# Neural Network Homework 1 Report

Michał Lemiec

April 14, 2022

# Contents

# 1 Problem description

The main problem was to construct an application that will be able to create a Multilayered Perceptron that will be trained with Backpropagation and tested on two sets from the UCI Machine Learning repository. Also such Multilayered Perceptron should solve classification task and it should be achieved without the usage of any external Machine Learning library such as Tensorflow or Pytorch. Multilayered Perceptron with implemented backpropagation can be called a basic Neural Network. Of course, it should be implemented in a way that user can pass several basic parameters e.g. learning rate or number of hidden layers. Before the presentation of the solution, there is a need for a more theoretical explanation.

## 1.1 What is a Neural Network?

As its name suggests Neural Network is a net made of neurons. A neuron, also known as perceptron, is a basic unit used for any type of decision problem. It was modeled after the real neuron from the human brain. Hence, it can predict the output based on the given input. In Figure 1, it can be seen that single neuron has three different layers: input layer, hidden layer and output layer. The input layer is made of input parameters.

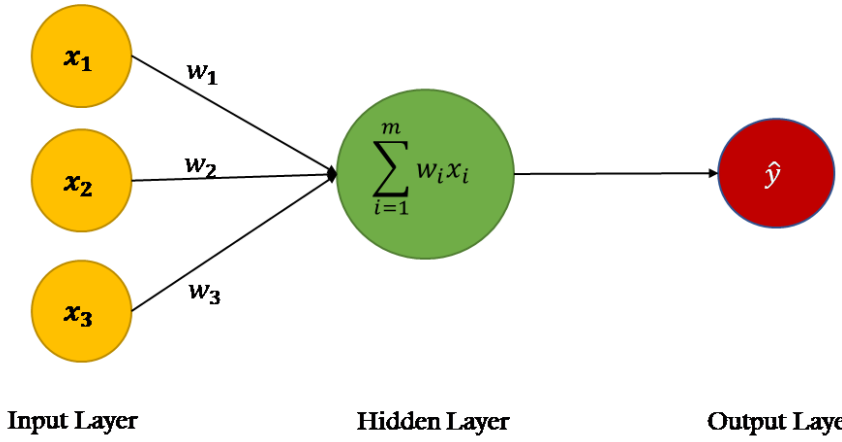



Figure 1: Figure representing neuron with 3 input values. Source: Towards Data Science [1].

Every synapse, known also as input, is represented by $x_1$, $x_2$ and $x_3$ and has a corresponding weight $w_1$, $w_2$ and $w_3$. Inside each neuron there is summation of every multiplication between input and corresponding weight e.g $x_1\dot{w}_1 + x_2\dot{w}_2 + x_3\dot{w}_3$ . What is not presented in the Figure 1 is the additional parameter called bias, which is used to shift the activation threshold for the neutron. After all additive operations, the resulting value is an input for activation function. The role of activation function is to normalize the previously discussed function, to "format" it into telling if neutron will be active or not. If it is active neuron in the next layer can receive its signal.

From that description, it can be deducted that can be used together in order to solve more complicated problems. As it can be seen on Figure 2, neurons create layers which create Neural Network. Here, there are 4 layers: one input(yellow), 2 hidden layers(green) and one output(red). This means that 3 input neurons create one layer, their outputs are passed thought synapses(blue lines) to the first hidden layer, as you can see every output is matched exactly one time with neuron. Then which is connected to next one and then they are connected to the next one. The between the two layers is done in a way that every neuron from first layer is matched with every neuron from second layer exactly once by synapse. Hence this whole concept of Neural Network might seem complicated at first. It is mainly because of the fact that Neural Network is often called a "black box" or function that under specific input should provide predicted output.

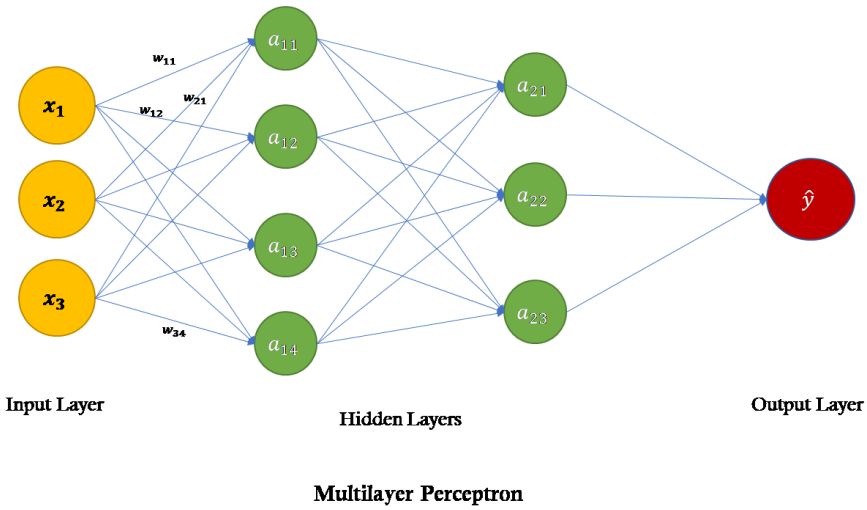



Figure 2: Figure representing neural network with 1 input layer, 2 hidden and 1 output layer. Source: Towards Data Science [1]

## 1.2 Training

What was described in the previous section is called a feed-forward pass in Neural Network. It means that the information is passed only in one direction in order to predict the output. However, the weights need to be updated. Therefore, the backpropagation algorithm is used to, after feed-forward, go from last layer and update the weights in order to minimize the loss(the difference between expected value and predicted). Training or learning is the iterative process of changing Neural Networks' weights. However, it is not so simple, there are some optimization techniques that allow us to make the most of the training process.

## 1.3 Gradient Descent

Gradient Descent is generally known as loss optimization technique. As the name Gradient Descent descends through the gradient, which means that it is responsible for adjusting the weights of the model in a way that loss value is decreasing. The variations of Gradient Descent algorithm are the following:

- Batch Gradient Descent – it is also called the "vanilla" gradient descent. This implementation calculates cost for row in the training dataset but weights are not updated unless all data has proceeded. This whole iteration is called an epoch.
- Stochastic Gradient Descent – this algorithm calculates cost for every row in a dataset and does a weight update after each one of them.
- Mini-Batch Gradient Descent – this implementation is a mixture of both previously discussed examples. It divides the training dataset into smaller batches and does calculate loss for every row in batch. Then it updates weights after the whole batch is learned. It is the most used solution out of those three proposed.

## 1.4 Activation function

As the name suggested the activation function is an algorithm that helps the Neural Network decide if specific neuron should be activated, which means it creates an output that is passed to the next layer as input. There are several activation functions but in this project 3 of them were used in this project for different purposes:

- sigmoid – $\sigma(x) = \frac{1}{1+e^{-x}}$. Hence the output value is in 0 – 1 range, which normalize the values that are passed to the next layer. It can be used for binary classification with some effect.

- hyperbolic tangent $-\sigma(x) = \frac{e^{2x}-1}{e^{2x}+1}$ Similar to the one above but value ranges from -1 to 1. Since, the rage is bigger the convergence is often faster than for sigmoid.
- softmax – is the activation function used often in classification tasks, on the output layer. Its goal is to create the probability distribution for different classes. Hence, number of softmax output neuron needs to be same as the number of classes in the dataset. Moreover, when taking class with highest probability we assume that it is the predicted output.

### 1.5 Learning rate

As it was already presented weights have to be calculated and updated during training process. The allowed amount of change to the mentioned weights during each iteration is called a learning rate. Learning rate determines how big are the steps that a particular gradient descent algorithm takes in the direction of global minimum. In fact it tells us how fast we will move towards the optimum values for weights.

As always there is no rule of thumb on how to find learning rate or any universal learning rate that works best in every case. Typically it is chosen by empirically trying and testing different values on building Neural Network.

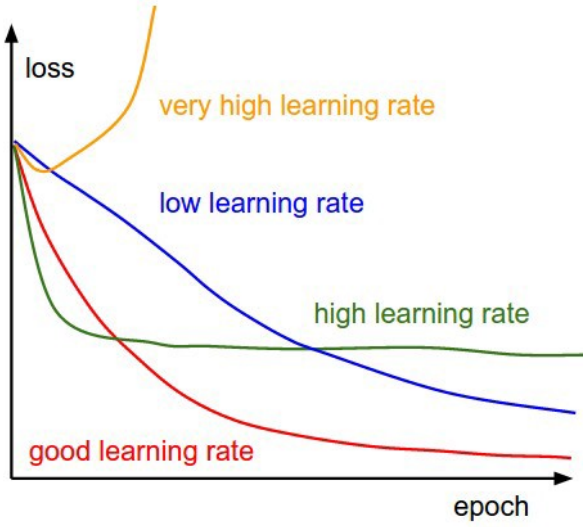



Figure 3: Figure representing loss function trends according to learning rate value. Source: Towards Data Science [2]

### 1.6 Momentum

In Neural Network, momentum is an additional parameter of a gradient descent. Together with learning rate, it allows to achieve optimal value faster, often by omitting local minima. In short it is a simple variant that, when implemented, improves both training speed and accuracy. In real life it could be compared to physical momentum. For instance, a ball which has momentum, has also the ability to roll down the slope or with enough value to even roll up a little bit.

## 2 Datasets description

In order to test the potential solution the categorisation data is needed. Therefore, the following two datasets were chosen from UCI repository: the bigger one connected with determining wine quality [3] and the smaller one with flower type classification. There are thoroughly discussed in the next subsections.

### 2.1 Wine quality dataset

The wine quality dataset is the larger of two proposed sources of data. It contains separate sets of red, 1599 observations, and white wine samples, 4898 observations with the same input format of 11 features per row. For this report the red wine samples were used due to the fact that there training is going to be faster.

As it was already said there are 11 attributes used in every row. All of them are real, positive numbers.

1. Fixed acidity
2. Volatile acidity
3. Citric acid
4. Residual sugar
5. Chlorides
6. Free sulfur dioxide
7. Total sulfur dioxide
8. Density
9. pH
10. Sulphates
11. Alcohol

There is also one additional column that will be used as labels for train and test data. This column is called quality and represents the score for each wine within the 0 - 10 range. Hence, there are 11 unbalanced which means that is a possibility that there are more mediocre wines than excellent ones.

### 2.2 Iris dataset

This is one of the most popular datasets in the Machine Learning community. It represents different flowers with their numerical characteristics. It was first used and created by Ronald Fisher [4]. The dataset is made of 150 samples and 5 columns. The first 4 are the numeric values given in centimeters:

1. Sepal length
2. Sepal width
3. Petal length
4. Petal width

The last column is like in the case of wine quality dataset also a class label. However, in this case the labels are not numbers but strings. The following flower names exist:

- Iris Setosa
- Iris Versicolour
- Iris Virginica

Additionally, unlike the wine quality dataset the classes are all balanced. It means that each class has one third of the whole data samples(50 samples per class).

## 3 Experimental setup

The following section will cover the work of implemented solution. This means that the manual of how to run the code as well as experiments done on Neural Networks will be discussed in the next sections.

## 3.1 Chosen solution

After the whole theoretical discussion this part is focused more on the chosen approach that was used for implementing solution. All previously mentioned aspects for Neural Network were implemented in order to create application ready to train and test on specific datasets. Firstly, the backpropagation algorithm was implemented, later the Mini-Batch Gradient Descent was chosen as the one most suited to do the right job. Both sigmoid and hyperbolic tangent were implemented for the activation function for hidden layers. Furthermore, on the final, output layer there is softmax activation function together with cross entropy loss function. All other experiments are discussed in details in the next sections.

## 3.2 Application manual

The solution was developed under the several assumptions. First of all, this is not an application with user interface. This means that there is no window with buttons to click on. Entrusting is it run from the perspective of the terminal or command. In order to start application from its source code, user has to have a Python version 3 installed on their PC. Then, user simply has to run main.py file. After starting application the messages will appear on screen. Table below lists all messages with example user response:

| Message shown by application | Proposed user response for wine quality |
|---|---|
| "Choose dataset<br>1. Wine quality<br>2. Iris dataset : " | 1 |
| "Enter number of neurons in input layer : " | 11 |
| "Enter number of hidden layers : " | 3 |
| "Enter number of neurons for hidden layer #1 : " | 22 |
| "Enter number of neurons for output layer : " | 11 |
| "Choose activation function for hidden layers<br>1.sigmoid<br>2.tanh : " | 1 |
| "Enter number of epochs : " | 100 |
| "Enter batch size : " | 16 |
| "Enter value for learning rate : " | 0.001 |
| "Enter value for momentum : " | 0.9 |

Table 1: Messages that show in application. Sorted by occurrence.

As a starting point for further explorations, the two separate sets of initial parameters were chosen. For the wine quality samples:

- layers – input layer with 11 neurons two hidden layers with 20 and 16 neurons and output layer of 11 nodes.
- activation function – hyperbolic tangent
- learning rate – 0.001
- momentum – 0.9

For the Iris data:

- layers – input layer with 4 neurons two hidden layers with 8 and 6 neurons and output layer of 3 nodes.
- activation function – hyperbolic tangent
- learning rate – 0.001
- momentum – 0.9

## 3.3 Batch size

The first hyperparameter taken under experiments was batch size. As it was already mentioned, the mini-batch gradient descent algorithm is the one used in this solution hence the size of those batches had to be specified at the beginning of the training.

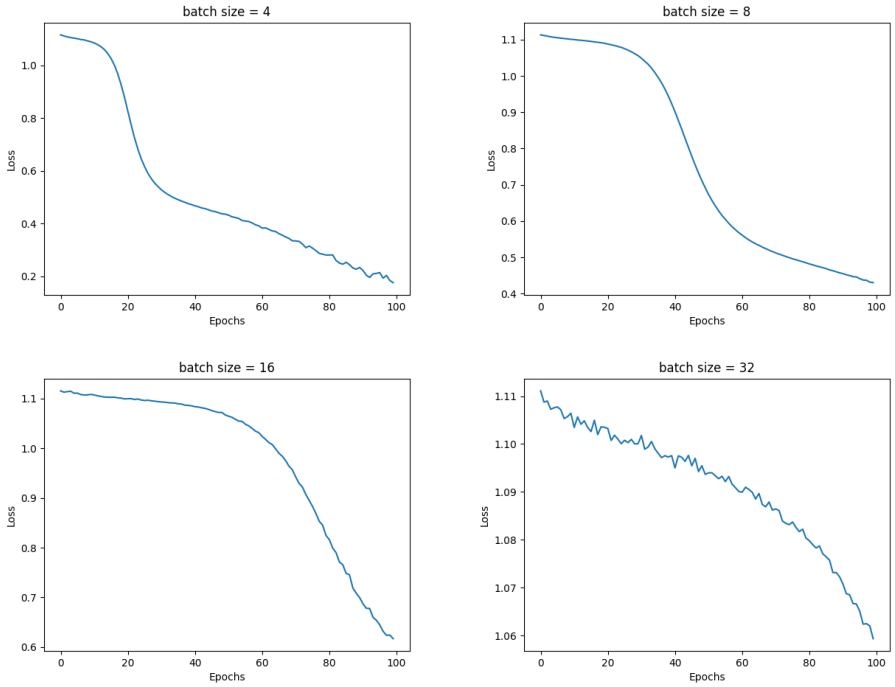
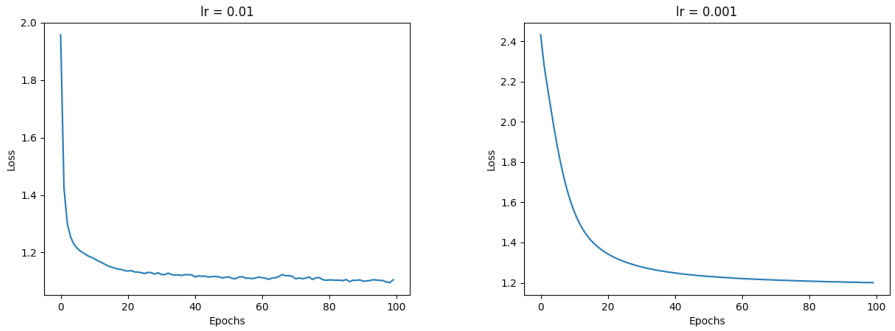



Figure 4: Graphs showing loss behaviour for 4 different batch sizes during training.

The above figures present learning sessions for Iris dataset. There were chosen due to the fact of having more interesting shape. According to the steepness and speed of learning the first figure for batch size 4 was chosen as the best one.

## 3.4 Learning rate

After finding the appropriate batch size Hence, it allows us to manipulate the batch size in order to find the value for which loss function has the best learning rate shape , see Figure 3.



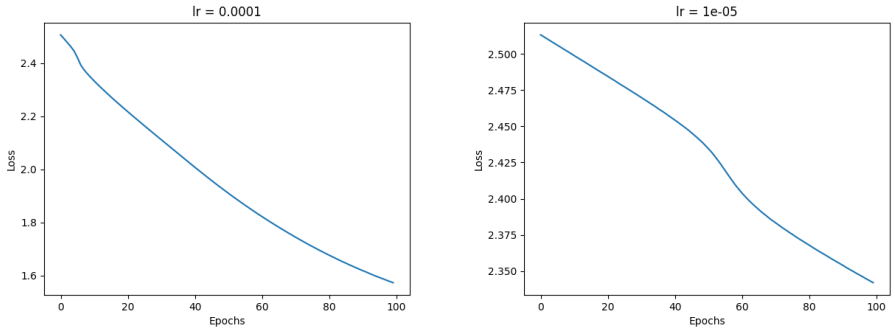



Figure 5: Graphs showing loss behaviour for 4 different during training.

Starting from the left-upper corner we can sum up the results in the following way. The first learning rate value is to big. The function takes a big step and then simply behaves almost like a straight line. For the second figure the shape is more curved, here the step is not so great. It generally the best one here. The last two figures have definitely too small learning rates. The figures here are based on the wine dataset but similar shape was achieved for Iris.

## 3.5 Sigmoid vs hyperbolic tangent

As a last experiment the difference between activation functions was analyzed. The graphs before were taken on Iris dataset.

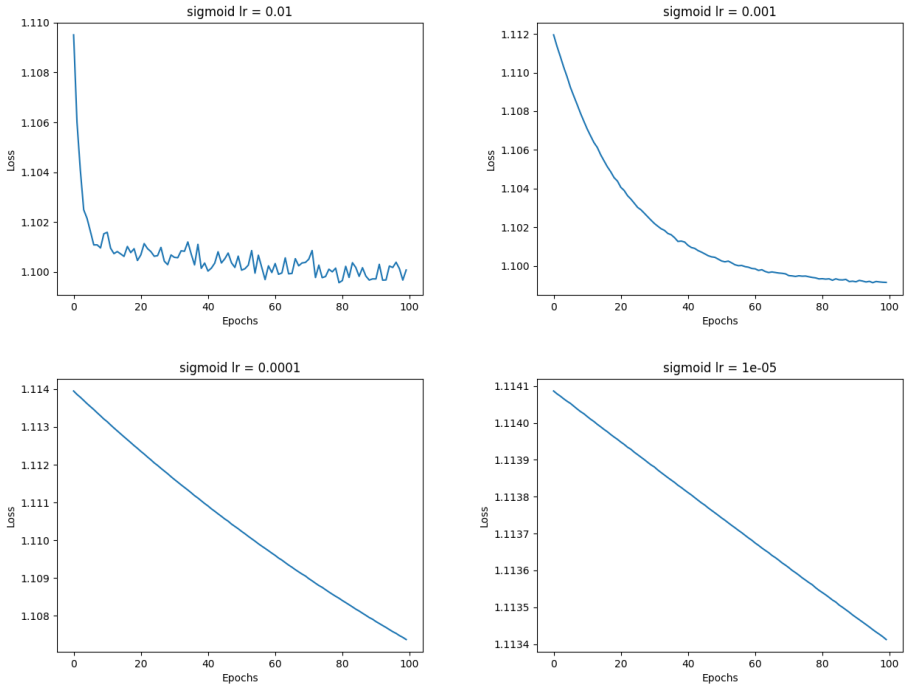



Figure 6: Graphs showing loss behaviour for 4 different learning rate values, beginning from left-upper corner learning rate values are .

The following figures were shown only for sigmoid function with different learning rates, the same values that were presented in Figure 3.4. The difference is obviously the loss value achieved at the end. It is much higher that the one achieved with hyperbolic tangent. However, no such change was observed for wine quality dataset. Therefore, there are no figures for these training sessions.

### 3.6 Momentum

After changing the learning rate it was time to change the momentum. However, there were no great improvement between the different values, values started from 0.1 and rose to 0.9 by 0.1. This was true for both wine quality dataset and Iris dataset. Since, 0.9 is often used as a momentum value it was chosen as the optimal for both datasets.

### 3.7 Optimal set of parameters

Finally, after many possible tryouts on both datasets, the following two sets of parameters were created. For the wine quality samples:

- layers – input layer with 11 neurons two hidden layers with 20 and 16 neurons and output layer of 11 nodes.
- number of epochs – 100
- batch size – 16
- activation function – hyperbolic tangent
- learning rate – 0.001
- momentum – 0.9

For the Iris data :

- layers – input layer with 4 neurons two hidden layers with 8 and 6 neurons and output layer of 3 nodes.
- number of epochs – 100
- batch size – 4
- activation function – hyperbolic tangent
- learning rate – 0.01
- momentum – 0.9

## 4 Conclusions

As it could be observed on the figures and tables from previous section, the parameters are depended on the datasets used. For, the smaller Iris dataset implemented Neural Network with proposed parameters was able to achieve accuracy around 90% with low loss value, around 0.15 for test data. However, for some slight alteration of the given parameters the overfitting occurred. For the wine quality dataset the highest accuracy achieved was around 46% with loss equal to about 1.2. The interestin thing is that not in both cases the hyperbolic tangent was better activation for hidden layers. In the case of wine quality data, there were no change in loss or accuracy value. It is probably due to the fact that there were more input values are presented in one row than in the case of Iris dataset. And as already mentioned, in the case of Iris dataset, accuracy for tangent was growing more rapidly with lower loss compared to sigmod. In both cases choosing the batch size near 2 or 4 was pretty close to changing mini-batch implementation to Batch Gradient Descent.

### 4.1 What could be done better?

After the implementation process there were several conclusions drawn about what can be done better. Firstly, the additional activation function could be implemented and used instead of sigomid or hyperbolic tangent. Rectified Linear Unit(ReLu) is the most used activation function right now. It is the best option when compared to others like sigmoid or hyperbolic tangent because it does not experience vanishing gradient problem, the problem encountered for wine quality dataset that there were no new

updates on weights, because the changes were to small. Usage of adaptive learning algorithms could also increase the accuracy in the case of larger dataset. Under the chosen constraints the learning rate could be adopted depending on the current training situation. If Neural Network gets stuck like in case of the one created for wine quality dataset, the good idea is to dynamically change the learning rate and omit the local minimum. Alternatively, the feature selection method could be implemented for wine quality dataset. The creators of this specific samples claimed that probably some input attributes are depended or related to each other. If this selection is done successfully, it will speed up the training process and most likely improve the accuracy and decrease the loss. Additionally, the number of hidden layers could also be modified. After setting semi-optimal number of hidden layers there were no further experiments with number of hidden layers as well as the number of neurons. Lastly, the application could be improved by catching errors like wrong input type or by adding graphical user interface. This could shorten time needed for testing other datasets than wine quality or Iris, which are currently hard-coded in the solution.

# References

[1] V. Sahu, "Power of a single neuron," Jun 2021.

[2] A. Mohanty, "The subtle art of fixing and modifying learning rate," Jul 2019.

[3] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," Jun 2009.

[4] R. Fisher, "The use of multiple measurements in taxonomic problems."