Classification and Regression

FYS-STK4155: Project 2

Kim-Roger Grønmo
http://github.com/kimgronmo/

November 20, 2021

Abstract

We investigate the suitability of using logistic regression and a feed forward neural network (FFNN) to study both regression- and classification problems. The Franke Function is used to generate data which we study using stochastic gradient descent for OLS and Ridge regression. This function also enable us to study regression using a FFNN. We also study classification problems using both a FFNN and logistic regression method for the Wisconsin Breast Cancer (WBC) dataset and the MNIST dataset of hand-written numbers.

Our results were obtained by tweaking the learning rate and regularization hyperparameters. We find that the Franke Function is best fitted using a neural network with 20 neurons in one layer and 10000 epochs. This gives us an R2 score of 0.97 for RELU activation function. For the image data set we find that the neural network performs slightly better than our own logistic regression code for the WBC with accuracy numbers of 0.77 versus 0.75. For the MNIST dataset the logistic regression method gave an accuracy of 0.96 versus 0.95 for the neural network.

1 Introduction

Many companies today incorporate Neural Networks in their techology. Examples include Facebook, Google and Apple. Whether you are watching a youtube video, listening to music on spotify or asking Apple's Siri a question there is often a neural network working in the background to give you a satisfying experience.

Classification problems are different from the regression problems we studied in project 1. The predictions we make are divided into discrete categories. This means that the methods we use will give a probability estimate as output. This will then enable us to predict a discrete result by assigning the output with the highest probability to different categories. In order to do this we will have to use a different toolset than standard regression. We will consider both logistic regression and Feed Forward Neural Networks to provide us with different probability estimates.

Neural Networks are models that can predict the outcome of new input data, by having learned from previous input/output pairs. The performance of these networks have in many cases far superceded human performance, and can to many seem like magic with no apparent connection between input and prediction. Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing.

The Neural Network is a connection of virtual neurons. These neurons consider the output from other neurons and makes their own calculations, eventually being able to provide a probability estimate. The weights and biases the individual neurons uses to consider input are tuned using a learning algorithm known as stochastic gradient descent.

In this project we will be using both logistic regression and a Neural Network to teach the computer to recognize cancer diagnosis using the Wisconsin Breast Cancer dataset. We will also use the MNIST dataset to teach the computer to recognize hand written digits.

We begin this report by looking at some theory behind the models and methods we will use, including a brief overview of a Feed Forward Neural Network. We then present some of our findings and discuss the results. The report is then concluded with a summary of our findings.

An interested reader can download both this report and supporting code from the following github address:

http://github.com/kimgronmo/FYS-STK4155-PROJECT2

2 Theory and Methods

The general references for this section is reference [1], [2] and [3]. For discussions about Ordinary Least Squares and Ridge regression we refer to our previous report [4]. We also refer to this report for information about the Franke Function.

When we investigated the use of linear regression methods we were interested in learning the coefficients of fitting for instance a polynomial to predict the response of a continuous variable. The fit to the continuous variable y_i is based on some independent variables $\hat{x}_i$. We were then able to get analytical expressions for Ordinary Least Squares and Ridge regression for different quantities such as the parameters $\hat{\beta}$ and the mean squared error. When we now consider classification problems we are interested in outcomes taking discrete variables or resulting categories.

By minimization the cost function we get a non-linear equation in the parameters $\hat{\beta}$. In order to optimize we will use minimization algorithms, more spesifically stochastic gradient descent. This method is used both in the case of logistic regression and in the Neural Network.

Lets consider the case where the dependent variables or responses y_i are discrete and only take values from $k = 0, \ldots, K-1$ (i.e. K classes). We want to predict the the output classes from our design matrix $\hat{X} \in \mathbb{R}^{n \times p}$ made of n samples, each of which carries p features. Our function takes values on the entire real axis. This is a problem when the labels y_i are discrete variables. We solve this by using the logistic function to output the probability of a given category.

2.1 The logistic function

In logistic regression, the probability that a data point x_i belongs to a category $y_i = \{0, 1\}$ is given by the Sigmoid function which represents the likelihood for a given event,

$$p(t) = \frac{1}{1 + \exp{-t}} = \frac{\exp t}{1 + \exp t}.$$

In order to maximize the likelihood we want to minimize the cost function given by

$$\mathcal{C}(\hat{\beta}) = -\sum_{i=1}^{n} \left(y_i(\beta_0 + \beta_1 x_i) - \log\left(1 + \exp\left(\beta_0 + \beta_1 x_i\right)\right)\right).$$

This function is also known as cross entropy. By computing the partial derivatives of this function we get the gradient given by

$$\nabla_\beta C(\beta) = \frac{2}{n} X^T (X\beta - \mathbf{y}),$$

Since $C(\beta)$ is a convex function we can find a global minimum for it. We can minimize $C(\beta)$ using the gradient descent method with a constant learning rate γ using the equation

$$\beta_{k+1} = \beta_k - \gamma \nabla_\beta C(\beta_k), \ k = 0, 1, \cdots$$

For Ridge regression the gradient is given by

$$\nabla_\beta C_{\text{ridge}}(\beta) = 2(X^T(X\beta - \mathbf{y}) + \lambda\beta).$$

Using gradient descent methods has some limitations. They can converge to local minima, thus giving poor performance, and are very sensitive to initial conditions and learning rates. Gradients are also computationally expensive to calculate for large datasets.

2.2 Stochastic Gradient Descent

In order to avoid some of the limitations using gradient descent methods we can instead use a Stochastic Gradient Descent method. The cost function can be written as a sum over n data points $\{\mathbf{x}_i\}_{i=1}^n$,

$$C(\beta) = \sum_{i=1}^{n} c_i(\mathbf{x}_i, \beta).$$

We can then compute the gradient as a sum over i-gradients

$$\nabla_\beta C(\beta) = \sum_{i}^{n} \nabla_\beta c_i(\mathbf{x}_i, \beta).$$

If we then take the gradient on a subset of the data called minibatches, we can get some randomness or stochasticity. If we have n data points and the size of each minibatch is M, there will be n/M minibatches. These minibatches are denoted by B_k where $k = 1, \cdots, n/M$.

A gradient step is then given by

$$\beta_{j+1} = \beta_j - \gamma_j \sum_{i \in B_k}^{n} \nabla_\beta c_i(\mathbf{x}_i, \beta)$$

where k is picked at random with equal probability from $[1, n/M]$. Iterating over the number of minibathces (n/M) is referred to as an epoch.

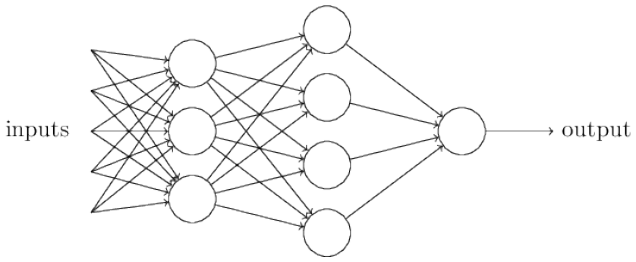

Figure 1: Image representing a neural network with two hidden layers and one output layer. Reference [3].

2.3 Feed Forward Neural Networks

A neural network is a computational model that consists of layers of connected neurons, or nodes. In a feed-forward neural network (FFNN) the information moves in only one direction, forward through the layers. The neurons are represented by circles, while the arrows display the connections between the nodes, including the direction of information flow. Each arrow corresponds to a weight variable.

The output y of a node is given by an activation function f where

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b_i\right) = f(z),$$

Where b_i is the bias. The value of z_i^1 is the argument to the activation function f_i of each node i, The variable M stands for all possible inputs to a given node i in the first layer. The output y_i^1 of all neurons in layer 1 is defined by

$$y_i^1 = f(z_i^1) = f\left(\sum_{j=1}^{M} w_{ij}^1 x_j + b_i^1\right)$$

The output of neuron i in layer 2 is thus

$$y_i^2 = f^2\left(\sum_{j=1}^{N} w_{ij}^2 y_j^1 + b_i^2\right)$$

This can be generalized to a model with l hidden layers

$$y_i^{l+1} = f^{l+1}\left[\sum_{j=1}^{N_l} w_{ij}^3 f^l\left(\sum_{k=1}^{N_{l-1}} w_{jk}^{l-1}\left(\dots f^1\left(\sum_{n=1}^{N_0} w_{mn}^1 x_n + b_m^1\right)\dots\right) + b_k^2\right) + b_1^3\right]$$

The network is trained by adjusting the biases and weights using back propagation.

2.4 Activation functions

A common activation function that we use is the Sigmoid function:

$$f(x) = \frac{1}{1+e^{-x}},$$

The RELU function is given by:

$$f(x) = x^+ = max(0, x)$$

The LeakyRELU function is given by:

$$f(x) = \begin{cases} x & if \ \ x>0 \\ 0.01x & otherwise \end{cases}$$

2.5 Back propagation

The back propagation adjustments are made using the following algorithm:

We set up the input data $\hat{x}$ and the activations $\hat{z}_1$ of the input layer. Then compute the activation function and the outputs $\hat{a}^1$.

We perform now the feed forward til we reach the output layer and compute all $\hat{z}_l$ of the input layer and compute the activation function and the outputs $\hat{a}^l$ for $l = 2, 3, \ldots, L$.

The ouput error $\hat{\delta}^L$ is computed by

$$\delta_j^L = f'(z_j^L)\frac{\partial \mathcal{C}}{\partial(a_j^L)}.$$

The back propagation error is computed for each $l = L-1, L-2, \ldots, 2$ as

$$\delta_j^l = \sum_k \delta_k^{l+1} w_{kj}^{l+1} f'(z_j^l).$$

We update the weights and the biases using gradient descent for each $l = L-1, L-2, \ldots, 2$ and update the weights and biases according to the following equations

$$w_{jk}^l \leftarrow= w_{jk}^l - \eta \delta_j^l a_k^{l-1},$$

$$b_j^l \leftarrow b_j^l - \eta \frac{\partial \mathcal{C}}{\partial b_j^l} = b_j^l - \eta \delta_j^l,$$

The parameter η is the learning parameter.

3 Results and Discussion

3.1 Part a

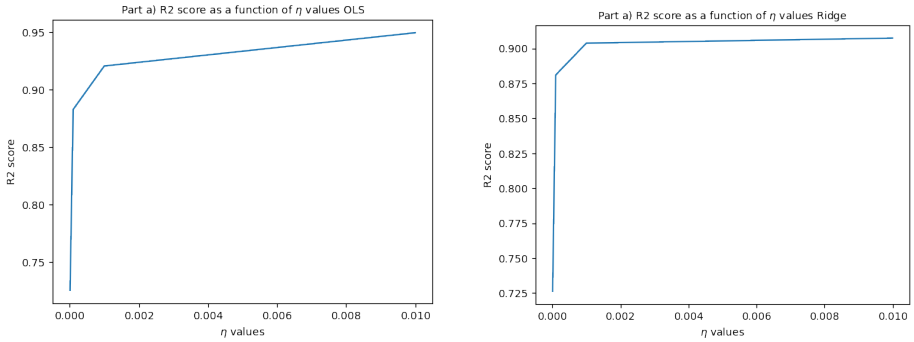

Figure 2: R^2 Scores as a function of learning rate η

We can see from Figure 2 that the R^2 score increases with an increased learning rate η for both OLS and Ridge regression.

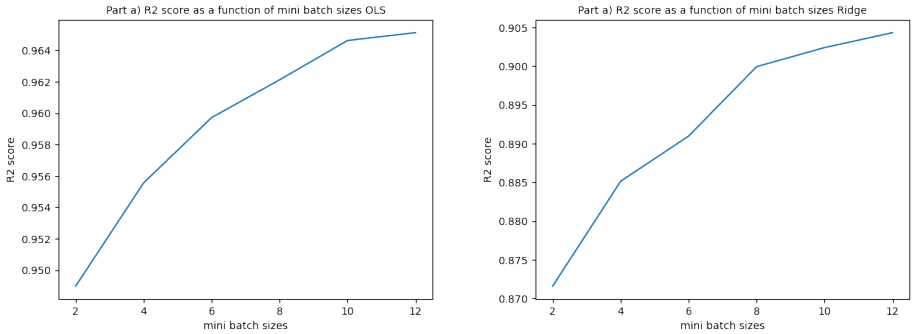

Figure 3: R^2 Scores as a function of mini batch size

A similar thing can be observed in Figure 3 that R^2 increases with the size of the mini batches. This means that the R^2 decreases as the number of mini batches increases.

In Figure 4 the same behaviour as in the last two figures are observed aswell. R^2 increases with an increasing number of epochs. Its worth to note that the apparent downward graph in the middle (for epochs 5k-9k) for the figure for Ridge regression might be due to randomness as the changes in the values on the y aksis is very small.

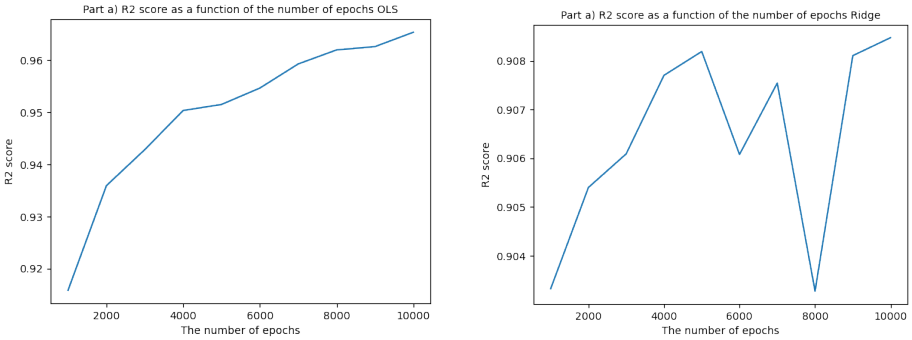

Figure 4: R^2 Scores as a function of the number of epochs

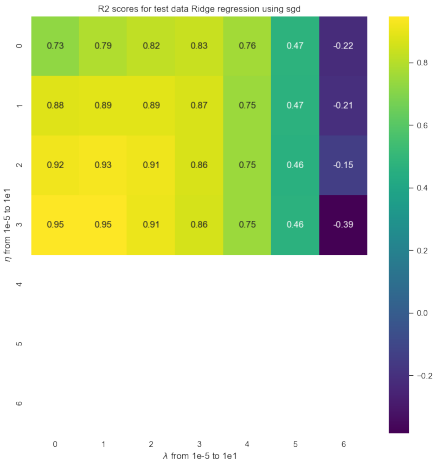

Figure 5: R^2 Scores as functions of the hyper parameter λ and the learning rate η

By using a grid search we can find the optimal R^2 score for Ridge regression as seen in Figure 5. It reaches its maximum of 0.05 for $\eta = 0.03$ and $\lambda = 0.1$. Note that the calculations are very susceptible to overflows. This is apparent in Figure 5 by the white squares lacking numbers.

3.2 Part b

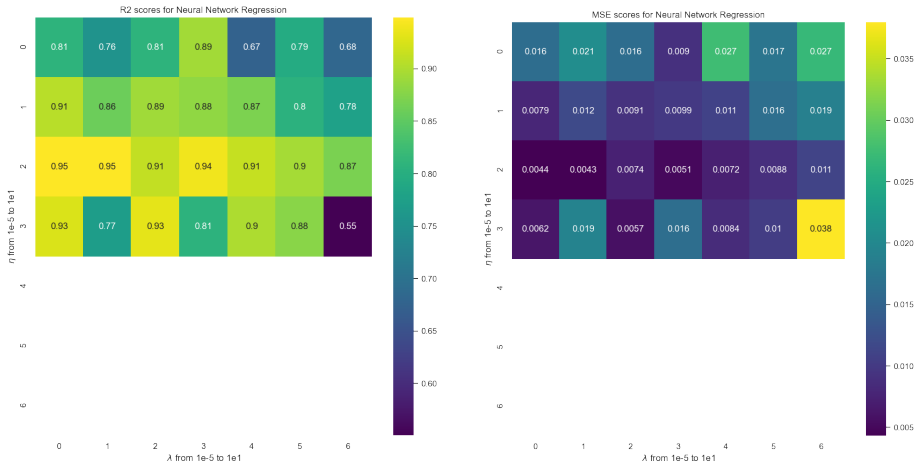

Figure 6: R^2 and MSE Scores as a function of η and λ values for 1 layer of 10 hidden nodes

Figure 6 contains a grid search for the optimal R^2 and MSE values. Note we are encountering overflows the same way as in Figure 5. The maximum R^2 and min MSE score is attained for η=0.0001 and λ=0.01.

3.3 Part c

Table 1: MSE and R^2 for activation functions

Activation method	**MSE**	R^2
Sigmoid	0.0104	0.8778
RELU	0.0044	0.9679
LeakyRELU	0.0062	0.9262

Table 1 contains MSE and R^2 value calculate for a neural network with 1 layer of hidden nodes containing 20 neurons. We can see from the information in Table 1 that RELU has both the lowest MSE and highest R^2 score.

3.4 Part d

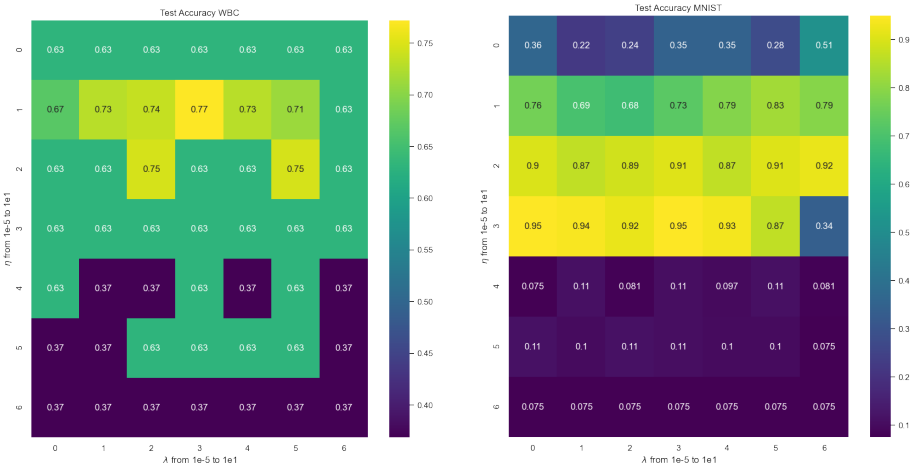

Figure 7: Accuracy scores as a function of η and λ values for 3 layers of 30,20,10 hidden nodes

Figure 7 contains accuracy scores for both the Wisconsin Breast Cancer dataset and the MNIST dataset. We can see that for the WBC set the optimal accuracy of 0.77 is attained for η=1 and λ=0.001 . Note that the low accuracy scores is most likely due to the low number of datapoints (569 rows) in the sklearn dataset.

The MNIST dataset is much larger and the neural network is much better at classifying the images. The optimal accuracy of 0.96 is attained for η=0.001 and λ=0.001 .

3.5 Part e

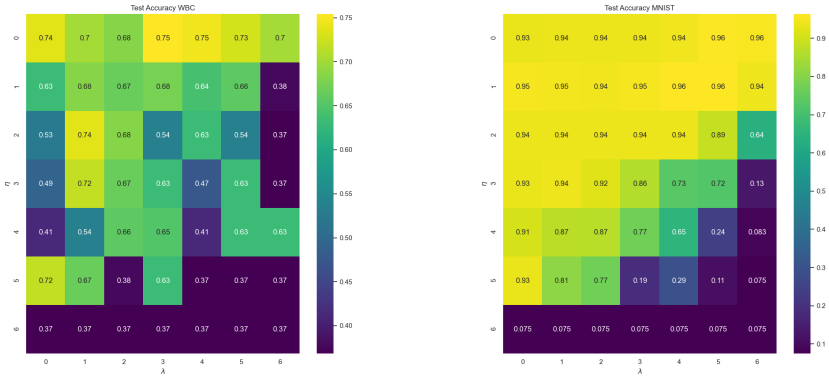

Figure 8: Accuracy scores as a function of η and λ values for logistic regression

We can see from Figure 8 that the optimal classification score of 0.75 for WBC dataset is attained for η=10 and λ=0.01. For the MNIST dataset the optimal classification score of 0.96 for WBC dataset is attained for η=1 and λ=0.0001.

4 Conclusion

In this report we set out to investigate the suitability of using logistic regression and feed forward neural networks to study regression and classification problems.

We have have performed linear regression fits for the Franke function using stochastic gradient descent methods, as well as fitting data from the same function using a neural network. These fits indicate that the neural network with RELU activation are better suited at fitting this data than stochastic gradient descent methods.

Since the neural network is very dependent on its parameters we might get a better fit with a more thourough investigation of the number of neurons and layers. For the classification of data we found that the neural network gave a slightly higher accuracy score than our own logistic regression method for the WBC dataset, but slightly lower on the MNIST datset. With a larger data set from the WBC database we might be able to improve the result using logistic regression.

5 Bibliography

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The elements of statistical learning: data mining, inference, and prediction. 2009.

[2] Lecture Notes for FYS-STK4155. https://compphysics.github.io/MachineLearning /doc/LectureNotes/_build/html/intro.html Accessed: 20.11.2021

[3] Michael A. Nielsen. Neural Networks and Deep Learning. Determination Press 2015

[4] https://github.com/kimgronmo/FYS-STK4155-PROJECT1/blob/main/Report /report%20project1.pdf Accessed: 20.11.2021