The Comparisons of Neural Networks and Regression Techniques

Simen Håpnes

(Dated: Sunday 10th November, 2019)

A numerical study regarding both a classification problem and a regression problem is presented. The classification problem aimed to reproduce the results from two data mining methods from the study of (Yeh & Lien)[3]. The data set was a credit card data set with 23 predictors and 30,000 data points and a binary output: whether or not the individuals would default their next credit card bill. The data was analyzed by neural networks (NN) and logistic regression (LR) and evaluated by error rates, area ratios in cumulative gain charts, and estimation of real probability of default. NN obtained better results than LR, which was shown by lower error rates and higher area ratios. The results from (Yeh & Lien) were partly reproduced. The NN obtained very similar area ratios, whereas LR obtained a slightly lower area ratio for the training data (0.37 vs. 0.41). This study did not achieve as low error rates as the previous study, and not very credible estimations of actual probability of default, due to low intercept coefficients of ≈ -0.2.

The regression problem was a data set of Franke's function (including Gaussian noise with $\sigma = 0.3$) with 40,000 data points, 2 predictors and 1 output. The aim was achieve the best R^2 score, by applying neural networks and a linear regression method, namely Ridge regression (RR). NN achieved an R^2 score of 0.48, which was only slightly better than RR, which achieved an R^2 score of 0.47. The NN converged slowly and did require several hundred epochs before it obtained a higher R^2 score than RR, which was a considerably slower algorithm.

I. INTRODUCTION

Machine learning is a very central topic in the modern world. There is a high value in the information that can be extracted from the vast amount of data that exists. Supervised learning is the case were an algorithm is learning from the data at hand and there are two main groups of problems. The first one is *classification*, were the outputs can be categorized in discrete groups, such as yes/no or 0-9. The second type of problem is *regression*, were the outputs takes continuous values, such as the speed of a car or the surface of the seafloor. The aim of the research presented, is to apply artificial neural networks on both a classification and a regression problem. For both problems, the neural networks will be compared with an additional method.

As a classification problem, a credit card data set from Taiwan 2006 will be studied. The data set is obtained from the *UCI Machine Learning Repository*[3]. The credit card data contains 30,000 data points, 23 predictors and 1 set of binary outputs: whether or not the individuals would default their credit card bill the next month. A previous study performed has been conducted on this subject, were six different data mining methods were performed on the credit card data set. In the research presented, two of those methods (logistic regression and artificial neural networks) will be implemented with the aim of reproducing the results of (Yeh & Lien)[3].

For the regression problem, a generated data set of Franke's function will be studied. Franke's function takes two input values x, y and produces a single and continuous output value $f(x, y)$. It is essentially a 3D surface consisting of a sum of three Gaussian peaks and one dip. The data set of Franke's function will include additional noise as an irreducible error. This data set will be fitted by applying both a neural network and a linear regression model, namely Ridge regression.

The intention of this research is to explore the strengths and weaknesses of all four models. It is especially a goal to identify the neural networks' predictive ability as opposed to the other two methods: which model performs best in the classification case and in the regression case?

In the next section, the necessary theory behind the models will be introduced. The longer and more mathematical derivations are added in the appendix as additional theory for the interested reader. The method section will in detail describe the algorithms for the models and the parameters used. In the result and discussion sections, the results of this research will be presented and examined. Finally, the conclusion will summarize the key insights and conclusions found in the study presented, as well as a suggestions for further research in this subject. For reproducibility or further research, the codes and algorithms, figures and results, may also be found in my GitHub repository at https://github.com/simehaa/MachineLearning/tree/master/NeuralNetworks.

II. THEORY

The aim of a regression problem is to predict outputs which can take continuous values, unlike a classification problem, where the outputs takes discrete variables. In the research presented, both regression and classification data sets will be studied, therefore the theory section is split in these two parts.

A. Classification

There are many machine learning options which can be applied to classification problems. (Yeh & Lien, 2009)[3] applied six different methods on a credit card data set from Taiwan with 23 predictors and 25,000 data points.

The conclusion of (Yeh & Lien)[3] was that artificial neural networks

1. showed the best performance based on R^2 scores, intercept and slope of linear regressions between predicted probability vs. actual probability.
2. produced a predictive default probability which is the only one that could be used to represent real probability of default.
3. should be employed to score clients instead of other data mining techniques, such as logistic regression.

This research aims to reproduce the results from two of the methods from (Yeh & Lien)[3]: *logistic regression* (LR) and *artificial neural networks* (ANNs).

It is necessary with methods to evaluate the performance of the models. (Yeh & Lien, 2009)[3] applied three different measurement of performance

1. Error rates in predictions.
2. Area ratio in cumulative gain curves.
3. R^2-score of the linear regression of real probability vs. predictive probability. The latter was estimated by using the Sorting Smoothing Method.

In this research, all three measurements will be reproduced, it is therefore necessary to explain the methodology behind these methods. The first and most simple metric is the error rate. It is given by

$$\begin{aligned} Err &= 1 - A \qquad (1) \\ &= 1 - \frac{1}{n}\sum_{i=1}^{n} I(\hat{y}_i, y_i), \qquad (2) \end{aligned}$$

where A is the accuracy score, n is the number of data points and the indicator function is given as

$$I(\hat{y}_i, y_i) = \begin{cases} 1 & \hat{y}_i = y_i \\ 0 & \hat{y}_i \neq y_i \end{cases} \qquad (3)$$

The error rate is simply a measure of what fraction of the predicted outputs $\hat{y}_i$ are not equal to the correct outputs y_i. This evaluation can provide an insight about the predictive abilities of the two models, but not an actual probability of default on the individual level.

The second method which will be used, is the area ratio in cumulative gain charts. As explained in the report of (Yeh & Lien)[3], the horizontal axis represents the total number of data and the vertical axis represents the cumulative number of target data. There are three curves in this diagram: the theoretical best curve C_{best}, a diagonal baseline curve C_{base} and the model curve C_{model}. The area ratio is defined as

$$\text{Area ratio} = \frac{\text{area between } C_{model} \text{ and } C_{base}}{\text{area between } C_{best} \text{ and } C_{base}}. \qquad (4)$$

The higher the area ratio is, the better the model. The last method of evaluation is to estimate the real probability of default. This will be done by the Sorting Smoothing Method (SSM). The approach is to use the continuous outputs from the model. These values are called the *predictive probabilities* and will be values in the range $[0, 1]$. These data points can be ordered from minimum to maximum. The *estimated probabilities* can be calculated by

$$P_i = \frac{1}{2n+1}\sum_{j=i-n}^{i+n} Y_j, \qquad (5)$$

where Y_i is the binary value of the default in the ith order of the sorted predictive probabilities. n is a fixed integer which decides how large neighborhood of data points to use for the smoothing: (Yeh & Lien) used $n = 50$. Lastly the estimated probability will be plotted vs. the predictive probability. This scatter plot will be fitted with a linear regression of a 1st order polynomial

$$y = ax + b, \qquad (6)$$

where a good model will have $a \approx 1$ and $b \approx 0$. These linear fits will also be evaluated with R^2 scores. In the linear regression part of the theory section, R^2 scores will be explained in more detail.

In the next two subsections, both logistic regression and neural networks for classification problem will be explained. The choice of cost function for the classification problem will be cross entropy. According to (Nielsen)[2], cross entropy is a better choice of cost function, provided that the output neurons are Sigmoid neurons. This cost function also avoids a learning slowdown which occurs in the case of a quadratic cost function.

1. Logistic Regression

Logistic regression (LR) is a statistical model that aims to predict discrete outputs. The method for logistic regression which will be applied is the stochastic gradient descent (SGD) with mini-batches. This is a method which is solved iteratively, with an initial guess for the parameter vector β. The derivation of SGD with mini-batches is also shown in section VII: appendix A. The iterative algorithm for SGD with a general cost function is given by

$$\beta^t_{opt} = \beta^{t-1}_{opt} - \frac{\eta}{m}\sum_{i\in B_k}\frac{\partial}{\partial\beta}c_i(x_i, \beta). \qquad (7)$$

where B_k denotes one mini-batch which is a subset of all the data points. m is the number of mini batches, and η is the learning rate. The cost function c_i can for instance be the cross entropy which is derived in section VII. The iterative algorithm for SGD with cross entropy as a cost function, is given by

$$\beta^t_{opt} = \beta^{t-1}_{opt} - \frac{\eta}{m}\sum_{i\in B_k} X_i^T(p_i - y_i), \qquad (8)$$

where $X_i^T, p_{,i}$ and y_i, which only contains one mini-batch indicated with the index i, which is randomly chosen m times during one epoch.

2. Neural Network

A feed forward neural network (FFNN) is a neural network where the information only moves forward from one layer to next layer. There are no loops or information going backwards. It essentially works by applying SGD between each neighboring layers, and it must have at least one hidden layer to differ from the regular logistic regression. It can obtain a high predictability with several hidden layers.

The input layer has a given number of features which is analogous to the number of nodes in the hidden layers. The output layer has a fixed number of outputs (e.g. 10 in the case of digit recognition). When theses numbers are fixed, the neural network can be trained with training data.

Between each pair of layers (including the input and output layer) there are a set of weights and biases. In the case of a fully connected FFNN, all inputs are linked to all nodes in the next layers, thus fully connected. The feed forward algorithm works by applying

$$a^l = f(z^l) = f\left(W^l y^{l-1} + b^l\right). \tag{9}$$

where $y^l, b^l \in \mathbb{R}^n$ and $y^{l-1} \in \mathbb{R}^m$ are column vectors and $W^l \in \mathbb{R}^{n\times m}$ is a matrix. The output of layer l is given by the previous layer $l-1$, then this can be used as input in the next layer. This process can be repeated until the output in the last layer, L is found. f is the activation function, which often is the Sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}}, \tag{10}$$

$$\frac{\mathrm{d}}{\mathrm{d}x}\sigma(x) = \sigma(x)(1-\sigma(x)). \tag{11}$$

The Sigmoid function works element-wise on the x if it is a vector. And whichever values x contain, $f(x)$ maps those values in the range $[0,1]$. The activation function can also be the hyperbolic tangent

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{12}$$

$$\frac{\mathrm{d}}{\mathrm{d}x}\tanh(x) = 1 - \tanh(x)^2. \tag{13}$$

The hyperbolic tangent maps the input x to an output value in the range $[-1,1]$. There are more options for activation functions, but only these will be used in the research presented. The choice of activation function should correspond with the way that the data is preprocessed. For instance, the input data could be processed with a so-called MinMaxScaler, in which all the data is in a certain range, e.g. $[0,1]$. With this scaling, the Sigmoid function is relevant.

The goal of the FFNN is to train the network so that it can accurately predict the correct output for new data points which it has never before seen. That is done tweaking the weights and biases, which again is done by the *back-propagation algorithm.* This algorithm can be summarized with four equations

$$\delta^L = \nabla_a C \odot f'(z^L), \tag{14}$$

$$\delta^l = \left(\left[w^{l+1}\right]^T \delta^{l+1}\right) \odot f'(z^l), \tag{15}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l, \tag{16}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l. \tag{17}$$

L denotes the output layer and l denotes all the other layers. f' represents the derivative of the activation function and ∇_a represents the derivative of the cost function wrt. a. Before these equations can be evaluated, all z^l and a^l must be evaluated through the feed forward algorithm described in equation 9. The weights and biases are updated by

$$w_{jk,new}^l = w_{jk,old}^l - \eta\frac{\partial C}{\partial w_{jk}^l}, \tag{18}$$

$$b_{j,new}^l = b_{j,old}^l - \eta\frac{\partial C}{\partial b_j^l}, \tag{19}$$

where η is the learning rate and C is the cost function.

The choice of cost function will be cross entropy with L2 regularization, which is given by

$$C = -\frac{1}{n}\sum_j \left[y_j \ln a_j^L + (1-y_j)\ln\left(1-a_j^L\right)\right] + \frac{\lambda}{2n}\sum_w w^2. \tag{20}$$

The first term is the negative log-likelihood, which is closely related to cross entropy. The second term is the regularization term, where λ is the L2 regularization parameter. w denotes all the weights.

It is necessary to know the derivative of C wrt. a, which turns out to be a very simple expression,

$$\nabla_a C = a - y, \tag{21}$$

in the case of C being cross entropy.

Another aspect of the backpropagation algorithm, is to introduce stochasticity, by the introduction of mini batches. This results in faster convergence and avoids the problem of local cost function minima. Consider that the number of data points in one batch is m, and hence the number of mini batches is $M = n/m$. During one epoch, there is an inner loop with M iterations. Each time, a random data sample of size m is picked. Further, the backpropagation algorithm is applied on that mini batch to yield a small step for all weights and biases, given by equations 18 and 19.

When the inner loop is finished, all the biases are updated by taking the average step of which all mini batches yielded

$$b^l_{j,new} = b^l_{j,old} - \frac{\eta}{M}\sum_{batches}\frac{\partial C}{\partial b^l_j}. \tag{22}$$

For the weights, the same is done, taking the average step from all the mini batches. Additionally, the L2 regularization from equation 20 now enters the term which updates the weights

$$w^l_{jk,new} = w^l_{jk,old}\left(1 - \frac{\eta\lambda}{n}\right) - \frac{\eta}{M}\sum_{batches}\frac{\partial C}{\partial w_{jk}}. \tag{23}$$

One epoch in a neural network can be summarized in these steps:

1. Split the data into mini batches. For each mini batch:
    - Do the feed forward algorithm in equation 9.
    - Do the backpropagation, described in equations 14 to 17.
2. Update weights and biases according to equation 22 and 23.

B. Regression

The aim of regression is to compare the predictive abilities of neural networks (NN) and a linear regression method. The data that will be used is a data set with two predictors x_1 and x_2 and a single output. This data will be generated by using the Franke function, with both $x_1, x_2 \in [0, 1]$ and the output $y = f(x_1, x_2)$ is a sum of four Gaussian distributions with different centers, standard deviations and heights.

The cost function in the case of regression problems, will be the mean squared error (MSE) as the goal is to provide a model that can as best as possible model the underlying function. When a prediction a is provided, the MSE is defined as

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(a_i - y_i)^2. \tag{24}$$

After each model has provided a prediction, a possible evaluation method is the R^2 score, which is defined by

$$RSS = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{25}$$

$$TSS = \sum_{i=1}^{N}(y_i - \bar{y})^2, \tag{26}$$

$$R^2 = 1 - \frac{RSS}{TSS}. \tag{27}$$

1. *Linear Regression*

Concerning linear regression, a previous study on the Franke function has been conducted[1], with design matrices consisting of polynomial terms of x_1 and x_2. The conclusions of this study were that Ridge regression provided the most accurate results, and therefore this method will be reapplied in the research presented. The parameters that may be varied in Ridge regression in this case, are the regularization parameter and the polynomial degree of the design matrix.

2. *Neural Network*

The neural network will be trained by using only the two predictors x_1 and x_2 and a single output. The method that the network can be trained is nearly the same as in the classification case, presented earlier in this section. What is different in this case, is most importantly the cost function, which now is the MSE. The activation functions between each hidden layer can still be e.g. the sigmoid function, tanh, etc, but between the last hidden layer and the output layer, the activation function must be a linear function, because this model is going to predict a linear regression problem.

The cost function comes to play in the first part of the backpropagation algorithm, which is showed in equation 14. It is necessary to know $\nabla_a C$, which in the case of cross entropy was

$$\nabla_a C = a - y. \tag{28}$$

What is interesting here, is the realization that $\nabla_a C$ for MSE can become identical to $\nabla_a C$ for cross entropy. If the expression for MSE is scaled to

$$\frac{N}{2}MSE = 0.5\sum_{i=1}^{N}(a_i - y_i)^2, \tag{29}$$

the derivative of MSE is also rescaled. On linearized form, the derivative of MSE wrt. a is now identical to equation 28. There is no problem associated with rescaling the MSE, because the overall goal is to minimize it. In fact, the actual cost function is now $RSS/2$ (residual sum of squares).

III. METHOD

A. Classification

The data set which was analysed was credit card data from Taiwan with 23 different attributes and 30,000 data points. The classification problem was to predict customers' default payments. In the research presented, two methods are presented: *logistic regression* (LR) and *artificial neural networks* (NN).

1. Processing and Description of the Data Set

The credit card data set contained 23 predictors. The outputs where one column of binary outputs, *default payment next month* where 1 means yes and 0 means no.

The processing of the data set was done with the aim of scaling the data and removing outliers.

The outliers were removed by simply deleting entire rows containing at least one value which were not defined in the data set. In total, 399 rows were deleted, resulting in 29601 unique data points.

Among the 23 predictors, some of the columns were categorical and some were continuous. The purely categorical columns: *gender, marital status* and *education* were one-hot encoded and placed first in the processed data set. These columns resulted in 9 one-hot encoded columns containing values 0 or 1:

- 0, SEX: Gender (1=male)
- 1, SEX: Gender (1=female)
- 2, EDUCATION: (1=graduate school)
- 3, EDUCATION: (1=university)
- 4, EDUCATION: (1=high school)
- 5, EDUCATION: (1=others)
- 6, MARRIAGE: Marital status (1=married)
- 7, MARRIAGE: Marital status (1=single)
- 8, MARRIAGE: Marital status (1=others)

Further, column 6-11 in the original data set contained integer values in the range $[-2, 9]$, thus also categorical.

- The values $\{-2, -1, 0\}$ were three different categories that all describes that the bill has been paid (-2: credit card was inactive in period, -1: the due amount and balance paid fully, 0: the minimum due amount paid)
- The integers $\in [1, 9]$ described payment delay in months.

In order to avoid a too high number of one-hot encoded columns, the first three values $\{-2, -1, 0\}$ were interpreted as the same category (bill paid) and one-hot encoded, thus the next 12 columns were

- 9, PAY_0: Repayment status (1=paid), Sep. 05
- 10, PAY_2: Repayment status (1=paid), Aug. 05
- 11, PAY_3: Repayment status (1=paid), July 05
- 12, PAY_4: Repayment status (1=paid), June 05
- 13, PAY_5: Repayment status (1=paid), May 05
- 14, PAY_6: Repayment status (1=paid), Apr. 05
- 15, PAY_0: Repayment status (1=not paid), Sep. 05
- 16, PAY_2: Repayment status (1=not paid), Aug. 05
- 17, PAY_3: Repayment status (1=not paid), July 05
- 18, PAY_4: Repayment status (1=not paid), June 05
- 19, PAY_5: Repayment status (1=not paid), May 05
- 20, PAY_6: Repayment status (1=not paid), Apr. 05

The latter values (1-9) were treated as continuous variables because these described something of the same unit, which in this case was *payment delay in months.*

The data set was split into a training set and test set of 80% and 20% of the data set, respectively. The training set was upscaled so that the number of rows with output $y_i = 0$ equal the number of row with output $y_i = 1$. This was not done for the test set.

Lastly, the entire data set was scaled by using a MinMaxScaler. Every continuous columns (and also the categorical columns), now contained values within the range $[0, 1]$. The remaining columns were

- 21, LIMIT_BAL: Am. of given credit in NT dollars
- 22, AGE: Age of credit card holder
- 23, PAY_0: Payment delay in Sep. 05
- 24, PAY_2: Payment delay in Aug. 05
- 25, PAY_3: Payment delay in July 05
- 26, PAY_4: Payment delay in June 05
- 27, PAY_5: Payment delay in May 05
- 28, PAY_6: Payment delay in Apr. 05
- 29, BILL_AMT1: Am. of bill statement in Sep. 05
- 30, BILL_AMT2: Am. of bill statement in Aug. 05
- 31, BILL_AMT3: Am. of bill statement in July 05
- 32, BILL_AMT4: Am. of bill statement in June 05
- 33, BILL_AMT5: Am. of bill statement in May 05
- 34, BILL_AMT6: Am. of bill statement in Apr. 05
- 35, PAY_AMT1: Am. of prev. payment in Sep. 05
- 36, PAY_AMT2: Am. of prev. payment in Aug. 05
- 37, PAY_AMT3: Am. of prev. payment in July 05
- 38, PAY_AMT4: Am. of prev. payment in June 05
- 39, PAY_AMT5: Am. of prev. payment in May 05
- 40, PAY_AMT6: Am. of prev. payment in Apr. 05

The scaling parameters were obtained from the training set, and both sets were scaled according to the training set. An important note regarding the scaling is that the training set was completely independent of the test set, but the test set is scaled according to the training set. In total, the processed data set contained 41 predictors and 29601 unique data points.

2. Logistic Regression

The logistic regression method that was applied and implemented in Python was *stochastic gradient descent* (SGD) with mini-batches. This was an iterative algorithm and the general stucture of the algorithm was

1. Random weights and bias was initialized
2. Loop over epochs/max no. of iterations
3. Inner loop over n mini batches, where one random mini batches is chosen each time (and resampling is allowed)
   (a) Evaluate p: $p = \frac{1}{1+\exp\{-X_i\beta+b\}}$
   (b) Adjust bias: $b^{new} = b^{old} - ave(p_i - y_i)$
   (c) Adjust weights $\beta^{new} = \beta^{old} - \gamma X_i^T(p - y)$

The weights are updated according to the cost function *cross entropy.* The parameter γ was the learning rate. This was set to be $1/(t + 10)$ where $t \in [0, epochs)$. This resulted in the learning rate becoming smaller throughout the epochs.

The SGD algorithm was trained with the training data and tested with both the training data and the test data. A wide variety of epochs were tested by trial and error with values from 100 to 3000 in order to produce the best results. Finally, a constant number of 1000 epochs was set.

The evaluation of the model consisted of obtaining the error rates and area ratios for both the test set and training set. Additionally, for the test set, the cumulative gain curve was plotted and the estimated probability was plotted vs. predictive probability as a scatter plot along with the corresponding linear regression.

3. Neural Network

A feed forward neural network was implemented by utilizing the back propagation algorithm. The NN was trained by using Sigmoid as activation functions between each hidden layer and by using cross entropy with L2 regularization as cost function. The mini batch size were set to 100.

Another parameter to consider, was the number of epochs. 10 % of the training data was set aside as validation data This analysis found that the accuracy in the validation data did not increase after around 10 epochs. Therefore, for the remaining analyses the number of epochs was kept fixed at 10, and the validation data was included as training data.

The layers were varied by trial and error, and one combination that seemed to provide stable and good results was a layer setup of **[41, 100, 75, 50, 34, 1]**, including the input and output layer. This setup was created by using a fairly high complexity to begin with (100 nodes in the first hidden layer), and then following roughly the 2/3 rule of slowly decreasing the number of nodes.

The L2 regularization parameter, λ and the learning rate, η were two parameters to be determined. In order to find the best combination, there was performed a grid search: the neural network was trained for all combinations of

$$log_{10}(\eta) \in \{0, -1, -2, -3\} \tag{30}$$

$$log_{10}(\lambda) \in \{-1, -2, -3, -4, -5, -6\} \tag{31}$$

For each trained neural network, the area ratio was determined both for the training set and the test set, and the parameters that resulted in the highest area ratio, were recorded. A heat map for the area ratios in the test data was plotted.

For the best combination of parameters for both the training and test set, both the area ratio and error rate were recorded. The cumulative gain curve of the test set was plotted, and the estimated probability vs. predictive probability was plotted as a scatter plot along with the corresponding linear regression.

B. Regression

1. Generating a data set

The matrices $X_{train} \in \mathbb{R}^{30,000\times 2}$ and $X_{test} \in \mathbb{R}^{10,000\times 2}$ was initialized. The two columns in these matrices correspond to x_1 and x_2, which were generated by using a uniform distribution in the range $[0, 1]$. The output vectors y_{train} and y_{test} was initialized by computing the Franke function with the values in X_{train} and X_{test}, respectively. There was an additional noise added to the y vectors with a standard deviation of 0.3 and a variance of 0.09.

2. Linear Regression

Ridge regression was applied to the training set. The matrix X_{train} were modified by using polynomial design matrices, with various polynomial degrees. The regression was performed by using 5-fold cross validation on the training data to obtain a stable estimate of the R^2 score.

The Ridge regression now has two parameters that can be adjusted, the polynomial degree of the design matrix and the L2 regularization parameter. In order to find the best set of parameters, the R^2 score was evaluated (by 5-fold cross validation) for grid of various polynomial degrees and regularization parameters. The polynomial degrees were varied in the range $[7, 15]$. The regularization parameters were varied in the range $[10^{-5}, 10^{-13}]$. The results were plotted as a heat map over R^2 scores.

3. Neural Network: Regression

The neural network was set up by redefining the cost function from cross entropy to mean squared error (MSE). Now the number of hidden layers and nodes per hidden layer were experimented with by trial and error. One setup that provided stable and good results were a layer setup of **[2, 100, 60, 1]**, including the input and output layer.

There are many options for activation functions, but because this is a linear regression problem, the activation function for the output layer must be a linear function. The activation function for the output layer was set to be $f(a) = a$, which is the most simple linear activation function and equivalent to no activation function at all.

For the hidden layers however, there are many options. In order to see how the neural network evolved during the epochs, five different activation functions (Sigmoid, *tanh*, ReLU, ReLU6 and Leaky ReLU) were tried. The time evolution of neural networks on a Franke function data set (without noise) were animated. By studying these animations (`https://github.com/simehaa/MachineLearning/tree/master/NeuralNetworks/animations`), ReLu and ReLU6 were the most unstable activation functions for this problem or this setup. Sigmoid did not quite capture the curves of the Franke function, whereas both *tanh* and leaky ReLu seemed to provide quite good results. Finally, the activation function for the two hidden layers was chosen to be *tanh*.

There are three parameters of the neural network which must be found in order to obtained the best regression: the number of epochs, learning rate and L2 regularization parameter. The learning rate and regularization parameter was found by using a grid search of various combinations with

- Learning rates $\in \{0.01, 1e-3\}$.
- Regularization parameters $\in \{0.1, 0.01, 1e-3, 1e-4\}$.

For all different combinations of learning rates and regularization parameters, the neural networks were trained for a total of 750 epochs. The behaviour of the number of epochs was explored by plotting the cost function (MSE) of the test set vs. the epoch number.

When the NN is not very trained, the model suffers from a high bias, but this is a value that decreases as the model becomes better. At the same time, the variance in the model increases over time. It can be shown that $MSE = Bias^2 + Var + \sigma^2$, where σ^2 is some irreducible error: essentially the noise in the data set. Since the goal is to minimize MSE, one must minimize the sum of $Bias^2$ and Var: this is called the Bias-Variance trade-off. When plotting MSE vs. epochs, the aim was to identify if overfitting would occur, which could happen if the NN starts to model the noise in the training data. This would lead to an increased variance, and hence MSE in the test data.

For the grid search of parameters, the R^2 score was recorded for each NN and plotted as a heat map. The best R^2 scores obtained from Ridge regression and neural networks were compared.

IV. RESULTS

A. Classification

Table I shows the obtained error rates for both logistic regression and neural networks on the credit card data set.

TABLE I. Logistic regression and a neural network model performed on the credit card data set. Error rates and area ratios from previous research conducted by (Yeh & Lien)[3] are included for comparison.

Method	Error rate		Area ratio	
	Training	Test	Training	Test
Logistic regression with SGD	0.29	0.22	0.37	0.44
Logistic regression (Yeh & Lien)	0.20	0.18	0.41	0.44
Neural network	0.29	0.21	0.55	0.54
Neural network (Yeh & Lien)	0.19	0.17	0.55	0.54

Figure 1 and 2 shows the cumulative gain curves of the test set for LR and NN, respectively.

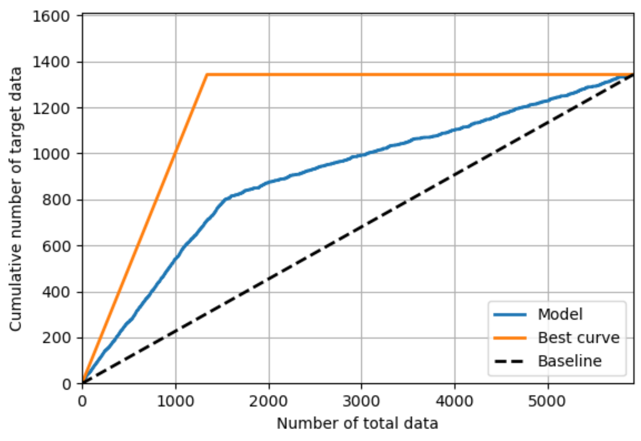


FIG. 1. Cumulative gain chart of the logistic regression model, performed with SGD with 1200 epochs and decreasing learning rate. This figure represents the test set.

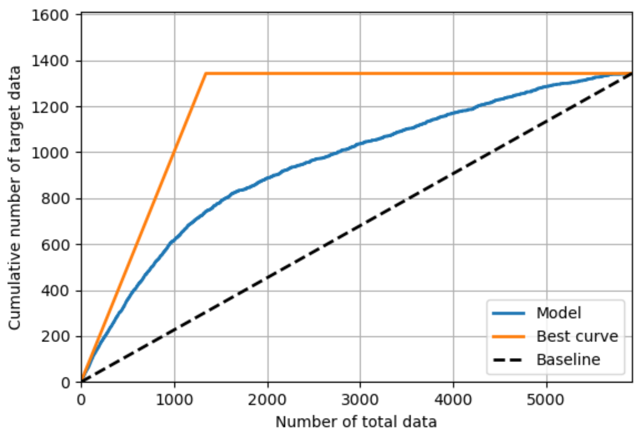
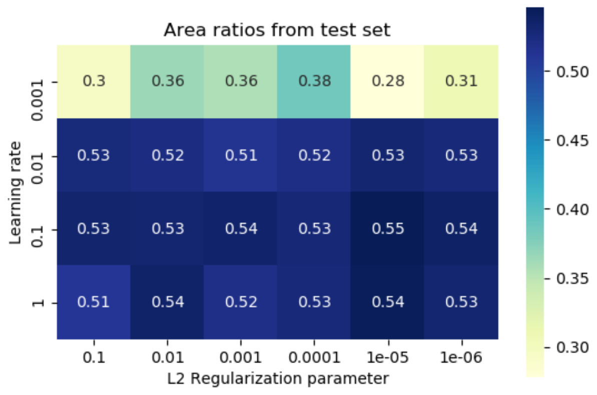


FIG. 2. Cumulative gain chart of the neural network model. This figure represents the test set. In the training of the network, the learning rate used was $\eta = 0.1$ and the regularization parameter was $= 10^{-5}$.

Figure 3 shows a heat map over area ratios of the test set, obtained for a grid of various L2 regularization parameters and learning rates. Only the heat map of the test set is included, because the training set gave very similar results.



Learning rate \ L2	0.1	0.01	0.001	0.0001	1e-05	1e-06
0.001	0.3	0.36	0.36	0.38	0.28	0.31
0.01	0.53	0.52	0.51	0.52	0.53	0.53
0.1	0.53	0.53	0.54	0.53	0.55	0.54
1	0.51	0.54	0.52	0.53	0.54	0.53

FIG. 3. Heat map over area ratios for test set for various combinations of regularization parameters and learning rates.

Figure 4 and 5 shows the estimated actual probability vs. the predicted probability for logistic regression and neural network. The estimated actual probability was calculated by using the Sorting Smoothing Method with $n = 50$. Additionally a linear regression line is included, which was calculated by ordinary least squares method.

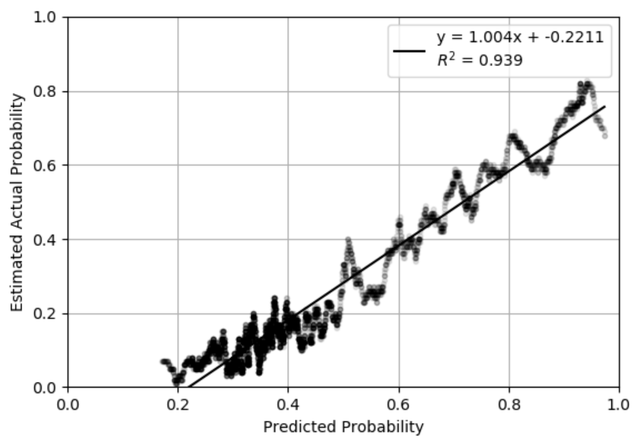


FIG. 4. The estimated actual probability (calculated by SSM) vs. the predictive probability for the test set and logistic regression. The figure also show a line fit, obtained by using the ordinary least squares method and the regression R^2 score.

Figure 5 was created by using the test set and the neural network with the optimal parameters (those which obtained the best area ratio): learning rate $\eta = 0.1$ and L2 regularization $\lambda = 1e-5$. The activation functions and hidden layers were the same as before.

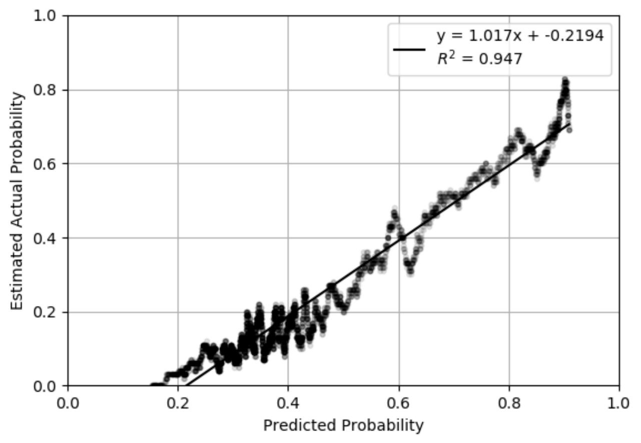


FIG. 5. The estimated actual probability (calculated by SSM) vs. the predictive probability for the test set and neural networks. The figure also show a line fit, obtained by using the ordinary least squares method and the regression R^2 score.

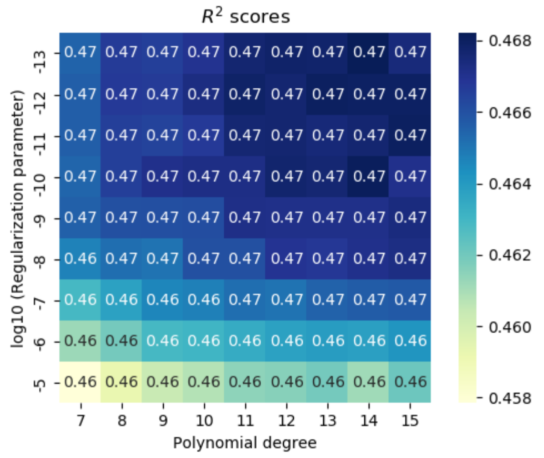
Table II shows a summary of the linear regressions applied in figure 4 and 5, as well as the results from (Yeh & Lien)[3] for comparison.

TABLE II. Summary of the linear regression on estimated real probability vs. predictive probability. The data is the test set of the credit card data set. Results from previous research conducted by (Yeh & Lien)[3] are included for comparison.

Method	Regr. Coefficient	Regr. Intercept	Regr. R^2
Logistic regression	1.004	−0.2211	0.939
Logistic regression (Yeh & Lien)	1.233	−0.0523	0.794
Neural network	1.017	−0.2194	0.947
Neural network (Yeh & Lien)	0.998	0.0145	0.965

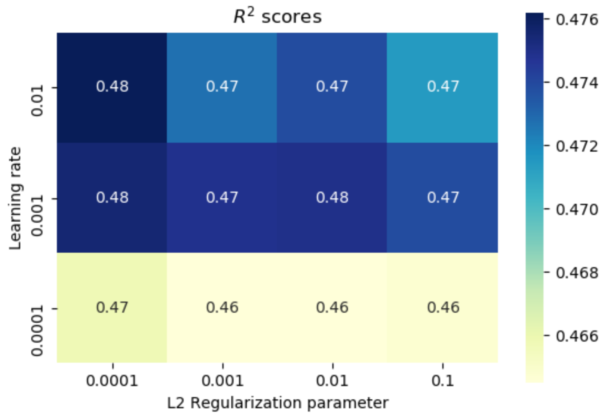
B. Regression

Figure 6 and 7 shows heat maps over R^2 scores for Ridge regression and neural networks, respectively.



log10 (Regularization parameter)	7	8	9	10	11	12	13	14	15
-13	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47
-12	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47
-11	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47
-10	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47
-9	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47
-8	0.46	0.47	0.47	0.47	0.47	0.47	0.47	0.47	0.47
-7	0.46	0.46	0.46	0.46	0.47	0.47	0.47	0.47	0.47
-6	0.46	0.46	0.46	0.46	0.46	0.46	0.46	0.46	0.46
-5	0.46	0.46	0.46	0.46	0.46	0.46	0.46	0.46	0.46

FIG. 6. Heat map over various R^2 scores obtained by using 5-fold CV and Ridge regression.



Learning rate	0.0001	0.001	0.01	0.1
0.01	0.48	0.47	0.47	0.47
0.001	0.48	0.47	0.48	0.47
0.0001	0.47	0.46	0.46	0.46

FIG. 7. Heat map over various R^2 scores obtained by using a neural network with 2 hidden layers with 100 and 60 nodes respectively, and 300 epochs.

Figure 8 shows the MSE for the various neural networks vs. epoch number. The minimum MSEs are also indicated with red dots.

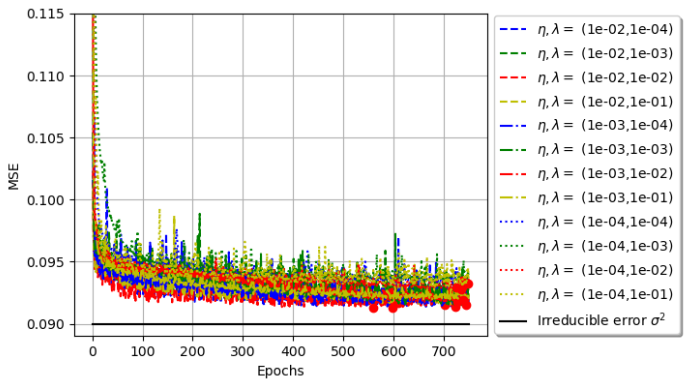


FIG. 8. Neural networks: the evolution of the MSE of the test set vs. the epoch number for a grid of various learning rates and regularization parameters.

Table III shows the obtained R^2 scores from the best combinations of parameters.

TABLE III. Linear regression and neural networks performed on the Franke function data set. The best R^2 scores of the parameters of the respective models are shown.

Method	R^2-Score	Regularization Parameter	Learning rate	Polynomial degree
Ridge regression	0.47	$1E-10$	–	14
Neural network	0.48	$1E-4$	0.01	–

V. DISCUSSION

A. Classification

Table I shows the achieved error rates and area ratios obtained from both logistic regression and from neural networks. The training errors are significantly worse than the results of (Yeh & Lien). The test errors were also worse, but closer to the previous research. The obtained area ratios were quite similar for all values, except for the training area ratio of LR which was 0.37 (the previous study obtained 0.41). Figure 1 and 2 shows the cumulative gain charts for the test set for both methods, LR and NN. These appear quite similar as to the results of (Yeh & Lien), and as discussed above, the area ratios are also close to those results.

Figure 4 and 5 shows the estimated probability (by the sorting smoothing method) vs. the predictive probability (from the models). The interesting part of these figures is the linear fit. A good model would result in a line with a slope close to 1 and an intercept close to 0. Another measure of a good model is a high R^2 score. These parameters are also shown in table II. Both LR and NN obtain a good regression coefficient close to 1 and high R^2 scores. An observation in the regressions, is that the results from LR and NN are not very different. (Yeh & Lien) found that NN performed much better than LR by this evaluation. In the study presented, LR obtains a much higher R^2 score (0.939) than the previous study (0.794). A very important feature for both LR and NN, was that the intercept was ≈ -0.2 which is quite a lot lower than 0 in this situation. It is unknown why this error occurred, but it is seemingly a systematic error. The reason behind this error is unknown at this point, but the consequence of it, is that this evaluation method gets a weaker credibility than error rates and area ratios.

Out of the two classification models, it is NN that performs best, because it achieves slightly better error rates and quite remarkably better area ratios. The reason behind this result is the fact that NN applies higher complexity with the introduction of several hidden layers that each performs SGD. Logistic regression on the other hand, only has one weight per feature in addition to the bias, and can only model some very clear relationships between the input layer and output layer. NN can, over the course of learning, model more complex relationships between the various features themselves.

B. Regression

Figure 6 shows the obtained R^2 scores for all parameter combinations for RR. All combinations resulted in an R^2 score of 0.46-0.47 which means that the results are fairly stable. Additionally, there is a general trend that indicates that higher polynomial degrees and lower regularization parameters results in better scores. Figure 7 shows the R^2 scores for all parameter combinations in NN, with 300 epochs. The two largest learning rates (0.01 and 0.001) provided the highest scores, but all scores were in the range 0.46-0.48.

Figure 8 shows the MSE for all 12 different set of combinations vs. the number of epochs trained. All NNs converged quickly for about 50 epochs, but beyond that, the trend is near linear and slow convergence. This indicates that the NN requires many epochs in order to improve the regression. The figure also indicates that the layer configuration of two hidden layers with 100 and 60 nodes, results in an underfitting of the data. If the network was deeper and the layers contained more nodes, the MSE would be expected to increase after a certain amount of training. This is not the case, therefore it is seemingly the bias which gradually decreases during the slow convergence.

Table III shows the best R^2 scores obtained from both Ridge regression (RR) and neural networks (NN). RR achieves a score of $R^2 = 0.47$, which is not a very high score generally speaking, but this can be explained by the noise in the data set. NN achieves a score of $R^2 = 0.48$, which is only better than RR by a very small margin. Therefore NN was better than RR in the regression problem, in terms of the result. There are of course more aspects to this comparison.

One aspect to consider is time. Ridge regression has a considerably shorter execution time, and it is also much faster to implement. A model is ready after just a few matrix-matrix multiplications (MMM) and one matrix inversion. The Neural networks on the other hand requires very many matrix-matrix multiplications, and other CPU expensive functions such as *tanh* for each epoch. In addition to that, it requires several hundred epochs in order to achieve a similar or better result than RR.

The other, and last aspect, to consider is the payoff between execution time vs. results. The time difference has already been discussed. The reward however, is a result which is only slightly better. Depending on the required precision desired, it might be considered an overkill to apply neural networks to a regression problem like the noisy Franke function. The payoff is small, and the cost is a much slower execution time.

C. Comparison of the Methods

Neural network performed well on the classification problem. It obtained better results than logistic regression with stochastic gradient descent. This is consistent with some of the final remarks of (Yeh & Lien), which also found that neural networks performed better. Some of the results from (Yeh & Lien) were reproduced and some were not. The cumulative gain charts and area ratios were reproduced, but the error rates were slightly worse for both neural networks and logistic regression, especially the training error. The estimated actual probability of default achieved good R^2 scores and slopes. On the other hand, they produced systematic low intercepts, which is problematic for the credibility of those results.

The regression problem was best solved by the neural network, but only by a small margin and at the cost of a much slower execution time. Therefore, it is very often the case that regression problems are most easily solved by applying linear regression methods, such as Ridge regression. Linear regression methods also contains fewer tuning parameters than neural networks. A final argument for linear regression methods, is that the theory behind them are much easier to understand, and hence are the results much easier to analyze.

D. Further Improvements

In the research presented, there have been some limitations to the methods applied. Therefore, some suggestions for future improvements will be presented.

Regarding the logistic regression, it is possible to modify the parameters in order to attempt to achieve better results. One could try various sizes of the mini batches, which in this research was kept constant at 100. Another possibility is to vary the learning rate, which can also be proportional to the gradient i.e. if the change in the weights are small, apply an accordingly small learning rate in the next epoch. Lastly, the number of epochs can be varied together along with the learning rate i.e. use more epochs with smaller learning rates.

Regarding the linear regression model, it is possible to explore other design matrices. The terms in the design matrix could include other terms than only polynomial expressions, such as exponential terms or Gaussian terms. When attempting to compare neural networks against linear regression methods, it is also important to study more than just Ridge regression, such as for instance LASSO regression. This has been done by (Håpnes & Hauser)[1], but not for this exact data set of the Franke function.

The most central topic of further analysis comprises in the study of neural networks. Both in the classification and regression problem, there are a high number of possibilities. Firstly, the number of hidden layers and the number of nodes per hidden layer can be varied infinitely. With computing power availability, one could go deeper and apply more hidden layers. Another important feature to explore, is which activation functions to use per hidden layers. In this research the Sigmoid function was used in classification and *tanh* was used in regression, but there are many more options, e.g. ReLU. It would also be interesting to explore adaptive learning rates and different types of regularization parameters, as these two parameters can improve the results remarkably.

VI. CONCLUSION

The classification problem in the study presented, aimed to reproduce the results from two data mining methods from the study of (Yeh & Lien)[3]. The data set was a credit card data set with 23 predictors and 30,000 data points, which was analyzed by neural networks (NN) and logistic regression (LR). The NN obtained better results than LR, which was shown by lower error rates and higher area ratios in cumulative gain charts. The results from (Yeh & Lien) were partly reproduced. The area ratios and cumulative gain charts were reproduced by a good accuracy. This research did not achieve as low error rates, especially for the training data. The estimations of actual probability of default lacks credibility due to a systematic low intercept, but it did achieve a better R^2 score for LR and a similar R^2 score for NN.

As a regression problem, a noisy data set of Franke function was analyzed by NN and Ridge regression (RR). NN achieved an R^2 score of 0.48, whereas RR achieved and R^2 score of 0.47. By considering the significantly longer execution time of neural networks, it might be more beneficial to apply linear regression methods instead of neural networks for regression problems.

Neural networks has been shown to be flexible in the sense that it can be applied to both classification and regression problems, but its predictive ability is most outstanding when applied on classification problems.

REFERENCES

[1] Håpnes, S. and Hauser, S. (2019). Regression methods on terrain data.
[2] Nielsen, M. A. (2018). Neural networks and deep learning.
[3] Yeh, I.-C. and hui Lien, C. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2):2473–2480.

VII. APPENDIX A: DERIVATION OF SGD WITH MINI BATCHES

Consider a dataset $\mathcal{D} = \{(x_i, y_i)\}$ of independent and identically distributed (iid.) events and $y_i \in \{0, 1\}$. The probability of a certain output value is given by the logistic (also called logit or Sigmoid) function

$$p(y_i = 1|x_i, \beta) = \frac{\exp(\beta^T x_i)}{1 + \exp(\beta^T x_i)}, \tag{32}$$

where x_i is a row in the design matrix (under the assumption that $x_{i,0} = 1$, which yields the bias term β_0). The next step is to use the maximum likelihood estimation principle (MLE)

$$P(\mathcal{D}|\beta) = \prod_{i=1}^{n} \left[P(y_i = 1|x_i\beta)\right]^{y_i} \left[1 - P(y_i = 1|x_i\beta)\right]^{1-y_i}. \tag{33}$$

A typical choice of cost function in logistic regression is the cross entropy, which can easily be found given MLE in equation 33. The negative log-likelihood of P, is the cross entropy,

$$C(\beta) = -\log P(\mathcal{D}|\beta) \tag{34}$$

$$= \sum_{i=1}^{n} y_i \left[\log P(y_i = 1|x_i, \beta)\right] + (1 - y_i) \log\left[1 - P(y_i = 1|x_i, \beta)\right]. \tag{35}$$

The cost function must be minimized, it is therefore convenient to find the gradient of $C(\beta)$ wrt. β. The lineraized expression for this gradient is

$$\frac{\partial C(\beta)}{\partial \beta} = -X^T(y - p), \tag{36}$$

where $X \in \mathbb{R}^{n\times p}$ is the design matrix, $y \in \ltimes$ is the output vector and $p \in \ltimes$ is the probability vector given by equation 32. The second derivative is given by

$$\frac{\partial^2 C(\beta)}{\partial\beta\partial\beta^T} = X^T W X, \tag{37}$$

with $W \in \mathbb{R}^{n\times n}$ being a diagonal matrix where the diagonal elements are

$$W_{i,i} = P(y_i|x_i, \beta)\left[1 - P(y_i|x_i, \beta)\right]. \tag{38}$$

Computationally, the optimal parameter vector β can be found iteratively by

$$\beta^t_{opt} = \beta^{t-1}_{opt} - \frac{\partial C(\beta)}{\partial\beta} \Big/ \frac{\partial^2 C(\beta)}{\partial\beta\partial\beta^T} \tag{39}$$

$$= \beta^{t-1}_{opt} - (X^T W X)^{-1} X^T(p - y). \tag{40}$$

After each iteration, W and p must be updated by using the new β^t_{opt}. $H = (X^T W X)^{-1}$ is the Hessian matrix, which requires a lot of computational power to compute, therefore H is normally replaced by a scalar η which is referred to as the *learning rate*. The iterative algorithm for gradient descent with cross entropy as cost function is given by

$$\beta^t_{opt} = \beta^{t-1}_{opt} - \eta X^T(p - y). \tag{41}$$

In the research presented, a stochastic gradient descent (SGD) with mini-batches will be applied. When the cost-function can be written as a sum over the n data points - which is the case for cross entropy in equation 41, then it is easy to introduce stochasticity to the algorithm. The data set can be split into m mini-batches, each with the size of $M = n/m$ data points. Instead of summing over all the data points, choose a random mini-batch, m times. The stochasticity arises from the fact that it is possible to choose the same mini batch several times during one epoch. The general form of the step size is given by

$$\frac{\partial}{\partial\beta} C(\beta) = \sum_{i\in B_k}^{n} \frac{\partial}{\partial\beta} c_i(\mathbf{x}_i, \beta). \tag{42}$$

In the case of cross entropy, simply rewrite equation 41 to

$$\beta^t_{opt} = \beta^{t-1}_{opt} - \frac{\eta}{m} \sum_{i\in B_k} X_i^T(p_i - y_i), \tag{43}$$

where i indicates a mini batch, which must be chosen randomly, m times.