Homework 4 **NSC 3270 / 6270**
Due Tues Oct 4 at start of class **Fall 2022**
20 points

In class, we discussed the simplest type of learning in neural networks, called Hebbian learning, and its variant that normalizes activations and weights, called Oja's Rule.

In this assignment, you will train a simple neural network to do some basic face recognition using Oja's variant of Hebbian learning.

Please refer to Homework4.ipynb, which is the starting point for this assignment, and the class slides where we went over details of the assignment. The coding you do should be added to Homework4.ipynb. Save it to a new file with your last name appended to "Homework4". Submit on Brightspace. There is no need to submit separate figures with your code, all figures can be embedded in your Jupyter Notebook. Make sure you submit your notebook after it has been run (with figures and calculations shown).

The code in Homework4.ipynb opens and displays four small (32x32) images of well-known people (Carrie Fisher, Jon Stewart, Michelle Obama, and Bryan Cranston). These images are included with the assignment as a ZIP file. Unzip the file and you'll have a folder called "faces" that should be placed in the same directory as the IPYNB. The IPYNB code centers the pixel values of each image around 0, reshapes each image from a matrix to a vector (like the code for Homework 3 did with the MNIST digits), and normalizes each vector to have a length equal to 1.

In the specific tasks described below, you will code up a simple neural network that takes vectors of pixel values as input and classifies each vector as being Carrie Fisher, Jon Stewart, Michelle Obama, or Bryan Cranston. The number of input units is equal to 32x32 = 1024 (the total number of pixels in the image, with the image represented as a vector) and the number of output units is equal to 4 (there are 4 face identities). The output activation function is a simple linear function (as discussed in class).

The network will be trained on noisy versions of each face. The Homework4.ipynb code creates 500 training patterns per face identity (2000 training patterns total) by adding a small amount of noise (normally distributed), as discussed in class (with normalization). The code displays four examples of noisy versions of each face.

`train_pats` is a 2000x1024 numpy array : 2000 training patterns (500 patterns x 4 identities), each of which is a vector produced from one of the 32x32 original images (centered on 0, with noise added, and normalized).

Style guidelines that will affect your grade: We will post a style guide on Piazza that gives guidance on Do's and Don'ts for your code. An example: Do not use hard-coded values in your code. For example, if you have a for loop that iterates over the 4 faces, don't type 4

directly into your code, rather, use the variable n_faces instead. This will improve the readability and robustness of your code.

Specific Tasks

Q1. Create teaching patterns for your Hebbian network. Create a new 2000x4 numpy array called `train_outs` that has the "teacher" (vector of output values you are training the network to reproduce) for each training pattern. For Q1—Q5, have the output of the correct answer equal to 1 and the output of the incorrect answers equal to 0.

Q2. Create a normalized weight matrix with the appropriate dimensions. Initialize the weights in the matrix to random numbers drawn from a normal distribution. Normalize the weights so that the vector of weights contributing to the net input of any given output node has length equal to 1.

Q3. Train the network using Oja's rule. As described in class, tor each of the 2000 patterns, the change in weight Δw_{ij} between input node i_i and output node o_j is given by

$$\Delta w_{ij} = \lambda\, i_i\, o_j$$

where λ is the learning rate (assume $\lambda = .01$). If you stopped here, this would be Hebbian learning. For Oja's rule, you will need to normalize the weights after updating the weights for each training pattern such that the vector of weights contributing to the net input of any given output node has length equal to 1 (you don't normalize after every individual Δw_i update, but after you update all the weights in the network).

Obviously, to train the network, you will need to write code to train the network. In Homework 3, we gave you Keras code to do the training. For this assignment, you need to write that training code from scratch. This should be only a few lines of code. It's not complicated. Each training pattern specifies the input unit activation values. For each training pattern there's a corresponding `train_outs` vector (from Q1) that determines the output unit activation values. With input activation and output activation, you can use the Hebbian learning rule to calculate "delta w" for each of the weights in the network.

You do not need to worry about making your code efficient (vectorizing and the like) – you can if you want, but you do not have to. Slow is fine, so long as it runs correctly. Note that it will take a while to train the network (several minutes, depending on the speed of your computer).

Above, the instructions describe one way to implement Oja's rule (use the simple Hebbian equation and then normalize the weights). Alternatively, you can implement Oja's rule directly, using the equation given in the Week5a class slides. (This is up to you, the two implementations behave identically.)

Q4. Visualize the weights going to each output unit of the network (like you did in Homework 3). For this task you will produce four figures of "weight images". If your network is working properly, the weight images should look like the non-noisy versions of the celebrity pictures.

Q5. Test your trained network. First, present each original image to the network (the four original faces, before noise was added). Second, present a new noisy version (using the same level of noise used during training) of each of these images to the network. For each test image presented to the network, print out the correct answer and the answer produced by the network (using the winner-take-all decision rule from Homework 3).

Try creating a new set of four test images (one for each identity) with a higher level of noise that used during training that causes the network to make classification errors.

That's 12 test patterns total:
4 original non-noisy face patterns
4 noisy face patterns with the same level of noise as the training patterns
And 4 noisy face patterns with noise increased enough that the network makes errors.

EXTRA CREDIT (2 points)

See what happens when you change the coding for the output units from what was assumed above (the output of the correct answer equal to 1 and of the incorrect answers equal to 0) to a different coding of output units, with the output of the correct answer equal to 1 and the incorrect answers equal to -1). Retrain the network (like Q3), visualize the weights (like Q4), and test the network (like Q5).

Why does the visualization of the weights look different with this coding of output units from the one used in the main of the assignment? What is different about what is learned (and hence represented in the weights in the network) between the two ways of coding outputs? Add your written responses to a markdown cell in your assignment (an explanation is required to receive full extra credit).

Reminder: Check the class slides for some additional hints and suggestions and examples of what some of the output might look like.

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.