CS 307 Introduction to Machine Learning
Fall 2022
Assignment 4: Neural Networks
Part I: *Due electronically by Monday, November 21, 11:59 p.m.*
Part II: *Due electronically by Monday, November 28, 11:59 p.m.*

Instructions:

1. Your solution to this assignment must be submitted via gradescope.
2. For the written problems, submit your solution to the *Assignment 4 - Part 2 (CS 307)* folder as a **single PDF** file.
   - We prefer that you type your solution. If you choose to hand-write your solution, scan your PDF using a **scanner** and upload it. Make sure your final PDF is **legible**. **Regrades due to non-compliance will receive a 30% score penalty**.
   - Verify that both your answers and procedure are **correct, ordered, clean, and self-explanatory** before writing. Please ask yourself the following questions before submitting:
     - Are my answers and procedure legible?
     - Are my answers and procedure in the same order as they were presented in the assignment? Do they follow the specified notation?
     - Are there any corrections or scratched out parts that reflect negatively on my work?
     - Can my work be easily understood by someone else? Did I properly define variables or functions that I am using? Can the different steps of my development of a problem be easily identified, followed, and understood by someone else? Are there any gaps in my development of the problem that need any sort of justification (be it calculations or a written explanation)? Is it clear how I arrived to each and every result in my procedure and final answers? Could someone describe my submission as messy?
3. **IMPORTANT:** As long as you follow these guidelines, your submission should be in good shape; if not, we reserve the right to penalize answers and/or submissions as we see fit.

Part I: Programming (40 points)

Implement the gradient descent algorithm that we discussed in class to train a sigmoid unit for binary classification tasks (i.e. each instance will have a class value of 0 or 1). To simplify the implementation, you may assume that all attributes are binary-valued (i.e. the only possible attribute values are 0 and 1) and that there are no missing values in the training or test data.

Sample training files (`train2.dat`, `train5.dat`) and test files (`test2.dat`, `test5.dat`) are available from the assignment page of the course website. In these files, only lines containing non-space characters are relevant. The first relevant line holds the attribute names. Each following relevant line defines a single example. Each column holds this example's value for the attribute named at the head of the column. The last column (labeled "class") holds the class label for the examples.

When applying the trained sigmoid unit to a test instance, use 0.5 as the classification threshold (i.e., classify the instance as 1 if the unit outputs a value that is at least 0.5; otherwise classify the instance as 0).

IMPORTANT:

- To implement the perceptron learning algorithm, you should use Python (3.6.9), C++ (g++ 7.5.0), or Java (openjdk 11.0.13 2021-10-19). Do **not** use any non-standard libraries (except numpy (1.19.5) and pandas (1.1.5)) in your code.
- Your program should be able to handle **any** binary classification task with **any** number of binary-valued attributes. Consequently, both the number and names of the attributes, as well as the number of training and test instances, should be determined at runtime. In other words, these values should **not** be hard-coded in your program.
- Your program should allow exactly four arguments to be specified in the **command line** invocation of your program: (1) the path to a training file, (2) the path to a test file, (3) a learning rate, and (4) the number of iterations to run the algorithm. Your program should take these four arguments in the same order as they are listed above. There should be no graphical user interface (GUI). Any program that does not conform to the above specification will receive no credit.
- All the network weights must be initialized to 0.
- The fake attribute is explicitly represented in the input data. Hence, you should **not** create an additional input to the sigmoid unit for representing the fake attribute.
- Your learning algorithm should process the training instances in the same order as they appear in the training set.
- You may submit as many source files as needed, but you must make sure you provide a main code entry that follows the following naming convention. Specifically, if you are using:
  - **Python**
    - Make sure that your primary source file is `main.py` and that your code runs successfully after executing `python main.py <path_to_train_file> <path_to_test_file> <learn_rate> <num_iter>`.

- **C++**
  - Make sure that your primary source file is `main.cpp` and that your code runs successfully after executing `g++ main.cpp -o a.out -std=c++17` and `./a.out <path_to_train_file> <path_to_test_file> <learn_rate> <num_iter>`.
- **Java**
  - Make sure that your primary source file is `Main.java` and that your code runs successfully after executing `javac Main.java` and `java Main <path_to_train_file> <path_to_test_file> <learn_rate> <num_iter>`.

What to Do

a. Train the sigmoid unit on the training instances, using the given learning rate and number of training iterations. Print to `stdout` the weights for each attribute and the network output after each **training iteration** (i.e., after processing each training example). For example, if the data set contains three attributes (A1, A2, A3) and four training instances, and the sigmoid is to be trained for five iterations, then the output may look like:

After iteration 1: w(A1)=0.324, w(A2)=-0.021, w(A3)=3.414, output=2.353
After iteration 2: w(A1)=0.567, w(A2)=-6.302, w(A3)=2.516, output=1.522
After iteration 3: w(A1)=0.678, w(A2)=-2.127, w(A3)=2.723, output=-0.211
After iteration 4: w(A1)=0.789, w(A2)=-1.315, w(A3)=-2.343, output=-3.097
After iteration 5: w(A1)=0.891, w(A2)=-0.112, w(A3)=-1.234, output=1.843

Note that (1) the output value is the network output, not the correct class of an instance, and (2) since the number of training iterations in this example is greater than the number of training instances, your program should loop over the training instances. For example, the weights shown after iteration 5 in the above example are those obtained after processing the first training instance (again).

b. Use the learned perceptron to classify the **training** instances. Print to `stdout` the accuracy of the perceptron. The accuracy should be computed as the percentage of examples that were correctly classified. For example, if 86 of 90 examples are classified correctly, then the accuracy of the perceptron would be 95.56%.

```
Accuracy on training set (90 instances):  95.56%
```

c. Use the learned perceptron to classify the **test** instances. Print to `stdout` the accuracy of the perceptron.

```
Accuracy on test set (10 instances):  60.00%
```

To facilitate the grading process, please follow the format of the sample outputs provided in the assignment page of the course website when printing to `stdout`. In particular, when printing the weight values, print exactly the four digits after the decimal place.

Submission

Once you are done, sign in to gradescope. You will be able to see *Assignment 4 - Part 1 (CS 307)* under the Assignments section. Directly submit all your source files to this submission folder. Do not create any folder and do not rename the files or upload the files in a zip file or folder (your homework will not be graded otherwise).

Grading

We will be using an output-based auto-grader for this submission, so make sure you follow the formatting from the example test files: be careful not to insert extra lines, tabs instead of spaces, etc ... When you submit, your code will be **graded using hidden test cases**, so we encourage you to test your code thoroughly. More information about the autograder will be available on Piazza shortly.

Part II: Written Problems (60 points)

1. **Representing Boolean Functions (10 points)**

   Design a neural network with no more than three nodes that computes the XOR function of two inputs using step-thresholded units.

2. **Network Behavior (12 points)**

   Consider a neural net that uses a step-thresholded activation function in each of its units.

   (a) Will its output change for any given input if you add a constant to all weights and thresholds? Briefly explain your answer.

   (b) Will its output change for any given input if you multiply all weights and thresholds by a constant? Briefly explain your answer.

3. **Linear Separability (12 points)**

   Consider the following set of training examples. Each of x_1, x_2, and x_3 is a Boolean attribute with 1 representing *true* and 0 representing *false*. Is the data linearly separable? If so, give a step-thresholded perceptron that perfectly classifies the data. Otherwise, show using linear programming that the data is not linearly separable.

4. **Gradient Descent (12 points)**

   You are given a set of data points $\{(x_i, y_i)|x_i, y_i \in \Re\}$ for i = 1, . . ., n, which is generated by a function of the form $y = a\sin(bx)$ for some a, $b \in \Re$. We want to use gradient descent to determine the values of a and b that minimize the sum of squared error:

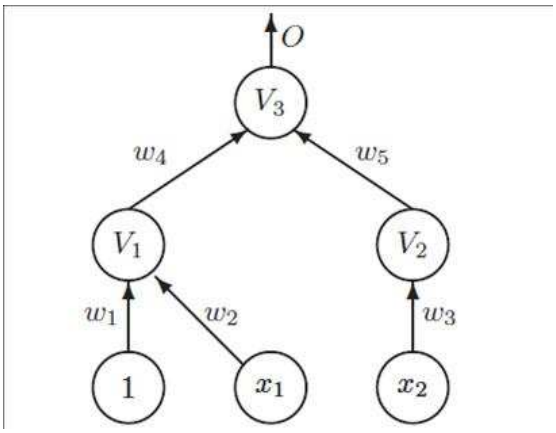
$$\sum_{i=1}^{n}(y_i - a\sin(bx_i))^2$$

Class	x_1	x_2	x_3
0	0	0	0
0	0	1	0
1	1	0	0
1	1	1	0
0	0	0	1
1	0	1	1
0	1	0	1
1	1	1	1

Derive the update rules for a and b. Show the steps in your derivation.

5. **Backpropagation (14 points)**

The following neural network has two layers with nodes, each of which uses a sigmoid activation function. There are no bias connections. Note that (1) one of the inputs to V_1 is always 1 and (2) the value of w_4 is 1 (and cannot be changed).



(a) **(6 pts)** Compute the output values of *all* nodes in forward propagation when the network is given the input $x_1 = 1$, $x_2 = -1$, with the desired output $y = 0$. Assume that the current values of the weights are: $w_1 = -1$, $w_2 = 2$, $w_3 = 1$, $w_4 = 1$, and $w_5 = 2$. Show your work.

(b) **(8 pts)** Compute the new weights w_1, w_2, w_3, and w_5 using backpropagation. Use a learning rate of 0.1. Show your work.