Project 1 Report

Multilayer perceptron implementation and analysis

Małgorzata Wachulec, 293790
Aleksandra Wichrowska, 291140

22 March 2021

1 Project scope

Our task was to implement a multilayer perceptron using only basic packages, that employs the backpropagation algorithm and takes in the following parameters:

- number of hidden layers and number of neurons in hidden layers
- activation function
- bias presence
- batch size
- number of epochs
- learning rate - eta
- momentum rate - alpha (when alpha equals 0 no momentum is added)
- problem type: classification or regression

We decided to include also other arguments, such as the weights' initialization method. There are four implemented methods of weights' initialization: one which initializes all weights to zeros, one which initializes weights with random numbers from uniform distribution, and two more sophisticated methods: He method and the default Xavier method, in which the distribution from which the random numbers are generated, depends on the number of neurons in each layer. Another added argument is the random state, which is a seed used when initializing the random number generator. The users can also specify the loss function they want to use by setting argument measure to one of the following: 'MSE', 'cross_entropy', 'binary_cross_entropy'. We also decided to add the stopping condition to our implementation. In order to do that we added two new arguments:

1. The minimum improvement of the loss function expected after each epoch (iteration)
2. Number of epochs after which the learning process is terminated in case the minimum improvement requirement is not met

Our implementation consists of two main classes: MLP and Layer, that are saved in the mlp.py and layer.py scripts, respectively, and functions implementing activation and evaluation functions, as well as enabling data transformation and plotting, that were grouped by their functionality and can be found in scripts: utils.py, activation.py, evaluation.py and plots.py.

File VisualizeNN.py was taken from the repository https://github.com/jzliu-100/visualize-neural-network, and adjusted to fit our purpose. This package allows for visualization of network's architecture and tracking learning progress iteration by iteration - this means visualization of the weights and propagated error on each edge of the network.

File main.py was used to test the functionalities of our solution, for example the to track weight changes during learning process and plot the outcomes of prediction on the test set for 2 dimensional data. File test.py contains the scripts we used in order to generate plots and tables for this report.

2 Choosing activation function

We implemented the following activation functions:

1. Linear function
2. Rectified linear unit function (returns a linear function for argument greater than 0, otherwise 0)
3. Hyperbolic tangent function
4. Sigmoid function
5. Softmax function

Linear function is the default activation function used in the output layer in a regression task. In a classification task, the default activation function in the output layer depends on the loss function used. When no loss function is specified, the default loss function for classification is cross entropy, which is used together with the softmax function in output layer. When the user specifies the loss function to binary loss function, then sigmoid function is used in the output layer.

In the hidden layers we assume that ReLU, hyperbolic tangent or sigmoid function is used and these are the functions tested in this section. Linear function will not be tested as using linear function in hidden layers can be reduced to a single hidden layer and hence, to a simple linear model, which will work only when the data is linearly separable.

In order to analyse how the activation function effects the accuracy of the model, we decided to keep the remaining arguments constant. For one of the provided classification sets we trained a simple neural network with 1 hidden layer with 5 neurons, with learning rate equal 0.01 and momentum rate equal 0.9 in order to speed up the convergence of the algorithm. We specified the maximum number of epochs to 100 epochs. In the output layer, the softmax function was used, together with the cross entropy loss function. In the hidden layer one of the three activation functions was used. Then we compared the accuracies of the model after every iteration for each activation function. The outcome is presented in Figure 1.

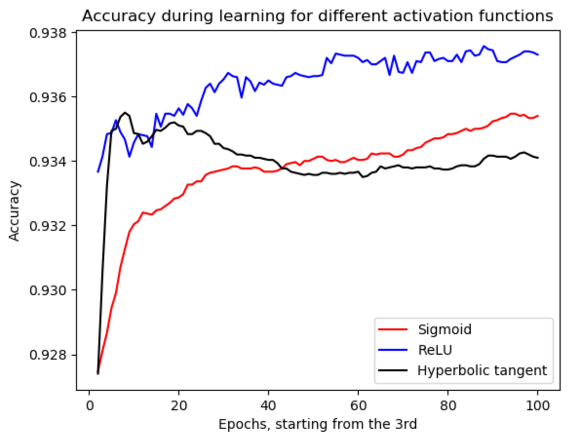


Figure 1: Accuracy change during learning process for different activation functions

However, the outcomes presented in Figure 1 are dependent on the random state and could not be significant. For this reason we decided to run the following experiment for 100 different random states and measure the mean accuracies on both training and test sets for each activation function. In order to speed up the process, we enabled the stopping condition in case of lack of further improvement of the loss function. Other parameters were kept the same as before. The results are presented in Table 1. In fact, they are not at all similar to what we could assume after looking at Figure 1. The best mean accuracies, presented in Table 1 for both sets were achieved for the sigmoid activation function. The second best were the accuracies of the model with the hyperbolic tangent. The worst outcomes were achieved for the ReLU function - mean of around 0.94, which is much lower than the other two activation functions.

Activation function	Sigmoid	ReLU	Hyperbolic tangent
Mean accuracy on training set	0.9958	0.9408	0.9931
Mean accuracy on test set	0.9957	0.9406	0.9927

Table 1: Mean accuracies for each activation function for 100 different initial random states

3 Analyzing network's architectures

We conducted some experiments to see how the neural network architecture (the number of layers and the number of neurons in the layers) affects the model performance.

We compared the results obtained by neural networks with one hidden layer consists of 5, 10, 25, 100, 200 and 500 neurons. The performance of these networks is shown in the Figure 2. The range of values on the Y axis is 0.93 to 0.94. In general, the more neurons in the layer, the better the results are obtained by the network. Accuracy achieved by the network with 500 neurons in the hidden layer are the highest but the results of the networks with 200 and 100 neurons are not much lower. With computation time, less complex architectures are often a better choice.

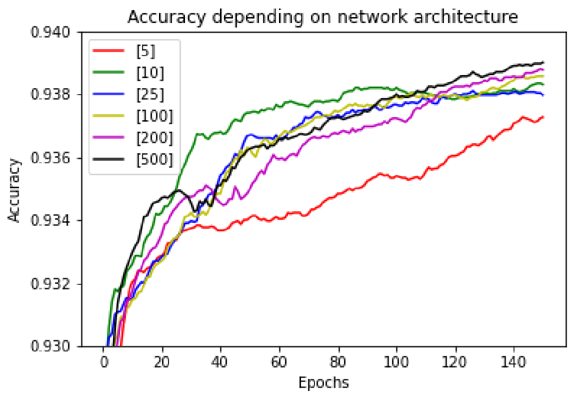


Figure 2: Perfomance of networks with one hidden layer and different number of neurons

We also checked how the number of layers affects the model results. For this purpose, we conducted tests on networks with one, two, three and four hidden layers, each consisting of 10 neurons. The performance of these networks is shown in the Figure 3. The range of values on the Y axis is 0.93 to 0.94. After 150 epochs the best accuracy is achieved by the network with four hidden layers, the results of the other networks are slightly lower.

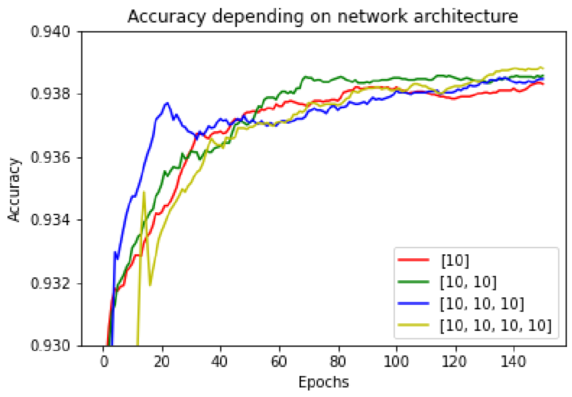


Figure 3: Perfomance of networks with different number of hidden layers

4 Analyzing loss function's effect on accuracy

We implemented the mean squared error as the loss function for the regression tasks and cross entropy error and binary cross entropy as loss functions for the classification tasks.

For the binary classification taks we compared two approches - binary cross entropy and cross entropy as loss functions. Binary version is paired up with the sigmoid function as activation function in the output layer and the standard cross entropy is paired up with the softmax in the outpul layer.

Accuracy obtained by trained networks is presented in the Figure 4. During the initial 100 epochs, the network with binary cross entropy performs better, then the network with cross entropy is slightly better. In the end, the results of both networks are almost the same.

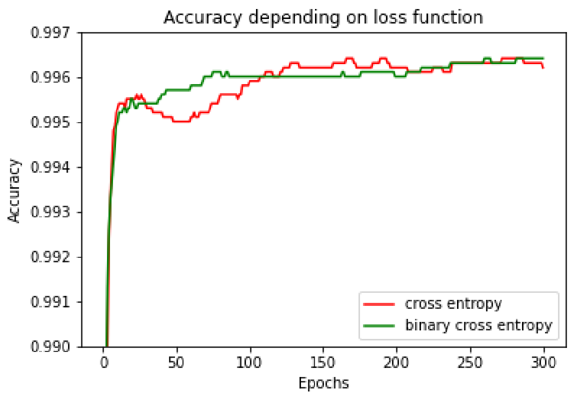


Figure 4: Perfomance of networks with different loss functions

5 Results on MNIST data set

In this section we train our network to classify digits from the MNIST data set. We consider the effect of the architecture, loss function, activation function and weights initialization method on the accuracy on the train and test sets.

5.1 Results for different architectures

First thing we compared was the accuracy depending on the architecture of the network. We kept all the other hyperparameters the same and equal to the default values. The accuracies on both training and test set for different number of hidden layers and different numbers of neurons in each layer are presented in table 2. The best accuracy on training set was achieved for a network with one hidden layer and 500 neurons, which was the maximum number of neurons considered. Also, having the same number of neurons within one hidden layer was more effective than splitting it among two or more hidden layers.

5.2 Results for different weights initialization methods and loss functions

We considered the effect of weights initialization method and loss function on the accuracy of the network. Table 3 presents accuracies for different weights initialization methods. The highest results on both train and test data sets were achieved for the Xavier initialization method.

# of neurons in hidden layers	Accuracy on train set	Accuracy on test set
20	0.9863	0.957
50	0.9962	0.971
100	0.9976	0.978
250	0.9981	**0.9803**
500	**0.9982**	0.9802
20, 20	0.9815	0.954
50, 50	0.9960	0.970
100, 100	0.9979	0.976
20, 20, 20	0.9760	0.949

Table 2: Results for different architectures

Initialization method	Accuracy on train set	Accuracy on test set
Uniform	0.9946	0.9777
He	0.9979	0.9798
Xavier	**0.9982**	0.9802

Table 3: Results for different weights initialization methods

The comparison of results for different loss functions was presented in Table 4. Although cross entropy with softmax activation function in the output layer resulted in higher accuracy on training set, binary cross entropy with sigmoid achieved better score on test set. Nevertheless, we assume we are not given the test set in advance and choose the hyperparameters that give the best score on the training set. For this reason we choose cross entropy with softmax for our final model.

Loss function	Accuracy on train set	Accuracy on test set
Cross entropy with softmax	**0.9982**	0.9802
Binary cross entropy with sigmoid	0.9947	**0.9820**

Table 4: Results for different loss functions and activation functions in the output layer

5.3 Results for different activation functions

We also compared the training processes and results of the network for different activation functions. The accuracy of the network on the training set for each iteration is shown in Figure 5. In this figure we can see that the algorithm did not converge for the ReLU activation function - only 20 iterations were considered as by default the algorithm runs for 20 more iterations when the loss function does not decrease (it is an argument of our network related to the stop condition). However, the hyperbolic tangent activation function converged quicker and achieved a better final accuracy on training set, equal to 0.9991, than the default sigmoid function, which gave accuracy equal to 0.9982. Also the accuracy on the test set was higher for the hyperbolic tangent - 0.9815 instead of 0.9802.

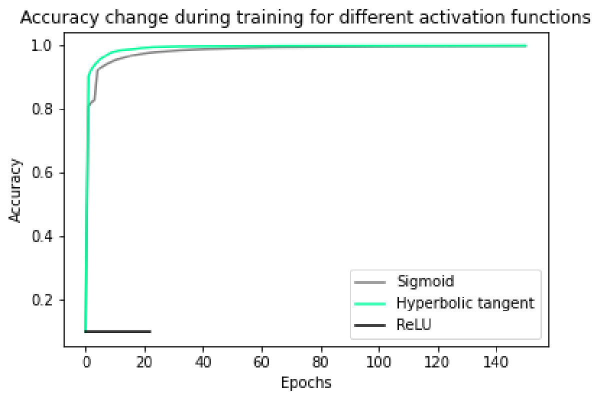


Figure 5: Perfomance of networks on MNIST dataset with different activation functions

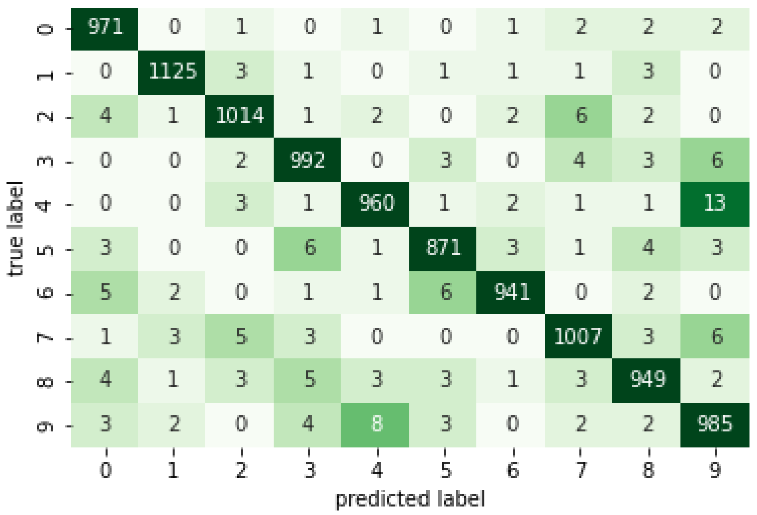
5.4 Best results and reflection

Table 5 presents the final hyper-parameters used in our network for the MNIST data set classification. They were chosen based on their results on training set, and not on the test set. Perhaps having an additional validation set would improve the performance of our network. Another limitation of our research is that we were only considering the effect of one hyper-parameter at the time - when comparing results for a specific hyper-parameter, the values of other hyper-parameters were kept constant and equal to values which were proven the best in the previous experiments. If given more time and computational power we could use a grid search or random search method instead, possibly obtaining a better solution. When training our network we used a specific initial random state, in order to achieve the reproducibility of the results. The accuracy of the final model on the training set was equal to 0.9991 and on the test set - to 0.9815.

Hyperparameter	**Value**
Architecture	1 hidden layer with 500 neurons
Learning rate	0.01
Momentum rate	0.9
Method of weights initialization	Xavier
Activation function	Tanh
Activation function in the output layer	Softmax
Loss function	Cross entropy

Table 5: Best hyperparameters chosen based on accuracy on training set

In addition we were wondering how well we are predicting each digit. The confusion matrix on the test set is presented in Figure 6. We have calculated that the algorithm predicts digit 1 with the highest accuracy of 0.9912 and digt 7 with the lowest accuracy of 0.9743. This does not mean that 7 was often misclassified as 1. In fact pairs of digits which were most often mixed up were 4 and 9, 2 and 7, 3 and 9.



true label \ predicted label	0	1	2	3	4	5	6	7	8	9
0	971	0	1	0	1	0	1	2	2	2
1	0	1125	3	1	0	1	1	1	3	0
2	4	1	1014	1	2	0	2	6	2	0
3	0	0	2	992	0	3	0	4	3	6
4	0	0	3	1	960	1	2	1	1	13
5	3	0	0	6	1	871	3	1	4	3
6	5	2	0	1	1	6	941	0	2	0
7	1	3	5	3	0	0	0	1007	3	6
8	4	1	3	5	3	3	1	3	949	2
9	3	2	0	4	8	3	0	2	2	985

Figure 6: Confusion matrix for chosen network