Gliwice, 03.04.19r

Semester: 2

Group: 1

Section: 1

COMPUTER PROGRAMMING LABORATORY

Author: Michał Ciszyński

E-mail: michcis796@student.polsl.pl

Tutor: mgr inż. Anna Gorawska

1. Task topic

Neural network. Neural networks simulate the behaviour of a human brain and is able to classify patterns after presenting some examples to them. Write a program that simulates a simple neural network. The user prepares a set of training data (examples, inputs and desired outputs), runs the "training", stores the trained network in a file and uses the network to recognize new patterns. The training may use so called back-propagation.

2. Project analysis

Main goal of implemented neural network was to distinguish different pictures containing hand written digits. To achieve the objective appropriate structure of neural network was proposed. First layer consists of 28*28 input perceptrons which correspond to all pixels in grayscale image of the picture. Input layer is followed by two hidden layers with 16 perceptrons each. Output layers has 10 perceptrons, each indicating probability of digits from 0 to 9.

Activation of all vectors is denoted by:

$$a^l = \sigma\left(w^l a^{l-1} + b_l\right)$$

a^l is vector of values stored in l-th layer of perceptrons
σ is activation function
w^l is l-th matrix of weights
b_l is vector of biases in l-th layer

During process of learning, backpropagation algorithm is applied. Firstly, after evaluating n input images, Loss function [C] is calculated as following:

$$C = \frac{1}{2n}\left\| y(x) - a^L(x) \right\|^2$$

y(x) denotes expected outcome.

Then proportional error with respect to each perceptron in last layer is calculated as:

$$\delta^L = \nabla_a C \odot \sigma'\left(z^l\right)$$

z^l is value stored in a perceptron before applying activation function

After that one can finally calculate error with respect to all perceptrons as follows:

$$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right)$$

Last step of backpropagation is to calculate derivatives of loss with respect to weights and biases:

$$\frac{\partial C}{\partial b^l} = \delta^l \qquad \frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

The remaining part is to change each weight and bias in proportion to derivative and learning rate.

In implemented neural network RELU was chosen as activation function in all layers except the last one, where sigmoid gave better results.

Initialization of all values is done randomly at the beginning. However the are some heuristics which help in preventing uniform errors and exploding/vanishing gradients:

$$w^l_{jk} = rand() \cdot \sqrt{(\frac{2}{size^{l-1}})}$$

where rand() generates random numbers with normal distribution, and $size^{l-1}$ denotes number of perceptrons in preceding layer.

3. External specification

In each step of operation of the program user has following choices:

- learning the neural network (l)- after choosing this option user has to initialize two parameters: learning rate and number of samples in learning packet. When it is done learning process starts. Additionally after terminating the learning process user has option to save created neural network, after specifying name of file.
- reading a neural network (r)- after choosing this option user has to provide name of file, where neural network is stored. If opening of file is impossible, then appropriate warning is displayed and program returns to „main menu”
- examining a neural network (e)- after choosing this option user decides how many test samples he deserves to see. In the command prompt guess of the neural network is displayed with according probability. In additional window picture itself is displayed. This option is available only after reading a network from file or training one.
- terminating (t)- after choosing this option program stops its operation

4. *Internal specification*

Structure of the project:

nn.cpp- file containing definitions of all used functions

NN.h- file containing declarations of functions and structures

Source.cpp- file with main function

Implemented functions:

```
void initialize(NNetwork *n, int num, ...)
```
- function dynamically allocating arrays of neural network, it also initializes appropriate weights and biases with random numbers

```
void destroy(NNetwork *n)
```
- function deallocating memory occupied by neural networks

```
void work(NNetwork *n)
```
- function which feeds forward values in neural network accordingly to equations above

```
void backprop(NNetwork *n, int num, double **inputs, double **outputs, double l_rate)
```
- function which implements backpropagation on array of given inputs and expected outputs. “num” denotes number of samples in a packet. “l_rate” is given learning rate

```
void learn(NNetwork *n, int num1, double l_rate, int packet, double **l_in, double **l_out, int num2, double **t_in, double **t_out, int tolerance, int max)
```
- function which is responsible for learning process. It runs backpropagation as long as performance on validation data is decreasing. “num1” is a number of learning samples, “num2” is a number of validation samples. “tolerance” denotes number of decreases of performance on validation data, after which learning terminates. “max” on the other hand points maximum number of epochs which can be performed during training, and after which function will terminate.

```
int save(NNetwork *n, const char *name)
```
- function saving neural network to a file

```
int read(NNetwork *n, const char *name)
```
- function reading neural network from a file

Used structure:

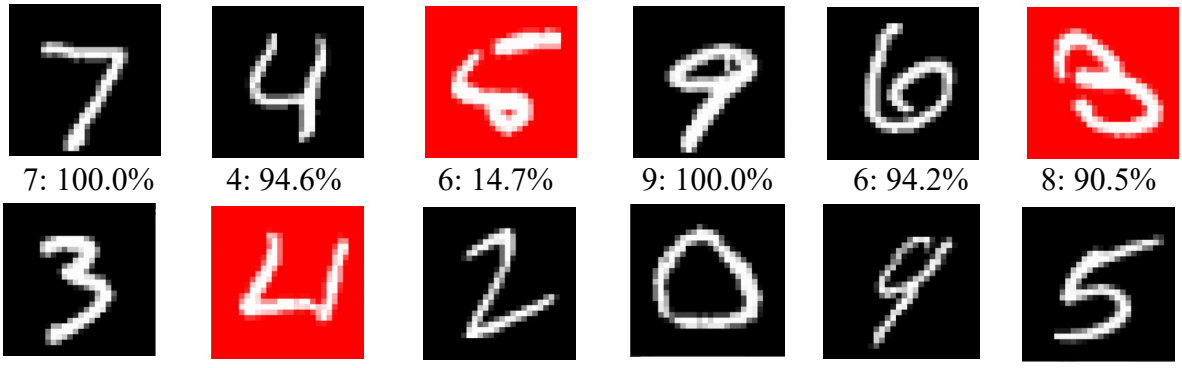
```
struct NNetwork
{
	int nLayers;
	int *nNodes;
	double **nodes;

	double ***weights;
	double **biases;
};
```

nLayers- number of layers of a neural network
nNodes- pointer to an array storing number of perceptrons in each layer
nodes- pointer to a two dimensional array storing values of each perceptron
weights- pointer to a three dimensional array storing all weights
(ex. weights[i][j][k] is weights between i-th and i+1-th layer between j-th and k-th perceptron)
biases- two dimensional array storing bias of each perceptron (there are no biases in the last layer)

5. *Testing*

During testing network is evaluated on 10 000 testing samples, with which network did not have any contact. The best examples of trained network showed over 90% accuracy in distinguishing digits from each other. Here are some examples of operation of neural network:



7: 100.0% 4: 94.6% 6: 14.7% 9: 100.0% 6: 94.2% 8: 90.5%

3: 100.0% 0: 0.6% 2: 91.1% 5: 84.0% 9: 3.5% 0: 100.0%

```
training and test data loaded

Choose: learning(l), reading(r), terminate(t): l
learning rate: 0.1
number of samples in a packet: 3
epoch = 2, error = 19.0%

Do you want to save trained network?(y/n):y
Give file name:example.nn

Choose: learning(l), reading(r), examining(e), terminate(t): e
how many test samples do you want to see?: 10
9: 99.7%
Choose: learning(l), reading(r), examining(e), terminate(t): r
Give file name:thebest.nn

Choose: learning(l), reading(r), examining(e), terminate(t): e
how many test samples do you want to see?: 10
9: 100.0%
Choose: learning(l), reading(r), examining(e), terminate(t): r
Give file name:unavailable.nn
Couldn't read the file

Choose: learning(l), reading(r), terminate(t): t
```

Exemplary execution of the program

6. *Conclusions and comments*

- Best performance was achieved with learning rate below 0.1 and packets containing 3-5 samples.
- Higher precision could be achieved using convolutional neural networks.
- Using sigmoid as activation function in all layers resulted in vanishing gradients and very long learning time.
- Using too high learning rate resulted in overshooting and unsatisfactory accuracy.
- Using heuristic for initializing weights gave a huge boost to initial parameters of neural network, it also prevented from unified errors with respect to all perceptrons.
- Samples used by the program were already preprocessed (they belong to MNIST database). They were downsampled to size 28*28, centered using center of mass and finally anti aliasing was performed.

7. *Sources*

Images used in the project are part of MNIST database:
http://yann.lecun.com/exdb/mnist/