	Institute of Informatics Division of Microinformatics and Automata Theory			
Academic year	Type of studies	Subject	Group	Section
2017/2018	**SSI**	**BIAI**	**GKiO3**	**3**
Assigned teacher	**dr inż. Grzegorz Baron**		Date of classes	
Section members: email:	Aleksander Patschek Mateusz Nieć matenie537@student.polsl.pl		**Thursday** **15:45 - 17:15**	

Final report

Project's topic:

Neural network which recognize alphanumeric signs

Date of submission: dd/mm/rrrr	11/07/2018

1. Problem definition 3
Functional requirements 3
Outline of neural network 3
Backward error propagation 4
2. Code analysis 5
Internal Specification 5
External Specification 6
3. Testing and results 7
Different number of neurons in hidden layer 7
Conclusion 10
Different learning factor 11
Conclusion 14
An attempt to improve accuracy 14
Tests on set of 20 signs 15
4. Final conclusion 15

1. Problem definition

The main objective of the project was implement program which is able to recognize alphanumeric sings on pictures. Picture always have 32x32 pixels. Signs can have different size and font.

Functional requirements

Prepared program besides the obvious aspect of generating set of weights, should also let the user possibility to check:

- number of epochs,
- value of all weights
- value of accuracy
- value of error in every epoch

Outline of neural network

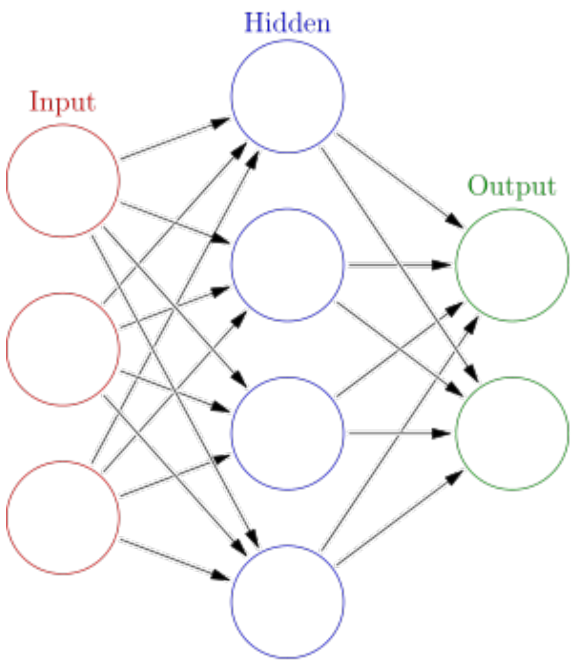
The concept of neural networks is very simple. First we must decide how many neurons we want to have in input layer. In our case we are having one neuron for one pixel of the picture.

Then we need to choose number of neurons in hidden layer. Every neuron in hidden layer depend from all neurons in previous layer but with different weight. We can have more than one hidden layer. The value of single neuron in hidden layer is calculated by the follow formula:

$$\sum_{i=1}^{n} value_i \quad * \quad weight_i$$

where n = number of neurons in previous layer

In the end there is the output layer. Number of neurons on this layer depend form number of outputs which we expect. In our case we have the same number of output neurons as number of different characters which we try to recognize. We are checking which neuron have the greatest value and this is result of recognition.



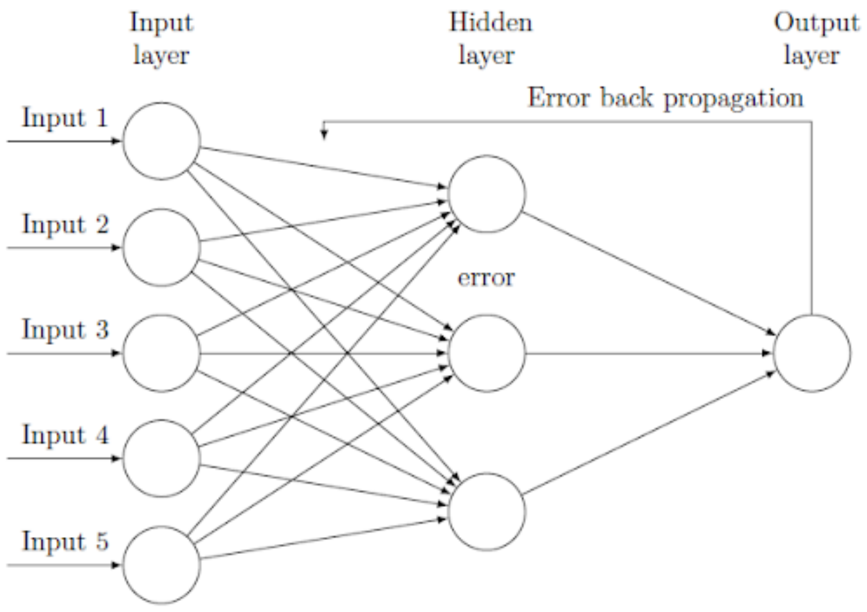
Schema of neural network

Backward error propagation

For the best accuracy improvement, program use backward error propagation. This method consists in to depend weights in the next epoch from the recognition error from current epoch. The formula to calculate new weight as fallows:

*new weight = old weight - (learning factor * error)*

As you can see, this method strong depend on the value of the factor which we choose.



2. Code analysis

We decided to use Python as our main language. We chose it because it's simple, quite fast and very popular in machine learning. Thanks to it we didn't have any problems with finding some examples of neural network or solution to problems. Out network is made from scratch but we used some libraries to help us. The most helpful was numpy which we used on nearly every computing. It helps with generating arrays, making some calculation on them, saving to file and also reading from it. Other libraries we used to create CLI and reading images and parsing to arrays.

As input, we used images downloaded from open source website which we downscale to size 32x32 px. We had one hidden layer, which size could be manipulated by CLI options. Also, the size of the output layer could be changed - a number of neurons in the output layer indicate how many signs we can recognize. To teach network we used backpropagation algorithm with sigmoidal activation function.

Internal Specification

Our solution code is divided into 4 files:

A. network.py - file is responsible for communicating with user in terminal. It creates possible arguments for command line and also validates them depending on picked mode by user (learn network or check). Also in this file results of learning (hidden weights, output weights and errors during learning) are saved to csv files with appropriate name.

B. generate_weights.py - it is simple function, which generates random initial weights for learning purpose. Both hidden and output weight are returned in tuple.

C. learn_network.py - file which contain main class used for learning - LearningNetwork. This class in constructor need following information:

- number of neurons in input, hidden and output layer
- initial weights for hidden and output layer
- eta - value for learning rate
- acc - accuracy for learning

Other function in this class:

1) calculateTotalNetworkError - calculates mean square error for network
2) calculateError - calculates error for output and hidden layer using backpropagation algorithm. Both errors are returned in tuple
3) setExpectedResult - return vector with valid result based on image number

4) getImagePath - get proper image based on image number

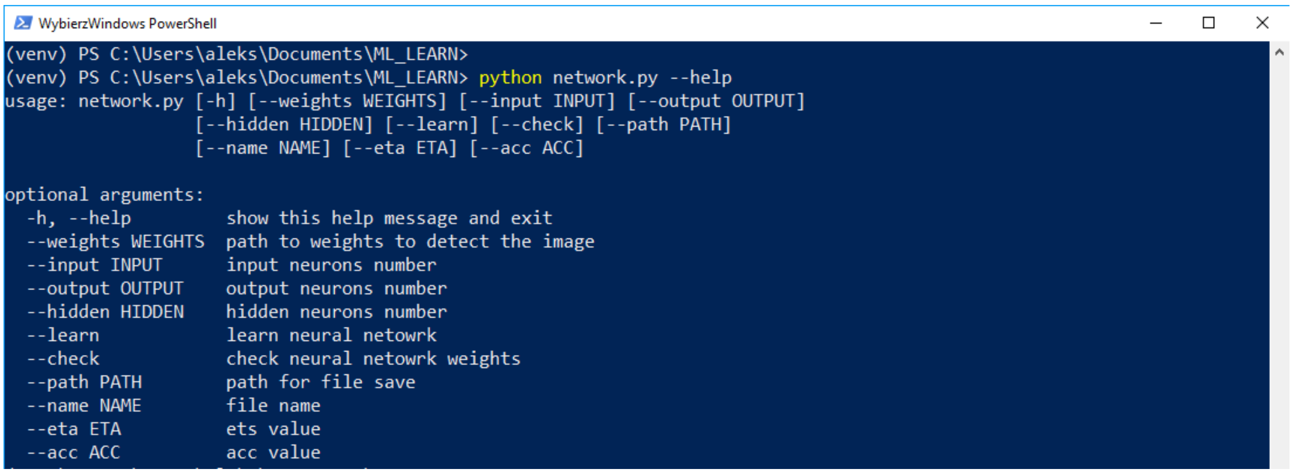
5) learn_network - main method which learn network. In each episode we learn network wich images for all classes. It takes image number (which is equal to learn class number) and based on it took appropriate file. Next based on image and current weights, hidden and output weights are calculated. Next error on output is calculated and error for hidden and output weights. At the end weights are updated based on calculated errors and eta factor. After single epoch there is calculated error for whole network and it checked if error is small enough to finish learning. Network is learned using about 80% of all images

D. recognize_image.py - like generate_weights it is file with single method to calculate output neurons based on input neurons and weights. Neurons are calculated using sigmoidal function

E. check_network.py - class which is used in testing learned network. It is very similar to learn_network class. Base method is check_network which check all remaining images (about 20%) which was not used to learn network so images are unknown. Also there are collected data about number or recognized images, badly recognized images and relation between them for statistic purposes.

External Specification

Program can be run using CLI. There need to be installed libraries for Python: Pillow, numpy, scipy. Next we can run command with python network.py with additional arguments. To see all arguments with description we can use command python network.py —help



```
(venv) PS C:\Users\aleks\Documents\ML_LEARN>
(venv) PS C:\Users\aleks\Documents\ML_LEARN> python network.py --help
usage: network.py [-h] [--weights WEIGHTS] [--input INPUT] [--output OUTPUT]
                  [--hidden HIDDEN] [--learn] [--check] [--path PATH]
                  [--name NAME] [--eta ETA] [--acc ACC]

optional arguments:
  -h, --help         show this help message and exit
  --weights WEIGHTS  path to weights to detect the image
  --input INPUT      input neurons number
  --output OUTPUT    output neurons number
  --hidden HIDDEN    hidden neurons number
  --learn            learn neural netowrk
  --check            check neural netowrk weights
  --path PATH        path for file save
  --name NAME        file name
  --eta ETA          ets value
  --acc ACC          acc value
```

During program run there are printed informations about current epoch, image and errors in recognition.

Example command for learning network

```
network.py --learn --input 1024 --hidden 500 --output 10 --path
„./path/” --name results --eta 0.2 --acc 0.1 --weights „./path/”
```

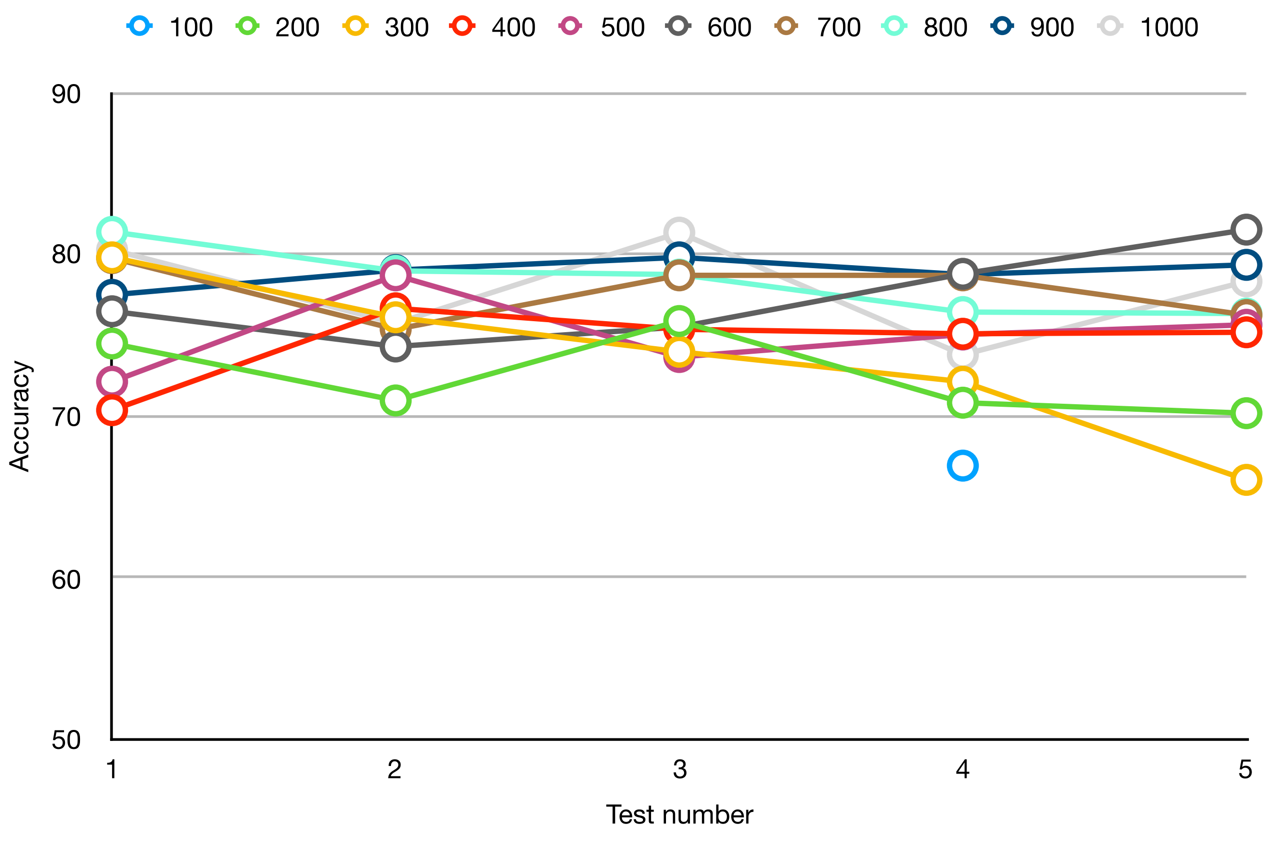
3. Testing and results

In our tests we took into account different number of neurons in hidden layer and different value of learning factor. All test was took on set of digits from 0 to 9. After all our tests we gathered almost 2GB of data. Every test we made few time to have enough statistical data. All tests were conducted on a set of images with Arabic numbers

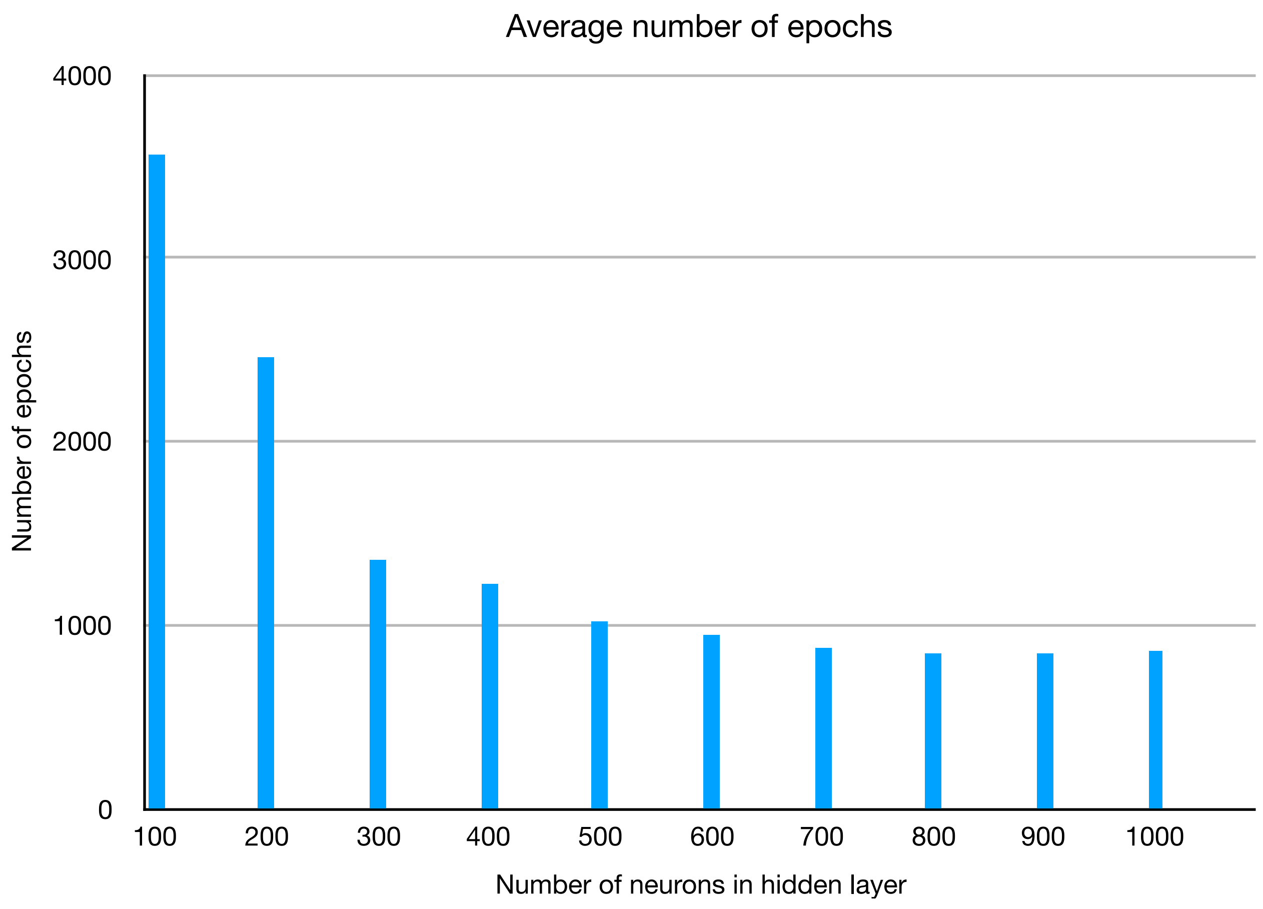
Different number of neurons in hidden layer

We tested from 100 to 1000 neurons in hidden layer, with step 100. For every step we made 5 test. For 100 neurons only one test learned network properly.

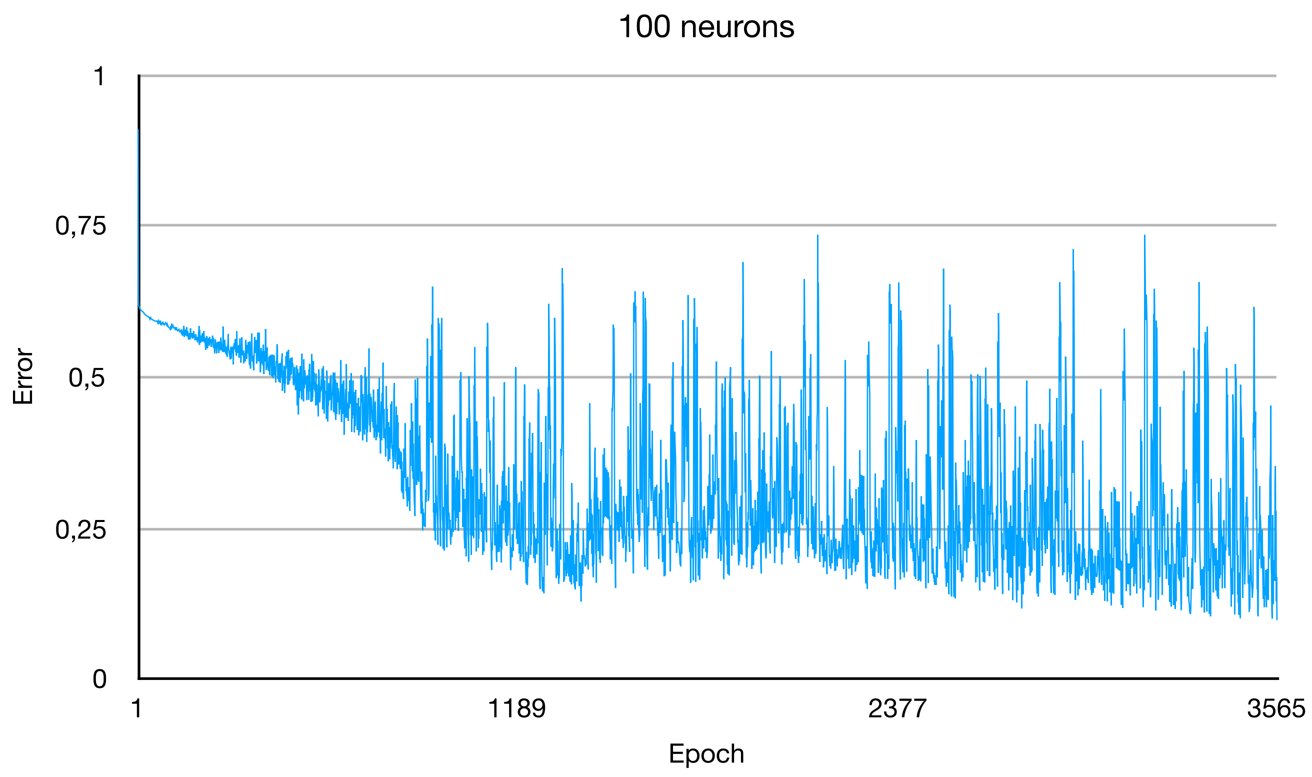
Accuracy was between 66% to 81%. We didn't observe strong depends between number of neurons in hidden layer and accuracy. Below charts show our results.

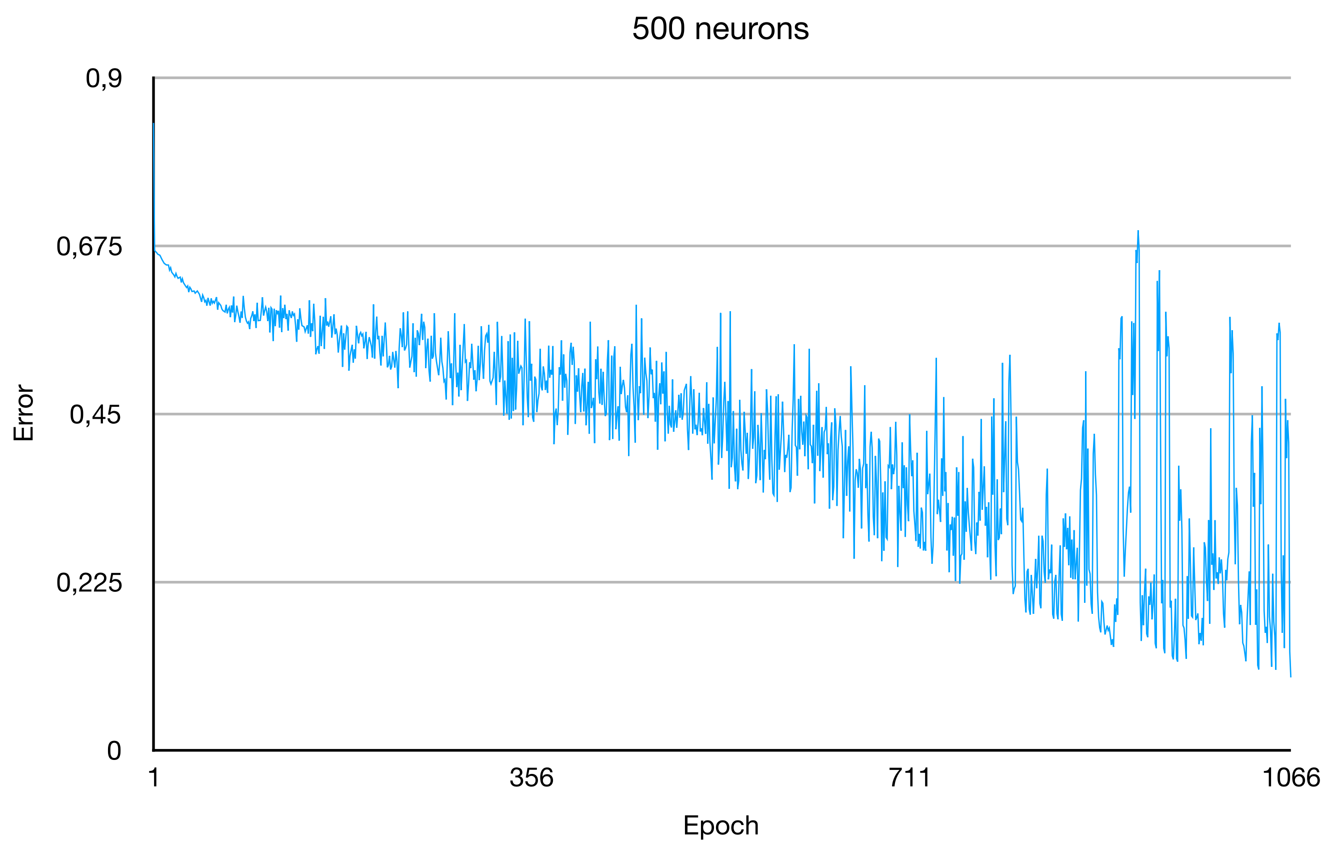


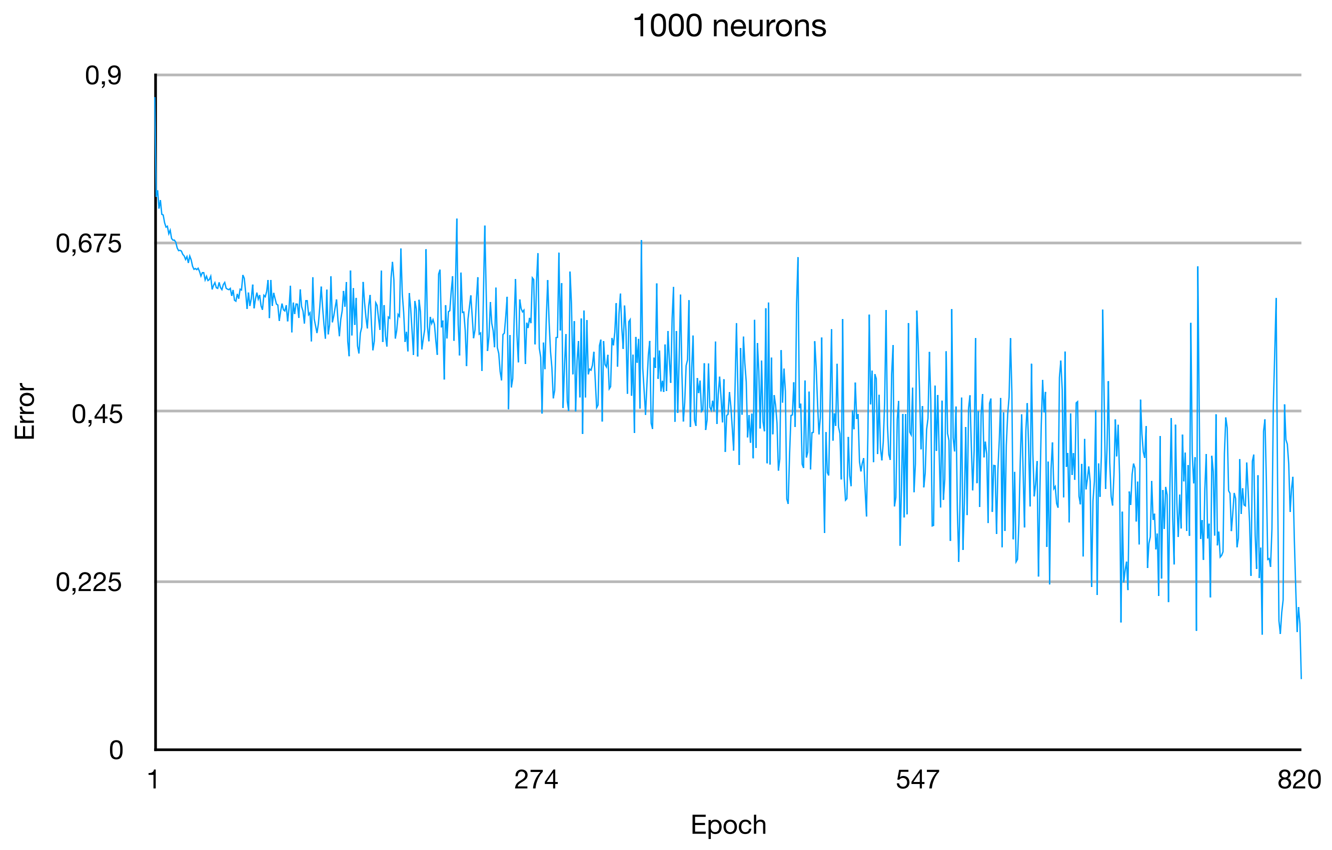
We also check how number of neuron affect on number of epochs. The dependency was as we expected.



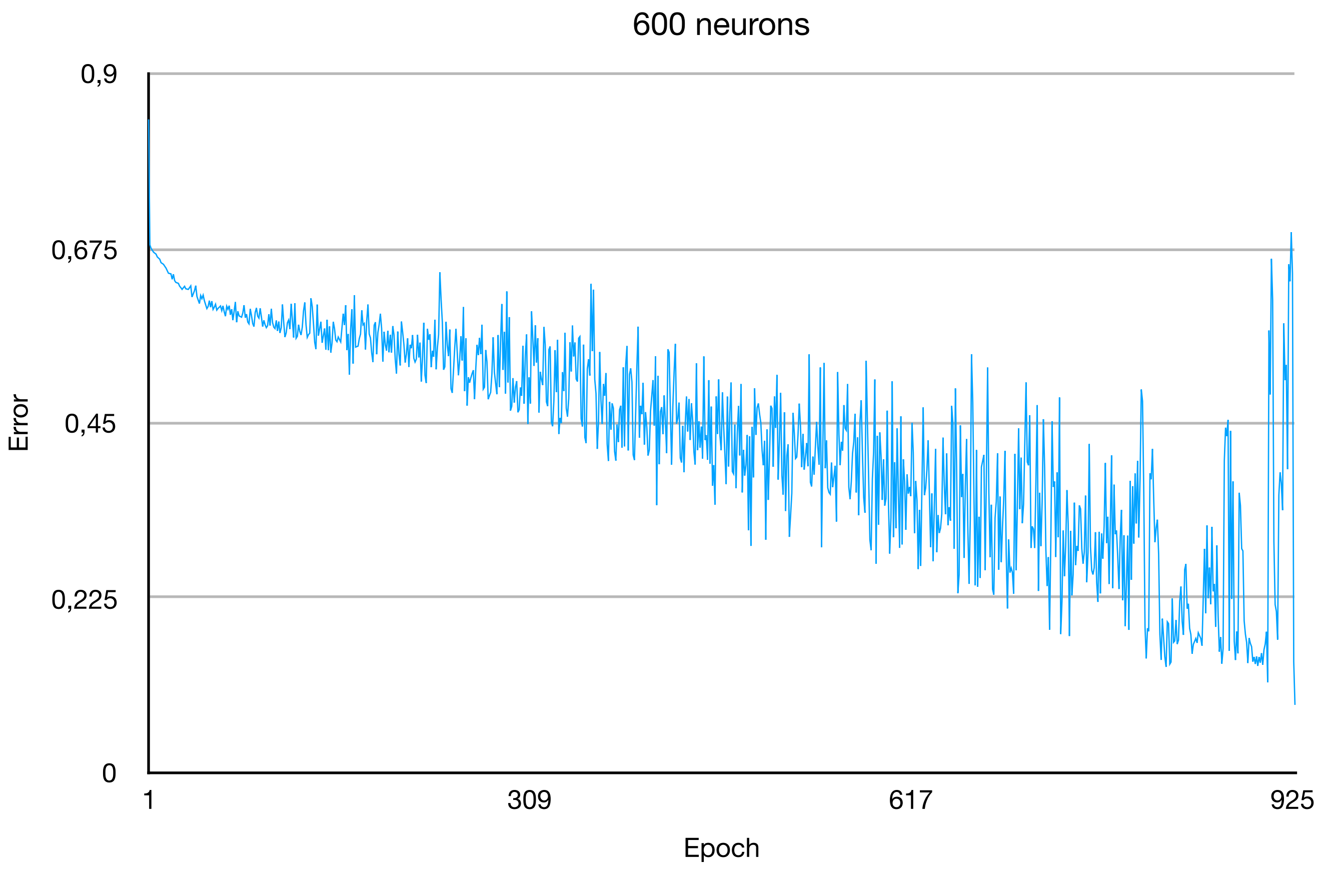
Finally we check value of error after each epoch. Here are charts of error vale after few tests.




500 neurons
0,9
0,675
0,45
0,225
0
Error
1
356
711
1066
Epoch


1000 neurons
0,9
0,675
0,45
0,225
0
Error
1
274
547
820
Epoch

Error chart for our best test:



Conclusion

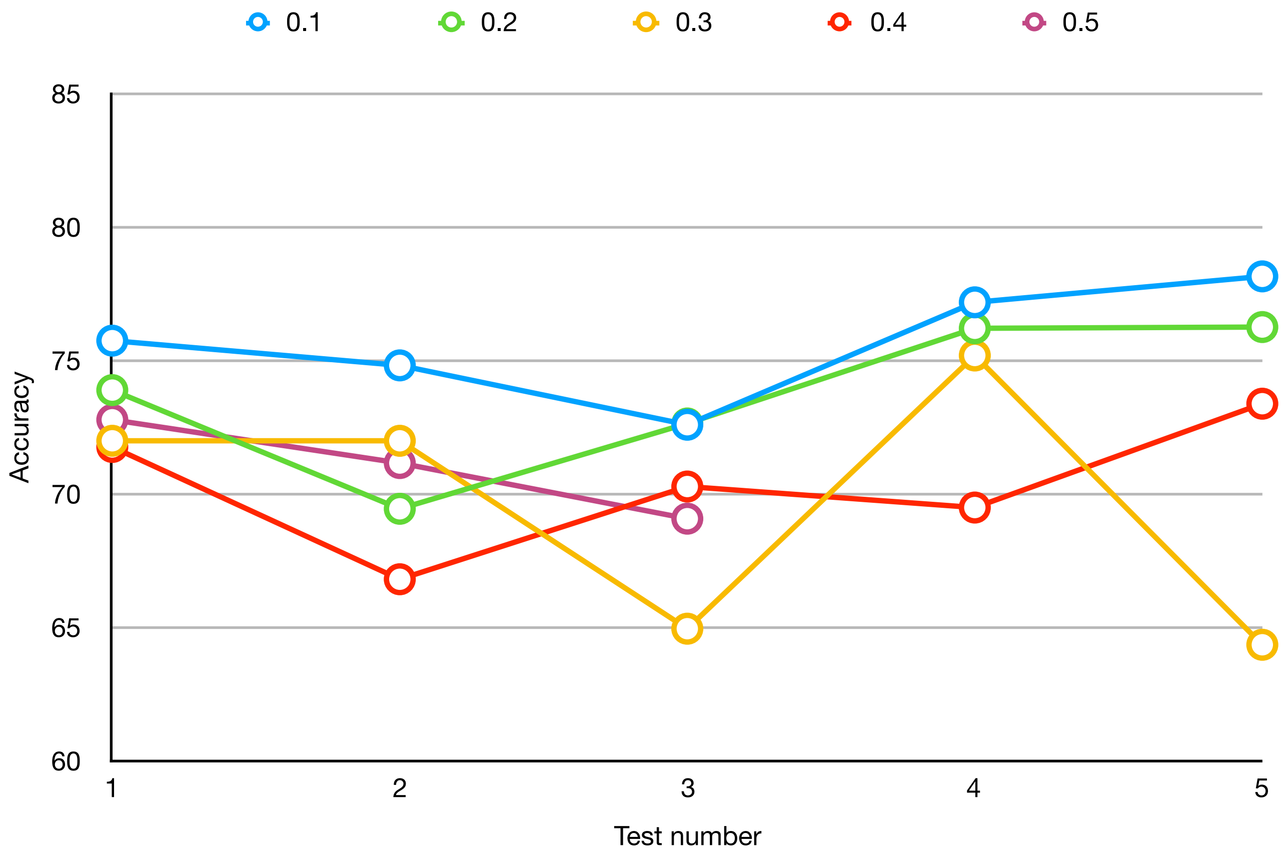
Result for different number of of neurons in hidden layer was easy to predicate. We notice that the accuracy results fluctuated between 70 and 80 percent. Accuracy is not dependent from number of neurons because when there are less number of neurons then learn process takes more time.

However, the number of neurons has a greater impact on the length of study time. When we have more neurons the learning process takes less time, but there is a limit beyond which an increase in the number of neurons does not give such a significant acceleration of the process

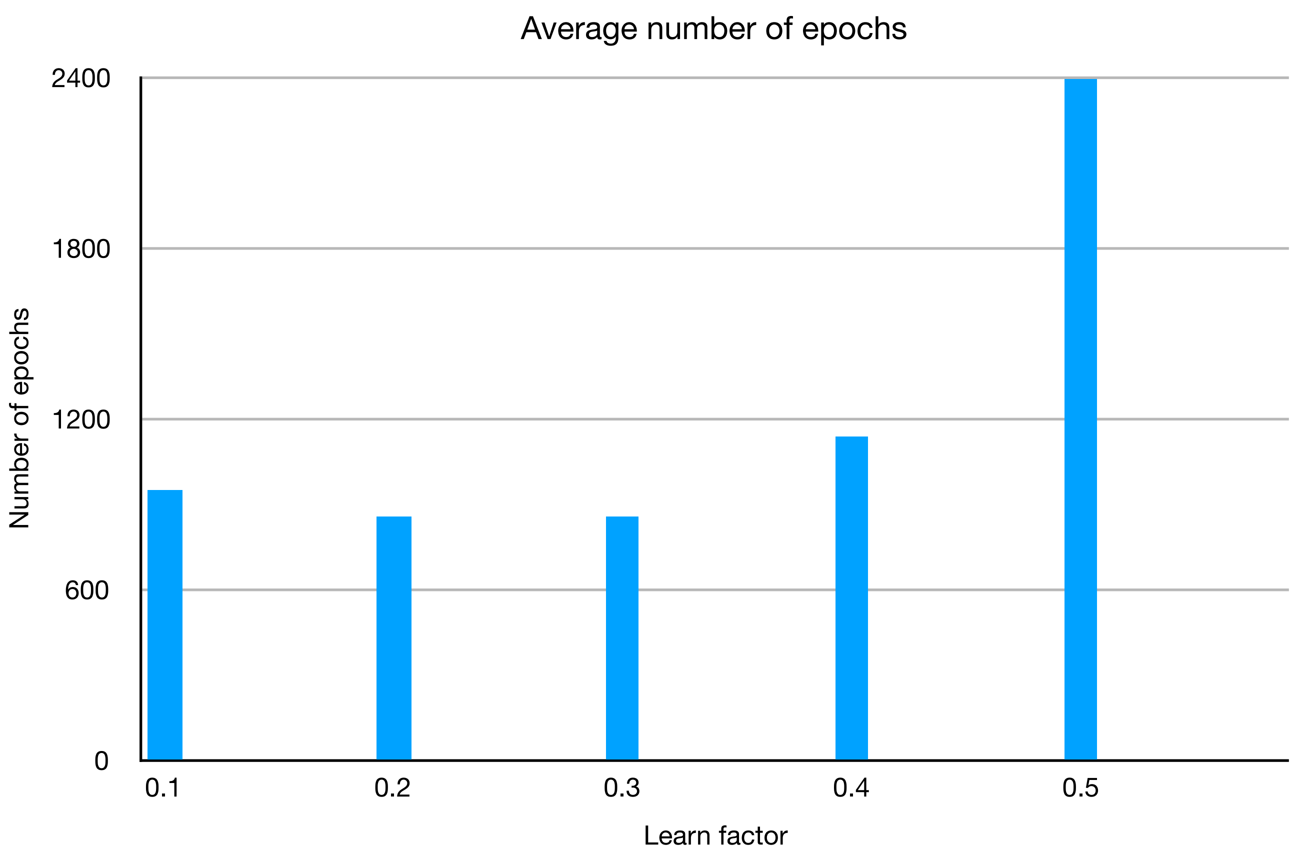
Error behavior is very little depend from number of layers. However the greater the number of neurons does not guarantee minor errors. In our case we noticed that the best results we get for about 500-600 neurons.

Different learning factor

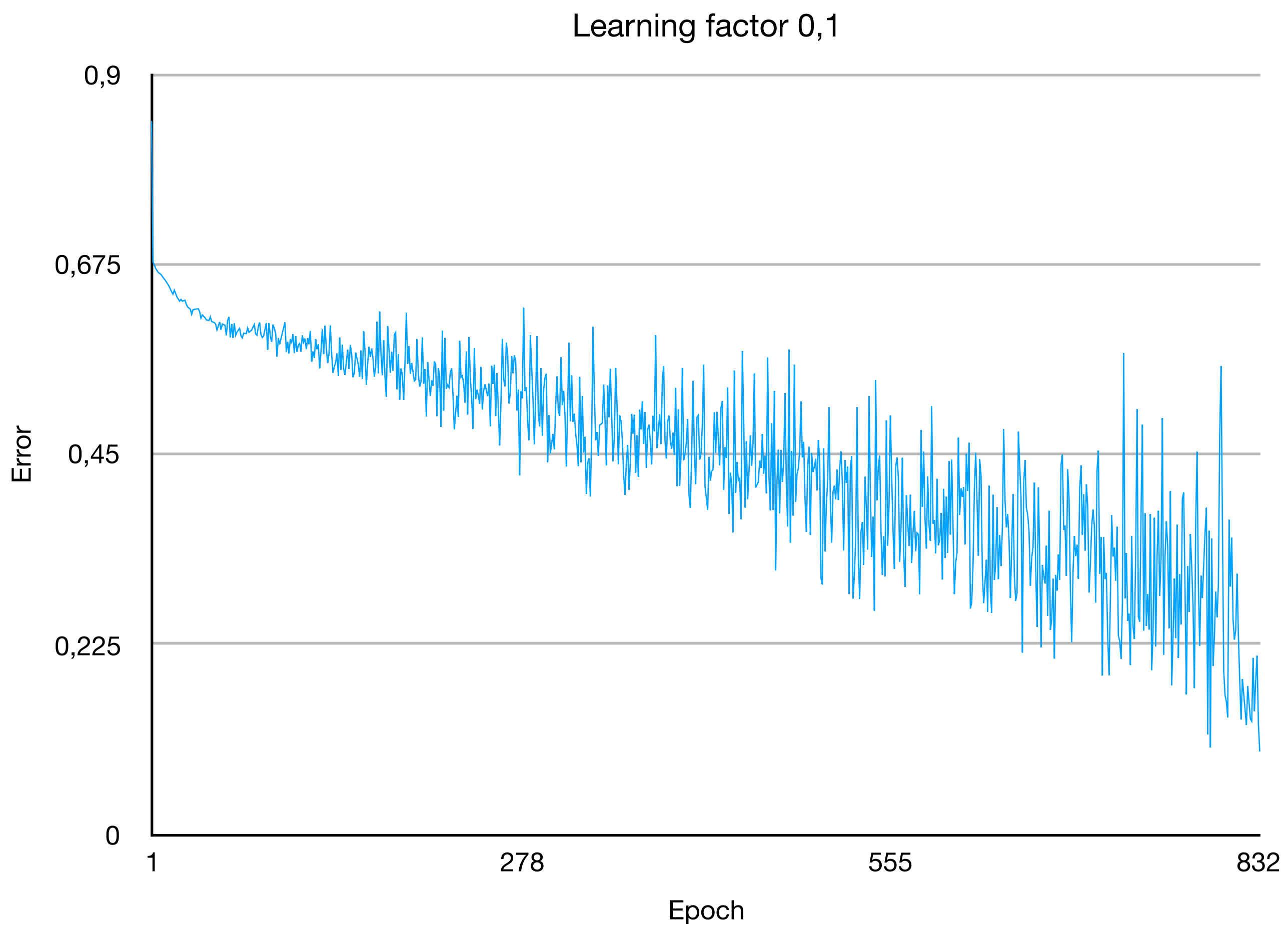
For different learn factor we carried out the same tests as for different number of neurons. Test started from learn factor equal 0,1 and rise grow with a step 0,1. However when we reached value 0,5 we managed to learn the network three times and for greater value learning process always fail.

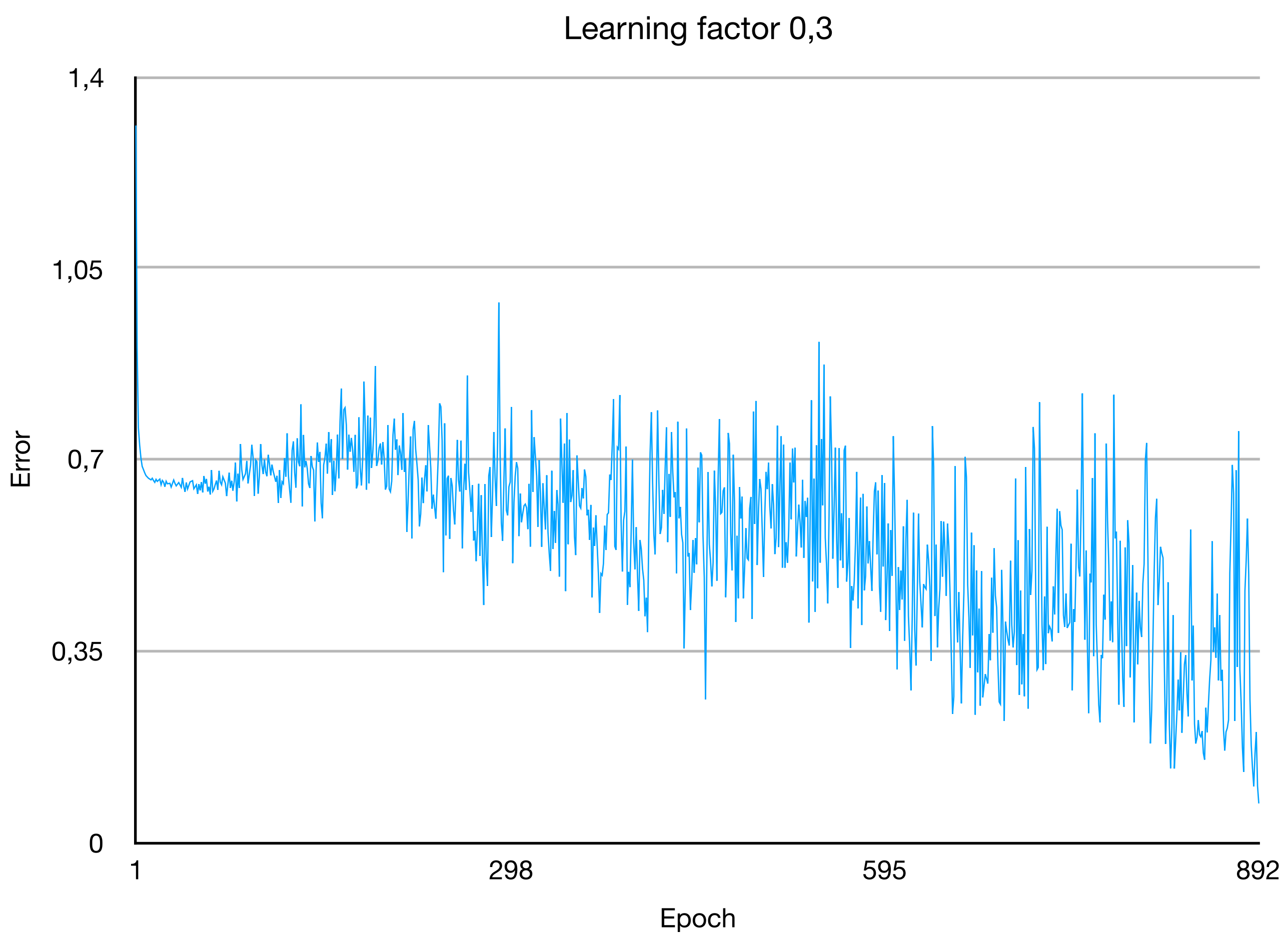


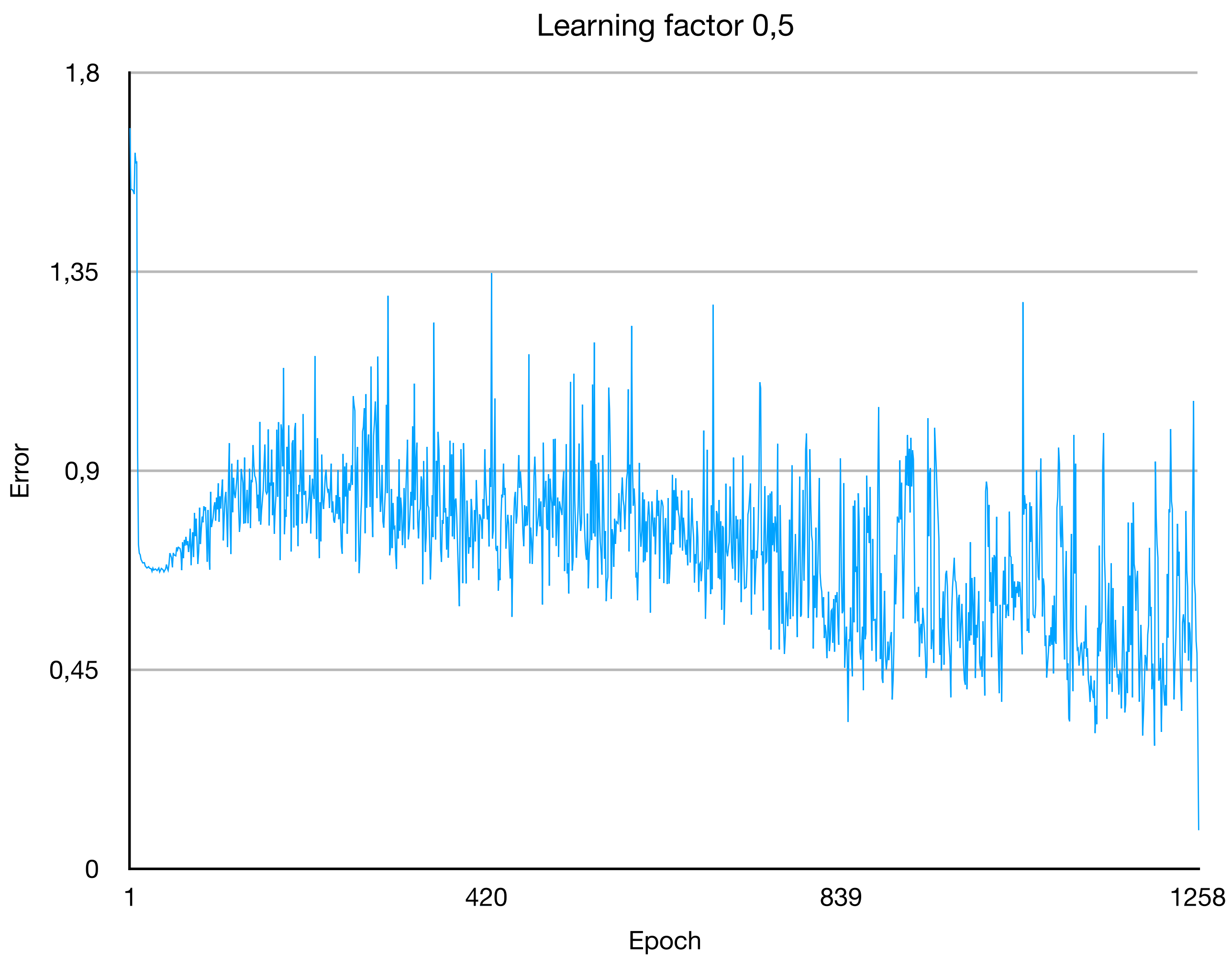
As we can see learning factor affects more on the accuracy than number of neurons. Next chart show that number of epochs depends inversely from learning factor.



Sample errors charts for different learn factor.




Learning factor 0,3
1,4
1,05
0,7
0,35
0
Error
1
298
595
892
Epoch


Learning factor 0,5
1,8
1,35
0,9
0,45
0
Error
1
420
839
1258
Epoch

Conclusion

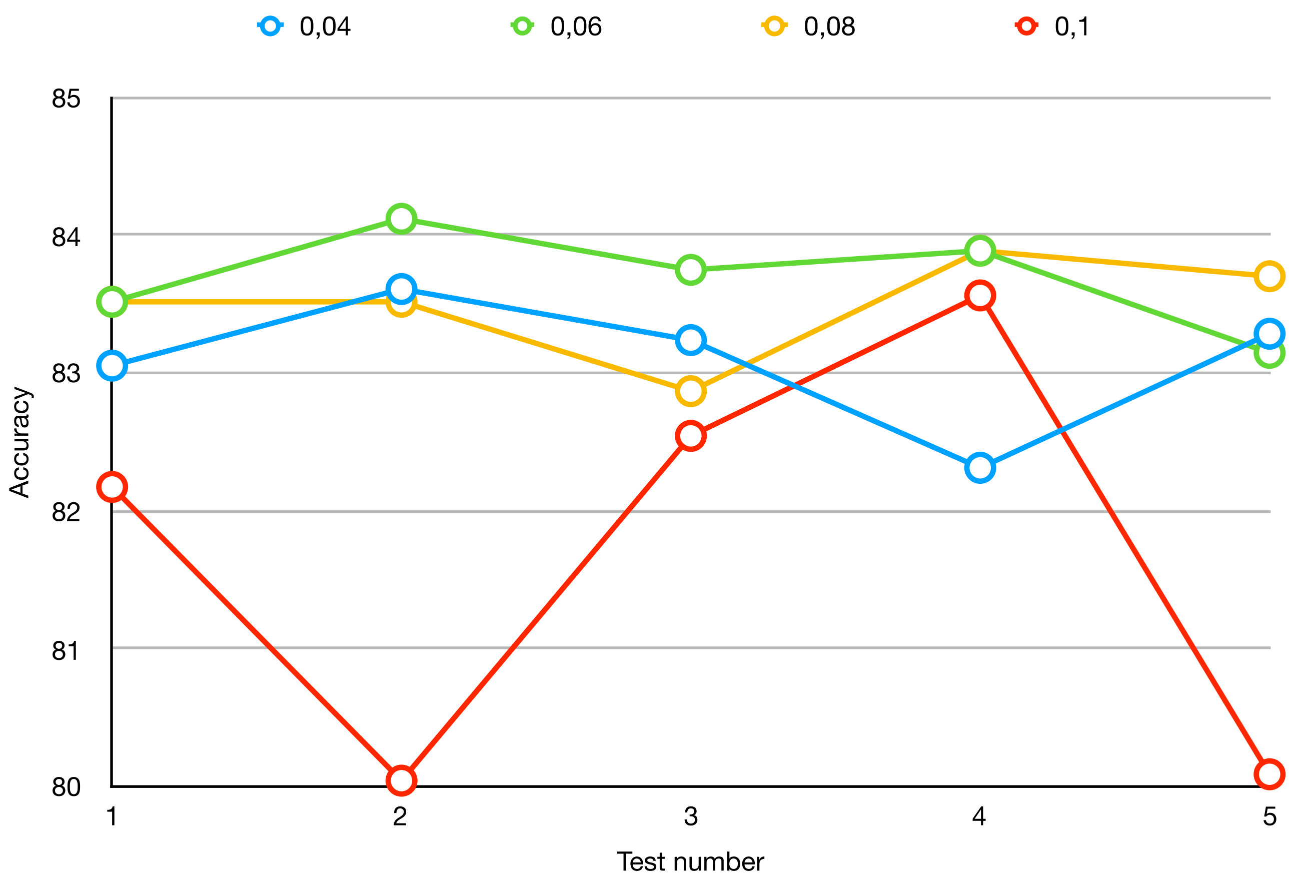
Learning factor has a greater impact on the results than number of neurons. We can observe that when we increase factor then accuracy drop down. Our the best result we get when factor was equal 0,1.

Also number of epochs increase when the factor is increasing. This may be due to too much impact of the backward error propagation, because it is more difficult to achieve the conditions for ending the learning process.

Error chart show very well that the higher the factor, the greater the fluctuation of errors. This confirms the thesis about too much influence of backward error propagation. More doesn’t mean better.

An attempt to improve accuracy

When we end first phase of tests then we chose the best cases for different numbers of neurons and different values of the factor. In our case it was 500 neurons and 0,1 learning factor. We carried out one more learning process for weights which are result of that combination. We made series of test for different learning factor from 0,04 to 0,1. The best result of accuracy we've achieved was 84% for 0,06.



Tests on set of 20 signs

We tried to learn network to recognize set of 20 different signs (digits from 0 to 9 and first ten alphabet letters) but only in few tests we were able to fully learn the network. We tried different value of factor and number of neurons I hidden layer. In all tests which passed, the best accuracy which we archive was 73% for 800 neurons and 0,14 learning factor.

4. Final conclusion

Despite our network work for a small number of output neurons, for bigger numbers it was slow and sometimes it was impossible to learn it. Above 20 output neurons, we weren't able to teach network in less than 5000 epoch which was our upper limit. Single learn for 20 output neurons last about 2-2,5h and sometimes was not successful.

The problem of neural networks is very interesting. We learn a lot of useful artificial intelligence. It can be can be widely used e.g. diagnosis of diseases, autonomous vehicles and inteligent assistant like Siri or Google assistant. We think it will be the most growing branch of technology in next few years.