Report for Classification and Regression
From Linear and Logistic Regression to Neural Networks

Katerina Tsilingiri (2806), Chrysa Noli (2790), Panos Petropoulos (2610)

May 24, 2022

Contents

Abstract

This report contains all the techniques we have followed in order to develop our own feed-forward neural network (FFNN). More specifically, we have studied two different types of problem, regression and classification and we have tried to estimate the values using our own Neural Network. Also, there were used some methods, like mean-squared error and R^2, that we have learned during the previous lectures of the course Machine Learning.

1 Introduction

The goal of this project is to become more familiar with the concept of Neural Networks and learn how to write one on our own. The project is divided in six parts: in the first part we have to study the gradient descent method, in the second and third part, we have to create our own Neural Network in order to be able to predict values of a Regression Problem, in the fourth part we have to modify our NN to make it work for a classification problem, in fifth part we have to write our own Logistic Regression code (using the code from the first part) and compare the results with the previous part and lastly, in the sixth part we have to study in depth the above methods and decide which one is better.

2 Method

2.1 Ordinary Least Squares (OLS) Regression

Ordinary Least Squares (OLS) regression is a technique for estimating coefficients of linear regression equations which describe the relationship between one or more independent quantitative variables and a dependent variable (simple or multiple linear regression).

In a few words, we want to fit a degree-k polynomial $h(x) = c_0 + c_1 x + \ldots + c_k x^k$, for various values of k, to a dataset sample $T = \{(x_0, y_0), \ldots, (x_{N-1}, y_{N-1})\}$

So, in order to find the h(x), we need to calculate the coefficients

$$\mathbf{c} = [c_0, \ldots, c_k]^T$$

that minimize the least squares error:

$$L_T(\mathbf{c}) = \sum_{n=0}^{N-1} [y_n - h(x_n)]^2 \ .$$

So, we have to solve the following linear equation:

$$c_0 1 + c_1 x_n + \ldots + c_k x_n^k = y_n$$

or with matrices:

$$A\mathbf{c} = \mathbf{a}$$

where

$$A = \begin{bmatrix} \mathbf{x}_0^T \\ \vdots \\ \mathbf{x}_{N-1}^T \end{bmatrix} \quad \text{and} \quad \mathbf{a} = \begin{bmatrix} y_0 \\ \vdots \\ y_{N-1} \end{bmatrix}$$

To find the coefficient vector c with OLS regression we need to minimize the least squares error:

$$\widehat{\mathbf{c}} \in \arg\min_{\mathbf{c}} \|A\mathbf{c} - \mathbf{a}\|^2 .$$

We can name the system which produced, depending in dimensions of matrix A as:

- exact system (interpolation)
- over-determined
- under-determined

We evaluate its accuracy by plotting the polynomial or by computing and plotting the errors in order to find out if the polynomial fits our data well.

2.2 Logistic Regression

Logistic Regression has as main goal to predict the class of an input using the natural logarithm and the method of Gradient Descent. Also, we are using the sigmoid function to help us classify the data. Since our dataset (Breast Cancer Prediction) has only 2 classes (0 and 1) and the range of the sigmoid function is (0,1), this function is the appropriate to separate the various datapoints and we usually call it the hypothesis function. Below is presented the sigmoid function :

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

Having evaluated the predictions of the Logistic Regression, it is necessary to measure the accuracy of the model. Therefore, we must use a Loss Function called Log Loss. It is important to note that, other Loss Functions like MSE are not very useful because our target values will be either 0 either 1 and we will not be able to estimate the accuracy with $(y_{act} - y_{pred})^2$. Below is the Log Loss :

$$LogLoss = -\frac{1}{N}\sum_{i=1}^{N}(y_i log(p(y_i)) + (1 - y_i) log(1 - p(y_i)))$$

where $p(y_i)$ represents the probability the output to be 1. The form of the above function can change, depending on the class that our data point (input) belongs.

2.2.1 Gradient Descent

The method of Gradient Descent is very important for the Logistic Regression. It is an iterative algorithm, which tries to minimize the Loss Function of the Regression. In order to achieve that, the gradient of the Log Loss is calculated with respect to each parameter θ. Also, each gradient is updated in every iteration. The procedure is continued until convergence, which means that the minimum cost has been achieved.

$$\theta_j \leftarrow \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Unfortunately, if our dataset has many datapoints, it is very possible that this procedure will take a lot of time. For this reason exists another method, similar to Gradient Descent, called Stochastic Gradient Descent. The only difference is that, instead of calculating the gradient in all the datapoints, the algorithms chooses randomly a few points to calculate the gradient. Therefore, the execution time is reduced significantly. Also, in this algorithm, the θ is updated very often, which helps us avoid the various local minimums that may exist in the Loss Function.

2.3 Neural Networks

In general, the Neural Networks contain multiple layers/neurons and take as input x and generate an output y. In this project, we have create a feed forward Neural Network, where the data (inputs) "travel" in one direction (from the input nodes to the output nodes, through the different hidden layers). We have studied two different types of NN : for Classification Problem and for Linear Regression Problem.

The back propagation algorithm begins by comparing the actual value output by the forward propagation process to the expected value and then moves backward through the network, slightly adjusting each of the weights in a direction that reduces the size of the error by a small degree (learning rate) as the mathematical equation describes:

$$w_{ij} = w_{ij} - lr * d^i y_h^j$$

Both forward and back propagation are re-run a plenty of times (iterations - parameter of our FFNN) on each input combination until the network can accurately predict the expected output of the possible inputs using forward propagation.

Parameters of our FFNN:

- learning rate
- number of neurons in hidden layer
- activation function (sigmoid for classification, relu for regression)
- iterations

For both Linear Regression Problem and Classification Problem we choose 2 hidden neurons so we have a neural network 2 x 2 x 1 (input x hidden x output layers) that is enough for our 2 input problem we have to deal with. We have 2 layers, the hidden one and the output layer.

2.3.1 Linear Regression Problem

For this type of model, we have used the output of the Franke Function defined below, using random numbers for x and y.

$$f(x,y) = \frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) + \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right)$$
$$+\frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) - \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right)$$

Firstly, we have chosen to initialise the weights and the bias of the hidden layers and the output using a uniform distribution.

For the hidden layers and the output layer, for the second part of the project, we have used the reLU activation function $f(x) = max(0, x)$. This function, according to our research, is the most frequently used activation function for hidden layers. Also, for positive values acts as a linear activation function, while for negative values acts as an non-linear activation function.

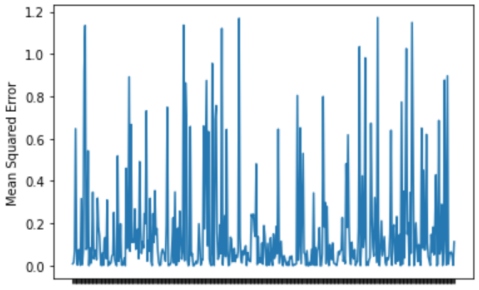

Figure 1: Predicted Values - MSE (for reLU)

For the third part, we have tried other activation functions for the hidden layers like :

- Rectified Linear Unit (ReLU) : $f(x) = max(0, x)$
- Leaky Rectified Linear Unit (Leaky ReLU) : $f(x) = max(0.01, x)$
- Sigmoid : $\sigma(x) = \frac{1}{1+e^{-x}}$

After the training of our Neural Network, we have calculated the MSE, so we can compare them and decide who is better for a Linear Regression Problem.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_{actual} - Y_{predicted})^2$$

The above Mean Squared Errors have been estimated for 2000 inputs and 5 iterations. As we have mentioned above, we were expecting the ReLU activation to have better results than the Sigmoid. Unfortunately, according to the Table 1, we can observe that the Sigmoid activation function has better MSE estimation than the other two. This may have happened because our inputs (of the Franke function) are generated randomly for each method, or maybe there is a mistake in our implementation.

Table 1: MSE estimations

Activation Function	MSE
ReLU	0.1609
leaky ReLU	0.1331
Sigmoid	0.1288

2.3.2 Classification Problem

In general, in Classification Problems we try to predict in which class belongs a datapoint. In the fourth part of this project, we have modified our Neural Network, in order to be able to make this type of prediction. The dataset we have used is called Breast Cancer and it contains only two classes (0 and 1), therefore we are trying to solve a Binary Classification Problem. As we have mentioned above in section 2.3 (Neural Networks), our NN is a Feed Forward NN with back propagation.

In terms of training our Network, again we have used all the activation functions that we have mentioned above (ReLU, leaky ReLU and Sigmoid). Taking into consideration that the range of the sigmoid function is (0,1), we are expecting to have larger accuracy when we train the network using sigmoid as activation function.

Table 2: Accuracy Score

Activation Function	Accuracy Score
ReLU	37%
leaky ReLU	43%
Sigmoid	77%

Regarding the accuracy, since it is a classification problem, we can no longer use the MSE to check how good our neural network is. According to the provided material by our professor, there is one method called Accuracy Score, which uses an indicator function I.

$$Accuracy = \frac{\sum_{i=1}^{n} I(t_i == y_i)}{n}$$

The output of the function I is 1 if t_i (the target/actual value) equals to y_i (the predicted value) and 0 if they are not the same.

According to optimizations, since the results on Table 2 are not very pleasing, we have found that there are other ways to initialize the weights and bias. Unfortunately, we did not have enough time to implement it.

References

[1] How to Choose an Activation Function for Deep Learning, https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/.

[2] Logistic Regression with Gradient Descent Explained — Machine Learning, https://medium.com/analytics-vidhya/logistic-regression-with-gradient-descent-explai

[3] Improving Neural Networks – Hyperparameter Tuning, Regularization, and More (deeplearning.ai Course 2), https://www.analyticsvidhya.com/blog/2018/11/neural-networks-hyperparameter-tuning-r

[4] How to Initialize Weights in Neural Networks?, https://www.analyticsvidhya.com/blog/2021/05/how-to-initialize-weights-in-neural-net