Classification and Regression, from linear and logistic regression to neural networks

FYS-STK3155: Project 2

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

November 14, 2020

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 MNIST data set of hand-written numbers. We find that the Frankefunction is best fitted using a neural network with 30 neurons in one layer and 100000 epochs. This gives us an R2 score of 0.9903 for RELU activation function. For the number data set we find that the neural network performs slightly better than our own logistic regression code with accuracy numbers of 0.95 versus 0.939.

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 The performance of these networks have in many cases far superceded human performance, and can to many seem like magic with no apparent connection between input and prediction.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.

In this project we will be using both logistic regression and a Neural Network to teach the computer to recognize hand written digits. 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.

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.

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. 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 $k = 0, \dots, K-1$ (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 can be 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.

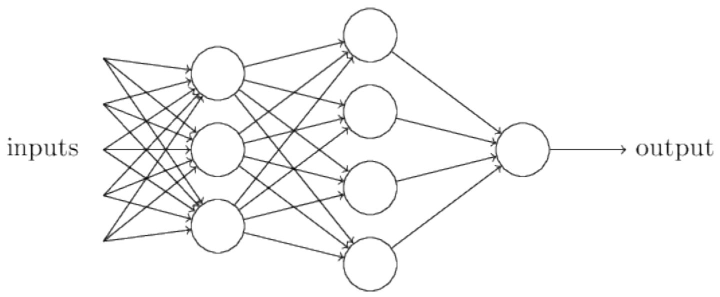

Figure 1: Image representing a neural network with two hidden layers and one output layer. Reference [2].

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

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

A common activation function that we use is the Sigmoid function

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

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

2.4 Back propagation

We use the following algorithm in performing the back propagation adjustments:

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 $l = 2, 3, \ldots, L$.

We compute the ouput error $\hat{\delta}^L$ by computing

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

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.

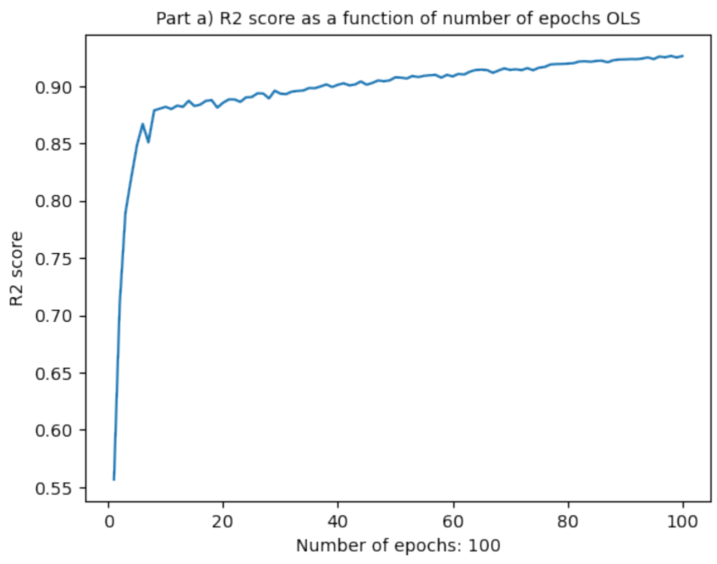

Figure 2: Plot of the R2 score versus number of epochs for OLS regression with stochastic gradient descent method. We can see that the graph flattens out approaching 0.935. Which is a much lower score than what was achieved with ordinary least squares.

3 Results and Discussion

Further results for running the code are are included in the github folder.

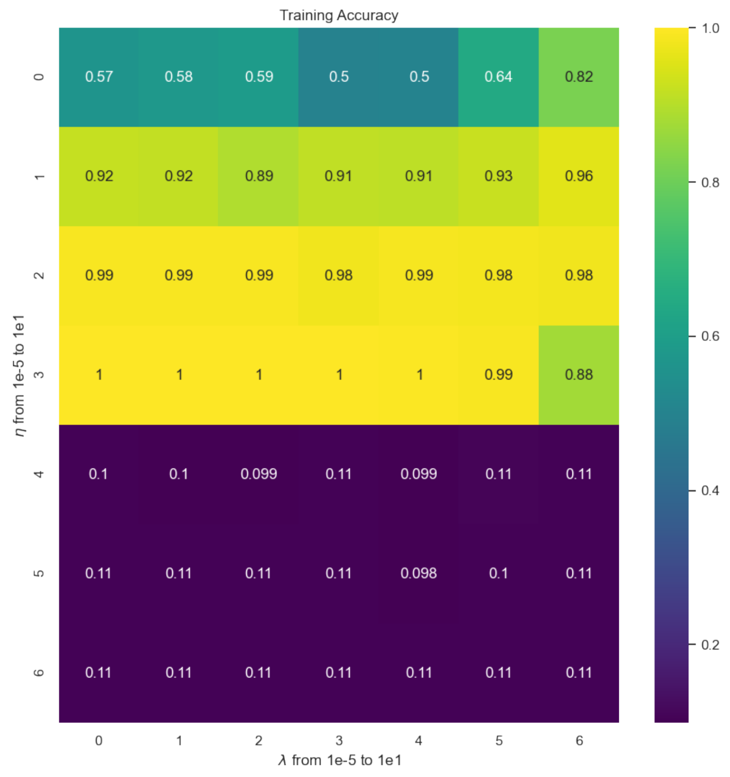

Figure 3: Training accuracy for neural network with 30 neurons in one hidden layer.

Table 1: R2 score for Polynomial 5 Franke Function

Regression method	**R2 values**	**lambda**
OLS	0.9728	Na
Ridge SGD	0.88	0.01
OLS SGD	0.933	Na

We note that in figure 3 that training accuracy for the neural network on the MNISt data is highest for $\eta = 0.01$ with $\lambda =$ 1e-7 with accuracy score of 1.0 . However we can see from figure 4 that these hyper parameters does not get the best accuracy score from the test set. Both $\eta = 0.01$ with λ values 0.001 and 1 give better test accuracy. This is most likely due to the model being overfitted on the training set so that test accuracy suffers. It is worth to note that the dataset from sklearn is very limited, so that using a larger test and training set from the MNISt database would likely get a more

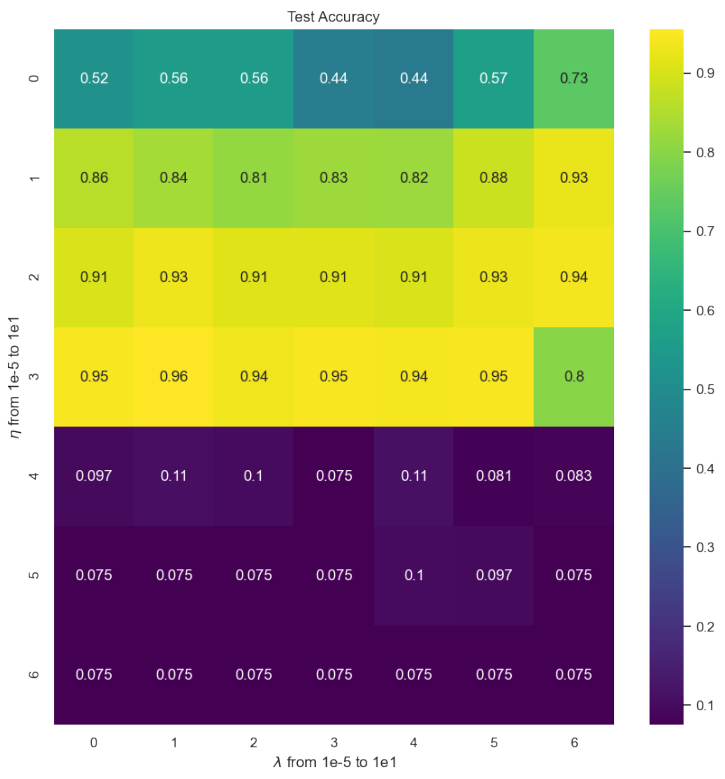

Figure 4: Test accuracy for neural network with 30 neurons in one hidden layer.

Table 2: MSE and R2 scores for Polynomial 5 Franke Function for test data

Type of neural network	**MSE values**	**R2**
Sigmoid	0.00298	0.9604
RELU	0.00073	0.9903
LeakyRELU	0.00106	0.98585

accurate result and avoid overfitting the model.

Table 3: Accuracy score for fitting the logistic regression model

Regression method	**MSE Values**	**lambda**
sklearn	0.9583	Na
logistic with SGD	0.939	0.001

As we can see from Table 2 our own method for logistic regression does not perform aswell as the importert function from sklearn. This is likely to be due to lack of tweaking the hyper parameters and also scaling of the learning rate.

4 Conclusion

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 or ordinary least squares regression. Its worth to note that I got a strange result (rather low R2 score) when using SGD for on the FrankeFunction data. This could be due to an error in our code and cause us to conclude incorrectly about the suitability of using a stochastic gradient descent method on this data set. Further investigation into this matter is necessary to draw a proper conclusion.

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 image data we found that the neural network gave a slightly higher accuracy score than our own logistic regression method. With a larger data set from the MNIST database we might be able to improve the result of our neural network.

5 Bibliography

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

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

[3] Lecture Notes for FYS-STK4155 at https://github.com/CompPhysics/MachineLearning