```cpp
//
// Created by Fabian Moik on 18.12.17.
//

#include "NeuralNet.h"
#include <assert.h>
#include <math.h>
#include <cmath>
#include <iostream>

NeuralNet::NeuralNet(const std::vector<unsigned> &topology)
{
    unsigned numLayers = topology.size();
    for (unsigned layerNum = 0; layerNum < numLayers; layerNum++) {
        layers_.push_back(Layer());

        // Defines the number of needed weight connections for each neuron in each layer
        // if it is a neuron of the last layer, no output connections are needed
        unsigned numOutputs = layerNum == topology.size() - 1 ? 0 : topology[layerNum + 1];

        // We have a new layer, now fill it with neurons, and
        // add a bias neuron in each layer (<= therefore).
        for (unsigned neuronNum = 0; neuronNum <= topology[layerNum]; ++neuronNum) {
            layers_.back().push_back(Neuron(numOutputs, neuronNum));
        }

        // Force the bias node's output to 1.0 (it was the last neuron pushed in this layer):
        // Value of bias is not so important is just has to be != 0
        layers_.back().back().setOutputVal(1.0);
    }
}

void NeuralNet::feedForward(const std::vector<double> &inputVals)
{
    assert(inputVals.size() == (layers_[0].size() - 1));

    // Assign (latch) the input values into the input neurons
    //TODO what does ++i do here instead of i++?
    for (unsigned i = 0; i < inputVals.size(); ++i) {
        layers_[0][i].setOutputVal(inputVals[i]);
    }

    // forward propagate
    // If we want to use softmax for the last layer, the activation needs to be done by the neural network
```

```
45      // because all output values of the neurons need to be know to perform a softmax activation
46      for (unsigned layerNum = 1; layerNum < layers_.size(); ++layerNum) {
47          Layer &prevLayer = layers_[layerNum - 1];
48
49          for (unsigned n = 0; n < layers_[layerNum].size() - 1; ++n) {
50              layers_[layerNum][n].feedForward(prevLayer);
51          }
52      }
53
54      //For the last layer use softmax activation
55      softmaxActivation(layers_.back());
56  }
57
58  //returns a vector of all output weights in the order of the layers, with suborder of neurons in the layer
59  //it is just a string of weights similar to a human DNA
60  std::vector<double> NeuralNet::getOutputWeights() {
61      std::vector<double> outputWeights;
62      for (auto &layer: layers_) {
63          for (auto &neuron: layer) {
64              for (auto &weight: neuron.getOutputWeights()) {
65                  outputWeights.push_back(weight.weight);
66              }
67          }
68      }
69      return outputWeights;
70  }
71
72  // gets a vector of weigths as input and sets it in the same order as they are retrieved when calling getOutputWeights()
73  void NeuralNet::setOutputWeights(std::vector<double> outputWeights) {
74      int index = 0;
75      for (auto &layer: layers_) {
76          for (auto &neuron: layer) {
77              for (auto &weight: neuron.getOutputWeights()) {
78                  if (index >= outputWeights.size()) {
79                      //Something went wrong
80                      std::cerr << "NeuralNet: invalid index for setting output weights" << std::endl;
81                      return;
82                  }
83                  weight.weight = outputWeights.at(index);
84                  index++;
85              }
86          }
87      }
88  }
```

```cpp

// Fills the rusults vector with the values of the last layer's neurons
// TODO differentiate between raise sizes and calculate a desired raise amount
void NeuralNet::getResults(std::vector<double> &resultVals) const
{
    resultVals.clear();

    for (unsigned n = 0; n < layers_.back().size() - 1; ++n) {
        resultVals.push_back(layers_.back()[n].getOutputVal());
    }
}

// This results in the output neurons summing up to 1
// Useful for a classification problem
void NeuralNet::softmaxActivation(Layer &layer) {
    //This is not the original softmax function because I just normalize the output to 1. the real one would use the exp
    // but there are problems when using the softmax with values between 0 ... 1
    double sum = 0;

    // size() - 1 because we don't want the bias neuron
    for (unsigned n = 0; n < layers_.back().size() - 1; ++n) {
        sum += layers_.back()[n].getOutputVal();
    }

    //Normalize values

    for (unsigned n = 0; n < layers_.back().size() - 1; ++n) {
        // round to two digits
        double softmax = layers_.back()[n].getOutputVal() / sum;
        double roundedOwnValue = std::floor(softmax * 100 + 0.5) / 100;
        layers_.back()[n].setOutputVal(roundedOwnValue);
    }
}
```

```cpp
//
// Created by Fabian Moik on 18.12.17.
//

#ifndef OWNPOKERSIMULATOR_NEURALNET_H
#define OWNPOKERSIMULATOR_NEURALNET_H

#include <vector>
#include "Neuron.h"
#include <string>

class NeuralNet {
public:
    NeuralNet(const std::vector<unsigned> &topology);
    void feedForward(const std::vector<double> &inputVals);
    void getResults(std::vector<double> &resultVals) const;
    void softmaxActivation(Layer &layer);
    std::vector<double> getOutputWeights();
    void setOutputWeights(std::vector<double>);

private:
    std::vector<Layer> layers_; // m_layers[layerNum][neuronNum]
};

#endif //OWNPOKERSIMULATOR_NEURALNET_H

```

```cpp
//
// Created by Fabian Moik on 18.12.17.
//

#include "Neuron.h"
#include <cmath>
#include <iostream>
#include <random>

Neuron::Neuron(unsigned numOutputs, unsigned myIndex)
{
    for (unsigned c = 0; c < numOutputs; ++c) {
        outputWeights_.push_back(Connection());
        outputWeights_.back().weight = randomWeight();
    }

    myIndex_ = myIndex;
}

void Neuron::feedForward(const Layer &prevLayer)
{
    double sum = 0.0;

    // Sum the previous layer's outputs (which are our inputs)
    // Include the bias node from the previous layer.

    for (unsigned n = 0; n < prevLayer.size(); ++n) {
        sum += prevLayer[n].getOutputVal() *
               prevLayer[n].outputWeights_[myIndex_].weight;
    }

    outputVal_ = Neuron::activationFunction(sum);
}

double Neuron::activationFunction(double x)
{
    // tanh - output range [-1.0..1.0]
    // use sigmoid in futur? - really slow to compute

    // possible activation functions
    //atan(pi*x/2)*2/pi   24.1 ns
    //atan(x)             23.0 ns
    //1/(1+exp(-x))       20.4 ns
    //1/sqrt(1+x^2)       13.4 ns
```

```
45      //erf(sqrt(pi)*x/2)    6.7 ns
46      //tanh(x)              5.5 ns
47      //x/(1+|x|)            5.5 ns
48      return 1/(1+exp(-x)); //sigmoid
49      //return tanh(x);
50  }
51
52  std::vector<Connection>& Neuron::getOutputWeights() {
53      return outputWeights_;
54  }
55
56  double Neuron::randomWeight() {
57
58     /*
59      * for hyperbolic tangent units: sample a Uniform(-r,r) with r = sqrt(6 / (fanIn + fanOut))
60      * where fanIn is the number of inputs of the unit and fanOut the number of outputweights
61      *
62      * for sigmoid units: use  r = 4 * sqrt(6 / (fanIn + fanOut))
63      *
64      *
65      *  GOOD:    another alternative approach recently often used is (only for sigmoid?? because tanh could output
66      *  negative values aswell):
67      *  U([0,n]) * sqrt(2.0/n) - where n is the number of inputs of your NN
68      */
69
70
71      // TODO find a way to get the number of input neurons
72      std::random_device rd;  //Will be used to obtain a seed for the random number engine
73      std::mt19937 gen(rd()); //Standard mersenne_twister_engine seeded with rd()
74      std::uniform_real_distribution<> dis(-1.0, 1.0);
75      return dis(gen); //Each call to dis(gen) generates a new random double
76  }
77
```

```cpp
//
// Created by Fabian Moik on 18.12.17.
//

#ifndef OWNPOKERSIMULATOR_NEURON_H
#define OWNPOKERSIMULATOR_NEURON_H

#include <vector>

class Neuron;

typedef std::vector<Neuron> Layer;

struct Connection
{
    double weight;
    double deltaWeight; // needed for later?
};

class Neuron
{
public:
    Neuron(unsigned numOutputs, unsigned myIndex);
    void setOutputVal(double val) { outputVal_ = val; }

    //TODO use different activation function for output layer
    double getOutputVal() const { return outputVal_; }
    void feedForward(const Layer &prevLayer);
    std::vector<Connection>& getOutputWeights();

private:
    unsigned myIndex_;
    double outputVal_;
    std::vector<Connection> outputWeights_; // a value for each weight to the next neuron

    // maps the output value of a neuron to a range between -1..1 or 0...1
    double activationFunction(double x);

    // Create a random weight - decide which function to use for random
    static double randomWeight();


};

```

```
45 #endif //OWNPOKERSIMULATOR_NEURON_H
46
```