Programming Assignment #2
Due: Beginning of class, Thursday, August 16th

In this programming assignment you will implement two different neural networks, and measure their ability to learn various functions. This will help you to better understand how neural networks actually work, as well as see what they are and are not capable of learning. As in the first programming assignment, you may write this assignment in either C, C++, or Java.

There are five functions that you should attempt to learn using the neural networks you implement. Each of these functions has as its input a 5-dimensional vector, denoted \( \mathbf{X} \), with components \( x_1, x_2, x_3, x_4, \) and \( x_5 \). Note that not all the functions you will be learning actually make use of all five input components. Still, every function has a 5-dimensional input (for the sake of uniformity). Each of the component values \( x_i \) in each vector is either a 1 or a 0. The output of these functions is also a 1 or a 0, depending on the value of the inputs.

These functions are described below:

**Function 1:**
If \( (x_1 = 1) \), then the output = 1.
Otherwise, the output = 0.

**Function 2:**
If \( ((x_2 = 0 \text{ and } x_3 = 1) \text{ or } (x_2 = 1 \text{ and } x_3 = 0)) \), then the output = 1.
Otherwise, the output = 0.

**Function 3:**
If \( ((x_1 + x_2 + x_3 + x_4 + x_5) \geq 2) \), then the output = 1.
Otherwise, the output = 0.

**Function 4:**
If \( ((x_1 + x_2 + x_3 + x_4 + x_5) \geq 4) \text{ or } ((x_1 + x_2 + x_3 + x_4 + x_5) \leq 1)) \), then the output = 1.
Otherwise, the output = 0.

**Function 5:**
If \( ((x_1 + x_2 + x_3 + x_4 + x_5) = 5) \text{ or } ((x_1 + x_2 + x_3 + x_4 + x_5) = 3) \text{ or } ((x_1 + x_2 + x_3 + x_4 + x_5) \leq 1)) \), then the output = 1.
Otherwise, the output = 0.
Generating Data for Training and Testing
In order to generate data to train and test your neural networks, you should use either the random number generation code in C (available from the cs121 web page) or use the java.util.Random package if you are programming in Java, to generate 5-dimensional vectors that have randomly selected 1’s and 0’s in their components. The output value associated with each vector should be determined by applying the appropriate function you are trying to learn to the randomly generated vector. In this way, you can generate as much data as you like, by simply creating a new vector of 5 randomly selected 1’s and 0’s, and then applying the appropriate function to get the corresponding output value for such a randomly generated vector.

It is also important to note that you do not need to save copies of all the vectors you randomly generate. In fact, since training the neural networks is done one vector at a time, you can just generate a random vector and its corresponding output, use it to update the weights of the network, then generate a new random vector and its corresponding output, use it to update the weights of the network, and so on. For testing, you can similarly generate a random vector and its corresponding output, see what value the neural network predicts for this vector, and then measure whether or not it predicted the correct output value.

Testing a Trained Network
Once you have trained a neural network using some form of weight updating scheme, you can then measure the classification accuracy of the network. The way you determine classification accuracy is as follows:

Number_correct = 0
Loop N times
  1. Generate a random 5-dimensional vector \( \mathbf{x} \)
  2. Output = the output value that a given function would give to \( \mathbf{x} \)
  3. Prediction = the value outputted by the neural network when given vector \( \mathbf{x} \) as input.
  4. If (((Prediction > threshold) and (Output = 1)) or ((Prediction <= threshold) and (Output = 0))) then
     Number_correct = Number_correct + 1
End loop
Classification_accuracy = (Number_correct / N) * 100

This procedure will give you a measure of the percentage of time that the neural network is correct in predicting the output value associated with an input vector that it given. Note that no weight updates occur during the testing phase. We simply use whatever weight values are obtained after training the network has completed to perform testing.

Learning Rate
In all of the neural network simulations that you run, you should use a learning rate of 0.5 for each of the weight updates you perform. In the notation used in the textbook, the learning rate is always shown as the variable c. You should just use \( c = 0.5 \) for this assignment.
Problem 1. (40 points) The Error-Correction Procedure

Write a simulator for a neural network that has only one node. The topology of this network is as shown below:

\[
\begin{array}{c}
\text{x}_1 \\
\text{x}_2 \\
\text{x}_3 \\
\text{x}_4 \\
\text{x}_5 \\
\end{array}
\rightarrow \text{output}
\]

Note that the network also has a threshold value that is not shown in the diagram above. As described in Section 3.2.2 of the textbook, you can simply augment the input vector \( \mathbf{x} \) with an additional (sixth) dimension that always has the value 1, and have six inputs into your neural network. This means that you will have a corresponding 6-dimensional weight vector \( \mathbf{W} \) in this neural network. The weight that corresponds to the sixth input dimension is then just the threshold for the neural network. You should initialize your networks to have random initial weights between 0.0 and 1.0.

Train your network separately on each of the five functions given on the first page of this handout. You should train the network using the Error-Correction Procedure explained in Section 3.2.6 of the textbook.

For each function, train the network with 10,000 presentations of randomly generated vectors, and update the weights of the network using the Error-Correction Procedure after every vector presentation.

For each function, print out the values of the weights and the threshold that the network contains after training is completed.

After training is completed, test the classification accuracy of your network (given the procedure above) for each function on 1,000 randomly generated vectors.

Answer the following questions regarding your simulation runs.

What classification accuracy do you get for each function?

Can you determine which functions are linearly separable and which are not? On what grounds do you base your decision for whether or not a function is linearly separable?
Problem 2. (60 points) The Back-Propagation Method

Write a simulator for a neural network that has two hidden nodes and one output node. The topology of this network is as shown below:

Again, note that each node in the network also has a threshold value that is not shown in the diagram above. And, as before, you can simply augment the input vector $\mathbf{X}$ with an additional (sixth) dimension that always has the value 1, and have six inputs into your neural network. Likewise, the node in the output layer (which is shown to have two inputs from the hidden units) can also have a third input that always has the value 1. This output node will then have a 3-dimensional weight vector, where the first two weights are applied to the outputs of the hidden units, respectively, and the third weight is simply the threshold for the output unit. As before, you should initialize your networks to have random initial weights between 0.0 and 1.0.

Train your network separately on each of the five functions given on the first page of this handout. You should train the network using the Back-propagation method explained in Sections 3.3.3, 3.3.4, and 3.3.5 of the textbook.

For each function, train the network with 10,000 presentations of randomly generated vectors, and update the weights of the network using the back-propagation method after every vector presentation.

For each function, print out the values of the weights and the thresholds (both in the two hidden nodes and in the output node) that the network contains after training is completed.

After training is completed, test the classification accuracy of your network (given the procedure above) for each function on 1,000 randomly generated vectors.
Answer the following questions regarding your simulation runs.

What classification accuracy do you get for each function?

Which functions are learnable by this sort of neural network topology and which are not? Is this neural network capable of representing a larger set of functions than the neural network you implemented in Problem 1? On what grounds do you base this decision?

Why might you sometimes want to use the neural network you implemented Problem 1 instead of the neural network you implemented in Problem 2?

What strategies might you use to pick a neural network topology if you were presented with a new learning problem?

Training on 10,000 training examples can be excessive for some simple functions. What methods would you propose to determine that a network should stop training prior to seeing all 10,000 training examples? (Note that you should not actually implement such methods. All you need to do is describe them.)

Deliverables
Turn in the printouts of the results of your simulation runs described above. Also, turn in the answers to each of the questions given above. Finally, turn in a printout of the source code to your neural network simulations.