Stockholms universitet

# Prediction Using Neural Networks and a Comparison With Linear Regression

Alexander Nyberg

Kandidatuppsats i matematisk statistik
Bachelor Thesis in Mathematical Statistics

Kandidatuppsats 2021:7
Matematisk statistik
Juni 2021

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Stockholm University

Mathematical Statistics
Stockholm University
Bachelor Thesis **2021:7**
http://www.math.su.se

# Prediction Using Neural Networks and a Comparison With Linear Regression

Alexander Nyberg*

June 2021

**Abstract**

In this thesis, we are to compare two commonly used methods when working with prediction; neural networks and linear regression. We begin by covering the relevant theory for both methods. Further, we simulate three different data sets where one is a linear data set and the other two are non-linear. We are to train models using the two methods in order to then use these models for predicting values. First of all, how do we compare these two methods? In which situations do we prefer one method over the other? These are both methods used as machine learning algorithms and we will use techniques alike in this thesis. The calculations of the parameters of each method differ, which will lead to the results being different. Since linear regression can have a hard time fitting to a non-linear data set, one interesting factor is how well the neural network will handle the same situation. There are techniques available when working with linear regression so that the method is applicable to non-linear data. This is, however, done manually with a basis function. The comparison is made using three different performance measures. These are calculated for each model in order for us to obtain the one that fits best for each of the methods. We conclude that the neural network fits the data well for all the data sets used in this thesis. However, linear regression has a clear advantage in the linear data set, not only when comparing the performance measures but also for further observation since a linear regression model is much easier to analyse. For the non-linear data sets, we can observe a difference in the two methods. Where the neural network still performs well, we now see a worse outcome from the linear regression model.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: allestudent1995@gmail.com. Supervisor: Ola Hössjer, Kristoffer Lindensjö.

**Acknowledgements**

I sincerely want to thank my supervisors Ola Hössjer and Kristoffer Lindensjö for their meaningful feedback and guidance during the writing of this thesis.

# Contents

# 1 Introduction

*The first section is an introduction to the thesis, a summary of what is included as well as the objective of the study.*

The phenomenon of prediction is a widely analysed topic. Being able to observe, analyse, visualise and use past data in order to, in the best way possible, use these for new values on new data is a topic of growing importance. What might look like a simple task can become very difficult when dealing with different characteristics of data. For instance, if we would be able to predict future values with precise accuracy within a large business corporation, the sales made and the profit of this corporation would be at its highest efficiency since they would tweak production and price according to the analysis of the prediction.[4] However, one cannot predict the future and sometimes hundreds of factors play a key role for future actual values. But even though this is a difficult task to do, there are several methods available to create mathematical models in order to predict future values. Models that to some extent are wrong, but might be useful.

In this paper, we will perform prediction using two kinds of methods. These methods are widely used in the field of *Machine Learning* [7], where we first train the model using some training data set and then test the model with another unseen data set from the same distribution. The two methods are Neural Networks and Linear Regression. In the next section, we offer theoretical explanations of the two methods that will be used and algorithms used for fitting these models. They are different in their way of fitting and so also different in performance. An underlying hypothesis throughout this thesis is whether each of the two methods is better in a specific situation and why.

The third section will focus on simulation of data sets in order to analyse the methods in different situations. Here, three data sets are simulated with different characteristics. These are then trained with cross-validated data sets and further compared using different performance measures. These performance measures have the advantage that they can be calculated for both methods, that can then be used to analyse and compare the two methods in terms of prediction performance.

In the fourth section we will deep dive into the analysis of the simulated data sets as well as the fitting of the networks to these data sets and how we eventually end up with a desired model fit for each of the two methods.

The fifth section will be a presentation of the results yielded by fitting of the networks. Here, we will include plots of predicted against actual data. Additionally, we will present tables containing the values of the performance measures for the fitted models obtained with each of the two methods, that is, those two fitted

models that performed best. Such a comparison is presented for each of the data sets.

We conclude the thesis with a conclusion of the results as well as a discussion.

# 2 Theory

*This section provides the theory and information needed for the understanding of the two methods. This is divided into two section with Linear Regression being the first method and Neural Networks being the second. Explanation such as fitting a model, error terms as well as obstacles of each methods will also be provided.*

The theory section will focus on the two chosen models, *Linear Regression* and *Neural networks*, and their characteristics respectively. A large portion of the theory has been obtained from [1], [2] and [3]. Apart form these three references, we will notify if information has been provided elsewhere.

## 2.1 Linear model for regression

The use of a linear model when performing regression and predictions has become a fundamental part of statistical data analysis. The assumptions behind the model are usually easy to understand and analyse, which makes the model a frequently used one in statistical analysis as well as in machine learning situations.

### 2.1.1 The Linear Model

The linear model for regression provides a simple yet effective way of estimating a model from several exploratory variables. It does assume that the expected response $E(Y|X)$ is linear in the input data $X$, also referred to as the explanatory variables. Further, the goal of regression is to predict one or several response variables from these explanatory variables. A common procedure is that of *cross validation*. In order to explain this concept, we first need to explain the concepts of training and validation data. Having a set $D$ of data, we can further divide this data into subsets. One subset may consist of training data which are being used to train the actual model, using regression techniques. From there, after choosing a model that seems like a good choice, predictions of the rest of the data subset are made to test or validate the model. Cross validation is a procedure for fitting parameters to training data, when all observations of training data alternate to serve as a validation set of size 1, as they are predicted from a model fit based on the remaining parts of training data. Once the model has been fitted in this way, so that all observations of training data are predicted as well a possible, focus is

on using new input variables of the validation dataset in order to predict their associated response variables.

We can express the linear model with input vector $X = (X_1, X_2, ..., X_p)^T$ as

$$Y = \beta_0 + \sum_{j=1}^{p} \beta_j X_j + e = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \epsilon$$

which is simply known as *Multiple Linear Regression* for the values of $X$ consist of more than one explanatory variable. With this setup, it can be seen that $E(Y|X)$ is a linear combination of the input variables and the objective is to estimate the parameters $\beta_j$. Each $\beta_j$ parameter describes how much the explanatory variable $X_j$ effects the expected value of the variable $Y$. The first regression parameter, $\beta_0$, is added as the intercept of the model, having a pre-fixed value $X_0 = 1$ of its input parameter. Additionally, an error term $\epsilon$ is added to the model. This is a random variable with $E(\epsilon) = 0$ and $Var(\epsilon) = \sigma^2$. If there is implication that more than one explanatory variable might be effecting the response values then multiple regression come in handy. Further, if we are interested in $n$ of these observations, we can express the response value $Y_i$ as

$$Y_i = \beta_0 + \sum_{j=1}^{p} \beta_j X_{ji} = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + ... + \beta_p X_{pi} + \epsilon_i$$

for $i = 1, ..., n$ where $\epsilon_i$ are independent error terms with $E(\epsilon_i) = 0$ and $Var(\epsilon_i) = \sigma^2$.

For simplicity, we can represent the variables in matrix form as

$$\mathbf{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & X_{11} & X_{21} & \cdots & X_{p1} \\ 1 & X_{12} & X_{22} & \cdots & X_{p2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_{1n} & X_{2n} & \cdots & X_{pn} \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix}, \quad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix},$$

which yields the representation output $\mathbf{Y} = \boldsymbol{\beta}\mathbf{X} + \boldsymbol{\epsilon}$ for all $Y_i, i = (1, ..., n)$. As for the intercept, we have added a first column of ones in $\boldsymbol{X}$, which is multiplied by $\beta_0$ in the matrix multiplication.

Further, when obtaining data, the values of $Y_i$ as well as $X_i = (X_{1i}, X_{2i}, ..., X_{pi})$ are known. The goal of prediction is to use these past data values and train the model which involves specifying the unknown parameters $\boldsymbol{\beta} = (\beta_1, \beta_2, ..., \beta_p)^T$ and $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, ..., \epsilon_n)^T$. Given the multiple linear regression model above, it is close to impossible to obtain a model that, with perfection, position all data $((X_{1i}, ..., X_{pi}), Y_i)$ along a hyperplane in $(p+1)$-dimensional Euclidean space. The

data points from the actual regression model will rather have an added error term vector $\boldsymbol{\epsilon}$. These errors consist of the difference between the true line of the regression model and the actual data points. Further, assumptions are made about the error vector $\boldsymbol{\epsilon}$ that the distribution of the terms is multivariate normal $\boldsymbol{\epsilon} \sim N((0,\ldots,0)^T, \sigma^2 \boldsymbol{I}_n)$, where $\boldsymbol{I}_n$ is the identity matrix of order $n$.

### 2.1.2 Least Squares Estimator

The creation of a model is typically focused on estimating the unknown parameters $\boldsymbol{\beta}$. The most frequently used estimation method involves the minimization of the residual sum of squares, *RSS*. That is, we want to minimize

$$RSS = \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j X_{ji})^2.$$

This method is referred to as the *least square* method.

The least square method is overall a satisfying method to use due to its simple nature. Since it measures fits a linear regression plane to the data it is very often used for all kinds of data.

So, minimizing the *RSS* is the strategy for obtaining a good model for prediction. For each collection of vectors $\boldsymbol{X}_i = (X_{1i}, \ldots, X_{pi})^T$, we can express the *RSS* as a matrix calculation in the form of

$$RSS = (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^T (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})$$

where $\mathbf{X}$ is defined as the $n$ x $(p+1)$ matrix, each row $(1, \boldsymbol{X}_i^T)$ being an input vector with the value 1 as its first position. Additionally, $\mathbf{Y}$ is defined as the $n$ x 1 response value vector with respective values in the training set.

Further, by differentiating with respect to $\boldsymbol{\beta}$, we obtain

$$\frac{\partial RSS}{\partial \boldsymbol{\beta}} = -2\mathbf{X}^T\mathbf{Y} + 2\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}.$$

Setting the expression equal to zero yields

$$\mathbf{X}^T\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}^T\mathbf{Y}$$

which finally results in the unique solution

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}.$$

whenever $\boldsymbol{X}^T\boldsymbol{X}$ has full rank $p+1$. So, using the input data of $\mathbf{X}$, we obtain the least square estimator $\hat{\boldsymbol{\beta}}$. It then implies that

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} = \mathbf{X}\hat{\boldsymbol{\beta}} + \boldsymbol{e}$$

where $\mathbf{X}$ contains the inputs and $\boldsymbol{e}$ now is the residual vector, that is, the part of data not explained by the fitted model. Further, using the fact that $\mathbf{X}\hat{\boldsymbol{\beta}} = \hat{\mathbf{Y}}$, we get that $\boldsymbol{\epsilon} = \mathbf{Y} - \mathbf{Y}_{est}$.

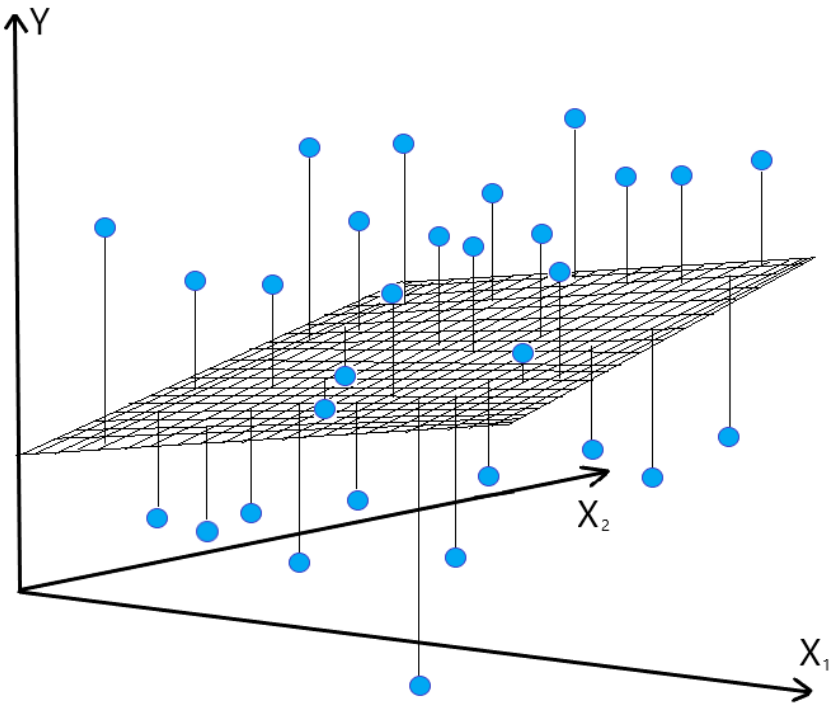


Figure 1: Least Square Method: Fitting data points to a estimated plane. Inspiration from [2].

### 2.1.3 Assumptions of the linear regression model

For the linear regression model to hold, certain assumptions have to be met. We have briefly mentioned that the function of regression $E(Y|X)$ has to be *linear* in the parameters. This holds even though the input data is so called non-linear.

We have also mentioned the error term vector of the model, which is distributed as $\boldsymbol{\epsilon} \sim N(0, \sigma^2\boldsymbol{I}_n)$. This does tell us that the conditional expected value of the error terms to the inputs are zero, i.e. $E(\boldsymbol{\epsilon}|\mathbf{X}) = \mathbf{0}$. Additionally, we see that $E(\epsilon_i) = E(E(\epsilon_i|\mathbf{X}_i)) = E(0) = 0$, for $i = 1, 2, ..., n$, which implies the unconditional expected values are also zero. This further implies that $E(\mathbf{Y}|\mathbf{X}) = \mathbf{X}\boldsymbol{\beta}$.

Additionally, we expect homoscedasticity, meaning that the values of the explanatory variables is independent of the error term variance, $V(\epsilon_i|\mathbf{X}_i) = \sigma^2$ for $i = 1, 2, ..., n$. Meanwhile, the value of $\sigma^2$ stays constant for all error terms.

We assume $\mathbf{X}$ to be a full rank $n$ x $k$ matrix, meaning that no column is a linear combination of another column and that there are at least $k$ observations.

We expect no autocorrelation in the model, meaning that the covariance value of any two error terms terms $(\epsilon_i, \epsilon_j)$ is independent of any value of the corresponding explanatory variables. This can be stated as $Cov(\epsilon_i, \epsilon_j|\mathbf{X}_i, \mathbf{X}_j) = 0, \quad \forall\ i \neq j$.

Lastly, we assume that the error terms are normal distributed which results in the model being multivariate normal distributed as $\boldsymbol{\epsilon}|\mathbf{X} \in N(\mathbf{0}, \sigma^2\boldsymbol{I}_n)$.

### 2.1.4 Basis Functions

An important characteristic of the regression model is that the $\beta_j$ parameters have a linear effect on the model. Thus, the model becomes rather limited when fitting a linear model on non-linear data. As a solution, we do have the possibility to remedy this using *basis functions*. The basis functions are used as transformations. Since the input variables $\boldsymbol{X}^T = (X_1, X_2, ..., X_p)$ are known, these can be modified using some functions $\phi_j(X_j)$. What characterizes the model as linear are the actual $\beta_j$ parameters. It is also common for the covariates to be measured in different ways on different scales, which makes the basis functions practical to use whenever this is the case.

Again we have the regression model as

$$Y_i = \beta_0 + \sum_{j=1}^{p} \beta_j X_{ji} + \epsilon_i.$$

We further apply the basis functions to the covariates in order to extend this model. We now have

$$Y_i = \beta_0 + \sum_{j=1}^{p} \beta_j \phi_j(X_{ji}) + \epsilon_i.$$

Recall, the parameters $\beta_j$ as well as the intercept still make the model linear, even though the input data $X_{ji}$ are transformed by possibly non-linear functions.

Very common choices of basis functions include the identity basis $\phi(X_{ji}) = X_{ji}$, quadratic basis $\phi(X_{ji}) = X_{ji}^2$ and the log basis $\phi(X_{ji}) = log(X_{ji})$. All these basis functions preserve the linearity of the model, since the relationship between the expected outcome and the parameters is still linear.

### 2.1.5 Obstacles with linear regression

While the linear regression model is widely used for fitting and predicting data, its simple nature makes it limited to more complex data characteristics. Since we expect the explanatory variables to have a linear relationship with the expected response, we directly put restrains to the model. However, this restriction can be relaxed by with the use of basis functions, as stated above. A problem with the basis function approach for linear regression is that it is only practical to use when the number of explanatory variables and hence the number of parameters is small. Since you have to add each basis function $\phi_j$ to each of the explanatory variables explicitly, with a large number of explanatory variables in a model, this task can becomes overwhelming. Additionally, with a high number of explanatory variables it can be very challenging applying the most appropriate basis function to each explanatory variable.

## 2.2 Neural Networks

Neural network as a term was first introduced in 1943.[8] It is largely focused on the information processing of the human brain and trying to mimic it using mathematical approaches and models. As input data are being observed, each of these inputs are being processed by a neuron within a layer of the network. Since each layer may consist of several neurons, each data point must now pass through the full layer and hence pass through every neuron. After each point has passed through all neurons of the first layer, each value of the neuron connects with the next layer in the same way with an entirely new set of neurons to be processed. This process is repeated until we finally connect with a set of *output neurons* or the *output layer*, where we now obtain the values that aim to approximate the expected response variable(s) associated with the input data.

This process is known as *feed-forward neural network* - the connections move from one layer to the next until all layers are passed through. Further explanation will be made below with more mathematical terms.

### 2.2.1 Setup of the Network

Throughout this thesis, we will consider a neural network for prediction with a single output variable. The network consist of three main parts; *Input layer*, one or more *hidden layers* and lastly, an *output layer*. For this thesis, we will restrict the number of hidden layers to be either one or two. The input layer consists of nodes with the actual input data. Hence, this connection does not have an effect on the data. From here, the neurons within the input layer are connected with the nodes within the second layer, which is the first (and sometimes the only)

hidden layer. These connections now have an effect on the output data due to the way signals are propagated through the network.

With the former statement, we can describe the input network with $K$ constructed linear combinations as

$$a_k = \sum_{j=0}^{p} w_{kj}^{(1)} x_j = w_{k0}^{(1)} + w_{k1}^{(1)} x_1 + ... + w_{kp}^{(1)} x_p$$

where $k = 1, ..., K$, $w_{kj}^{(1)}$ is the *j'th weight* of linear combination $k$ and it is multiplied with data value $x_j$, where $x_j \in \{x_0, ..., x_p\}$. Note that the equation above defines the actual connection of weights and the superscript (1) defines which hidden layer the weights are connected to. The weights have a large impact on the properties of the nodes, the training of the network and the actual predicted values. For $j = 0$, we assign data point $x_0 = 1$ and hence the first term of the right hand side of the above equation is $w_{k0}^{(1)}$. This is referred to as the *bias* parameter of the current $k$'th node. Weights and bias parameters are referred to as *regression coefficients* and *intercept*, respectively, in analogy with linear regression notation.

We do define the term as $a_k$ for being the quantity of the activations - depending on the values obtained from the previous layer of nodes, the quantities of this layer of nodes are modified by the *activation function*. Each of the $a_k$'s is now transformed using a non-linear activation function, most likely a logistic sigmoidial or a tanh function, which here functions as a type of basis function similar to the one of linear regression. The knowledge of these non-linear activation functions play a central role in the training of the network, where a differentiable function is needed.

We can describe this new data set as $s_k = h(a_k)$ where $h(\cdot)$ is the activation function. The variables $s_k$ now function as the different values of the nodes within the first hidden layer. When all the quantities of activation $a_k, k = 1, ..., K$ have been calculated, new connections are made to the next layer of nodes. This could very well be another hidden layer or the output layer.

So far, we have an input layer, the first hidden layer and a set of weights connecting these two. For the next hidden layer, we obtain another set of connections in the form of

$$b_l = \sum_{k=0}^{K} w_{lk}^{(2)} s_k,$$

where we now consider $s_k$ as the data from the first hidden layer with $k = 1, ..., K$ outputs as well as $l = 1, ..., L$ being the number of nodes within the new hidden layer. Now, all $w_{lk}^{(2)}$ denote the values of the weights in the connection between the first and the second hidden layer and $w_{l0}^{(2)}$ will again function as a bias parameter.

Since we are using the neural network for regression, i.e. to predict a certain value of a response variable associated with the input data, a wanted output value consists of only one variable. We further add an output layer, constructed as

$$c_m = \sum_{l=0}^{L} w_{ml}^{(3)} t_l,$$

where $m = 1, ..., M$ is the new set of nodes within the output layer and $t_l = h(b_l)$ is the data from the second hidden layer. By setting $M = 1$, the output layer now consist of one node and it is connected with the second hidden layer, resulting in only one predicted value $\hat{y}_m$. As a matter of fact, since the last activation function for a regression is the *identity function*, the output value will be $\hat{y}_M = c_M, M = 1$. We will denote the actual output values as $\hat{y}_m$.

With the above notation of every layer respectively, we are now ready to formulate a neural network mathematically with input layer, two hidden layers and output layer with weight connections between them. We denote this as

$$\begin{aligned} \hat{y}_m(\mathbf{x}, \mathbf{w}) &= g\left(\sum_{l=0}^{L} w_{ml}^{(3)} h\left(\sum_{k=0}^{K} w_{lk}^{(2)} h\left(\sum_{j=0}^{p} w_{kj}^{(1)} x_j\right)\right)\right) \\ &= \\ \sum_{l=0}^{L} w_{ml}^{(3)} h\left(\sum_{k=0}^{K} w_{lk}^{(2)} h\left(\sum_{j=0}^{p} w_{kj}^{(1)} x_j\right)\right) &= f_m(\mathbf{x}, \mathbf{w}) \end{aligned}$$

where we remove the function $g(\cdot)$ since, in case of regression, it is the identity function whereas $h(\cdot)$ is the within node activation function. We also define $f_m(\mathbf{x}, \mathbf{w})$ for upcoming sections.

### 2.2.2 Fitting a Neural Network

Now, how do we make a network to estimate its parameters from training data in order to predict the outcome variables of validation data? As the feed-forward procedure are active, the weight and bias parameters that connect the nodes from one layer to another are to be estimated. Reaching the last layer of the network, the output layer, a prediction is made. From these values, we can now derive a so called *loss function*. This is closely related to that of linear regression, where the interest lies in minimizing the sum of squared prediction errors.

For input data $\mathbf{x} = (x_0, x_1, \ldots, x_p)$ we further denote $\boldsymbol{\theta}$ as the full set of weights. This now includes $K(p + 1)$ connection of weights from the input of $p$ values to the first hidden layer of $K$ nodes. For any node of the hidden layer each of the $p$

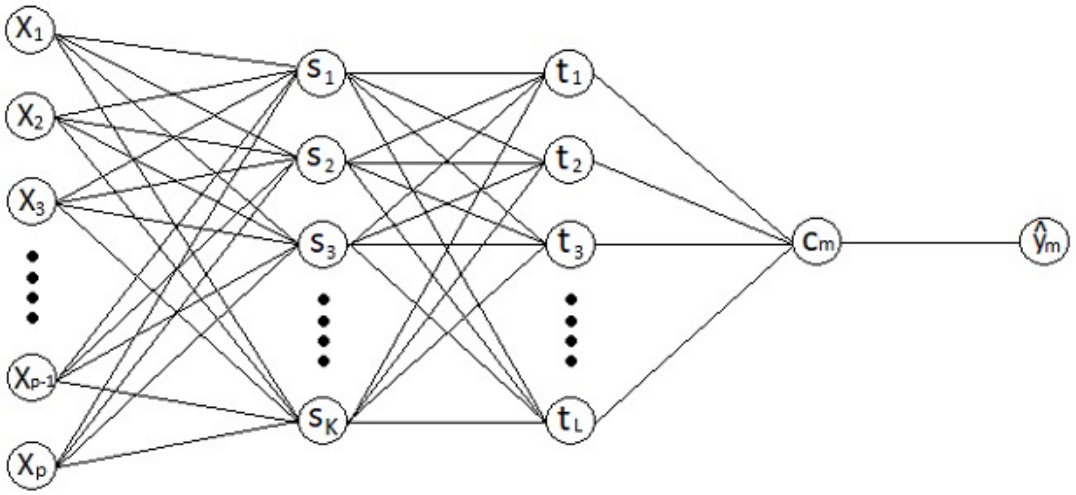


Figure 2: A neural network with two hidden layers. Inspiration from [2].

input variables contribute with one parameter, and the last parameter denotes the bias parameter, representing $w_{k0}x_0 = w_{k0}$. Likewise, we obtain another $L(K+1)$ connection of weights from the first hidden layer of $K$ nodes to the second hidden layer of $L$ nodes. Finally, the connection between the second hidden layer and the output layer consists of another $M(L+1)$ parameters. Again, since we are using the network for regression, we want one output as prediction and hence, $M = 1$. As we now obtained the first predicted value $\hat{y}_{im} = f_m(\mathbf{x}_i)$ of each response variable $y_i$ based on the corresponding input data $\mathbf{x}_i = (x_{i0}, \ldots, x_{ip})$, the objective is to minimize the sum of squared errors, represented as

$$R(\boldsymbol{\theta}) = \sum_{m=1}^{M} \sum_{i=1}^{n} (y_{im} - f_m(x_i))^2 = \sum_{i=1}^{n} R_i(\boldsymbol{\theta}).$$

This approach of minimization in a feed-forward network will make use of *gradient descent*, a technique where we use the activation functions within the neurons that are differentiable using the chain rule of differentiation. We now move in the opposite direction, *backwards*, to receive information about the gradient vector of $R(\boldsymbol{\theta})$ to change $\boldsymbol{\theta}$ along this direction and thereby further reduce the value of the loss function. When this is done, the feed-forward process is applied again, now with updated weights and biases, to obtain possibly better predicted values. This process, that is repeated back and forth until the predicted values converge, is

called *error back propagation*.

### 2.2.3 Error Backpropagation and Gradient Descent

Let us look at this in more detail. When the feed-forward calculations have been made, we obtain an output value which is the predicted value of the response variable that we want to examine. Further, the loss function is calculated using the predicted value minus the actual value of the output as prediction error. The gradient of the loss function value is then calculated, this with respect to each single weight that has been calculated during the feed-forward process in the network. Since each of the nodes within the network has a non-linear activation function, the gradient is then a non-constant value and therefore applicable for gradient descent. In other words, for all $i$, $j$, $k$, $l$ and $m = 1$ we calculate

$$\frac{\partial R_i}{\partial w_{kj}^{(1)}}, \quad \frac{\partial R_i}{\partial w_{lk}^{(2)}}, \quad \frac{\partial R_i}{\partial w_{ml}^{(3)}}.$$

We can now use these gradient values as gradient descent updates. Further, dividing the network process into two parts, we then have a forward sweep, which includes input of data, propagating this input data using the current weights, to the hidden and output layers, and finally calculating the loss function. Given the above derivatives of the loss function with respect to the weights, we now perform a backward sweep, where each parameter is updated from iteration $r$ to $r + 1$ in the form of

$$\begin{aligned}
w_{kj}^{(1)(r+1)} &= w_{kj}^{(1)(r)} - \gamma_r \sum_{i=1}^{n} \frac{\partial R_i}{\partial w_{kj}^{(1)(r)}}, \\
w_{lk}^{(2)(r+1)} &= w_{lk}^{(2)(r)} - \gamma_r \sum_{i=1}^{n} \frac{\partial R_i}{\partial w_{lk}^{(2)(r)}}, \\
w_{ml}^{(3)(r+1)} &= w_{ml}^{(3)(r)} - \gamma_r \sum_{i=1}^{n} \frac{\partial R_i}{\partial w_{ml}^{(3)(r)}}.
\end{aligned}$$

This subtraction of the gradient from the weight value will reduce the error value. Since the gradient is the actual direction of the steepest descent, by subtracting this from the weights with a somehow good choice of *learning rate* $\gamma_r$, the updated value of the loss function will be close to a point of *local* or *global minimum*. These are different points within the gradient space that functions as minima of the loss function. The learning rate is commonly a constant, chosen with respect to obtaining a balance between accuracy and the number of iterations while not making the network spend too much time iterating. It is, however, possible to

find an optimal $\gamma_r$ using gradient techniques as well, but this is often an expensive calculation in high-dimensional space.

A scenario where we end up in the global minimum of the loss function is typically not desired, since this is likely to overfit the model. Because the network needs to handle derivatives of non-linear functions, it may not need to access this global minimum but instead one of many local minima that are present. Further, one can compare these minima to eventually choose which one may suit the model best.

### 2.2.4 Obstacles with Neural Networks

As mentioned, a common problem during fitting of a neural network model is the amount of parameters that is estimated. This can often lead to an overfit to data and leads to the loss function ending up in the global minimum. Overfitting is when a model is fitted to the training data too well. It is then difficult for the model to generalize to new data. The most common solution involves *early stopping*, a technique where we use the cross-validated data in groups of $D_t$ as the training data and $D_v$ as validation data. As we trained the network with $D_t$ we now use $D_v$ and implement an intentional iteration stop whenever we think that the network is within or very close to a local minimum of the loss function. In this way, the network will not iterate in the wrong direction of the gradient descent. For another method, similar to that of the *ridge regression* which is a type of shrinkage method (along with Lasso), a penalty value is added to the error function. This happens in the form of $R(\theta) + \lambda J(\theta)$ for a constant tuning parameter $\lambda \geq 0$. The penalty can be expressed as

$$J(\boldsymbol{\theta}) = \sum_{k,j} w_{kj}^{(1)} + \sum_{l,k} w_{lk}^{(2)} + \sum_{m,l} w_{ml}^{(3)}.$$

A value of $\lambda$ is usually estimated by cross-validation.

Standardization of the inputs is another important factor when dealing with neural networks. To be sure that all input data is treated the same, standardization are made so that inputs have mean equals zero and standard deviation equals one. This also ensures that the first connection of weights between the input layer and the first hidden layer has a range well optimized for input data. These first connections are typically randomly simulated from a uniform distribution with range $[-0.7, 0.7]$. This is also used in the following simulations. Starting with weights being equal to zero will lead to zero derivatives within the network, whereas large values of starting weights tend to effect the results badly.

# 3 Simulation and Fitting Models

*Chapter 3 will focus on the creation of models to simulate data from, and how to evaluate their performance. We will explain the technical parts of of fitting linear regression models and neural networks, the performance measures that are being used in order to compare the models, the data simulation structure as well as what types of networks to choose from when creating neural network models.*

The following simulation study will largely focus on three specific data sets. These data sets have been simulated to train models that has been fitted using either linear regression or neural networks. We will be using packages from `R`, namely `neuralnet` package and the `lm` function. The `lm` function is frequently used for fitting linear models and performing linear regression analyses. The `neuralnet` package might be less common since it primarily focuses on neural network analyses. Therefore, a short description is made for further understanding of the network fitting used in the simulations. Further, we will also introduce different measures of performance for comparison between the fitted models. Mentioned in the previous chapters is the sum of squared errors as the loss function. We can further use the *mean squared errors*, simplified as *MSE*, to use as a performance measure. We will compare this with the *mean absolute error*, simplified as *MAE*. Lastly, the coefficient of determination value $R^2$ will be calculated in order to assess whether a model has a good fit or not. The data sets will be arranged into cross-validated subsets of training, validation and test data. The choice of network structure will be presented after iterating through models with either one or two hidden layers with one to ten nodes in each.

## 3.1 The neuralnet package

To able to fit the neural networks provided in this study, we make use of the `neuralnet` package. Being a flexible package, it is very much up to the user to choose the number of input nodes, the number of hidden layers, the number of nodes within each layer as well as an output layer. A primary concern is usually the amount of hidden layers as well as the amount of nodes within each of the layers. Since we need to choose a network structure before fitting the network, we decided to first find the best network structure in an iterative way, and then use this structure for comparing its fit with that of a linear model.

Even though this package competes with other packages, like `nnet` and `AMORE`, the `neuralnet` is built more towards training networks for the sake of regression and thus was the clear choice when choosing how to fit the network.[6]

## 3.2 Fitting the Linear Regression Model

For the fitting of the linear regression model, we use the `lm` function which is a standard built in function within `R`.

## 3.3 Performance Measures

To be able to compare the two models we are in need of diverse measures. These measures make use of the sum of the errors but in unique ways for each of them.

### MSE

The MSE is defined as

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

and is a common technique to measure the performance. The result is obtained from calculating the difference of the actual value and the predicted value of each response variable for the used model in order to analyse if the fitted model performs well. However, since MSE is calculating the square residual, some data points have an effect on the MSE that might be misleading. Outliers, for example, have a bigger impact on the MSE. This is something to consider when analyzing the data sets that have been simulated. Further, a smaller MSE generally tends to indicate a better fit of the model to data.

### MAE

We define the mean absolute error as

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|,$$

which measures the average absolute difference between the response variables and the predictions of them obtained from the model fit. MAE is commonly used for prediction accuracy in forecasting, but also for calculating average error for model performance.[9] Outliers are handled better by the MAE measure since large prediction errors have a lower impact compared to the MSE criterion. Since we are calculating absolute values, the measure can be interpreted as an average distance between the data points and the projections of the data points on the fitted hyperplane. It is therefore also always positive.

### $R^2$

The $R^2$, also known as the coefficient of determination, is a commonly used value in determining the proportion of the total sample variability of the response values that is explained by the covariates of the chosen model. We define this as

$$R^2 = \frac{SS_{reg}}{SS_{total}} = 1 - \frac{RSS}{SS_{total}},$$

where $SS_{reg}$ is the regression sum of squares, $SS_{total}$ is the total sum of squares and $RSS$ is the residual sum of squares. The value of $R^2$ ranges from 0 to 1. This is a very convenient range for assessing whether the fitted model explains the data well whereas it is more difficult to interpret the other performance measures. This measure $R^2$ is used here to confirm that a model provides a good fit to the test data.[5]

## 3.4 Simulated data

We have simulated three different data sets with unique characteristics. This includes data sets from one linear model and two non-linear models. The simulation is described in more detail in section 4. Next, standardization of the explanatory variables of input data is made with mean $\mu = 0$ and standard deviation $\sigma = 1$. This standardization is usually made so that the parameter estimation of the network converges faster than otherwise since the data processing is simplified. We will further perform model fitting and performance tests on the different data sets. With each data set we perform cross-validation in order to train and test the two models.

### 3.4.1 Cross-Validation

Each of the simulated data sets has been divided into three subsets using cross-validation of training, validation and test data. While the training and the validation data is a 80:20 split, we simulate another set of data from the same distribution and use this as the test data set. This is then repeated for each of the three different kinds of data sets. We do this to avoid any over- or underfitting of the models.

## 3.5 Choice of network

While the `neuralnet` package performs the calculations in order to fit the network, the structure of the network has to be chosen at first in order to perform the model fitting. The structure involves the input weights connection between the input data and the first layer of nodes, the number of nodes within this first layer and whether we want a second layer for which the number of nodes has to be decided.

The connection of weights between the input data and the first hidden layer are randomly chosen from a uniform distribution over the range of (-0.7, 0.7). The number of variables that are generated depends on the number of nodes within the first hidden layer. For the first test, the first hidden layer will now contain one single node which increases with one node for each iteration. Meanwhile, all the above performance measures will be calculated. Further, we will perform the same type of test but for a second hidden layer, resulting in ten times as many iterations. Like before, the performance measures are calculated and presented.

# 4 Testing

*The fourth section investigates in detail the prediction performance of the linear and neural network models. This is mainly focused on the simulated data as well as an explanation of the iteration procedure needed to obtain the best neural network model.*

The data sets are all simulated using the standard library within `R` as well as using the `mvrnorm` function. Further, we will cross-validate the data as previously mentioned. The modelling and the fitting of the two methods are performed on each of the data sets where the linear regression is fitted with the `lm` function to obtain predictions of response variables in the validation dataset. With the neural network, we use the `neuralnet` package to fit the model and perform the iteration procedure as mentioned above. This will yield several fitted models to along with the associated *MSE*, *MAE* and $R^2$ values for each neural network model. These performance measures, and the corresponding ones for linear least squares, are further compared and used to draw conclusions about each of the methods.

## 4.1 Data description

### 4.1.1 First Data Set

The simulation process has been divided into different parts for each of the data sets. The explanatory variables in the first data set are simulated from a multivariate normal distribution with mean $(0, 1, 5)$ and the identity matrix as covariance matrix with the number of observations being $n = 500$. This result in a design matrix with $p + 1 = 4$ columns, with 500 observations in each column. Further, we simulate and add the error term vector $\boldsymbol{\epsilon}$, whose components $\epsilon_i$ are independent with a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$. Lastly, in order to generate response variables $Y_i$ we add to the error terms $\epsilon_i$, the intercept term $\beta_0$ as well as a linear combination of the effect parameters $\beta_j$ and covariates $x_{ji}$ for $j = 1, 2, 3$. The intercept and the three effect parameters

were given values $\beta_0 = 10$ and $\beta_1 = 5$, $\beta_2 = 1.7$ and $\beta_3 = 1.2$ respectively. This data set is intended to be fitted well from both the trained linear regression model and the trained neural network since both the linear regression method and the neural network method will be able to fit a model where the expected value of the response variables depends linearly on the explanatory variables.

### 4.1.2 Second Data Set

For the second data set, the explanatory variables are simulated differently for each column of the design matrix. The column that corresponds to the first explanatory variable is a sequence of equidistant and deterministically chosen numbers $(0.05, ..., 15)$ with an distance of 0.05 for each number. The columns that correspond to the second and third explanatory variables are simulated from a uniform distribution on the interval $[2, 3]$ and a Poisson distribution with mean $\lambda = 0.7$, respectively. The number of observations is $n = 300$, resulting in a design matrix with $p + 1 = 4$ columns and $n = 300$ rows. Further, we simulate the error term vector $\boldsymbol{\epsilon}$ with independent components from a normal distribution with mean $\mu = 0$ and $\sigma = 1$. The intercept term $\beta_0 = -20$ as well as a non-linear function of the three effect parameters $\beta_1 = 30$, $\beta_2 = 3$,$\beta_3 = -2$ and covariates $x_{1i}$, $x_{2i}$, $x_{3i}$ are then added to the error term in order to generate response variables

$$Y_i = \beta_0 + \frac{\beta_1 x_{1i}}{1 + x_{1i}} + \beta_2 x_{2i} + \beta_3 x_{3i} + \varepsilon_i.$$

These simulated values are now used as response variables of a a non-linear regression model, since the expected value of each response variable is a non-linear function of the corresponding explanatory variables. This data set is further intended to catch the difference between the methods as the neural network is expected to perform better than the linear regression method due to the non-linear characteristic of the data set. However, we do have the interest to apply a basis function transformation to one or more of the explanatory variables within the linear regression model to increase its accuracy.

### 4.1.3 Third Data Set

The third data set is also intended to showcase the difference between the two methods. The simulation of the explanatory variables is similar to that of the second data set - the column of the design matrix that corresponds to the first explanatory variable is a sequence of equidistant numbers $(-2, ..., 3.98)$ with a distance of 0.02. The column that corresponds to the second explanatory variable has independent and normally distributed components with mean $\mu = 1$ and standard deviation $\sigma = 2$ and the column that corresponds to the third column

is generated from a uniform distribution on the interval $[2, 3]$. Each of these columns consists of $n = 300$ observations. We also simulate another $n = 300$ error term observations with mean $\mu = 0$ and standard deviation $\sigma = 1$. As previously, the intercept term $\beta_0 = 2$ as well as the three effect parameters $\beta_1 = 2$, $\beta_2 = -1.3$ and $\beta_3 = 0.2$ are then added to the model. For each observation the explanatory variables and effect parameters are transformed non-linearly, and then the intercept and the error terms are added, in order to generate the response variable

$$Y_i = \beta_0 + \frac{\beta_1 x_{1i}^2}{2 + sin(5x_{1i})} + \beta_2 x_{2i} + \beta_3 exp(x_{3i}) + \varepsilon_i.$$

This data set is intended to demonstrate the difficulty in having a good fit of a linear least squares model, even when one or several of the explanatory variables are transformed by basis functions.

## 4.2 Neural network iteration

As mentioned previously, one of the core challenges when fitting a neural network is to choose the number of hidden layers and the number of nodes within each of them. Using the simulated data with standardized explanatory variables, we first perform the fitting of ten neural networks with only one hidden layer, ranging from one node to ten. Whenever these networks are trained, we apply the performance measures in order to obtain a network that fits data in the best way.

Further, we iterate in the same way but now with two hidden layers of each network. This also involves more calculations. Starting of with a two hidden layer network with one node in each layer, we increase nodes within the second layer as we fit networks. Reaching a maximum of ten nodes within that second hidden layer, we then add another node to the first layer, so that it contains two nodes. The second layer is refreshed with one node and the iteration is repeated until the second layer has ten nodes. The size of the first hidden layer is then iteratively increased to have up to ten nodes, with ten choices of second layer for each. This gives a total of 100 fitted neural networks, all with their corresponding performance measures of the validation data. This simplifies the task of choosing the right structure for the network that we want to showcase when comparing the prediction ability of the neural network to that of the least squares method for the linear model. We then use the network that yields the best performance measures from the validation data on another test data set. For this data set we predict response variables and calculate the performance measures. This is the result that will be presented in the next section.

# 5 Results

*As the simulation, the fitting and the calculations of the performance measures has been presented, we provide the results of the simulations in this section. Here, we focus mainly of the obtained values of the performance measures as well as to visualize these result.*

## 5.1 First Data Set - Linear Regression

Since both the neural network and the linear regression methods make use of the least squares method for fitting and that both methods temper linear data well, the results of both methods are similar to each other. Observing the actual data against the predicted data points for the test data, we see that the two methods yield almost identical results (see Figure 1). This suggest a good predictive ability for both methods when applied to this first dataset.

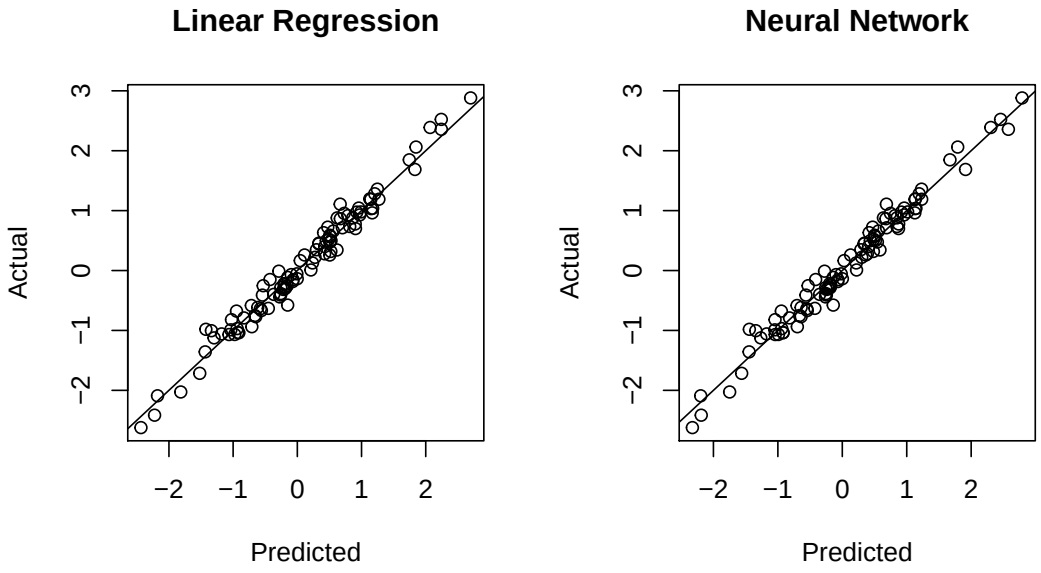


Figure 3: Comparisons of plots of actual versus predicted response variables for data set 1.

The performance measures also give similar results for the two methods, with a slight advantage of the fitted linear regression model when comparing the MSE and and the MAE. The coefficient of determination is also presented to evaluate the fit of the model (Table 1). We showcase the neural network model with

the lowest MSE with *NN(MSE)* and likewise for the MAE. The linear model is simply *LR-model*. This is somewhat expected, since the linear model is described with less parameters than the neural network, and since data is simulated from a linear model. We also observe that the same neural network model, with four nodes within the single layer gives the lowest MSE and the highest coefficient of determination value. Interesting enough, the lowest MAE value is present for another, more complicated neural network. This model consist of four nodes within the first layer and six nodes within the second.

Table 1: Performance Measures - Data set 1. The numbers $K$ and $L$ refer to the number of nodes of the first and second hidden layer of a neural network, with a line when such a layer does not exist.

| Type | K | L | MSE | MAE | $R^2$ |
|---|---|---|---|---|---|
| LR-model | - | - | 0.02709 | 0.13398 | 0.97446 |
| NN (MSE) | 4 | 6 | 0.0401 | 0.1627 | 0.9551 |
| NN (MAE) | 2 | 6 | 0.0406 | 0.1626 | 0.9547 |

Table 2: Topp 10 neural networks with lowest MSE for dataset 1.

| K | L | MSE | MAE | $R^2$ |
|---|---|---|---|---|
| 4 | 6 | 0.0401 | 0.1627 | 0.9551 |
| 3 | - | 0.04029 | 0.16318 | 0.95495 |
| 3 | 6 | 0.0404 | 0.1631 | 0.9548 |
| 2 | 10 | 0.0404 | 0.1634 | 0.9548 |
| 2 | 3 | 0.0406 | 0.1637 | 0.9546 |
| 3 | 5 | 0.0406 | 0.1636 | 0.9546 |
| 2 | 6 | 0.0406 | 0.1626 | 0.9547 |
| 2 | 8 | 0.0406 | 0.1626 | 0.9546 |
| 2 | - | 0.04069 | 0.16262 | 0.9545 |
| 6 | 6 | 0.0408 | 0.1648 | 0.9543 |

## 5.2 Second Data Set - Nonlinear Regression I

For the result of the first dataset from a non-linear regression model we immediately see a difference in performance of the two methods. Observing the actual against predicted plots the linear regression struggles to fit to the data while the neural network performs well (see Figure 2).

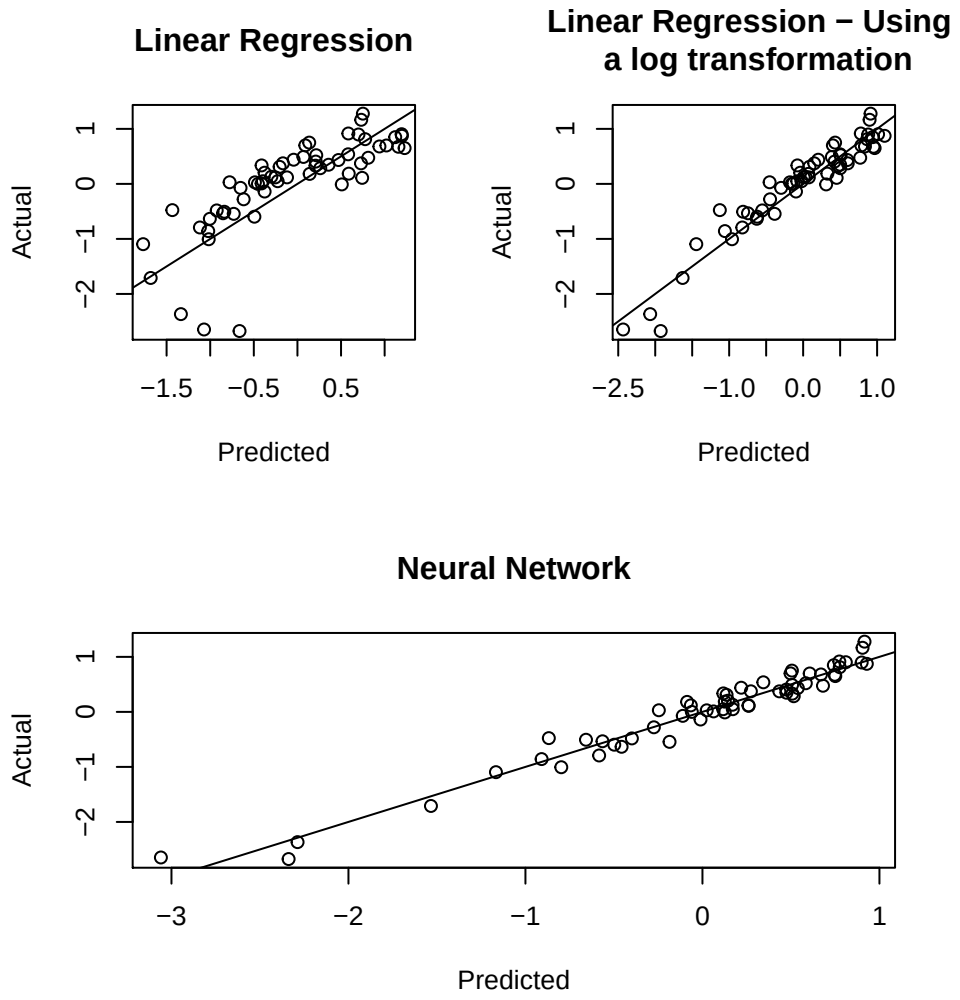


Figure 4: Comparisons of plots of actual versus predicted response variables for data set 2

Since the neural network is adapting to the non-linear data through the non-linear activation function, the training algorithm is able to utilize the gradient descent in a better way in order to fit a network to the training data.

Observing the table, we get a better understanding of the performance measures. The linear regression method does a poor job of fitting the simulated data as well as producing predictions of the response variables. However, the use of a basis function to transform the input data improves performance of the linear regression method. This can be seen in the table as well. We now obtain a significantly better model for the data after using a $\phi(X) = log(X)$ transformation. After such a transformation the linear least squares method performs almost as well as the neural network.

The result of the fitted neural networks is that one and the same model obtains the best results when analysing the MSE, the MAE and the coefficient of determination. The results are also better than the fit from the linear regression model, either with or without the transformation of input data (see table 3).

Table 3: Performance Measures - Data set 2.

| Type | K | L | MSE | MAE | $R^2$ |
|---|---|---|---|---|---|
| LR-model | - | - | 0.29314 | 0.42041 | 0.5812 |
| LR-model, transformed | - | - | 0.05843 | 0.19345 | 0.91541 |
| NN (MSE) | 2 | 2 | 0.0304 | 0.1467 | 0.9583 |
| NN (MAE) | 2 | 2 | 0.0304 | 0.1467 | 0.9583 |

Table 4: Topp 10 neural networks with lowest MSE for dataset 2.

| K | L | MSE | MAE | $R^2$ |
|---|---|---|---|---|
| 2 | 2 | 0.0304 | 0.1467 | 0.9583 |
| 5 | 4 | 0.0307 | 0.1472 | 0.9579 |
| 4 | 2 | 0.0309 | 0.1471 | 0.9577 |
| 10 | 2 | 0.0311 | 0.1472 | 0.9573 |
| 5 | 2 | 0.0314 | 0.1506 | 0.9569 |
| 3 | 3 | 0.0316 | 0.1499 | 0.9566 |
| 9 | 4 | 0.0317 | 0.1527 | 0.9565 |
| 2 | 8 | 0.0321 | 0.154 | 0.9559 |
| 2 | 9 | 0.0321 | 0.1541 | 0.956 |
| 2 | 6 | 0.0322 | 0.1544 | 0.9558 |

## 5.3 Third Data Set - Nonlinear Regression II

The final data set is generated from another non-linear regression model, with a more complex characteristic. In the plot where we analyse actual against predicted values, we observe a similar situation as with the previous data set. The linear regression method gives a poor fit to the data (Figure 3). However, applying a polynomial basis function transformation increases the accuracy to fit a model using the linear regression method (table 3). It does not match the performance of the neural network as can be compared in the same table.

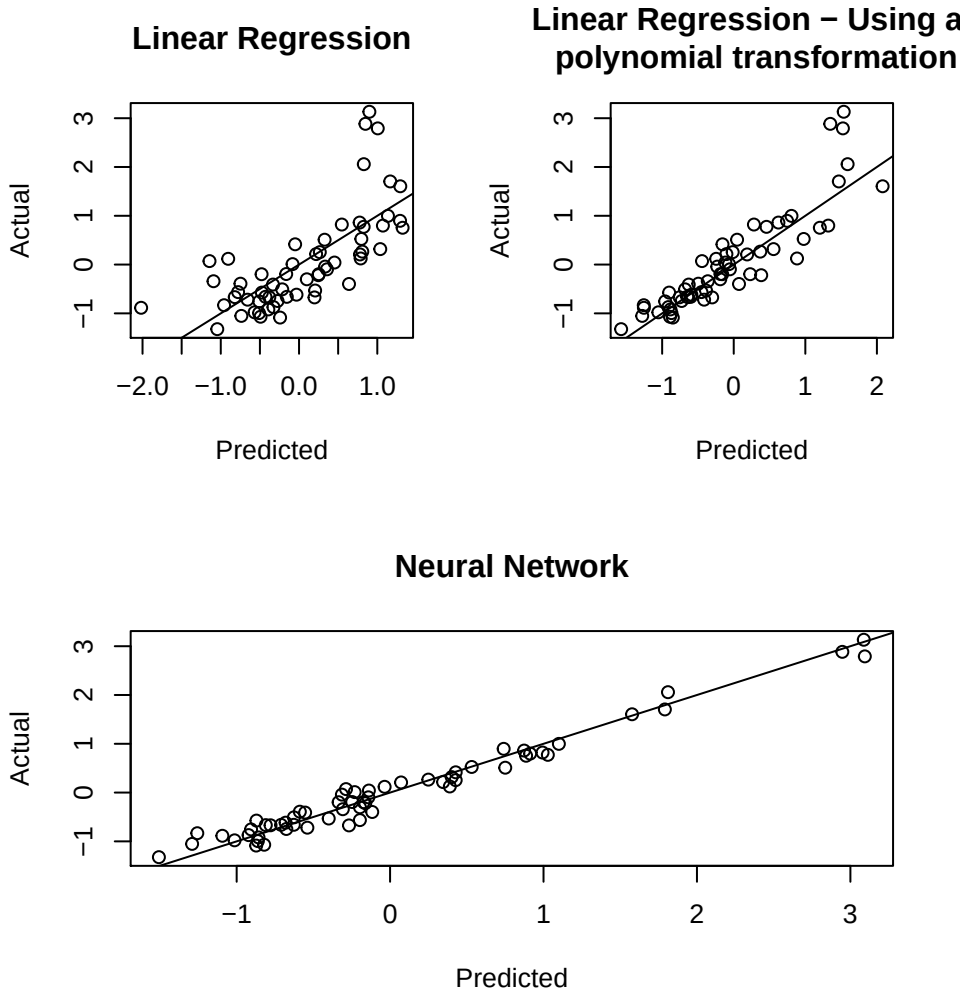


Figure 5: Comparisons of plots of actual versus predicted response variables for data set 3

The neural network with the best performance is shown below (see Table 3). We conclude that the neural network performs well again, with a coefficient of determination value of 0.97 which shows that such a model fits well to the test data. We get the same network structure for each of the performance measures which indicate a model that is applicable for prediction of response variables.

Table 5: Performance Measures - Data set 3.

| Type | K | L | MSE | MAE | $R^2$ |
|---|---|---|---|---|---|
| LR-model | - | - | 0.47088 | 0.51291 | 0.51476 |
| LR-model, transformed | - | - | 0.19757 | 0.30585 | 0.79806 |
| NN (MSE) | 3 | 10 | 0.03 | 0.1341 | 0.9702 |
| NN (MAE) | 3 | 10 | 0.03 | 0.1341 | 0.9702 |

Table 6: Topp 10 neural networks with lowest MSE for dataset 3.

| K | L | MSE | MAE | $R^2$ |
|---|---|---|---|---|
| 3 | 10 | 0.03 | 0.1341 | 0.9702 |
| 5 | 10 | 0.0328 | 0.1423 | 0.9675 |
| 7 | 5 | 0.033 | 0.1455 | 0.9672 |
| 4 | 6 | 0.0346 | 0.1387 | 0.9656 |
| 4 | 7 | 0.0354 | 0.1578 | 0.9648 |
| 6 | 9 | 0.0357 | 0.1507 | 0.9645 |
| 8 | - | 0.03594 | 0.15276 | 0.96424 |
| 6 | 8 | 0.0362 | 0.1456 | 0.9641 |
| 6 | 5 | 0.0365 | 0.1506 | 0.9637 |
| 3 | 7 | 0.0378 | 0.1552 | 0.9624 |

# 6 Conclusion

The main goal for this thesis was to compare the two chosen methods, neural networks and linear regression, when fitting models to use for prediction. The theory part of the thesis showcases how a model is fitted for each of the methods and how the calculations of this fit are made. For a neural network model, the importance lies in the number of hidden layers and the number nodes for each of these layers, being referred to as the structure of the network. After we have chosen the structure of the network, we make the calculations of the weights that connect the nodes to each other. These weights are the parameters of the network and play an important role. A loss function is calculated when the calculations of the weights has been finished. This loss function is the minimization of the sum of squared errors. To be able to improve the prediction of a network model, we make use of the gradient descent in order to lower the error of the network.

The explanatory variables of the data sets were simulated from different distributions. This was made for each of the data sets. The response variables were then calculated using these explanatory variables in their own unique way for each of the data sets. This was made so that the three data sets had different characteristics in order to observe in what situations one of the methods are to prefer over the other.

To be able to compare the methods and to pick the best model, we calculate different performance measures. In this thesis, the performance measures were the $MSE$, $MAE$ and the coefficient of determination. These are calculated for each model in order to choose the best fit for the current dataset out of the three simulated data sets. Further, we presented the results with plots that compare the actual values of the response variables against what the chosen model predicts. This gives a clear view of the difference between the actual performance and the models. We also showcase the model, for each of the methods, with the best fit according to the performance measures.

Throughout the study, when analyzing performance measures, we obtained a high value for the coefficient of determination for the neural network model that was selected, all being above $R^2 > 0.95$. This indicates that the network model fits well to the data. Observing the results of the first linear data set suggests that the linear regression model is a better fit than the neural network model. Since data is simulated from a linear model, this comes as no surprise. The result of the second data set clearly suggests a neural network model over the linear regression model. However, by applying a log transformation type of basis function to one of the explanatory variables, the result of a transformed linear regression model is close to that of the neural network. Since a linear regression model is easier

to understand as well as to analyse, a transformed linear regression model can be a better pick even though the result is slightly worse than that of a neural network model. Lastly, the result of the third data set indicates that a neural network model fits better to data than the linear regression model. Here, the linear regression model obtains a small $R^2$ value while for the neural network $R^2$ still had high value.

# 7 Discussion

During testing, we also experimented whether a third hidden layer would be possible or not. This would result in ten times as many neural network models, meaning we would be able to choose from another 900 models. The main issue here was that fitting different models with three hidden layers was too time expensive. With this result, the thesis focused on one hidden layer or two hidden layers for the neural network models.

Observing the results of the second data set, a basis function applies well to the data and we are were able to obtain results for the linear regression model that were close to the results of the neural network. The question remains whether or not the performance of the two models is close enough for actually choosing the linear model. An advantage is that a linear regression model is much easier to analyse and understand and could therefore be a better choice.

For the third dataset, a basis function transformation did increase the performance of the linear regression model. However, there was a difference between the neural network model that fitted the best and the transformed linear model when comparing the coefficient of determination. So even though an appropriate basis function transformation for the linear model was found, the neural network still had a better performance and was preferable to use in comparison to the linear regression.

When simulating the data, the goal was to simulate the explanatory variables that were being used. We then generated the response variables by adding simulated error terms to a pre-chosen function for the mean response. Models are then fitted to training data and compared in order to choose the model that fit response variables of test data the best. This makes it possible to choose the best model as the one for which the outcome of data can be compared to an estimate of the expected value of the outcome. From the results, we can conclude that the models are able to predict well when tested with the test data set, but these findings might not generalize well to other data sets.

For further analysis, an interesting approach is to implement other machine learn-

ing methods to analyze the simulated data as done in this thesis. With comparisons of several methods and models, it can be simpler to outrule a method that does not fit the data well. This could very well be one of the methods above. In that way, comparing several methods is efficient in order to choose the best one and obtain a model that fits the data in the best way possible.

# References


[1] Christopher M. Bishop. *Pattern recognition and machine learning*, Springer, 2006.

[2] Trevor Hastie, Robert Tibshirani, Jerome Friedman. *The Elements of Statistical Learning*, Springer, 2nd Edition, 2017.

[3] Patrik Andersson, Kristoffer Lindensjö, Joanna Tyrcha. *Notes in Econometrics*, Department of Mathematics, Stockholm University, 2019.

[4] Sendhil Mullainathan and Jann Spiess *Machine Learning: An Applied Econometric Approach*, Journal of Economic Perspectives, 2017.

[5] Simon J. Sheather *A Modern Approach to Regression with R*, Springer, 2009.

[6] Frauke Günther and Stefan Fritsch *neuralnet: Training of Neural Networks*, The R Journal Vol. 2/1, June 2010.

[7] Iqbal H. Sarker *Machine Learning: Algorithms, Real-World Applications and Research Directions*, Springer Nature, 2021.

[8] Bohdan Macukow *Neural Networks – State of Art, Brief History, Basic Models and Architecture*, Faculty of Applied Mathematics and Information Science, 2016.

[9] T.Chai and R.R.Draxler *Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature*, NOAA Air Resources Laboratory, Cooperative Institute for Climate and Satellites, 2014