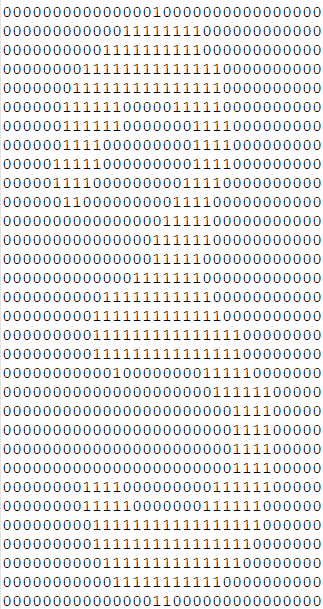
**Digits Recognizing Project Report**

from Valuiev Ruslan

**Description**:

Neural Network gets a bit picture ( 0 or 1 ) and it must recognize which number on the picture. Original picture contains bitmap 64x64, but it compressed to array which length 64. It contains numbers from 0 to 16.

Example or original picture:



Example of compressed data

0,2,15,15,6,0,0,0,0,0,10,13,16,5,0,0,0,0,0,2,16,9,0,0,0,0,0,3,16,11,0,0,0,0,0,0,13,14,1,0,0,0,0,0,7,16,5,0,0,1,4,6,13,15,1,0,0,3,15,14,11,2,0,0

**Neural Network Model:**

Network consists of 64 inputs, hidden layer 45 perceptrons and output layer 10 perceptrons. Each output perceptron corresponds to number from 0 to 9.

**Algorithms:**

1. Data normalization

Input data it is numbers from 0 to 16, so I used this way of normalization:

**Xi = | (Xi – Avg) / σ |**

1. Activation function

As activation function, I have chosen unipolar sigmoid:

**f(x) = 1 / (1 + e-x)**

1. Backpropagation algorithm

Weights are adjusted by the formula:

**Wij = Wij + LerningRate \* Errorj \* xi \* f’(xj)**

1. Initial weights

Weights are randomly generated **range (0.1,0.5)**

**Experimental Results**

After 10 000 generation neural network can recognize training data is **90%.** The hardest classes to recognize is **4** and **7**

**Class 4 –** 88%

**Class 7 –** 87%

Using test data, it can recognize is **88%.** The most complicated classes are **1** and **4**.

**Class 1 –** 83%

**Class 4 –** 90%

**Class describing:**

1. **App** – the main program class which reads data, creates and trains neural network
2. **Methods**:
3. readData(); - reads data from given path
4. **NeuralNetwork** – class which represents neural network. Contains all perceptrons layers as a double array of double. Each perceptron it is array with the length of 2,

[0] – NET

[1] – error

Also, this class contains arrays of weights and learning rate. (unchangeable for all generations)

1. **Constructor** gets amount of perceptrons of hidden layer and learning rate, also it creates all arrays and fill weights with random numbers range [0.1:0.5]
2. **Methods** :
3. changeWeights(); it is changing weights after each training
4. findNETs(); it is calculate NETs on all levels
5. normalize(); it is normalize input data.
6. recognize(); it is method to recognizing
7. showWeights(); debug method to show all weights ( never used)
8. showNETs(); debug method to show all NETs ( never used)

**Conclusion**

As for me 10 000 generation with learning rate 0.002 is too much, but the result worth it but as far as I consider 88% it is very good result.