CS 540-3: Introduction to Artificial Intelligence
Homework Assignment #4

Assigned: Friday, March 27
Due: Friday, April 17 by 11:59 p.m.

Hand-in Instructions

This homework includes two written questions and a programming problem. Hand in all parts electronically by copying them to the Moodle dropbox called "HW4 Hand-In," including your answers to the written problems. Your answers to each written problem should be turned in as separate pdf files called **P1.pdf** and **P2.pdf**. The first page of each pdf must include a header with: **your name, Wisc username, class section, HW#, date and, if handed in late, how many days late it is**. The programming problem must be written in Java, and all files needed to run your program, including any support (**do** also include unmodified files in the provided code) files but not input/output/debug files, should be submitted. Finally, create a folder called <**wisc username**>_**HW4**, put **all your code** as well as **P1.pdf** and **P2.pdf** into the folder and compress it as <**wisc username**>_**HW4.zip**. This final zip file should then be uploaded to the course Moodle website.

Late Policy

All assignments are due at 11:59 p.m. on the due date. One (1) day late, defined as a 24-hour period from the deadline (weekday or weekend), will result in 10% of the total points for the assignment deducted. So, for example, if a 100-point assignment is due on a Wednesday 11:59 p.m., and it is handed in between Wednesday 11:59 p.m. and Thursday 11:59 p.m., 10 points will be deducted. Two (2) days late, 25% off; three (3) days late, 50% off. No homework can be turned in more than three (3) days late, regardless of whether any free late days are used. Written problems and programming problems have the same deadline. A total of three (3) free late days may be used throughout the semester without penalty. Free late days will be used automatically on a first-needed basis. **Note that no submission will be accepted after three late days even if you have free late days left.**

Assignment grading questions must be raised with the instructor within one week after the assignment is returned.

Collaboration Policy

You are to complete this assignment individually. However, you are encouraged to discuss the general algorithms and ideas with classmates, TAs, and instructor in order to help you answer the questions. You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems. But we require you to:

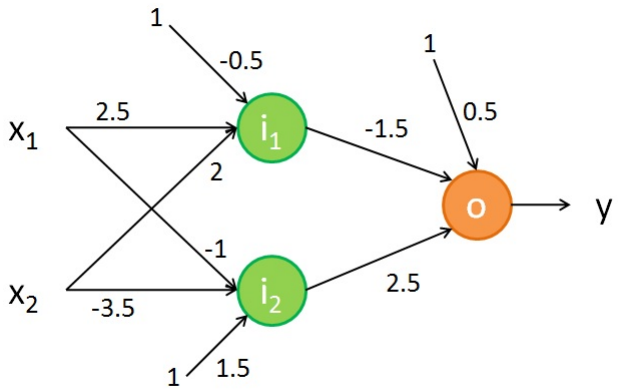
- not explicitly tell each other the answers
- not to copy answers or code fragments from anyone or anywhere
- not to allow your answers to be copied
- not to get any code on the Web

Problem 1. Neural Networks [30]

(a) [15] The figure below shows a 2-layer, feed-forward neural network with two hidden-layer nodes and one output node. x_1 and x_2 are the input variables. For the following questions, assume that the learning rate $\alpha = 0.1$. Each node also has a bias input value of $+1$. Also, assume that there is a sigmoid activation function at the hidden layer nodes and at the output layer node. A sigmoid activation function takes the form:

$$g(z) = \frac{1}{1 + e^{-(z)}}$$

where $z = \sum_{i=1}^{n} w_i x_i$. Here each w_i and x_i are incoming weights and inputs to that particular node.



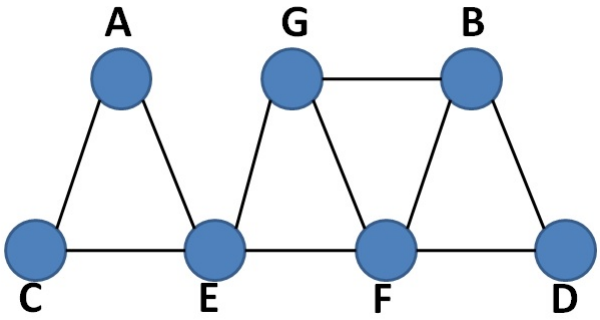
(i) [7] Calculate the output values of nodes i_1, i_2 and o of this network for the input $\{x_1 = 0, x_2 = 1\}$. Show all steps in your calculation.

(ii) [8] Compute one step of the backpropagation algorithm for a given example with input $\{x_1 = 0, x_2 = 1\}$ and teacher output $y = 1$. Compute the updated weights for the output layer (the three incoming weights to the node o) by performing ONE step of gradient descent. Show all steps in your calculation.

(b) [15] Define a neural network that represents the logical function $y = (x_1 \vee x_2) \wedge (x_3 \vee x_4)$. Show the network topology, weights, and biases. Assume that the hidden and output units use sigmoid functions, and an output activation of 0.5 or greater represents a value of True for y.

Problem 2. Constraint Satisfaction Problem [20]

Consider the following graph representing 7 countries on a map that needs to be colored using three different colors, 1, 2 and 3, so that no adjacent countries have the same color. Adjacencies are represented by edges in the graph. We can represent this problem as a CSP where the variables are the countries and the values are the colors.



(a) What are the domains of all the variables after applying Forward Checking inference with variables ordered alphabetically and values ordered increasingly, assuming you start with each variable having all possible values except it is known that A has value 1 and E has value 2?

(b) Apply the Backtracking Search algorithm (Figure 6.5 in the textbook) with Forward Checking inference (Section 6.3.2) and fixed, alphabetical ordering of variables and increasing order of values, starting with each variable having all possible values. Show your result as a search tree where each node in the tree shows each variable with its set of possible values. Arcs in the search tree should be labeled with an assignment of a selected value to a selected variable. If a solution is found, show the final coloring of the map.

(c) What are the domains of all the variables after applying Arc-Consistency (AC-3) inference (Figure 6.3) with variables ordered alphabetically and values ordered increasingly, assuming you start with each variable having all possible values except it is known that A has value 1, B has value 1, and C has value 2?

(d) Apply the Backtracking Search algorithm (Figure 6.5) with Maintaining Arc Consistency (MAC) inference (see Section 6.3.2 and the AC-3 algorithm in Figure 6.3), and fixed, alphabetical ordering of variables and increasing order of values, starting with each variable having all possible values. Show your result as a search tree as described in (b). If a solution is found, show the final coloring of the map.

Problem 3. Back-Propagation Algorithm Programming [50]

In this problem you are to write a program that builds a 2-layer, feed-forward neural network and trains it using the back-propagation algorithm. The problem that the neural network will handle is a binary (two) classification problem. All inputs to the neural network will be **numeric**. The neural network has **one hidden layer** only. The network is also **fully connected** between consecutive layers, meaning each node in the input layer is connected to all nodes in the hidden layer, and each node in the hidden layer is connected to the one node (since we are doing binary classification we only need one) in the output layer. Each node in the hidden layer and the output layer will also have an extra input from a "bias node" that has constant value +1. So, we can consider both the input layer and the hidden layer as containing one additional node called a bias node. All nodes in the hidden and output layers (except for the bias node) use the sigmoid activation function. The initial weights of the network will be fixed. They will be provided through a file. Assuming that input examples (aka instances) have m attributes (hence there are m input nodes, not counting the bias node) and we want h nodes (not counting the bias node) in the hidden layer, then the total number of weights in the network is $(m + 1) * h$ between the input and hidden layers, and $h + 1$ connecting the hidden and output layers. The number of nodes in the hidden layer will also be given as input.

You are required to implement the following two methods from the class `NNImpl` and one method for the class `Node`:

```
public class Node{
       public void calculateOutput()
}

public class NNImpl{
       public double calculateOutputForInstance(Instance inst);
       public void train();
}
```

`void calculateOutput()` calculates the output at a particular node and stores that value in a member variable called `outputValue`. `double calculateOutputForInstance(Instance inst)` calculates the output for the neural network for a given example. `void train()` trains the neural network using a training set, fixed learning rate, and maximum number of epochs (provided as input to the program).

Data set

The data set we will use is called *ecoli* (https://archive.ics.uci.edu/ml/datasets/Ecoli). We have modified the data set to suit this homework. Our modified data set has 7 attributes and one binary class label. All attributes in the feature vector are numeric with varying ranges. The class labels have value 0 and 1. When training the neural network this should be your target value.

In each data file there will be a header that describes the data set, an example of which is shown below. First, there may be several lines starting with // that provide a description and comments for the data set. Then, the line starting with "##" lists the number of attributes. We have written the data set loading part for you according to this header, so do NOT change it.

```
// Description and comments for the data set file
##7
```

Implementation Details

We have created **four** classes to assist your coding, called `Instance`, `Node`, `NeuralNetworkImpl` and `NodeWeightPair`. Their data members and methods are carefully commented. We also give an overview of these classes here.

The `Instance` class has two data members: `ArrayList<Double> attributes` and `int classValue`. It is used to represent one example (aka instance) as the name suggests. `attributes` is a list of all the attributes of that example (all of them are `Double`) and `classValue` is the class (either 0 or 1) for that example.

The most important data member of the `Node` class is `int type`. It can only take the values 0, 1, 2, 3 and 4. Each value represents a particular type of node. The meanings of these values are:

- 0: an input node
- 1: a bias node that is connected to hidden layer nodes
- 2: a hidden layer node
- 3: a bias node that is connected to the output layer node
- 4: an output layer node (since we are doing binary classification there will be only one such node)

`Node` also has a data member `double inputValue` which is only relevant if the `type` is 0. Its value can be updated using the method `void setInput(double inputValue)`. It also has a data member `ArrayList<NodeWeightPair> parents`, which is a list of all the nodes that are connected to this node from the previous layer (along with the weight connecting these two nodes). This data member is relevant only for `type` 2 and 4. The neural network is fully connected, which means that all nodes in the input layer (including the bias node) are connected to each node in the hidden layer and, similarly, all nodes in the hidden layer (including the bias node) are connected to the node in the output layer. The output of the node in the output layer is stored in `double outputValue`. You can access this value by using the method `double getOutput()`, which is already implemented. You only need to complete the method `void calculateOutput()`. This method should calculate the output value at this node if `type` is 2 or 4. The calculated output should be stored in `outputValue`. The formula for calculating this value is:

$$\texttt{outputValue} = g\left(\sum_{i=1}^{n} w_i x_i\right) \tag{1}$$

where $g()$ is the sigmoid function defined as:

$$g(z) = \frac{1}{1+e^{-z}} \tag{2}$$

`NodeWeightPair` has two data members, *Node* and *weight*. These should be self explanatory.

`NNImpl` is the class that maintains the whole neural network. It maintains lists of all the input nodes (`ArrayList<Node> inputNodes`) and the hidden layer nodes (`ArrayList<Node> hiddenNodes`). **The last node in both of these lists is the bias node for that layer.** It also maintains the output node in `Node outputNode`. Its constructor creates the whole network and maintains the links. So, you do not have to worry about that. As mentioned before, you have to implement two methods here. The data members `ArrayList<Instance> trainingSet, double learningRate` and `int maxEpoch` will be required for this. To train the network, use the algorithms given in the lecture slides. Also, do not forget to change the input values of each input layer node (**except the bias node**) when using a new training example to train the network. **Also, do not shuffle the order of the training set.** Otherwise, results may vary.

Classification

You do not have to worry about the classification part for this assignment. Based on the output of `calculateOutputForInstance(Instance inst)` the existing code will classify the example. For this purpose we will use the threshold value of 0.5 meaning an output value greater than or equal to 0.5 will be considered class 1, and otherwise it will be classified as class 0.

Testing

We will test your program on multiple training/tuning/testing sets, and the format of testing commands will be:

```
java HW4 <modeFlag> <noHiddenNode> <learningRate> <maxEpoch> <trainFile> <weightFile>
<testFile>
```

where `trainFile`, `tuneFile`, and `testFile` are the names of training, tuning, and testing data sets, respectively. `noHiddenNode` specifies the number of nodes in the hidden layer (excluding the bias node at the hidden layer). `learningRate` and `maxEpoch` are the learning rate and the maximum number of epochs that the network will be trained, respectively. `modeFlag` is an integer value from 0 to 3, controlling what the program will output:

- 0: print the output of each training example for the initial neural network
- 1: print the modified weights of the neural network after training it using **the first** training example after the **first iteration**.
- 2: print the modified weights of the neural network after training it using all the training examples for `maxEpoch` epochs
- 3: train the neural network with all the training examples for `maxEpoch` epochs and then print the output class for each test example

In order to facilitate debugging, we are providing you with sample input files and their corresponding output files. They are `train1.txt`, `weights1.txt` and `test1.txt` for the input, and `output1_0.txt`, `output1_1.txt`, `output1_2(1).txt`, `output1_2(50).txt` and `output1_3.txt` for the output, as seen in the zip file. For these output files the `learningRate` was set to 0.01, and the hidden layer contained 3 nodes (not counting the bias node). `maxEpoch` was set to 1 for `output1_2(1).txt` and 50 for the other cases. Here is an example command:

```
java HW4 1 3 0.01 50 train1.txt weights1.txt test1.txt
```

You are **NOT** responsible for any file input or console output. We have written the class `HW4` for you, which will load the data and pass it to the method you are implementing.

As part of our testing process, we will unzip the file you submit to Moodle, remove any class files, call `javac HW4.java` to compile your code, and call the main method, `HW4`, with parameters of our choosing. Make sure that your code runs on one of the computers in the Department because we will run our tests on these computers.

Deliverables

1. Hand in your modified version of the code skeleton we provided to you along with your implementation of the back-propagation algorithm. Include any additional java class files needed to run your program, including provided ones that you did not modify.
2. Optionally, submit a file called `P3.pdf` containing any comments about your program that you would like us to know.