

# Using Neural Networks and Logistic Regression for Classification and Regression Problems

Eina B. Jørgensen, Anna Lina P. Sjur and Jan-Adrian H. Kallmyr

November 15, 2019

## Abstract

We use a neural network (NN) and logistic regression (LR) to classify the 2005 Taiwan credit card data. There are two categories: non-default and default. A comparison is made between the two methods, and we find that the NN yields the highest accuracy score 0.823 compared with 0.820 from the LR algorithm. Area under the receiver operating characteristic curve (AUC) was also used as a performance measure, the NN producing the highest score of 0.782, compared with 0.765 from logistic regression. The highest accuracy and AUC were found from different hyperparameters respectively. We also produced a confusion matrix, which showed that NN classified more true positives and true negatives correctly than LR. In terms of efficiency, the LR algorithm converges quite rapidly compared to the NN, and is overall more stable, possibly due to an implementation of dynamic learning rate. Furthermore, we use the NN to fit the Franke function (with noise), and compare this to linear regression algorithms used in our earlier paper. Our findings suggest that the neural network achieved the lowest MSE of 0.011, and a R2-score of 0.89.

## 1 Introduction

Artificial Neural Networks (NNs) offer a flexible way of performing data analysis, and can be used for many different problems in machine learning. In particular, we can use NNs for classification and regression problems such as determining if a planet is terrestrial or not, or fitting terrain data. It is therefore of interest to measure the performance and quality of NNs compared with other machine learning algorithms.

In this article, we will study the 2005 Taiwan credit card data set (Yeh and Lien, 2009) with the purpose of classifying risky/non-risky credit card holders. For this, we use two methods: Logistic Regression and Neural Networks. We also use a NN to fit Franke’s function (see our earlier paper Jørgensen et al. (2019)). In

both cases, we compare our NN to the respective methods: Logistic and Linear Regression (Jørgensen et al., 2019).

Starting with Theory and methods in section 2, we describe the data set and the methods used in detail, as well as our chosen activation, cost, and score functions. Moving on to Results in section 3, we present our most important results such as tuning parameters and accuracy scores for each method. In section 4 we discuss the method comparisons in more detail, and consider our results. Furthermore, possible lacks and improvements are suggested, before concluding our paper in section 5.

## 2 Theory and methods

### 2.1 Data sets

In this report, the default of credit card clients data set (Yeh and Lien, 2009) was used to study the performance and compare the Logistic Regression method and Neural Networks for classification. A description of the attributes of the data set can be found in Yeh and Lien (2009). Upon inspection, it is notable that the data set contains a considerable amount of values different from their valid values as described by Yeh and Lien. Mostly, this apply for the categorical variables, where the given value does not correspond to any category. By removing data points with invalid values, as well as entries where the client does not have any history of past payments or bill statements, the data set is reduced from 30000 data points to 3792 points. As this is a considerable reduction of data points, two additional data sets were constructed. In one, all the invalid PAY values were kept. These invalid values were some  $-2$ s, but mostly 0s. In the second additional set, all  $-2$ s,  $-1$ s and 0s were set to 0, since these were assumed to all represent duly payments. Both sets contained 26908 data points, with 78% of the labels equal to zero, i.e. not default payment next month. The last set gave best results on initial runs, and was used for the rest of the project. To handle categorical variables, One hot encoding was used. For a more in-depth inspection of the data set, see the Jupyter notebook `inspect_data.ipynb` in the [GitHub repository](#) of janadr.

In addition to the default of credit card clients data set, data produced with Franke’s function was applied to Neural Networks. For details on Franke’s function, see Jørgensen et al. (2019).

### 2.2 Logistic Regression

Classification problems aim to predict the behaviour of a given object, and produces outcomes in form of discrete variables (i.e. categories). Logistic Regression can be used to solve such problems, commonly by the use of binary outcomes such as true/false, positive/negative, success/failure etc., or in the specific credit card case: *risky/non-risky* Hjort-Jensen (2019b).

As opposed to Linear Regression, the equation one gets as a result of minimizing the cost function by  $\beta$  using Logistic Regression, is non-linear, and can not be solved explicitly. Instead it can be solved using minimization algorithms, for example by *gradient descent methods*.

When predicting the output classes in which an object belongs, the prediction is based on the design matrix  $\mathbf{X} \in \mathbb{R}^{n \times p}$  that contains  $n$  samples that each carry  $p$  features.

A distinction is made between *hard classification* - determines the variable to a category deterministically, and *soft classification* - determines the probability that a given variable belongs in a certain category. The latter is favourable in many cases, and logistic regression is the most used example of this type of classifier.

When using Logistic Regression, the probability that a given data point  $\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{ip}]$  belongs in a category  $y_i$  is given by the Sigmoid-function (or logistic function):

$$p(t) = \frac{1}{1 + e^{-t}} = \frac{e^t}{1 + e^t} \quad (1)$$
$$1 - p(t) = p(-t)$$

Assuming a binary classification problem, i.e.  $y_i$  can be either 0 or 1, and a set of predictors  $\beta$  the Sigmoid function (1) gives the

probabilities with relation:

$$p(y_i = 0|\mathbf{x}_i, \boldsymbol{\beta}) = 1 - p(y_i = 1|\mathbf{x}_i, \boldsymbol{\beta})$$

The total likelihood for all possible outcomes  $\mathcal{D} = \{(y_i, x_i)\}$  is used in the Maximum Likelihood Estimation (MLE), aiming at maximizing the log/likelihood function (2). The likelihood function can be expressed with  $\mathcal{D}$ :

$$P(\mathcal{D}|\boldsymbol{\beta}) = \prod_{i=1}^n [p(y_i = 1|\mathbf{x}_i, \boldsymbol{\beta})]^{y_i} [1 - p(y_i = 1|\mathbf{x}_i, \boldsymbol{\beta})]^{1-y_i}$$

And the log/likelihood function is then:

$$P_{\log}(\boldsymbol{\beta}) = \sum_{i=1}^n \left( y_i \log [p(y_i = 1|\mathbf{x}_i, \boldsymbol{\beta})] + (1 - y_i) \log [1 - p(y_i = 1|\mathbf{x}_i, \boldsymbol{\beta})] \right) \quad (2)$$

The cost/error-function  $\mathcal{C}$  (also called cross-entropy in statistics) is the negative of the log/likelihood. Maximizing  $P_{\log}$  is thus the same as minimizing the cost function. The cost function is:

$$\mathcal{C}(\boldsymbol{\beta}) = -P_{\log}(\boldsymbol{\beta}) = -\sum_{i=1}^n \left( y_i \log [p(y_i = 1|\mathbf{x}_i, \boldsymbol{\beta})] + (1 - y_i) \log [1 - p(y_i = 1|\mathbf{x}_i, \boldsymbol{\beta})] \right) \quad (3)$$

Finding the parameters  $\boldsymbol{\beta}$  that minimize the cost function is then done through differentiation. Defining the vector  $\mathbf{y}$  containing  $n$  elements  $y_i$ , the  $n \times p$  matrix  $\mathbf{X}$  containing the  $\mathbf{x}_i$  elements as rows, and the vector  $\mathbf{p}$  that is the fitted probabilities  $p(y_i|\mathbf{x}_i, \boldsymbol{\beta})$ , the first derivative of  $\mathcal{C}$  is

$$\nabla_{\boldsymbol{\beta}} \mathcal{C} = \frac{\partial \mathcal{C}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -\mathbf{X}^T(\mathbf{y} - \mathbf{p}) \quad (4)$$

This gives rise to a set of linear equations, where the aim is to solve the system for  $\boldsymbol{\beta}$ .

With  $\mathbf{x} = [1, x_1, x_2, \dots, x_p]$  and  $p$  predictors  $\boldsymbol{\beta} = [\beta_0, \beta_1, \beta_2, \dots, \beta_p]$  the relation between likelihoods of outcome is:

$$\log \frac{p(\boldsymbol{\beta}\mathbf{x})}{1 - p(\boldsymbol{\beta}\mathbf{x})} = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p \quad (5)$$

and  $p(\boldsymbol{\beta}\mathbf{x})$  is defined by:

$$p(\boldsymbol{\beta}\mathbf{x}) = \frac{e^{\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p}}{1 + e^{\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p}} \quad (6)$$

## 2.3 Gradient Descent Methods

### The General Idea

With the gradient of  $\mathcal{C}$  defined as in (4), we used this to find the minimum of the cost function. The basic idea is that by moving in the direction of the negative gradient of a function, we can move towards the value (in this case the  $\boldsymbol{\beta}$ ) that minimizes the function (in this case  $\mathcal{C}(\boldsymbol{\beta})$ ). Hjort-Jensen (2019a)

This is done by repeating the algorithm

$$\boldsymbol{\beta}_{j+1} = \boldsymbol{\beta}_j - \gamma \nabla_{\boldsymbol{\beta}} \mathcal{C}(\boldsymbol{\beta}) \quad j = 0, 1, 2, \dots \quad (7)$$

When a minimum is approached,  $\nabla_{\boldsymbol{\beta}} \mathcal{C}(\boldsymbol{\beta}) \rightarrow 0$ , and thus we can set a limit when  $\boldsymbol{\beta}_{k+1} \approx \boldsymbol{\beta}_k$  given a certain tolerance, and the  $\boldsymbol{\beta}$  which minimizes the cost function is found.  $\gamma$  is in this case called the *learning rate*, and is a parameter that must be tuned to each specific case in order to optimize the regression.

### Stochastic Gradient Descent

In this project we use a stochastic version of gradient descent, which is an improvement upon the regular gradient descent. This is done by expressing the cost function (and thus also its gradient) as a sum

$$\nabla_{\boldsymbol{\beta}} \mathcal{C}(\boldsymbol{\beta}) = \sum_{i=1}^n \nabla_{\boldsymbol{\beta}} C_i(\mathbf{x}_i, \boldsymbol{\beta}), \quad (8)$$

and by only calculating the gradient  $\nabla_{\boldsymbol{\beta}} C_i$  of a subset of the data at the time. These subsets,

called *minibatches* are of size  $m$ , and the total amount is  $\frac{n}{m}$  where  $n$  is the amount of data points. The minibatches are denoted  $\mathbf{B}_k$ , with  $k = 1, 2, \dots, \frac{n}{m}$ .

Instead of a sum over all the data points  $i \in [1, n]$  we now sum over all the data points in the given minibatch  $i \in \mathbf{B}_k$  where  $k$  is picked randomly with uniform probability from  $[1, \frac{n}{m}]$ .

The stochastic and final version of (7) is therefore given by the algorithm

$$\beta_{j+1} = \beta_j - \gamma_j \sum_{i \in \mathbf{B}_k} \nabla_{\beta} C_i(\mathbf{x}_i, \beta) \quad (9)$$

An iteration over the total number of minibatches is commonly referred to as an *epoch*.

By using the stochastic gradient descent method (9) to minimize the cost function (3) we can find the  $\beta$  values that give the most accurate classification.

### Dynamic learning rate

Convergence rate might be slow, and a possible solution to this is changing learning rate as the gradient descent gets closer to a local minima. Instead of a static learning rate, we have a learning schedule. As we expect convergence as iterations increase, it is natural that the learning schedule be a function of epochs and minibatches. Specifically, the learning rate should decrease as it approaches a minima. A possible function for the learning rate is then

$$\gamma_j = \frac{\gamma_0}{im + j}, \quad (10)$$

where  $\gamma_0$  is the initial learning rate,  $i$  the current epoch,  $j$  the current minibatch, and  $m$  the minibatch size.

## 2.4 Neural Networks

In this section, the equations used are based off the book by Nielsen (2015), unless otherwise specified.

### The structure of a network

Neural Networks, as the name suggests, are inspired by our understanding of how networks of neurons function in the brain. As can be seen in the example network in Figure 1, neurons are structured in layers. We always have a input and an output layer, in addition to a varying number of hidden layers. The input layer has as many neurons as there are input variables, while the output layer has one neuron for each output. How many neurons you have in the output layer depends on the specific problem. The number of neurons in each hidden layer, on the other hand, is not directly related to inputs or outputs, and must be decided in some other way.

As the diagram in Figure 1 suggests, the neurons in each layer are not connected with each other, but takes in inputs from the previous layer and passes on an output to the neurons in the next layer, as illustrated with arrows. This way, the inputs are fed through the network and processed, resulting in an output.

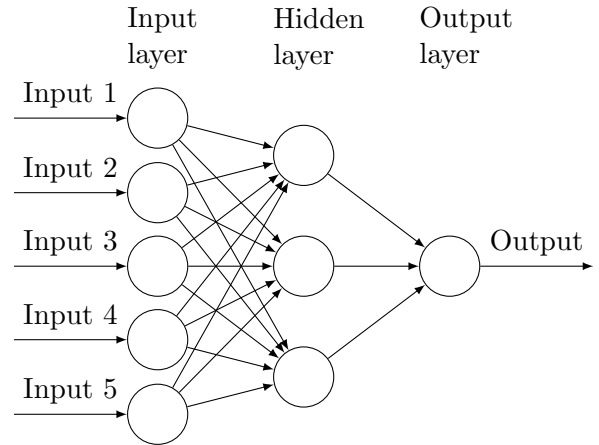


Figure 1: Schematic diagram of a neural network with five input neurons in the input layer, one hidden layer with three neurons and a single output neuron in the output layer.

## Forward feeding

Each neuron has one or multiple inputs, as illustrated with arrows in Figure 1. Each of these inputs has a weight associated with it. To clarify the notation used, let's take a look at the  $j$ th neuron in the  $l$ th layer. The weight associated with the input coming from the  $k$ th neuron in the previous layer is denoted as  $w_{jk}^l$ . In addition, each neuron has a bias associated with it, for the neuron in question denoted as  $b_j^l$ . Summing the weighted inputs and the bias, and feeding this to a function  $f$ , gives the activation  $a_j^l$ :

$$a_j^l = f \left( \left( \sum_k w_{jk}^l a_k^{l-1} \right) + b_j^l \right)$$

This activation is then fed forward as input to all the neuron in the next layer.

In matrix notation, the activation for the whole layer  $l$  can be written as

$$\mathbf{a}^l = f \left( \mathbf{w}^l \mathbf{a}^{l-1} + \mathbf{b}^l \right) \quad (11)$$

Here,  $\mathbf{a}^l$  and  $\mathbf{b}^l$  are vertical vectors containing the activations and biases of the  $l$ th layer, while  $\mathbf{w}^l$  is a matrix with elements  $w_{jk}^l$ , i.e. the  $j$ th column contains the weights of the inputs reaching the  $j$ th neuron.

Let's look at the activation function in Eq. (11) denoted with a  $f$ . In the case of classification, the sigmoid function stated in Eq. (1) is often used in introductory texts. As we will see in the backpropagation algorithm, the sigmoid is a good choice for activation function, since a small change in the output can be propagated backwards, resulting in small changes in the weights and biases through the network.

Another activation function is the so-called rectified linear unit (ReLU) function, given as  $f(z) = \max(0, z)$ . This gives an activation function which only fires when  $z$  is positive. In the case of regression, a variant of ReLU,

called leaky ReLU, was used in this project. This function is given as

$$f(z) = \begin{cases} z & z \geq 0 \\ 0.01z & z < 0 \end{cases} \quad (12)$$

This form of ReLU has a small positive gradient for negative values, which allows for gradient based learning even when the  $z$ -value is negative (Wang et al., 2018).

With a basis in Eq. (11), the algorithm for forward feeding is given in Algorithm 1. Here  $L$  is the total number of layers.

```

Set  $\mathbf{a}^1 = \text{input}$ ;
foreach  $l=2:L$  do
    | Compute  $\mathbf{z} = \mathbf{w}^l \mathbf{a}^{l-1} + \mathbf{b}^l$ ;
    | Compute  $\mathbf{a}^l = f(\mathbf{z})$ ;
Set output to  $\mathbf{a}^L$ ;

```

**Algorithm 1:** The forward feeding algorithm.

Note that the output will have values between 0 and 1, when the sigmoid function is used to compute the activations of all the layers. In a classification problem, this corresponds to the likelihood of an outcome. For example, in a classification problem with five classes, the network would have five output neurons, each representing a class. The final classification of an input would then be the class with the highest probability. In our classification problem, we have a single neuron that gives the probability of default payment.

In the regression case, we see that when using the leaky ReLU activation function from Eq. (12), the output can be any positive value (strictly also negative values, due to the leakage, but this would call for large negative  $z$ -values).

## Backpropagation

When training the network, the goal is to find the weights and biases that minimize the cost function  $C$ . In this project, the cross-entropy cost function was used for classification. This is given as follows for a Neural Network:

$$C = - \sum_{i=1}^n [y_i \ln a_i^L + (1 - y_i) \ln(1 - a_i^L)] \quad (13)$$

Here, we are summing over  $n$  points of training data, where  $a_i^L$  is the output with the corresponding correct value  $y_i$ .

In the regression case, the quadratic cost function, given as

$$C = \frac{1}{2} \sum_{i=1}^n \|a_i^L - y_i\|^2 \quad (14)$$

is used.

To find the weights and biases that minimize Eq.(13) and (14), one can use Stochastic Gradient Decent, as described previously. But in order to use SGD, the derivatives of  $C$  with respect to all the weights and biases must be computed, and it is here that backpropagation comes in. It can be shown, by repeating the chain rule, that the derivatives are given as in Eq. (15).

$$\begin{aligned} \delta^L &= \nabla_a C \odot f'(z^L) \\ \delta^l &= ((\mathbf{w}^{l+1})^T \delta^{l+1}) \odot f'(z^l) \\ \frac{\partial C}{\partial b_j^l} &= \delta_j^l \\ \frac{\partial C}{\partial w_{jk}^l} &= a_k^{l-1} \delta_j^l \end{aligned} \quad (15)$$

Taking a look at the expression for  $\delta^L$ , one can show that this reduces to  $\delta^L = \mathbf{a} - \mathbf{y}$ , when the cross-entropy cost function is combined with the sigmoid activation function, and similarly for the quadratic cost and ReLU when we have positive output values. This means that

we get rid of the  $f'(z^L)$  factor, which can slow down learning significantly if the derivative is close to 0.

Equation (15) are the basis for the backpropagating algorithm, described in Algorithm 2.

```

Compute  $\{\mathbf{a}^l\}_{l=1}^L$  with feed forward;
Compute  $\delta^L$ ;
Set  $\frac{\partial C}{\partial \mathbf{b}^L} = \delta^L$ ;
Compute  $\frac{\partial C}{\partial \mathbf{w}^L} = \delta^L (\mathbf{a}^{L-1})^T$ ;
foreach  $l=L-1:2$  do
    Compute  $\delta^l$ ;
    Set  $\frac{\partial C}{\partial \mathbf{b}^l} = \delta^l$ ;
    Compute  $\frac{\partial C}{\partial \mathbf{w}^l} = \delta^l (\mathbf{a}^{l-1})^T$ ;

```

**Algorithm 2:** The backpropagation algorithm.

## Overfitting and Regularization

A problem that can arise when fitting a regression model, as discussed in Jørgensen et al. (2019), is overfitting to data. This is a major problem also in Neural Networks, where the network can fit too heavily to outliers in the training set, especially when we have a large number of neurons (Nielsen, 2015). Apart from increasing the training data set and decreasing the number of neurons, a third approach is to use a regularization technique. Here, we will use the weight decay or L2 regularization technique. The idea is to add a term to the cost function. For both cross-entropy and quadratic cost, the new cost function can be written as

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

where  $C_0$  is the original cost function and  $\lambda > 0$  is the regularization parameter. The algorithm for updating the weights, when using stochas-



tic gradient descent, then becomes

$$w \rightarrow \left(1 - \frac{\gamma\lambda}{n}\right) w - \frac{\gamma}{m} \sum_{i \in \mathbf{B}_k} \frac{\partial C_i}{\partial w} \quad (16)$$

Here,  $m$  is the mini batch size. The algorithm for updating the biases is unchanged:

$$b \rightarrow b - \frac{\gamma}{m} \sum_{i \in \mathbf{B}_k} \frac{\partial C_i}{\partial b} \quad (17)$$

Note the factor  $m^{-1}$  in Eq. (16) and (17) when comparing to Eq. (9). This factor can be understood as taking the mean of all the gradients, instead of the sum. In this project, the sum was used for logistic regression, while the mean was used in the neural network. In practise, this corresponds to a scaling of the learning rate.

## 2.5 Quality of Measurements

To measure how good the different methods were at classifying the credit card data, the data was split into a training set and a test set. After training the model on the test set, it was applied on the training set, and a performance score was calculated. Both accuracy, given in Eq. (18) and the area under the receiver operating characteristic curve, denoted AUC, was applied.

$$\text{Accuracy} = \frac{\text{Correct classifications}}{\text{Total \# of classifications}} \quad (18)$$

For calculations of AUC, `sklearn.metrics.roc_auc_score` was used. AUC can be interpreted as the probability that a random true positive sample is ranked higher than a random true negative. This score is not dependent on the chosen classification threshold, i.e. the value that must be exceed to classify an input as one instead of zero. For reference, the classification threshold was set to 0.5 in this project.

For data generated with Franke’s function, the same test-train-split was made, and the mean squared error was calculated, as in Jørgensen et al. (2019).

## 2.6 Implementation

All our algorithms were implemented in python3.6 using the numpy, matplotlib, pandas, seaborn, and scikit-learn packages. We wrote our own class ”Regression” for linear and logistic regression, as well as class ”Network” for Neural Network. In the way our Neural Network was implemented, we could use the same code for classification and regression by simply replacing activation functions, while at the same time ensuring that the appropriate cost function was used. Scikit-learn was used to verify our implementation by comparing accuracy and AUC scores. Code along with data and figures can be found on our github repository under ”Project 2”.<sup>1</sup>

## 3 Results

### 3.1 Tuning Learning Rate and Mini-batch Size

#### Logistic Regression

Figure 2 shows a heatmap of the AUC of the Logistic Regression model used on the classification problem for different minibatch sizes and initial learning rates, where the variables are distributed logarithmically.

Figure 3 is similar to Figure 2, except that the heatmap shows the accuracy, and not the AUC, for the Logistic Regression.

In both Figure 3 and 2, the blanc field to the top right represents parameters that gave overflow.

We see that both AUC and accuracy are highest in a band spanning from relatively high

<sup>1</sup><https://github.com/janadr/FYS-STK4155.git>

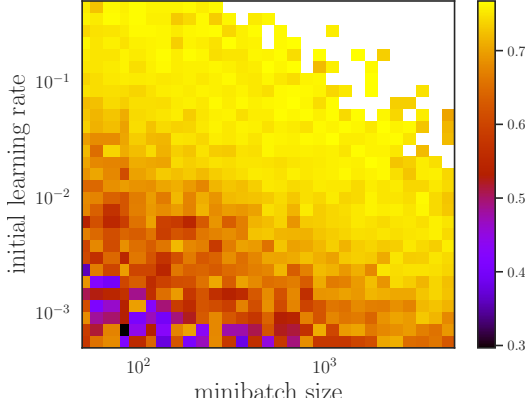


Figure 2: Heatmap showing the AUC of the Logistic Regression applied on the classification problem for minibatch sizes  $m \in [5, 5000]$  and learning rates  $\gamma \in [0.5 \cdot 10^{-3}, 0.5]$  in logarithmic scale. AUC increases from blue to yellow.

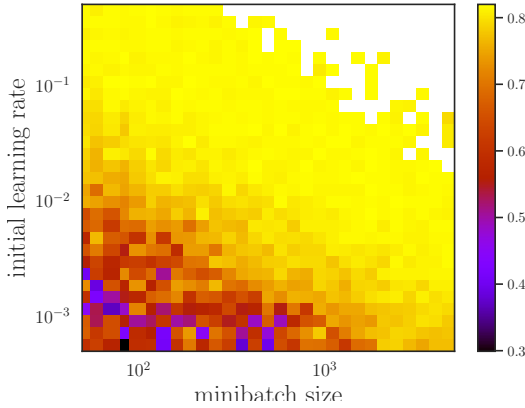


Figure 3: Heatmap showing the accuracy of the Logistic Regression applied on the classification problem for minibatch sizes  $m \in [5, 5000]$  and learning rates  $\gamma \in [0.5 \cdot 10^{-3}, 0.5]$  in logarithmic scale. Accuracy increases from blue to yellow.

initial learning rates and relatively small batch sizes to smaller initial learning rates and larger batch sizes.

## Neural network

Figure 5 and 4 shows heatmaps of the accuracy and AUC of the Neural Network. As before, blanc fields represent values that gave overflow.

For both accuracy and AUC, large minibatch sizes combined with small learning rates gave lower scores. In the heatmap of AUC, the combination of relatively high learning rates and low minibatch sizes gave lower AUC values.

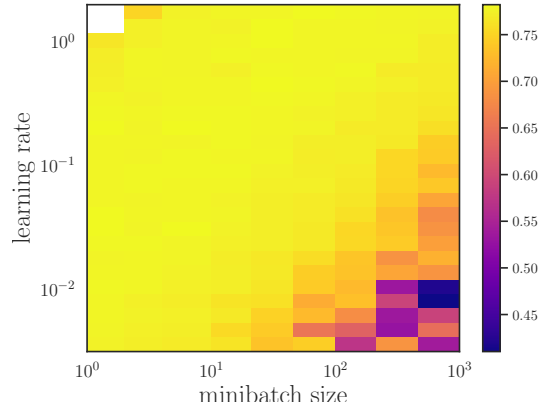


Figure 4: Heatmap showing the AUC of the Neural Network applied on the classification problem for minibatch sizes  $m \in [1, 1000]$  and learning rates  $\gamma \in [10^{-2.5}, 10^{0.3}]$  in logarithmic scale. The hidden layers were configured as  $[20, 20]$ . AUC increases from blue to yellow.

## Optimal Parameters and Associated Results

Table 1 spells out the values that gave the highest AUC for Logistic Regression and Neural Network in Figure 2 and 4.



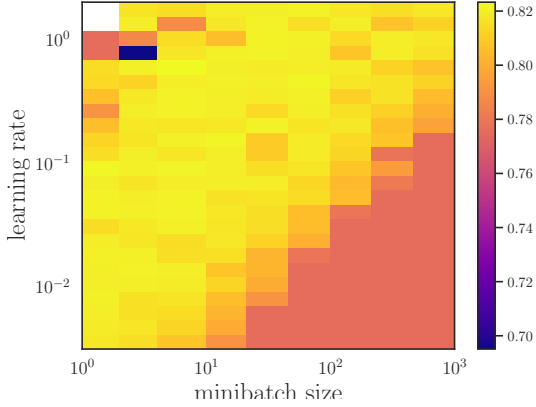


Figure 5: Heatmap showing the accuracy of the Neural Network applied on the classification problem for minibatch sizes  $m \in [1, 1000]$  and learning rates  $\gamma \in [10^{-2.5}, 10^{0.3}]$  in logarithmic scale. The hidden layers were configured as  $[20, 20]$ . Accuracy increases from blue to yellow.

Table 1: Table of parameters that gave the highest **AUC** in the classification problem. Learning rate  $\gamma$ , minibatch size  $m$ , accuracy  $acc$  and **AUC** is shown for Logistic Regression and Neural Network.

Method	$\gamma$	$m$	$acc$	<b>AUC</b>
Logistic regression	0.15	870	0.817	0.765
Neural network	0.035	1	0.821	0.782

Similarly, Table 2 shows the parameters that gave the highest accuracy for Logistic Regression and Neural Network in Figure 3 and 5.

Table 2: Table of parameters that gave the highest **accuracy** in the classification problem. Learning rate  $\gamma$ , minibatch size  $m$ , accuracy  $acc$  and **AUC** is shown for Logistic Regression and Neural Network.

Method	$\gamma$	$m$	$acc$	<b>AUC</b>
Logistic regression	0.0069	1645	0.820	0.758
Neural network	0.52	4	0.823	0.777

In Table 3, we can see the number of true and false positives and negatives, when the parameters stated in Table 1 are used, i.e. the parameters that gave the highest **AUC**.

Table 3: Fraction of true and false negatives and positives for Logistic regression and Neural Network applied on the classification problem, when optimal parameters are used.

Method	Positive		Negative	
	True	False	True	False
Logistic regression	432	210	3964	776
Neural network	452	200	3974	756

### 3.2 Neural Networks for Regression on Franke’s Function

Looking at Figure 6, we show a heatmap for MSE as a function of minibatch size and learning rate. We observe a zone of low MSE for  $m=1$  and  $\gamma \approx 10^{-2}$ . Additionally, for approximate ranges  $m \in [10^2, 10^3]$  and  $\gamma \in$

$[10^{-4}, 10^{-2}]$ , in the lower right corner, we observe a zone where the MSE is an order of magnitude higher or more. The lowest MSE obtained was 0.011, while the highest R2-score was 0.89.

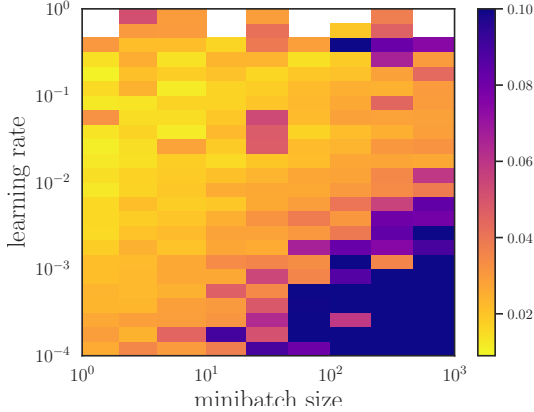


Figure 6: Heatmap showing the MSE of the Neural Network applied on Franke’s function for minibatch sizes  $m \in [1, 1000]$  and learning rates  $\gamma \in [10^{-4}, 1]$  in logarithmic scale. The hidden layers were configured as  $[100, 20]$ . MSE decreases from blue to yellow. Minimum MSE and maximum R2-score was achieved with  $m = 1$  and  $\gamma = 0.146780$ .

## 4 Discussion

### 4.1 Tuning to optimal parameters

As previously stated, two measurements were used to determine the quality of the different methods for classification. Accuracy as described in (18), and the AUC-score. The AUC-score is a good measurement, not only because it is independent of the classifier threshold, but also, as we can see from figure 2 and 4, it gives a more nuanced view of the belts in which the optimal regularisation parameters are located. For both Logistic Regression and Neural Network, the areas of the heatmaps that give the

best accuracy are more or less the same that give the best AUC-score, however for the two classifiers the behaviour is rather different.

For the Logistic Regression, we see an increase in the quality of the classifier with increased initial learning rate  $\gamma$  and increased batch size. If the learning rate becomes too large however, overflow is encountered, and we get invalid results. The fluctuations of the accuracy and AUC-score within the yellow belts in figure 3 and 2 are relatively small, and as long as the parameters are chosen from this area, we can expect results close to the optimal. The true optimal parameters for accuracy and AUC-score are presented in table 2 and 1, and show that the best results we could obtain with the Logistic Regression classifier, was an accuracy of 0.820 and an AUC score of 0.765. This accuracy corresponds nicely to the accuracy calculated with the logistic regression provided by scikit-learn.

For the Neural Network classifier, we see what appears to be an opposite behaviour of the Logistic Regression. The classifier produces the best results for smaller batch sizes, and a smaller learning rate. Also in this case, the impression is that, as long as we choose parameters from within the yellow belt in figure 5 and 4, the difference in the quality is relatively small, and we are close to the optimal behaviour of the classifier. For the neural network, with the parameters as described, again in table 1 and 2 we are able to get an accuracy of 0.823 and an AUC-score of 0.782, which is an accuracy that just like for Logistic Regression, corresponds well with the results produced by scikit-learn’s neural network, and thus strengthens our results.

One explanation of the different behaviour of the two classifiers are the difference in how the stochastic gradient descent is implemented, as described in the Theory section. This result in different scaling of the learning rate, as the minibatch size changes.

#### 4.1.1 Comparing the classifiers

For the accuracy the two classifiers obtain very similar results, with Neural Network only slightly better than Logistic Regression. When comparing the AUC score, however, the Neural Network classifier outperforms the Logistic Regression by 0.017 points. This is another reason why the score is an important measurement to include. Taking this in to consideration it becomes clear that Neural Network is the better option for this case. When looking at the confusion matrix in table 3, we also see that the network is better than the Logistic Regression at identifying both the default and non-default payments, and in this case, where the negative results is a minority, that becomes important. That Neural Network is the better option, is also the conclusion that is reached in Yeh and Lien (2009). Here their error rate of 0.17 also corresponds well with our calculated accuracy of 0.82. Regarding the AUC-score, this cannot be directly compared as they have used Area Ratio rather than the AUC-score. What is clear however, is that it is when looking at the optimal curve and area scores that one truly sees that Neural Networks are better, as this was the case both for us, and Yeh and Lien (2009).

However, looking at the true positives and false negatives in 3, our Neural Network is quite bad at identifying positive instances. As we see that there are more false negatives than true positives, most of the positives, i.e. default payments, are not correctly classified. This could be due to the small fraction of default payments in the data set. One possible way of dealing with this could be to resample the data set, so that the network would train on an equal amount of positive and negative instances.

#### 4.2 Comparing methods for regression

Shown in Table 4 is a comparison between the MSE obtained from our neural network and other values obtained from linear regression algorithms. We see that the neural network performs better compared with any other method. A suggestion for why this is the case is that the Franke function is not a polynomial. The neural network, however, constructs a general function, as discussed by Nielsen (2015), and so is not bound by assumption of the design matrix on our part. With that said, the choice of activation function can have a large impact, and we found that the leaky ReLU was the only one that converged with reasonable tuning.

Table 4: From descending order, the first three values were collected from Jørgensen et al. (2019) and represent the lowest MSE achieved from the linear regression algorithms with complexity 6, 9, and 11, respectively. The last value is the MSE obtained from our neural network.

Method	MSE
Ordinary Least Square	0.017
Ridge regression	0.015
Lasso regression	0.021
Neural Network	0.011

## 5 Conclusion

We studied classification and regression using a neural network and logistic regression algorithm. For classification we used the 2005 Taiwan credit card data, and for regression we fitted the Franke function. In both cases, the neural network was compared to other methods: logistic and linear regression, respectively.

To compare methods we used the accuracy and ROC AUC score functions for classification, and R2 and MSE for regression. For classification, we found that the neural network gave the highest accuracy of 0.823, slightly higher than logistic regression, which had a accuracy of 0.820. As for the ROC AUC score, the neural network also performed better with a score of 0.782 compared with 0.765 from the LR algorithm. Comparing confusion matrices, it was difficult to discern significant differences between the algorithms. This might have been because of the chosen thresholds for classification in each method. Representing a hyperparameter on its own, tuning of the threshold could achieve higher accuracy. The AUC score takes this into account however, and so we deemed it the most reliable performance measure. In general, the LR algorithm seemed to converge faster compared to the neural network, although no formal analysis was made.

As for regression, we found that the neural network achieved a lower MSE of 0.011 compared with values obtained from linear, ridge, and lasso regression (Jørgensen et al., 2019). R2-scores could not be compared between methods, but we found that the same hyperparameters yielding the lowest MSE also yielded highest R2-score of 0.89, and so this indicated that MSE is a good measure. The neural network therefore produced the best model.

For classification and regression, it seems neural networks perform just as good, if not better compared with logistic and linear regression methods.

## References

- Morten Hjort-Jensen. *Data Analysis and Machine Learning Lectures: Optimization and Gradient Methods*. 2019a.
- Morten Hjort-Jensen. *Data Analysis and Machine Learning: Logistic Regression*. 2019b.
- Eina B. Jørgensen, Anna Lina P. Sjur, and Jan-Adrian H. Kallmyr. *Using linear regression for fitting terrain data*. 2019.
- Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.
- Shui-Hua Wang, Preetha Phillips, Yuxiu Sui, Bin Liu, Ming Yang, and Hong Cheng. Classification of alzheimer’s disease based on eight-layer convolutional neural network with leaky rectified linear unit and max pooling. *Journal of medical systems*, 42(5): 85, 2018.
- I-Cheng Yeh and Che-hui Lien. 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, 2009.