Character Recognition Using Neural Networks

Rókus Arnold[*], Póth Miklós[*]
[*] Subotica Tech, 24000 Subotica, Srbija, arnold@eunet.rs, pmiki@vts.su.ac.rs,

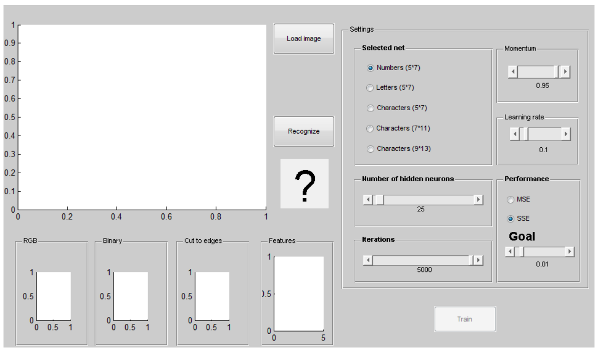
Abstract— **It has been 50 years since the idea popped up that calculating systems can be made on the replica of the biological neural networks. Still, the development of this science branch made the improvement of these systems possible only in the last 25-30 years [6]. Nowadays, neural computing is a very extensive, separate science. Its solid theory basis made it possible to use them to solve many kind of problems in artificial computing, thus improving the experience of the science. Neural networks are commonly used to solve sample-recognition problems. One of these is character recognition. The solution of this problem is one of the easier implementations of neural networks. With the help of Matlab's Neural Network Toolbox, we tried to recognize printed and handwritten characters by projecting them on different sized grids (5x7, 7x11, 9x13). The results showed that the precision of the character recognition depends on the resolution of the character projection. Also, we realized, that not every writing style can be recognized using the same network with the same precision. This shows that the variety of human handwriting habits can't be fully covered with one neural network.**

Keywords: Neural Networks, Character Recognition, Training.

I. INTRODUCTION

The goal of optical character recognition is to classify the alphanumeric or other optical samples which are mostly saved as digital images. The process of the recognition consists of several steps. Matlab has a special toolbox, called Neural Network Toolbox, which made our job easier when constructing the best suiting Neural Network. Though this toolbox gave us a lot of help, the knowledge of the theory is still needed.

Our program's goal is to show the precision and speed of character recognition depending on the parameters of the implemented neural network. It communicates with the user through a GUI. We can load a preferred picture and try to recognize the characters on it with five different sized neural networks. We can train these networks with the preferred parameters.



Settings

Selected net: [✓ Numbers (5*7)] Letters (5*7) Characters (5*7) Characters (7*11) Characters (9*13)

Momentum: 0.95

Learning rate: 0.1

Number of hidden neurons: 25

Performance: MSE [✓ SSE]

Iterations: 5000

Goal: 0.01

Figure 1. GUI of the program

II. IMPORTANT METHODS

The following methods are very important when we want to know how the neural network works when recognizing characters. These methods define the way in which the neural network goes through with adapting its parameters to recognize the wanted character-set.

A. Feedforward Backpropagation

The backpropagation (BP) algorithm's most specific feature is the error that the neural network gets on its output. This error is equal with the difference between the real and required output values. When using backpropagation, we look at the weight/change, which mostly minimizes the error in the output [2].

This algorithm is a method that depends on the gradient value of the moment. The learning starts when all of the training data was showed to the network at least once. The learning method, as for every network learning algorithm, consists of the modification of the weights.

The BP algorithm works with small iterative steps: after showing all samples to the network, it will generate an output value with the pre-initialized weights. This output value will be compared to the required output value which we gave, and the mean square error is calculated. After this, our network will go backwards through the weights. It will use the gradient of the criteria-field to determine the best weight/modification to minimize the mean square error. This method will go through all samples, minimizing the error until a level that we specified. After this, we can say, that our network learned to solve the problem good enough. The network will never learn how to solve the problem perfectly, but it will close on it asymptotically.

The algorithm consists of the following steps:

1. Define a training-sample for the network.
2. Compare the gotten output value with the required ones, and calculate the error for every output neuron.
3. Calculate the required output for every single neuron. We must also take into consideration the incremental factor, which shows us how much every neuron weight has to be changed, so that they will be perfect in values. This shows the local error.
4. Modify the weight in from of every neuron in the way to minimize the local error.
5. Give a level of blame to every neuron, this way giving higher responsibility for those neurons with greater weights before them.
6. Repeat the method from step 3 for the neurons of the previous layer, using the "blame" as factor.

Now we have equation (1):

$$x_{k+1} = x_k - a_k g_k \qquad (1)$$

where $\mathbf{x_k}$ is a vector representing the momentary weights and biases, $\mathbf{g_k}$ is a momentary gradient, and $\mathbf{a_k}$ is the learning rate.

B. Learning Rate and Momentum

At present, there is no general method for the selection of the **learning rate** (LR). When speaking about a perceptron, it is not hard to specify the learning rate, but when speaking about a multi-layer perceptron, the selection of the learning rate is based on experience. For every layer, it is needed to choose a different LR value. The learning rate is symbolized with a "μ". Choosing a low μ value will result in slow learning, while a high μ value will result in the danger of jumping through the minimum point. It is recommended to start our training session with high μ values, and decrease them during the learning process until we find the best suitable value [2].

The momentum method is a heuristic method, where in every iterative step the weight modification consists of two steps. First we define the weight-modifying values with the help of the gradient, but with this, the modification is not at its end, because we add the effect of the previous weight-modification, too. This way, the weight modification is shown in equation (2):

$$\Delta w(k+1) = \mu(-\nabla(k)) + \eta \Delta w(k) \qquad (2)$$

where η stands for the momentum which is a value between 0 and 1.

The new value gives a kind of inconsistency to the equation, because in the momentary step the direction of the previous modification is counted in the direction of the momentary modification, and this way, the older directions, too. This is very good when our criteria-field consists of lots of little valleys. Without the momentum, the learning process would take too much time passing these valleys. We have to choose the learning rate and momentum wisely. We have to choose them in a way which minimizes the error.

C. Gradeint-descent

During the training session we search the optimization of the criteria-field, more exactly that "w" weight value, where the criteria-function will minimize the error level. Visually, the criteria-function is a surface that depends on a parameter. This is called the criteria-field. The finding of the values that define the maximum or the minimum of this field is our goal. So these learning functions are optimization functions. We have a lot of these methods in the neural terminology, also used in other branches of science. In case of neural networks, the gradient-descent method is the most important. This searches the minimum of the field.

$$\nabla[C(w)] = \frac{\partial C(w)}{\partial w} = 0$$

where *∂C(w)* is the generalized error. This will provide us with an analytic result, which can cause us problems in harder cases. In a lot of cases, we can't give an analytic result. In these cases we change the parameters of the system until we reach the minimum of the criteria-field.
The most important method when speaking about character recognition is the steepest descent or gradient descent method. This method iterates in the negative direction of the gradient. The iterative equation (3) is:

$$w(k+1)=w(k)+\mu(-\Delta C(k)) \qquad (3)$$

III. Neural Networks

There are five neural networks included, all with the same construction. They have one input layer, and two active layers, one being the output and the other the hidden layer. With the help of the GUI, we can change the number of the hidden neurons in the hidden layer. This number can be between 1 and 5000, and this defines the size of the hidden layer. The input and the output layer sizes depend on the selected neural network. The five neural networks differ in this parameter. This describes the resolution of the neural network or more exactly the resolution of the character we are recognizing. The dimension shows the resolution of the image we are recognizing. This defines a grid on which the character is projected. The products of the two dimension sizes give us the input size of that neural network, while the number of characters in it gives the output size. As of default, the first neural network is loaded in the program.

Table I. Neural Networks Used In The Program

		Dimension	Inputs	Outputs
1	Numbers	5x7	35	10
2	Letters	5x7	35	35
3	Characters	5x7	35	45
4	Characters	7x11	77	45
5	Characters	9x13	117	45

IV. Main Functions

1. Training
    a. Preprocessing – processing the data in the form needed.
    b. Feature extraction – we minimize the size of the data with saving only the needed information. This will give us a vector with scalar values.
    c. Estimation of the model – with a number of vectors we have to estimate the model for every class of training data.
2. Testing
    a. Preprocessing
    b. Feature extraction
    c. Classification – we compare the vector to the model with the extracted features to find the best suiting value. There are several methods to do this.

The preprocessing consists of several steps [4]:

- Converting to binary.
- Morphological methods.
- Segmentation.

The segmentation is the most important part of the preprocessing method. It makes us possible to extract every little detail of every separate character. After the segmentation we have to decide which details are important for us. Feature extraction can have the following steps [1]:

- Momentum based details.
- Hough and Chain code-transformation
- Fourier transformations

The estimation of the model consists of an estimation of the statistical model defined for every character. For example we make a statistical review of every number's orientation and size for every writing style. We will get a statistical review where every number will have its own range in both statistical factors, separately from the others. After all , the main goal of a recognizing system is to make decisions in case of classifying data with the dependency of their samples. This is the classification. An OCR program meets extracted features from which it has to decide which character is on the image.

V. Definig the Training Data

Every neural network needs data to be trained. These data consist of input and target data. The training of one character can be done with an input and an output vector. The length of the input vector will be the resolution of the input. I.e. if we work with the fourth network, then the length will be 77 elements, every row coming after each other in it. The elements can have values 0 or 1. If the grid part of the image represented by the vector element has more than 50% coverage of the letter's features, then the vector element's value will be 1, otherwise it will be 0. By checking this for all of the grid parts, we get the vector. In the first neural network we have 10 numbers. This means we will have 10 rows of this kind of vectors, every row representing a number. This describes the input matrix. This kind of data form is expected as inputs of the neural networks in Matlab.

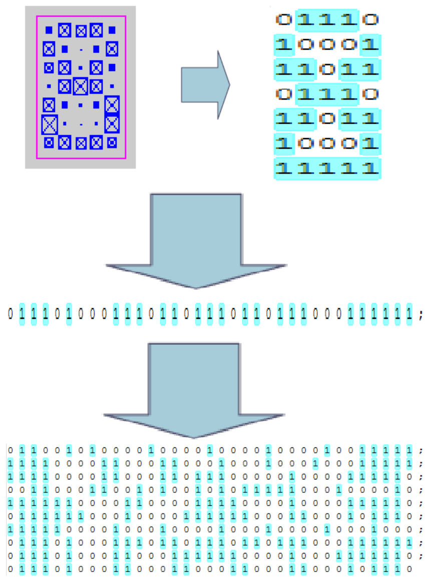


Figure 2. The look of the input data

For a network with N outputs, the output matrix will be an NxN dimension matrix, filled with zeros, and having ones on the main diagonal. This way we classified every character, making every row to differ in an element. This step of the recognition was done manually.

VI. Program Features

We can change the performance type, performance goal, momentum, and learning rate values of the selected neural network. After altering the values, we have to retrain our network. The performance type can be Sum Squared Error and Mean Squared Error. The goal describes the minimal error value, at which the training will stop. Momentum and learning rate helps us get through local minimums easily, so we have to choose them wisely. Bigger values make training faster, but we can slip past the best minimum. Clicking the train button, we start the training of the network with the data described for the selected network and with the selected parameter values. If we have an image loaded, and something selected on it, clicking the recognize button, the recognition will start. The selected image part is cropped, it is transformed to binary image, then it gets cut to its edges, and finally it will be projected on the grid.

This process is called feature extraction. The last step produces a vector. The elements of it show us how many features of the recognized character are at the appropriate grid part in an interval of 0 and 1. After this, the vector is put on the input of the selected net. In the output vector, we will have to check which element has the greatest value. With this we can conclude which number was recognized.

VII. Research Results

Fig. 5 shows the numbers of iterations needed to train the network depending on the size of the hidden layer. According to Tou and Gonzalez [5], the number of hidden neurons must be at least the half the number of the output neurons. We can see that with 10 output neurons, the time of a successful training was the shortest with 120 neurons. Though with a slow computer, this time can be more, given the technique of nowadays computing, we can use 10 times more neurons for the hidden as for the output layer. Bigger resolution of input data increased the number of iterations needed, but had no effect on performance. If the number of hidden neurons is less than the half of output neurons, we get an instable network.

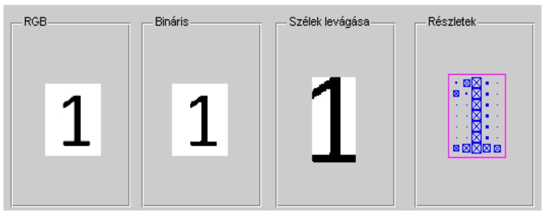


Figure 3. Feature extraction

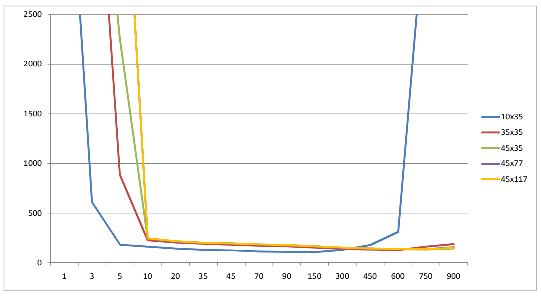


Figure 5. Iterations needed for good recognition ability depending on the number of hidden neurons

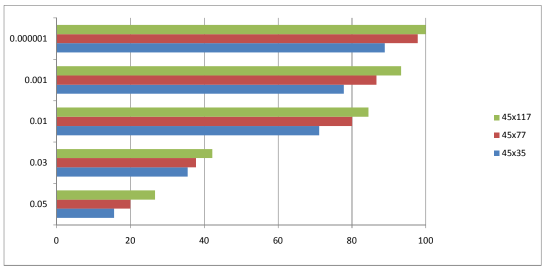


Figure 6. Recognition success depending on the performance

An optimal implementation according to documentations and the diagram is at least twice as many hidden neurons than output neurons. Fig. 6 shows the performance of the networks. We can see that the most precise network is the one with the best resolution. This test was made with printed text. It can be seen that the lower the performance goal is, the better precision we get. For the recognition of handwritten characters, we used the final two neural networks.

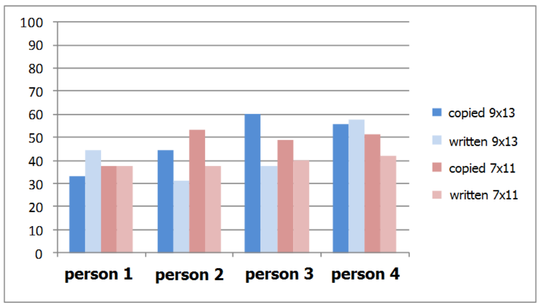


Figure 7. Recognition accuracy by the four persons

Four people wrote down 45 letters needed for the test by copying the letters exactly the same way as they were printed, and writing them with their own style. Fig. 7. shows the accuracy.

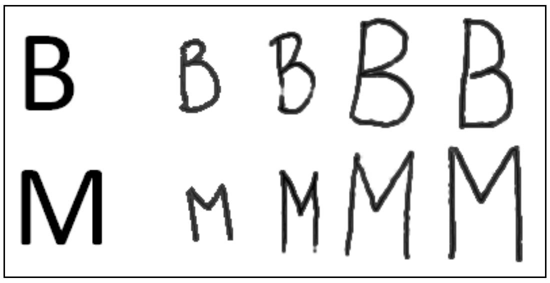


Figure 8. Differences in writing styles

During the process of handwriting recognition, the accuracy fell a lot. The reason for this was because the network was faced with data it never saw before. The copied text recognized with the network with the highest resolution gave the best results. Fig. 8. shows, why some characters were absolutely not recognized. The upper part of letter "B" and the middle part of letter "M" differs very seemingly, which caused the network not to recognize it.

Also, Fig. 8 shows us why it is hard to build a network that recognizes all writing styles. Writing the same letters, even the users have made differences in letter orientation and size. The smaller letters were better recognized with the network with smaller resolution. Regardless the difference of the orientation, size and place of the characters, the network still had a 60% precision. This is far from the 97% minimum, but 2D recognition is only a part of the solution. Combining this with several other modes of minimizing the searching space and helping the recognition with dictionary methods, neural networks prove to be a promising solution [1].

References

[1] Alexander J. Faaborg (Cornell University, Ithaca NY) – „Using Neural Networks to Create an Adaptive Character Recognition System" (May 14, 2002)

[2] Altrichter Márta, Horváth Gábor, Pataki Béla, Strausz György, Takács Gábor, Valyon József – „Neurális Hálózatok" (2006, Budapest, Panem Könyvkiadó Kft.)

[3] Deepayan Sarkar (University of Wisconsin, Madison) – „Optical Character Recognition using Neural Networks" (December 18, 2003)

[4] Jesse Hansen – „A Matlab Project in Optical Character Recognition" (OCR)

[5] J.T. Tou and R.C. Gonzalez (Reading, Massachusetts) – „Pattern Recognition Principles" , Addison-Wesley Publishing (1974)

[6] Hertz J., Krogh A., Palmer R. G. – „Intorduction to the Theory of Neural Computation", Addison-Wesley Publishing (1991)