

## Introduction

This report describes what has been done to solve exercises in Home Assignment 3 and what were the results. To run code, execute 'main.m' file. Code repository: <https://gitlab.cs.ttu.ee/totahv/iti8565>

## Exercise 1. Neural Network [1]

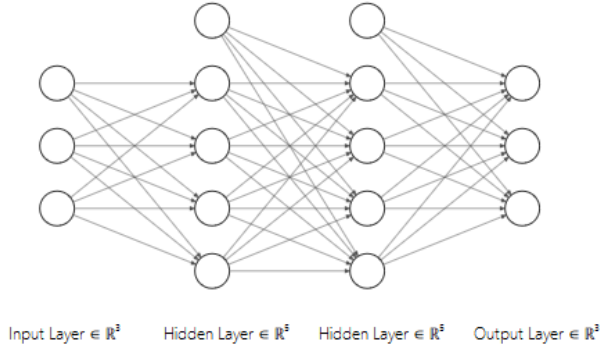


Figure 1 Structure of neural network.

Implemented neural network with two hidden layers, each layer supports N neurons. Used Sigmoid function in hidden layers and SoftMax function in the output layer to support multiclass classification. Used cross-entropy function for minimizing error, because partial derivatives were easier to find this way. Figure 1 illustrates structure of the neural network for 3D data and 3 output classes.

Formula (1) describes feedforward step of neural network. Moving from 'input layer' to '1<sup>st</sup> hidden layer', then to '2<sup>nd</sup> hidden layer' and 'output layer'.

$$\begin{cases} z_{h1} = X \cdot w_{h1} + b_{h1} \\ a_{h1} = \sigma(z_{h1}) \\ z_{h2} = a_{h1} \cdot w_{h2} + b_{h2} \\ a_{h2} = \sigma(z_{h2}) \\ z_o = a_{h2} \cdot w_o + b_o \\ a_o = \text{SoftMax}(z_o) \end{cases} \quad (1)$$

After that cost function and partial derivatives are needed in backpropagation.

Formula (2) describes cost function and partial derivatives for updating weights between output layer and 2<sup>nd</sup> hidden layer.

$$\begin{cases} C = -Y \cdot \log(a_o) \\ \frac{\partial C}{\partial w_o} = \frac{\partial C}{\partial a_o} \frac{\partial a_o}{\partial z_o} \frac{\partial z_o}{\partial w_o} = (a_o - Y) a_{h2} \end{cases} \quad (2)$$

Formula (3) describes cost function and partial derivatives for updating weights between 2<sup>nd</sup> hidden layer and 1<sup>st</sup> hidden layer.

$$\begin{cases} C = -a_o \cdot \log(a_{h2}) \\ \frac{\partial C}{\partial w_{h2}} = \frac{\partial C}{\partial a_{h2}} \frac{\partial a_{h2}}{\partial z_{h2}} \frac{\partial z_{h2}}{\partial w_{h2}} = \left( \frac{\partial C}{\partial z_o} \frac{\partial z_o}{\partial a_{h2}} \right) \frac{\partial a_{h2}}{\partial z_{h2}} \frac{\partial z_{h2}}{\partial w_{h2}} = \left[ \left( \frac{\partial C}{\partial a_o} \frac{\partial a_o}{\partial z_o} \right) \frac{\partial z_o}{\partial a_{h2}} \right] \frac{\partial a_{h2}}{\partial z_{h2}} \frac{\partial z_{h2}}{\partial w_{h2}} = (a_o - Y) w_o \sigma'(z_{h1}) a_{h2} \end{cases} \quad (3)$$

Formula (4) describes cost function and partial derivatives for updating weights between 1<sup>st</sup> hidden layer and input layer.

$$\begin{cases} C = -a_{h2} \cdot \log(a_{h1}) \\ \frac{\partial C}{\partial w_{h1}} = \frac{\partial C}{\partial a_{h1}} \frac{\partial a_{h1}}{\partial z_{h1}} \frac{\partial z_{h1}}{\partial w_{h1}} = \dots = (a_o - Y) w_o \sigma'(z_{h2}) w_{h2} \sigma'(z_{h1}) X \end{cases} \quad (4)$$

As a result, this neural network with 2 hidden layers supports arbitrary number of input neurons, arbitrary number of hidden neurons in each layer and any number of output neurons for multiclass classification.

## Exercise 2. Gradient boosting [2]

Implemented gradient boosting algorithm for regression model. Used MATLAB built-in function to create decision stumps. Figure 2 describes boosted decision stump model using gradient boosting after 50 epochs. Figure 3 describes error reduction when growing model. Created animation in gifs folder that animates growing model over 50 epochs.

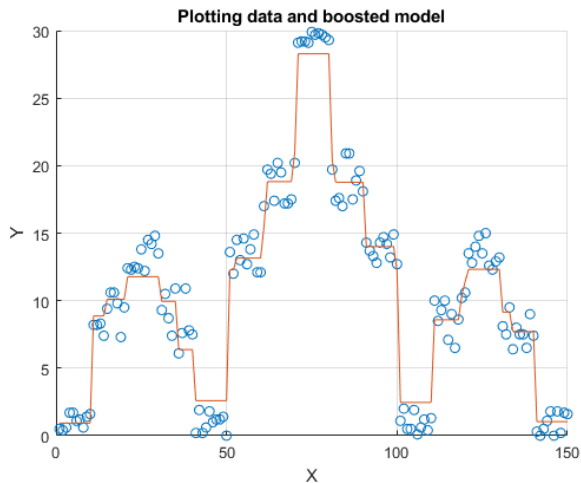


Figure 2 Boosted decision stumps using gradient boosting.

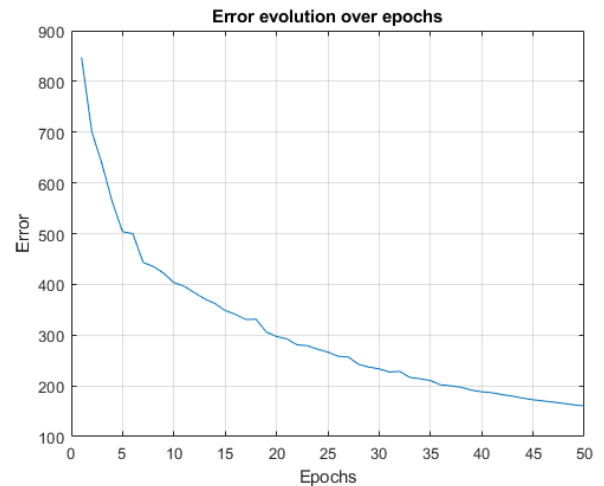


Figure 3 Error evolution over epochs.

## Conclusion

Implemented neural network for multiclass classification and gradient boosting algorithm for regression model.

## Used materials

[1] Creating a Neural Network from Scratch in Python, Usman Malik,

<https://stackabuse.com/creating-a-neural-network-from-scratch-in-python/>

[2] Gradient Boosting from scratch, Prince Grover,

<https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d>