Gliwice, 23.06.14

Semester: II

Group: 3

Section: I

Computer Programming

Neural network

Author: Grzegorz Mrukwa

Tutor: Piotr Fabian

1. Task topic

Neural network. Neural networks simulate the behaviour of a human brain and are able to classify patterns after presenting some examples to them. Write a program that simulates a neural network. The user prepares a set of training data (examples, inputs and desired outputs), runs the "training", stores the trained network in a file and uses the network to recognise new patterns. Details to discuss.

My program is focused on problem of prediction secondary structure of proteins, based on their primary structure. Training files for such network should be organized in such manner:

```
> <code>

<amino acids symbols>

<secondary structure types symbols>
```

This may be repeated as many times as necessary in used file.

Second utility is recognition of such secondary structure of protein based on learned patterns. Input file for such operation should contain only one line with protein described by its amino acids' symbols.

2. Project analysis

First conclusion based on type of problem to solve is that we will need nonlinear neural network, having at least several hidden layers for good approximation of solving pattern. Network with no loops seems to be enough for such considerations, but backward loop could give us better results – we may consider it later on.

In program there are used neurons with so-called 'logistic' function of activation:

$$y = \frac{1}{1 + e^{-\beta s}}$$

Where:

- β is an arbitrary coefficient changing graph of such function (in program it is just equal to 1)
- s which is linear combination of neuron's inputs:

$$s = \sum_{i=0}^{n-1} w_i x_i$$

- w_i is wage connected with i-th input (counted from 0)
- x_i is signal at i-th input (also counted from 0)

Learning of such network is realized by simple back propagation algorithm. We may also use modified version of it to ensure higher speed of learning, which seems to be very long.

3. External specification

Program may be run by clicking its icon in File Explorer. Then it prints out the help message to inform how to use it:

```
1. create - creates new neural network
2. select x - selects network number x
3. list - prints names of all networks
4. teach - teaches selected network
5. save - saves selected network
6. epochs - sets the number of epochs used in training
7. report - sets the number of epochs when report comes
8. read from file - creates new network from selected directory
9. work - uses network to work on single case
10. help - displays this message
11. quit - closes program
```

For example: if we want to read previously saved neural network from file, we write `8` and confirm it by pressing `Enter` key, then we are asked for neural network directory name.

Program grants no error coverage in this field – user has to be sure that every data passed to program is correct, otherwise it may fail.

If we are creating new neural network manually, we have to remember, that first layer of all parts has to be `171` neurons big (considered window of amino acids is `9` elements wide and we have `19` types of amino acids used, coded in 1-out-of-19 code on input), output layer of first and third part has to have `12` neurons (considered `4` places from the beginning or from the end of secondary structure coded in 1-out-of-3 code on output). Second part has to have `3` neurons at the output as it considers only one place in the middle at once.

Using `work` module we have to remember, that result is printed in file in following way:

- Each row contains description of one place in protein chain
- Each row contains 3 numbers – they correspond to the type of secondary structure. First corresponds to `C`, second to `E`, third to `H`. The greater number on such position, the bigger probability, that it is the proper local structure of considered protein.

4. Internal specification

- `Main.cpp` contains:
  - `int main(){}` – main function of a program. Creates object of type `Program` and runs it.
  - `class Program{};` - class performing all actions user wants to do
    - `void Run(){}` – runs the program in the way user wants to

- `NeuralNetwork.h` contains:
  - `class NeuralNetwork<FloatingNumber>{};` – template class of neural network
    - `NeuralNetwork(const vector<int>& structureDescription){}` – creates new network according to the structure given as the parameter. Adds one bias neuron in every hidden layer.
    - `NeuralNetwork(const string& filename){}` – creates network from file
    - `vector<FloatingNumber> Use(const vector<FloatingNumber>& input){}` – returns a result of network's work on a given data
    - `void Teach(TeachingSet& teachingSet, int epochs){}` – teaches network on a given teaching set for the determined number of epochs
    - `void Teach(TeachingSet& teachingSet, int epochs, int whenReport){}` – teaches network on a given teaching set for the determined number of epochs printing statistics of average error after each `whenReport` epochs
    - `void Teach(TeachingSet& teachingSet, FloatingNumber error, int whenReport){}` – teaches network unless average mean square error is lower than `error`, printing report after each `whenReport` epochs
    - `void Save(const string& filename){}` – saves network to file
    - `typedef list<pair<vector<FloatingNumber>,vector<FloatingNumber> > > TeachingSet` – defines type of teaching set for the network
- `Neuron.h` contains:
  - `class Neuron<FloatingNumber>{};` - template class of neuron
    - `vector<FloatingNumber> wages` – wages used to calculate linear combination of inputs
    - `FloatingNumber lastResult` – last result of work of neuron. Used by `NeuralNetwork<>` class while teaching
    - `enum ActivationFunction type` – type of neuron activation function
    - `Neuron(int size){}` – standard constructor for `Neuron<>` class
    - `Neuron(){}` – no parameter constructor defined for ability to use `vector<Neuron<> >`. Should not be used anywhere else.
    - `Neuron& operator=(const Neuron& neuron){}` – standard copying operator
- `Parser.h` contains:
  - `Parser<FloatingNumber>{};` - template class for parser. `FloatingNumber` should be the same type that neural network is.
    - `Parser(){}` – standard constructor for parser with window length equal to 9
    - `Parser(int windowWidth){}` – constructor which allows user to set width of window considered
    - `vector<TeachSet> Parse(const string& inputFilename)` – creates teaching set from file for neural network
    - `vector<vector<FloatingNumber> > ParseSingleInput(const string& inputFilename)` – parses single case from a file to the form used in program
    - `typedef list<pair<vector<FloatingNumber>,vector<FloatingNumber> > > TeachSet` – defines type which the result is given in
- `Types.h` contains:
  - `Enum ActivationFunction{};` - contains implemented types of activation function used in neurons

5. Source code

All the source code is organized in 5 files:

- `Main.cpp`
  Main source file. Here is the mechanics of the program in class `Program`.

- `NeuralNetwork.h`
  Here is `NeuralNetwork<>` class implemented.

- `Neuron.h`
  Here is implemented `Neuron<>` class used by `NeuralNetwork<>` class.

- `Parser.h`
  Header file with `Parser<>` class implementation. It maintains creating teaching sets of neural networks and preparation to recognize single case.

- `Types.h`
  Header file where types of neuron activation function are stored.

6. Testing

For checking the correctness of solution, there were performed some simple tests:

a) To recognize that neural network is implemented properly, there were performed tests to teach a network XOR operation. After several tries and corrections, network was able to match good solution for this problem.
b) To check other things, class `Tester` was implemented (stored in file `Tester.h`) which was checking if the results of other functions were proper.
c) After some initial tries to teach several distinct configurations of neural networks prediction of proteins' secondary structure, results were checked after saving them into file. They seem the same despite different inputs, so they were checked further – usage of breakpoints in Visual Studio allowed to distinguish these outputs as different but with very low difference, what means that many teaching iterations have to be performed.

7. Conclusions

- To improve prediction of secondary structure we may try to use neural network with backward loop or widen considered window of amino acids
- To speed up learning process, we may try following enhancements:
  - Determining corrections of wages for every teaching case in one epoch and at the end changing wages
  - Multi-threading
  - CUDA technology
  - Thinner window of considered amino acids, but this solution decreases correctness of prediction