Assignment 1: Notebooks, Python Review; NumPy, Matplotlib, Image Representation

Deadline: Thursday September 17, 2020 at 9:00pm
Late Penalty: There is a penalty-free grace period of one hour past the deadline. Any work that is submitted between 1 hour and 24 hours past the deadline will receive a 20% grade deduction. No other late work is accepted.
Original Author TA: Harris Chan

Welcome to the first assignment of ECE 324! This assignment is a warmup to get you used to the programming environment used in the course, review and renew your knowledge of Python, and learn to use several software libraries that we will use in the course. This assignment must be done individually. The specific learning objectives in this assignment are:

1. Set up the computing environment used in this course: the Python language interpreter and either a Jupyter Notebook or Google Colab Notebook.
2. Review and re-familiarize yourself with Python and learn/review the libraries `NumPy` and `Matplotlib`.
3. Get comfortable with callable objects, and use them to write code that looks a little like the neural nets we'll use in this course.
4. Learn to load, process, and visualize image data.

What To Submit

You should hand in the following files, to this assignment on Quercus:

- A PDF file `assign1.pdf` containing your answers to the written questions in this assignment.
- Your code for parts 1, 2 and 3 in the form of ipython notebook files `part1.ipynb`, `part2.ipynb`, and `part3.ipynb`.

1 Setting Up Your Environment

There are two choices of environments to use in this course: **Google Colab** (the next section) or using **Anaconda Python and Jupyter Notebook** (the section after that).

1.1 Using Google Colab

If you have access to Google (i.e. you have a google account, and access to Google from your location), you can use the Google Colaboratory. When logged in to google, go to https://colab.research.google.com. To learn how to write python code into a Google Colab notebook, read and follow the following links:

1. Read What is Colaboratory?
2. Near the bottom of *What is Colaboratory?* click on these links:

- Overview of Colaboratory
- Guide To Markdown
- Guide to Local Files, Drive, Sheets and Cloud Storage

Once you've walked through these sections, you can proceed to 2 below.

1.2 Install Anaconda Distribution of Python 3.8

If you don't have access to Google, as above, or prefer to run everything on your own computer you can use **Anaconda** distribution of Python **3.8**, which comes pre-installed with several scientific computing libraries including `NumPy` and `Matplotlib` and `Jupyter Notebook`.

1. Download the latest Python 3.8 version from https://www.anaconda.com/distribution/ for your specific operating system (OS), one of Windows, macOS, or Linux. Choose the "64-Bit Graphical Installer" to do the installation. (It is also fine to choose the "64-Bit Command line installer" if you are familiar with the command line.)

2. Follow the detailed installation instruction steps that are given in https://docs.anaconda.com/anaconda/install/ for each OS. You do not need to install Microsoft Visual Studio Code when prompted. For Linux, you can skip step 2 (hash check) as it is optional.

1.2.1 Setting up your Virtual Environment

It is good practice to create a 'virtual environment' which ensures that the Python tools and libraries are the right ones that we specify. You will create a virtual environment, called `ece324`, using the Anaconda 'conda' command as described in the following steps:

1. Open up a *command line terminal*: To do this on a Windows PC, search for "Command" and open Command Prompt; On Mac and Linux, you should open the "Terminal" application.

2. To create the virtual environment, run the following command in the terminal:

   ```
   conda create -n ece324 python=3.8 anaconda
   ```

   This process will take several minutes, possibly longer if you have an older computer.

3. To test that the environment works, activate the environment by running:

   `conda activate ece324` (for Mac/Linux)
   `activate ece324` (for Windows)

   After this, you should see a `(ece324)` as the command line prompt.

4. To exit from the environment, you can simply close the window, or run:

   `conda deactivate` (Mac/Linux)
   `deactivate` (Windows)

   Then the `(ece324)` should disappear as the command line prompt.

1.2.2 Lauching, Learning and Using Jupyter Notebook

Once you've got the virtual environment working, launch it again in a command/terminal window. Then simply type:

```
jupyter notebook
```

After a few moments, a new web browser will launch, and it will contain a list of files that were in the folder/directory that you ran the `jupyter` command in. To get started with using Jupyter Notebooks as your development platform, you'll need to read a tutorial, such as this one: https://www.dataquest.io/blog/jupyter-notebook-tutorial/. Once you've gone through this, then you can move on to the next section.

2 Preparatory Readings

Before you attempt the following exercises, read through the following Python, NumPy, and Matplotlib tutorials:

1. For a concise summary of Python, see: https://learnxinyminutes.com/docs/python3/. You only need read up to (and including) section 6.1 (Inheritance). Focus on the simpler functionalities like for-loops and manipulating lists. A good exercise is to type out the code in command line or pycharm and run it to see what happens, if you do not understand a specific part.
2. See the NumPy and Matplotlib section of the Stanford CS231n course Python Tutorial: https://cs231n.github.io/python-numpy-tutorial/. For NumPy, focus on different ways to create and manipulate (i.e. slicing) arrays, as well as vector and matrix mathematics.
3. (Optional) NumPy Tutorial on https://engineering.ucsb.edu/~shell/che210d/numpy.pdf.
4. (Optional) Matplotlib Tutorial: http://scipy-lectures.org/intro/matplotlib/index.html. Another tutorial that focuses more on the image visualization: https://matplotlib.org/users/image_tutorial.html

You may find the following reference (cheat) sheets are useful:

1. `NumPy` cheatsheet: https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf
2. `Matplotlib` cheatsheet: https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_Matplotlib_Cheat_Sheet.pdf

3 Coding & NumPy Exercise (5 points)

The purpose of this section is to get you re-used to the basics of Python, and the use of helpful Python libraries. In the first part of the assignment, you will be manipulating arrays using NumPy input functions, computing with arrays using for-loops, and then doing the same thing using the built-in NumPy functions. You will need the files `matrix.csv` and `vector.npy` which can be found as part of this assignment.

Write a Python program as a Google Colab or Jupyter Notebook called `part1.ipynb` (note that the file type `.ipynb` is used by both Colab and Jupyter) to accomplish the following tasks:

1. Load the `matrix.csv` file into a NumPy array variable called `matrix`, using the `numpy.loadtxt` function. For those using Google Colab, you will have to learn how to upload files to be accessible to your Colab code, as described in the first few sections of Guide to Local Files, Drive, Sheets and Cloud Storage.

2. Load the `vector.npy` file into a NumPy array variable called `vector`, using the `numpy.load` function.

3. Perform matrix multiplication: `output` = `matrix` × `vector` using for loops to iterate through the column and rows. Do not use any built-in NumPy functions. Save `output` variable into a **CSV** file called `output_forloop.csv` using `numpy.savetxt`.

4. Perform matrix multiplication:`output_2` = `matrix` × `vector` by using the built in NumPy function `numpy.dot`. Save `output_2` variable into a **NumPy Array** (.npy) file called `output_dot.npy` using `numpy.save`.

5. As a way to test for consistency, make sure that the outputs match by computing the difference between `output` and `output_2` and saving it into a CSV file called `output_difference.csv`.

Answer the following question: If the two files you compared above are the same, does it prove that your code is correct? Explain your answer.

4 Callable Objects (10 points)

A useful programming concept that is used extensively in this course is a *callable object*. A callable object is any object that can be called like a function. In Python, any object whose class has a `__call__` method will be callable. For example, we can define an `AddConst` class that is initialized with a value `val`. When the object of the `AddConst` class is called with `input`, it will return the sum of `val` and `input`:

```
class AddConst(object):
    def __init__(self, val):
        self.val = val

    def __call__(self, input):
        return self.val + input

foo = AddConst(4)
foo(3) # Output: 7
```

You can think of the syntax `foo(3)` as a short form for `foo.__call__(3)`.

In this second part of the assignment, you will implement several callable classes to emulate a layer in a neural network. Each class will implement a function that is parameterized by the object's initialization parameters. Figure 1 illustrates this diagram.

Create a Colab/Jupyter Notebook `part2.ipynb` to accomplish the following tasks. Your implementation should be able to handle both Python scalars (int/float) or NumPy arrays (of arbitrary dimensions) as inputs:

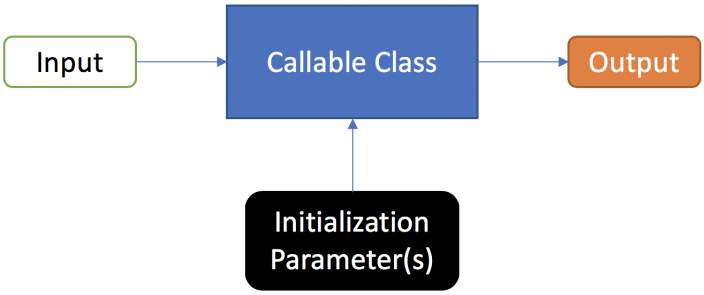

Figure 1: A Callable Class initialized with parameters; it is used to call on an input to produce and output. In the `AddConst` class example, the initialization parameter is `val` and the input (when called) is the scalar value 4 with the scalar output 7

1. Create a callable object class `ElementwiseMultiply` that is initialized with `weight`, a numpy array (with 1-dimension). When called on `input` of the same shape as `weight`, the object will output an elementwise product of `input` and `weight`. For example, the 1st element in the output will be the product of the first element of `input` and the first element of `weight`.

2. Create a callable object class `AddBias` that is initialized with `bias`, a scalar number. When called on `input`, the object will output the sum of `input` and `bias`. Note that `input` can be a numpy array, so the same bias value is added to all elements of `input`.

3. Create a callable object class `LeakyRelu` that is initialized with `alpha`, a scalar value. When called on `input`, which may be a NumPy array, the object will output:

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases} \tag{1}$$

   Hint: You can use NumPy functions to help you implement this class without using any for-loops. Refer to the `numpy.where` function.

4. Create a callable object class `Compose` that is initialized with `layers`, which is a list of callable objects (like those described above) that take in one parameter when called. The object `Compose` will compute the first 'layer' function on the input parameter, and then the second and so on, producing an output that is a composition of object calls in `layers`. The order of the computations is in the order given in `layers`.

5. To test your code from above, copy your code into the notebook file provided `part2_test.ipynb` and run your code and then the test code. What is the output in the terminal?

5 Image Processing (10 points)

A picture or image can be represented as a NumPy array of "pixels", with dimensions $H \times W \times C$, where H is the height, W is the width, and C is the number of colour channels.

Figure 2 illustrates the coordinate system. The origin is at the top left corner, and the first dimension indicates the Y (row) direction, while the second dimension indicates the X (column)

dimension. Typically we will use an image with channels that give the the **R**ed, **G**reen, and **B**lue "level" of each pixel, which is referred to with the short form **RGB**. The value for each channel ranges from 0 (darkest) to 255 (lightest). However, when loading an image through `Matplotlib`, this range will be scaled to be from 0 (darkest) to 1 (brightest) instead, and will be a real number, rather than an integer.

Figure 2: The image coordinate system

You will write Python code to load an image and perform several manipulations to the image and visualize their effects. You'll need to get the file `pink_lake.png` from the same place you downloaded this assignment. Save the output images in the **same directory** as the Python file that you will be implementing.

5.1 Python Program Specification

Write a Python program in a Colab/Jupyter Notebook called `part3.ipynb` to accomplish the following tasks:

1. Load the image `pink_lake.png` into the variable `img`, using the `pyplot.imread` function from matplotlib. You can assume that the image file is located in the same directory as the Python `part3.py` file.

2. Visualize the image by using the `pyplot.imshow` function. To make the plot appear on the screen until you dismiss it, also use the `pyplot.show` function.

3. Modify the image by adding a constant value of 0.25 to each pixel in the `img` and store the result in the variable `img_add`. Note that, since the range for the pixels needs to be between $[0, 1]$, you will also need to implement a "clip" function on the image, where any value in the image that is outside of the desired range to the closest endpoint. You can use the numpy function `numpy.clip` to do this. Save the resulting image in a **PNG** file called `img_add.png` in the same directory as the `part3.py` Python file, using the function `pyplot.imsave`.

4. From the original image, create three images that separate out the three colour channels (red, green and blue), saving each as `img_chan_0.png`,`img_chan_1.png`,`img_chan_2.png`.
   Hint: First create an array initialized with zeros, then copy over the specific channel's 2D content from `img`.

5. Convert the original image to a grayscale image. In a grayscale image, the pixel value of the 3 channels will be the same for a particular X, Y coordinate. The equation for the pixel value [1] is given by:

$$p = 0.299R + 0.587G + 0.114B \tag{2}$$

   Where the R, G, B are the values for each of the corresponding channels. We will do this by creating an array called `img_gray` with the same shape as `img`. Save the output image in a file called `img_gray.png`.

6. Crop the image to be the **top half** of the image. Save the output image in a file called `img_crop.png`.

7. Flip the original image vertically - that is flip it about a horizontal line that is half-way down the image. Save the flipped image to the file `img_flip_vert.png`.

5.2 Qualitative Questions

Answer the following qualitative questions in `assign1.pdf`:

1. How does the image `img_add.png` differ from the original image? What would happen if we had subtracted 0.25 from the original image instead of adding?

2. Describe your programming experience in a few paragraphs. This can include the courses you have taken here at UofT, but if you have more experience, describe that as well.

3. Describe your experience with Assignment 1: how clear were the installation instructions and questions? How can we make it more helpful?

References

[1] Grayscale. Wikipedia, The Free Encyclopedia.
`https://en.wikipedia.org/wiki/Grayscale`