CIS 421 Artificial Intelligence, Fall 2019
Assignment 5: Artificial Neural Network

For this assignment, you will implement a simple multi-layer neural network to learn logical XOR with two input nodes, a hidden layer, and a single output node. You may write your implementation in either Java or Python. Regardless of implementation language, you must write all the code for the network yourself – you are not allowed to use any toolkits.

General Note

I am compelling you to make design and implementation decisions in this assignment. If you need ideas or guidance, your textbook is a good source of easily-digested information and bits of code that may prove helpful. You are welcome to adapt the code from the textbook for this assignment.

Deliverables

- A README that explains clearly and completely how to compile and run your code.
- All the code for your neural network.
- A client program that trains and tests your network (as described below).

Neurons and the Network Structure

For a good idea of what your network might look at, see Figure 8.7 on page 172 of the textbook (Chapter 8). This was the network that we used for the simple backpropagation example that I did in lecture. Your network may have more neurons in the hidden layer if you wish.

Neurons in the hidden and output layers will be fully connected to the previous layer and must have a sigmoid activation function. Hidden and output neurons should have a bias weight as well. Input neurons do not have an activation function or a bias weight, they simply pass on the input value.

Training and Testing the Network

Your client program must first train then test your network. For training, you will present a sample (see below) to your initialized network, feed forward, and use backpropagation:

1. **`initialize()`:** Create the network with the appropriate structure (as described above):
    - 2 input neurons,
    - Hidden layer with at least 2 neurons,
    - One output neuron,
    - Randomized weights (between -0.5 and 0.5, unless you choose to change the range of initial weights; -1 to 1 also works well) on connections between neurons. You may have better results if you start the bias weights at 1, but you can treat them like any other connection in the network just as easily and initialize them randomly.
2. **`feedforward()`:** For a training sample (more info below), for each neuron:
    - **`computeActivation()`:** Apply sigmoid function to weighted sum of inputs to the neuron.
    - **`computeNetworkError()`:** Compute error of the network, *i.e.,* the mean-squared error of the actual output and the expected output this training sample.

3. `backpropagation()`: Starting with the output neuron and the hidden layer, compute error between this neuron and each neuron in the previous layer and update weights.
   - `computeError()`: Compute error between the current neuron and the neurons in the previous layer.
   - `computeWeightChange()`: Calculate the current weight change.
   - `updateWeights()`: Update changes to weights for this sample. You will update on a per sample basis (not as a batch).

During training, you should output the activation values, error values, weight changes, and updated weight values so that you can see if your implementation is working and your network is changing appropriately. Be sure your output is well labeled, since I'll look at it when I'm grading your assignment. Remember that the bias weights will be trained just like any other weight in the network.

Parameters: You will need to choose values for the following parameters:

- Learning rate (named RHO in the textbook, also known as ALPHA) between 0 and 1.
- Number of epochs (maximum learning episodes, which here translate into how many times you present a training sample). 5000 is a good starting place, but you may need to increase that.
- Error criterion, which is a threshold for error that will cause training to stop if it is reached before the maximum epochs elapse.

You may want to incorporate a parameter called *momentum*, which we did not talk about in lecture. Since backpropagation is doing gradient descent on the error function, it is subject to getting stuck on local minima. A momentum term can help reduce the volatility of updating the network and can speed up convergence. Momentum essentially helps the network "remember" which direction it was moving, so it serves to smooth out the sometimes wild fluctuations of network behavior and also to help the learning "jump" out of a local minimum where it might get stuck. If you're interested in incorporating momentum, here is some reading to get you started: https://cs.stackexchange.com/questions/31874/how-does-the-momentum-term-for-backpropagation-algorithm-work . The use of momentum is very typical in NN implementations, but entirely optional for this assignment.

Training Samples: You will need to have samples to use in both training and testing. A single data sample is quite simple – it will be the two input values and the expected output. Samples for XOR are:

```
0 0 0
1 0 1
0 1 1
1 1 0
```

where the first 2 values are the values of the 2 inputs and the third value is the label (expected result).

Since there are only 4 samples, training will require you to give the same samples to the network many times.

For testing, you should be able to use the same feed forward process you used in training to evaluate if your network has learned the correct function (more information below). You shouldn't need to write a different feed forward method for testing.

Client program

You must write your own client to both train and test your network. Your client needs to first train the network as described above. Be sure that your program is showing output during training; if your network is really slow, talk to me about what to show in the training phase.

Once the network has converged (or time has elapsed), the program should automatically test your trained network. Testing must do the following:

for each sample:

- Present data sample to the network by feeding forward. Output the values of the sample as it is given to the network.
- Test the actual network output against the expected output, showing the true values of each.
- Identify the test as PASSED or FAILED.

Code Structure, Style and Commenting

Make all efforts to structure your code well. Methods ought to be concise and modular. Divide code into multiple files as appropriate (avoid one giant file). Please follow published standards for code style and commenting.

Submission

Submit the assignment deliverables program through Moodle (see the top of this assignment for a deliverables list). A scoring rubric is available in Moodle.