Silesian University of Technology
Faculty of Automatic Control, Electronics and Computer Science

Gliwice, 7.02.2017

Computer Programming Laboratory Project Report

Author:Maciej Sowiar
Tutor: Jakub Nalepa, Ph.D.

1. **Task topic**
   Neural network.
2. **Project analysis**
   In this project i created flexible neural network. We may easily add new layers and more neurons to our network depending on our expectation. I Used several classes and everything is based on vectors, every layer and all connections. Neurons are stored in layers vectors.
3. **Internal specification:**
   Neural network has four classes:

-class Connections - It is used to represent connections between neurons.
It contains private variable named weight, it contains weight of specific connection.

-class Neuron - representation of neuron i network. Contains three private variable
Input, and output of neuron and propagated error which is used to calculate the error
Which we are decreasing through our network until the moment we obtain so small
error so we can stop our calculations.
We also got method calculated derivative which is set to calculate derivative of sigmoid function. This only one method despite of set and get methods.

-class Layer - It represents layers in my neural network. Vector<Neuron> layer_of_neurons, contains neurons on specific layer. In constructor of this class, we pass number of neurons which will be created.

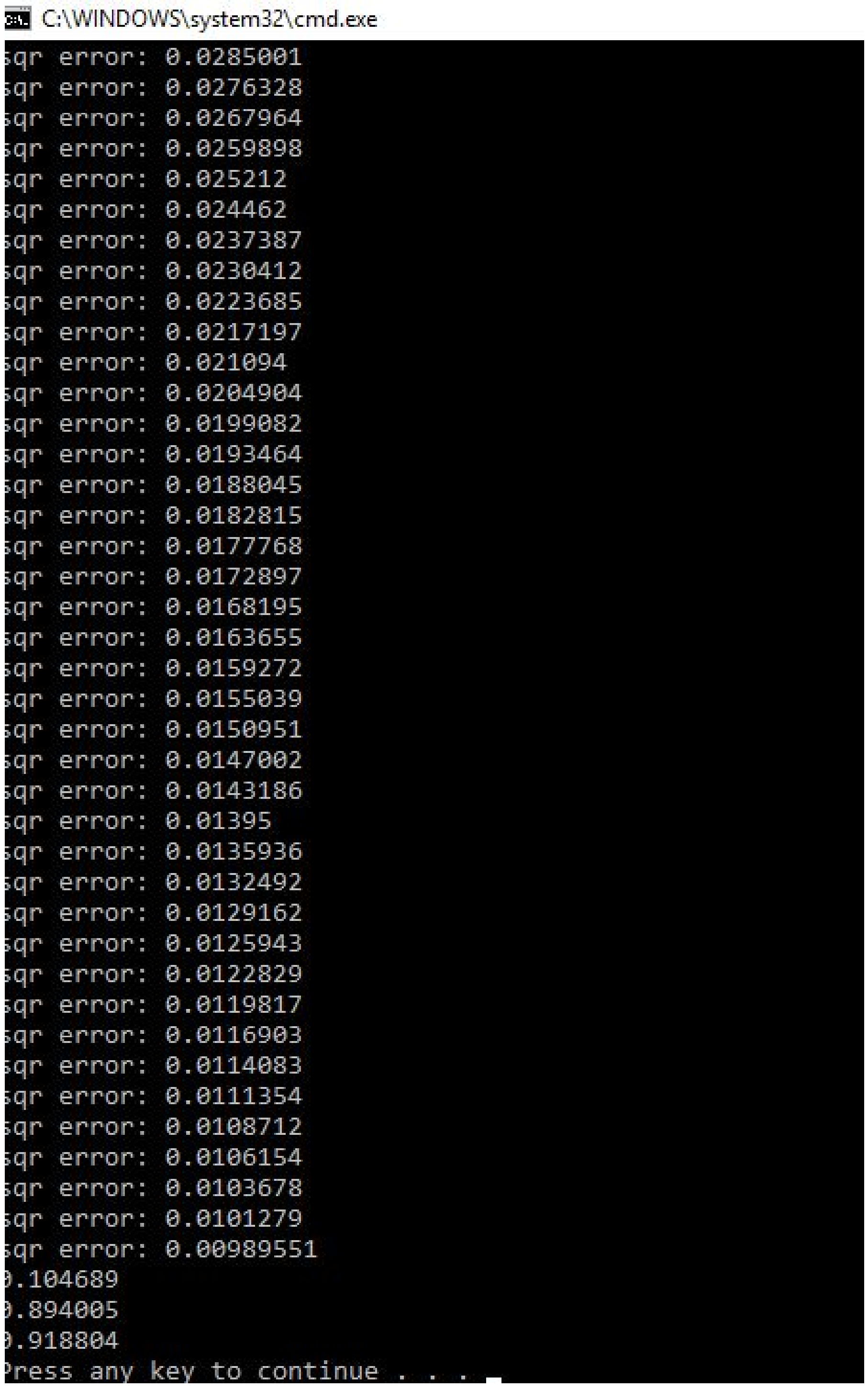
-class Net- main class in this project. It is used to create topology of network with usage of vector named topology, and with usage of constructor. Basing on numbers which we pass in main function it creates input layer with number of neurons which are pushed back to topology vector. Next call of push back will create next layer with other number of neurons, and so on.
Vector layer is used to represnt layers, and two dimensional vector named connections is representing number of connections between layers, second dimension stands for the index of these connections. For example between first and second layer we got index [0] of this connections.

- Method create_layers - it creates input layer with given number of neurons and adds bias neuron to it, then it iterates until topology size using for loop and we don’t perform this for input and output layer, to all hidden layers we add also bias neurons. Output layer is added manually without bias.
- Method create_connections - first of all we create temp vector of conections, than we are iterating untill topology.size - 1. In this for loop we create second one which is supposed to iterate and create number of connections equal to product of number of neurons in two neighbour layers, and also asigning random values to all connections in moment we create them. Then we pass all temp vector to class connections using set method.
- Feed_forward - method is calculation which is used to calculate through our network in forward direction. We are starting from first hidden layer, and we get weight of first connection and multiply it by the output of neuron from previous layer. We do it for all

weights between all neurons and sum up all single inputs we obtain. We pass our sumed up input to activation function and set it as output of our neuron.
At the end we reset all inputs in our network analogically set all inputs to 0.
- Backprop_lastlayer - we perform it calculating through every single neuron at the output layer. We just simply calculate our propagated error error using our desired_output value minus neuron output multiplied by its derivative..
- Backprop_hiddenlayers - in case of this method we are supposed to iterate through everysingle neuron in the rest of layers. First of all we are supposed to create several iterator one represents index of connection the second one is used for index of vector of connections. We are calculating variable epsilon which is equal to it previous value plus propagated error which we calculated in previous method times weight of connections connected to this neuron. This calculation we perform iterating through from the back of network (starting from next to the last layer) for every neuron. Than we are moving to the next neuron in previous layer. We calculate now our error which is equal to epsilon which we calculated above times the neuron derivative, and add this error to our propagated error. After this calculations we proceced to next two layers and again calculate, first we set epsilon equal to 0 and propagated through all neurons at right layer and we move to next neuron at left layer and we follow we the same operations.
- Update_weights - As in previous method we iterate through every single connection this time and set it weight equal to previous weight + 2 times our learning rate times our propagated error and times output of previous neuron. At the end we reset propagated error.

4.Test

To test my network i used operations on xor gate i created performed operations for four set of inputs [1,1], [1,0]. [0,1] and [0,0] and one output neuron which was 1 or 0 based on xor gate operations.
For three learning set of inputs and network Containing two input layer neurons (+bias) one hidden layer with two neurons(+bias) and output layer with 1 neuron



```
sqr error: 0.0285001
sqr error: 0.0276328
sqr error: 0.0267964
sqr error: 0.0259898
sqr error: 0.025212
sqr error: 0.024462
sqr error: 0.0237387
sqr error: 0.0230412
sqr error: 0.0223685
sqr error: 0.0217197
sqr error: 0.021094
sqr error: 0.0204904
sqr error: 0.0199082
sqr error: 0.0193464
sqr error: 0.0188045
sqr error: 0.0182815
sqr error: 0.0177768
sqr error: 0.0172897
sqr error: 0.0168195
sqr error: 0.0163655
sqr error: 0.0159272
sqr error: 0.0155039
sqr error: 0.0150951
sqr error: 0.0147002
sqr error: 0.0143186
sqr error: 0.01395
sqr error: 0.0135936
sqr error: 0.0132492
sqr error: 0.0129162
sqr error: 0.0125943
sqr error: 0.0122829
sqr error: 0.0119817
sqr error: 0.0116903
sqr error: 0.0114083
sqr error: 0.0111354
sqr error: 0.0108712
sqr error: 0.0106154
sqr error: 0.0103678
sqr error: 0.0101279
sqr error: 0.00989551
.104689
.894005
.918804
ress any key to continue . . .
```

This is last part of calculations, we can see that error was decreasing properly and at the end we obtained output 0.1 for desired 0 , 0.89 for desired 1 and 0.91 for desired 1. So the results were almost equal to our desired output.

In case of four learning sets of inputs squared error mostly stopped and could not decrease in next iterations. Several times i managed to prepare such number of neurons in hidden layer, that squared error was lower than 0.01 but sadly at this moment i do not have any screenshots.