
	<p align="center">Institute of Informatics Silesian University of Technology</p> <p align="center">Division of Microinformatics and Automata Theory</p>			
Academic year:	Type of studies	Course:	Group:	Section:
2017/2018	SSI	BIAI	PSI	
Name:	Rafał	Exercise supervisor: OA/JP/KT/GD/BSz/GB	GB	
Surname:	Suszka			
<p align="center"><i>Final report</i></p>				
<p>Subject:</p> <p align="center">Emoticons recognition using a neural network</p>				
Date: dd/mm/yyyy		12.09.2018		

Table of Contents:

1. The subject of the project and a description of the assumptions.....	3
2. Task analysis	3
2.1. Input data format.....	3
2.2. Natural network	4
2.3. Learning network.....	5
3. Internal specification.....	6
4. External specification.....	8
5. Tests and conclusions.....	9
6. Github.....	10
7. Bibliography.....	10

1. The subject of the project and a description of the assumptions

Topic of the project is: "Emoticons recognition using a neural network".

The goal of the project is to write a program that recognizes emoticons.

Assumptions implemented in the program:

- windows application written in C#,
- implementing own neural network,
- recognition of 5 different emoticons,
- input data: bitmaps any sizes,
- output data: information about the recognized emoticon.

2. Task analysis

Before writing a program, the task was analyzed.

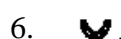
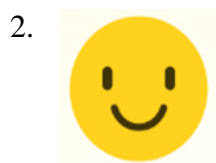
2.1. Input data format



Input data can be any sizes bitmaps with emoticons. The program analyzes processes the image to a binary form (0, 1).

The processing algorithm is:

1. Image provided by the user
2. Resize image to size 100 x 100 pixels.
3. Image cropping (so that the lips are visible).
4. Resize image to size 31 x 16 pixels.
5. Applying Sobel filter.
6. Invert colors.
7. Correct colors.
8. Resize image to size 20 x 10 pixels.
9. Convert bitmap to binary table.

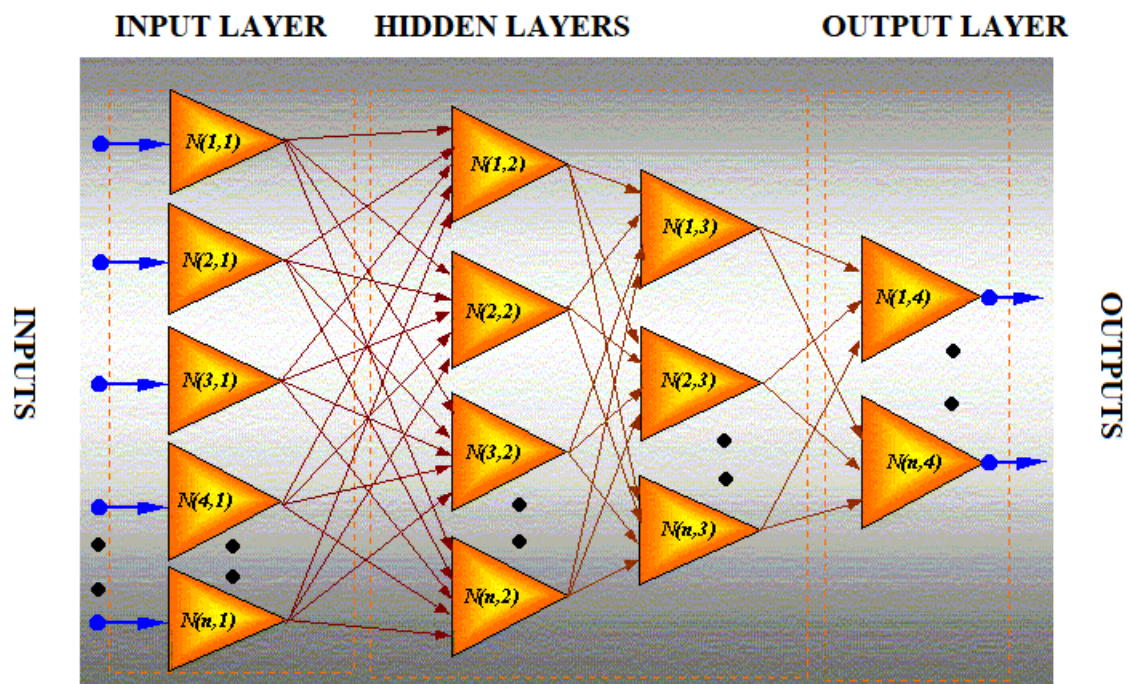
Example:



7. 
8. 
9. 00000000000000000000
00000000000000000000
00001100000001110000
00001110000001110000
00001110000001110000
00001111000001010000
00001101111111000000
00000010111011000000
00000011111100000000
00000000111000000000

2.2. Natural network

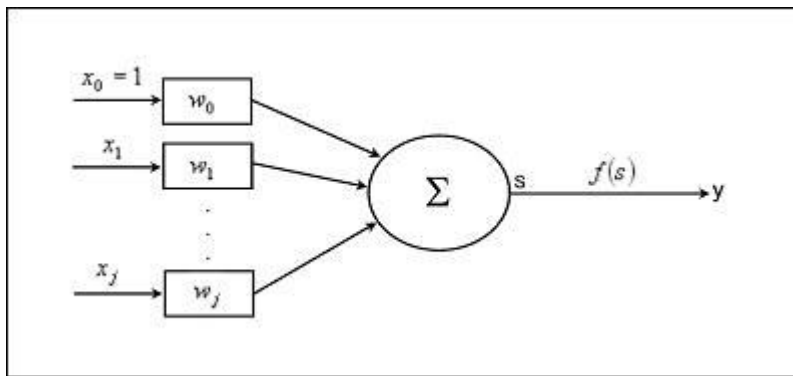
The neural network used in the program consists of the input layer, two hidden layers and the output layer. Natural network scheme:



Picture 1. Natural network scheme.

Neural network has 200 inputs and 5 outputs. There are 40 neurons in the first hidden layer. In the second hidden layer there are 15 neurons.

Neuron construction:



Picture 2. Neuron scheme.

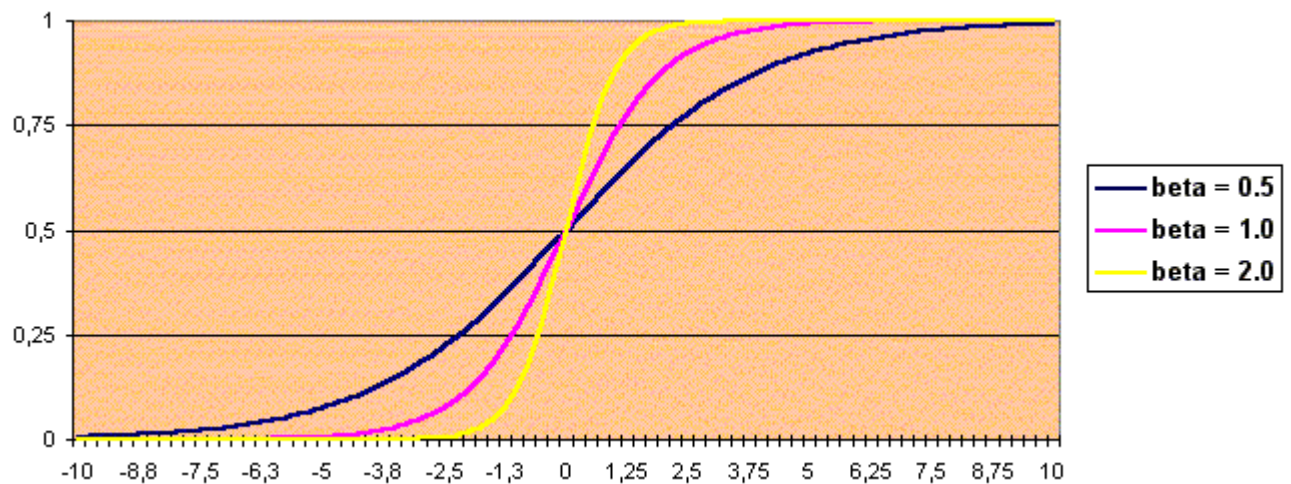
x_j – input number i

w_j – weight on input number i

$$s = \sum_{j=0}^N w_j * x_j$$

$$f(s) = \frac{1}{1 + e^{-(\text{beta} * s)}}$$

Neurons are activated by sigmoidal function:



Picture 3. Sigmoidal function

2.3. Learning network

The neural network is learned using the back propagation algorithm.

The learning algorithm is:

1. Data processing via the network.
2. Calculation of errors at network outputs.
3. Calculation of errors on every neuron in the network (back error propagation).
4. Weight correction.
5. Calculation of network error for a single epoch.

The algorithm repeats a fixed number of epochs.

3. Internal specification

The following are the classes of the program along with a description of their fields and methods.

1. **MyPicture** – class that stores a bitmap with an image and provides methods for its processing.

Fields:

- `private` `Bitmap` `inputData` – variable that stores bitmap.

Methods:

- `public` `MyPicture(string path)` – constructor. The path to the file is an argument.
- `public` `Bitmap` `InputBitmap` – getter field `inputData`.
- `public void` `CropImage(int x, int y, int width, int height)` – method that cropping bitmap. The arguments are successively the x, y coordinates of the starting point, height and width.
- `public void` `CorrectCollors()` – method that corrects colors on bitmap. Grey pixels are converted to black or white depending on the level of gray.
- `public void` `InvertColors()` – method that changes white pixels to black pixels and black pixels to white pixels.
- `public void` `Resize(int width, int height)` – method that changes the size of the image. The arguments are new width and height.
- `public void` `SobellFilter()` – method that applies a Sobel filter to bitmap.
- `public int[]` `ToTable()` – method that converts bitmap to a binary table (0 – white pixel, 1 – black pixel).

2. **Neuron** – class that stores information about single neuron.

Fields:

- `private static` `Random` `n` – static variable used in randomize neuron weights.
- `private double` `beta` – variable that stores beta coefficient.
- `private double` `delta` – variable that stores error.
- `private` `List<double>` `inputsWeights` – list that stores actually weights of subsequent neuron inputs.
- `private` `List<double>` `oldInputsWeights` – list that stores old weights of subsequent neuron inputs.
- `private double` `value` – variable that stores actually neuron output.

Methods:

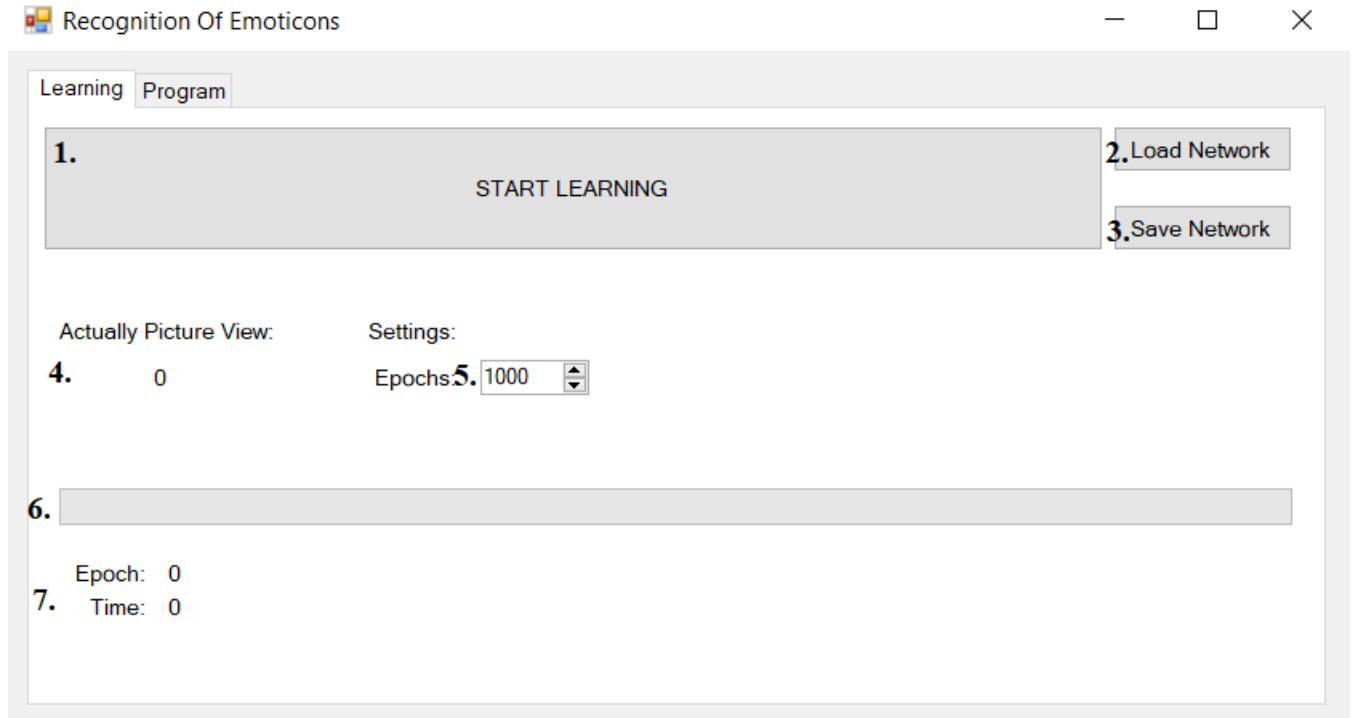
- `public` `Neuron()` – constructor.
- `public double` `Beta` – getter field `beta`
- `public double` `Delta` – getter and setter field `delta`.
- `public int` `InputsCount` – getter that return number of inputs.
- `public double` `Value` – getter field `value`.
- `private double` `ActivationFunction(double s)` – method that returns value of activation function. The argument is sum of multiplication of weights and inputs.
- `public void` `AddInputs(int numbersOfInputs)` – method that add inputs in the number given as its argument.
- `public void` `ChangeInputWeight(int numberOfInput, double newWeight)` – method that change inputs weight on selected input. The arguments are numbers of input and new weight.
- `public double` `GetOutput(List<double> inputs)` – method that calculates and returns neuron output. The argument is list with neuron inputs.
- `public double` `GetOutput(List<int> inputs)` – method that calculates and returns neuron output. The argument is list with neuron inputs.
- `public double` `GetOldWeight(int numberOfInput)` – method that returns old weight on input given as its argument.

- `public double GetWeight(int numberOfInput)` – method that returns actually weight on input given as its argument.
3. **NeuronNetwork** – class that stores information about natural network.
- Fields:
- `private List<int> inputLayer` – list that stores network inputs.
 - `private List<Neuron> hiddenLayerFirst` – list that stores network neurons from first hidden layer.
 - `private List<Neuron> hiddenLayerSecond` – list that stores network neurons from second hidden layer.
 - `private List<Neuron> outputLayer` – list that stores network neurons from output layer.
 - `private int numberOfNeuronsInInputLayer` – variable that stores number of neurons in input layer. Default it is 200.
 - `private int numberOfNeuronsInHiddenLayerFirst` – variable that stores number of neurons in first hidden layer. Default it is 40.
 - `private int numberOfNeuronsInHiddenLayerSecond` – variable that stores number of neurons in second hidden layer. Default it is 15.
 - `private int numberOfNeuronsInOutputLayer` – variable that stores number of neurons in output layer. Default it is 5.
 - `private double eta` – variable that stores eta parameter.
 - `private double alfa` – variable that stores alfa parameter.
 - `private string[] classNames` – table that stores names of output classes.
- Methods:
- `public NeuronNetwork()` – constructor.
 - `public string[] ClassNames` – getter field classNames.
 - `public void CreateNetwork(int neuronsInInputLayer)` – method that creates natural network. The arguments is number of neurons in input layer.
 - `public List<double> RunNetwork(int[] inputData)` – method that returns outputs of the natural network based on given inputs as argument.
 - `public void LearnNetwork(int[] inputData, int[] correctOutputData)` – method that learn natural network. The arguments are table of inputs and table of correct outputs.
 - `public void SaveNetworkToFile()` – method that saves natural network to file.
 - `public void LoadNetworkFromFile(string path)` – method that loads natural network from file. The argument is path to file.
4. **Form1** – class that combines the operation of a window application and a neural network.
- Fields:
- `private string inputDataPath` – variable that stores path to input data.
 - `private NeuronNetwork neuronNetwork` – variable that stores natural network.
 - `private int learningEpoches` – variable that stores number of learning epochs. Default it is 1000.
- Methods:
- `public Form1()` – constructor.
 - `private void button_path_Click(object sender, EventArgs e)` – method that it is called after pressing the button ‘Set Path’.
 - `private void button_run_Click(object sender, EventArgs e)` – method that it is called after pressing the button ‘Run’.
 - `private void button_save_learning_Click(object sender, EventArgs e)` – method that it is called after pressing the button ‘Save Network’.
 - `private void button_load_learning_Click(object sender, EventArgs e)` – method that it is called after pressing the button ‘Load Network’.

- `private void button1_Click(object sender, EventArgs e)` – method that it is called after pressing the button ‘START LEARNING’.

4. External specification

Learning mode:



Description:

1. Start natural network learning.
2. Load natural network from file.
3. Save natural network to file.
4. Show actually learning picture.
5. Selection of the number of epochs.
6. Learning progress bar.
7. Learning statistics.

After completing each action, the program informs about it with an appropriate message.

Program mode:

Description:

1. Set path to emoticon (bitmap).
2. Run emoticon to natural network.
3. Detailed natural network outputs.
4. Natural network outputs.

After completing each action, the program informs about it with an appropriate message.

5. Tests and conclusions

144 different emoticons were collected. They were divided into two collections: for natural network learning (129 emoticons) and for natural network testing (15 emoticons). A lot of tests were carried out with different program configuration (networks and number of learning epochs). The results are presented as follows.

	Inputs	First hidden layer	Second hidden layer	Outputs	Eta	Alfa	Learning epochs	Max ERMS	recognized	sum	%
1.	200	30	10	5	0,2	0,9	1000	0,2	10	15	66,67%
2.	200	30	10	5	0,2	0,6	1000	0,2	8	15	53,33%
3.	200	30	10	5	0,1	0,6	1000	0,2	7	15	46,67%
4.	200	20	10	5	0,2	0,6	1000	0,2	1	15	6,67%
5.	200	20	10	5	0,2	0,6	2000	0,2	2	15	13,33%
6.	200	40	10	5	0,2	0,6	1000	0,2	7	15	46,67%
7.	200	40	10	5	0,2	0,6	2000	0,2	8	15	53,33%
8.	200	40	15	5	0,2	0,6	1000	0,2	10	15	66,67%
9.	200	40	15	5	0,2	0,6	2000	0,2	6	15	40,00%
10.	200	40	15	5	0,2	0,6	800	0,2	8	15	53,33%
11.	200	40	15	5	0,2	0,6	1250	0,2	8	15	53,33%
12.	200	50	20	5	0,2	0,6	1000	0,2	8	15	53,33%
13.	200	50	20	5	0,2	0,6	1500	0,2	6	15	40,00%

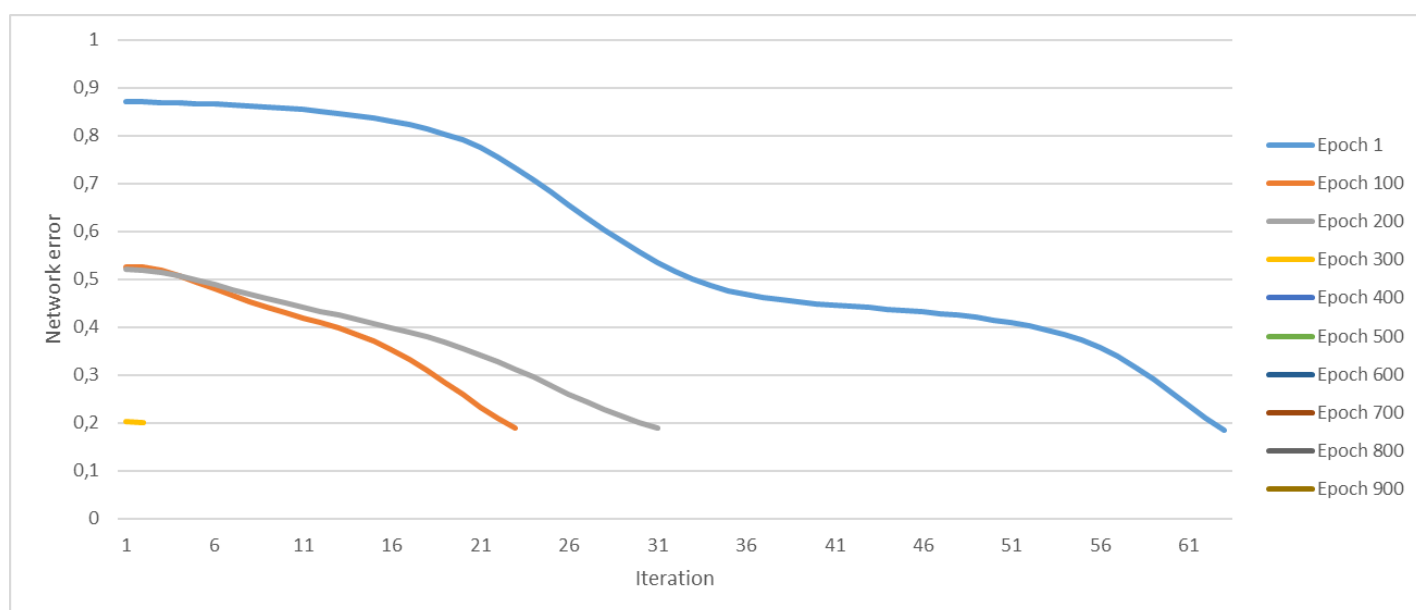
14.	200	40	15	5	0,2	0,6	1000	0,1	9	15	60,00%
15.	200	40	15	5	0,2	0,6	1000	0,05	8	15	53,33%
16.	200	40	15	5	0,2	0,6	2000	0,05	9	15	60,00%

The best results were obtained on network number 1 and 8.

First network consists of 200 inputs, 30 neuron in first hidden layer, 10 neuron in second hidden layer and 5 outputs. Natural network was learned by 1000 epochs with eta 0.2, alpha 0.9 and maximum ERMS 0.2.

Eight network consists of 200 inputs, 40 neuron in first hidden layer, 15 neuron in second hidden layer and 5 outputs. Natural network was learned by 1000 epochs with eta 0.2, alpha 0.6 and maximum ERMS 0.2.

Graph of network errors for the selected image depending on the iteration.



Epochs greater than or equal to 400 have an error smaller than the one already established (0,2) in the first iteration.

6. Github

<https://github.com/rsuszka/rsuszka/tree/master/RecognitionOfEmoticons>

7. Bibliography

Picture 1. – http://www.ai.c-labtech.net/sn/pod_prakt.html

Picture 2. – https://4programmers.net/Z_pogranicza/Sztuczne_sieci_neuronowe_i_algorytmy_genetyczne

Picture 3. – http://www.ai.c-labtech.net/sn/pod_prakt.html