic v Mikhail Youzhny	8.16e-8	0.0549688	4.747e-7	0.696831
Andy Murray v Stanislas Wawrinka	1.97e-8	-0.448154	6.03e-8	1.34006
Philip Kohlschreiber v Rafael Nadal	1.1174e-8	0.755812	4.90e-8	0.481184
Fabrice Santoro v Novak Djokovic	1.1941e-8	-2.47317	8.03e-8	-2.19293
Ivan Ljubicic v Mario Ancic	7.89e-8	1.67934	2.95e-8	2.24536
Stefan Koubek v Olivier Rochus	1.1265e-8	-7.71312	4.19e-8	1.29985
Janko Tipsarevic v Feliciano Lopez	8.771e-8	-2.34542	462e-8	-0.374319
Carlos Berlocq v Luis Horna	1.096e-8	1.98648	3.15e-8	2.78051
Average	**4.5915e-8**	**-1.215009**	**5.55960e-7**	**0.4672646**

Table 5.10: Comparison of the results from Test 1-1 and Test 2-1, trained with the MSE and custom cost function, respectively

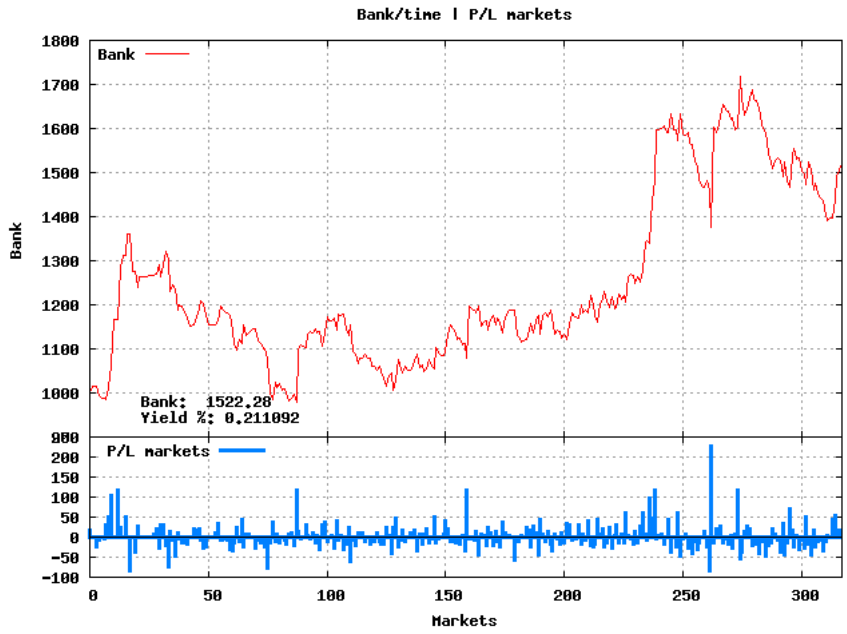


Figure 5.26: The full scale version of test 2-1. Uses Kelly staking strategy. Starts with a bank of 1000, final bank is 1522.28 after training and trading 317 individual markets

Results of trading					
	-2 Yield %	**-3 Yield %**	**-4 Yield %**	**-5 Yield %**	**-6 Yield %**
Test 1-	**0.243899**	**0.506836**	**0.223963**	**0.158223**	**0.363955**
Test 2-	**0.378753**	**1.8436**	**3.17309**	**0.815817**	**1.84849**

Table 5.11: Comparison of the yields achieved in the Initial tests vs. the custom cost function tests

Chapter 6

Discussion

This chapter will present and discuss the findings from the tests and experiments conducted in Chapter 5

6.1 General Approach

As mentioned previously, and as the reader likely already knows, neural network learning is a field of trial-and-error and is thus time consuming. It is never guaranteed to find an optimal solution to the problem at hand, and the different settings and configurations that can be used to train a neural net are endless. All the settings can not be tested, but in this thesis an effort has been made to systematically identify favorable configurations and then go on from there. The process can be seen as a brick-by-brick process. Although not all the nets trained during this project are included in this report, the ones that are were the most promising.

6.2 Evaluating Performance

Evaluating the performance of the trained ANNs are first and foremost executed in light the yield percentage returns on the various test sets. The overall motivation 1.4 for this thesis was to utilizing ANNs to trade the Betfair in-play tennis match odds markets profitably. However, in order for the analysis of the results to be meaningful, other measures of performance are identified. These measures are not only part of the actual performance evaluation, but also a key indicator of related difficulties and potential improvement. The alternative measures does not only relate to the actual results, but also the process of achieving the results.

Profit

The main performance metric is the overall profit achieved when testing on previously unseen data. In finance, the yield is defined as the profit obtained from an investment,

and more specifically the annual rate of return expressed as a percentage. More specifically, in betting terminology, the yield is used to describe the ratio of profit to the total amount staked. In trading, only the entering trade affects the total amount staked.

Training Error

The training error basically defines how much the network output differs from the desired output. Many of the most common functions for how this error is calculated was described in Section 2.4.2, and the error from the proposed custom cost function was described in Section 4.1.1. Evaluating in terms of training error is important; not only is it desirable to achieve small errors, but analyzing how the training error develops over subsequent epochs indicates how the network is able to learn and (hopefully) increase the generalization ability.

Hit rate

Hit rates was extensively used in [17], but also identified by the authors as a suboptimal performance measure. The hit rate is the simple correct versus incorrect predictions ratio on the total testing set. It is included as an important evaluation measure, mostly because it can analyzed in relation to the profit measurement and thereby identify important trading system properties. Hit rate is only evaluated when analyzing separate matches.

Training Times

Training times varies with regard to the number of input features, input patterns, nodes in the network, training epochs and complexity of applied functions. Training times are a key factor when evaluating the "live" networks, which are constantly being trained as the in-play tennis markets evolves. To long training times will lead to not being able to make predictions in time for it to be useful. Training times are also evaluated when comparing the different cost functions, but not extensively.

6.3 Test 1 - Initial tests

The initial tests used the standard MSE cost function, and formed the foundation for later testing. Different approach strategies were used. Test 1-1 5.2.1 was trained on portions individual matches while testing on the remainder of the match, Test 1-2,3,4,5,6 trained on a set of matches and was tested on the remainder of the matches in the whole data set 5.2.2, and Test 1-7 trained on 80 % of the matches in the data set while tested on the remaining 20 %. When trading, a flat staking strategy was used.

6.3.1 Test 1-1: Cascade Correlation - Separate markets

The basic idea of training individual markets separately was that each tennis match has its own separate characteristics 3.6. It was hoped that in capturing these characteristics,

the trained ANNs could learn an implicit trading pattern, and therefore successfully trade the rest of the market. The training technique used was the Cascade correlation algorithm, elaborated in Section 2.4.3. Ten separate matches was trained and tested. Some of the matches yielded a small profit, with the largest positive yield being 1.986 % in one of the matches. However, the largest negative yield was -7.713 %, and the overall average yield of the ten matches was -1.215 %. Looking at the graphs of the tests, it seems that adding a total of 100 neurons in each of tests were sufficient. The training error stagnates in all of the tests, and further training would most likely not produce lower errors. Studying the overall profit graphs of the training, showing how the overall yield develops during the training, shows that there is no clear pattern of when in the training the profit peaks. This could indicate that that it had not necessarily beneficial to stop the training at earlier epochs. The hit rates of the ANNs prove that in this test, there is no clear relation between high hit rates and positive yields. The average hit rate was 0.4964 and this is in accordance to the negative yield. However, looking at the individual results, there are no correlations between high hit rates and positive results or vice versa. The training times of these tests are crucial, in order to use this approach to trade tennis markets, the training times must be very little. The networks must be trained and ready to run new input patterns within minutes, after the fixed training period is over. Timing of the training showed that the longest training time was 1 minute and 47 seconds and the least training time was 1 minute and 26 seconds. Included in the times is the building of the input patterns from the database. These are acceptable training times and it is thus possible to realize the live runs of new input patterns. The low training times are mostly due to the relatively few training input patterns.

6.3.2 Test 1-2,3,4,5,6

In Tests 1-2,3,4,5,6 5.2.2, the approach was different. The training set consisted of the same ten markets as in the previous test 5.2.1, but was composed as one large training set. Also, training was conducted on the whole matches, not a fixed portion. One of the ideas behind these tests, was to compare training algorithms and different input parameters to explore and possibly identify the most profitable approach to subsequent tests. The trained networks were then traded on a set of 378 markets. This is an unusual approach, most neural network theory suggests using approximately 80 % of the data set for training, and 20 % for testing [3]. However, this was the approach chosen as the goal of these tests were not to use the trained ANNs for trading explicitly, but as an indicator of the most favorable approach with regard to future experiments on larger training sets. The training error graphs of the ANNs are showed in Figure 5.12

Test 1-2: Cascade Correlation - Combined Markets

The training method used was Cascade correlation, with equal input parameters as in Test 1-1 5.2.1. The trained network of Test 1-2 showed a total yield on the traded matches of 0.243899 %. Studying the training graph of the test, it can be argued that

the training has yet to reach its minimum error, as the curve is still decreasing when the training is aborted. However, it is not guaranteed that the yield will increase as a consequence of expanded training, and the net might also suffer from over fitting [17] due to this. Training for more epochs would also increase the training time in an exponential fashion, as the network grows as it is being trained. The training time of this net was approximately 20 minutes. Hit rates were not recorded as it is of little relevance to this part of the discussion.

Test 1-3: Incremental

Test 1-3 5.2.3 used the incremental training algorithm and the Sigmoid activation function in both the hidden and output layer. The hidden layer consisted of 64 neurons, and was trained for a total of 10,000 epochs. The overall yield when trading the net on the 378 markets was 0.506836 %, the highest yield among the initial tests. Looking at the training graph of test, at the end of the training the test had the second smallest overall training error. As in Test 1-2, this net could also possibly benefit from further training, as the graph is still in a downward slope at the abortion of the training. The training time of the net was relatively short, around 4 minutes. This is due to the relative small network structure, which only contains one hidden layer.

Test 1-4: Batch

Test 1-4 5.2.4was trained with the batch training algorithm, which calculates the net error for the whole training set before doing updating the weights. Structurally, the net was expanded with two extra hidden layers. The hidden layers contained 48, 64 and 16 neurons respectively. The learning rate was adjusted to 0.6. Using the trained net to produce trading signals and trade the 378 markets produced a yield of 0.223963 %. The training graph indicates that further training would not produce significantly lower errors, as the curve is almost flat at the abortion of the training at 10,000 epochs. It could also indicate that the training is stuck in a local minimum, and perhaps adjusting the momentum parameter of the training could be beneficial. The training process was recorded to take approximately 5 minutes. Usually, batch training is consider to be faster than incremental training [3], but in this case, due to the more hidden layers and more neurons in the network structure, the training took a little longer than in the previous test.

Test 1-5: R-prop

The R-Prop training algorithm was utilized when training the ANN of Test 1-5 5.2.5. This time the network structure consisted of two hidden layers, with 128 and 96 neurons respectively. The Gaussian activation function was used in the hidden layers, and the standard Sigmoid activation function in the output layer. The network was trained for a total of 10,000 epochs, and produced a yield on the training set of 0.158223 %. The error graph shows that the training error is far below what was produced in the other tests.

At the end of the training, it appears that the training error has reached its minimum, as it starts to increase. The net trained for ca 12 minutes.

Test 1-6: Quick-prop

The ANN of Test 1-6 5.2.6 was trained with the Quick-prop training algorithm. The Sigmoid activation function was used in the three hidden layers, which consisted of 128,96 and 16 neurons. Sigmoid was also the choice of activation function in the output layer. When trading on the test set of 378 markets, the trained net showed a yield of 0.363955 %, the second highest of the compared ANNs in the initial tests. The training error graph shows that the error is in the same range as the other tests, except for Test 1-5, when the training is stopped. The difference quotient of the error at the end of the training is negative, and the net could therefore potentially acquire lower errors if exposed to further training. The training time was roughly 12 minutes, proving it to be a faster training algorithm compared to the R-prop, due to the difference in net structure size.

Of the tests above, Test 1-5 5.2.5 was the one that had the smallest training error. Curiously, Test 1-5 also had the lowest yield of all the tests, and was consequently not chosen as the subject for the next test. Test 1-3 5.2.3 showed the most potential profit wise, and had the second lowest training error at the end of the training. In addition, the graph indicated that it could benefit from more training. Consequently, the ANN of Test 1-3 was selected as subject for a full scale training-set test 5.3.7. In terms of yield, the results of the tests did not show any particular promise. However all the tests produced a positive yield, i.e. did not lose any money. Whether the positive yields are due to the generalization ability of the nets or the result of other factors, can not be decisively asserted. Suppose that the time-series behaves like a random-walk process [17] and it is not possible to find an underlying function, then the consistently positive results must be acclaimed to some other factor. Combining the prices of the binary market onto one ladder, described in Section 3.2.2, may be one such factor. By using this technique, the trader has the potential to exploit inefficiencies in the markets, e.g. obtain odds that over time beats the book-percentage [17]. To find out if that is the case, a detailed analysis has to be carried out, and is not in scope of this project. The motivation behind using the technique is to always acquire the best price. If by utilizing this method to calculate the optimal prices when trading is sufficient to produce long term profit, then that is nothing but good news. This discussion however, is interested in analyzing the properties of the neural network.

6.3.3 Test 1-7: Full Scale Incremental

Test 1-7 5.3.7 was the extended test on the most promising candidate from the initial tests 5.2.2. The parameters and network structure from Test 1-3 5.2.3 was used to train the net on approximately 80 % of the data set, or precisely 320 markets. It was then tested, also while during training, on the remaining 68 markets in the set. The net was trained for 15,000 epochs, in hope of finding the minimum error. For every 150'th epoch

the network trained up to that point was used to trade the 68 previously unseen markets to see how the yield evolved during training. The training graph is seen in Figure 5.13. The training time was, including the trading runs which are run while training, approximately 7 hours.

Although the training error constantly diminishes, it does not seem help increase the result of trading the test set. The final training error was 2.57979e-3, an increase from 1.34998e-3 in Test 1-3. The increased error is only to be expected when training with larger training sets. Although the testing set yield is higher at earlier training epochs, this seems a bit arbitrary, and it probably can not be claimed that earlier termination of the training would produce higher yields on new test sets. The final yield when the training is interrupted is 0.2908 %. The result was disappointing and combined with the amount of time it took to train, this particular approach did not receive more attention.

6.4 Test 2 - Custom cost function tests

The second part of the testing utilized the custom cost function specified in 4.1.1, and not the standard MSE. A dynamic trading strategy was used, with the individual stakes determined by the network output trading signal. Test 2-1 5.3.1 used the same approach as that of Test 1-1 5.2.1, and were trained on a fixed portion of ten individual markets separately. Test 2-2,3,4,5,6 5.3.1 was trained with the same settings as the equivalent initial tests 5.2.2, with the difference being cost function and staking strategy. Test 2-7 5.3.7 was a full scale training test on the best of the candidate network structures.

6.4.1 Test 2-1: Cascade Correlation - Separate Markets

Test 2-1 was trained on the same ten individual markets as in the first initial test, for comparison purposes. All configuration settings, including training method, input parameters and data were identical. All the ten markets were trained and tested, while training. The training and trading graphs can be studied in detail in Appendix B. Looking first at the trading results, there was a mixed outcome, with a total of 6 markets with positive yield and 4 markets with negative yield. The biggest yield came from trading the market Carlos Berlocq vs. Luis Horna, i.e. the last of the ten markets. This was also the case in the corresponding test in Test 1-1. However, the yield achieved in this test was bigger, with the highest yield being 2.780 %. The poorest result of the ten traded markets achieved a negative yield of -2.87 %, but in a different market than that Test 1-1. Interestingly, the market which produced the biggest trading loss in Test 1-1, achieved a profit in terms of yield of 1.3 % in the current test. The average yield of the ten markets was 0.467 %. Studying the P/L graphs of the training, it seems that they are much more volatile than in Test 1-1. This could be the result of the cost function tries to aggressively targets larger fluctuations, and combined with the staking strategy produces this "jagged" up and down effect. It could be argued that aborting the training at an earlier stage, could perhaps produce a bigger overall yield. Many of

the P/L graphs reaches its culminating point at around 20-40 added neurons. However it is hard to make this general statement. Look for instance at the P/L graph of the 7th test, i.e. Stefan Koubek vs. Olivier Rochus which have an extreme loss at around 20-40 neurons. This would consequently neutralize all the minor profits of the other tests at this point in training. The average training error is larger in this test compared to Test 1-1. This indicates something about the relation of training error and overall training objective. The hit rates of the ten traded markets ranges from 0.43 to 0.62. The average hit rate is 0.4942. As in Test 1-1, the training times were acceptable in regard to the "live" training usage. The longest training time was 1 minute and 19 seconds and the least training time was 1 minute and 1 second.

To see if less training could give higher yields, the ten markets were trained again, with only 30 added neurons. The average yield was -0.6341 %, and as assumed above, terminating training earlier did not achieve higher yields.

6.4.2 Test 2-2,3,4,5,6

The goal of these tests was not only to identify which training method that could be most profitable together with the new cost function, but also give grounds for the analysis of comparing the different cost functions. The training set was the same ten markets as before combined, and the test set constituted of the remaining 378 markets. The training graphs of the tests are shown in Figure 5.24, and the results are listed in Table 5.9. The training times of these tests took generally longer than in the corresponding initial tests, probably due to the increased complexity of the cost function. This will not be discussed further.

Test 2-2: Cascade Correlation - Combined Markets

Test 2-2 was trained with the same credentials as Test 1-2. Trading simulations produced a yield of 0.3788 %. This is a little increase, but not significantly. Looking at the training graph, the training error is much lower than in the other tests trained with the same cost function. At the end of the graph, it culminates and increases before it decreases again. This could indicate that training error has come to a conclusion and the further training might not lower the error any more. However, the training errors of this test and the subsequent tests can not be directly compared. This is due the properties of the Cascade-correlation algorithm which does not train for a fixed number of epochs, but a fixed number of added neurons. Adding 100 neurons to this network correspond to training the network, in this case, for approximately 28,000 epochs. The error can thus only be compared to that of Test 1-1. The train error at the abortion of training was 2.7664e-6. This is much lower than Test 1-1s error of 3.52421e-3. This noticeable decrease must arise from the properties of the custom cost function.

Test 2-3: Incremental

The most promising of the networks in the initial tests 5.2.3, was in this test trained with the custom cost function. The dynamic staking strategy was utilized. The trained network produced a yield of 1.8436 %. This is higher than the corresponding MSE trained test. The overall training error of the net is 1.7615013e-3 , a small increase compared to the 1.34998e-3 of Test 1-3. Studying the train error graph reveals that the error is on a downward slope at the termination of the training, and continued training might lead to improvements. However this improvement could possible only relate to the actual error, and not the yield. Only further training and testing can conclude this.

Test 2-4: Batch

Test 2-4 yielded the most promising result among the candidates. A result of 3.17309 % yield seems promising. The corresponding initial test, only showed a yield of 0.22 %. This was the highest yield of the candidate nets trained with the custom cost function, and was consequently selected as subject to further testing. The error graph reveals that the network error develops almost identical to that of Test 1-4. However, the error of this test was 3.7411132e-3, a decrease from the 5.61425e-3 of Test 1-4.

Test 2-5: R-prop

Training the net with the R-Prop training algorithm in Test 1-5 produced the lowest train error of the candidates in the initial tests, 1.11787e-4 . Training the same network with the custom cost function also produced a relatively low training error, 3.511778e-4. This was, if excluding the cascade-correlation test, the lowest of the networks trained with the new cost function. The training graph proves that the network could potentially show lower training errors if trained further. The yield was 0.815817, the second lowest yield of the candidates, but is still a substantial increase from Test 1-5.

Test 2-6: Quick-prop

The last of the tests was trained with the Quick-prop algorithm. Also this test showed some potential with the second highest yield of the candidates, 1.84849 %. This is a clear improvement from Test 1-6. The error at the end of the training was 3.5292108e-3. Comparing it to the initial test with 4.4432e-3, this is a minor decrease. The error graph shows that the error is descending when the training is interrupted.

Training with the same network structures, parameters and data sets as in the initial tests, with a custom designed cost function increased the profit when doing the trading simulations in all the cases. The most notable increase was that of Test 2-4 vs. Test 1-4. These tests were the only ones trained with a lower learning rate, i.e. 0.6, and a linear activation function in the output layer. While not producing satisfying results in the test trained with the standard squared cost function, it showed promise with the custom cost function. This could be an indication that the custom cost function responds better to

less aggressive learning, due to the functions' already aggressive nature. Batch learning calculates the error for the whole training set, before starting the weight updating. This kind of training seems to better fit the new cost function, and was therefore selected as subject of a full scale training set test 5.3.7. On an overall note of the training error, it generally can not be stated that lower training errors necessarily gives better yields. Also, the new cost function does not consistently produce higher nor lower training errors.

6.4.3 Test 2-7: Full Scale Batch

Test 2-7 was trained with the settings from Test 2-4, on 320 markets, and then ran trading simulations on the remaining 68 markets. This was identical to the data set of test 1-7. Due to the properties of the training graph in Test 2-4, and the time consuming process of training on large data sets, the network was trained for (only) 10,000 epochs. The simulation trading runs were also executed while training, to get a detailed picture of how the yield % evolved. The training graph is shown in Figure 5.25

The training error graph reveals that the net is able to learn, i.e. constantly reduce the error according to the cost function utilized. However, the desired effect is not achieved; the yield graph of the test set does not constantly increase as the network is being trained. At the end of the training, the final yield is 0.7723 %. Compared to Test 1-7, this is more than twice as profitable, but still not very remarkable. Although not directly comparable to Test 1-7, due to the difference in the ANNs being trained, it seems that all the neural nets trained in the custom cost function tests, achieved higher profits than the one trained in the initial tests. As there is one other factor that separates the two set of tests, the author is somewhat cautious when reaching a conclusion. The staking strategy in the second tests is dynamic, while the staking in the first tests is static, or "flat". Because one of the ideas behind the custom cost function was to reward bigger fluctuation predictions more small ones, the result is that it more often overshoots the trading signals. The trading signal is then used to calculate the stake, and this result in bigger stakes for bigger signals. However, the hit-rate from Test 2-1 showed that it predicts wrong more often than not, i.e. 0.4942 on average, so the staking strategy may also work against its purpose. Bigger stakes when the trading signal is wrongly classified, and generally bigger magnitudes, means bigger losses. Although this requires further analysis, it is (for now) assumed that as long as the hit-rate is around 0.5, over time, the two staking strategies have the same impact.

6.4.4 Test 2-8: Full Scale Cascade Correlation - Seperate Markets

Test 2-8 used the same approach and configurations as Test 2-1, this time training and trading on each of the 378 markets not already trained on. The discussion of training error is not included here, as it does follow the exactly the same patterns as the corresponding small scale test. The yield % ended up being 0.21109 %, a little lower than the 0.467 % achieved when training and trading on only ten matches. This test implemented

a modified version of the Kelly staking strategy as described in Section 3.3.3. As the probability of winning the trade is not a output produced by the networks, the P factor in the Kelly formula consisted of the predicted magnitude of the trading signal (|y|). The initial bank was set to 1000. Not all the 378 markets were traded, as the Kelly formula determines the staking. Also, some markets were not trained and traded as they did not last the required 60 minutes used for training. All in all 317 markets were successfully trained and traded. The bank ended up being 1522.28.

This indicates, that even though the actual prediction strategy is not superior in terms of enormous hit rates or yields, as long the yield is positive, in combination with the appropriate strategies, the use of ANNs for Betfair trading prediction can be profitable. However, is cautious when making this statement. These are historical tests, and real time testing needs to be carried out to in order to decisively conclude with the above statement.

6.5 General Issues

Some of the overall challenges in the above training and testing are discussed here.

In using the sliding window technique, the problem of selecting the "correct" window size is a challenge that most often has to be solved experimentally rather than analytically [15]. In this problem several different window sizes and time-slice lengths were explored. As the supplied data were guaranteed to by sampled at least every 30th second 4.3.1, this of course was the minimum time-slice length. A Tennis match only lasts for around a couple of hours, and because of this, the time-slice size can not be set too big either, as it would then reduce the trading prediction task into betting. The number of slices to include also presents a problem. Experiments in [17] and further tests in this project showed that using 8 slices were not necessarily best, but neither better nor worse than other configurations. Consequently, the sliding window technique used 8 time-slices each with the duration of one minute - and was "locked" at this setting during all the tests. This was of course so that the different nets could be directly comparable. Whether this sliding windows configuration is optimal for the problem at hand is very difficult to determine, and only extensive testing could prove this.

Although a decent attempt of constructing an optimal cost function took place, it could be argued that the proposed function does not hold the relevant properties. It would be optimal to train the networks directly on the profit function, however, the optimal profit function is (part of) the prediction task! With this in mind, some properties of well known trading theory were identified (e.g. big fluctuations predictions favorable, dynamic staking etc.) and attempted implemented in the cost function. It is not straight forward to identify and capture the underlying problem in a single function only having to explicit inputs.

On a final note of discussion, the random-walk [17] properties are revisited. If in fact, the in-play tennis markets on Betfair behave like a random-walk process, it can be stated that there exists no underlying function, and it can not be predicted. It does seem like, if not completely, that it has some of these properties. In the full scale tests, the networks were presented with over 45,000 input patterns. It could be argued that the many of these input patterns weighs up for each other, thus resetting the corresponding property already learned by the net. This was also identified in [17] and was one of the reasons behind the approach of training and trading individual markets by themselves.

Chapter 7

Conclusion and Future Work

The goal of the thesis was stated in the introduction 1.5:

The goal of this thesis is to explore how different composed artificial neural networks, can be used to predict in-play tennis odds market time series on Betfair. The thesis seek to identify a relevant problem specific cost function which can be used for ANN training, and analyze how implementing the problem specific properties directly into the training process compares to standard statistical cost functions with regard to training error and potential trading profitability.

7.1 Conclusion

In this thesis I have studied, analyzed and implemented a solution for using artificial neural networks for prediction of in-play tennis match odds markets on Betfair. The overall prediction task was concentrated on maximizing potential profit, rather than just minimizing some standard error. The properties of odds trading were studied, and on the basis of this, a new cost function suitable to the underlying problem was proposed. The new cost function tried to capture some of the problem specific characteristics, and aimed to maximize the return of each trade. Training with both the new and the standard cost function was conducted, together with a range of training algorithms, ANN structures and parameters.

The approach to the testing was two-fold. The first approach concentrated on training and testing on individual markets, and the other on combining more than one market in the training set. The first approach were intended to train during live matches, and then produce trading signals to trade the remainder of the match. The training times showed that this was possible, however the achieved yield proved that it not necessarily is profitable. In the small scale tests, raining with the standard cost function gave an overall negative yield, training with the custom cost function gave a small positive yield. Testing this approach together with the custom cost function and a specific staking strategy on the full data set showed that it was potentially profitable. However, further

testing, including real time testing needs to be executed in order to definitively conclude this. The approach in which the net were trained on sets of markets also explored which training algorithms, net structures and parameters were most suitable for the two different cost functions and the underlying problem. The small and full scale tests showed that all the nets produced a small overall profit, but the custom cost function trained nets consistently outperformed the others. It also showed that lower training error does not automatically lead to better results, and that this is related to the nature of the problem it is trying to predict.

Going back to the original motivation and goals behind this thesis, some of the questions have received new knowledge while other remains unanswered. Is it possible to use artificial neural networks, trained with problem specific cost functions, and in combination with well known trading strategies and methods to profitably trade the in-play tennis odds markets on Betfair? The immediate answer is yes, however the contemplated one is maybe. As discussed in the preceding chapter 6 it can be difficult to determine if the positive results can be acclaimed to the properties of the neural nets alone. There may be other unidentified factors that are influencing the results, only further work and testing, including validating "live" tests could definitively prove this.

An attempt of finding the optimal net structure, training algorithms and parameters, and data representation took place. No definite superior configurations were found. These settings seem to be not only specific to whatever underlying representation of the problem was implemented, but also the choice of cost functions. The two different cost function training schemes did not produce each of its best results with the same settings. No conclusion has been reached in this regard, and more testing needs to be conducted. Other network structures relevant to time series prediction, such as Time-delay-neural networks (TDNN) or recurrent nets such as Jordan, Elman or Hopfield networks must also be experimented with. The chosen ANN library did not support these types of nets.

The results showed that, in all the tests, the nets that were trained with the new cost function provided a higher yield than the ones trained with the standard cost function. This proves that the custom designed cost function holds some properties that "fit" the underlying problem structure with more accuracy than the standard cost function. However, the proposed cost function may not be optimal for the odds trading task. Finding the relevant optimal cost function must receive a lot more attention, including detailed mathematical analysis of both the function and its derivative. This must be based on a complete, in-depth understanding of the underlying problem.

Custom designed cost functions can potentially capture the real world problem structures in more detail than the standard cost function methods. Using squared error measures in neural network training may be motivated by a number of factors, one of which is analytically simplicity. However, it may not always lead to the optimal results, and this is highly dependant of the problem structure and implemented data represen-

tation. Identifying a suitable cost function is not trivial in most cases, and in some cases impossible. Additionally, the identified cost function may not always be suited for backpropagation training due to the nature of its derivative.

The low yields of all the tests means that you are better off putting your money in the bank, risk free. But with over a thousand ATP tennis matches a year, most of them, if not all present as Betfair markets, the yearly returns of using ANNs for trading could be well worth it. Test 2-8 proved that, together with the correct money management strategy, yearly returns could potentially reach over 50 %. However, the element of risk is not yet determined and is not only a result of the underlying prediction strategy, but also staking and money management strategies. Also, there is no guarantee that the training and trading future markets will achieve similar yields as the one in Test 2-8. Specially since this yield is as low as it is, 0.21109 %, the margins of error small and it could potentially only require minor differences before future yields becomes negative. A negative yield would result in negative returns, no matter what money management or risk strategy is implemented. Such differences may be unidentified factors, that previously have been unaccounted for. These factors may be identified when validating the results with runs on real live markets. The commission [17] charged by Betfair is one such factor, although the awareness of this factor has been present throughout this work. This factor has been left out of the results because of its variable nature and for sake of simplicity of the analysis. However the impact of this overhead cost of trading is very real and results in lower profits among all trading simulations. In the case of Test 2-8, when applying the most aggressive commission scheme, the returns after commission are 32 %.

On a final note, almost all the tests resulted in small trading profits during trading simulation testing. However the returns are too meager to receive any particular notice. That being said, similar to stock prediction, the area of odds prediction is a very complex task, and results "off the chart" are not to be expected. The important factor is the positive expectation [9] the trained networks give, and any trading system is seeking to find. Successfully using ANNs for prediction tasks does not only require sound knowledge of artificial neural networks, but also a deep understanding of the problem of which it is set to solve. The search for finding the optimal training cost function and ANN configurations to achieve even higher trading returns continues.

7.2 Future Work

Thorough and validating "live" testing must be executed in order to definitively conclude with the above positive results. Such testing would involve implementing an agent which, on basis of the trading signals produced by the nets, actually carried out the trading on Betfair. This implies designing an agent which through the Betfair API executed the trading in exactly the same way as the trading simulations on the historical data in this thesis were executed.

Unexplored neural network types could be experimented with. The network types mentioned earlier, including Jordan, Elman or Hopfield nets should be investigated. As this work is concentrated around the open source Fast Artificial Neural Network (FANN) library, and the current implementation is specifically targeted towards this library, the above network types must first be implemented as part of FANN. Alternatively, the current implementation must be re-implemented to work with another ANN library that has the desired functionality. Of the two, the first approach seems the most convenient in terms of time and effort required.

In the tests conducted in this thesis, the single prediction output from the nets was a short-term predictor, i.e. it is only concerned with what goes on in the following time-slice. Defining other prediction outputs, including factors which generally define what will happen to the market over bigger time perspectives should be investigated. Also, experimentation with more than one network output could be carried out, e.g. with one classifying market direction and one output defining either magnitude or confidence-level.

Finally, although not in the scope of the current work, comparisons to other prediction strategies, including both machine learning techniques and statistical approaches to time-series prediction should be examined. A broader knowledge of the advantages/disadvantages such techniques would provide, could also be exploited in combination with the current work - to produce the optimal prediction strategy.

Bibliography

[1] Michael Baulch. Using machine learning to predict the results of sporting matches. 2001.

[2] Betfair. Say it ain't so - the shadow over tennis: http://betting.betfair.com/tennis/general/say-it-aint-so-the-shadow-over-090807.html.

[3] Robert Callan. *The Essence of Neural Networks*. 1999.

[4] Buntin P. She L. Sutjahjo S. Sommer C. Neely D. Chen, H. Expert prediction, symbolic learning, and neural networks: An experiment on greyhound racing. 1992.

[5] Sven F. Crone. Training artificial neural networks for time series prediction using asymmetric cost functions. 2002.

[6] G. Dorffner, E. Leitgeb, and H. Koller. Toward improving exercise ecg for detecting ischemic heart disease with recurrent and feedforward neural nets, 1994.

[7] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, Denver 1989, 1990. Morgan Kaufmann, San Mateo.

[8] Scott E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report Computer Science Technical Report, 1988.

[9] Curtis M. Faith. *The way of the turtle*. 2007.

[10] T. Falas and A.-G. Stafylopatis. The impact of the error function selection in neural network-based classifiers. *Neural Networks, 1999. IJCNN '99. International Joint Conference on*, 3:1799–1804 vol.3, 1999.

[11] Christian Igel and Michael Hüsken. Improving the Rprop learning algorithm, 2000.

[12] The Independent. Internet betting scandal sparks atp investigation: http://www.independent.co.uk/news/business/news/internet-betting-scandal-sparks-atp-investigation-583157.html.

[13] Kwasny S.C. Kalman, B.L. A superior error function for training neural networks. 1991.

[14] Dr Alan McCabe. An artificially intelligent sports tipper. 2002.

[15] Dieter Merkl. Partially recurrent neural networks in stock forecasting.

[16] Steffen Nissen. Large scale reinforcement learning using q-sarsa() and cascading neural networks. 2007.

[17] Ø yvind Ø vregård and Henrik Vatnar. Predicting betting exchange markets using neural networks and genetic programming. 2007.

[18] Dimitri Pissarenko. Neural networks for financial time series predicition: Overview over recent research. 2001-2002.

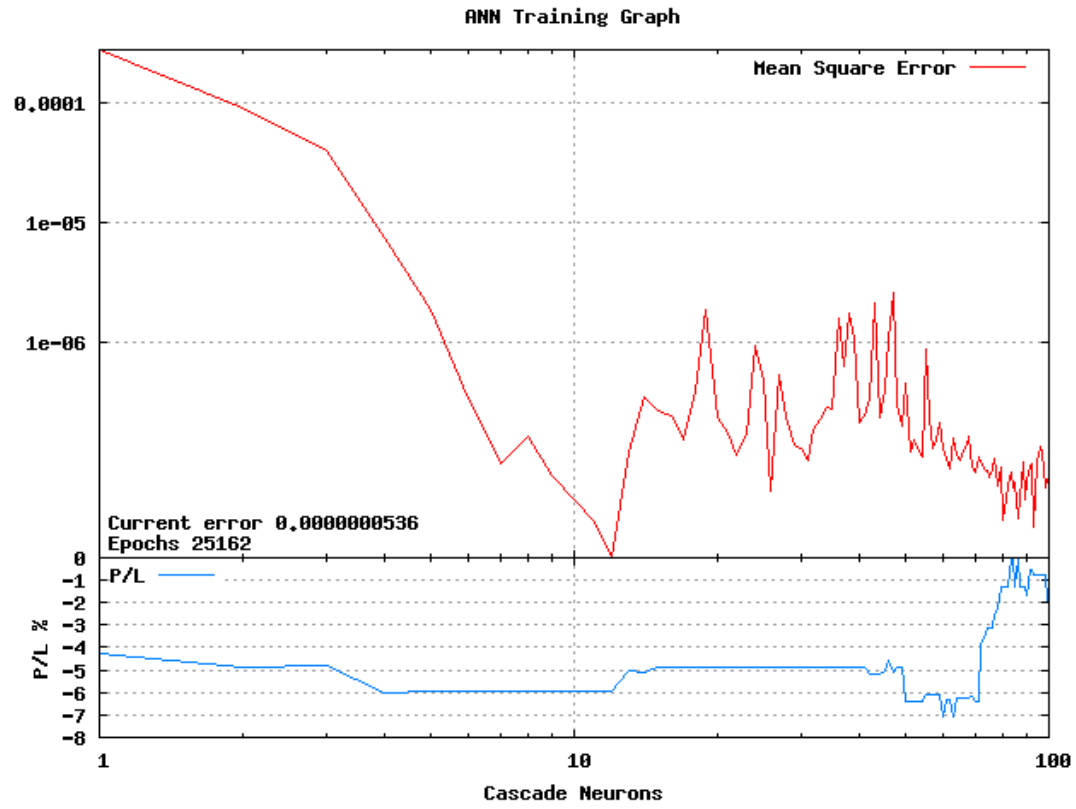
[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. pages 673–695, 1988.

[20] BBC Sports. Davydenko faces betting inquiry: http://news.bbc.co.uk/sport2/hi/tennis/6928635.stm.

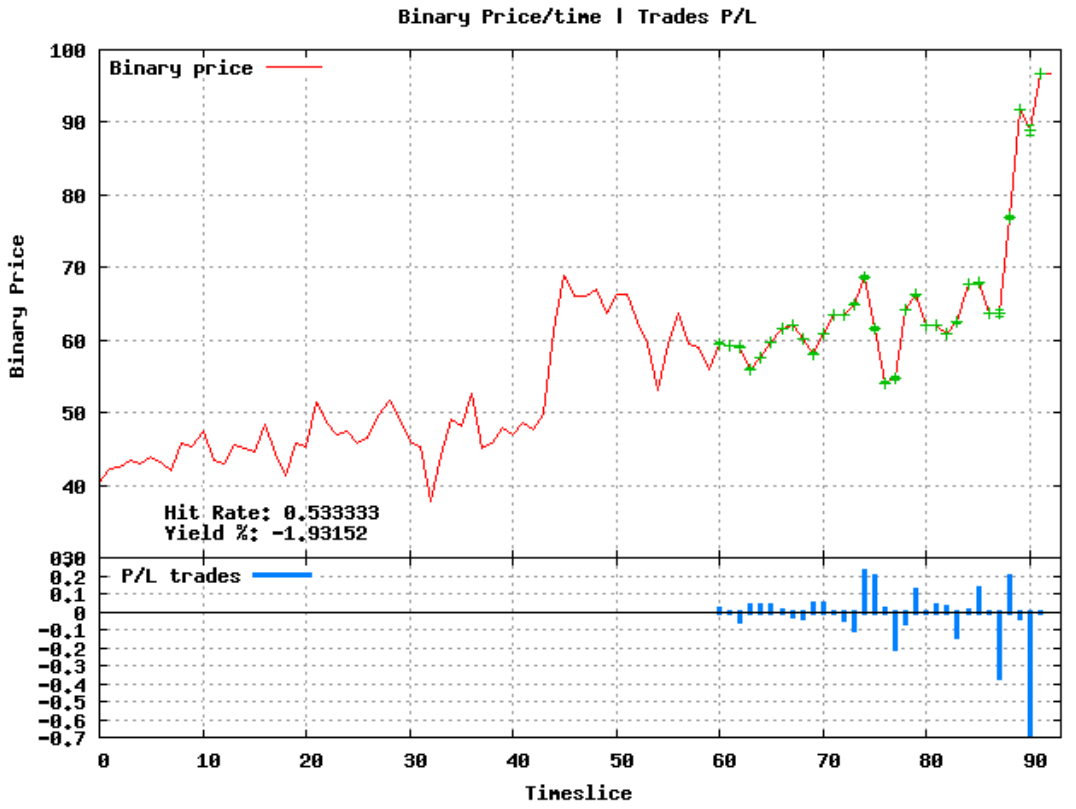
[21] H White. Artificial neural networks: Approximation and learning theory. 1992.

Appendix A

TEST 1-1: Initital Testing Graphs

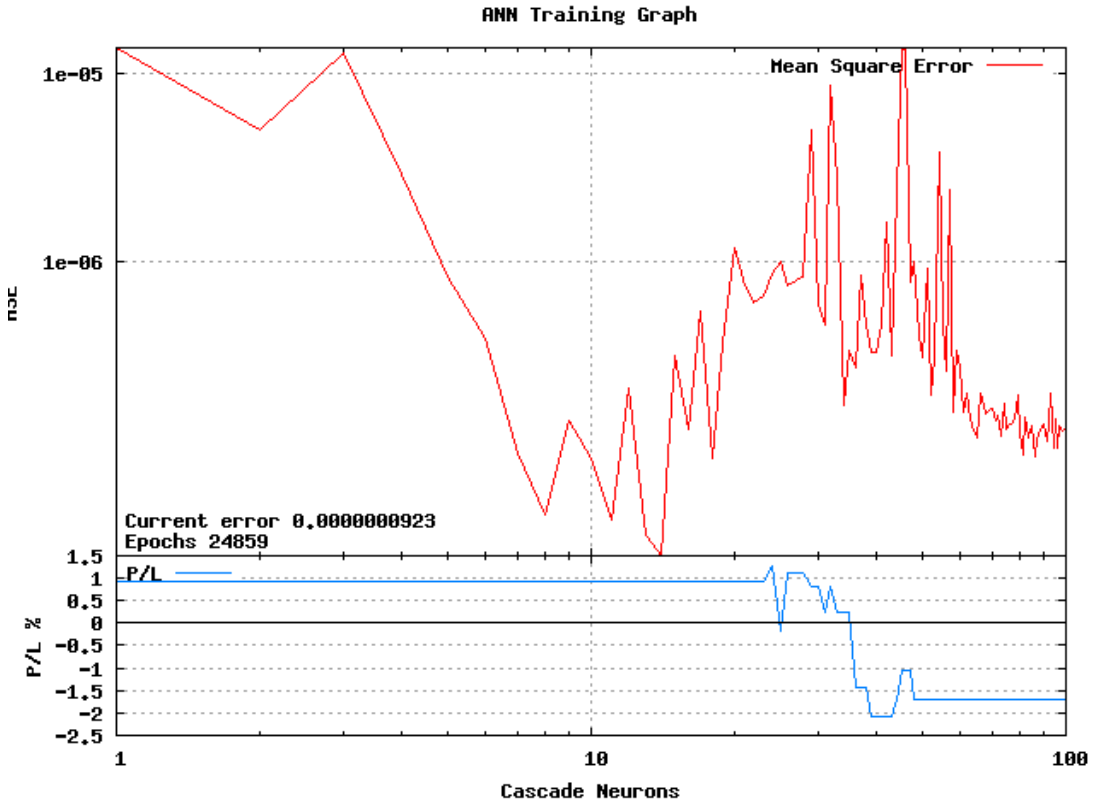


(a) Training Data

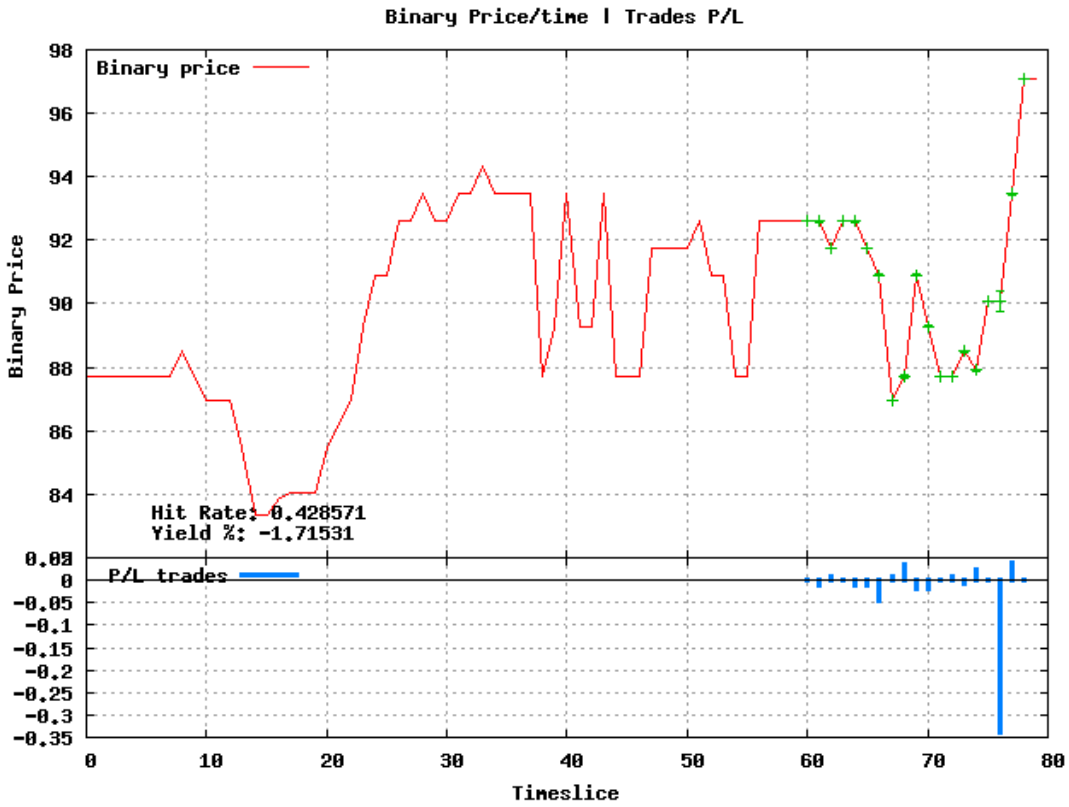


(b) Trading Data

Figure A.1: Test 1-1: Amer Delic v Jurgen Melzer

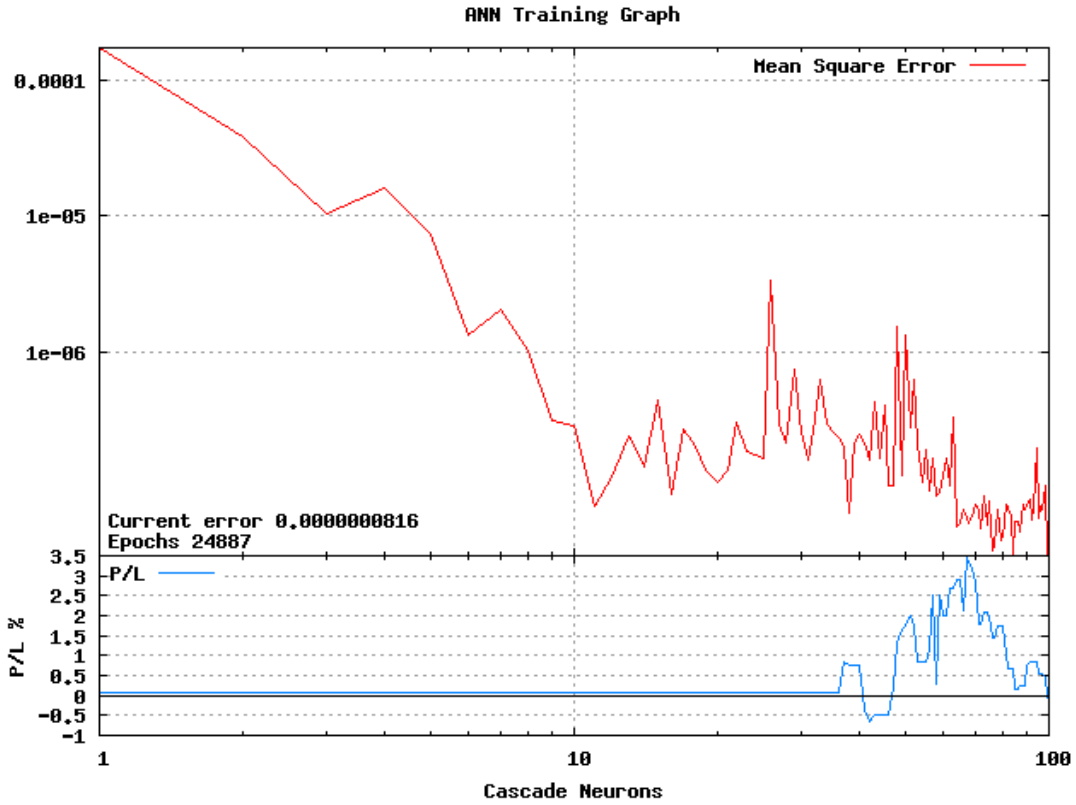


(a) Training Data

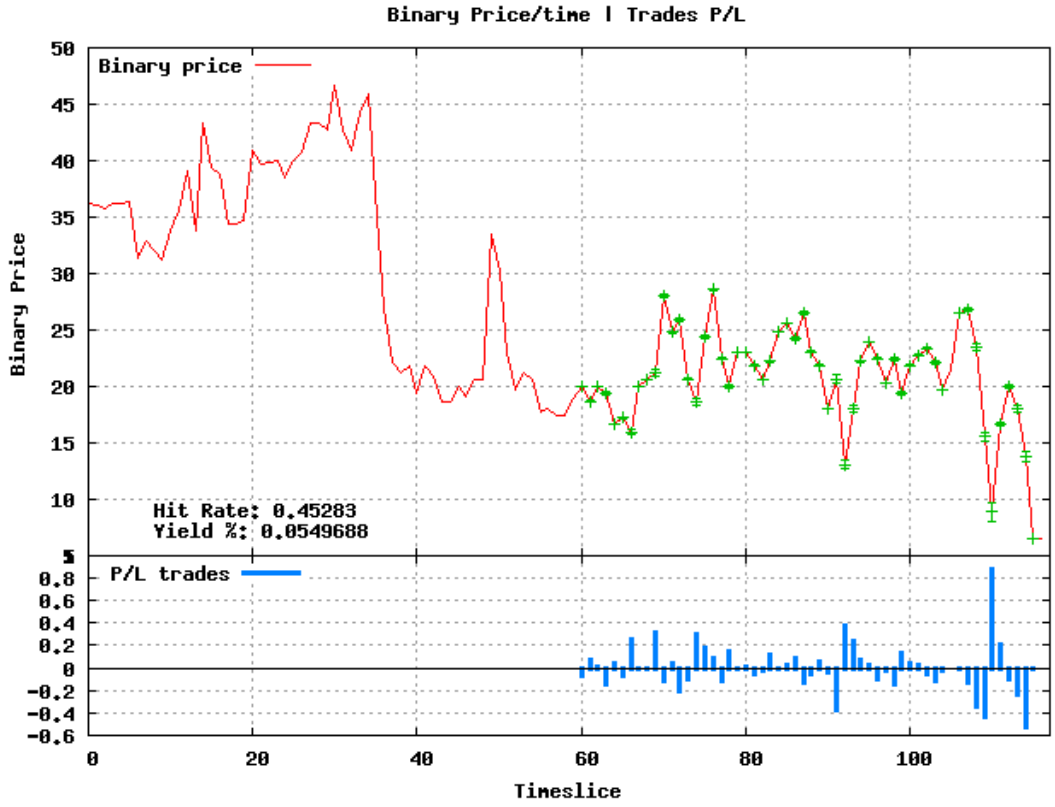


(b) Trading Data

Figure A.2: Test 1-1: Andy Roddick v Gilles Muller

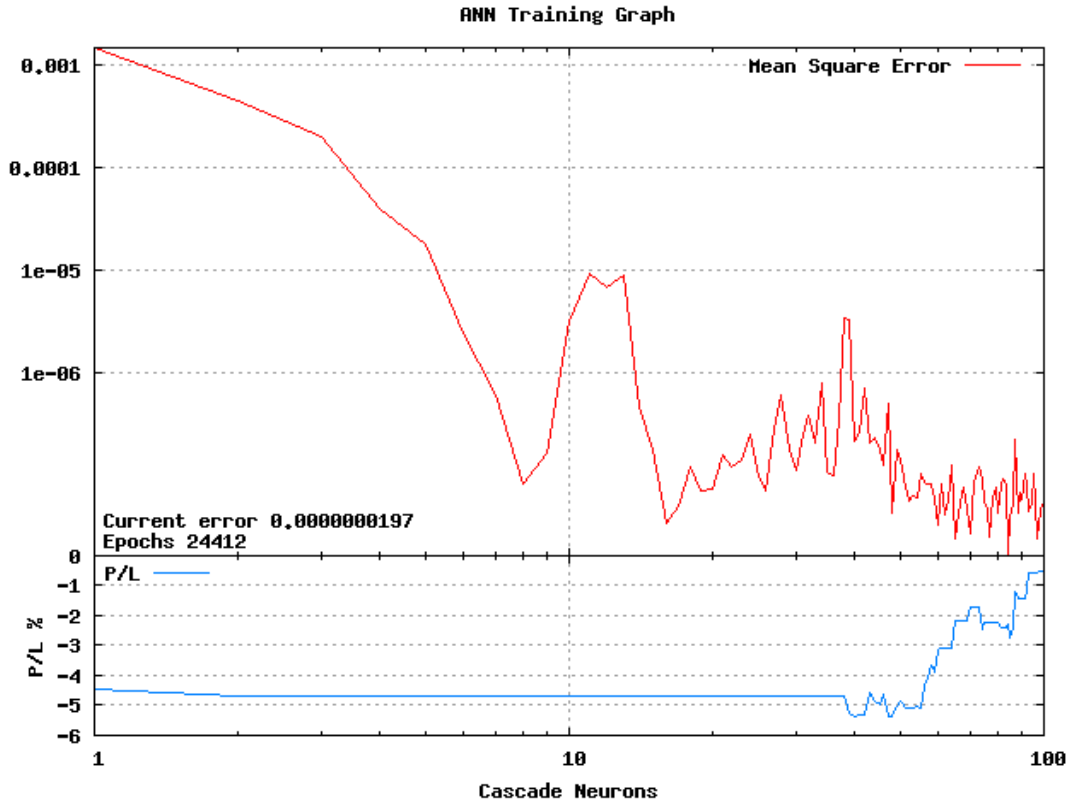


(a) Training Data

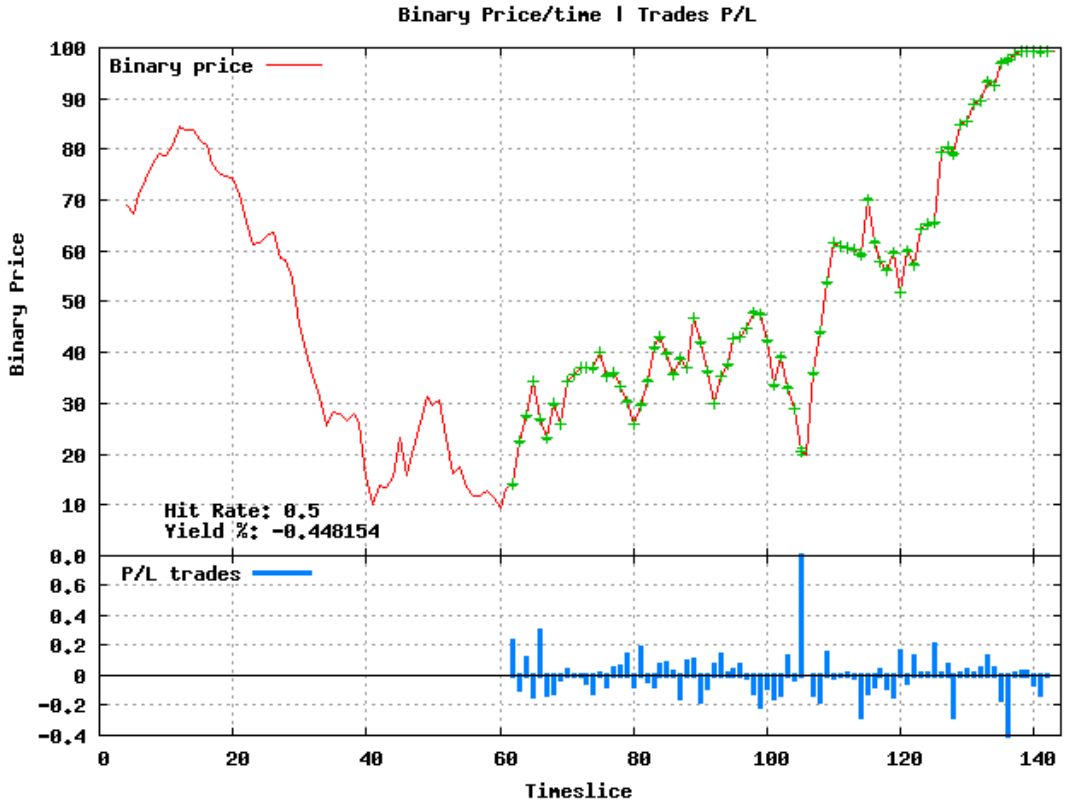


(b) Trading Data

Figure A.3: Test 1-1: Marin Cilic v Mikhail Youzhny

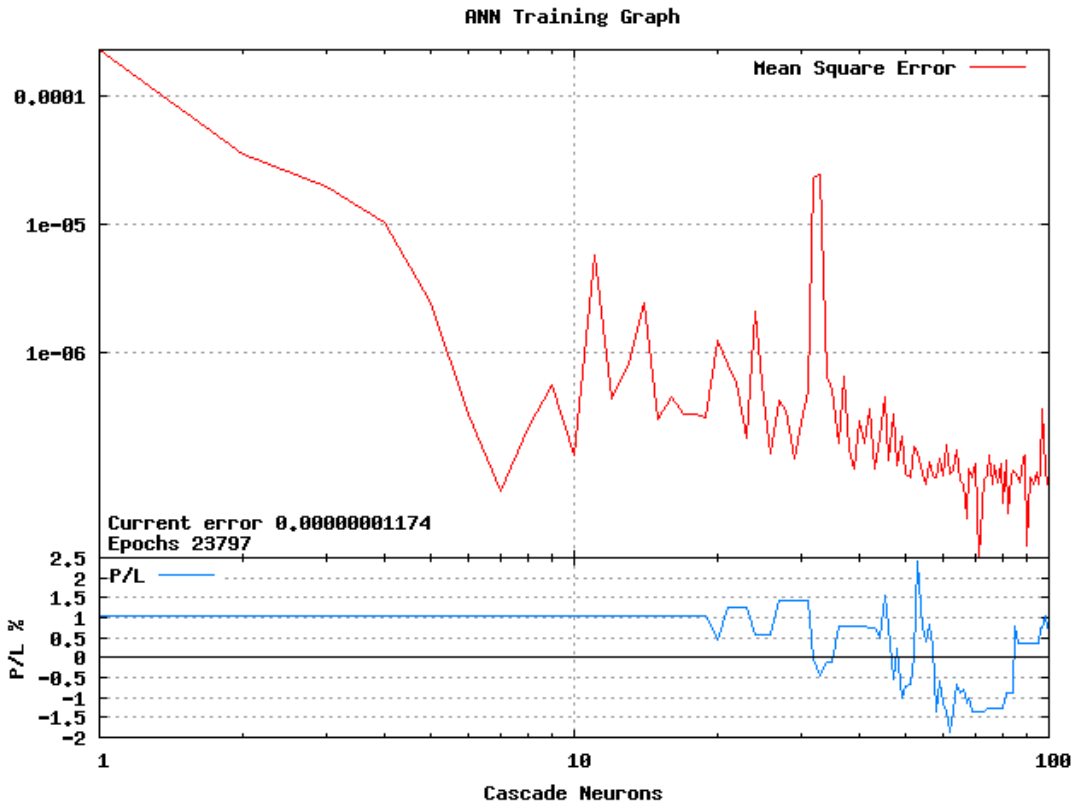


(a) Training Data

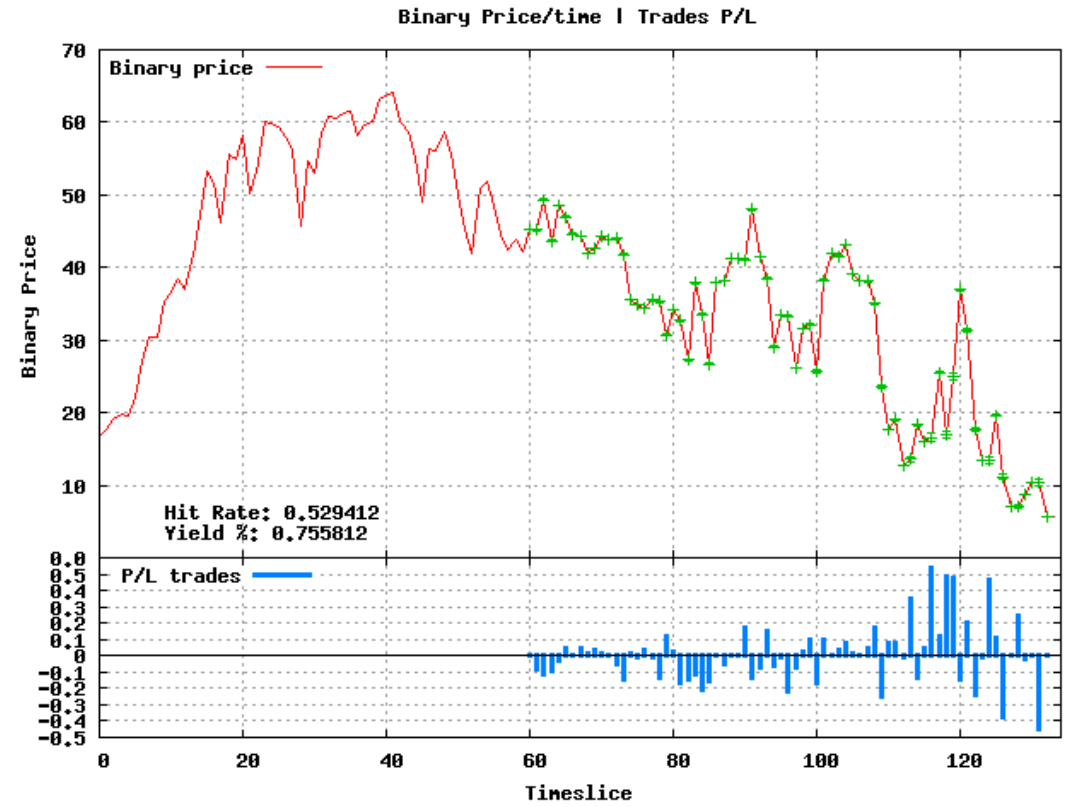


(b) Trading Data

Figure A.4: Test 1-1: Andy Murray v Stanislas Wawrinka

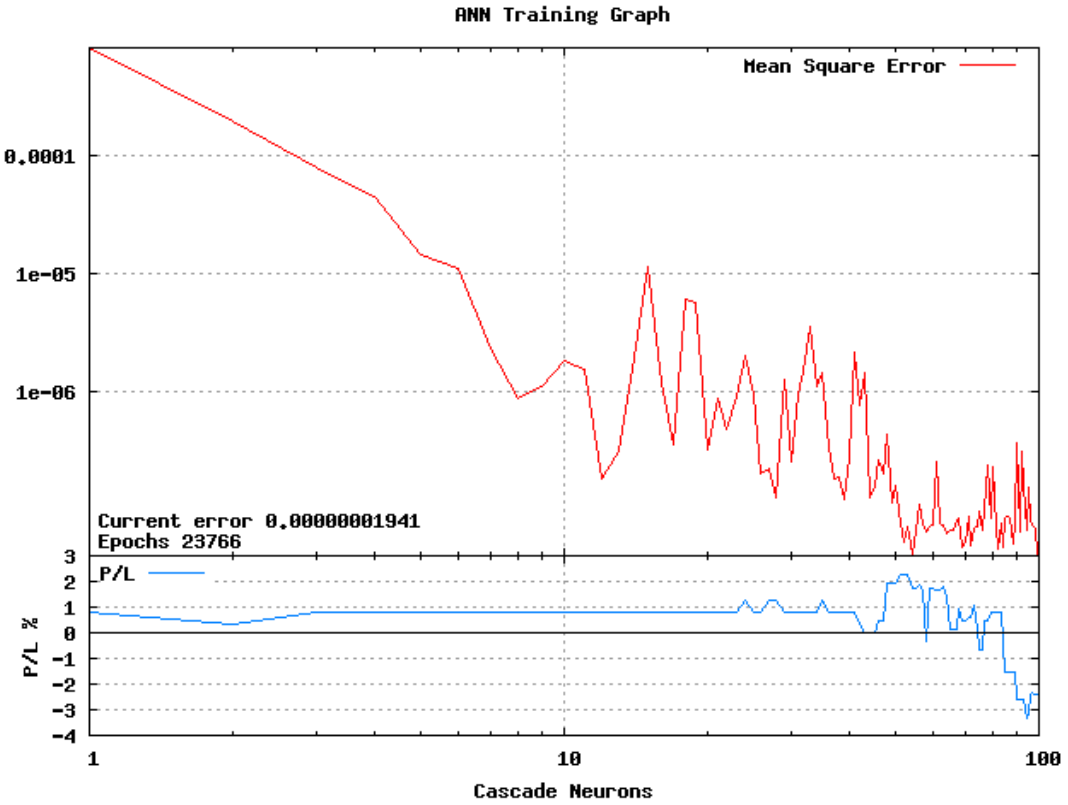


(a) Training Data

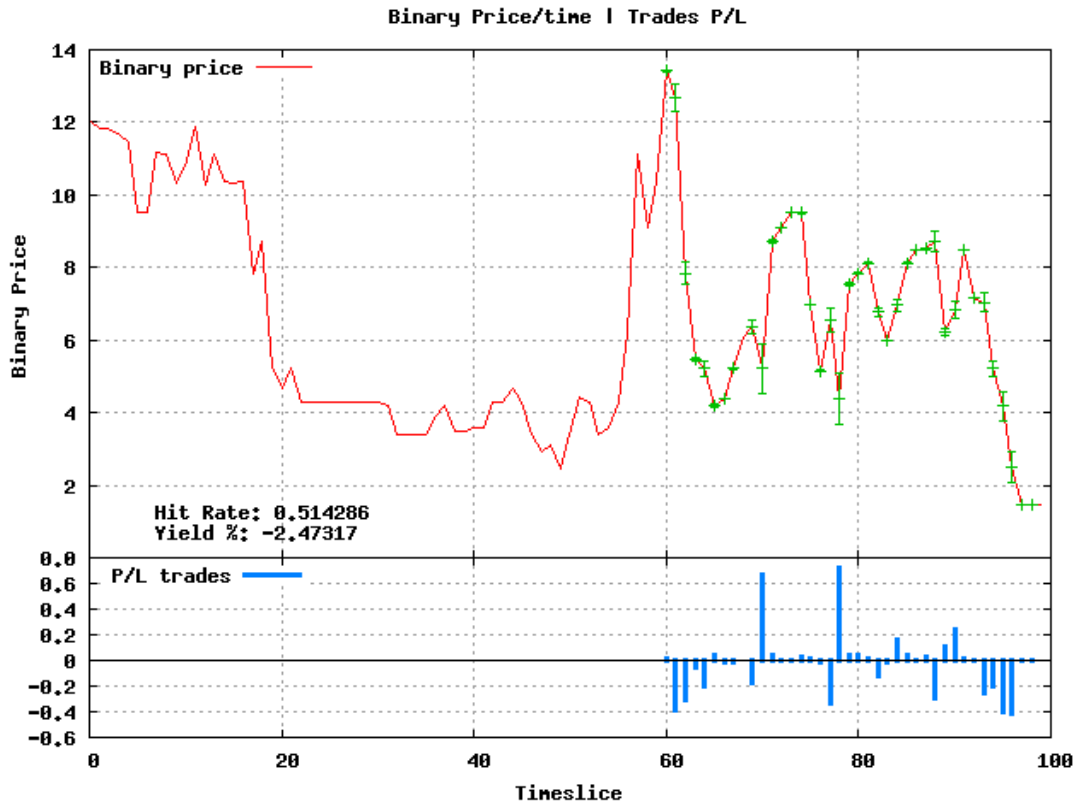


(b) Trading Data

Figure A.5: Test 1-1: Philip Kohlschreiber v Rafael Nadal

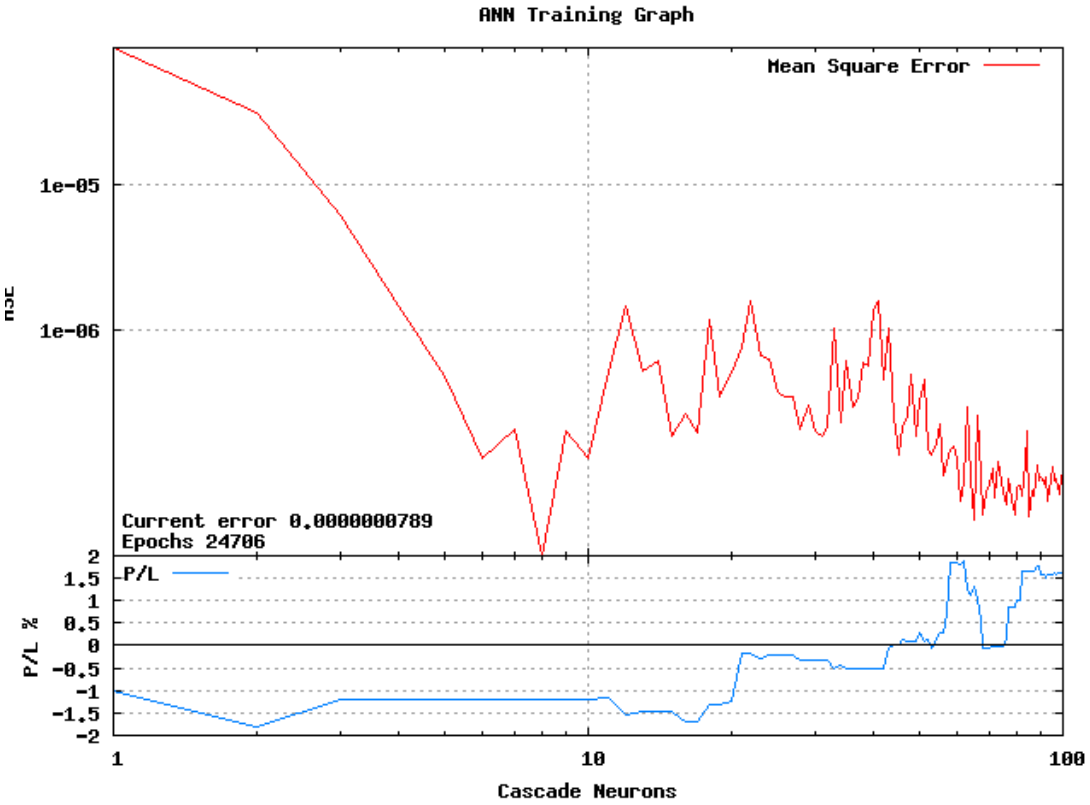


(a) Training Data

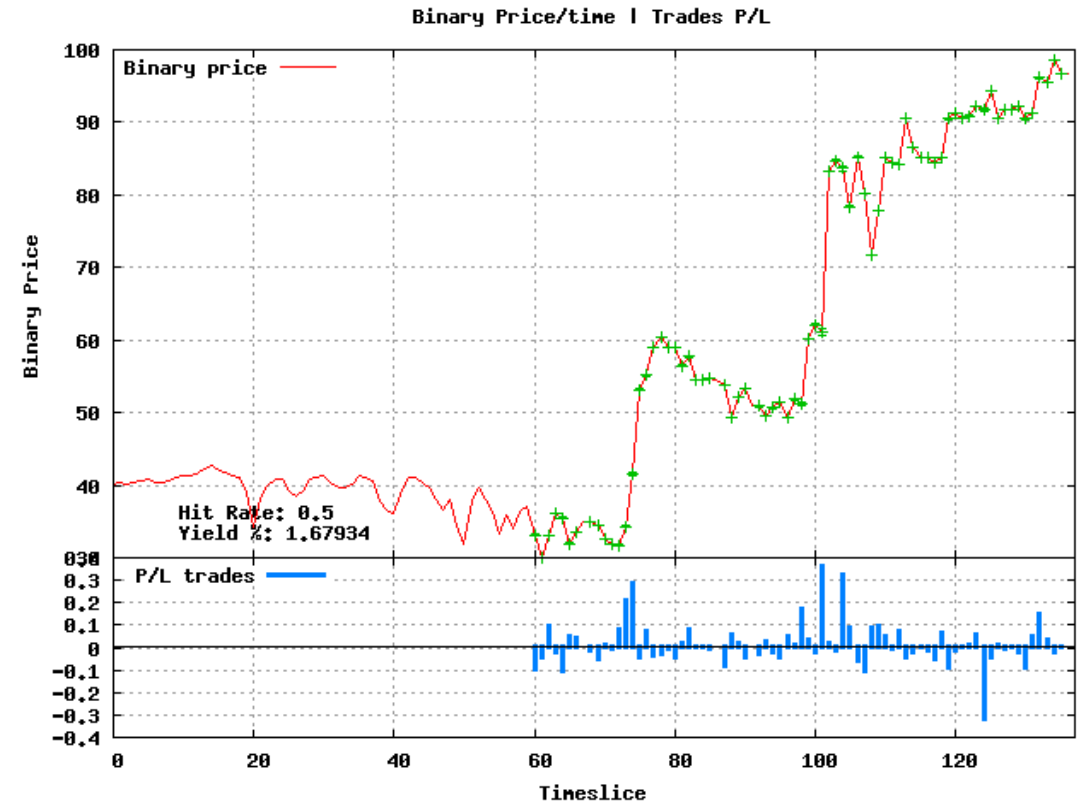


(b) Trading Data

Figure A.6: Test 1-1: Fabrice Santoro v Novak Djokovic

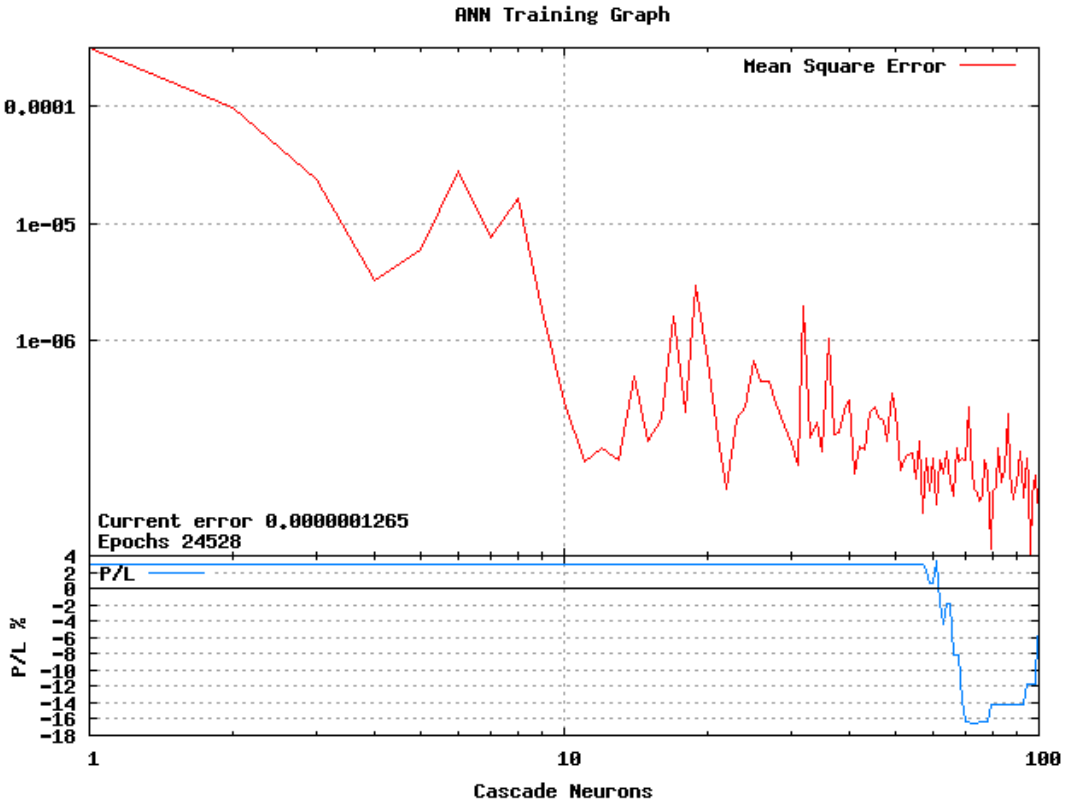


(a) Training Data

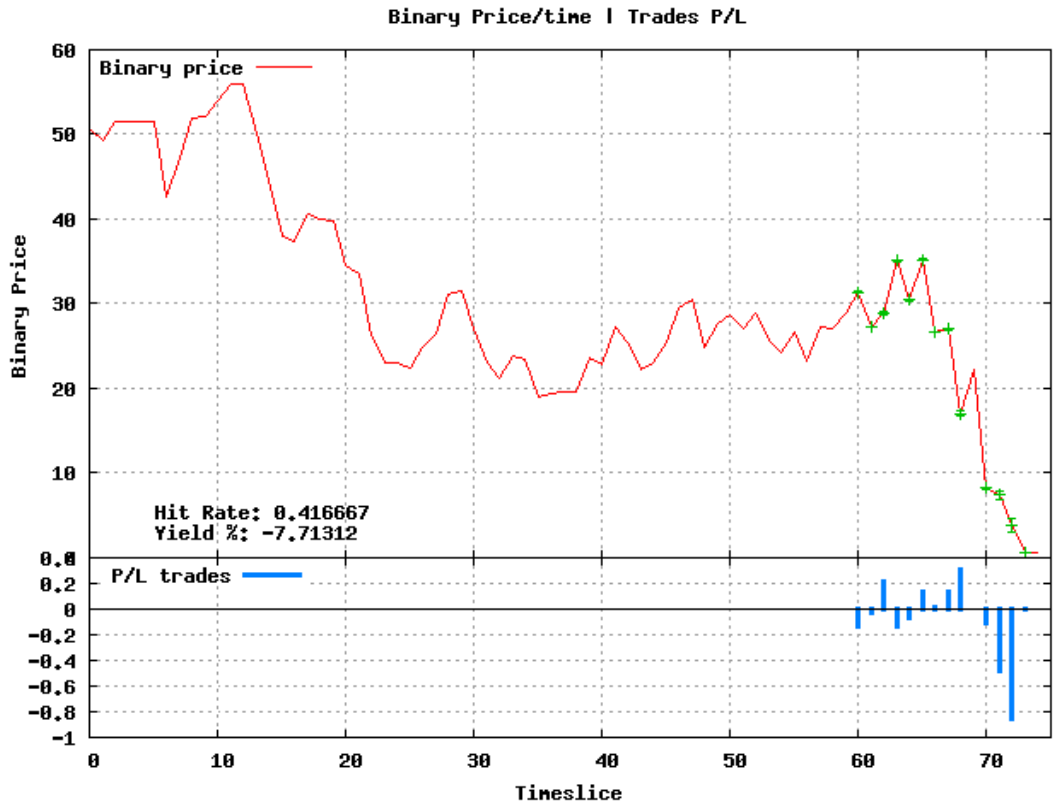


(b) Trading Data

Figure A.7: Test 1-1: Ivan Ljubicic v Mario Ancic

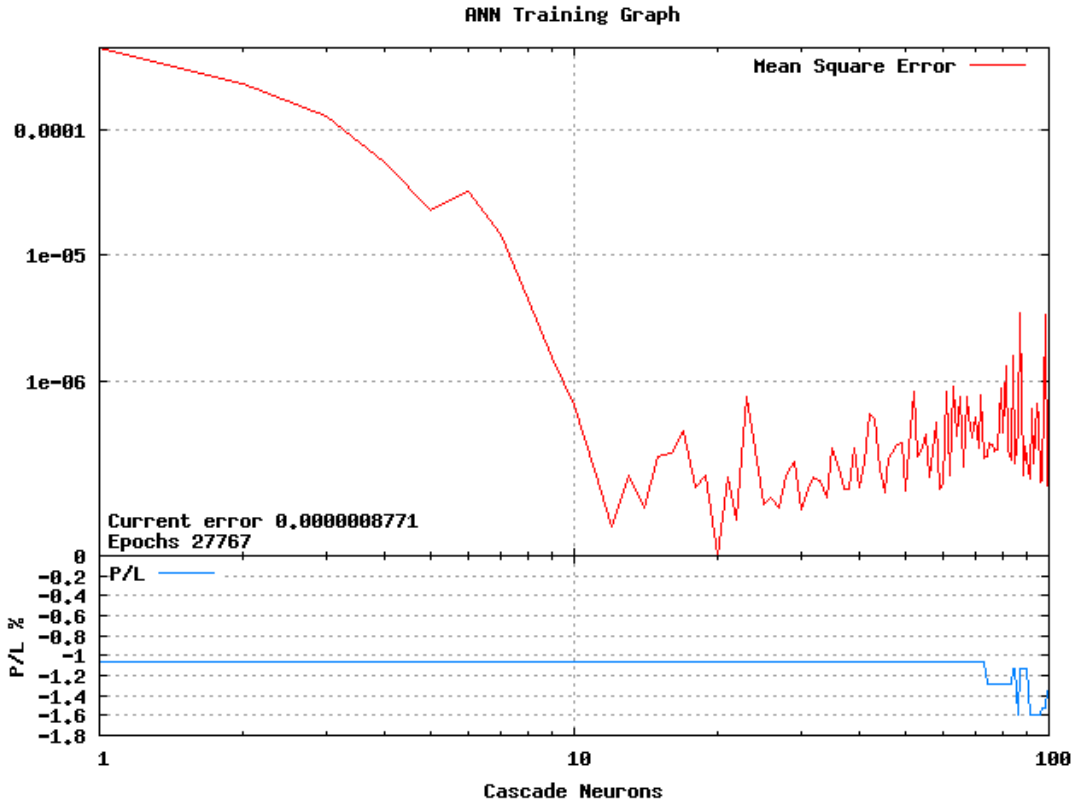


(a) Training Data

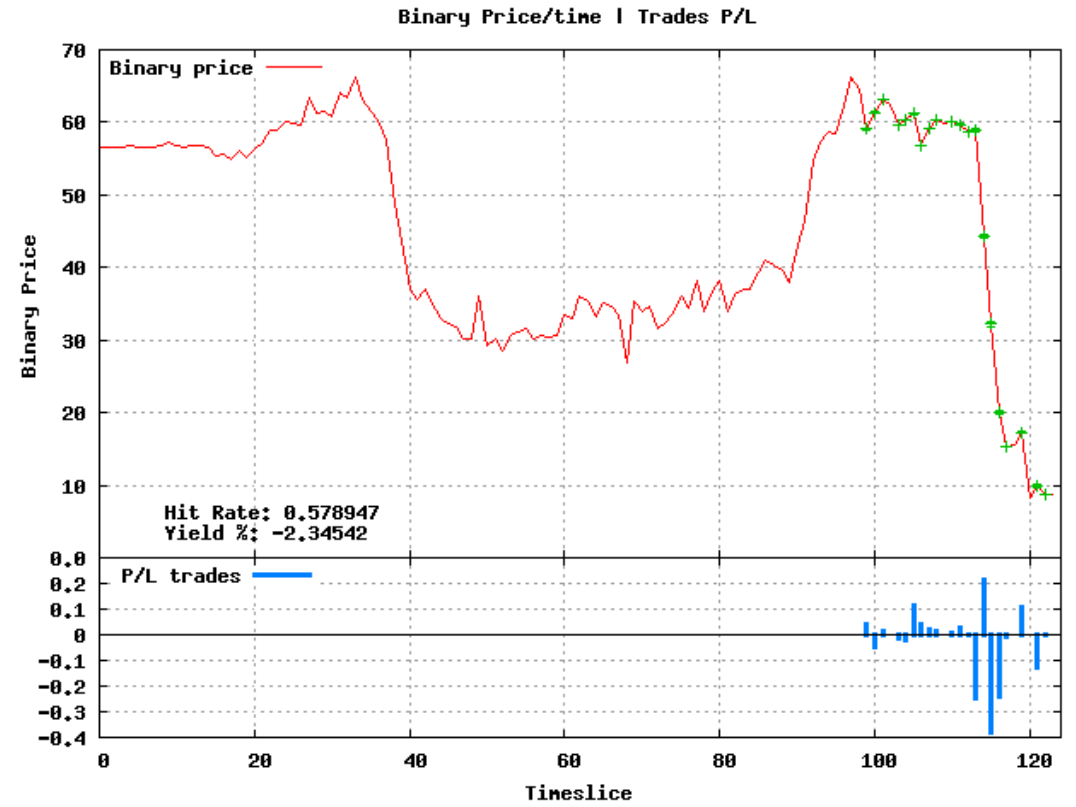


(b) Trading Data

Figure A.8: Test 1-1: Stefan Koubek v Olivier Rochus

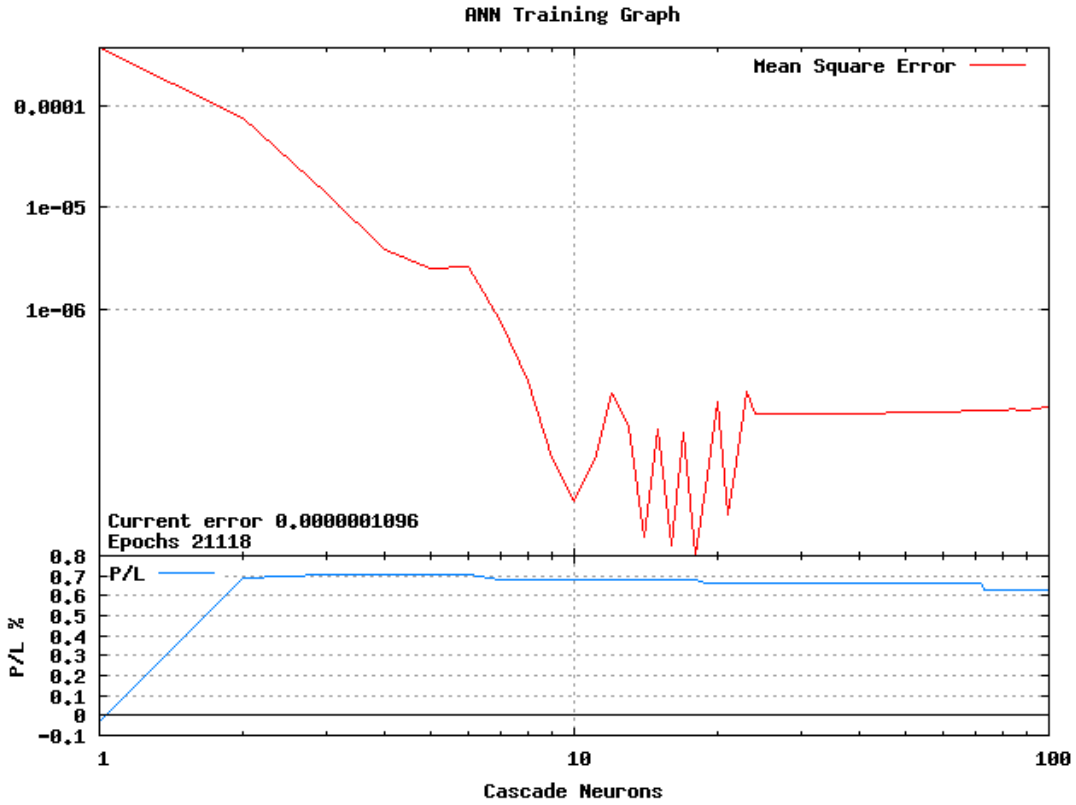


(a) Training Data

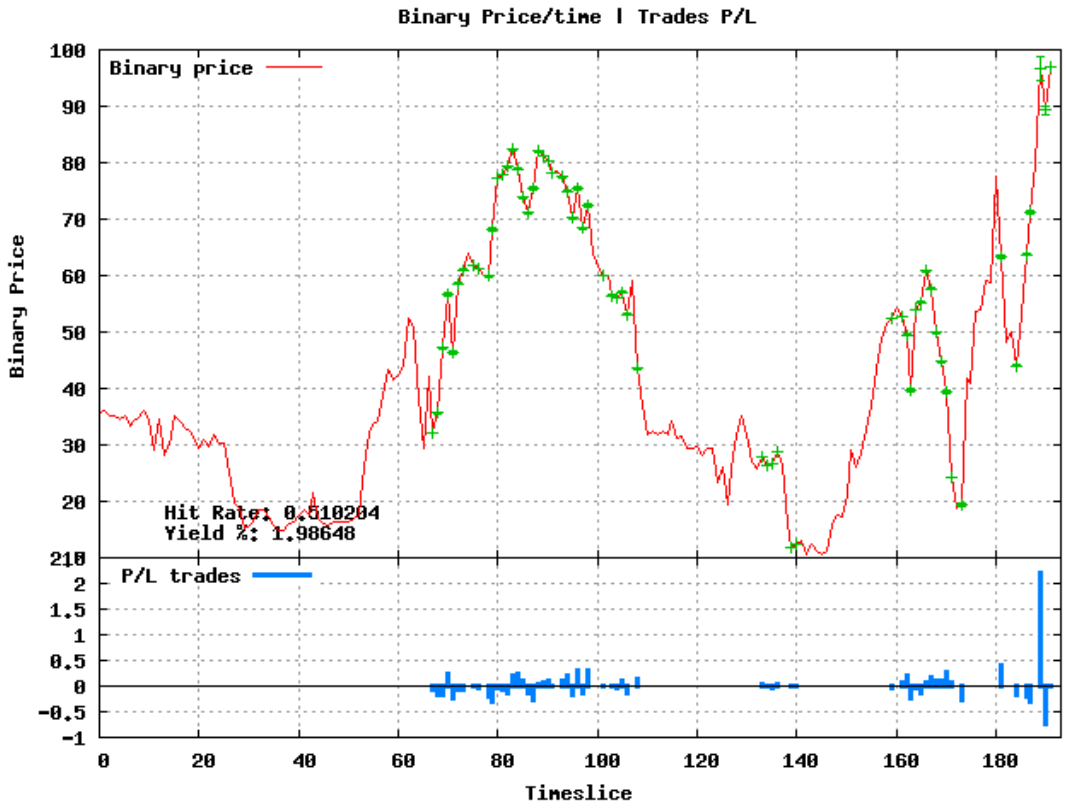


(b) Trading Data

Figure A.9: Test 1-1: Janko Tipsarevic v Feliciano Lopez



(a) Training Data



(b) Trading Data

Figure A.10: Test 1-1: Carlos Berlocq v Luis Horna

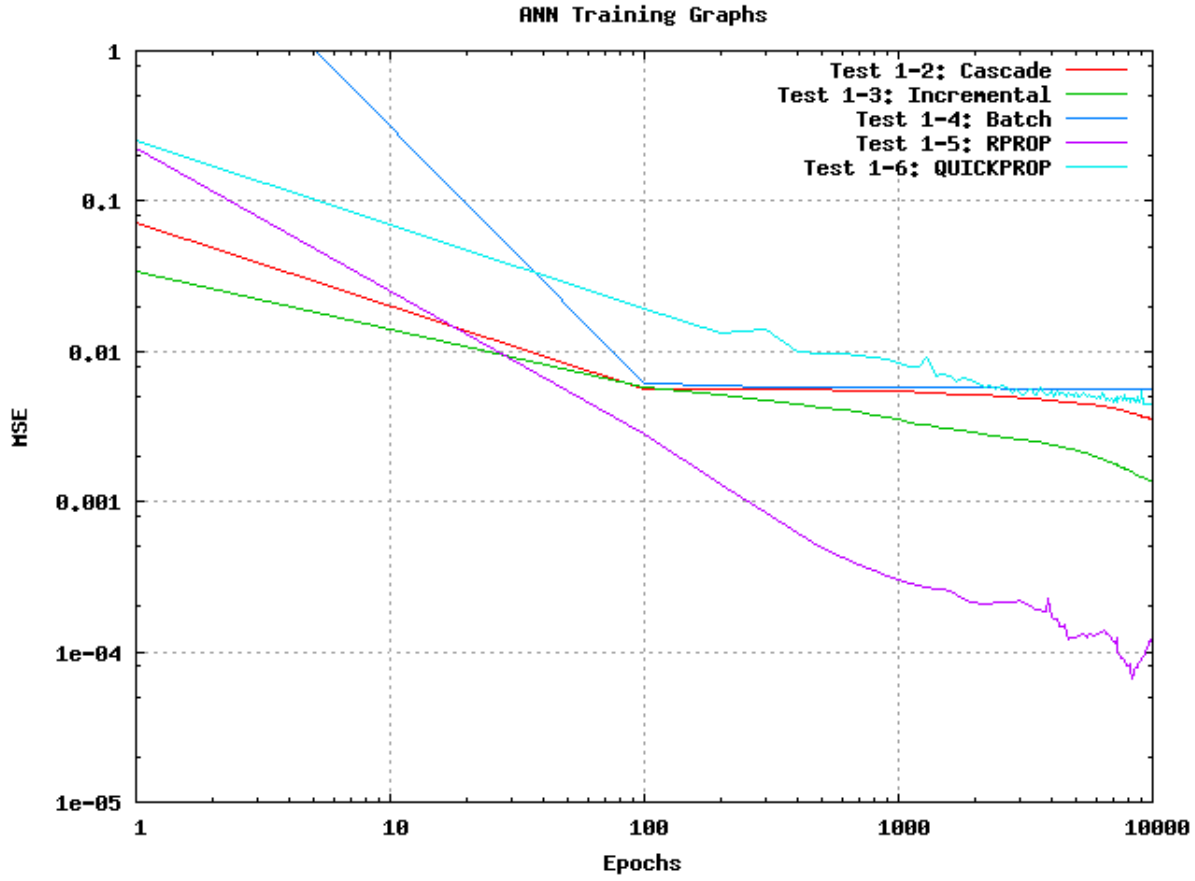


Figure A.11: The MSE of the trained networks. Test 1-2 Cascade was trained with 100 neurons, but appear in the graph for comparison reasons. The x-axis is logarithmically scaled.

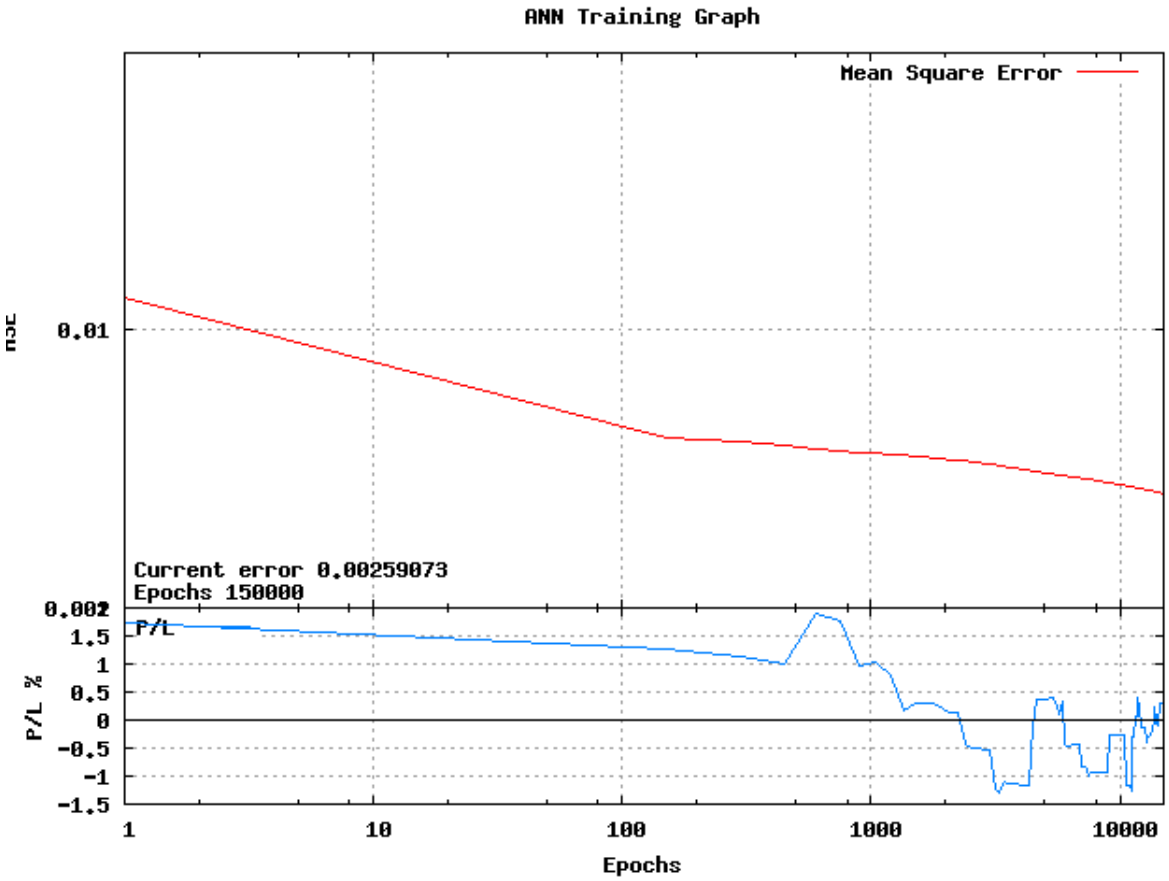
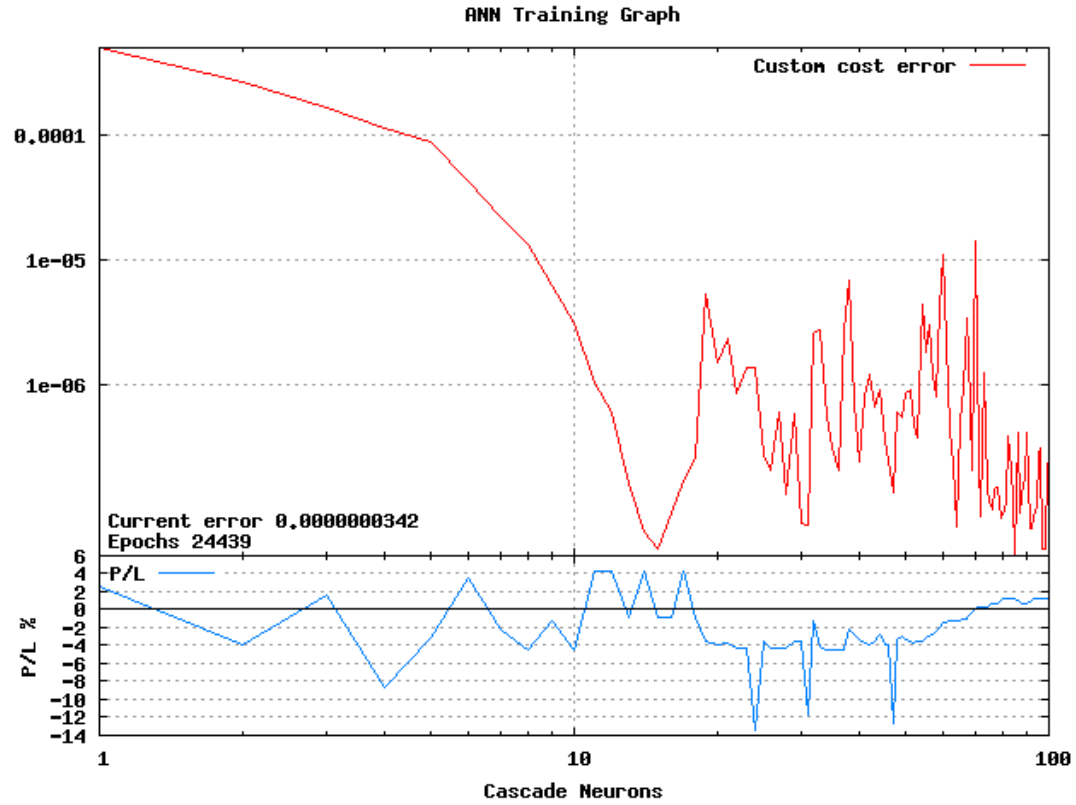


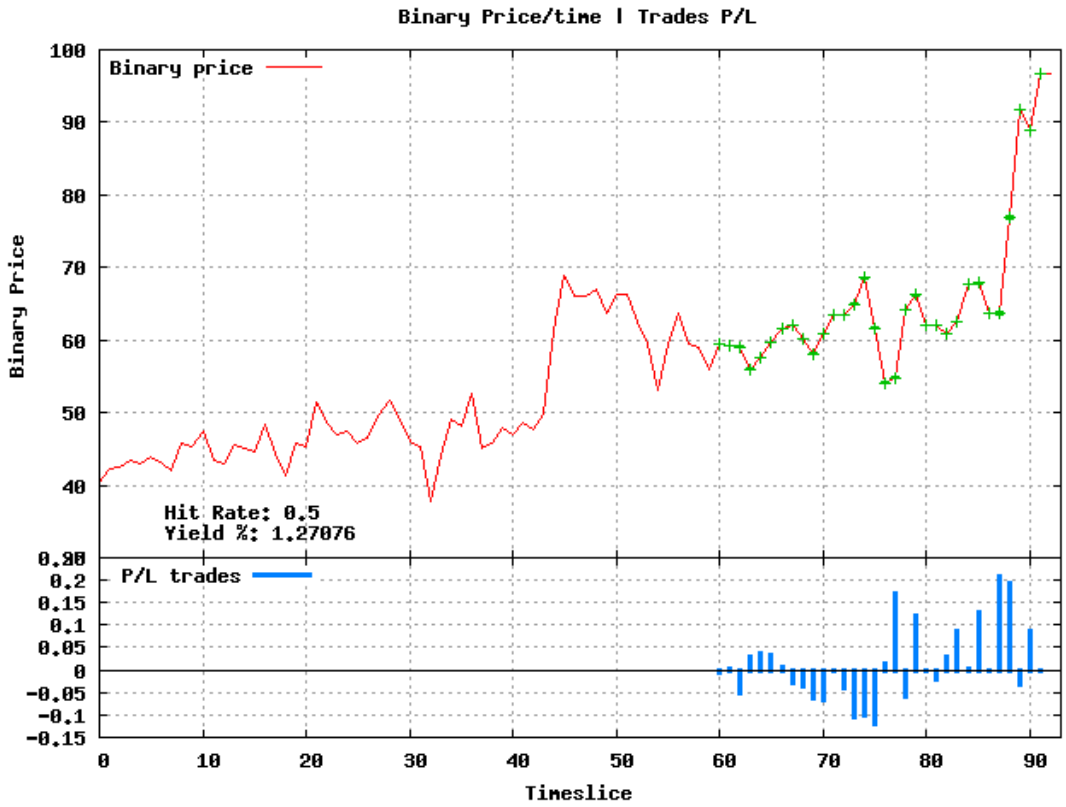
Figure A.12: The error of the network trained in Test 1-7. The sub-graph shows how the profit/loss of the test set evolves as the network are being trained. The x-axis is logarithmically scaled.

Appendix B

TEST 2-1: Custom Cost Function Testing Graphs

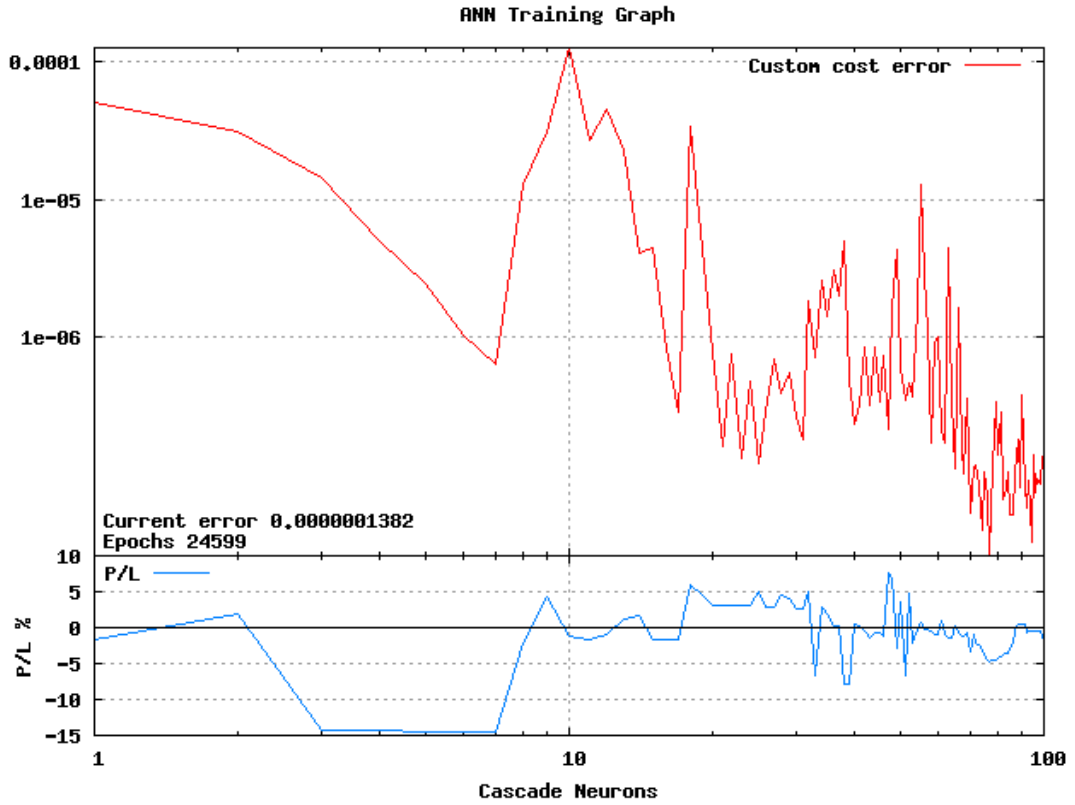


(a) Training Data

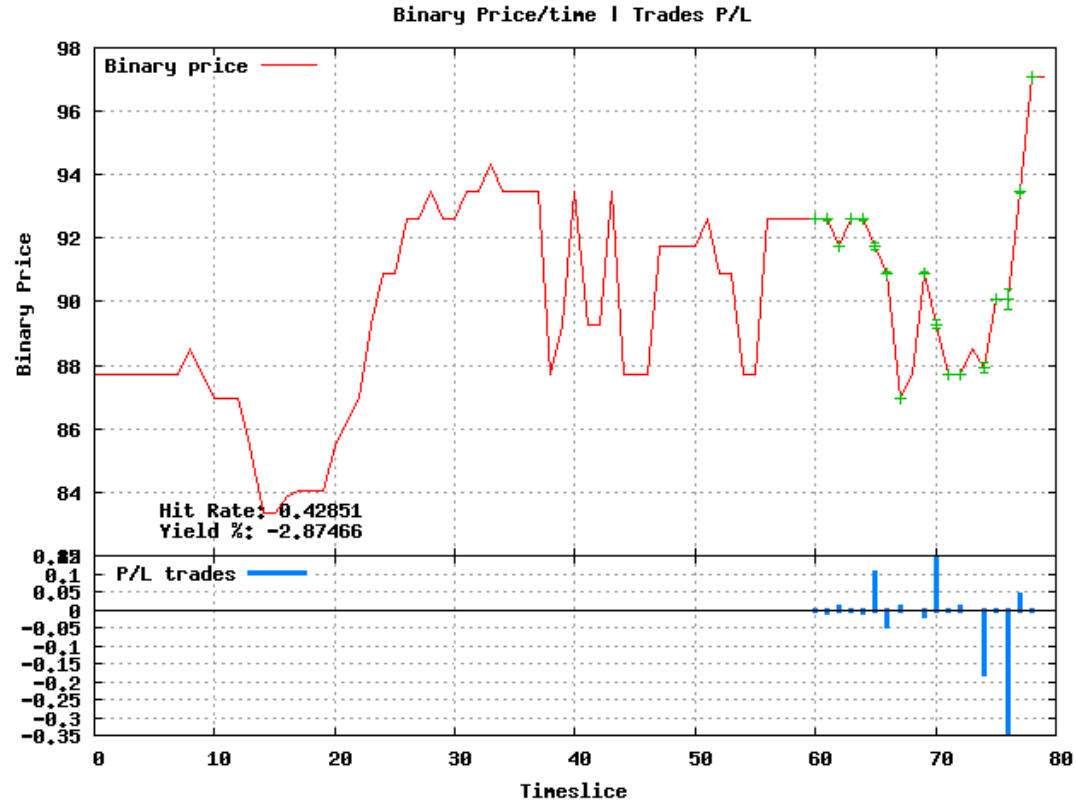


(b) Trading Data

Figure B.1: Test 2-1: Amer Delic v Jurgen Melzer

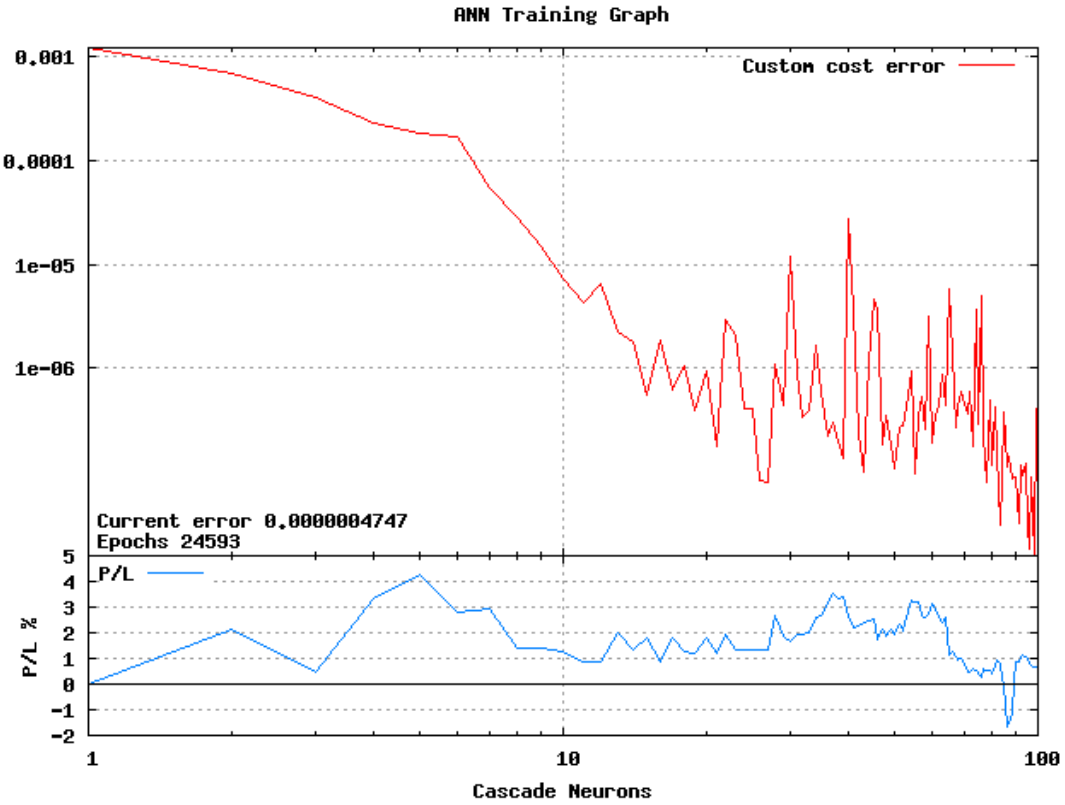


(a) Training Data

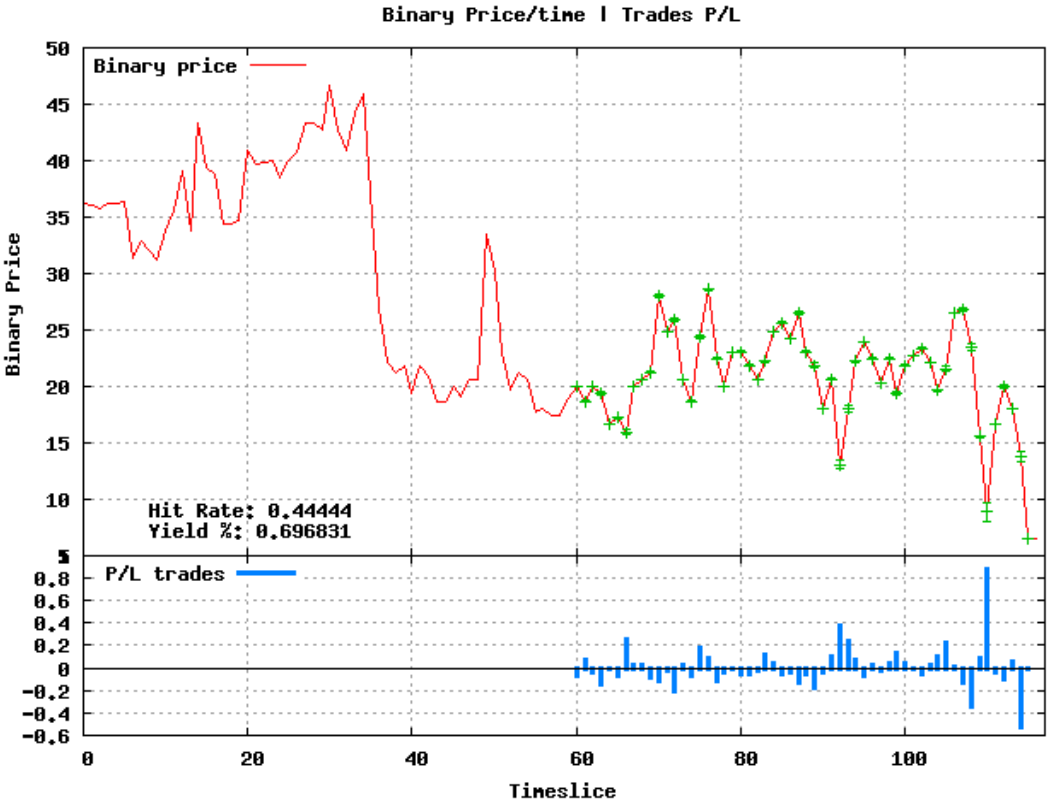


(b) Trading Data

Figure B.2: Test 2-1: Andy Roddick v Gilles Muller

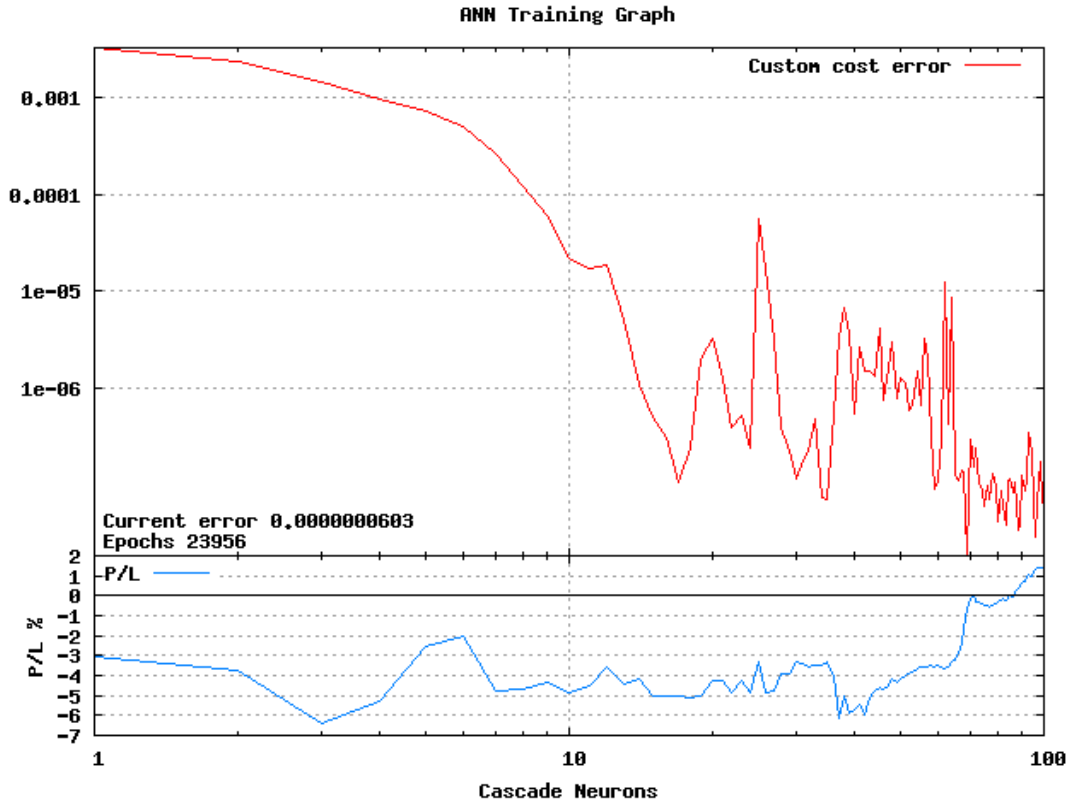


(a) Training Data

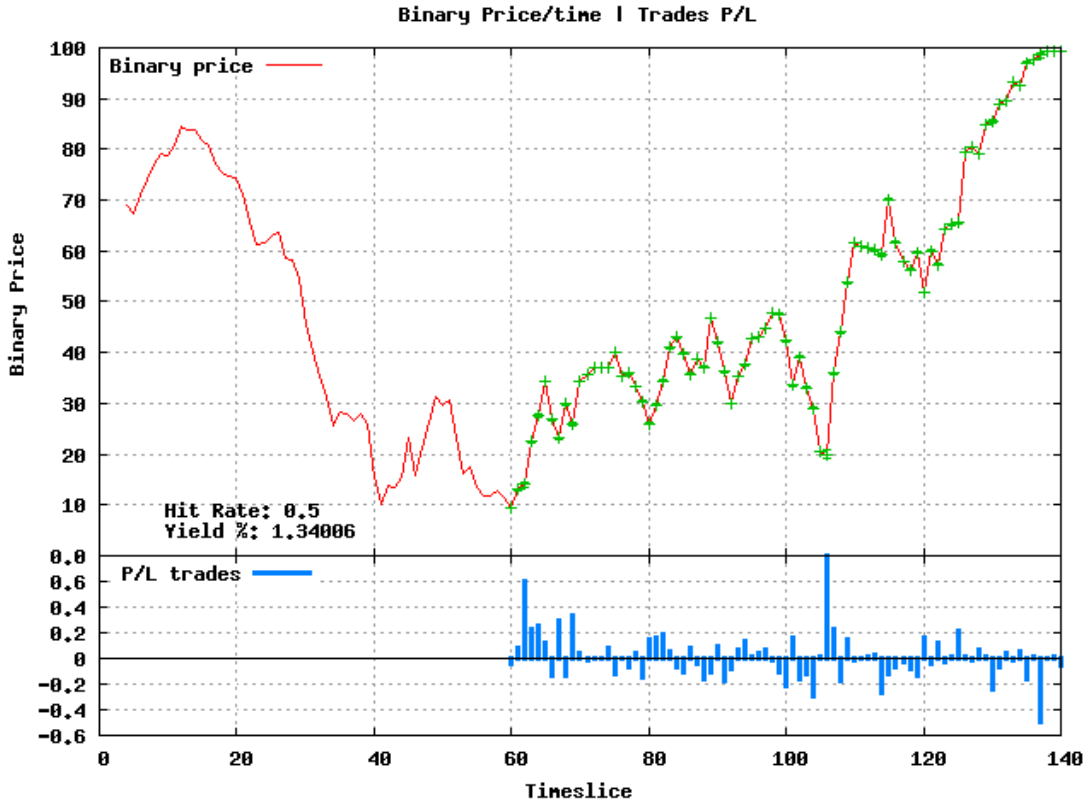


(b) Trading Data

Figure B.3: Test 2-1: Marin Cilic v Mikhail Youzhny

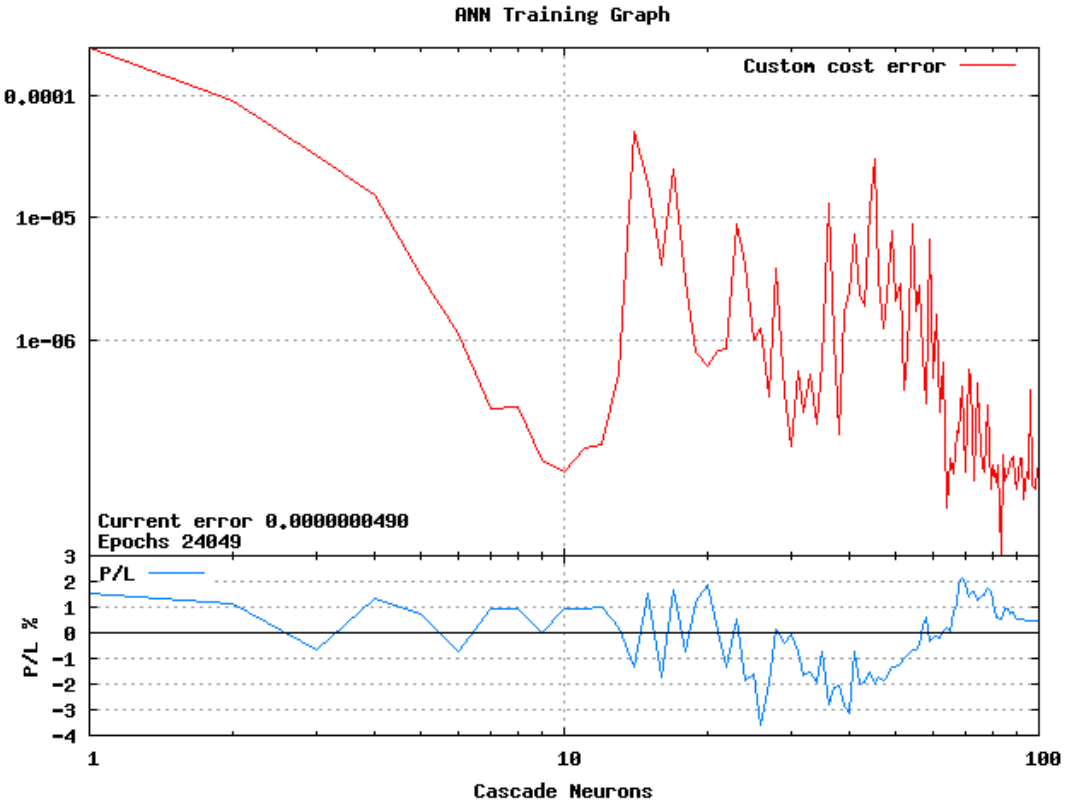


(a) Training Data

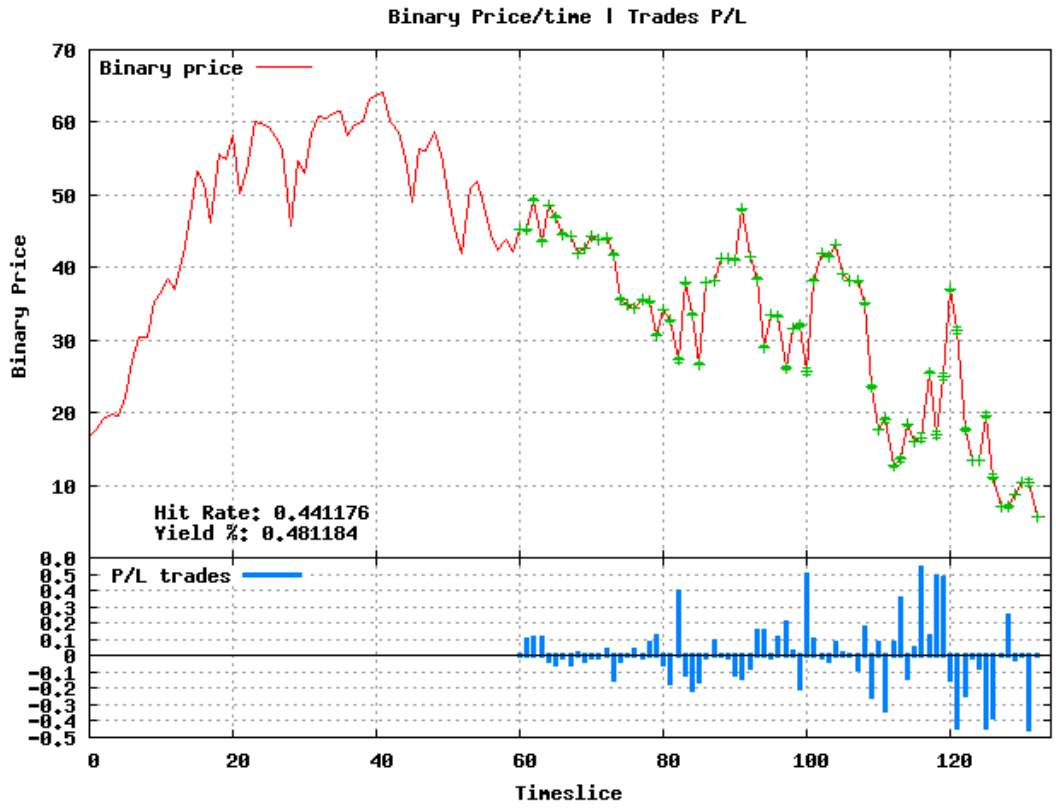


(b) Trading Data

Figure B.4: Test 2-1: Andy Murray v Stanislas Wawrinka

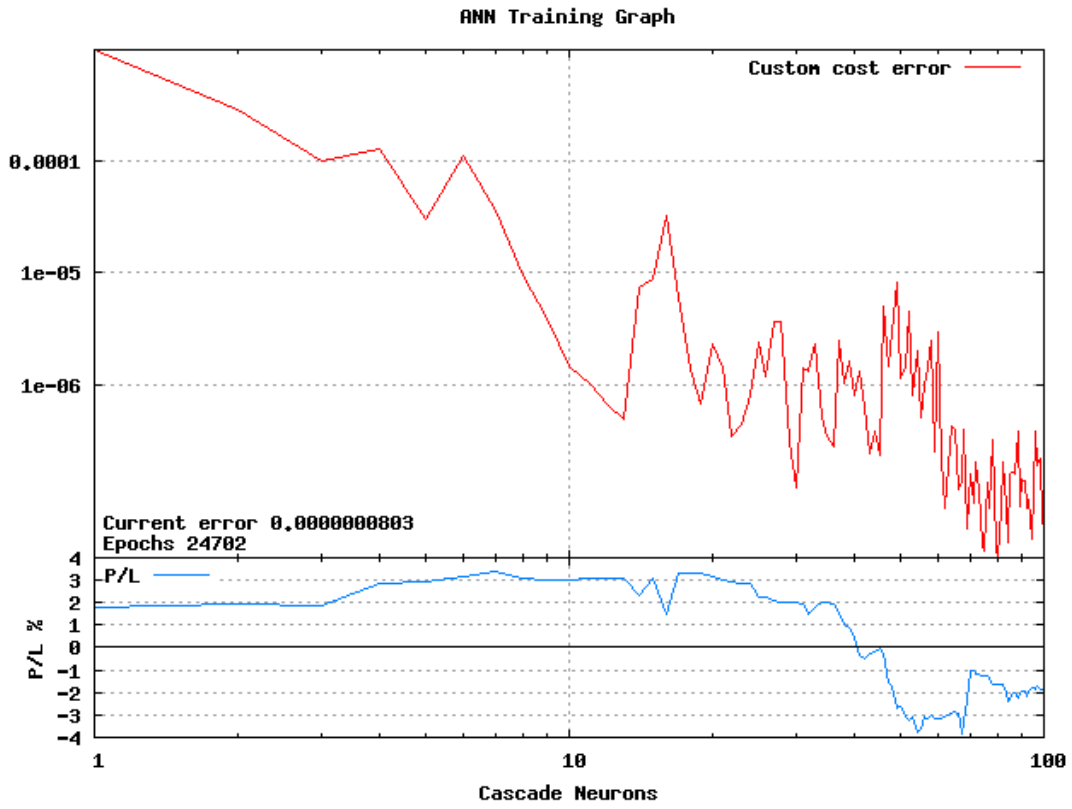


(a) Training Data

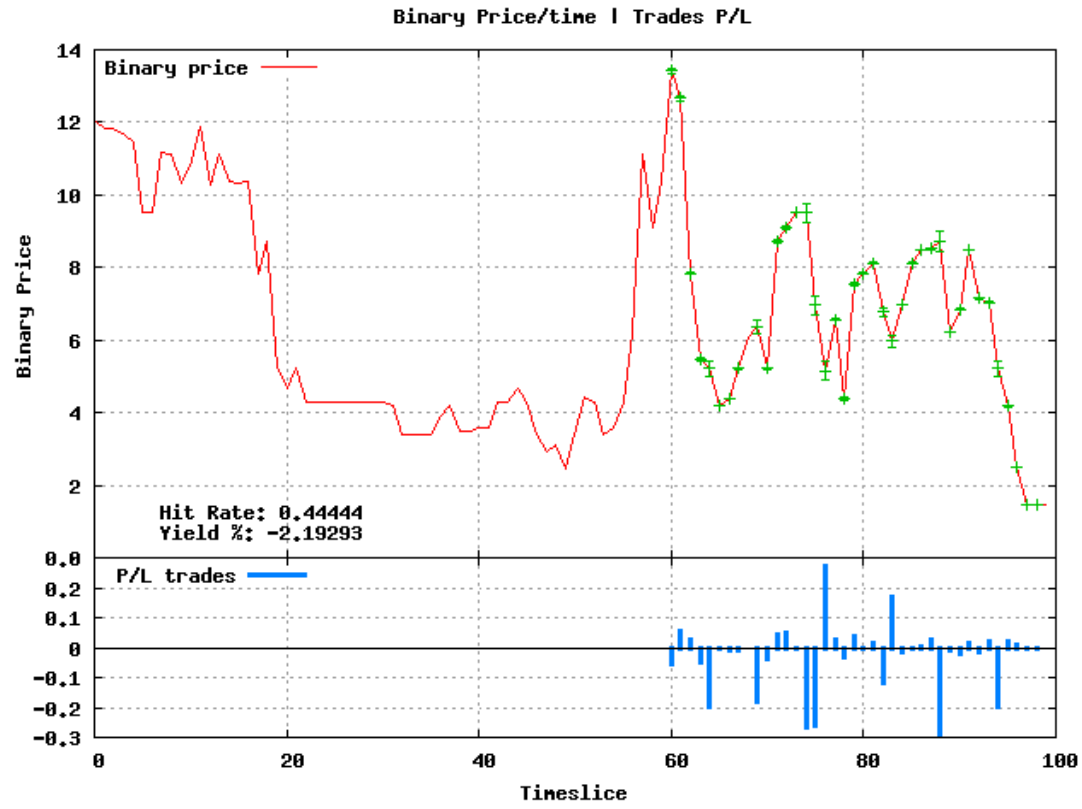


(b) Trading Data

Figure B.5: Test 2-1: Philip Kohlschreiber v Rafael Nadal

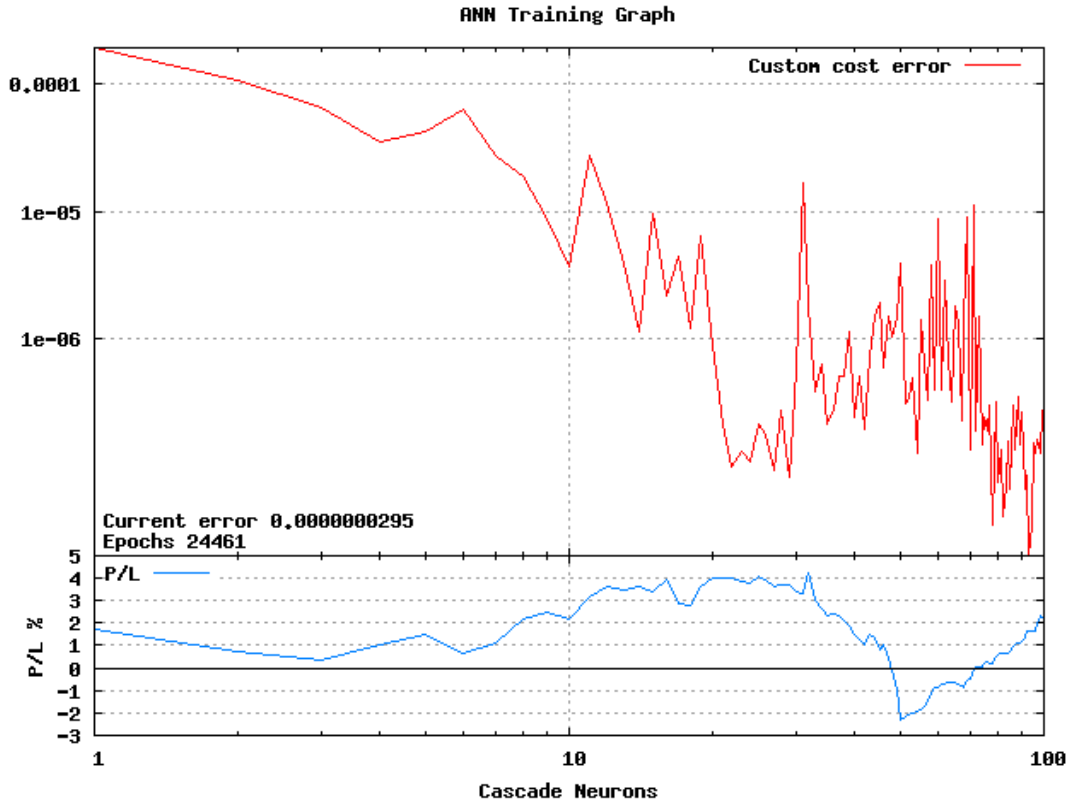


(a) Training Data

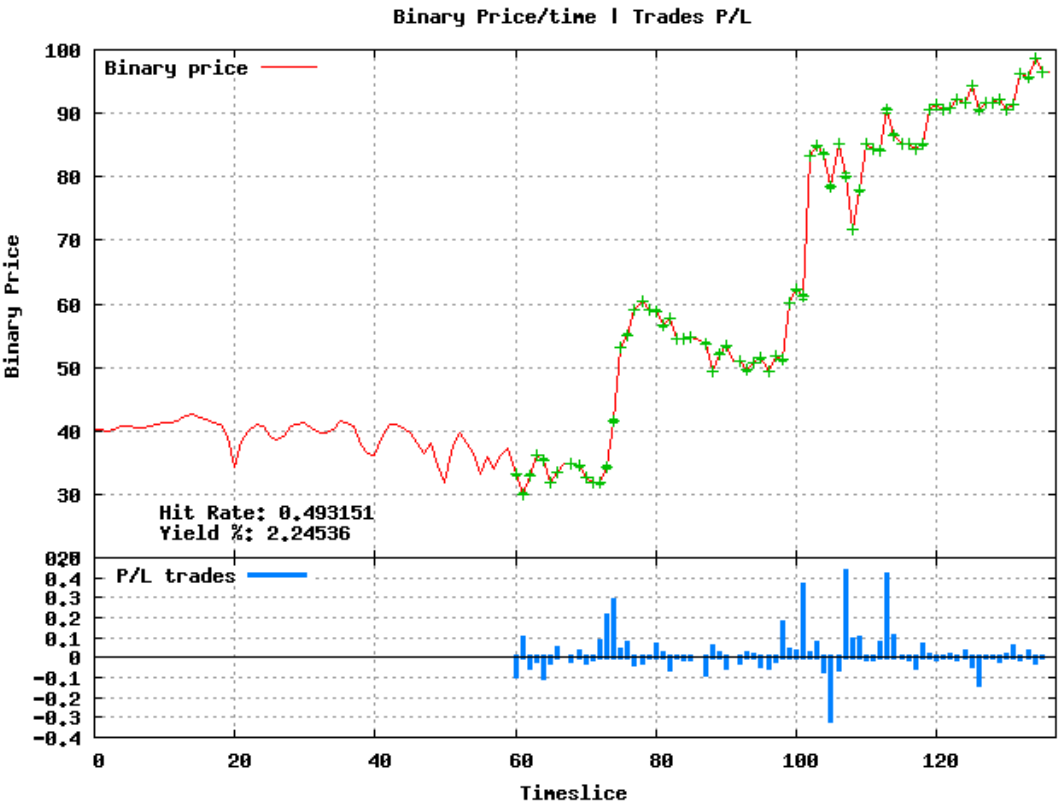


(b) Trading Data

Figure B.6: Test 2-1: Fabrice Santoro v Novak Djokovic

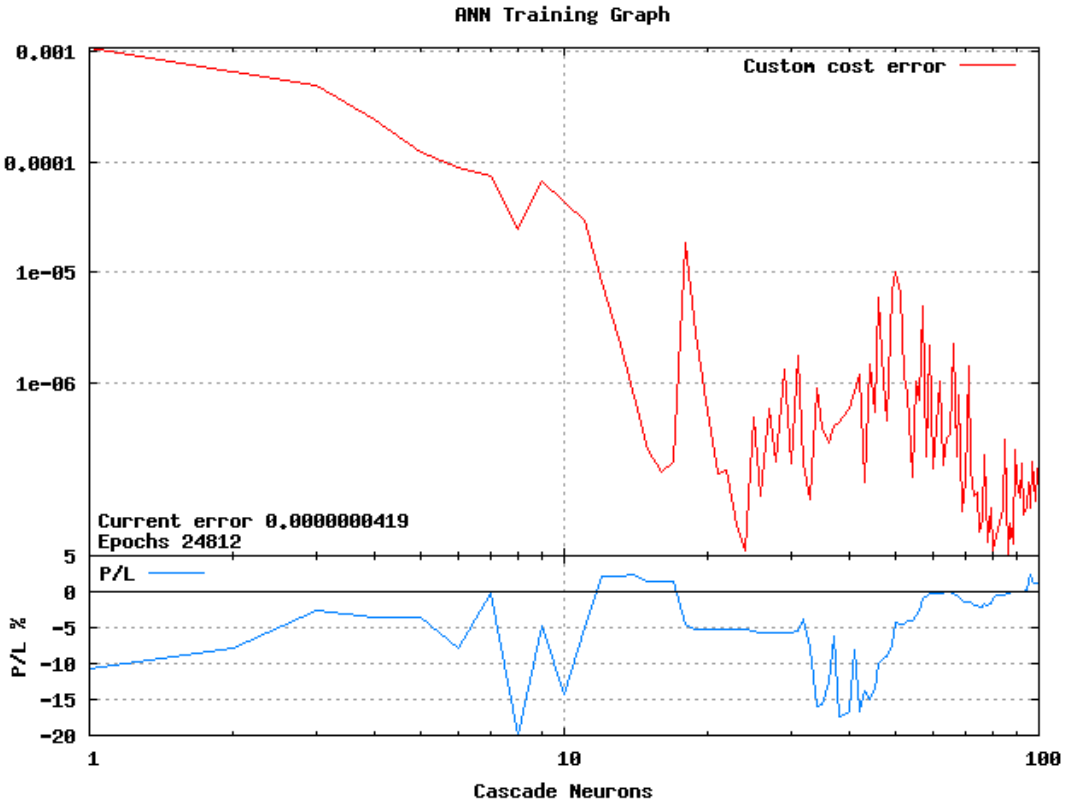


(a) Training Data

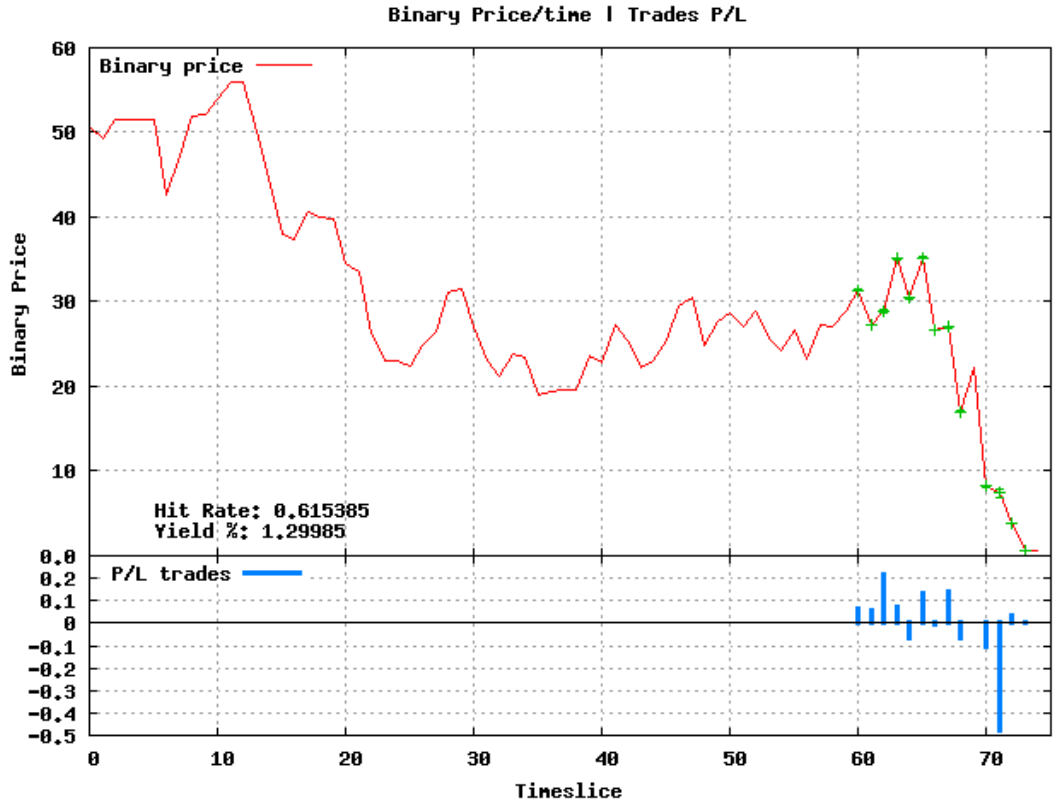


(b) Trading Data

Figure B.7: Test 2-1: Ivan Ljubicic v Mario Ancic

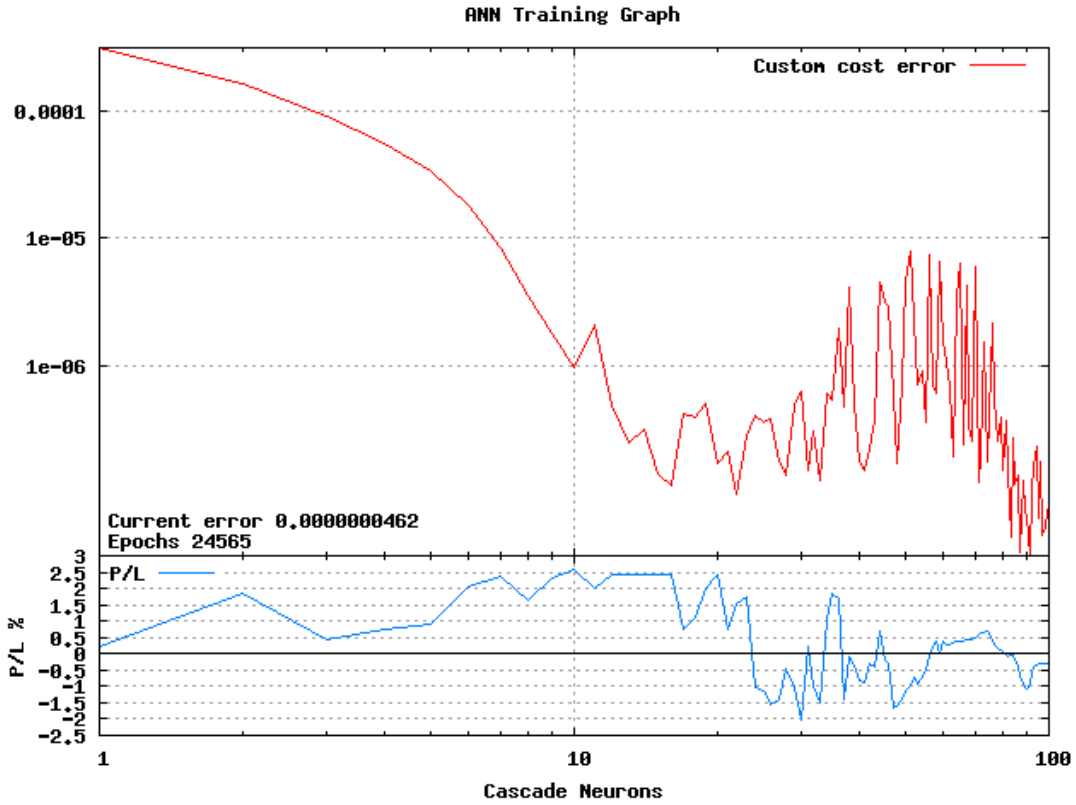


(a) Training Data

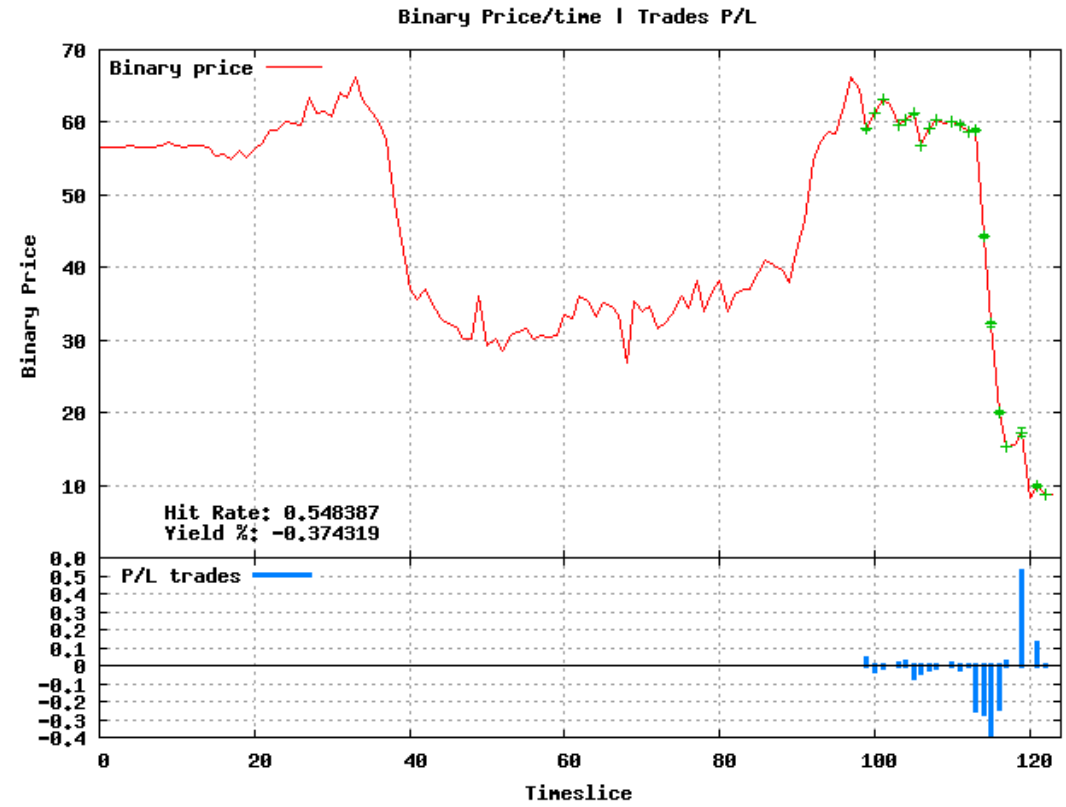


(b) Trading Data

Figure B.8: Test 2-1: Stefan Koubek v Olivier Rochus

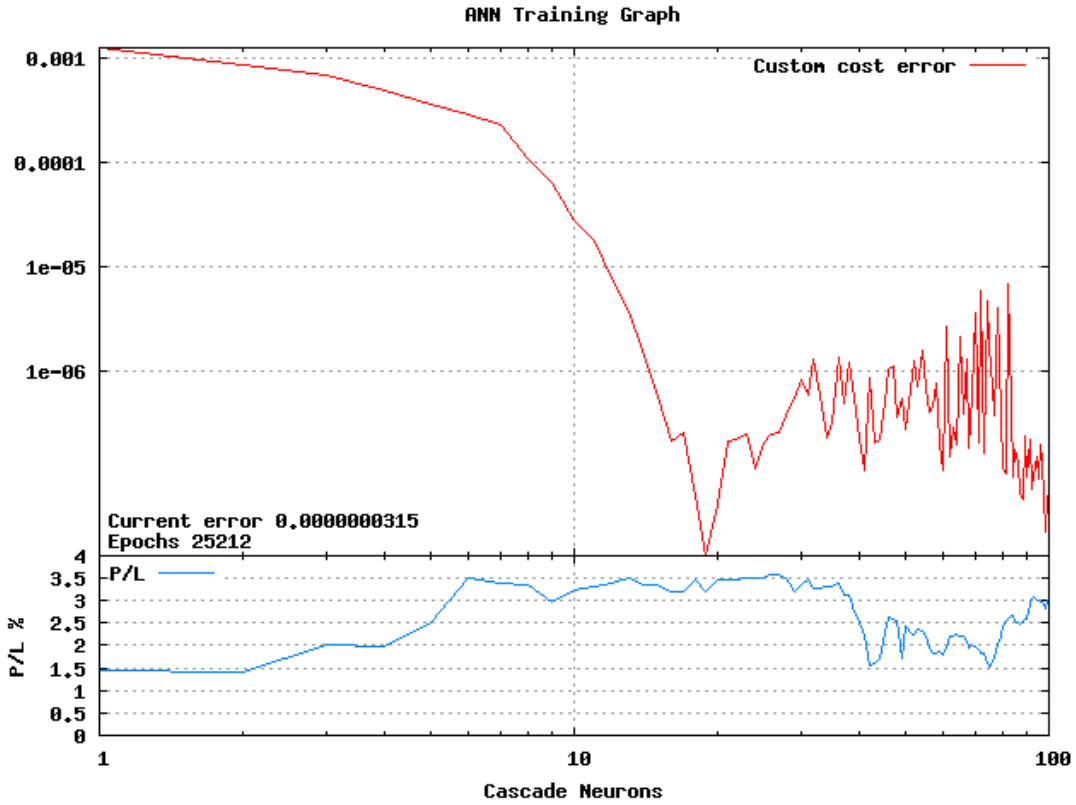


(a) Training Data

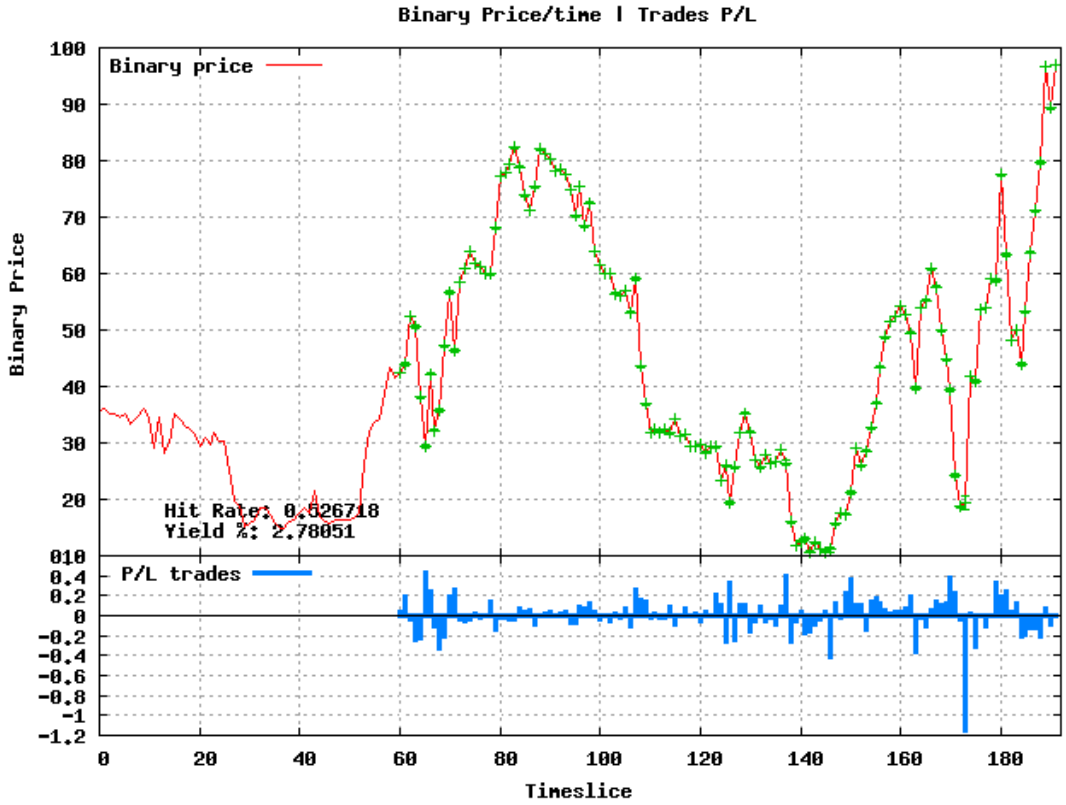


(b) Trading Data

Figure B.9: Test 2-1: Janko Tipsarevic v Feliciano Lopez



(a) Training Data



(b) Trading Data

Figure B.10: Test 2-1: Carlos Berlocq v Luis Horna

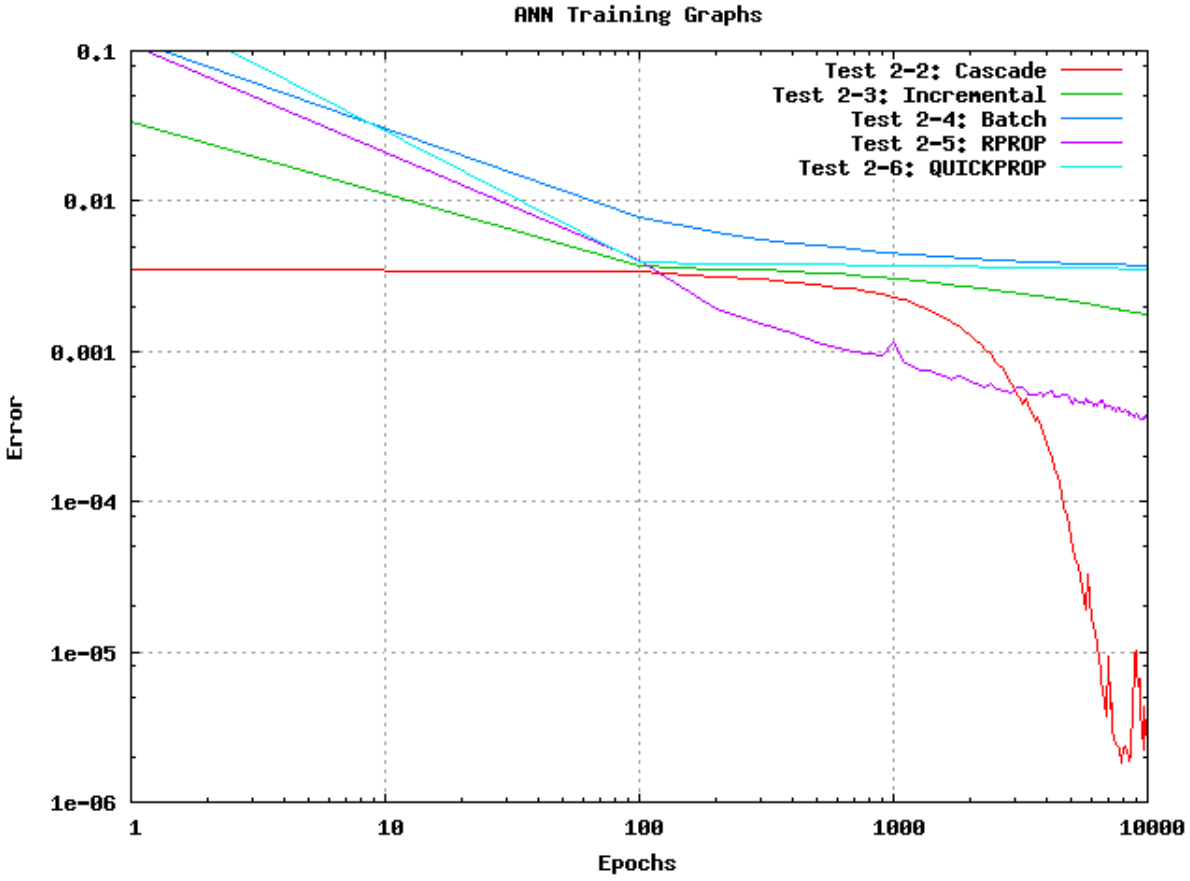


Figure B.11: The error of the trained networks. Test 2-2 Cascade was trained with 100 neurons, but appears in the graph for comparison reasons. The x-axis is logarithmically scaled.

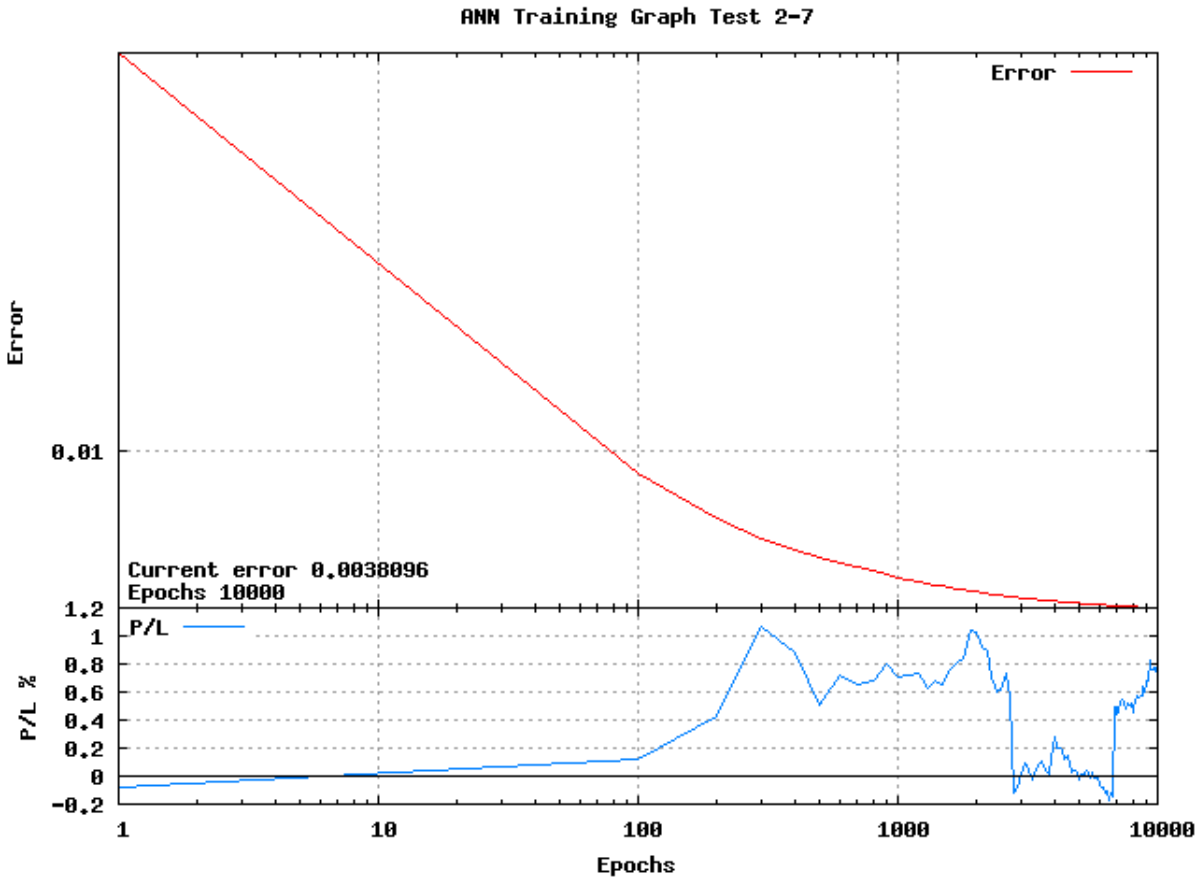


Figure B.12: The error of the network trained in Test 2-7. The sub graph shows how the profit/loss of the test set evolves as the network are being trained. The x-axis is logarithmically scaled.

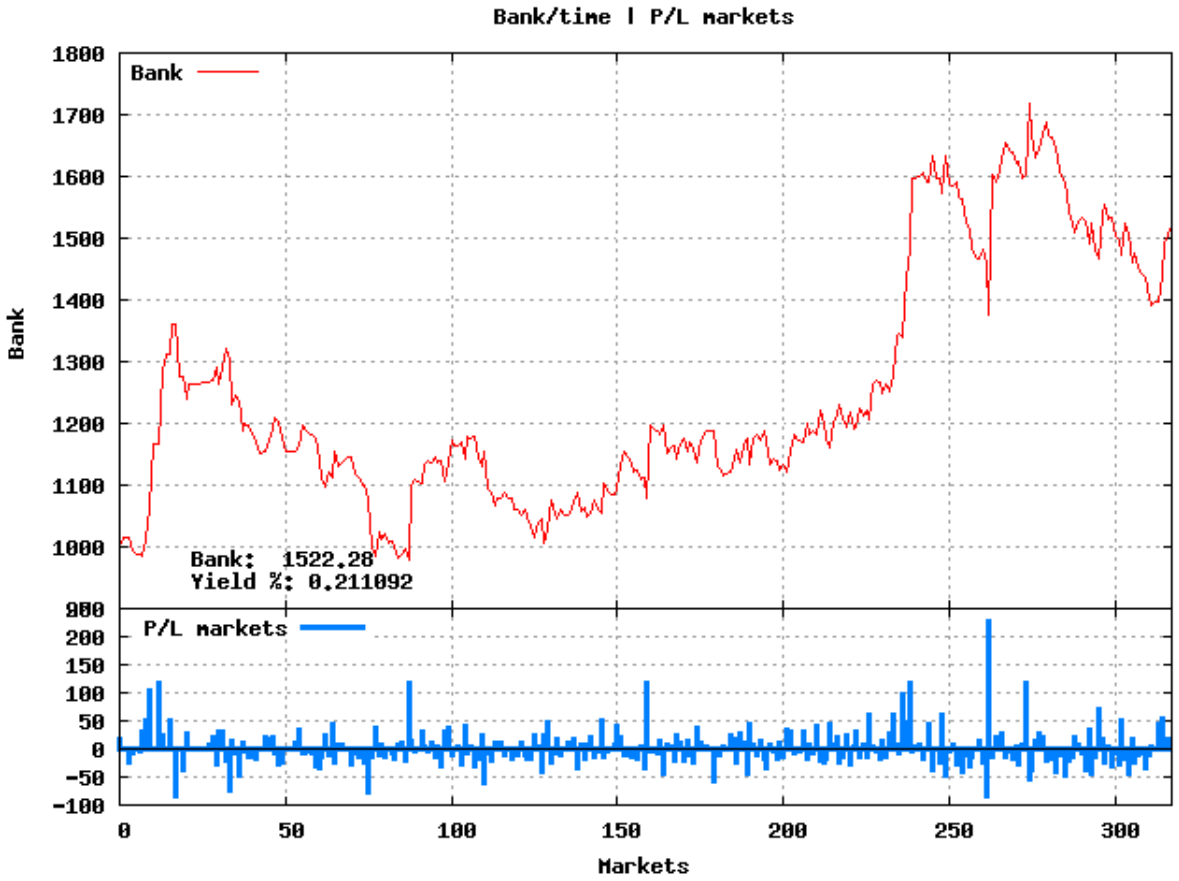


Figure B.13: The full scale version of test 2-1. Uses Kelly staking strategy. Starts with a bank of 1000, final bank is 1522.28 after training and trading 317 individual markets

Appendix C

DB Scheme

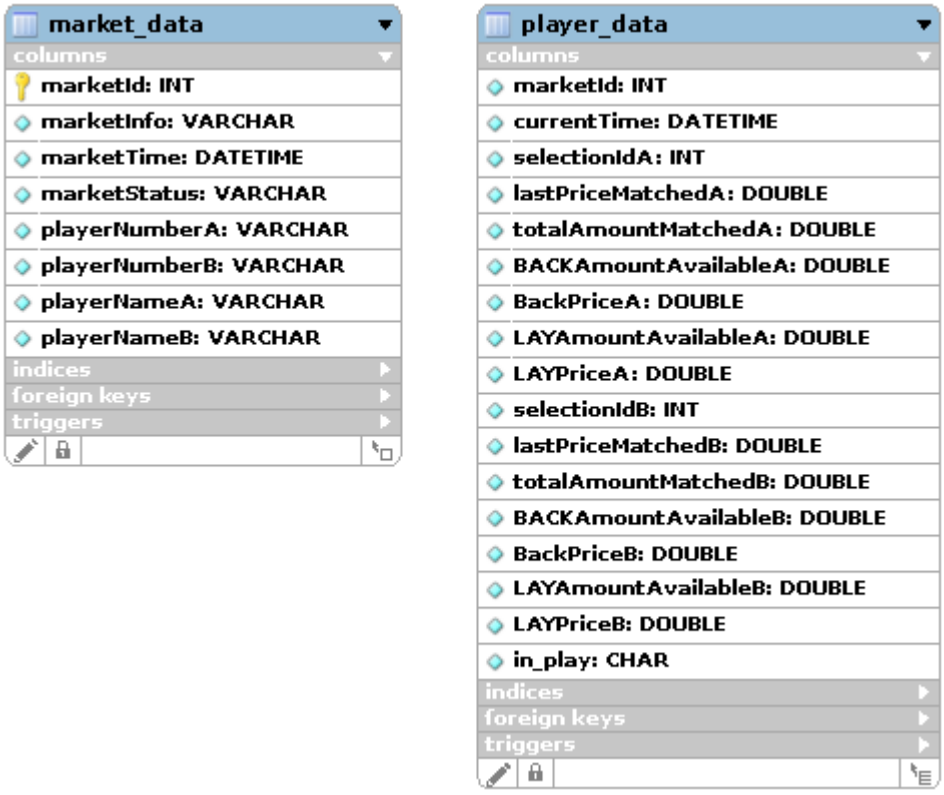


Figure C.1: The database Scheme.

Appendix D

List of Figures

List of Figures

Appendix E

List of Tables

List of Tables