**CS2400 Lab6: Tic-Tac Neural Network Spring 2020**

Overview: In this lab you will be building pieces of your own neural network to train both logic gates as well as a winning Tic-Tac-Toe board recognizer. You are not allowed to use any neural network or machine learning libraries to implement this. Instead you are given a stubbed-out neural network implementation in Python that you will be updating.

This lab is inspired partially by this post: <https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6> You are welcome to use code found in this post to complete this assignment; however, it should be cited appropriately.

This lab is also loosely based on exercises developed by Matthew Mayo, and James Loy albeit using slightly less sophisticated networks <https://www.kdnuggets.com/2017/09/neural-networks-tic-tac-toe-keras.html>

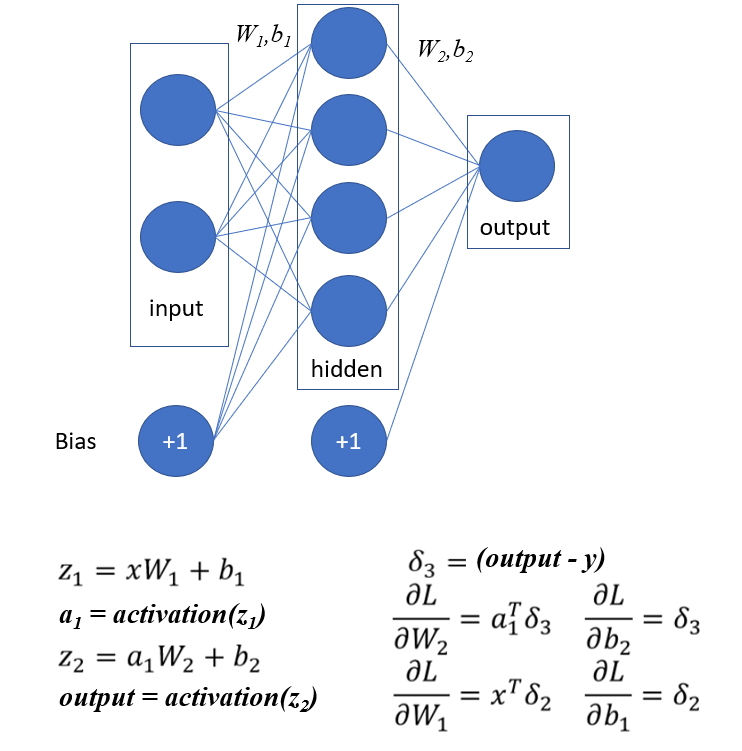
Learning Outcomes:

* Implementation of a neural network

Instructions:

You are given a basic neural network implementation (nn.py) that will serve as your starting point to implement this lab. The NeuralNetwork class will contain basic forward and back propagation functions and a sigmoid activation function.

You are also given a set of test cases (test\_nn.py) that includes logic gate and tic-tac-toe recognition tests for the NeuralNetwork class. The neural network contains an input layer, hidden layer (made from a selectable number of neurons), and a single output neuron. Each layer is fully connected, and the weights are encoded as values of a numpy vector at each layer.



**Part 1:** Finish implementation of nn.py setting all bias values to 0 (don’t worry about “learning” the bias values yet). You are welcome to read through this post to help you along: <https://www.kdnuggets.com/2019/08/numpy-neural-networks-computational-graphs.html> All code that is copied/heavily inspired by external sources should be appropriately commented. Complete the following (Identified as #FIXME in nn.py):

* Function sigmoid(x)
* Function sigmoid\_derivative(x)
* Function backprop() for only the weights
* Function train(epochs)
* Function loss()

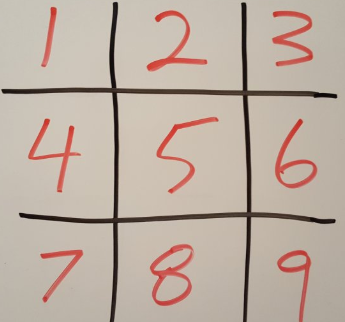
Without the bias term, some of the provided tests will fail. As long as your code is passing some of the tests, you likely have this part implemented correctly.

Fun fact 1: There are 958 possible end-of-game board configurations if you include blanks. We aren’t going to consider blanks until next week😊

Fun fact 2: There are 78 possible end-of-game configurations if you don’t allow blank spaces!

You have been given two initial input files (tic-tac-toe.csv, tic-tac-toeFull.csv) that contain training data for your neural network not considering blank spaces. Each row of this file contains a “winning” configuration of a tic-tac-toe board as well as which letter (x or o) won. The file tic-tac-toe.csv is NOT a comprehensive list of all winning configurations. You wouldn’t want to train using the full set anyway to avoid overfitting, right? The “Full” file has all the valid game configurations (without blanks).

The format of the training data file is as follows: Columns 1-9 correspond to the position on the tic-tac-toe board below. The values in the column represent whether an “X” or an “O” occupies the space. The last column represents whether X wins (Xwin) or Y wins (Ywin). The code to load the file is provided in test\_nn.py.



**Part 2:** After your network is training appropriately, write code to identify cases the network “fails” to predict correctly for the tic-tac-toe.csv examples. You should see if there are trends in this data and write discovered information in a lab report. Try training your network with the same hyperparameters and structure with the Full dataset and see if similar mistakes are made. Document your findings in your lab report.

**Part 3:** Now it is time to add bias values to your network to improve the ability for the network to detect more complex dynamics. After adding the bias values and appropriate derivatives, your neural network should be able to pass all the tests.

Hints:

* self.\_biases2 is actually a single value
* Start by implementing the inference with bias (you can earn partial credit for this)
* Experiment first with the logic gate data when testing your implementation
* Look at the shape of the arrays!
* To calculate the bias derivative with numpy, you will use
  + np.sum(…, axis=0, keepdims=True)

There are more mathematical details of how feedforward and backpropagation work here: <https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a> if you need an additional source of information to complete this lab.

Grades:

|  |  |
| --- | --- |
| Code Style | 20% |
| Part 1 | 40% |
| Part 2 | 20% |
| Part 3 | 20% |

Due date:

4/28/2020 11:59PM

Submission Instructions:

Upload report to blackboard. Upload lab5\_main.py as well as your nn.py to Esubmit.

If you think your network should be passing all the tests on Esubmit, but it is not, try, reuploading your code and the new random values may allow it to pass the next upload time. If a test fails 2 times in a row, there is most likely a bug in your code☹