

*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 .....                      | 2 |
| 1.1. Feed forward .....                        | 2 |
| 1.2. Backpropagation .....                     | 2 |
| 2. Solution analysis .....                     | 3 |
| 3. Internal specification .....                | 3 |
| 3.1. class Model .....                         | 3 |
| 3.2. class Layer .....                         | 4 |
| 3.3. class InputLayer : public Layer .....     | 5 |
| 3.4. class HiddenLayer : public Layer .....    | 5 |
| 3.5. class OutputLayer : public Layer .....    | 5 |
| 3.6. class Digit .....                         | 6 |
| 3.7. class Data .....                          | 6 |
| 3.8. class Minibatch .....                     | 6 |
| 4. External specification .....                | 6 |
| 5. Testing .....                               | 7 |
| 5.1. Ordinary conditions .....                 | 7 |
| 5.2. File to load network does not exist ..... | 8 |
| 5.3. Wrong input for layer size .....          | 8 |
| 5.4. Trying to delete first layer .....        | 8 |
| 5.5. Trying to end assembly on one layer ..... | 9 |
| 6. Conclusions .....                           | 9 |

## 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).

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

$$f'(x) = f(x)(1 - f(x)) \quad (1.1)'$$

### 1.1. Feed forward

The forward feed process is realized by following matrix operations:

$$S_i = Z_{i-1} \cdot W_{i-1 \rightarrow i} \quad (1.2)$$

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

$$S_i = X \cdot W_{i-1 \rightarrow i} \quad (1.3)$$

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

$$Z_i = f(S_i) \quad (1.4)$$

### 1.2. Backpropagation

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

$$D_i = F'_i \odot (D_{i+1} \cdot W_{i \rightarrow i+1}) \quad (1.5)$$

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

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

$$\Delta W_{i \rightarrow i+1} = -\eta \cdot Z_i^T \cdot D_{i+1} \quad (1.6)$$

where  $\eta$  is learning rate.

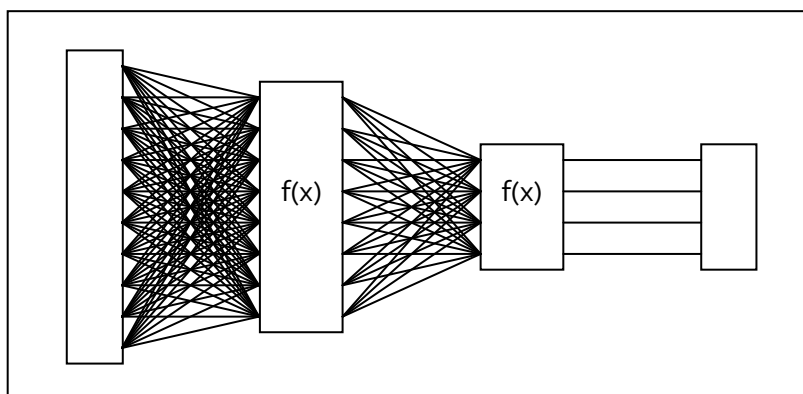


Figure 1 Neural network scheme

## 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

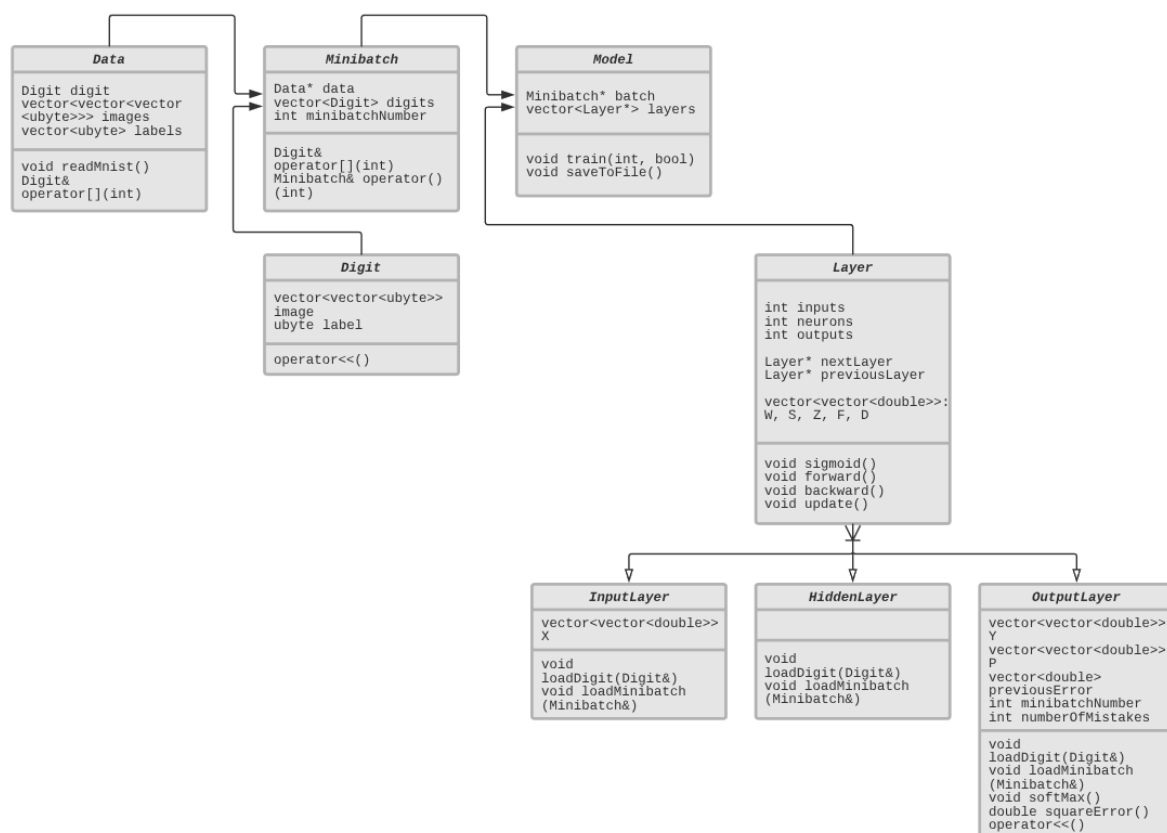


Figure 2 Classes scheme

### 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:

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

$$\frac{\partial C}{\partial Z_{out}} = Z_{out} - Y \quad (3.1)'$$

#### `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:

$$p_i = \frac{e^{z_i}}{\sum_i e^{z_i}} \quad (3.2)$$

**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.

```

D:\Biblioteki\Studia\AEII\CP\Project\Digit-Recognition\Debug\Digit-Recognition.exe
Learning rate: eta = 0.01
Generating model: 784-10

Reading mnist to memory:
100 %

Epoch: 1 / 10
0:   Square error: 3.3439          1 / 10
Epoch: 2 / 10
0:   Square error: 0.3755 (-2.9684) 5 / 10
Epoch: 3 / 10
0:   Square error: 0.6597 (+0.2842) 7 / 10
Epoch: 4 / 10
0:   Square error: 0.1250 (-0.5347) 9 / 10
Epoch: 5 / 10
0:   Square error: 0.0000 (-0.1250) 10 / 10
Epoch: 6 / 10
0:   Square error: 0.0000 (-0.0000) 10 / 10
Epoch: 7 / 10
0:   Square error: 0.0000 (-0.0000) 10 / 10
Epoch: 8 / 10
0:   Square error: 0.0000 (-0.0000) 10 / 10
Epoch: 9 / 10
0:   Square error: 0.0000 (-0.0000) 10 / 10
Epoch: 10 / 10
0:   Square error: 0.0000 (-0.0000) 10 / 10

Model saved to: "784-10.nn"

Press any key to continue . . .

```

## 5. Testing

### 5.1. Ordinary conditions

```

C:\Users\Rafał\Desktop\Digit-Recognition.exe
Reading mnist to memory:
100 %
Epoch: 1 / 1000
0: Square error: 3.1292 2 / 16
Epoch: 2 / 1000
0: Square error: 1.0947 (-2.0344) 0 / 16
Epoch: 3 / 1000
0: Square error: 0.9571 (-0.1377) 3 / 16
Epoch: 4 / 1000
0: Square error: 0.9175 (-0.0396) 3 / 16
Epoch: 5 / 1000
0: Square error: 0.9067 (-0.0107) 3 / 16
Epoch: 6 / 1000
0: Square error: 0.8939 (-0.0129) 3 / 16
Epoch: 7 / 1000
0: Square error: 0.8842 (-0.0097) 4 / 16
Epoch: 8 / 1000
0: Square error: 0.8811 (-0.0031) 4 / 16
Epoch: 9 / 1000
0: Square error: 0.8621 (-0.0190) 4 / 16
Epoch: 10 / 1000
0: Square error: 0.8632 (+0.0011) 4 / 16
Epoch: 11 / 1000
0: Square error: 0.8358 (-0.0274) 4 / 16
Epoch: 12 / 1000

```

```

C:\Users\Rafał\Desktop\Digit-Recognition.exe
Epoch: 75 / 1000
0: Square error: 0.6267 (-0.0025) 12 / 16
Epoch: 76 / 1000
0: Square error: 0.6217 (-0.0050) 12 / 16
Epoch: 77 / 1000
0: Square error: 0.6191 (-0.0026) 12 / 16
Epoch: 78 / 1000
0: Square error: 0.6142 (-0.0049) 12 / 16
Epoch: 79 / 1000
0: Square error: 0.6116 (-0.0025) 12 / 16
Epoch: 80 / 1000
0: Square error: 0.6069 (-0.0047) 12 / 16
Epoch: 81 / 1000
0: Square error: 0.6046 (-0.0023) 12 / 16
Epoch: 82 / 1000
0: Square error: 0.6003 (-0.0043) 13 / 16
Epoch: 83 / 1000
0: Square error: 0.5985 (-0.0018) 13 / 16
Epoch: 84 / 1000
0: Square error: 0.5947 (-0.0038) 14 / 16
Epoch: 85 / 1000
0: Square error: 0.5934 (-0.0013) 12 / 16
Epoch: 86 / 1000
0: Square error: 0.5903 (-0.0032) 13 / 16
Epoch: 87 / 1000

```

```

C:\Users\Rafał\Desktop\Digit-Recognition.exe
Epoch: 989 / 1000
0: Square error: 0.5058 (-0.0000) 15 / 16
Epoch: 990 / 1000
0: Square error: 0.5058 (-0.0000) 15 / 16
Epoch: 991 / 1000
0: Square error: 0.5057 (-0.0000) 15 / 16
Epoch: 992 / 1000
0: Square error: 0.5057 (-0.0000) 15 / 16
Epoch: 993 / 1000
0: Square error: 0.5057 (-0.0000) 15 / 16
Epoch: 994 / 1000
0: Square error: 0.5057 (-0.0000) 15 / 16
Epoch: 995 / 1000
0: Square error: 0.5057 (-0.0000) 15 / 16
Epoch: 996 / 1000
0: Square error: 0.5057 (-0.0000) 15 / 16
Epoch: 997 / 1000
0: Square error: 0.5057 (-0.0000) 15 / 16
Epoch: 998 / 1000
0: Square error: 0.5056 (-0.0000) 15 / 16
Epoch: 999 / 1000
0: Square error: 0.5056 (-0.0000) 15 / 16
Epoch: 1000 / 1000
0: Square error: 0.5056 (-0.0000) 15 / 16

```



## 5.2. File to load network does not exist

```
D:\Biblioteki\Studia\AEi\CP\Project\Digit-Rec
Enter file name: asdafdf
```

```
D:\Biblioteki\Studia\AEi\CP\Project\Digit-Recognition\Debug\Digit-Recog
Enter file name: asdafdf
<"asdafdf" cannot be opened for reading>
Initialise network from file? ["yes"/"no"] _
```

The initialization runs again.

## 5.3. Wrong input for layer size

```
D:\Biblioteki\Studia\AEi\CP\Project\Digit-Recognition\Debug\Digit-R
Initialise network from file? ["yes"/"no"] no
Give layers size:
  [number] + [enter]
  "end" to break
  "del" to delete last layer
784-asdasd_
```

```
D:\Biblioteki\Studia\AEi\CP\Project\Digit-Recognition\Debug\Digit-R
Initialise network from file? ["yes"/"no"] no
Give layers size:
  [number] + [enter]
  "end" to break
  "del" to delete last layer
<not an integer> 784-_
```

## 5.4. Trying to delete first layer

```
D:\Biblioteki\Studia\AEi\CP\Project\Digit-Recognition\Debug\Dig
Initialise network from file? ["yes"/"no"] no
Give layers size:
  [number] + [enter]
  "end" to break
  "del" to delete last layer
<cannot delete> 784-_
```

## 5.5. Trying to end assembly on one layer

```
D:\Biblioteki\Studia\AEil\CP\Project\Digit-Recognition\Debug\Digi
Initialise network from file? ["yes"/"no"] no
Give layers size:
[number] + [enter]
"end" to break
"del" to delete last layer

<two layer required> 784-  
```

## 6. Testing of algorithm

Calculations of one iteration of simple neural network algorithm.

Input:                      Output:

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

---

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:

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

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

---

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:

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

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

Layer 2 - Output

$$W_{2out} := \begin{pmatrix} 1 & 0.6 \\ 0.6 & 0.6 \\ 0.6 & 0.6 \\ 0.6 & 0.6 \\ 0.6 & 0.6 \end{pmatrix}$$

Output

$$S3 := Z2 \cdot W_{2out} = (2.86 \quad 2.511)$$

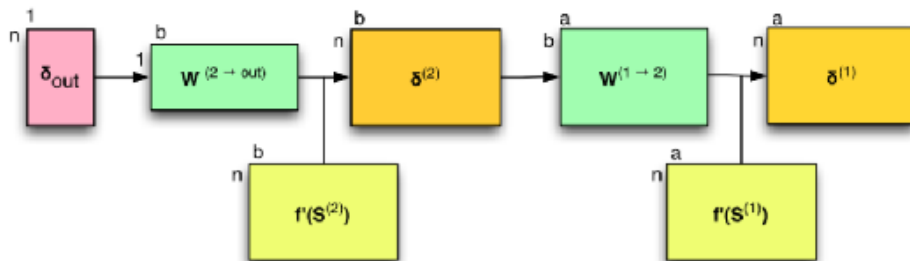
$$Z3 := \frac{1}{1 + e^{-S3}} = (0.946 \quad 0.925) \quad F3 := \left( \frac{d}{dS3} \frac{1}{1 + e^{-S3}} \right)^T = \begin{pmatrix} 0.051 \\ 0.069 \end{pmatrix}$$

Softmax:

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

$$Z4 := \begin{pmatrix} \frac{e^{Z3_{0,0}}}{e^{Z3_{0,0}} + e^{Z3_{0,1}}} & \frac{e^{Z3_{0,1}}}{e^{Z3_{0,0}} + e^{Z3_{0,1}}} \end{pmatrix} = (0.505 \quad 0.495)$$

Error propagation:



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

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

$$\_D2 := W2_{out} \cdot D3 = \begin{pmatrix} 0.005 \\ -0.006 \\ -0.006 \\ -0.006 \\ -0.006 \end{pmatrix}$$

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

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

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

Learning rate:  $\eta := 10$

Weights update:

$$\Delta W_{in1} := -\eta \cdot (D1 \cdot X^T)^T = \begin{pmatrix} 0.001 & 0.002 & 0.002 \\ 0.001 & 0.002 & 0.002 \\ 0 & 0 & 0 \\ 0.001 & 0.002 & 0.002 \end{pmatrix}$$

$$W_{in1} := W_{in1} + \Delta W_{in1} = \begin{pmatrix} 1.001 & 0.602 & 0.602 \\ 0.601 & 0.602 & 0.602 \\ 0.6 & 0.6 & 0.6 \\ 0.601 & 0.602 & 0.602 \end{pmatrix}$$

$$\Delta W_{12} := -\eta \cdot (D2 \cdot Z1)^T = \begin{pmatrix} -0.005 & 0.007 & 0.007 & 0.007 & 0.007 \\ -0.005 & 0.007 & 0.007 & 0.007 & 0.007 \\ -0.005 & 0.007 & 0.007 & 0.007 & 0.007 \end{pmatrix}$$

$$W_{12} := W_{12} + \Delta W_{12} = \begin{pmatrix} 0.995 & 0.607 & 0.607 & 0.607 & 0.607 \\ 0.595 & 0.607 & 0.607 & 0.607 & 0.607 \\ 0.595 & 0.607 & 0.607 & 0.607 & 0.607 \end{pmatrix}$$

$$\Delta W_{2out} := -\eta \cdot (D3 \cdot Z2)^T = \begin{pmatrix} -0.226 & 0.307 \\ -0.214 & 0.291 \\ -0.214 & 0.291 \\ -0.214 & 0.291 \\ -0.214 & 0.291 \end{pmatrix}$$

$$W_{2out} := W_{2out} + \Delta W_{2out} = \begin{pmatrix} 0.774 & 0.907 \\ 0.386 & 0.891 \\ 0.386 & 0.891 \\ 0.386 & 0.891 \\ 0.386 & 0.891 \end{pmatrix}$$

## 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.