DEGREE PROJECT, IN MASTER'S THESIS AT NADA , SECOND LEVEL
STOCKHOLM, SWEDEN 2015

Training a Multilayer Perceptron to predict the final selling price of an apartment in co-operative housing society sold in Stockholm city with features stemming from open data.

MASTER'S PROJECT IN COMPUTER SCIENCE

RASMUS TIBELL

KTH ROYAL INSTITUTE OF TECHNOLOGY

COMPUTER SCIENCE & COMMUNICATION

KTH Computer Science
and Communication

Training a Multilayer Perceptron to predict the final selling price of an apartment in co-operative housing society sold in Stockholm city with features stemming from open data.

Master's Project in Computer Science

RASMUS TIBELL

Master's Thesis at NADA
Supervisor: Professor Stefan Arnborg
Examiner: Professor Stefan Arnborg

Abstract

The need for a robust model for predicting the value of condominiums and houses are becoming more apparent as further evidence of systematic errors in existing models are presented. Traditional valuation methods fail to produce good predictions of condominium sales prices and systematic patterns in the errors linked to for example the repeat sales methodology and the hedonic pricing model have been pointed out by papers referenced in this thesis. This inability can lead to monetary problems for individuals and in worst-case economic crises for whole societies.

In this master thesis paper we present how a predictive model constructed from a multilayer perceptron can predict the price of a condominium in the centre of Stockholm using objective data from sources publicly available. The value produced by the model is enriched with a predictive interval using the Inductive Conformal Prediction algorithm to give a clear view of the quality of the prediction. In addition, the Multilayer Perceptron is compared with the commonly used Support Vector Regression algorithm to underline the hallmark of neural networks handling of a broad spectrum of features.

The features used to construct the Multilayer Perceptron model are gathered from multiple "Open Data" sources and includes data as: 5,990 apartment sales prices from 2011-2013, interest rates for condominium loans from two major banks, national election results from 2010, geographic information and nineteen local features. Several well-known techniques of improving performance of Multilayer Perceptrons are applied and evaluated. A Genetic Algorithm is deployed to facilitate the process of determine appropriate parameters used by the backpropagation algorithm.

Finally, we conclude that the model created as a Multilayer Perceptron using backpropagation can produce good predictions and outperforms the results from the Support Vector Regression models and the studies in the referenced papers.

Referat

Träning av en "Multilayer Perceptron" att förutsäga försäljningspriset för en bostadsrättslägenhet till försäljning i Stockholm city med egenskaper från öppna datakällor

Behovet av en robust modell för att förutsäga värdet på bostadsrättslägenheter och hus blir allt mer uppenbart alt eftersom ytterligare bevis på systematiska fel i befintliga modeller läggs fram. I artiklar refererade i denna avhandling påvisas systematiska fel i de estimat som görs av metoder som bygger på priser från repetitiv försäljning och hedoniska prismodeller. Detta tillkortakommandet kan leda till monetära problem för individer och i värsta fall ekonomisk kris för hela samhällen.

I detta examensarbete påvisar vi att en prediktiv modell konstruerad utifrån en "Multilayer Perceptron" kan estimera priset på en bostadsrättslägenhet i centrala Stockholm baserad på allmänt tillgängligt data ("Öppen Data"). Modellens resultat har utökats med ett prediktivt intervall beräknat utifrån "Inductive Conformal Prediction"-algoritmen som ger en klar bild över estimatets tillförlitlighet. Utöver detta jämförs "Multilayer Perceptron"-algoritmen med en annan vanlig algoritm för maskinlärande, den så kallade "Support Vector Regression" för att påvisa neurala nätverks kvalité och förmåga att hantera dataset med många variabler.

De variabler som används för att konstruera "Multilayer Perceptron"-modellen är sammanställda utifrån allmänt tillgängliga öppna datakällor och innehåller information så som: priser från 5990 sålda lägenheter under perioden 2011-2013, ränteläget för bostadsrättslån från två av de stora bankerna, valresultat från riksdagsvalet 2010, geografisk information och nitton lokala särdrag. Ett flertal välkända förbättringar för "Multilayer Perceptron"-algoritmen har applicerats och evaluerats. En genetisk algoritm har använts för att stödja processen att hitta lämpliga parametrar till "Backpropagation"-algoritmen.

I detta arbete drar vi slutsatsen att modellen kan producera goda förutsägelser med en modell konstruerad utifrån ett neuralt nätverk av typen "Multilayer Perceptron" beräknad med "backpropagation", och därmed utklassar de resultat som levereras av Support Vector Regression modellen och de studier som refererats i denna avhandling.

Acknowledgement

Firstly, I would like to thank my supervisor Professor Stefan Arnborg for his useful guidance and discussions throughout the process. Moreover, for his help with suggestions and comments during the writing of this thesis. In addition, I would like to thank my four boys: Linus, Julius, Marcus and Cornelius for their patience. Finally, my appreciation to Susanne for all her support and encouragement.

Contents

Contents

Chapter 1

Introduction

Research on neural networks in general and multilayer perceptrons in particular has lead to many novel ideas that enhance the quality of the predictions and reduces the running time of the algorithm. That in combination with the gained knowledge in using multilayer perceptrons has opened new fields for their use. The question at hand is if a multilayer perceptron can predict the final selling price of an apartment in co-operative housing society sold in Stockholm city with reasonable accuracy. Further, can it outperform a more commonly used machine learning system like the support vector regression (SVR)?

1.1 Background

The predicting power of the machine learning system has steadily increased over the years mainly due to intensive research that has refined and improved the underlying algorithms. The same development holds for the Multilayer Perceptrons but the path to the current abilities and performance has had its difficulties. In the early 1960's the perceptrons became popular and the expectations were high but in 1969 Minsky and Papert analysed their limitations in non-linear problems and dampened the enthusiasm. Adding hidden layers (multilayer) does not help to break these limitations as long as they are linear. The power comes from the combination of multiple layers of hidden units using non-linear activation functions. One major restriction is that the perceptron-learning algorithm is ill suited for neural networks with multiple layers of non-linear units. The solution came with the backpropagation algorithm that generalizes well with multilayer perceptrons with linear and non-linear units. However, it is a bit misleading that they still are called multilayer perceptrons despite the fact that the original perceptron algorithm is seldom used these days.
Predicting the price of a house or an apartment is a classic and commonly used example within the machine learning community. This in combination with the author being a resident in Stockholm makes the market interesting to study. Even as a resident it is often hard to understand this, which factors are affecting the prices

of apartments and is the motivation for this project. The notion is that it probably is more complex to analyse the market of apartments in Stockholm than determine the housing price in other parts of Sweden. The idea was born to try to predict the prices of apartments for sale and to find some of the major factors affecting the final price. All data used in the project stems from numerous public available sources, so called "Open Data" sources. This hopefully makes it easy for those who want to look into this machine learning example domain and experiment and draw their own conclusions from this work.
There is a belief amongst some of the experts that the price for which the apartments are sold does not reflect the true value of the object. Prices have been constantly rising since the mid nineties and the proportion of the salary spent on living has increased for the habitants in the Stockholm area. The driving forces behind this are increasing GDP, low production of new apartments, the constant influx of people to Stockholm and a low rate of interest for apartment loans. All these factors together makes the market of apartment quite complicated and increases the complexity of predicting an accurate price. It is also a concern for politicians and their economy advisors that a price drop can result in a crisis.

1.1.1 Problem description

This paper explores the feasibility of creating a machine learning model that can predict the price of a apartment sold in central Stockholm with a fair precision and based on a neural network of type Multilayer Perceptron with a handful of contemporary techniques applied. The Multilayer Perceptron is henceforth often shortened to MLP.

1.1.2 Traditional pricing model

One of the most widespread model used to analyse property values is the hedonic price model, which is often used by condominium brokers, banks and lending institutions. This model is based on the assumption that apartments are not homogeneous but differ in their attributes and that this is reflected in the selling price where the buyer implicitly pays for these favourable attributes. The hedonic price equation can be written as:

$$y_i = \beta_j X_{j,i} + \epsilon_i \tag{1.1}$$

In the above equation (1.1) y_i is the i-th observed sales price where $i \in 1 \dots N$, N are the number of sales. The implicit prices of the attributes mentioned above is found in β where $j \in 1 \dots F$, X is the sales data and F is the number of attributes. Errors are captured by the error term ϵ. Attributes often encapsulate the characteristics of the apartment, location and features of the neighbourhood.
Another methodology often used in USA is the so called repeat sales methodology that tries to solve the heterogeneity problem by assuming that houses and apartments do not change attributes over time and that the prices from repeated sales

can be used to estimate the price. The model is as follows:

$$\frac{P_{it_1}}{P_{it_0}} = \frac{B_{t_1}}{B_{t_0}} U_{it_0t_1} \tag{1.2}$$

where P_{it} is the price of house i sold at time t, B_t is the price index at time t and $U_{it_nt_{n+1}}$ is the error term. An assumption is that the same house is sold frequently.

1.1.3 Objective

As mentioned before the goal was to find a machine learning model that can predict the selling price of an apartment situated in the centre of Stockholm. This has to be done with a good quality to be meaningful for the end user. This kind of model can be use by apartment agencies to predict the future selling price or by financial institutions (loan givers) to find out the value of the pledge. To construct the model a neural network of the type multilayer perceptron was used in combination with some novel techniques to increase the precision of the predictions. Below is a list of the techniques use:

- Gradient descent
- MiniBatch
- Early stopping
- Adjustable learning rate and momentum
- Random initialization of weights
- Conformal Prediction

The performance of the produced model is compared with support vector regression (SVR) to verify that the multilayer perceptron can outperforms a regular SVR.

1.1.4 Restrictions

This paper will not describe the whole process of how to construct a complete application. Nor will it result in any deployable software solution. However, all the code snippets that were developed during this research will be made available as Open Source at github.com. The purpose of this thesis was to study the prospects of constructing a predictive model that can produce results usable in practical situations and that can be interpreted by the general audience. Information gathering for this work was treated as a necessity rather than as a feature. This work does not claim to use or collect a complete set of data relevant to the domain. The goal was to gather as much information needed to indicate that a good predictive model could be constructed using only "Open Data". Further this work does not intend to include all novel methods used to improve the construction of neural networks,

improvement opportunities are discussed in chapter 7.2. A number of interesting articles and techniques have been put forward the recent years. However, we have incorporated some of the most obvious improvements that are common ground to contemporary work.

Chapter 2

Literature Review

The literature review is divided into two parts since the nature of the reviewed papers naturally falls into two categories: research in the apartment and housing market domain and theory and techniques applicable to machine learning.
In the first part of this chapter, studies regarding pricing structure and factors affecting the final pricing are reviewed. This part of the review is structured on a per article basis; the main reason for selecting this strategy is the diversity of the perspectives in the reviewed papers. The goal is to identify the main factors affecting the pricing of apartments and explore the relevant knowledge in the field accumulated in previous research.
The second part considers articles discussing novel techniques to improve predictions and performance of neural networks, common practices used when evaluating models and recommendations concerning tuning and parameter adjustments for multilayer perceptrons. For these topics the reviews were performed on a per subject basis and each subject is covered in a separate section.

2.1 Literature covering apartment and housing markets

All papers reviewed in this section share the common opinion that real estate assets and apartments are heterogeneous to their nature. The price may be affected by hundreds of factors and that many different outcomes are possible due to buyer's preferences, available information and circumstances of sale. This gives rise to the fact that the price of a property in a given point in time can be modelled as a random variable and a random error.

2.1.1 A prominent role in society

As predictive models used to estimate values of apartments and real estate get more exact they are more likely to play a more prominent role in society and politics. In the article [1] the authors argue for the importance of a firm pricing model. The situation on the Swedish market has many similarities with the UK market

discussed in the paper. Residential properties are mortgaged in about sixty-five per cent in the UK market. Lenders use the current market price as the key metric for determining the lending ceiling which fuels the risk of overheating the market. This can inflict socio-economic damages and big losses for credit institutions. These problems have been observed in vulnerable countries in Europe and are a big concern for many politicians and researchers in the field. A metric with a more sustainable value could reduce the risk of over heating and would be preferable. One possible source for such a metric could be a predictive model generated from a MLP. For this scenario to be plausible the predictive model has to be refined and give a prudent price estimate of the property.
The predictive model evolved in the paper [1] can predict the UK Average House Price (House Price Index - HPI) from the Nationwide Building Society. The results obtained in the paper are a range of 3 per cent and an average of 1.1 per cent for the index from the second quarter 1999 until the first quarter 2001.

2.1.2 Crimes impact on apartment prices

In the study of crimes impact on apartment prices in Stockholm [2], Caccato and Wilhelmsson conclude from their findings that the apartment price is expected to fall by 0.04 per cent for each per cent in increased crime rate. The decrease in prices rose to 0.21 per cent for each per cent increase in crime rate if only residential burglary is considered. Although the effect is not homogeneous over space, apartment prices are often affected to a lower degree in the central part of Stockholm compared to the outskirts. Taking this knowledge into concern in combination with the fact that this report is targeted against apartment in the central part of the city instigated the decision to exclude crime rate and resembling data from this work. Although several feature related approaches referred to by Caccato and Wilhelmsson are used in the model or have inspired feature selection in our work.

2.1.3 Features used in study of crimes impact on apartment prices

The study [2] previously mentioned is based on 9,622 sale transactions of condominiums in Stockholm during 2008; Mäklarstatistik AB supplied data. To enrich the dataset with supplementary features cross-sectional data from the Stockholm Police, Stockholm Statistics and Stockholm City was merged together with the sales transactions. The information was then used to form features like proximity to water, underground stations, crimes per 10,000 citizens and other characteristics of the neighbourhood. Geographical data was used to divide the region into four quadrants with the central business district (CBD) as a centre point, the distance between the given apartment and the centre point is also used as a feature. Due to the fact that effects of crime can spill-over to neighbouring areas Caccato and Wilhelmsson has incorporated so called smoothed (lagged) variables, these are weighted averages of values for neighbouring locations. This type of feature is not used in this paper;

this follows from the fact that no crime statistics are used. Table 2.1 contains a condensed list of the attributes used.

Table 2.1. List of features used by Caccato and Wilhelmsson

Feature group	
Transaction price	Living area
Number of rooms	Monthly fee
Age of building	Newly built
Elevator in house	Balcony belonging to apartment
Apartment at top floor	Apartment at first floor
Distance to CBD	North-east quadrant
North-west quadrant	South-west quadrant
Distance to water	Distance to underground station
Distance to highway	Distance to main street
Crime rate	Rate of robbery
Vandalism	Outdoor violence
Residential burglary	Shoplifting
Drug related crime	Theft
Theft of cars	Theft from cars
Assaults	

2.1.4 Traditional real estate valuation

Max Kummerow at Curtin University describes the real estate market and their valuation methods in the article [3]. In this paper he elucidates the theory behind the methods used by the real estate market in USA. Valuation methods can be divided into two main categories: objective and subjective, where the objective method stems from the rational paradigms of science in contrast to the subjective method that can be viewed as more of an "art". The fundamental property of the objective method is that conclusions are based on evidence such that when viewed by others the same result should be derived. Often the property price is perceived as a random variable, this induces the notion that there are no "true value" of the properties price, rather there are multiple prices that are possible with varying probability.
The heterogeneity in the market gives rise to models based on price differences. This model uses selected sets of previous sales with similar characteristics, those that affect the price are identified and their values are estimated in order to calculate the price implication. This model is based on the notion that for a complex product the customer pays for the utility and the price paid is the sum of the utility of the characteristics. Two types of errors arise in this model: random variation in sales price and estimation errors for the value implication, the total error is the sum of the two. Let σ be the standard deviation of the price distribution and n the sample size of sales. So the standard deviation of the mean is $\frac{\sigma}{\sqrt{n}}$ and decreases as the

sample size increases. Due to the heterogeneity of the properties the variance σ^2 increases when the sample size increases. This leads to an error trade-off, when increasing the sample size the variance σ^2 of the sample increases while the mean of the sampling distribution decreases.
One commonly used model is the hedonic price model described in chapter 1.1.2, hedonic stems from Greek and means pleasure. The equation is written as: $y_i = \beta_j X_{j,i} + \epsilon_i$ where y_i is the i'th observed sales price $i \in 1 \dots N$, N is the number of sales, β is the implicit prices, X contains the sales information, F is the number of attributes and $j \in 1 \dots F$. Errors are captured by ϵ the error term. The fundamental principle in the hedonic price model is that buyer acquires a bundle of characteristics for which she is willing to pay a certain amount for each of them.
In the article [3] by Kummerow, he refers to economic theory that states that the long running cost relates to the value and that the supply will be adjusted until price and cost is at equilibrium. Though the adjustment is protracted due to the time required for exploration and construction of housing, which causes the market to seldom be at equilibrium, hence the cost does not equal price for a lengthy period of time. The driving factor of the short-term price is the supply and demand and the actual sales transaction is a necessity to disclose the price and cost relationship.

2.1.5 Shortcomings with contemporary real estate valuation

Shortcomings in contemporary methods are studied in the work [4] where several examples and test results gives rise to a somewhat harsh critique to the methods used in the US market. Similar critique is put forward in the article [1] where a study was performed on the UK market; this paper is discussed in section 2.1.1. The critique put forward in the paper [4] is mainly targeting the fact that systematic patterns can be found in the errors made by the repeat sales methodology which is the base of the most common methods used and foremost Case-Shiller (CS). Patterns for three types of errors can be found, these are as follows:

- Cheaper homes are predicted to have a lower value than the actual value. The opposite holds true for expensive homes.
- Systematic over-prediction is done on homes where transaction has not occurred recently. As the time between transactions increases the performance gets worse.
- In the period July to December 2007 the prices dropped significantly but CS systematically over-priced the objects.

These shortcomings of the models can be harmful to the business and society; one example of this is the sub-prime crisis.
The evaluation of the methods was performed on a data set with 367,973 single transactions and 591,239 repeat sales, all in the Los Angeles county. Some of the significant findings in this paper was that for the period 2000 to 2005 the median

actual error changed from -3.96 per cent to 3.2 per cent, a change of over 7 per cent. From 2005 to 2008 it fell by more than 8 per cent. Looking at the median error of prediction done with different period between transactions gives that transactions performed within one year has an error of 5.86 per cent. That increases for transactions within one to two years where the median error is 7.88 per cent. For the transaction interval of 15-20 years the error is staggering 17.12 per cent. Accounting for geographic information eliminates the patterns in the error with respect to the turnover time. For the authors spatio-temporal model the median error is stable between -1.5 and -2.5 per cent for the whole range of turnover time.

2.1.6 Real estate valuation using neural networks

Artificial Neural Networks (ANN) is well suited to construct models used to predict real estate values. The networks ability to catch non-linear behaviours is one of its aptitudes to predict prices of apartments and houses. Many factors affecting the housing market like inflation, social concerns and interest rates are non-linear. Another beneficial feature of the neural networks are their ability to cope with noisy data sets. In a study [5] performed on the Malaysian housing market based on data from a time period of three years (1994-1996) in total 300 sales (45 used for validation and 15 for testing), a MLP was used to predict the prices of terrace houses using nine features (sample year, land area, type of house, ownership type, build area, age of house, distance from city, environment and building quality). The network was configured with nine input nodes corresponding to the input features, a hidden layer comprising of 3 to 7 nodes and a single predicting output node. Using this method a root mean square error of 0.061717 was obtained for a MLP with five hidden units and a linear activation function. Both the hyperbolic tangent function and the sigmoid function produced results with higher error rate, though this outcome is somewhat unexpected. The results obtained in the study indicates that the MLP is well suited for the task of predicting house prices and that it can outperform a multiple regression based algorithm.
A more elaborate approach is discussed in the paper [1] where a Gamma Test [1] (GT) is used to drive a Genetic Algorithm (GA) in the process of selecting useful features from a data set. This data was then used to train a ANN on the root mean squared error produced by the GT. Without any prior knowledge of the system the Gamma Test is able to estimate the noise level in the data, a quality measure that indicates whether a smooth model exists or not. This approach has the advantage of being able to handle a dataset with large numbers of useful inputs but with high levels of noise or sparse data. Finding a good combination of features is an optimization problem and here a Genetic Algorithm is used to search for the optimal feature set. To find the optimum the GA uses a population of individual's who each has their own genome describing their characteristics. An objective function is used to estimate

[1]The Gamma Test is a data analysis routine that pursues to estimate the best Mean Squared Error that can be achieved by an continuous data model. Further discussions regarding the Gamma Test can be found in the paper [6]

the individual's quality with respect to the problem at hand. The algorithm evolves as the generation's progresses. For every generation the population is adjusted where lower ranking individuals are excluded and new created by a mating process where genome parts are exchanged (crossover) between spouses and with a small probability a mutation is performed. To be able to mate the individual has to succeed in a tournament. Genetic Algorithms are further described in section 3.4. Here the genome is a Boolean mask indicating whether the column of the data should be in the feature set or not. The objective function is quite complex in this paper but in essence it calculates the Gamma statistics for the given genome, hence the included columns weighted Gamma value in combination with weighted measures of: the amount of noise, complexity of underlying relationships between output and input date and finally the complexity of the ANN layout.
The data sets were not pre-processed before put to use in the Gamma test, thus using the data without a priori knowledge of its characteristics. Eight economic indicators where converted to a time series of length 6, adding 48 inputs to the feature set. The results from the Gamma test shows that the more recent figures for average earnings have more significance than older and that the top ranking features: retail price index and bank rates are consistently very significant. After the selection of features from the data set done by the GA using the GT in its objective function, the resulting data set was fed into an ANN with two layers of hidden nodes with four nodes in each layer. Weight updating was performed with a learning rate of 0.25 and momentum at 0.1. The model predicted the change in the house price index with a root mean square error of 9.6 per cent.

2.1.7 Condominium price estimation using open data

Price estimation using support vector regression (SVR) and Open Data from the New York condominium market is explored by [7] Arul and Morales in their paper. Initially they use a 10 dimensional data set with features related to the price containing 4,950 data points from 2011 and 2012. Their principal components analysis yielded 7 features that could account for 98.3% of the data, high-ranking features were: building classification, construction year, the building's gross income and expense per square foot. The SVR model based on the 10 open data features predicted prices with an average error of 38.2 per cent, further 36.11 per cent of the predictions where within 15 per cent of the actual price. In order to improve the predictions GPS based location data was included in to the data set as a distance to an origin point. This in combination with a feature selection gave an average error of 21.8 per cent where 49.1 per cent of the predictions fell into the 15 per cent range of the actual price. In the conclusion the authors suggest inclusion of features regarding: crime data, school district data and socio-economic data.

2.2 Literature in the field perceptrons and machine learning

In this section, we discuss several improvements and known techniques to enhance the performance of the Multilayer Perceptron and the backpropagation algorithm. The improvements in performance are reflected in the precision of the model and the running time of the backpropagation algorithm.

2.2.1 Difficulties training neural networks

One of the pitfalls when training Neural Networks is to select a good regime for weight and bias initialization in combination with the activation function. These issues are covered by X. Glorot and Y. Bengio in their paper [8] where they study the effect of random initialization of weights and how they effect the gradient descent algorithm. The Sigmoid activation function (see subsection 3.3.1) has a non-zero mean and this is known to cause singular values in the Hessian matrix. They further discovered that the weights associated with the last hidden layer rapidly obtains their saturation value of 0 and that this situation can last very long. This makes the Sigmoid activation function less suitable for MLP:s and especially used in combination with the traditional initialization schema described by equation 3.1 in chapter 3.3.2. The recommendation is to use the hyperbolic tangent (see subsection 3.3.1) or softsign activation function in combination with the so-called normalized initialization, see equation 3.2 in chapter 3.3.2. Effects of changing the schema for weight initialization are explored in chapter 5.3.3. Further Glorot and Bengio conclude that the normalized initialization is well suited for the hyperbolic tangent activation function.

2.2.2 Using dropout on hidden nodes

In the paper [9] Improving neural networks by preventing co-adaptation of feature detectors G. Hinton et al. describes the regime of using "dropout" to avoid "over-fitting". This is achieved by randomly omitting 50 per cent of the training data on the hidden nodes for each training case. When the model is used to predict e.g. verification or test data the outgoing weights has to be compensated by dividing them by 2, this because when predicting all the nodes are on unlike in the training phase. Additional improvement can be obtained by randomly dropping out 20 per cent of the inputs as shown in their paper. This regime can also be viewed as a method of averaging different models. Significant improvement is shown for this method on e.g. the MNIST data set.

2.2.3 Inductive Conformal Prediction

Presenting results from a predictive model without mentioning anything about the reliability of the data gives the receiver low confidence in the figures and therefore

no usable information in the worst case. We would like to combine the result with some confidence values to give the consumer the ability to assess the validity of the prediction. In 1999 a new approach called Conformal Prediction (CP) was proposed which are built on top of traditional machine learning algorithms, in the context of CP called underlying algorithms. The underlying algorithms are used to calculate a measure of confidence and credibility. Though the CP has one big disadvantage and that is its computational inefficiency, which in the case of Neural Networks render it almost infeasible to use.
The shortcoming of the original conformal prediction algorithm is its transductive inference property, which requires that the calculations have to be restarted from the beginning for each test case. A solution to this problem was presented in the article [10] where the authors propose a shift to inductive inference, henceforth called ICP. This algorithm lifts this restriction and adds moderate overhead to the total computation making it usable for practical applications. The idea behind this regime is as follows, given a training set $TS = (x_1, y_1), \ldots, (x_l, y_l)$ of size l where x_i are attributes and y_i the labels, partition it into two subsets: proper training set $PT = (x_1, y_1), \ldots, (x_m, y_m)$ of size m where $m < l$ and a calibration set $CS = (x_{m+1}, y_{m+1}), \ldots, (x_l, y_l)$ of size $k = l - m$. Employ the underlying algorithm to the proper training set PT and for each example $(x_i, y_i) \in CS$ calculate the non-conformity score α_i:

$$\alpha_i = \mid y_{m+i} - o_{m+i} \mid, i = 1, \ldots, k \tag{2.1}$$

for the unlabelled example y this becomes

$$\alpha_{k+1} = \mid y - o_{l+1} \mid \tag{2.2}$$

Thus the prediction o_i for the example x_i has to be computed using the underlying algorithm and than together with the label y_i and using 2.2 to finally compute the non-conformal score α_i. The concept of the non-conformity score is further discussed in subsection 2.2.3. The p-value of the potential label y is defined as:

$$p(y) = \frac{\#\{i = 1, \ldots, k+1 : \alpha_i \geq \alpha_{k+1}\}}{k+1} \tag{2.3}$$

Here $\#A$ is the cardinality of the set. The predictive region constructed by the ICP is $\{y : p(y) > \delta\}$ where $1 - \delta$ is the confidence level and δ the significance level. We construct the sorted set $\Psi_\alpha^{k*} = \alpha_{(1)}, \ldots, \alpha_{(k*)}$, sorted in descending order from the α_i values corresponding to the calibration set CS. Let

$$j_s = \#\{\alpha_i : \alpha_i \geq \alpha_{(s)}\}, s = 1, \ldots, k* \tag{2.4}$$

be the number of α_is larger or equal to the elements in Ψ_α^{k*}. Henceforth the s will be denoted ICP_{index}. The predictive region is defined as

$$(o_{l+1} - \alpha_{(ICP_{index})}, o_{l+1} + \alpha_{(ICP_{index})}) \tag{2.5}$$

so we need to find the ICP_{index}. This can be derived from the equation

$$\delta - \epsilon = \frac{j_s}{k+1}, s = 1, \ldots, k* \tag{2.6}$$

where $ICP_{index} = s$ for the s yielding the smallest ϵ.
The above algorithm can be used in two different modes:

- Find a predictive region for a given significance level δ with the property that we can be $1 - \delta$ confident that the true result is within the region.
- Find the maximum level at which we can be confident that the true result is within a given fixed region.

The first case corresponds to the regression ICP described above. For the later case we have a given interval $[a, b]$ so $\alpha_{(s)} \geq max(\mid o_i - a \mid, \mid o_i - b \mid)$ for the largest s. This gives the maximum confidence interval that is given by $1 - \frac{j_s}{k+1}$.
The authors conclude the paper [10] by examining the performance of the ICP versus two different TCP algorithms. Here the ICP outperform the first variant but for the second it performs worse but looses with a small marginal. One of the explanations for this is that the training data set presented to the underlying algorithm is smaller for the ICP due to the fact that some of the data has to be put aside for the calibration set and those gives the underlying algorithm a disadvantage for the ICP.

Non-conformity Measure

The non-conformity measure makes an assessment of the strangeness of every pair (x_i, y_i) in the calibration set. In the paper [10] two equations for how to calculate the non-conformity measure are presented, these equations 2.7 and 2.8 stated below.

$$\alpha_i = \mid y_{m+i} - o_{m+i} \mid, i = 1, \ldots, k \tag{2.7}$$

$$\alpha_i = \mid \frac{y_{m+i} - o_{m+i}}{\sigma_i} \mid, i = 1, \ldots, k \tag{2.8}$$

where $\sigma_i = e^{\mu_i}$ and $\mu_i = \Gamma(log(\mid y_i - f(x_i) \mid)$
here Γ is the Ridge Regression and $f(x_i)$ is the predictive function trained from the proper training set PT. Experiments presented in the paper demonstrate that the second equation 2.8 produces tighter intervals than the former 2.7.
Even more interesting non-conformal measurements with experiments relating to this work is found in [11] where six novel measurements are proposed and experiments performed on the *Boston Housing* benchmark are presented. These measurements can be divided into two groups; one based on the distance to the k nearest neighbours and the second is using the standard deviation to its neighbours. Below are two of the non-conformal measurements presented, one from each group whom

performed well on the *Boston Housing* benchmark.
The first group is based on the measurement of distance between neighbours.

$$\alpha_i = | \frac{y_{m+i} - o_{m+i}}{exp(\gamma \lambda_i^k)} |, i = 1, \ldots, k \tag{2.9}$$

where $d_i^k = \sum_{j=1}^{k} distance(x_i, x_{i_j})$ is the distance to the k nearest neighbours and

$$\lambda_i^k = \frac{d_i^k}{median(\{d_j^k : x_j \in PT\})}$$

which compares the distance d_i^k with the median of the distance from their nearest neighbour for all training examples. Here γ is used to control the sensitivity of the measure, increasing γ gives a more sensitive measure.
Equations in the second group utilize the standard deviation between neighbours to give a measure of the strangeness.

$$\alpha_i = | \frac{y_{m+i} - o_{m+i}}{exp(\gamma \xi i^k)} |, i = 1, \ldots, k \tag{2.10}$$

where
$s_i^k = \sqrt{\frac{1}{k} \sum_{j=1}^{k} (y_{i_j} - \Upsilon)}$ and $\Upsilon = \frac{1}{k} \sum_{j=1}^{k} y_{i_j}$ for a given example measures the standard deviation to its k nearest neighbours. Similar to equation 2.9 the standard deviation is divided by the median standard deviation, which gives

$$\xi i^k = \frac{s_i^k}{median(\{s_j^k : x_j \in PT\})}$$

.
The result from the *Boston Housing* benchmark are quite interesting and gives a indication on what non-conformity measurement to chose and the benefits of selecting the right one. Results of these tests are presented in table 2.2 where the differences between the different measurements are striking. It should also be noted that the ICP algorithm is slightly better on doing predictions inside the region.

Table 2.2. Results from *Boston Housing* benchmark

		Median Width			per cent outside predictive region		
Method	Measure	90%	95%	99%	90%	95%	99%
TCP	2.1	12.143	17.842	33.205	10.24	5.06	0.97
	2.9	10.897	14.468	24.585	9.92	4.92	0.91
ICP	2.1	13.710	19.442	38.808	9.47	4.88	0.91
	2.9	11.531	16.702	30.912	10.08	4.72	0.69
	2.10	10.211	14.228	34.679	9.68	4.60	0.65

The width of the predictive region is narrower for the TCP algorithm than for the ICP but the difference is not significant compared to the gain in computational

effort. The narrowest predictive region is produced by measurement 2.10 except for the 90 per cent confidence level where measurement 2.9 produces a narrower region.

Chapter 3

Method

In this chapter the methodology that was used to facilitate the making of this master thesis is described. Finding appropriate feature candidates was one of the first challenges faced in this work. There is a significant amount of papers discussing potential candidates to bring in to the model, those with similar conditions has inspired the selection of features and they are reviewed in chapter 2. When the selection of features was completed, a search for appropriate data sources begun. All the required information was found from open data sources. To facilitate the data capture and transformation a toolbox of shell- and Python scripts was created. Generation of the feature space was performed by a Scala application that rendered the features to the database and for future processing by Weka. In this stage the features were normalised and abnormal records where dismissed. For a detailed description of this process see chapter 3.2.2. The built in PCA and MLP functionality of Weka was used to select the features that contributed most to the solution of the model. Weka was enriched with a hyperbolic tangent activation function to promote the ability to capture the non-linearity in the model. Early testing indicated that hyperbolic tangent outperformed the traditional sigmoid activation function, both with respect to the performance of the model and the speed of convergence of the gradient descent algorithm. Finally the selected features were exported to Weka's native file format "arff".
Comparison with a more traditional machine learning algorithm is one of the prerequisites of this paper. To stage this a commonly used Python library named LibSVM was used. A Python script reads the Weka data file (arff), partitions it into three parts, builds the model and validates it. Tuning of the SVR was done by iterating over a separate interval for each parameter searching for the appropriate values by evaluating the model against the test data set. As the icing on the cake some additional traits were added to the multilayer perceptron calculations like: conformal predictions, random initialization of weights, adjustable learning rates and momentum. Finally the parameters for the MLP algorithm were fine-tuned and the definitive result is presented in chapter 5.

The main features of the work flow were as follows:

- Identify potentially useful features to include in the model
- Find data sources for features
- Capture data from selected sources
- Generate feature space
- Select features for model
- Compare performance between multilayer perceptron (MLP) and support vector regression (SVR)
- Incorporate additional functionality to multilayer perceptron
- Fine tune multilayer perceptron parameters
- Evaluate result

3.1 Data collection

All data sources used in this research are publicly available, so called "Open Data". The Swedish parliament has issued a directive that requires all government departments and municipalities to make their data publicly available. This is one of the major reasons behind the rich variety of information available from Stockholm City which is one of the major information sources.
The information gathering was performed incrementally starting with the retrieval of a list containing the streets of central Stockholm from the website svenskagator.se [1]. This list were read into the MySQL database handling the persistent data, see chapter 3.1.1. The list of streets were then used to fetch the final prices of sold apartments for the given time period, one a per street basis from slutpris.se [2]. For details see chapter 3.1.2.

3.1.1 Streets searched for sales

Street names from svenskagator.se [1] was read from the site using the Unix command line tool *curl* and processed by a Bourne shell script that produces a SQL file which contains insertion statements. This file was then used to load the MySQL database with the street names. In total 1.341 street names were read into the database.

3.1.2 Apartment sale statistic

Information from the REST API at slutpris.se [2] is structured according to the Json standard. A program written in Scala was used to retrieve the information from slutpris.se. For each street in the list specified in chapter 3.1.1 a separate HTTP request had to be performed, the response was retrieved, parsed and transformed into class instances that were stored in the database. The request URL and parameters for the REST call is as specified in table 3.1. Features collected from slutpris.se are described in table 3.2.

Table 3.1. Request to slutpris.se

http://slutpris.se/main_application/get_search_result/			
dateLimit	order	minrum	maxrum
minarea	maxarea	minpris	maxpris
minavg	maxavg	area	

Table 3.2. Attributes in response from slutpris.se

Feature	
Construction year of building	Building has elevator
Fireplace in apartment	Apartment has a duplex
Apartment is a penthouse	Apartment has balcony
Date of transaction	Price per square meter
Area in square meters	Number of rooms, kitchen excluded
Stores from the street level	Monthly fee payed to association
Selling price	Latitude and longitude coordinates
Street address of apartment	Name of realtor
Realtor identification	

Apartment statistics were gathered for transactions finalized in the time frame 2011-08-01 until 2013-06-08. In total about 8,900 transactions were retained from slutpris.se, they were then filtered and only apartments in the bounding box defined in table 3.3 were included. For more information regarding the generation and cleaning of the data, se chapter 3.2.2 and 3.2.1 respectively.

Table 3.3. Bounding box of centre of Stockholm

Type	Min	Max
Latitude	59.298149	59.356296
Longitude	18.021784	18.115082

3.1.3 Street information

Information retrieved from slutpris.se was enhanced with the zip code and latitude and longitude were replaced with more accurate information retrieved from the open data portal of Stockholm City, openstreetws.stockholm.se [3]. Sales statistics were sparse compared to the number of unique street addresses in the surveyed area. In this situation a lazy evaluating solution was more efficient than pre loading all the adequate data prior to the substitution. The solution was based on a caching schema that used the database to persist the retrieved information. When a street address was searched a lookup in the cache was done, if the information was available in the cache it was returned to the requester. If the street address was not in the cache it was retrieved from Stockholm's open data portal. Enhancement of the sales statistic was achieved by looping through all entries of the sales statistic and for each entry lookup the street information from the cache.

3.1.4 Historic inflation figures

Statistics over the apartments sales was derived from a two year period and therefore inflation has an impact on the sales price over time, closed sales from 2011 would appear to have an actual lower price than more resent deals. To avoid this bias on the price feature the final sales price was adjusted according to the monthly inflation up to the last date of the examined time interval. Inflation statistics was retrieved from the Statistics Sweden (SCB) who is responsible for the official inflation figures. Information regarding the inflation rate and underlying inflation rate can be fund at [4] and [5] respectively. The retrieved information was pre processed by a Python script and then read into the database.

3.1.5 Interest rates for apartment loans

There is an on-going debate regarding the interest rates influence on the sales price of apartments in Stockholm. The prerequisite for being able to examine whether the interest rate has an impact on the sales price, historical interest rates were fetched from Skandinaviska Enskilda Banken (SEB) [6] and Swebank [7]. This information was temporarily stored as CSV-files, preprocessed by a Python script and inserted into the database.

3.1.6 National election result

Political preferences of buyers, neighbours and policies of governing parties is a potential feature that affects the market. Statistics from the previous Swedish election in 2010 and geographic information of the layout of the constituency's are available from the Swedish authority Valmyndigheten [8]. This statistics were downloaded as Excel files and loaded into the database via a Python script. Geographic information regarding the constituency's layout is supplied as so called shape files. The coordinates were transformed and the GeoTool Java library is used to find out

which constituency a given apartment belongs to and its unique id. Finally this id was used to find the election figures from the database.

3.1.7 Local features

Stockholm City [9] provides data about over eighty local feature types. Fourteen of those were selected as candidates for the models feature space. Each feature has a unique identifier that has to be fetched from a catalogue using REST-calls. This catalogue was retrieved whit the Unix tool curl and stored in XML-format, a Python script reads the file en inserts the unit types into the database. A second Python script reads the entries of the selected feature types from the REST-service and stores them into the database. Each feature entry was associated with a geographic coordinate (latitude and longitude), this information was used when the feature space was generated to calculate the distance to the nearest feature of each type.

Table 3.4. REST services at api.stochlolm.se

http //api.stockholm.se	
Type	URL
unit types	/ServiceGuideService/ServiceUnitTypes
unit entry	/ServiceGuideService/ServiceUnitTypes/{id}/ServiceUnits

3.2 Features

The features can be classified into six major groups depending on their data domain and origin. The groups are in order taken from table A.1: sales information regarding the apartment, geographical reference information and from table A.2: distance to local feature units, geographic position and from table A.3: interest rates, election result from 2010. A complete listing of all the features used to create the model can be found in appendix A.

3.2.1 Feature aggregation

When the generation of the feature space was performed, all the required data had already been loaded into the database. A special export table was created where the features were gathered into a feature set with one attribute for each feature and one for the label. This table was populated with data from the previous collected information by joining rows in the adequate tables and inserting them into the flat export table. This table was then read by the machine learning tool Weka and saved as a so-called “arff” file.

3.2.2 Cleansing data

When the data had been imported to Weka it was inspected and validated. The ocular inspection of the data revealed several outlying apartment transactions. Several

transactions had exceptionally low selling price compared to the area and number of rooms in the apartment, those records were deleted in view of the fact that this implied that other means of compensation had been involved in the transaction, creating a unfavourable bias of the price. This lead to the removal of apartments with a price per square meter less than 35,000 SEK, because this is a highly unlikely selling price in central Stockholm. Records where the number of rooms was missing were left out together with apartments with missing construction year. Finally records with a selling price higher than 100,000 SEK per square meter were removed due to the fact that these objects were not regular apartments but rather small houses or other special accommodations.

3.2.3 Partitioning data

The data set was finally partitioned into three distinct datasets used for: training, validation and testing. Prior to the partitioning, the rows in the data set were scrambled (randomly rearranged) to avoid uneven distribution of similar data and thereby distorting the result. After studying the literature referenced in this thesis and observing commonly used data petitioning schemas, a 70%/15%/15% split ratio was selected. On our complete data set with 5,991 entries this resulted in the following partitioning.

Table 3.5. Final partitioning of data

Partition	Relative size	Number of entries
Training set	70%	4000
Validation set	15%	996
Test set	15%	995

3.3 Construction of the Multilayer Perceptron

Multilayer Perceptrons are a powerful tool used to build predictive models from a set of input data containing a given number of features and as result predict one or more target variables. The topology of a MLP is simple and straightforward, see figure 3.1. The network is divided into layers, which comes in three flavours: input layer, hidden layers and output layers. For each feature in the input data a separate node is created in the input layer. One or more hidden layers that can contain different number of nodes each and finally an output layer with one or more nodes follow this layer. In this paper a single output node was used to produce the estimation of the apartment price (regression). When the MLP model is used for classification a 'Soft Max' is created which consists of one node for each expected class.
Each node in the lower layer is connected to all nodes in the layer above, thus forming a complete bipartite graph. A weight is associated with the connection, for example w_{hi} in figure 3.1. The output of a node in the network is calculated with an

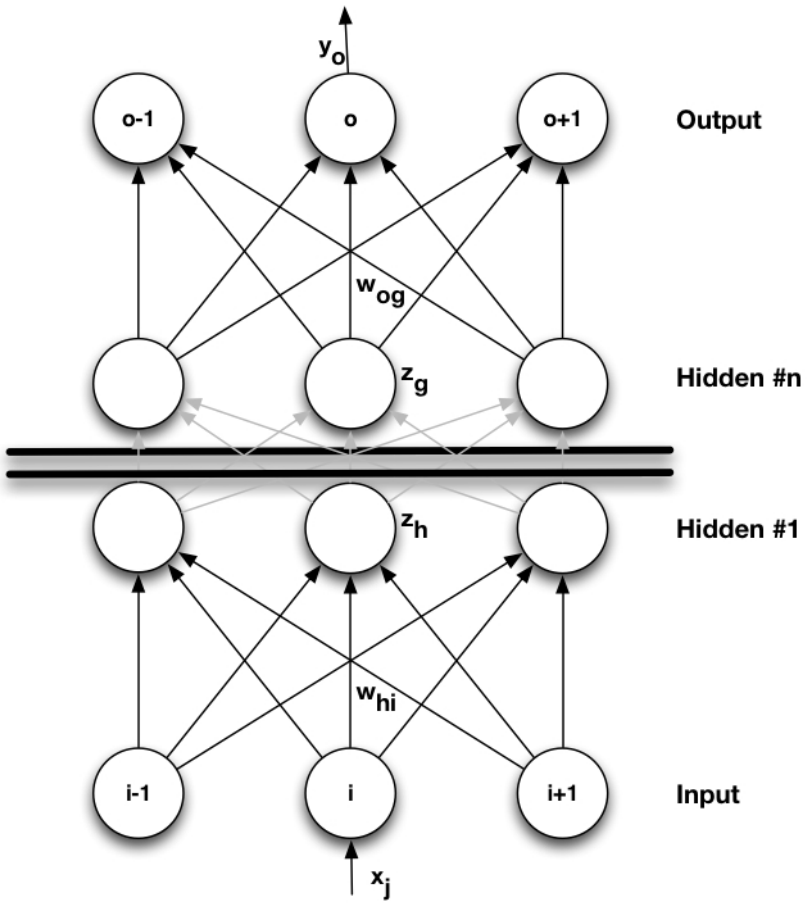

Figure 3.1. Layout of *Multilayer Perceptron.*

activation function which input is the weighted sum of the incoming connections. Activation functions are covered in section 3.3.1. An error function is used to calculate the difference between the output from the network and the target (desire) value, this is typically the mean square error function $E = \frac{1}{2}(t - y)^2$ but other functions are also used. In this paper the supervised training algorithm called back-propagation was used to create the statistical model. This algorithm can briefly be described as follows:

1. Sample from the training set is presented (input data to input nodes).

2. Inputs are propagated forward in the network by calculating output values for the nodes in each layer by applying the activation function from input nodes towards output node. *Forward propagation*

3. Output error is calculated by the error function $E = \frac{1}{2}(t - y)^2$. Here t is the target value and y is the output of the network.

4. Calculate the gradient, momentum ($M(t) = M(t-1) * \lambda - \textit{gradient}$) and update weights ($W(t) = \alpha * M(t)$), here λ = velocity decay, α = learning rate. *Backward propagation*

The algorithm described above is applied on the whole test data set and repeated for the desired number of iterations. At this point all weights are adjusted to represent a good model of the problem. Initialization of the weights is discussed in section 3.3.2. Three different regimes of weight updates are often used:

- *Online*. Weights are updated after every sample in the test dataset.
- *Batch*. Weights are updated after passing all training data.
- *Mini-batch*. Divide the training data set into chunks of equal size and update the weights after passing a chunk.

The selected regime of weight update foremost affects the speed of the learning. In section 5.3.2 the speed gain of using the mini-batch versus batch regime is explored. In some situations the learning algorithm can give rise to so called overfitting which leads to less favourable predictions for the verification set. Overfitting can be avoided by adding noise to the weights, this is future elaborated in section 3.3.4. The first assessment was to use Weka as the major platform for testing and building models. However it was soon obvious that the performance of the platform was too poor to offer a workable environment. Though Weka was used to get a feel of the dataset, test out MLP configurations and perform a principal component analyse. Several MLP configurations were tested and evaluated to gather basic data. The sigmoid activation function was not able to create good models so Weka was supplemented with a hyperbolic tangent function, which outperformed the sigmoid networks and produced good models.
To address the problem Weka was substituted for the Open Source package Octave that is more suited for the task and offering a broad repertoire of constructions and modules usable in the construction of a MLP. The resulting program far outperformed Weka and lent itself to almost effortless modification and extension of its functionality.

3.3.1 Activation function

In the early work of this thesis the problem was studied using the machine learning tool Weka to find the best path for the work at hand and what restrictions to be aware off. Ample efforts were used trying out different activation functions, assorted features sets and different parameter settings. It was obvious early on in this work that it was not feasible to use linear activation functions and that the sigmoid activation function was unable to produce good models using the features

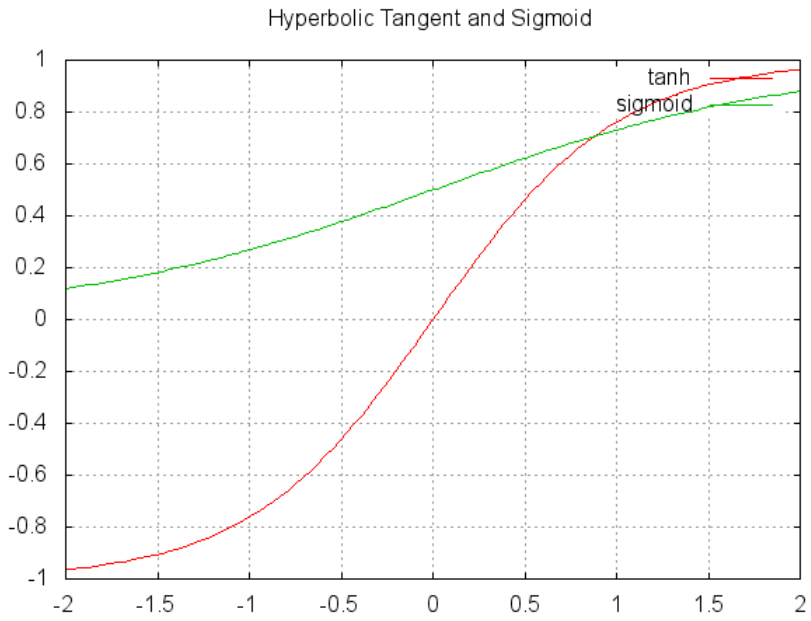

Figure 3.2. Activation functions

available in this study. To overcome this problem Weka was enriched with a hyperbolic tangent activation function in order to study its behaviour on the feature set at hand. This initiative rewarded itself by producing good predictive models far outperforming previously used activation functions. Knowledge from this early work has influenced the subsequent work by giving it a greater focus on the usage of the hyperbolic tangent activation function. The hyperbolic tangent function defined by equation 4.10 is described in subsection 4.3.3. One of the features of the hyperbolic tangent function is that it is symmetric around 0 in contrast to the sigmoid function, see figure 3.2. The equation 4.7 describing the sigmoid function can be found in subsection 4.3.2.

3.3.2 Weight and bias initialization

The most commonly used weight initialization scheme used is often referred to as regular initialization presented in equation 3.1 below.

$$W_{ij} \sim U[-\frac{1}{\sqrt{n_{i-1}}}, \frac{1}{\sqrt{n_{i-1}}}] \tag{3.1}$$

Inadequate initialization of the weight can lead to saturation problems for the weights and give negative effects upon the gradient descent algorithm used to build

the model. This can render an inexpert model that is unable to do adequate prediction. These problems are studied in the paper [8] and discussed in subsection 2.2.1. To explore this potential shortcoming the normalized initialization schema was introduced and evaluated in subsection 5.3.3. The normalized initialization is presented in equation 3.2.

$$W_{ij} \sim U[-\frac{\sqrt{6}}{\sqrt{n_{i-1}+n_i}}, \frac{\sqrt{6}}{\sqrt{n_{i-1}+n_i}}] \tag{3.2}$$

where U[-x,x] is the uniform distribution in the interval $-x < a < x$ and n_i is the number of nodes in layer i. This holds for both equation 3.1 and 3.2.

3.3.3 Weight update regime

Multilayer perceptron models can with advantage be built with use of the Backpropagation algorithm and have Gradient Descent as one of its corner stones. Choosing a good regime of weight updating is therefore crucial. It is more rewarding to follow small but consistent gradients when updating the weights than bigger and more inconsistent ones. In this paper we used two mechanisms to refine the process of updating the weight: learning rate and momentum. The concept of adding learning rate can be viewed, as a way to control how fast the weights should be learned in an update. For data sets with redundant data the learning rate can be low though too low learning rate will slow down the learning considerable, too high rate can make the learning overshoot. It is often favourable to keep the learning rate high in the beginning and to turn it down further along in the update process.
The method of using momentum stems from the idea of adding a momentum to the current gradient in the gradient descent algorithm rather than following steepest descent. Adding a momentum based on the previous weight updates to the current gradient makes it keep going in the previous direction, a momentum, see equation 3.3.

$$\mathbf{v}_t = \alpha \mathbf{v}_{t-1} - \epsilon \frac{\partial E_t}{\partial \mathbf{w}_t} \tag{3.3}$$

$$\Delta \mathbf{w}_t = \mathbf{v}_t \tag{3.4}$$

$$\Delta \mathbf{w}_t = \alpha \Delta \mathbf{w}_{t-1} - \epsilon \frac{\partial E_t}{\partial \mathbf{w}_t} \tag{3.5}$$

Weight update can be expressed in terms of the velocity, see equation 3.4. Expressing the update in terms of previous weight update gives the equation 3.5. This combined with the learning rate gives the final update function $\lambda \Delta \mathbf{w}_t$ where λ is the learning rate and α the momentum multiplier.

3.3.4 Dropout regime

The regime of dropout is described in subsection 2.2.2 and steams from the work in the paper [9]. Dropout is used to avoid overfitting the network and is accomplished by leaving out some of the weight updates performed by the backpropagation algorithm. The results of applying dropout to our model are further discussed in subsection 5.3.3 where the impact is explored.

3.4 Optimization with Genetic Algorithm

Genetic Algorithms henceforth called GA for short are suited for solving optimization problems by an adaptive search method mimicking the evolutionary process used by nature. The idea is to let a population evolve over many generations, improving every generation by selecting the best individuals and let them mate, the individuals that performs the worst are taken out of the population. The performance of an individual is measured by the objective function. To be given the opportunity to mate the individual has to participate in a tournament and outperform its competitors. Mating means that the two parties participating split their DNA at a randomly selected crossover point and the parts are interchanged. The process is repeated for a predetermined amount of generations.

3.4.1 Genome

The genome is the heart of the GA algorithm and is the principal information carrier. It is a coded form of the parameter space that is to be searched. Each parameter has to be transformed to a sequence of bits that is transferred to its specific position within the genome. A parameter can be viewed as a chromosome making out the parts of the genome. When the GA is initiated, genomes of the population are generated from randomly generated bit arrays. Three operations are possible on a genome: partitioning, composition and mutation, these operations are further described in subsection 3.4.3. The mapping of the genome used in this paper is described in table 5.11.

3.4.2 Objective function

The objective function determines the fitness of the individual; it evaluates the result of the underlying algorithm. First the genome has to be transformed from the bit-patterns representation into parameters used by the underlying algorithm. When the parameters are transformed, the underlying function is called and the result processed to give a fitness value that is returned to the GA.

3.4.3 Crossover and mutation

Crossover is mimicking natures reproduction process, genome material from two spouses are mixed to form a new individual that hopefully has new improved qual-

ities. Mutation is nature's way to alter the course of evolution, adding some randomness to the process, leading the new generations into an unexpected path. The crossover and mutation functions need three operations to perform their task. These operations are:

- Split the genome at a specific split point.
- Compose two genome parts to a complete genome.
- Mutation of a genome, changing random bits in the genome. A parameter controls how often a permutation occurs.

In the mating process the two spouses genomes are split at a so called split point, then one part from each partitioned genome is interchanged and then composed to a new complete genome. In this way two new offspring's are created in the mating process. Mutation is rarely applied to a child and is determined by a GA-parameter.

3.4.4 The search process

The Genetic search algorithm first initiates the environment and then enters a loop performing six steps for each generation, this continues for a given number of generations.

- Measure the fitness of all individuals in the population. This involves evaluating the objective function for those individuals (children) that have not yet been evaluated.
- Institute a tournament to find out which individuals are empowered to mate, controlled by parameter.
- Remove unfit individuals, controlled by parameter.
- Perform crossover and possibly mutations.
- Add offspring to the generation.
- Start a new generation.

After the final generation the optimal solution is the highest ranking individual and the genome is the optimal parameter setting.
In this paper we used a Genetic Algorithm developed in Python by Google named *Deap*. The objective function calls an Octave program that runs the MLP with the given parameters and returns the RMS error or a linear combination of RMS error and the width of the confidence interval. In the final generation the best MLP configuration can be extracted from the highest ranking individuals genome.
The Genetic Algorithm was used to find appropriate parameter settings for the backpropagation algorithm used to build the MLP model and to find parameters for the SVR algorithms. Throughout the process the same basic GA was used with

the modification of the genome and objective function that needed to be adopted to the different parameter sets. Results from the trials performed with the GA are discussed in section 5.4.

3.5 Conformal Prediction

References to theoretical work on conformal production can be found in subsection 2.2.3 where the process is outlined. Generating the ICP interval was done via the underlying algorithm, in this case this was the well known backpropagation algorithm. The phase generating the ICP interval was performed as a regular model build with the exception that the training data for the ICP model, here 3 per cent or 120 examples was set aside from the initial training set resulting in a calibration set of 120 examples and a proper training set with 3,880 examples. When building the ICP model the backpropagation algorithm was allowed to use half of the regular model's iterations or a maximum of 2,000, whatever occurs first. In the final test with the genetic algorithm the parameters used to build the ICP model was given its own genome space so the parameters for the ICP model was totally decoupled from the regular model build.

Chapter 4

The mathematics of Backpropagation

The model consists of L feature variables, this is the same number as input neurons, figure 4.1 shows the configuration of the network. Each record in the training data is comprised of the feature variables $\mathbf{x} = \{x_{(1)}, x_{(2)}, \ldots, x_{(L)}\}$ and a target variable y. The training set consists of M tuples as follows:
$\mathbf{T} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(M)}, y^{(M)})\}$

4.1 Layout of the neural network

This paper will mainly cover multilayer perceptrons with input nodes, one or two hidden layers and a regression output node. Note that only regression output will be used so the output unit is linear and no Softmax will be included.
Let I denote the number of output neurons and H,G the number of hidden units, see figure 4.1. Finally the number of input neurons is given by L.

4.2 Error function

The error function is used to measure the error between the actual value and the prediction. Define the error function E (equation 4.1) as the sum of the squared difference between the expected and calculated output , $n \in trainingset$. Nota bene that the error function for the regression case is different from the function used with classification.

$$E = \frac{1}{2} \sum_{n} (t^{(n)} - y^{(n)})^2 \tag{4.1}$$

Taking the derivative of E (equation 4.1) with respect to the weights gives us the following function:

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_{n} \frac{\partial y^{(n)}}{\partial w_i} \frac{\partial E}{\partial y^{(n)}} = - \sum_{n} \frac{\partial y^{(n)}}{\partial w_i} (t^{(n)} - y^{(n)}) \tag{4.2}$$

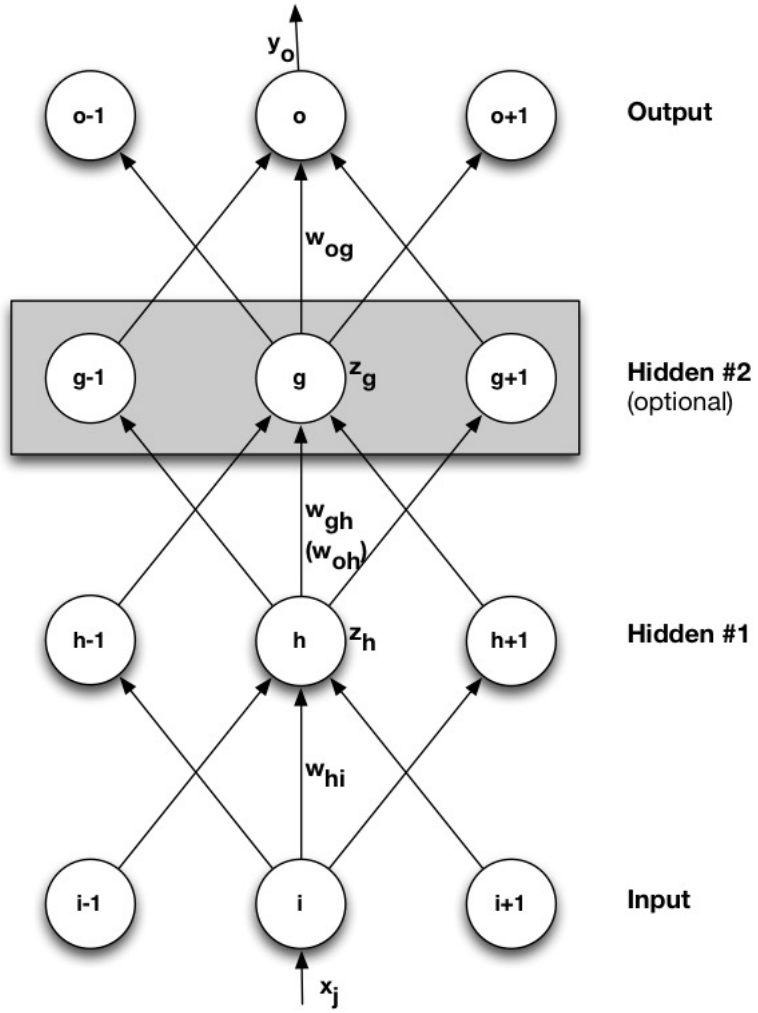

Figure 4.1. Layout of *neural network*. Note that the hidden layer in the lower part of the figure are optional, both network with one and two hidden layers will be discussed.

We can now form the batch delta rule Δw_i as

$$\Delta w_i = -\epsilon \frac{\partial E}{\partial w_i} = \sum_n \epsilon \frac{\partial y^{(n)}}{\partial w_i}(t^{(n)} - y^{(n)}) \tag{4.3}$$

4.3 Activation functions in the nodes

In this paper we are using three different activation functions. In the hidden nodes we use either a sigmoid or hyperbolic activation function. The output regression node is linear. Here we present the equations for the activation functions used and their derivatives.

4.3.1 Output neuron (Linear)

The linear neurons activation function is defined as

$$y = \sum_i x_i w_i \tag{4.4}$$

The partial derivatives of the linear activation function are

$$\frac{\partial y}{\partial w_i} = x_i \tag{4.5}$$

and

$$\frac{\partial y}{\partial x_i} = w_i \tag{4.6}$$

4.3.2 Logistic neuron (Sigmoid)

Define the activation function for the logistic neuron as

$$y = \frac{1}{1 + e^{-z}} \tag{4.7}$$

where

$$z = b + \sum_i x_i w_i \tag{4.8}$$

The derivative of y is $\frac{dy}{dz} = y(1 - y)$
Partial derivatives for z are $\frac{\partial z}{\partial w_i} = x_i$ and $\frac{\partial z}{\partial x_i} = w_i$. Now can the partial derivative of y with respect to w_i be defined

$$\frac{\partial y}{\partial w_i} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i} = y(1 - y)x_i \tag{4.9}$$

4.3.3 Hyperbolic tangent neuron

Hyperbolic neuron is defined as

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{4.10}$$

where

$$z = b + \sum_i x_i w_i \tag{4.11}$$

The derivative of y is $\frac{dy}{dz} = (1+y)(1-y)$
Partial derivatives for z are $\frac{\partial z}{\partial w_i} = x_i$ and $\frac{\partial z}{\partial x_i} = w_i$. From this follows the partial derivative of y with respect to w_i is

$$\frac{\partial y}{\partial w_i} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i} = (1+y)(1-y)x_i \tag{4.12}$$

and that the partial derivative of y with respect to x_i is

$$\frac{\partial y}{\partial x_i} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x_i} = (1+y)(1-y)w_i \tag{4.13}$$

4.4 Finding the gradients for the error function

In this section the gradient for the error function both for multilayer perceptrons with singe and dual layers of hidden units is defined. The notation is based on the network configuration shown in figure 4.1.

4.4.1 Single hidden layer with hyperbolic activation function

We need to find $\frac{\partial E}{\partial w_{oh}}$ and $\frac{\partial E}{\partial w_{hi}}$ in order to perform the calculations required by the back propagation algorithm. As before ϵ is the learning of the gradient decent. We use Δw_{oh} and Δw_{hi} to update the weights w_{oh} and w_{hi} respectively. N is the number of observations in a minibatch and $n \in minibatch$.
For the first weight between the output node and the hidden unit we need to calculate Δw_{oh}.

$$\Delta w_{oh} = \epsilon\frac{\partial E}{\partial w_{oh}} = \epsilon\frac{\partial E}{\partial y_o^{(n)}}\frac{\partial y_o^{(n)}}{\partial w_{oh}} \tag{4.14}$$

From equation (4.2) we get $\frac{\partial E}{\partial y}$ and from equation (4.5) we get $\frac{\partial y}{\partial w_{oh}}$. This gives us the solution for equation (4.14) as

$$\Delta w_{oh} = \epsilon\sum_n (t^{(n)_o} - y_o^{(n)})z_h^{(n)} \tag{4.15}$$

The delta for the weights between the input node and the hidden layer is given by Δw_{hi}.

$$\Delta w_{hi} = \epsilon\frac{\partial E}{\partial w_{hi}} = \epsilon\frac{\partial E}{\partial y_o^{(n)}}\frac{\partial y_o^{(n)}}{\partial z_h^{(n)}}\frac{\partial z_h^{(n)}}{\partial w_{hi}} \tag{4.16}$$

From equation (4.2) we get $\frac{\partial E}{\partial y}$ and from equation (4.6) we get $\frac{\partial y}{\partial z_h}$. The final part $\frac{\partial z_h}{\partial w_{hi}}$ can we obtain from equation (4.12). This gives us a solution for equation (4.16)

as

$$\Delta w_{hi} = \epsilon \frac{\partial E}{\partial y_o^{(n)}} \frac{\partial y_o^{(n)}}{\partial z_h^{(n)}} \frac{\partial z_h^{(n)}}{\partial w_{hi}} = \epsilon \sum_n \sum_o (t_o^{(n)} - y_o^{(n)}) w_{oh} (1 + z_h^{(n)})(1 - z_h^{(n)}) x_i^{(n)} \quad (4.17)$$

4.4.2 Dual hidden layers with hyperbolic activation function

For the dual hidden layer configuration we are going to use the two delta weights $\frac{\partial E}{\partial w_{og}}$ (previously named $\frac{\partial E}{\partial w_{oh}}$) and $\frac{\partial E}{\partial w_{gh}}$ (previously named $\frac{\partial E}{\partial w_{hi}}$) from previous section 4.4.1 and one additional weight between the input nodes and lower hidden layer $\frac{\partial E}{\partial w_{hi}}$, for the notations see figure 4.1.
The delta for the weights between the output units and the upper hidden layer, Δw_{og} is given by equation (4.14) but with the indexes renamed.

$$\Delta w_{og} = \epsilon \frac{\partial E}{\partial w_{og}} = \epsilon \frac{\partial E}{\partial y_o^{(n)}} \frac{\partial y_o^{(n)}}{\partial w_{og}} = \epsilon \sum_n (t_o^{(n)} - y_o^{(n)}) z_g^{(n)} \quad (4.18)$$

The same holds for the delta weights between the two hidden layers Δw_{gh} that can be obtained by rewriting the indexes of equation (4.16) with the indexes renamed.

$$\Delta w_{gh} = \epsilon \frac{\partial E}{\partial w_{gh}} = \epsilon \frac{\partial E}{\partial y_o^{(n)}} \frac{\partial y_o^{(n)}}{\partial z_g^{(n)}} \frac{\partial z_g^{(n)}}{\partial w_{gh}} = \epsilon \sum_n \sum_o (t_o^{(n)} - y_o^{(n)}) w_{og} (1 + z_g^{(n)})(1 - z_g^{(n)}) z_h^{(n)} \quad (4.19)$$

For the final delta weight Δw_{hi} we have to perform some additional work to solve the equation. We need to find the partial derivative $\frac{\partial z_g^{(n)}}{\partial z_h^{(n)}}$. Using the equation (4.13) we can get

$$\frac{\partial z_g^{(n)}}{\partial z_h^{(n)}} = (1 + z_g^{(n)})(1 - z_g^{(n)}) w_{gh}$$

Now we can find the equation for Δw_{hi} using the above partial derivative

$$\Delta w_{hi} = \epsilon \frac{\partial E}{\partial w_{hi}} = \epsilon \frac{\partial E}{\partial y_o^{(n)}} \frac{\partial y_o^{(n)}}{\partial z_g^{(n)}} \frac{\partial z_g^{(n)}}{\partial z_h^{(n)}} \frac{\partial z_h^{(n)}}{\partial w_{hi}} =$$
$$\epsilon \sum_n \sum_o \sum_g (t_o^{(n)} - y_o^{(n)}) w_{og} (1 + z_g^{(n)})(1 - z_g^{(n)}) w_{gh} (1 + z_h^{(n)})(1 - z_h^{(n)}) x_i^{(n)} \quad (4.20)$$

4.5 Matrix calculations for the MLP

Let H_1 and H_2 be the number of hidden units for layer 1 and 2. The number of inputs are represented by I and the number of observation in a minibatch is N. For regression output, as in this paper the number of output units is equal to one and is denoted by O.
Let $\mathbf{X}$ be a $I \times N$ matrix of the input features in a minibatch, $\mathbf{Y}$ and $\mathbf{T}$ are matrices of size $O \times N$ and denotes the predicted value and target value respectively.

4.5.1 Single hidden layer with hyperbolic activation function

Let $\mathbf{\Theta}$ be the $O \times H_1$ matrix of weights on the connection from hidden units to output units and $\mathbf{W}$ is the $I \times H_1$ matrix with weights on the connections from input units to the hidden units. The output of the hidden unit is represented as a $H_1 \times N$ matrix $\mathbf{\Psi}$ for a single minibatch.
Now we can represent the equation (4.15) in vectored form as

$$\frac{1}{N}(\mathbf{Y} - \mathbf{T})\mathbf{\Psi}^{\mathbf{T}}$$

and for equation (4.17) we get

$$\frac{1}{N}(\mathbf{\Theta}^{\mathbf{T}}(\mathbf{Y} - \mathbf{T})) \circ (\mathbf{1} - \mathbf{\Psi}) \circ (\mathbf{1} - \mathbf{\Psi})\mathbf{X}^{\mathbf{T}}$$

4.5.2 Dual hidden layers with hyperbolic activation function

Now let $\mathbf{\Theta}$ be the $O \times H_2$ matrix of weights on the connection from the second layer of hidden units to output units and $\mathbf{W}$ is the $I \times H_1$ matrix with weights on the connections from input units to the first layer of hidden units. The output of the first hidden unit is represented as a $H_1 \times N$ matrix $\mathbf{\Upsilon}$ and a $H_2 \times N$ matrix $\mathbf{\Psi}$ for the second layer. Let $\mathbf{\Xi}$ be the $H_1 \times H_2$ matrix of weights on the connections between the hidden layers.
Now we can represent the equation (4.18) in vectored form as

$$\frac{1}{N}(\mathbf{Y} - \mathbf{T})\mathbf{\Psi}^{\mathbf{T}}$$

and for equation (4.19) we get

$$\frac{1}{N}(\mathbf{\Theta}^{\mathbf{T}}(\mathbf{Y} - \mathbf{T})) \circ (\mathbf{1} - \mathbf{\Psi}) \circ (\mathbf{1} - \mathbf{\Psi})\mathbf{\Upsilon}^{\mathbf{T}}$$

and finally for equation (4.20) we get

$$\frac{1}{N}\Xi((\mathbf{\Theta}^{\mathbf{T}}(\mathbf{Y} - \mathbf{T})) \circ (\mathbf{1} - \mathbf{\Psi}) \circ (\mathbf{1} + \mathbf{\Psi})) \circ (\mathbf{1} - \mathbf{\Upsilon}) \circ (\mathbf{1} + \mathbf{\Upsilon})\mathbf{X}^{\mathbf{T}}$$

If we take the weight decay into account the final matrix representation for equations (4.18, 4.19, 4.20) becomes

$$\frac{1}{N}(\mathbf{Y}-\mathbf{T})\mathbf{\Psi}^{\mathbf{T}}+\lambda\Theta$$

$$\frac{1}{N}(\mathbf{\Theta}^{\mathbf{T}}(\mathbf{Y}-\mathbf{T}))\circ(\mathbf{1}-\mathbf{\Psi})\circ(\mathbf{1}-\mathbf{\Psi})\mathbf{\Upsilon}^{\mathbf{T}}+\lambda\Xi$$

$$\frac{1}{N}\Xi((\mathbf{\Theta}^{\mathbf{T}}(\mathbf{Y}-\mathbf{T}))\circ(\mathbf{1}-\mathbf{\Psi})\circ(\mathbf{1}+\mathbf{\Psi}))\circ(\mathbf{1}-\mathbf{\Upsilon})\circ(\mathbf{1}+\mathbf{\Upsilon})\mathbf{X}^{\mathbf{T}}-\lambda W$$

Where $\circ$ is the element-wise multiplication operator for two equal sized matrices.

Chapter 5

Experiments and Results

In this chapter we present the results obtained during the experiments done for this thesis. In the first section 5.1 of this chapter the Support Vector Regression trials are discussed. Three different kernels have been tested and manually tuned, these are: radial, sigmoid and polynomial kernels. In the presentation of the results findings from the trials with parameters determined by the GA is included for comparison. The evaluation of the results obtained by the GA is discussed in subsection 5.4.1. Experiments with the Multilayer Perceptron are divided into two sections where the first discuss the tuning of the general backpropagation parameters. In the second section the findings regarding the improvements are discussed. Results from the experiments with: early stopping, mini-batch, random initialization of weights and construction of conformal prediction regions are presented. Findings from the experiment with the Genetic Algorithm used to tune parameters for the SVR and MLP are discussed in section 5.4 where the two paradigms are discussed in separate subsections. This chapter is concluded with a section summing up the important findings from the experiments.

5.1 Performance of support vector regression (SVR)

Performance of the Multilayer Perceptron is compared against a more commonly used machine learning algorithm, the support vector regression algorithm was chosen as a basis for this comparisons. Henceforth the Support Vector Machine and Support Vector Regression will be shortened to SVM and SVR respectively. Tests of the SVR performance are based on the open source library Libsvm using the Python implementation. Calculations are done using the epsilon-SVM, which uses the epsilon intensive loss function. Common parameters used by the SVR are: ϵ (epsilon) used in the loss function and the cost parameter C. To give the SVR model a fair chance three different kernels were evaluated: Radial, Sigmoid and Polynomial. These test runs are described in the tables 5.1, 5.2 and 5.3 of the following sections. Two different methods of finding the parameters were used, one with manually selected parameters and the other using a genetic algorithm. In the

first approach parameter ranges were selected manually in an initial run and with knowledge from that stage a second run was performed with narrowed parameter intervals. Each parameter where tested with about 4-5 points in the interval. The second approach was to let a genetic algorithm select the best parameter set; this procedure is described in details in section 5.4.

5.1.1 Radial Kernel

The Radial kernel is calculated using the radial basis function (RBF) $e^{-\gamma|u-v|^2}$. One kernel parameter γ is used.

Table 5.1. Finding parameters for Radial SVR

Kernel	Run	Param	Low	Hi	Best
Radial	Course	Cost param	2	512	256
		Kernel gamma	3.051758e-05	1.0	0.015625
		Loss epsilon	0.000244	1.0	0.015625
		Verify	Error in %		0.040715
			RMS error		0.055867
		Test	Error in %		0.054865
			RMS error		0.073860
	Fine	Cost param	2.853117	4.594973	3.797498
		Kernel gamma	0.016600	0.751315	0.111678
		Loss epsilon	0.002039	0.092296	0.005289
		Verify	Error in %		0.040370
			RMS error		0.055633
		Test	Error in %		0.056366
			RMS error		0.075934
	GA	Cost param	0.25	64	5.75
		Kernel gamma	0.031250	0.156250	0.085449
		Loss epsilon	0.003906	0.019531	0.008057
		Verify	Error in %		0.040241
			RMS error		0.055460
		Test	Error in %		0.056041
			RMS error		0.075436

Radial: $e^{-\gamma|u-v|^2}$, GA: population=200, generations=100, selection=50% X-over=50%, mutation=5%

From the table 5.1 it follows that the lowest verification error is produced by the GA selected parameters, giving a RMS error of 0.055460. Applying the test data on this model gives a RMS error of 0.075436 that actually is higher than what the manually selected parameters would have produced.

5.1.2 Sigmoid Kernel

The Sigmoid kernel is calculated using the equation $tanh(\gamma u^T v + c_0)$. Parameters for this kernel are γ and the coefficient c_0.

Table 5.2. Finding parameters for Sigmoid SVR

Kernel	Run	Param	Low	Hi	Best
Sigmoid	Course	Cost param	789.75	1024	1024
		Kernel gamma	0.000188	8	0.000244
		$coef_0$	0.001266	1	0.001953
		Loss epsilon	0.013719	1	0.015625
		Verify	Error in %		0.0466276
			RMS error		0.063818
		Test	Error in %		0.062559
			RMS error		0.084042
	Fine	Cost param	955.59	1692.89	1692.89
		Kernel gamma	0.000367	0.000590	0.000537
		$coef_0$	0.000865	0.001532	0.001266
		Loss epsilon	0.022095	0.032349	0.022095
		Verify	Error in %		0.045878
			RMS error		0.062007
		Test	Error in %		0.062063
			RMS error		0.082555
	GA	Cost param	0.03125	7.9688	7.8750
		Kernel gamma	0.01562	1.01562	0.01562
		$coef_0$	-0.5	0.5	-0.50000
		Loss epsilon	0.00098	1.0010	0.02344
		Verify	Error in %		0.044256
			RMS error		0.060323
		Test	Error in %		0.058019
			RMS error		0.078112

Sigmoid: $tanh(\gamma u^T v + c_0)$, GA: population=200, generations=100, selection=50% X-over=50%, mutation=5%

Results from the Sigmoid kernel are presented in the table 5.2. For this kernel the best results, based on the RMS error, are obtained for the model with parameters picked by the GA, with an RMS error of 0.060323 for the verification set and 0.078112 for the test set. This is also the over all lowest error rate for the verification sets.

5.1.3 Polynomial Kernel

For the polynomial kernel the kernel function are; $(\gamma u^T v + c_0)^d$. This kernel uses three parameters: γ and the coefficient c_0 as for the Sigmoid kernel in section 5.1.2

and with an additional parameter determining the polynomial degree.

Table 5.3. Finding parameters for Polygon SVR

Kernel	Run	Param	Low	Hi	Best
Poly	Course	Cost param	0.015625	16	0.5
		Kernel gamma	0.003906	0.25	0.03125
		Kernel degree	2	8	8
		Loss epsilon	0.007812	0.125	0.007812
		Verify	Error in %		0.0411920
			RMS error		0.0560074
		Test	Error in %		0.0579445
			RMS error		0.077401
	Fine	Cost param	0.31863	1.0	0.683013
		Kernel gamma	0.022095	0.385543	0.057309
		Kernel degree	2	8	5
		Loss epsilon	0.003284	0.239392	0.003284
		Verify	Error in %		0.040356
			RMS error		0.055638
		Test	Error in %		0.056168
			RMS error		0.075729
	GA	Cost param	0.001	4.984375	4.140625
		Kernel gamma	0.003906	1.0	0.09375
		Kernel degree	1	5	3
		Loss epsilon	0.001953	0.017578	0.010986
		Verify	Error in %		0.040886
			RMS error		0.055898
		Test	Error in %		0.058598
			RMS error		0.078522

Poly: $(\gamma u^T v + c_0)^d$, GA: population=200, generations=100, selection=50% X-over=50%, mutation=5%

Results from the tests are in the table 5.3 where it can be seen that the preferred model based on the RMS error of the verification set originates from the manually fine tuned parameter setting. This selection gives a model that generates a RMS error of 0.075729 when applying the test set.

5.2 Tuning parameters for the Multilayer Perceptron

In this section the process of finding appropriate values for the parameters controlling the calculations generating the prediction model are explored. The process of finding good parameters is iterative and requires several round trips. This fact is though not reflected in this section but we have chosen to present the results

grouped by the different parameter classes. For each table presenting the result the conditions under which it has been performed is clearly stated.

5.2.1 Finding values for learning rate and momentum

The importance of how the weights are updated is stressed in section 3.3.3 where we introduce the concepts: learning rate and momentum. Here we study the effect of these parameters on the model and determines favourable values for these. During this process the MLP configuration was fixed to use two hidden layers with 10 nods each, weight decay was set to 10^{-6}. In the initial trial the algorithm was allowed to run for 500 iterations. The parameter space for the two parameters were initially tested with the values 0.1 0.3 0.6 0.9 corresponding to the first row of table 5.4.

Table 5.4. Searching for learning rate and momentum using 500 iterations, averages over 5 runs

Test interval		Best test result		Validation error	Test error
L-rate	Moment	L-rate	Moment	RMS	RMS
0.10 - 0.90	0.10 - 0.90	0.60	0.90	0.04537	0.04840
0.10 - 0.70	0.90 - 0.90	0.70	0.90	0.04551	0.04843
0.60 - 0.70	0.85 - 0.95	0.675	0.95	0.04417	0.04738
0.65 - 0.70	0.92 - 0.98	0.675	0.94	0.04440	0.04735
0.66 - 0.68	0.94 - 0.96	0.68	0.95	0.04360	0.04686

Average over 5 runs, weight decay = 1e-6, Nodes = [10,10], Iterations = 500, Mini-batch = 194, no noise added

This test indicates that the best parameter setting is located near a momentum of 0.9 and a learning rate near 0.6. After further testing with parameters in the neighbourhood of the favourable values found in the initial test, the best values for the learning rate is 0.68 and for the momentum it is 0.95. The best results for each test are presented in table 5.4 and are given as the mean of the results from five runs.
The previous procedure was repeated using 4,000 iterations in order to study the effect on the studied parameters. From these tests it is clear that the learning rate has to be reduced to about 0.30 and with the momentum kept fixed. Increasing the number of iterations favours a lower learning rate that avoids "overfitting" the model. Results from these tests are gathered in table 5.5.

Table 5.5. Searching for learning rate and momentum using 4000 iterations, averages over 5 runs

Test interval		Best test result		Validation error	Test error
L-rate	Moment	L-rate	Moment	RMS	RMS
0.10 - 0.90	0.10 - 0.90	0.30	0.90	0.04227	0.04577
0.10 - 0.70	0.90 - 0.90	0.30	0.90	0.04227	0.04579
0.10 - 0.30	0.85 - 0.95	0.30	0.95	0.04185	0.04561
0.28 - 0.32	0.93 - 0.97	0.32	0.95	0.04190	0.04576
0.31 - 0.33	0.96 - 0.98	0.32	0.98	0.04204	0.04567
0.30 - 0.50	0.90 - 0.99	0.30	0.95	0.04181	0.04561

Average over 5 runs, weight decay = 1e-6, Nodes = [10,10], Iterations = 4000, Mini-batch = 194, no noise added

To close up this section we found that the best parameters for a long running algorithm are in the neighbourhood of 0.30 for the learning rate and 0.95 for the momentum. For configurations that use fever iterations to find a suitable model should use values in the vicinity of 0.68 and 0.95 respectively.

5.2.2 Searching for appropriate MLP configuration

Configuration of the network has a profound impact on the models performance. In this work the basic configuration of the network was fixed to four layers: input layer with 54 nodes, one for each feature, two hidden layers with configurable number of nodes for each layer and a single output node producing the regression result. In this section the challenge was to find an apposite number of nodes for the internal layers of the network. The initial trials were restricted to sixteen nodes for each of the internal layers, see figure 4.1. For this configuration the best result was obtained for the configuration with 14 nodes in the first hidden layer and 8 in the second. As shown in table 5.6 the results are quite unvarying with small difference between the best and worst results for the RMS error for the verification set. The low value for the standard deviation for the best configuration is also a sign of a better consistency among those models.

Table 5.6. Searching for node configuration for L1 and L2, averages over 5 runs

	Test interval		Result		Validation error		Test error	
Type	L1	L2	L1	L2	RMS	S-dev	RMS	S-dev
Best	4 - 16	4 - 16	14	8	0.04167	0.19e-3	0.04554	0.20e-3
Worst	4 - 16	4 - 16	4	14	0.04315	1.25e-3	0.04638	0.79e-3

Average over 5 runs, weight decay = 1e-6, learning rate = 0.30, momentum = 0.95, Iterations = 4000, Mini-batch = 194, no noise added, regular init

The ability to create a high quality model depends on the number of nodes in the network and number of iterations of the algorithm. Results of this investigation are

presented in table 5.7 where the search for best node configuration from table 5.6 has been performed with five different values for the iteration count. As expected increasing the number of iteration generates models, which have lower error rate and produces more homogeneous results.

Table 5.7. Impact of iteration length on node configuration for L1 and L2, averages over 5 runs

	Best result		Validation error		Test error	
Iterations	L1	L2	RMS	S-dev	RMS	S-dev
125	4	4	0.05745	3.92e-3	0.05937	3.71e-3
250	4	16	0.04930	1.47e-3	0.05191	1.37e-3
500	12	8	0.04437	0.29e-3	0.04772	0.39e-3
750	12	16	0.04327	0.55e-3	0.04673	0.24e-3
1000	12	8	0.04281	0.16e-3	0.04609	0.49e-3

Average over 5 runs, weight decay = 1e-6, Nodes[4-16,4-16], learning rate = 0.30, momentum = 0.95, Mini-batch = 194, no noise added, regular init

Further conclusions can be drawn from the results in table 5.7 where it shows that models with fewer nodes performs better when the number of iterations is low and contrary for models that are permitted more iterations. This is due to the fact that a more complex MLP requires additional iterations to find a good solution. Each new node contributes to the total dimension of the n-dimensional space to search with gradient descent.

5.3 Boosting multilayer perceptron performance

There are several techniques that can improve the results and speed up the running time of the backpropagation algorithm. In this section we discuss early stopping that can prevent overfitting of the model, running time can be improved by the weight update regime called mini-batch. Random initialization of the weights and selection of interval can impact which solution is found by the gradient descent algorithm and in the end affect the quality of the models prediction. Finally the concept of conformal prediction is explored, this algorithm predicts an interval in which the result will stay within with a parameterised likelihood.

5.3.1 Early stopping

Early stopping is a regime intended to reduce the problem of overfitting of the model especially if the network has more nodes than actually required to represent the learning problem. One symptom of this is that the weights start to move away from 0 particularly for the most important nodes. As the training progresses less important weights are also moving away from 0 the training error decreases but the models ability to generalize is reduced.

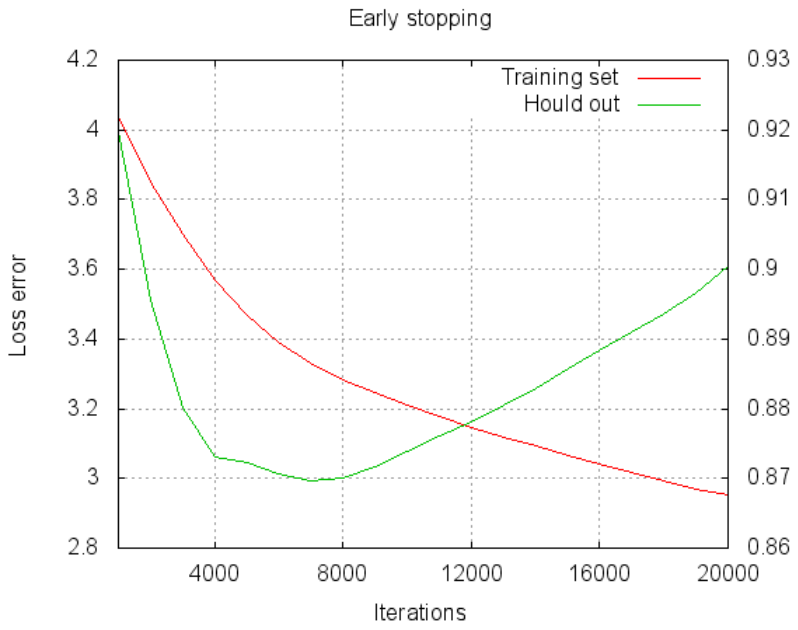

Figure 5.1. Early stopping

The idea behind early stopping is to verify the error rate on a hold out set, in this case the validation set, and stops the backpropagation when it increases. This is achieved by monitoring the output of the loss function for the validation data during training. Behaviour of the loss for training and validation sets can be seen in diagram 5.1. As long as the loss value decreases the current model are saved and the back propagation continues. If the error on the validation set increases the model is not saved. When the algorithm reaches the predetermined number of iteration the saved model (taken at the lowest seen error rate) is returned as result. So if the error rate decreases asymptotically during the whole backpropagation the final model will be returned if not the intermediary model is used.

Table 5.8. Early stopping, averages over 5 runs

Testing	Validation set error		Test set error	
	Percent	RMS	Percent	RMS
No early stopping	0.03066	0.04484	0.03306	0.04780
Early stopping	0.02940	0.04154	0.03246	0.04586

Average over 5 runs, weight decay = 1e-6, Nodes = [10,10], learning rate = 0.30, momentum = 0.95, Iterations = 50000, Mini-batch = 194, no noise added

The result of applying early stopping can be seen in table 5.8. Here the best result is obtained after about 7.000 iterations on average. The favourable effect is significant for both the test and validation set.

5.3.2 Mini-batch

Multilayer Perceptrons can be trained using one of three different schemas discussed in section 3.3. The size of the mini-batch chunks affects the speed of the algorithm, in table 5.9 the algorithm is applied with mini-batch chunks of size 97, 194, 388, 970 and 1940.

Table 5.9. Mini batch performance, averages over 5 runs

Testing	Validation error		Runtime	
Batch size	Percent	RMS error	Running time	Speed up percent
None	0.02944	0.04191	159.6 s	
1940	0.02952	0.04180	129.2 s	19.0
970	0.02958	0.04208	91.6 s	42.6
388	0.02970	0.04236	81.3 s	49.0
194	0.02936	0.04187	78.1 s	51.0
97	0.02987	0.04228	75.5 s	52.7

Weight decay = 1e-7, Nodes = [10,10], learning rate = 0.30, momentum = 0.95, Iterations = 10000, Early stopping = on, no noise added

The rightmost column holds the speed increase of the mini-batch regime over batch processing. As the chunk size of the mini-batch is decreased the root mean square error of the prediction slightly increases due to the fact that fewer samples are used in the calculation that affects the weight update.
The results of the tests shows that with a sensible selection of mini-batch chunk size, in this case 970, a good model can be produced in about half the time with a minor increase for the root mean square error. Studying figure 5.2 shows that the RMS error is relatively unaffected by the change of mini-batch size, this behaviour is the same for both the test and verification set.
If the ICP interval is taken into account the conditions changes somewhat. We can see from the figure 5.3 that a good choice of mini-batch size is 1940 with respect to the performance of the ICP-interval. This gives a speed gain of about 20 per cent without increasing the width of the ICP-interval or affects the RMS error significantly. More about the conformal prediction findings in section 5.3.4.
In some applications of the MLP processing time is limited due to hardware restrictions or business rules and those gives rise to the mini-batch approach where a good model can be built in a fair amount of time. The total runtime of the model construction can be regulated via the mini-batch size.

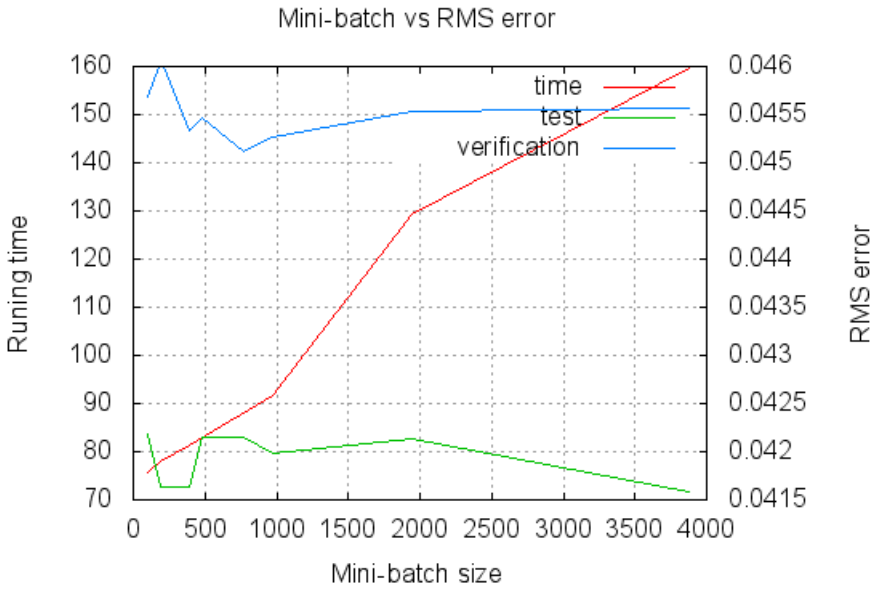

Figure 5.2. Impact of mini-batch size on RMS error

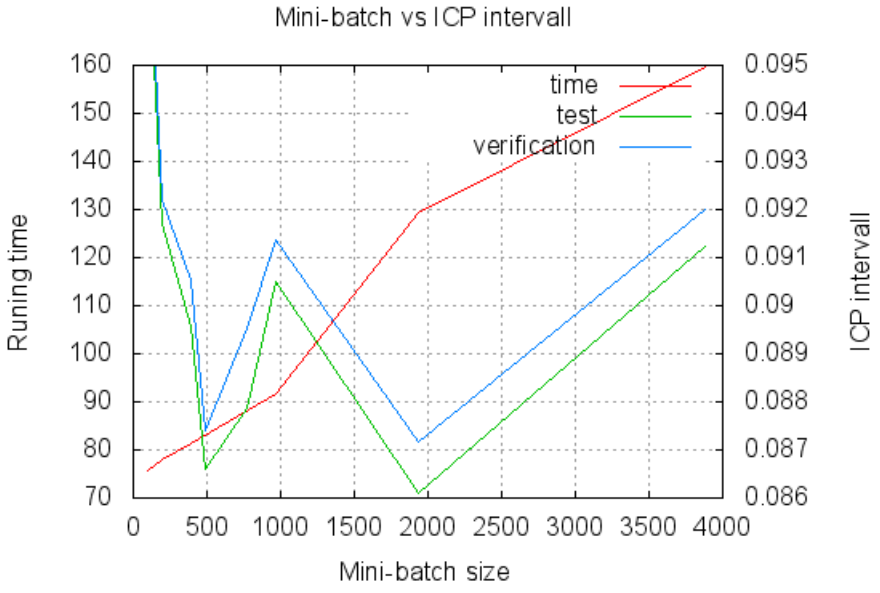

Figure 5.3. Impact of mini-batch size on ICP-interval

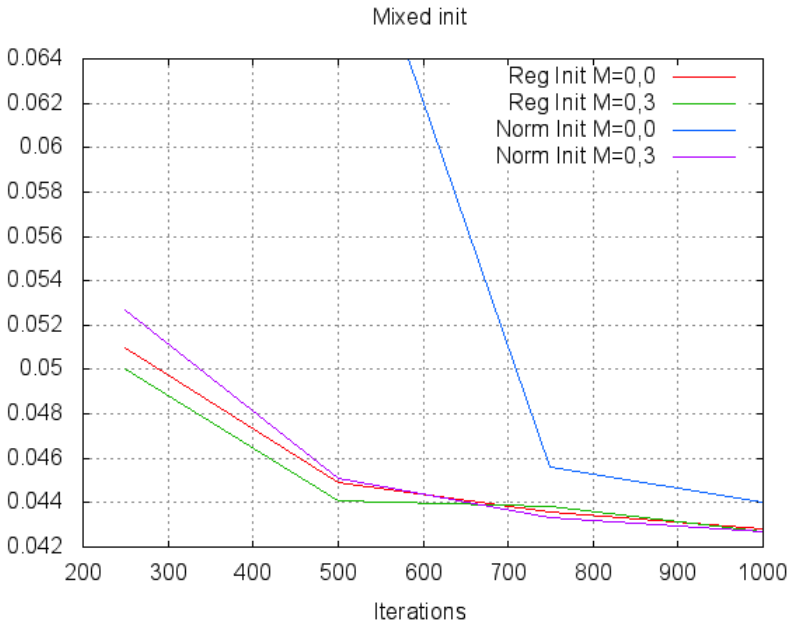

Figure 5.4. Normalized vs regular initialization

5.3.3 Random initialization of weights and dropout

In this section two different schemas for weight initialization are evaluated and the paradigm of "dropout" is investigated.
Initialization of weights is discussed in chapter 2.2.1 and steams from the work done by Glorot and Bengio in their paper [8]. The initialization of the weights is important and can have significant effect on the resulting model especial for deep networks. In this work the effect can only be elicited when the noise is added to the weights. When all weights are updated in each iteration the effect of not using the normalized weight initialization is masked by the gradient descent, which is still able to find a good solution. This topic is future discussed in chapter 3.3.2.
The regime of "dropout" is discussed in chapter 2.2.2 where the paper [9] by Hinton et al. is explored. To investigate the effect five levels of "dropout" were tested $\Psi_M : M = 0.0 - 0.4$ (M = proportion of weights left out in weight update), where $\Psi_{0.0}$ corresponds to no "dropout". In diagram 5.4 the $\Psi_{0.0}, \Psi_{0.3}$ values are presented in conjunction with two weight initialization regimes presented above. From this diagram it can be deduced that "dropout" gives an advantage when fewer iteration are used, when the amount of iterations is increased the effect fades out but has clearly a favourable effect for situations where few iteration are used. Further discussions on this topic can be found in chapter 2.2.2.

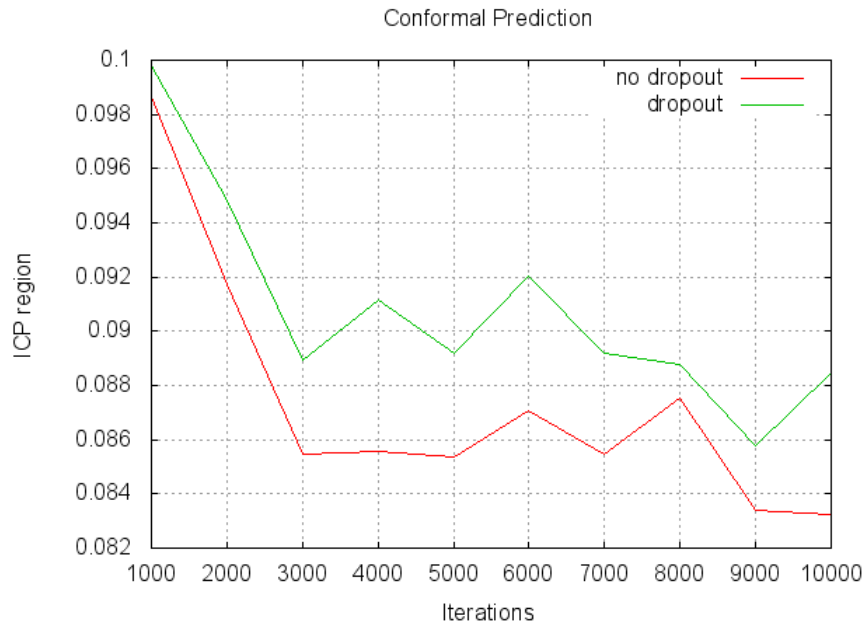

Figure 5.5. Effect of dropout on ICP interval

5.3.4 Conformal Prediction

The regime of dropout affects the calculations of the ICP interval negatively and creates a wider region compared to the calculations not using dropout, see figure 5.5. This effect is not surprising due to the fact that the calibrating set used to train the ICP model is quite small; in this context we use 120 examples. As expected the widening of the predictive region increases the amount of predictions made within the region, see figure 5.6.
After about 3,000 iterations the ICP model is able to generate tight predictive regions, about $\pm$ 0.0855 wide and with 96 per cent of the predictions within the interval. The best result is obtained after 10,000 iterations with a region about $\pm$ 0.0832 wide and predictions still within 96 per cent of the time. Note that these are average values taken over 5 runs. The single best result is a width of 0.0749 and with 94 per cent predictions within the region.
These results are promising and shows that the apparatus has the ability to find tight predictive regions and where it can make predictions within that region meeting the given confidence level.

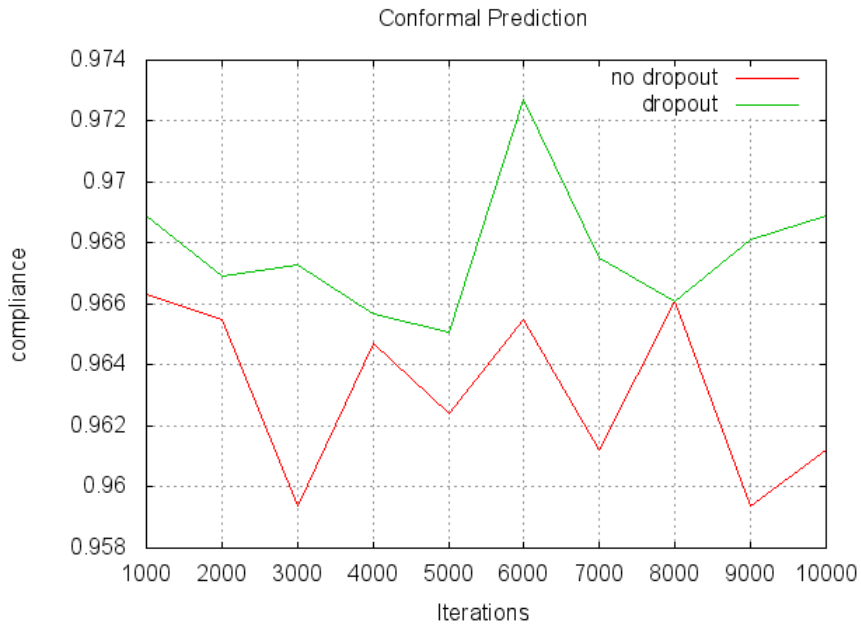

Figure 5.6. Effect of dropout on ICP confidence

5.4 Fine tuning of parameters with Genetic Algorithm

Finding appropriate parameters for the Support Vector Regression and the Multilayer Perceptron is a challenge in it self. In article [1] referred to in subsection 2.1.6 discusses the usage of genetic algorithms to find good parameters. Genetic Algorithms will henceforth often be shortened to GA. The result of the parameter search using a genetic algorithm is presented in the section below. An open source package called DEAP (Distributed Evolutionary Algorithms in Python) was used to drive the genetic algorithm in Python. The following parameter configuration was used by the algorithm: a population of 200 individuals running for 100 generations with a selection of 50%, crossover of 50%, mutation rate at 5% and a tournament size of 16. More about how the genetic algorithm was structured can be found under section 3.4 and related work using genetic algorithms are referenced in subsection 2.1.6.

5.4.1 Tuning of SVR parameters

The SVR parameters were tuned using a GA previously described. Data from these runs are presented in the tables 5.1, 5.2 and 5.3 of chapter 5.1 and summarised in table 5.10 below.

Table 5.10. Tuning of SVR Perceptron using Genetic Algorithm

	GA tuned		Manually tuned	
SVR- type	Test error	Test RMS error	Test error	Test RMS error
Radial	0.056041	0.075436	0.056366	0.075934
Sigmoid	0.058019	0.078112	0.062063	0.082555
Polynomial	0.058598	0.078522	0.056168	0.075729

From the results in table 5.10 it can be seen that no exceptional improvement is gained with the parameters generated using the GA. Performance for the SVR based on a Radial kernel was somewhat better, for the Sigmoid kernel the improvement was the highest in this trial, but the Polynomial kernel performed slightly worse for the model that used GA generated parameters. This vouches for that reasonably good parameters are found and that this results can serve as a good comparison with the results from the MLP. This gives that the MLP has to predict the outcome with a *RMS error below 0.075436* in order to outperform the SVR model.

5.4.2 Tuning of Multilayer Perceptron

In this paper we present two different schemas where the optimization is done on two different criterions. Initially we constructed a GA which goal was to optimize the error in the model. The genome was constructed out of seven parameters used to drive the MLP algorithm, this are presented in table 5.11 together with their range and segment of the genome.

Table 5.11. Translation of genome to parameter space

	Genome size	Interval		
Parameter	# bits	Lower	Upper	Winner
Nodes Layer 1	6	1	16	16
Nodes Layer 2	6	1	16	16
Weight decay	2	1e-5	1e-8	1e-5
Learning rate	7	0.0078125	0.9921875	0.15625
Momentum	7	0.0078125	0.9921875	0.984375
Add noise	1	false	true	false
Regular initialization	1	false	true	false

The first unpretentious approach to the task of using a GA to optimize the MLP parameters used the negative RMS error as objective function. Several trials where conducted to find appropriate parameter boundaries for the GA to work with. The results of the MLP models produced by the GA are presented in table 5.12. Here the best and second best results are shown for the three different approaches to MLP optimization. Studying the results from the unsophisticated trial to optimize the RMS errors shows that the model produces predictions with a low error but the conformal prediction interval is exceptionally wide, for the best result it is about

0.23 that is almost ± one quarter of the predicted result. This renders a vain conformal prediction that contributes very little confidence for the model, same situation holds for the second best result.

Table 5.12. Result from MLP model selected by GA

GA goal	Position	Set	RMS error	ICP interval	Confidence
RMS error	first	Validation	0.040941	0.245985	0.969880
		Test	0.044461	0.231206	0.972864
	second	Validation	0.041765	0.180229	0.955823
		Test	0.046044	0.194983	0.965829
ICP	first	Validation	0.042136	0.073344	0.945783
		Test	0.045737	0.074467	0.928643
	second	Validation	0.042575	0.074479	0.941767
		Test	0.046370	0.075442	0.922613
ICP and RMS	first	Validation	0.041491	0.069185	0.929719
		Test	0.045339	0.069497	0.910553
	second	Validation	0.041454	0.071395	0.935743
		Test	0.045476	0.071886	0.930653

GA configuration: population=200, generations=100, selection=50% X-over=70%, mutation=5%

The discouraging results from the first GA trial were tackled by changing the objective function to return the ICP for the MLP model. As shown in the table 5.12, this gives more confidence to the model by giving a conformal prediction interval of 0.0754 and meats the 95 per cent confidence at 92.26 per cent, still with a low RMS error of 0.04637. This result is quite useful but the fact that it only can predict the result within the conformal prediction interval 92.26 per cent of the time is still a bit disappointing.
Actually we want to find a good balance between the RMS error and the ICP interval to raise the belief for the model. This was achieved by modifying the objective function to weight the result of the conformal prediction interval (ICP) and RMS error ($ERMS$) with their expected values. The resulting objective function becomes

$$f_{obj} = \frac{1}{2}\frac{ICP}{ICP_{Expected}} + \frac{1}{2}\frac{ERMS}{ERMS_{Expected}}$$

. In the previous experiments the MLP used to build the model for the conformal prediction used the same parameters as the MLP for the predicting model. Here the GA was used to select independent parameters for the MLP building the ICP model. We expanded the genome from 24 to 48 bits and used the same parameter transformation described in table 5.11 for the second MLP calculating the results for the ICP. This constellation of perceptrons performed well and constructed a model that could predict the price with a RMS error of 0.0455 and conformal prediction interval of ±0.0719. Though note that the goal of a 95 per cent confidence was not

reached, the model could only predict results for 93 per cent of the test set within the conformal prediction interval.

5.5 Summation of results

The thesis in this paper is that apartment prices on the Stockholm market can be predicted with a Multilayer Perceptron. Results from this paper can be compared with the findings from article [5] discussed in section 2.1.6 where prediction of the real estate values in the Malaysian housing market was studied and where they as result obtained a root mean square error of 0.061717 for their MLP based model. This should be compared to the root mean square error of 0.0455 shown in this paper. To evaluate the performance of the models the result has to be translated to actual denormalised apartment prices, this is presented in table 5.13 below.

Table 5.13. Result from above experiments

Approach	Error	RMS error	ICP interval	Confidence
SVR Radial	481,000 SEK	647,931 SEK		
MLP	273,000 SEK	356,000 SEK	613,000 SEK	93.6 %

From table 5.13 we can see that the Multilayer Perceptron outperforms the Support Vector Regression and the result from the latter is too deficient to be used in many commercial applications. The results from the MLP are more promising and can predict the price with a root mean square error of about 356,000 SEK and estimate the price within an interval of ± 613,000 SEK 93.6 per cent of the time. This precision should be sufficient to be used as a tool or indicator for realtors and consumers. Note that this estimate should be compared with a realtor estimating a price without the ability to visit the apartment or get any description of it apart from the address. In this context the estimate produced by the MLP can probably compete with an experienced estate agents appraise.

Chapter 6

Conclusion

In this thesis we have shown that a predictive model estimating the price of an apartment in the central part of Stockholm using a Multi Layer Perceptron facilitated by the backpropagation algorithm can be constructed. All data used comes from "Open Data" sources available to the general public. Several enhancement techniques have been applied to the solution in order to increase the quality of the prediction and speed up the computations. Finally the result is compared with previous studies in the field using different approaches and performed on varied geographical locations.

6.1 Proceedings to improve quality and speed of backpropagation algorithm

In section 5.3.1 we show how early stopping can hinder the model from overfitting and result in a better prediction both for the test and validation set. The running speed can be cut considerably by applying the mini-batch regime of weight update. In section 5.3.2 a performance gain of 50 per cent with respect to running time is presented and with a marginal increase of the RMS error. The investigations regarding the effects of implementing normalized weight initialization and dropout could only show improvements for dropout when the number of iterations was low. For the normalized weight initialization no significant improvements could be concluded.

6.2 Benefits of using GA to find appropriate parameter settings

Tuning a MLP configuration is a challenging task and requires both skill and effort. Initial trials adjusting the parameters were performed manually to get a sense of the models behaviour. Parameters selected by the GA did outperform the manually selected ones. The improvement is not strikingly significant for the RMS error but was effective in shrinking the predictive region from 0.0832 to 0.0719 per cent.

6.3 Performance of MLP model in general

From the results established via experiments documented in chapter 5 and summarised in section 5.5 it is clear that a well performing model can be built. In this thesis paper we have established the following:

- Enriched the prediction from the MLP model with a predictive region produced by the Inductive Conformal Prediction algorithm.
- Fabricated a Multilayer Perceptron model that can predict the price of a condominium with a root mean square error of 0.0455 per cent and conformal prediction region of ± 0.0719 at the 95 per cent level. Predicting 93.6 per cent within the region. This corresponds to root mean square error of 356,000 kr and prediction interval of 613,000 kr.
- Show that the MLP can perform better than a SVR with a root mean square error of 0.0453 and 0.0759 respectively.
- Enhance the results by applying a Genetic Algorithm selecting the parameters driving the backpropagation algorithm. Improving the RMS error result from 0.0458 to 0.0455 per cent and shrinking the predictive region from 0.0832 to 0.0719.
- Demonstrated how the mini-batch regime of weight updates can improve the running time of the backpropagation algorithm with 50 per cent.

Chapter 7

Discussion

During the work on this master thesis a number of new insights and ideas regarding improvements has been born and scrutinized. This gives rise to continued efforts on improving the apparatus devised in this work. They can be divided into two groups: enrichment of the data set by adding new features and improvements on algorithms. Some of these ideas are discussed here in the final section of the paper.

7.1 Improving the feature space

A variety of information sources were used to create the feature set but no Internet based realtor's were willing to share their textual descriptions of the apartments sold. This source of information is probably the single most important piece of information needed to make a leap in the quality of the model. Studying the textual descriptions of apartment sales advertising indicates that the realtor's use a common jargon when writing these descriptions. Hence the textual description would be partitioned into tokens, from these so called N-grams (the N nearest neighbouring tokens are clustered together) can be constructed. From this list of N-grams the most frequent are selected, each N-gram becomes a feature and is added to the feature set fed into the NLP. This arrangement would hopefully be able to detect some of the more soft (non metric) qualities of the apartment.
Transportation to and from workplaces and social activities can be time consuming in Stockholm, especially to areas with spares local bus traffic. Adding features, which hold the commuting time from the apartment to a handful of carefully selected destinations. This information could be facilitated using "Resguiden" an application maintained by Stockholm Lokaltrafik.

7.2 Algorithmic improvements

The regime described in [1] of using a GT to compute a objective function used by a GA to select a preferred feature set that is fed into the MLP constructing the predictive model is probably a good extension of this work that probably can raise

the quality of this work. Combining the GT feature selection with the extended feature set discussed in previous section 7.1 is a natural extension of this work. The GT feature selection could be viewed as a pre processing of the feature set and the MLP and GA parameter optimization would take place as mentioned in this paper. There are probably good prospects of reducing the predictive region by investigating different non-conformity measures. In the paper [11] the authors present a improvement of the predictive region from 19.442 for the algorithm used in this paper to 14.228 using algorithm 2.10 at the 95 per cent level.

References to articles

[1] I. D. Wilson, Antonia. J. Jones, D. H. Jenkins, and J. A. Ware. Predicting housing value: Attribute selection and dependence modelling utilising the gamma test. In *Advances in Econometrics*, pages 243–275, 2004.

[2] Vânia. Ceccato and Mats Wilhelmsson. The impact of crime on apartment prices: Evidence from Stockholm, Sweden. *Geografiska Annaler: Series B, Human Geography*, 93(1):81–103, March 2011.

[3] Max Kummerow. Theory for real estate valuation: An alternative way to teach real estate price estimation methods. White paper, `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.12&rep=rep1&type=pdf` Last visited: 2014-11-14.

[4] Andrew Caplin, Sumit Chopra, John Leahy, Yann Lecun, and Trivikrmaman Thampy. Machine learning and the spatial structure of house prices and housing returns. White paper, `http://cess.nyu.edu/caplin/wp-content/uploads/2010/02/Machine-Learning-and-the-Spatial-Structure-of-House-Prices-and-Housing-Returns.pdf` Last visited: 2014-11-14, December 14 2008.

[5] Ku. Ruhana. Ku. Mahamud, Azuraliza. Abu. Baker, and Norita. Md. Norwawi. Multi layer perceptron modelling in the housing market. *Malaysian Management Journal*, 3(1):61–69, 1999.

[6] A. Stefánsson, N. Končar, and Antonia J. Jones. A note on the gamma test. *Neural Computing and Applications*, volume 5:131–133, 1997.

[7] Hari. Arul and Andres. Morales. NYC condo price estimation using NYC open data. White paper, `http://cs229.stanford.edu/proj2011/ArulMorales-NYCCondoPriceEstimationUsingNYCOpenData.pdf` Last visited: 2014-11-14.

[8] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research*, Proceedings Track 01:249–256, 2010.

[9] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[10] Harris Papadopoulos, Kostas Proedrou, Volodya Vovk, and Alex Gammerman. Inductive confidence machines for regression. *ECML*, volume 2430:345–356, Springer, 2002.

[11] Harris Papadopoulos, Vladimir Vovk, and Alex Gammerman. Regression conformal prediction with nearest neighbours. *Journal of Artificial Intelligence Research*, 40:815–840, April 2011.

Data sources

[1] Svenskagator. Lista över alla gator i Stockholm - svenskagator.se. Website, http://www.svenskagator.se/Stockholm/ Last visited: 2014-11-14.

[2] Slutpris. Slutpris.se. Website, http://slutpris.se/ Last visited: 2014-11-14.

[3] Openstreetgs - street informaton from Stockholm Municipal. Website, http://openstreetws.stockholm.se Last visited: 2014-11-14.

[4] SCB. KPIF, 12-månadersförändring. Website, http://www.scb.se/sv_/Hitta-statistik/Statistik-efter-amne/Priser-och-konsumtion/Konsumentprisindex/Konsumentprisindex-KPI/33772/33779/Underliggande-inflation-KPIX-och-KPIF/KPIF-12-manadersforandring/ Last visited: 2014-11-14.

[5] SCB. KPI, 12-månadersförändring (inflationstakten). Website, http://www.scb.se/sv_/Hitta-statistik/Statistik-efter-amne/Priser-och-konsumtion/Konsumentprisindex/Konsumentprisindex-KPI/33772/33779/Konsumentprisindex-KPI/KPI-12-manadersforandring-Inflationstakten Last visited: 2014-11-14.

[6] SEB. Räntehistorik - villa. Website, http://www.seb.se/pow/apps/HistoriskaBorantor/villaframe.aspx Last visited: 2014-11-14.

[7] Swebank. Historik bostadsräntor 2008-2013. Website, http://hypotek.swedbank.se/rantor/historiska-rantor Last visited: 2014-11-14.

[8] Valmyndigheten. Övergripande statistik om valet 2010. Website, http://www.val.se/val/val2010/statistik/index.html#slutligt Last visited: 2014-11-14.

[9] Stockholms Municipal. Open Data Stockholm. Website, http://api.stockholm.se/ Last visited: 2014-11-14.

Appendix A

Features

Table A.1. List of features part 1

Nr	Feature	Description
1	construction_year	Year when constructed
2	elevator	Building has elevator installed
3	fireplace	Fireplace available in apartment
4	duplex	Apartment has a duplex
5	penthouse	Apartment is a penthouse
6	balcony	Apartment has balcony
7	squares	Area of apartment in square meters
8	rooms	Total number of rooms, kitchen excluded
9	floor	Apartments floor (from the street level)
10	fee	Annual fee payed to the housing association
11	agencyid	Realtor identification
12	postal_code	Zip code
13	fix_point_1	Distance to KTH Royal Institute of Technology
14	fix_point_2	Distance to The Royal Palace of Stockholm
15	fix_point_3	Distance to Sergels Torg (CBD)

Table A.2. List of features part 2

Nr	Feature	Description
16	jog_track_dist	Distance to nearest jogging track
17	pad_pool_dist	Distance to nearest wading pool
18	daycare_dist	Distance to nearest daycare center
19	pool_dist	Distance to nearest pool facility
20	open_daycare_dist	Distance to nearest daycare center
21	sports_hall_dist	Distance to nearest sports hall
22	outdoor_gym_dist	Distance to nearest outdoor gymnasium
23	sports_field_dist	Distance to nearest sports field
24	playing_field_dist	Distance to nearest paying field
25	library_dist	Distance to nearest common library
26	env_station_dist	Distance to nearest environment station
27	preschool_dist	Distance to nearest preschool
28	bath_dist	Distance to nearest bath facility
29	playground_dist	Distance to nearest play ground
30	subway_dist	Distance to nearest subway station
31	station_dist	Distance to nearest train/commuter station.
32	park_dist	Distance to nearest park
33	forest_dist	Distance to nearest forest
34	water_dist	Distance to nearest watercourse
35	lat	Latitude coded in WGS84
36	lng	Longitude coded in WGS84
37	zone1	Id of grid square for apartment, from a 7x7 grid.
38	zone2	Id of grid square for apartment, from a 9x9 grid.

Table A.3. List of features part 3

Nr	Feature	Description
39	seb_interest_3m	SEB's interest rate for a 3 month loan from sell date
40	seb_interest_2y	SEB's interest rate for a 2 year loan from sell date
41	seb_interest_5y	SEB's interest rate for a 5 year loan from sell date
42	seb_interest_10y	SEB's interest rate for a 10 year loan from sell date
43	swebank_interest_3m	Swebank's interest rate for a 3 month loan from sell date
44	swebank_interest_2y	Swebank's interest rate for a 2 year loan from sell date
45	swebank_interest_5y	Swebank's interest rate for a 5 year loan from sell date
46	swebank_interest_10y	Swebank's interest rate for a 10 year loan from sell date
47	proc_M	Per cent of votes won by Moderata samlingspartiet
48	proc_C	Per cent of votes won by Centerpartiet
49	proc_FP	Per cent of votes won by Folkpartiet Liberalerna
50	proc_KD	Per cent of votes won by Kristdemokraterna
51	proc_S	Per cent of votes won by Socialdemokratiska arbetareparti
52	proc_V	Per cent of votes won by Vänsterpartiet
53	proc_MP	Per cent of votes won by Miljőpartiet de Grőna
54	proc_SD	Per cent of votes won by Sverigedemokraterna
55	proc_Alians	Majority for the Alliance parties
56	proc_RodGron	Majority for the opposition parties
57	majority_Alians	Per cent of votes to the majority