Computer Programming:

Project Report

Deep neural network for digit recognition

Author: Rafał Potempa
Supervisor: Anna Gorawska, MSc.
Date: 20 February 2019

Table of content

1. Task description

Neural networks are mathematical models performing similarly to living organisms' neurons. Neurons are organized in layers which are connected together between each neuron. The nonlinearity is applied with sigmoid activation function (1.1).

$$f(x) = \frac{1}{1+e^{-x}} \tag{1.1}$$

$$f'(x) = \mathrm{f(x)}\big(1 - \mathrm{f(x)}\big) \tag{1.1}'$$

1.1. Feed forward

The forward feed process is realized by following matrix operations:

$$S_i = \mathrm{Z_{i-1}} \cdot \mathrm{W_{i-1 \to i}} \tag{1.2}$$

where $\mathrm{S_i}$ is input of current layer, $\mathrm{Z_{i-1}}$ is output of previous later. In case of input layer Z is replaced with X (input vector) (1.3).

$$S_i = \mathrm{X} \cdot \mathrm{W_{i-1 \to i}} \tag{1.3}$$

Output of each layer is calculated with use of activation function.

$$Z_i = \mathrm{f(S_i)} \tag{1.4}$$

1.2. Backpropagation

The backpropagation algorithm is achieved by calculating error and propagating it backward through the network.

$$D_i = \mathrm{F'_i} \odot (\mathrm{D_{i+1}} \cdot \mathrm{W_{i \to i+1}}) \tag{1.5}$$

where $\odot$ is element-wise multiplication and $\mathrm{F'_i} = f'(S_i)$ (1.1)'.

Having calculated the influence of each weight on final output, the update of weights can be done.

$$\Delta W_{i \to i+1} = -\eta \cdot \mathrm{Z_i^T} \cdot \mathrm{D_{i+1}} \tag{1.6}$$

where η is learning rate.

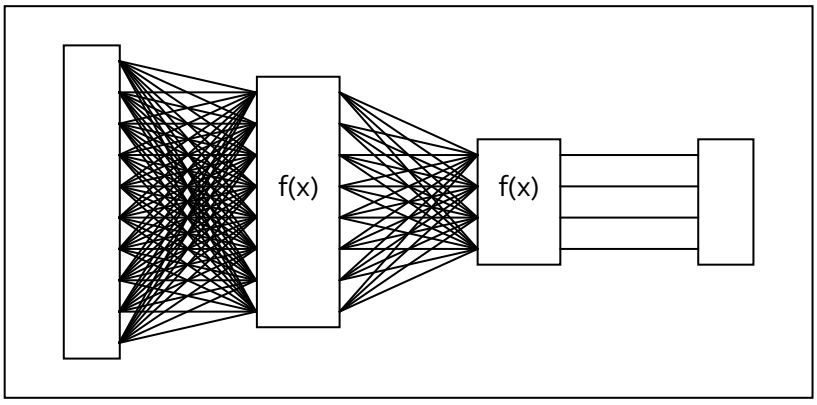


Figure 1 Neural network scheme

2. Solution analysis

The program uses **vectors** of doubles as representation of all matrices.

Neural network consists of Layers , that can be tied together as dynamic list with two directions. Assembly of network begins with output layer and previous layers can be attached to one another.

Solution uses simple interface to let user construct the neural network, and to see the progress and efficiency of learning.

Model can be saved to or load from file.

3. Internal specification

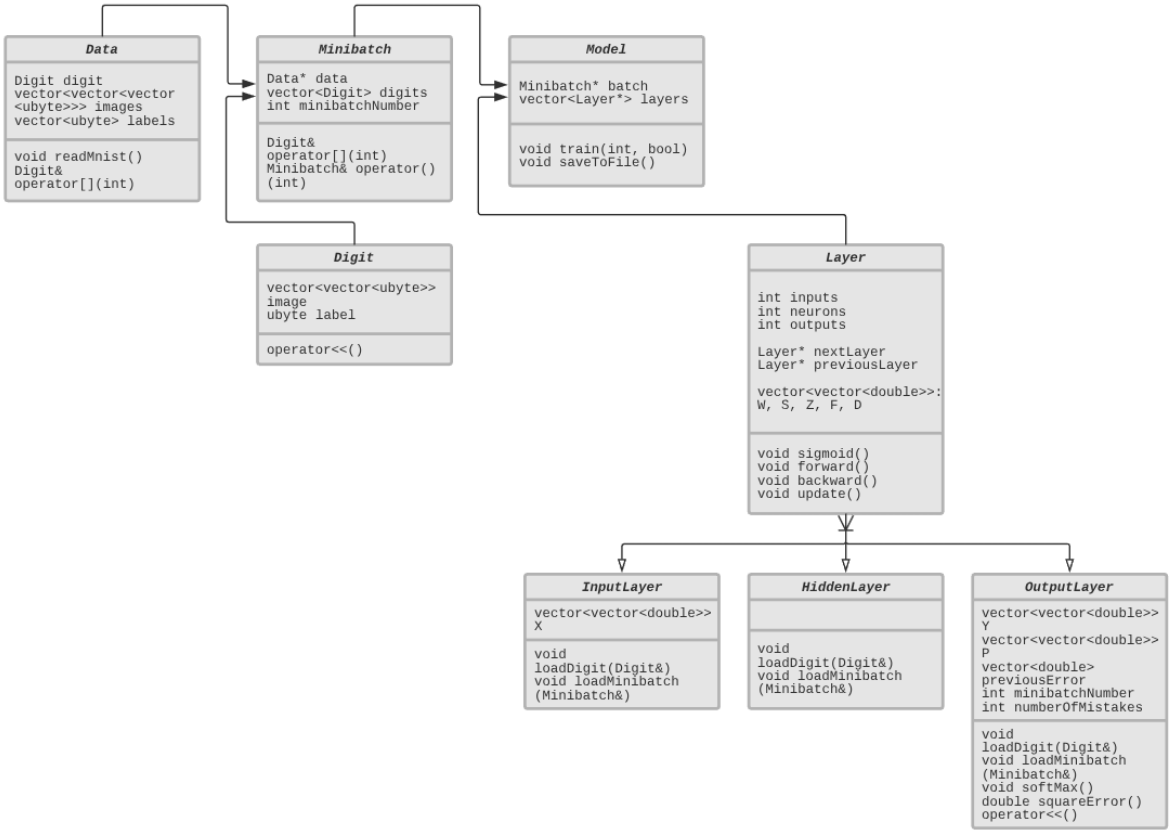


Figure 2 Classes scheme

3.1. class Model

Object of this class is fully operable neural network with layers and methods required for its training. Can be constructed with use of vector of layer sizes (entered by user) or from file.

`void train()`

Method loads minibatch to input layer and iterates through all layers in neural network for every minibatch and for given number of epochs – forward feed, backpropagation – and when all minibatches in epoch are finished. It calculates mean weight update and updates.

```
for(epochs)
{
    for(minibatches)
    {
        inputLayer.loadMinibatch();

        for(layers)
            layer.forward();

        for(layers)
            layer.backward();
    }
    for(layers)
        layer.update();
}
```

void saveToFile()

Saves neural network weights to .nn file described by layer sizes.

```
path = createPathName();
file.open(path);

for(layers)
{
    file << layer.w;
}
```

3.2. class Layer

This is base class for specific layers that assembly neural network.

void sigmoid()

Activation function for layers' neurons, that calculates outputs Z and derivatives F for every layer with sigmoid neurons (1.1), (1.1)'.

3.3. class InputLayer : public Layer

Objects of this class are input layers with 784 neurons by default .

void forward()

Calculates input of layer (1.3). When done activation function for layer is called, which calculates sigmoid output (1.4) and its derivative.

void backward()

Calculates error propagated from following layer (1.5).

void update()

Calculates value of update and updates the layer's weights (1.6).

3.4. class HiddenLayer : public Layer

Objects of this class are hidden(deep) layers neural network.

void forward()

Calculates input of layer (1.2). When done activation function for layer is called, which calculates sigmoid output (1.4) and its derivative.

void backward()

Calculates error propagated from following layer (1.5).

void update()

Calculates value of update and updates the layer's weights (1.6).

3.5. class OutputLayer : public Layer

Objects of this class are output layers with additional softmax classification. Cost function:

$$C = \frac{1}{2}(\mathrm{Z_{out}} - \mathrm{Y})^2 \tag{3.1}$$

$$\frac{\partial C}{\partial Z_{out}} = \mathrm{Z_{out}} - \mathrm{Y} \tag{3.1}'$$

void forward()

Calculates input of layer (1.2). When done activation function for layer is called, which calculates sigmoid output (1.4) and its derivative.

void backward()

Calculates error propagated from following layer (1.5).

void update()

Calculates value of update and updates the layer's weights (1.6).

void softMax()

Calculates the probability of each output with use of softmax function:

$$p_i = \frac{e^{z_i}}{\sum_i e^{z_i}} \tag{3.2}$$

3.6. class Digit

Objects of this class carry complete information about single digit from MNIST database (image and label) for further assessment.

3.7. class Data

Objects of this class hold information from MNIST database with ability to easy access.

void readMnist()

Reads MNIST from files containing labels and images to the vectors of **ubyte** for better performance.

3.8. class Minibatch

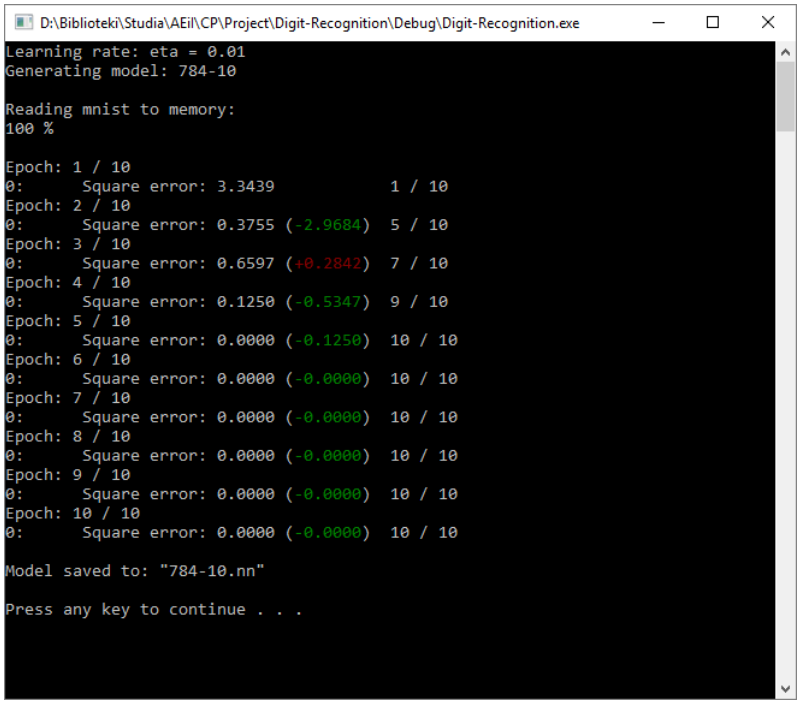
Objects of this class carries minibatch vector for computations.

4. External specification

This program is designed to work with command line prompt. There are two methods of assembly of neural network structure.

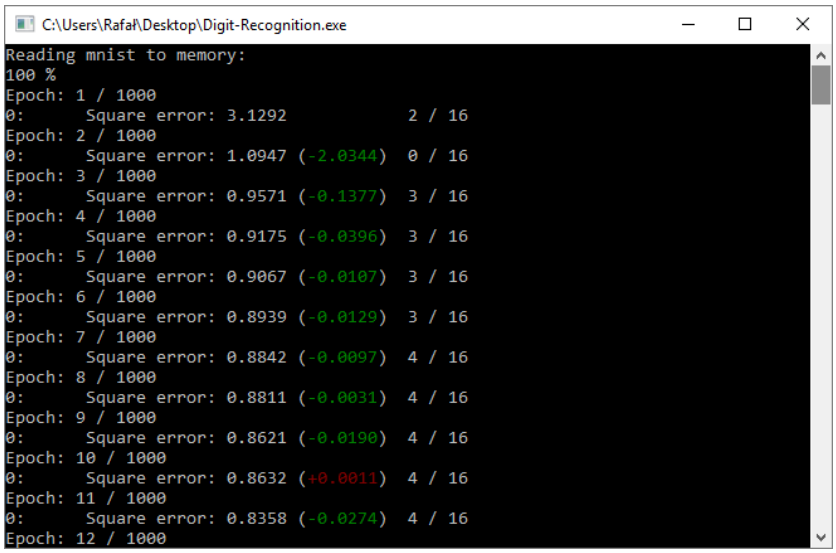
a) load network from file,
b) assembly of network by console through layout of layers.

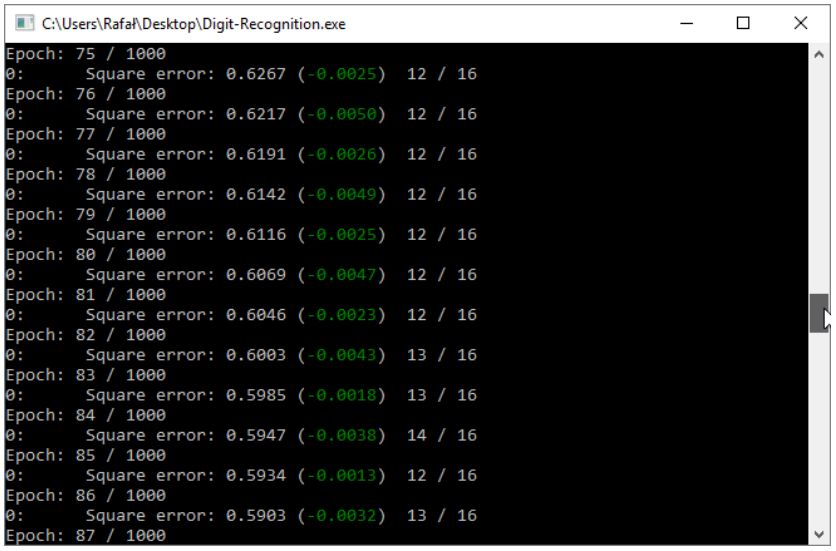
After training the model is saved to .nn file.

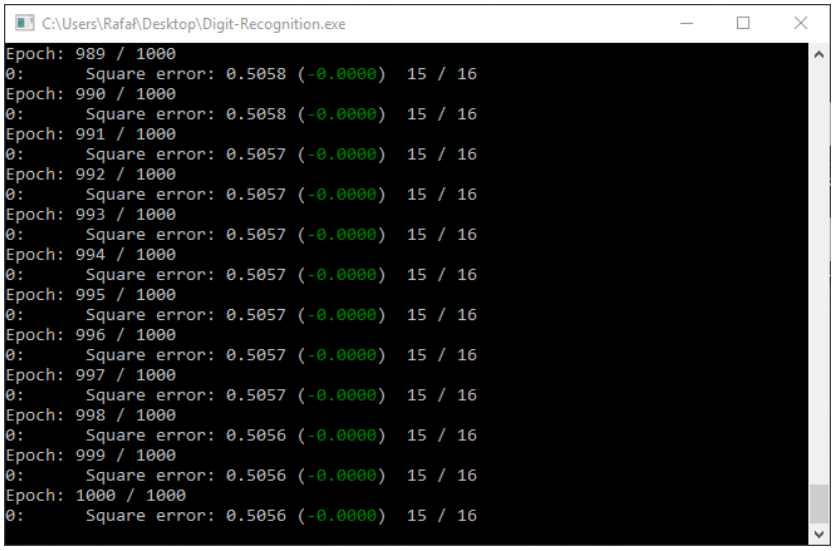


5. Testing

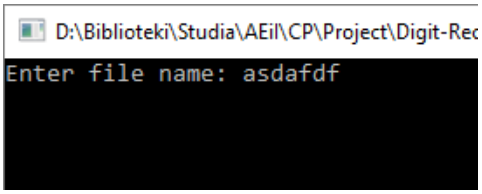
5.1. Ordinary conditions

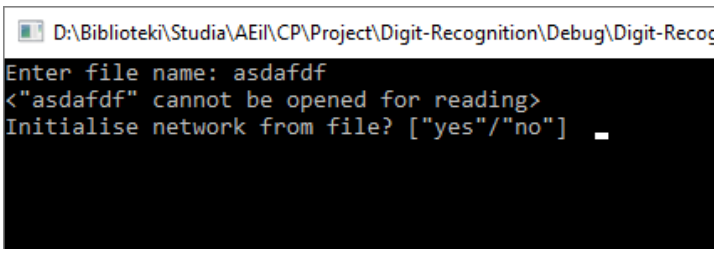






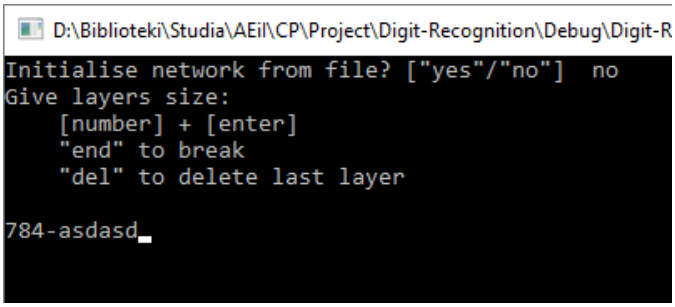
5.2. File to load network does not exist

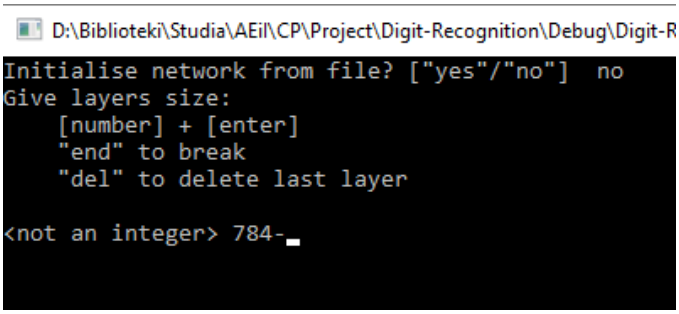




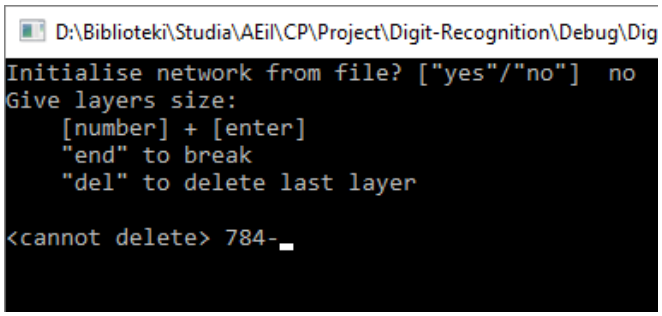
The initialization runs again.

5.3. Wrong input for layer size

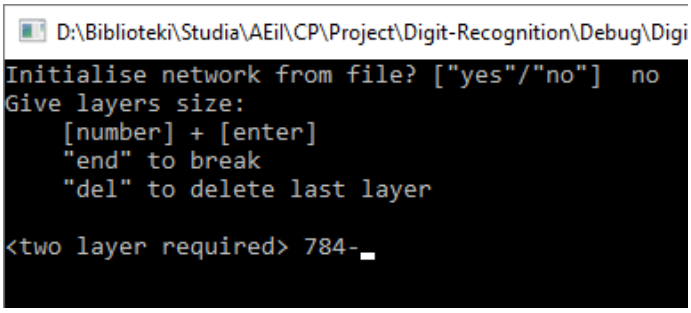




5.4. Trying to delete first layer



5.5. Trying to end assembly on one layer



6. Testing of algorithm

Calculations of one iteration of simple neural network algorithm.

Input: Output:

$$X := \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \qquad Y := \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Input - Layer 1

$$Win1 := \begin{pmatrix} 1 & 0.6 & 0.6 \\ 0.6 & 0.6 & 0.6 \\ 0.6 & 0.6 & 0.6 \\ 0.6 & 0.6 & 0.6 \end{pmatrix}$$

Layer 1:

$$S1 := X^T \cdot Win1 = (2.2 \quad 1.8 \quad 1.8)$$

$$Z1 := \overrightarrow{\frac{1}{1+e^{-S1}}} = (0.9 \quad 0.858 \quad 0.858) \qquad F1 := \left(\overrightarrow{\frac{d}{dS1}\frac{1}{1+e^{-S1}}}\right)^T = \begin{pmatrix} 0.09 \\ 0.122 \\ 0.122 \end{pmatrix}$$

Layer 1 - Layer 2

$$W12 := \begin{pmatrix} 1 & 0.6 & 0.6 & 0.6 & 0.6 \\ 0.6 & 0.6 & 0.6 & 0.6 & 0.6 \\ 0.6 & 0.6 & 0.6 & 0.6 & 0.6 \end{pmatrix}$$

Layer 2:

$$S2 := Z1 \cdot W12 = (1.93 \quad 1.57 \quad 1.57 \quad 1.57 \quad 1.57)$$

$$Z2 := \overrightarrow{\frac{1}{1+e^{-S2}}} = (0.873 \quad 0.828 \quad 0.828 \quad 0.828 \quad 0.828) \qquad F2 := \left(\overrightarrow{\frac{d}{dS2}\frac{1}{1+e^{-S2}}}\right)^T = \begin{pmatrix} 0.111 \\ 0.143 \\ 0.143 \\ 0.143 \\ 0.143 \end{pmatrix}$$

Layer 2 - Output

$$W2out := \begin{pmatrix} 1 & 0.6 \\ 0.6 & 0.6 \\ 0.6 & 0.6 \\ 0.6 & 0.6 \\ 0.6 & 0.6 \end{pmatrix}$$

Output

$$S3 := Z2 \cdot W2out = (2.86 \quad 2.511)$$

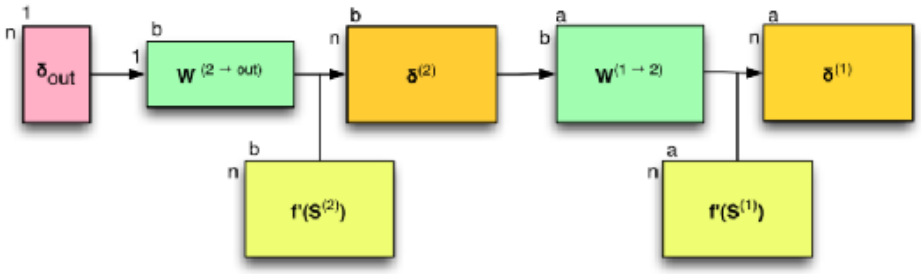
$$Z3 := \overrightarrow{\frac{1}{1 + e^{-S3}}} = (0.946 \quad 0.925) \qquad F3 := \left(\overrightarrow{\frac{d}{dS3}\frac{1}{1 + e^{-S3}}}\right)^T = \begin{pmatrix} 0.051 \\ 0.069 \end{pmatrix}$$

Softmax:

$$S4 := S3 = (2.86 \quad 2.511)$$

$$Z4 := \left(\frac{e^{Z3_{0,0}}}{e^{Z3_{0,0}} + e^{Z3_{0,1}}} \quad \frac{e^{Z3_{0,1}}}{e^{Z3_{0,0}} + e^{Z3_{0,1}}}\right) = (0.505 \quad 0.495)$$

Error propagation:



$$D4 := \left(Z4 - Y^T\right)^T = \begin{pmatrix} 0.505 \\ -0.505 \end{pmatrix}$$

$$D3 := \overrightarrow{(F3 \cdot D4)} = \begin{pmatrix} 0.026 \\ -0.035 \end{pmatrix}$$

$$\_D2 := W2out \cdot D3 = \begin{pmatrix} 0.005 \\ -0.006 \\ -0.006 \\ -0.006 \\ -0.006 \end{pmatrix}$$

$$D2 := \overrightarrow{[F2 \cdot (\_D2)]} = \begin{pmatrix} 0.0005 \\ -0.0008 \\ -0.0008 \\ -0.0008 \\ -0.0008 \end{pmatrix}$$

$$\_D1 := W12 \cdot D2 = \begin{pmatrix} -0.001 \\ -0.002 \\ -0.002 \end{pmatrix}$$

$$D1 := \overrightarrow{(F1 \cdot \_D1)} = \begin{pmatrix} -1.219 \times 10^{-4} \\ -1.913 \times 10^{-4} \\ -1.913 \times 10^{-4} \end{pmatrix}$$

Learning rate: $\eta := 10$

Weights update:

$$\Delta Win1 := -\eta \cdot \left(D1 \cdot X^T\right)^T = \begin{pmatrix} 0.001 & 0.002 & 0.002 \\ 0.001 & 0.002 & 0.002 \\ 0 & 0 & 0 \\ 0.001 & 0.002 & 0.002 \end{pmatrix}$$

$$Win1 := Win1 + \Delta Win1 = \begin{pmatrix} 1.001 & 0.602 & 0.602 \\ 0.601 & 0.602 & 0.602 \\ 0.6 & 0.6 & 0.6 \\ 0.601 & 0.602 & 0.602 \end{pmatrix}$$

$$\Delta W12 := -\eta \cdot (D2 \cdot Z1)^T = \begin{pmatrix} -0.005 & 0.007 & 0.007 & 0.007 & 0.007 \\ -0.005 & 0.007 & 0.007 & 0.007 & 0.007 \\ -0.005 & 0.007 & 0.007 & 0.007 & 0.007 \end{pmatrix}$$

$$W12 := W12 + \Delta W12 = \begin{pmatrix} 0.995 & 0.607 & 0.607 & 0.607 & 0.607 \\ 0.595 & 0.607 & 0.607 & 0.607 & 0.607 \\ 0.595 & 0.607 & 0.607 & 0.607 & 0.607 \end{pmatrix}$$

$$\Delta W2out := -\eta \cdot (D3 \cdot Z2)^T = \begin{pmatrix} -0.226 & 0.307 \\ -0.214 & 0.291 \\ -0.214 & 0.291 \\ -0.214 & 0.291 \\ -0.214 & 0.291 \end{pmatrix}$$

$$W2out := W2out + \Delta W2out = \begin{pmatrix} 0.774 & 0.907 \\ 0.386 & 0.891 \\ 0.386 & 0.891 \\ 0.386 & 0.891 \\ 0.386 & 0.891 \end{pmatrix}$$

7. Conclusions

Algorithm testing yields the same results for program and mathematical calculations – model is correct.

Neural network training depends on its size, but given enough time weights can be adjusted to optimal level. Application of parallel, multithread calculations would provide better performance.

Nevertheless the algorithm is able to perform gradient descent in proper direction. The network can be saved to or read from a file, also assembled from console view.