Neural network specification

MNIST DATASET

ross erskine
[COMPANY NAME] | [COMPANY ADDRESS]

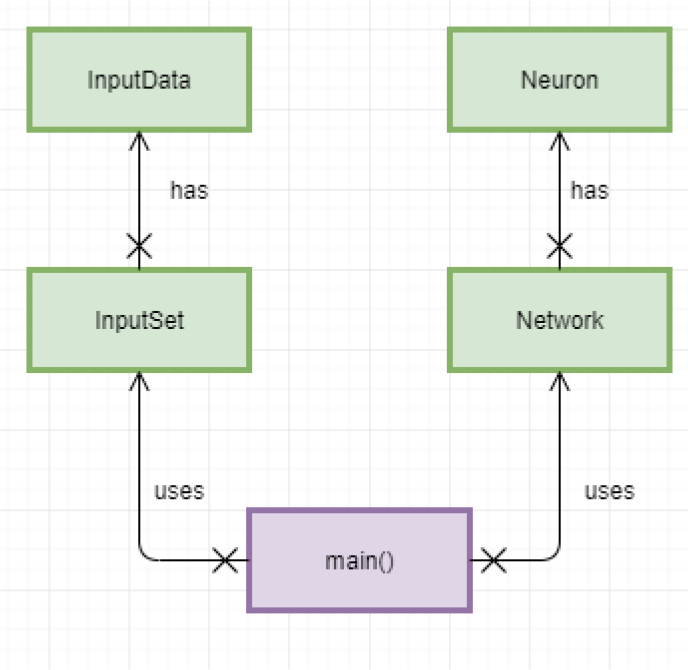
Contents

Artificial neural networks technical documentation

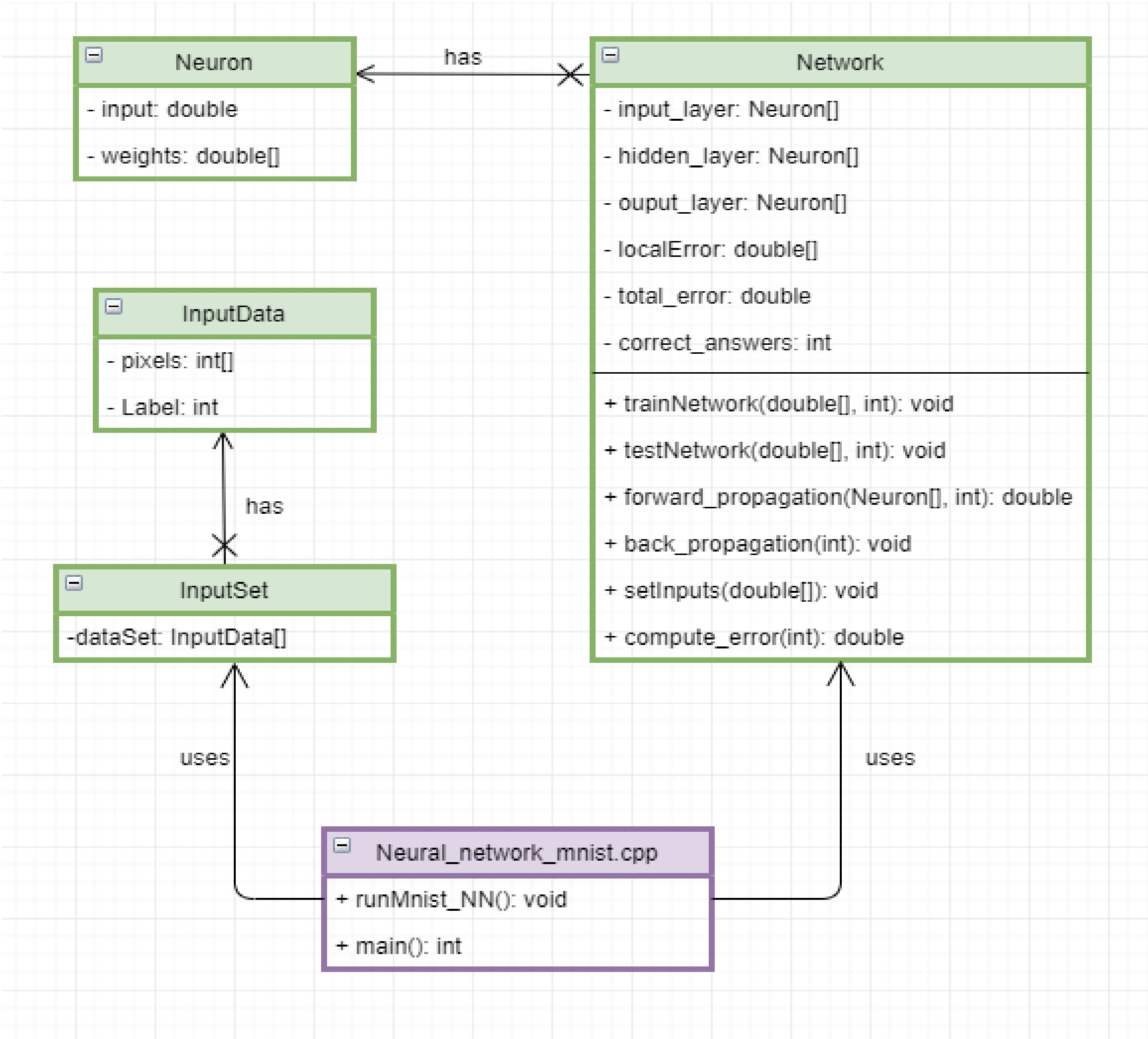
Technical documentation for the code *Artificial neural networks,* consisting of class diagram, program flow, sequence diagrams, design, data-dictionary test-table and results.

Class diagrams

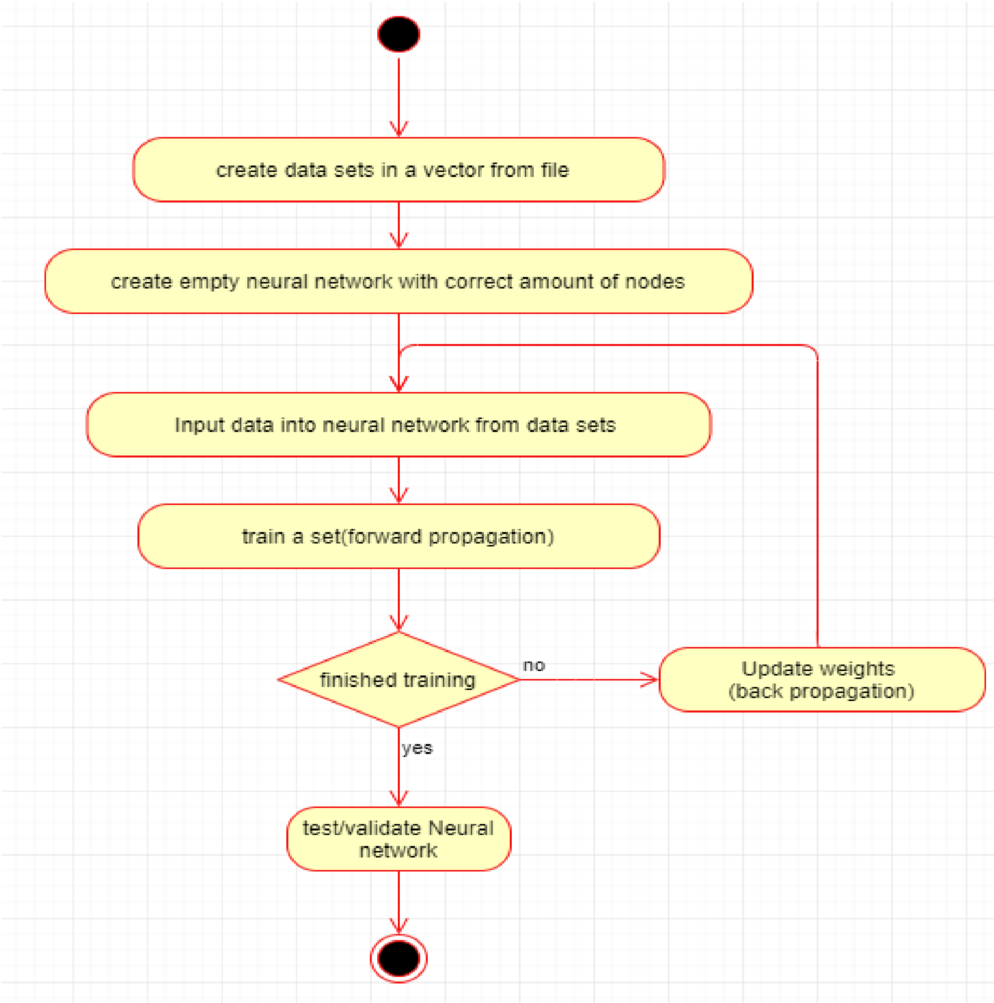
Conceptual Class diagrams



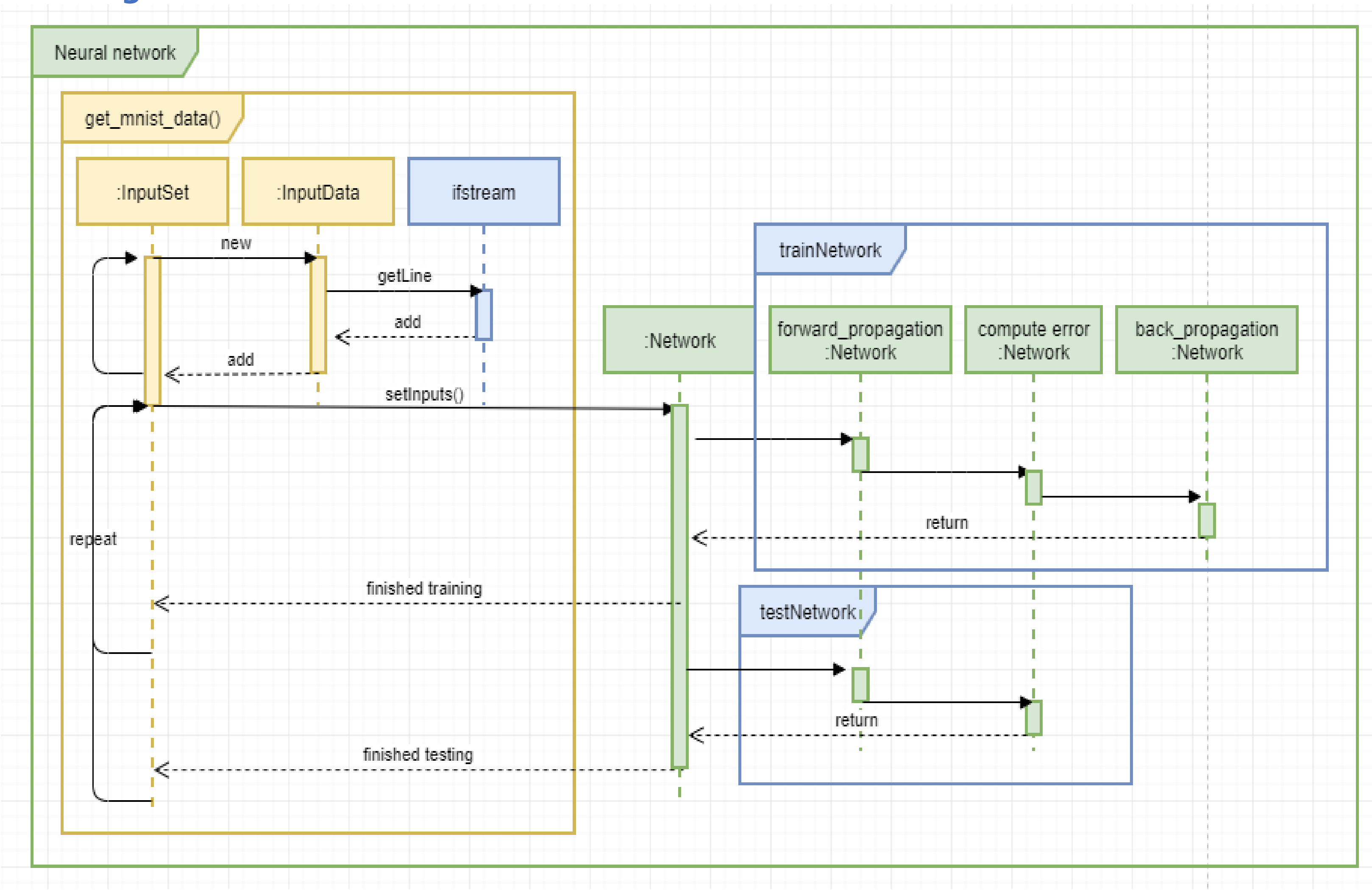
Logical Class diagrams



Program flow



Sequence diagram



Data dictionary

InputData is a single data set in a vector with attached label

Class: InputData				
Field	**Datatype**	**Validation**	**Example**	**description**
Label	Int	Private member	5,7,9	Target input
Pixels	Double[]	Private member	0,145,255	A single picture 28x28 , Pixels of data 784 of them

InputSet is a vector of InputData or collection of datasets.

Class: InputSet				
Field	**Datatype**	**Validation**	**Example**	**description**
dataset	InputData[]	Private member		A data set of 28x28 size pictures with attached labels

Class: Neuron				
Field	**Datatype**	**Validation**	**Example**	**description**
Input	double	Private member		One Input of one neuron
Weights	Double	Private member, Size equals next layer size.	30,15,10	Number of weights depends on next layer, all weights start with random number between 0 and 1.

Network class is a collection of layer Neuron's. the constructor can create the size network such as (3,2,1) and random size weights added to that input. Data to inputs are added at the start of train or test network.IL = input layer, HL = Hidden layer, OL = output layer, size_of = number of nodes in layer

Class: Network				
Field	**Datatype**	**Validation**	**Example**	**description**
Input_layer	Neuron[]	Private member	784	First layer of network size depends on amount of inputs example 28 x 28 = 784 input neurons
Hidden_layer	Neuron[]	Private member	100,30,15	Hidden layer is layer between input and output layer, can bea any size.
Ouput_layer	Neuron[]	Private member, size 10	(0,0,0,0,0,0,0,1,0,0).	Output layer should be size of 10, to indicate which number being guessed example: if label is 7 output would be trying to give this output (0,0,0,0,0,0,0,1,0,0).
localError	Double[]	Private member, size 10	(1,1,1,1,1,1,1,0,1,1)	Error given by for each output layer with loss function $(t_i - z_i)^2$ t_i = target output z_i = actual output
Total_error	double	Private member		function mean Squared error: MSE = $\frac{1}{n}\sum_{i=1}^{n}(t_i - z_i)^2$
Correct_answers	Int	Private member	1,10	Used in validation phase to see how many correct guesses the trained network makes.

Function	**Return type**	**Parameters**	**description**
trainNetwork	Void	Double[], int	trainNetwork consists of first initial data added to inputs of input layer, forward propagation sends the data to each layer, the total and local error is calculated, information is sent to csv file then finally ends with back propagation, needs to be in loop for multiple sets
testNetwork	Void	Double[], int	test network is same as train except uses final guess and does not upload to file or back propagation

Forward_propagation	Double	Neuron[], int	forward propagation uses summation and activation functions then sets input of next layer
Back_propagation	Void	Int	starts by updating hidden to output layer by calculating the individual weight partial derivative using the chain rule, then the same is done updating weights on input to hidden layers weights
setInputs	Void	Double[]	sets first data into network
Compute_error	Double	Int	function mean Squared error: MSE = $\frac{1}{n}\sum_{i=1}^{n}\left(t_i - z_i\right)^2$
Compute_final_error	Double		creates final error from output of output layer then averaged and places local errors in local_error.
Final_guess	Int		final guess, number closest to 1 wins.
randDouble	Double	Double,double	"random number engine is a function that generates uniformly distributed sequence of integer values. Distribution is a function that generates a sequence of values according to a mathematical formaula given a sequence of value from an engine." Stroustrup (2014: p914)
randInt	Int	Int, int	
Summation_operator	Double	Neuron[], int	Summation operator the sum of all inputs multiplied by weights that corresponds to the next layers input. summation = bias + $\sum_{i=1}^{n} x_i \theta_{ij}$ x_i = each input in layer w_{ij}= each weight that corresponds to each input Bias = extra input that; input is set to 1
Activation	Double	Double	Activation function squashes the number and signifies how much a neuron has fired. Next neurons input = $f(x)=\frac{1}{1+e^{-x}}$ X = summation e = Euler’s constant = 2.71828
Is_target_output	Bool	Int, int	represents the target output if the label is the same as the iteration it

			will send back one. example label 4 (0,0,0,0,1,0,0,0,0,0)
NN_data	Void	Double, int	uploads error and label to csv file

Calculating Neural networks gradient

Based on formulas from Taylor (2017).

Math notation and description

Table 1: artificial neural network math notation and description

Notation	description
∂E	partial derivative of total error
$\partial \theta_i$	partial derivative of a specific weight
∂Z_i	partial derivative of output from specific Neuron in output layer
∂Z	partial derivative of sum of all outputs
∂A_j	partial derivative of output from specific neuron in input layer
∂B_j	partial derivative of output in specific neuron in hidden layer
∂B	partial derivative of sum of all inputs to hidden layer
δ	Delta notation to represent part of an equation

Always start with hidden to output layer.

Between hidden and output:

$$\frac{\partial E}{\partial \theta_i} = \frac{\dfrac{\dfrac{\partial E}{\partial Z_i} * \partial Z_i}{\partial Z} * \partial Z,}{\partial \theta_i}$$

Partial derivative broken down using

$$\frac{\partial E}{\partial Z_i} = (z_i - t_i),\ \frac{\partial Z_i}{\partial Z} = z_i(1 - z_i),\ \frac{\partial Z}{\partial \theta_i} = \left(B_i \theta_i\right),$$

$$\frac{\partial E}{\partial \theta_i} = \left(z_i - t_i\right) z_i \left(1 - z_i\right) ¿),$$

Delta_z = $\delta_z = \left(z_i - t_i\right) z_i(1 - z_i)$,

$$\frac{\partial E}{\partial \theta_i} = \delta_z ¿)$$

Equation 1: artificial neural network between hidden and output partial derivative.

Output Bias weights

$$\frac{\partial E}{\partial B\theta_i} = \delta_z$$

Equation 2: : artificial neural network output bias weights partial derivative

```
for (int i = 0; i != mHidden_layer.size(); i++)
{
    // == update weights in hiddenlayer == //
    for (int j = 0; j != mOutput_layer.size(); j++)
    {
        delta_z = (mOutput_layer[j].getInput() - is_target_output(label, j))
            * (mOutput_layer[j].getInput() * (1 - mOutput_layer[j].getInput()));

        partial_derivative_output_layer = delta_z
            * (mHidden_layer[i].getInput() * mHidden_layer[i].getWeights()[j]);

        mHidden_layer[i].setWeight(
            (mHidden_layer[i].getWeights()[j] - learningConstant * partial_derivative_output_layer), j);

        // ===== avg sum of delta z ====== //
        sumof_delta_z += delta_z * mHidden_layer[i].getWeights()[j];
    }
}
sumof_delta_z /= mHidden_layer[0].getWeightSize();
```

Between input and hidden:

$$\frac{\partial E}{\partial \theta_j} = \frac{\dfrac{\dfrac{\dfrac{\partial E}{\partial Z_i} * \partial Z_i}{\partial B_j} * \partial B_j}{\partial B} * \partial B'}{\partial \theta_j}$$

$$\frac{\partial E}{\partial Z_i} = \delta_z, \quad \frac{\partial Z_i}{\partial A_j} = \theta_i, \quad \frac{\partial A_j}{\partial B_j} = B_j(1-B_j), \quad \frac{\partial B}{\partial \theta_j} = (A_j \theta_j),$$

$$\frac{\partial E}{\partial \theta_j} = \left(\sum_z \delta_z \theta_i\right) B_j (1-B_j) A_j,$$

Delta_b = $\delta_B = (\delta_Z \theta_i) B_j (1-B_j),$

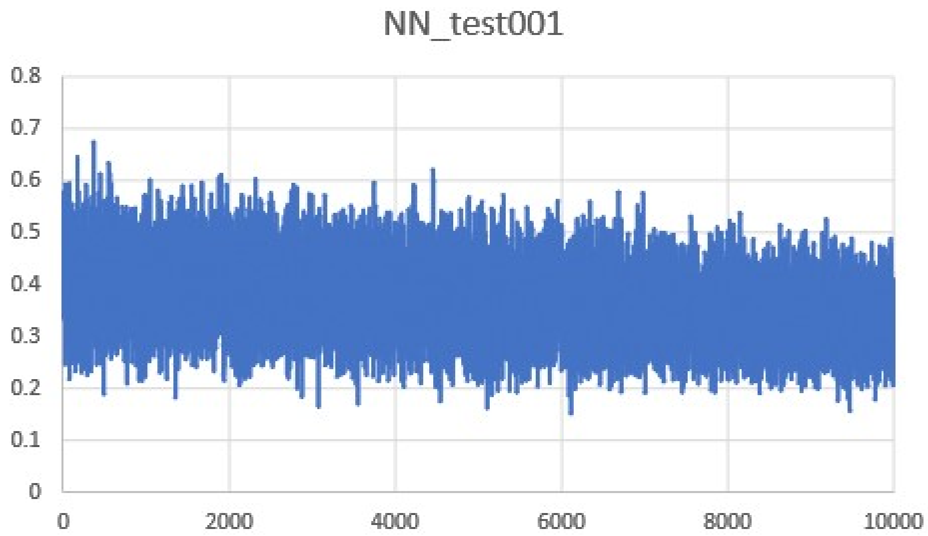
$$\frac{\partial E}{\partial \theta_j} = \delta_B \left(A_j \theta_j \right)$$

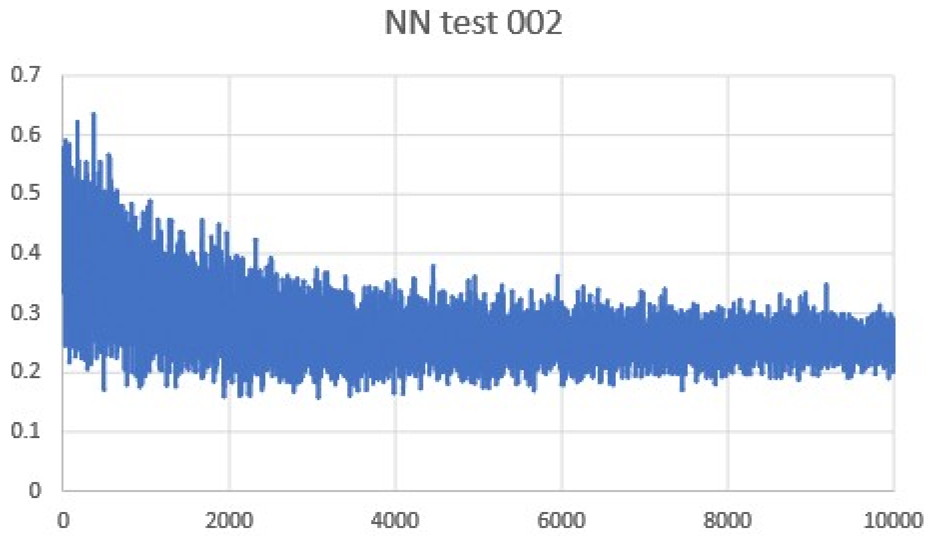
```
// ==  loops through inputlayer Neurons == //
for (int i = 0; i != mInput_layer.size(); i++)
{

    // == update weights in inputlayer == //
    for (int j = 0; j != mHidden_layer.size(); j++)
    {
        delta_b = sumof_delta_z * mInput_layer[i].getWeights()[j]
            * mHidden_layer[j].getInput() * (1 - mHidden_layer[j].getInput());

        partial_derivative_hidden_layer = delta_b
            * (mInput_layer[i].getInput() * mInput_layer[i].getWeights()[j]);


        mInput_layer[i].setWeight(
            (mInput_layer[i].getWeights()[j] - learningConstant * partial_derivative_hidden_layer), j);

    }
}
```

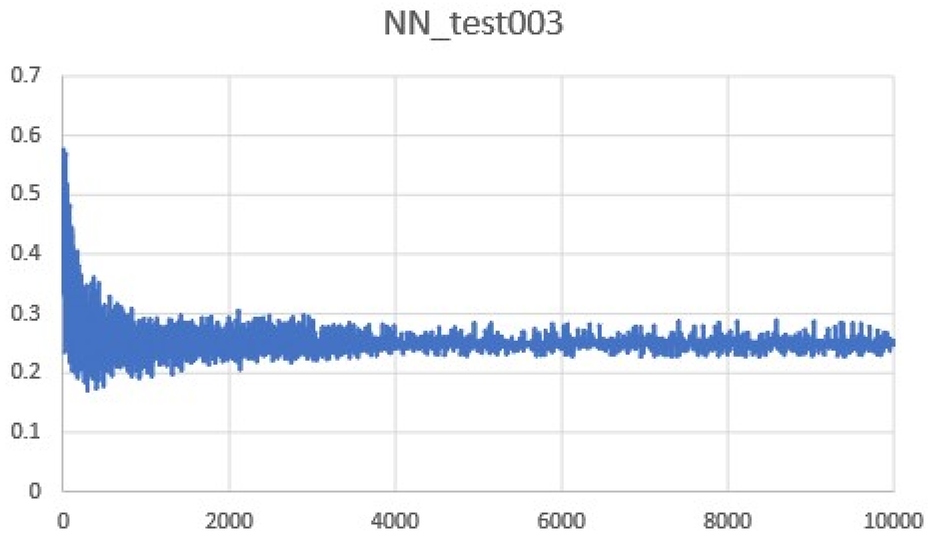
Tests

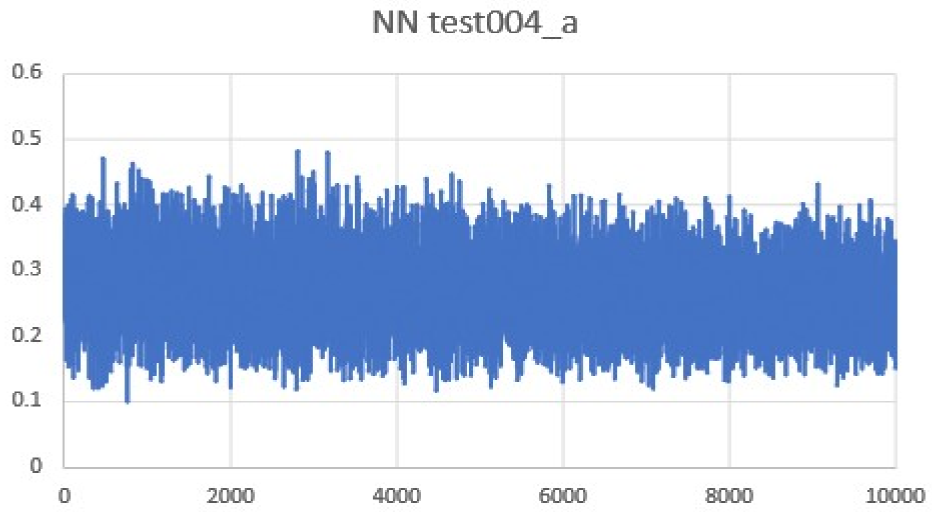
Table 2: Artificial neural network test-table

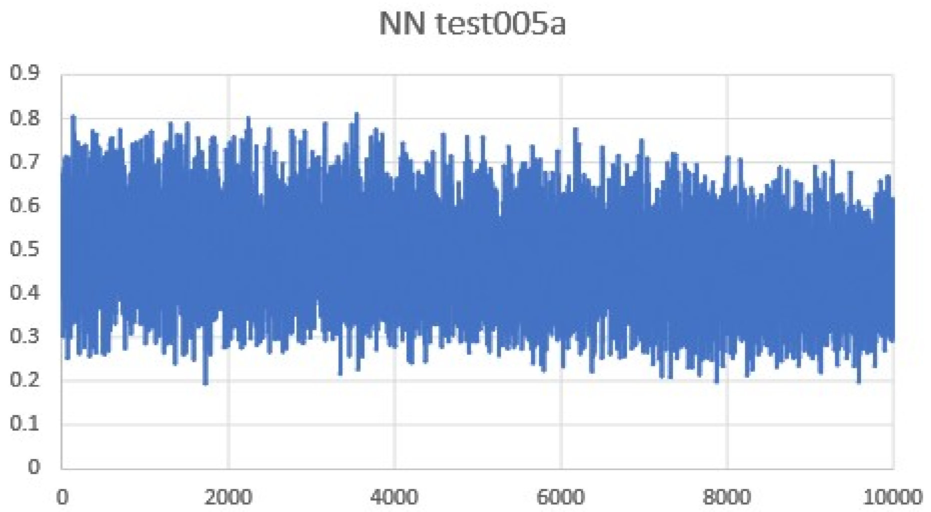
Test:	Learning rate	Hidden layer neurons	Epochs	Validation	filename	results
01	0.001	30	10000	131/1000	NN_data_file_001	Figure 30
02	0.01	30	10000	120/1000	NN_data_file_002	Figure 31
03	0.1	30	10000	82/1000	NN_data_file_003	Figure 32
04	0.001	15	10000	79/1000	NN_data_file_004	Figure 33
05	0.001	60	10000	78/1000	NN_data_file_005	Figure 34
06	0.001	100	10000	101/1000	NN_data_file_006	Figure 35

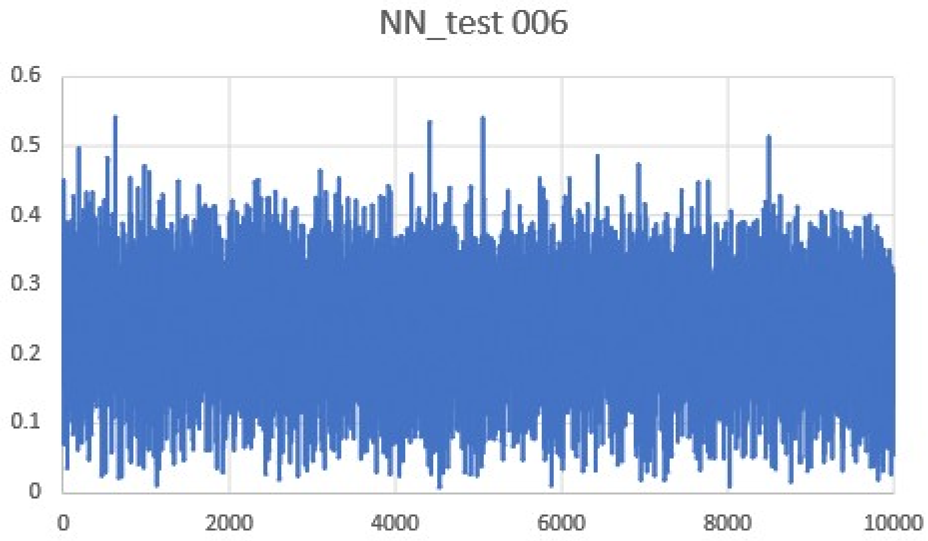



NN test 002
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2000
4000
6000
8000
10000


NN_test003
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2000
4000
6000
8000
10000


NN test004_a
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2000
4000
6000
8000
10000


NN test005a
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2000
4000
6000
8000
10000


NN_test 006
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2000
4000
6000
8000
10000