Homework 5 **NSC 3270 / 6270**
Due Weds Oct 12 11:59pm **Fall 2022**
20 points

Goal. To understand how the error-driven learning rule works, and get some basic experience with pattern classification with a simple toy environment.

Neuroscientific and educational motivation. Error-driven learning rules allow us to teach things to a neural network. By providing a set of teaching labels along with training patterns we tell the network what the "right answer" is, for that training pattern. Understanding this basic form of error-driven learning will give you a good foundation for the backpropagation learning rule, which is widely used today in cutting-edge AI doing object recognition.

Overview. For this assignment, you will use Keras to run one of the simple classification problems we looked at on: `http://playground.tensorflow.org/`

You will generate sets of training patterns, train and test a simple neural network (with an input layer and an output layer, but no hidden layer) using Keras/Tensorflow, and visualize the predictions of the network after training. We will review code in class that you can use as a starting point for your work. The next homework will involve adding a hidden layer to the network.

Please refer to `Homework5.ipynb`, which is the starting point for this assignment, and the class slides. Your coding will be added to `Homework5.ipynb`. As always, save it as a new file with your last name appended to "Homework5". Submit on Brightspace. Any figures should be embedded in the Jupyter Notebook, as usual. Make sure you submit your notebook after it has been run (with figures and calculations shown).

Check the style guide on Piazza for guidance on style! Try to use functions where appropriate, use variables rather than hard-coded values in your code, and use comments and/or markdown cells to document your code.

To complete this assignment, you should review the example Keras and Tensorflow code from class. All of the coding elements (functions and methods to call, parameters of functions and methods to pass) that you need to use (and adapt) are in various places throughout those examples.

Specific Tasks.

Q1 (10 points). **Train a Keras network to do a simple classification.**
`Homework5.ipynb` includes code to generate examples for a simple classification problem, with two classes, with instances from each class drawn from a multivariate normal distribution (as discussed and demonstrated in class).

Write Keras code to learn this classification problem. Your neural network will have an input layer containing two units (corresponding to the x and y dimensions from the visualization plots) and one output unit (no hidden layers). Two input units means each training pattern will have 2 values. One output unit means each target 'pattern' will have one value.

This is a classification problem with 2 classes. We will call them *class 0* and *class 1*. You don't need 2 output units for two classes! Only one is needed. You'll train the network to produce an output activation value of 0 for patterns from class 0 and an output activation value of 1 for patterns from class 1.

Making training patterns. This assignment uses a function called `gensamples()` to generate sample training and testing patterns from class 0 and class 1. The example code provided in `Homework5.ipynb` generates two numpy arrays containing training patterns (`sample0` and `sample1`). You will need to *concatenate* these two arrays to make a single set of training patterns. I'll tell you about concatenation in class, here it basically means append one set to the end of the other set. You can do this explicitly with for loops; you can also look up how to concatenate two numpy arrays using `np.concatenate()` (also check in `NumpyArrays.ipynb` posted earlier in the semester).

Making teacher patterns. You will also need to create a numpy array containing the teacher values associated with each training pattern. This tells the output unit what the target value is for this training pattern. Otherwise, your network will not know whether the training pattern belongs to class 0 or class 1.

Setting Keras options. You will need to pick the appropriate activation function for a classification network with these particular target output activation values. For this assignment, you should use stochastic gradient descent as the optimizer; follow the example code I gave you where SGD is configured so that `learning_rate=0.01`, `decay=1e-6`, and `momentum=0.9`. Use mean squared error (`mean_squared_error`) as the loss function. You should save `'accuracy'` as one of the metrics when you compile your network. For now, use a batch size of 20.

Training the network. Determine the proper number of training epochs so that the accuracy asymptotes (i.e., accuracy changes by no more than .5% from epoch to epoch) but not so many that you're running the network wastefully. Write a note in markdown or code comments to briefly justify the number of training epochs you choose.

Helpful tips. Each time you run gensamples it will create a new random set of training patterns. Try training the network using several different "randomizations" of the training patterns. You should try training at least 10 times to find the proper number of epochs that is reasonable for most randomizations of the training patterns. The proper number of epochs will be large enough so that the network always learns (to asymptote) but not so large that it runs for an unreasonably long time. Set `verbose=True` during training to get a printed report of how the performance of the network evolves over training epochs.

Q2 (3 points). Visualize the performance of the network over the course of training. An object called `history` is returned when you train the network with `history = network.fit(...)`. Recall (from class) that `history.history` is stored as a Python dictionary. Some of the sample Keras code demonstrates how to access the relevant information from the `history` dictionary. Create a plot of training accuracy as a function of epoch (with axes properly labeled). (Note that this plot can be useful in determining the proper number of epochs from Q1 instead of using `verbose=True`.)

Q3 (3 points) Exploration of different training protocols. Explore what happens when you (a) set the batch size equal to 1 (so that weights get updated after every training pattern), and (b) set the batch size equal to the total number of training patterns (so that weights get updated once per epoch after all of the training patterns have been shown). Train a network with each method, using the "proper" number of epochs you found in Q1.

In a markdown cell, describe the difference between these two training protocols. Try to explain why they behave differently than the training in Q1.

Q4 (4 points) Visualize the classification decision performance. `Homework5.ipynb` provides code that generates an array of test patterns and provides a `plottest()` function that displays a shaded contour plot of network predictions on these test patterns (as discussed in class). Present these test patterns to the trained network from Q1 and plot the results using the `plottest()` function.

For this task, use the network trained in Q1 (using a batch size of 20), not the networks trained in Q3. You can ensure that you save each of the individual trained networks by recognizing that you can give the networks different names. Don't call them all `network`; the initialized network object returned when you call `models.Sequential()` can be called any valid variable name (e.g., `network1`, `network2`, whatever you like!).

Tip: Don't overthink this – Q4 can be accomplished with just a few lines of code.

Unexcused late assignments will be penalized 10% for every 24 hours late, starting from the time class ends, for a maximum of two days, after which they will earn a 0.