Homework 3 **NSC 3270 / 6270**
Due Thurs Sep 22 at start of class **Fall 2022**
20 points

Goal. To attain familiarity with the basic computational operations of the Keras toolbox and neural network classifiers more generally.

Neuroscientific and educational motivation. Keras is a high-level toolbox that provides tools for deep learning and machine learning applications. It runs on top of the TensorFlow machine learning platform. This means the functions you call from Keras in turn call functions from TensorFlow. For now, you don't need to understand all of the pieces of Keras and how they work (we will go over those in the coming weeks). Between the class lectures, this assignment PDF, and the code in Homework3.ipynb, you'll have all the information you need to complete this assignment.

In class, we went over the simplest possible neural network consisting of two input units and a single output unit. For this assignment, you will be exploring a more complicated neural network with 784 input units and 10 output units. The concepts are the same. The mathematics are the same. The increased scale means just that these models can only be simulated and evaluated computationally.

The neural network you are given in Homework3.ipynb learns to classify hand-written digits from 0 to 9. The database of digits is called MNIST and originally came from the National Institute of Standards and Technology (NIST) for evaluating systems that could read hand-written digits. In fact, the earliest "deep learning" networks were first used 20 years ago by the US Postal Service to read the (often hand-written) ZIP codes of the mail it processed. Each image in the MNIST database is a 28x28 grayscale image containing an example of a single handwritten digit (see the class slides for more details).

The provided code demonstrates (with working code) how to load the MNIST handwritten digit images, how to display the digit images graphically, and how to convert the 28x28 digit images to usable training and testing patterns. It also demonstrates how to define (i.e. create), train, and test a simple neural network classifier.

The code in Homework3.ipynb learns to classify the digits from training data (a set of 60,000 images) and then evaluates the performance of the network with testing data (a set of 10,000 images). For this assignment, you will evaluate the performance of the network, find examples of correct and incorrect test classifications, visualize the weights of the trained network, and use the weights and bias values of the trained network to calculate on your own the output values (confirming that calculated values match the outputs given by Keras).

Helpful notes. Note that this assignment is worth 20 points (the previous one was worth 10, the first worth 5), reflecting its (potentially) greater difficulty and complexity. Do not wait until the day before it is due to start working on this assignment. The probability of

getting a helpful response to a question is greatly amplified if you ask the question well before the due date for the assignment.

For this assignment, the Homework3.ipynb file contains placeholder cells at the end for you to write your code for the assignment. You will add your answers to each of the questions below to the Homework3.ipynb file and save it as a new file with your last name appended to "Homework3". As always, submit this through Brightspace. For this assignment, you'll use your Jupyter notebook to create all the figures you need. You won't need to submit the figures separately. Make sure you submit your notebook after it has been run, that way the TA will be able to see your figures and code outputs.

Specific Tasks:

Q1. Write a function that returns the network's classification responses.
The original MNIST `test_labels` numpy array contains the digit value associated with the corresponding digit image (`test_images`). The output from the network (from `out = network.predict(test_images_vec)` in the code) contains the activations of the 10 output nodes for every test image presented to the network.

Write a function that takes the (10000,10) numpy array of output (`out`) (of type float) activations and returns a (10000,) numpy array of discrete digit classification by the network (of type int).

Specifically, create a `test_decisions` numpy array of the same size and type as the MNIST `test_labels` array you started with. Whereas `test_labels` shows the correct answer, `test_decisions` shows the decision made by the network. Below (in Q2) you will use both arrays to identify and display test images that the network classifies correctly or incorrectly.

Here are some tips on how to turn a numpy array of continuous output activations into a discrete digit classification. For each 10-d row of the output activations matrix, find the output unit with the maximum value. This is the "winner" that "takes all" (a phrase I'll talk about in class). Remember, each output unit corresponds to a digit, and the output unit with the maximum value tells you which digit the network chose for the corresponding test pattern in `test_images_vec`. This digit is the network's decision / guess / classification.

In your function, feel free to use for loops. Here, we are looking to see that you understand how to use the outputs generated by the network, not whether you can program using the most efficient Python style.

Q2. Graphically depict example digit images for correct and incorrect classifications.
Here, you will examine the correct answers (`test_labels`) and the network's classification responses (`test_decisions`, the array you made for Q1). For each digit 0 to 9, find one test image (`test_image`) that is classified by the network correctly and one test image that is classified by the network incorrectly. To find correct and incorrect

examples you'll need to compare the test labels (the right answers) to the classification responses, and find entries where they match (a correct answer) and mismatch (a wrong answer). You'll have to also filter by the identity of the digit (in `test_labels`) so you can get an example for each digit.

Create a 2x10 plot of digit images (feel free to adapt the provided code that uses subplot to do something like this). There should be a column for each digit 0 to 9. The first row should show correctly classified examples (one example for each digit) and the second row should show the incorrectly classified examples (one example for each digit). Each subplot title should show the answer and the classification response (e.g., displaying 1/6 as the title, if the correct answer was 1 and the classification was 6).

Q3. Create "images" of the connection weights coming from the input units to each one of the output units.

The weight values in a neural network can often be displayed graphically to give some insight into what the network has learned. After all, each weight is a single number, and numbers can be converted into pixel brightness values. You can adapt the code used to display the actual digit images. Because there are 10 output units, there should be 10 weight images. That's one image for each set of weights connecting the input layer (784 inputs) to each output unit. You can use subplot again, to get all the weight images into the same figure. You will want to reshape each (784,1) vector of weights to a (28,28) image. You can display the result using `imshow()`.

Q4. Reproduce the output matrix of the network without using Keras.

The sample code shows how to extract the weight matrix and bias vector from the trained network ("some pieces needed to complete Homework 3" with comment: # get learned network weights and bias). Use the weight matrix (W), bias vector (B), and activation function (simple sigmoid) to reproduce in your own code the outputs (out) generated by the network.

Here's how Keras does it in the code we provided: `out = network.predict(test_images_vec)`. We want you to re-calculate these output vectors without using Keras and make sure your output vectors match up with the Keras output vectors (in out).

The Keras code we provided uses a simple sigmoid activation function, defined as follows:
f(x) = 1 / (1+exp(-x)).

As part of Q4, demonstrate that your output vectors and the Keras-produced output vectors are the same. It is likely that they will not be exactly the same, but they should be within some small epsilon value. This is because minor differences in floating point calculations across computers can cause small differences in results. If your calculated values are within 0.0001 of the values produced by Keras, all is well. You have some

flexibility in how you determine this close correspondence. One good technique involves subtracting one set of values from the other, and checking that the remainders from the subtraction are less than the epsilon value you chose.

Helpful tips

When working with unfamiliar vectors and arrays, you can use the interactive Jupyter notebook to quickly check whether a line of code accessing elements of those arrays is written correctly and returns the expected value. You can make a new code cell to check something out, and then once you figure it out you can delete the cell.

In the sample code provided you'll see a call to `mnist.load_data()`, which loads the MNIST handwritten digit images, as well as the labels saying what digit that image corresponds to.

The digit images are 28x28 grayscale images, and the code reshapes each of these into a 784-element vector, so they can be presented to the network as training images.

The provided code creates a neural network classifier, and trains it to recognize which digit is which, in a training set. Then the network is used to evaluate a set of test images that it wasn't trained on.

`network.evaluate()`: presents each of the test images to the network, and returns measures of network performance.

`network.predict()`: presents each of the test images to the network, and returns the actual output unit activation values for each of the 10,000 test images.

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