Assignment 1: Python/Numpy + MNIST Neural Network

A precocious toddler has just arrived at UC, but is having trouble understanding lectures and any handwritten notes because the only symbols they recognize are in Burbank Big Condensed Black font. They want to learn more about the outside world, specifically the content of their MATH102 course but don't know how to recognize any digits. Can you help them build a classifier to recognize some digits?

In this assignment you have 3 parts to complete:

1. Complete the the python/numpy review notebook
2. Code a single-layer neural network to classify handwritten digits using only NumPy. You will not be using TensorFlow, or any other deep learning framework, for this assignment. You must follow the structure of the code stencil to receive credit.
3. Answer some conceptual questions in your README.txt file.

Please read this writeup in its entirety before beginning the assignment. Zip and upload your full solution for all 3 parts to learn as your submission. The web interface to google drive (https://drive.google.com/) allows you to download a folder as a zip file so this is probably the easiest way to export a folder.

Note: you are welcome to install libraries and run everything locally instead of on Google Cloud this semester. We recommend using conda to install a jupyter lab environment so that you can open all .ipynb files locally. You can also use this environment with an IDE like PyCharm. See https://saturncloud.io/blog/how-to-install-tensorflow-in-jupyter-notebook/ for an example installation. For this lab you need python modules for numpy and matplotlib and in the future you will need tensorflow.

Part 1

Create a folder on your google drive for assignment1 and place in it a copy of the notebook here:
https://colab.research.google.com/drive/1kSHni0qqYydEETuGignc-3E4sCMK3qmn

Complete the TODO section towards the end and make sure that your python_numpy.ipynb is included in your zip submission.

Part 2

Getting the Stencil (template, skeleton, stub, etc etc :))

You can find the files located on learn under the Assignments page. The files are compressed into a ZIP file, and to unzip the ZIP file, you can just double click on the ZIP file. There should be the files: assignment.py, preprocess.py, README.txt, and MNIST_data folder. Place these in your google drive folder for assignment1 to get started.

Logistics

Work on this assignment off of the stencil code provided, but do not change the stencil except where specified. You shouldn't change any method signatures or add any trainable parameters to init that we don't give you (other instance variables are fine).

Make sure that you are using Python version 3.7+ on assignments. This assignment also requires the NumPy and Matplotlib packages. All of these are pre-installed on Colab. You can upload the included files to Google Drive and then in Colab create a notebook and call your code. Using my drive folder locations, this is the starting code running:

```
[1] from google.colab import drive
    drive.mount('/content/gdrive')

    import sys
    sys.path.append('/content/gdrive/My Drive/UC/teaching/COSC440/assignments/a1')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
```

```
import assignment
from importlib import reload
#reloading allows "hotcode" swapping so the python runtime doesn't need to restart
reload(assignment)
assignment.main("/content/gdrive/My Drive/UC/teaching/COSC440/assignments/a1/MNIST_data/")
```

```
end of assignment 1
```

and after swapping in an acceptable reference solution:

```
import assignment
from importlib import reload
#reloading allows "hotcode" swapping so the python runtime doesn't need to restart
reload(assignment)
assignment.main("/content/gdrive/My Drive/UC/teaching/COSC440/assignments/a1/MNIST_data/")
```

```
train accuracy is  0.885
```

PL = Predicted Label
AL = Actual Label


```
test accuracy is 0.8856
```

```
import unittest
unittest.main(module='assignment_tests',argv=[''], exit=False)
```

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.013s

OK
<unittest.main.TestProgram at 0x7f96fc869ef0>
```

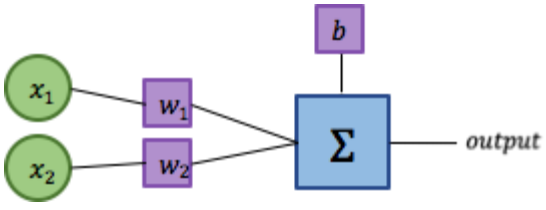
You can edit the python files directly in colab (double click on them) and then re-run the cell that reloads the assignment module file and runs your main function.

Assignment Roadmap

Your task is a multi-classification problem, you will build a one layer neural network to take an image of a handwritten digit and predict its class. This roadmap walks you through the pipeline of training a neural net, including the structure of the model class and the methods you will have to fill in. Our stencil provides a model class with several methods and hyperparameters you need to use for your network. We also have provided some small unit tests in assignment_tests.py -- these are not exhaustive so use them only as a baseline.

Algorithm overview

Review your lecture notes for the Single Perceptron Learning Algorithm. In our lecture example we built and trained an “Is it 2?” perceptron. Here is a brief recap.



Forward pass: $f(x) = output = w_1 \cdot x_1 + w_2 \cdot x_2 + b$

Back propagation: $y = expected - (f(x)>0)$, $\Delta w_1 = y \cdot x_1$, $\Delta w_2 = y \cdot x_2$, $\Delta b = y \cdot 1$

Gradient descent: $w_1=w_1+\lambda\cdot\Delta w_1$, $w_2=w_2+\lambda\cdot\Delta w_2$, $b=b+\lambda\cdot\Delta b$

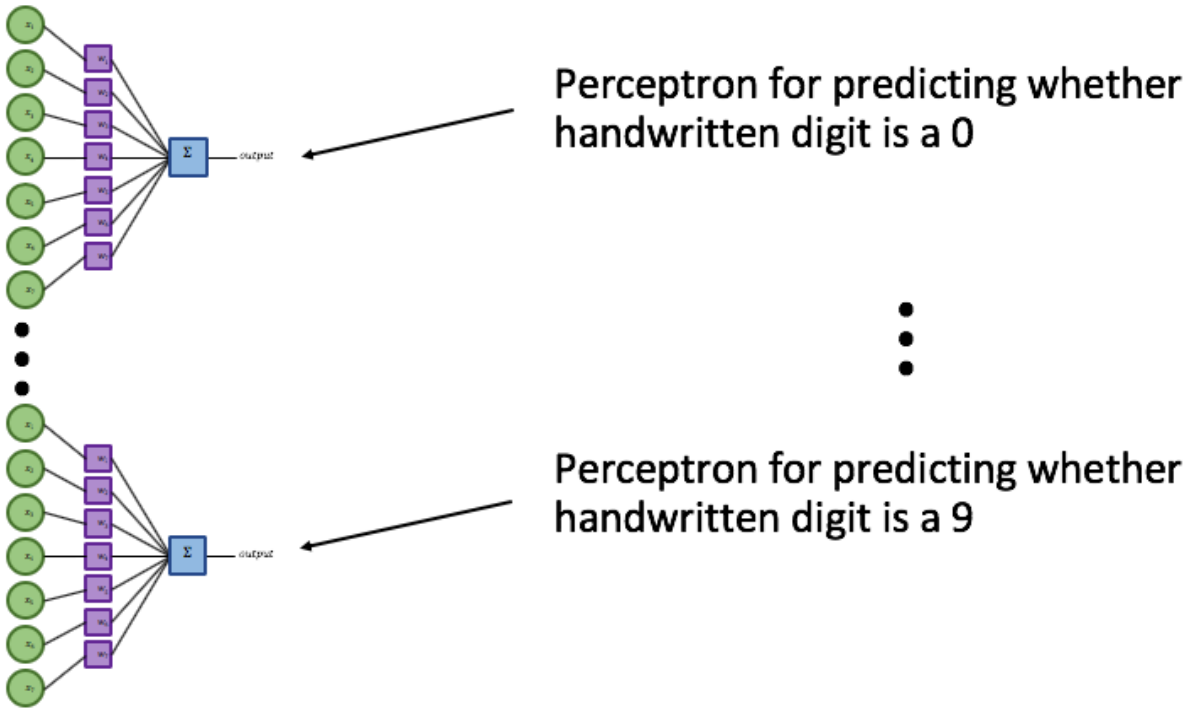
Where λ is the learning rate and Δ indicates a gradient. In our lecture example λ was 1.

We extended this for learning in batches by summing the gradients in back propagation of all k images in a batch and taking the average. This is a minor change:

Back propagation: y^k = expected - $(f(x^k)>0)$, $\Delta w^k{}_1=y^k\cdot x^k{}_1$, $\Delta w^k{}_2=y^k\cdot x^k{}_2$, $\Delta b^k=y^k\cdot 1$,

$\Delta w_1=\frac{1}{n}\cdot\sum_{k=0}^{n}\Delta w_1^k$, $\Delta w_2=\frac{1}{n}\cdot\sum_{k=0}^{n}\Delta w_2^k$, $\Delta b=\frac{1}{n}\cdot\sum_{k=0}^{n}\Delta b^k$

To turn this into a Multiclass Perceptron Learning Algorithm we duplicate the model for each class:



The algorithm does not change for this extension but we must be more careful with how we compute y since we now have a y for each of our 10 perceptrons. To understand the change, the original formula, y = expected - (f(x)>0), resulted in three possible outcomes:

y = +1 if we expected 1 and f(x)>0 is 0
-1 if we expected 0 and f(x)>0 is 1
0 otherwise

This means in two cases we have a gradient update and one case we multiply everything by zero and have no update. With multiple perceptrons, we adjust the definition of y so that y belongs to a class:

y^c = +1 if expected == c and argmax(f(x)) != c
-1 if expected != c and argmax(f(x)) == c
0 otherwise

We then update each perceptron from all inputs in the batch using its own y^c

Implementation

Step 1. Preprocess the data

- Before training a network, you need to clean your data. This includes retrieving the data, altering the data, and formatting them into the inputs for your network. For this assignment, you will be working on the MNIST dataset (see more detailed explanation of MNIST below). MNIST is a dataset containing of images of handwritten digits (0 - 9). You want the inputs for your model to be batch size of images, where each image is a 28 x 28 matrix of pixel values.
- In preprocessing.py, we have provided you with a get_data(inputs_file_path, labels_file_path, num_examples) method that you will have to fill in.
- Please normalize the pixel values.

Step 2. Fill in your model

- Your next step should be to fill out the methods in the model, including all of the TODOs listed in assignment.py. This entails setting your hyperparameters and trainable parameters within the constructor of the model class, filling out the call function (forward pass), doing back propagation + gradient descent (the Perceptron Learning Algorithm), and writing a function to calculate your model's accuracy.
- You should initialize all hyperparameters and trainable parameters in the constructor of the class. Hyperparameters are typically not members of the model class, but doing this is necessary so that when we run your model, we use the exact same hyperparameters you did. Trainable parameters are modified through and through the entirety of training the model (that's what the model is learning!) so it makes sense to initialize this in the model class.
- You should NOT edit the constructor of the Model class to take in any arguments. As mentioned above, everything should be initialized (hard coded) in the constructor.
- You should also fill out the model's gradient_descent(self, gradW, gradB) method. Generally, optimizations of the model's parameters are done outside of the model, but we simplify it in this assignment.
- You will then initialize an instance of your model class in the main function, train your model by doing the forward pass and backward pass many times (call then back_progagate then descent), and the test your model using the testing data and the accuracy function.

Step 3. Train and test

- In the main function, you will want to get your train and test data, initialize your model, and train it for one epoch. An epoch consists of going through the entirety of the training

set once. We have provided for you a train and test method. The train method will take in the model and do the forward and backward pass for one epoch.
- The test method will take in the same model, now with trained parameters, and return the accuracy given the test data and test labels.
- At the very end, we have written a method for you to visualize your results.

Model Parameters

- Take batch size images of 784 values as input (784 values representing the 28x28 image) and output the probabilities for each image belonging to each of the 10 class labels (one class for each digit from 0-9 for each image).
- Have a total of 7850 parameters per image. These are the weights and the biases for each of the 10 perceptrons. All parameters should be initialized to 0.
- Train your network on all 60,000 training examples with a learning rate of 0.5.
- We recommend using a batch size of 100.

Data

As mentioned above, you will be using the MNIST dataset to train and test your network. The dataset can be found here: http://yann.lecun.com/exdb/mnist/ , but we have also provided it to you within the ZIP file.
The training data contains 60,000 examples broken into two files: one file contains the image pixel data and the other contains the class label.
You should train your network using only the training data and then test your network's accuracy on the testing data. Your program should print its accuracy over the test dataset upon completion.

Reading in the Data

The MNIST data files are gzipped. You can use the gzip library to read these files from Python. To open a gzipped file from Python you can use the following code:

```
import gzip
gz_filename = ...
with gzip.open(gz_filename, 'rb') as f:
  buffer = f.read(...)
  # buffer contains ... number of bytes from the file
  # If you use f.read(n) twice,
  # the first call reads the first n bytes, the second reads the second n bytes
```

You might find the function numpy.frombuffer (https://docs.scipy.org/doc/numpy/reference/generated/numpy.frombuffer.html) helpful to convert from a buffer of bytes to a NumPy array.

Note: You should normalize the pixel values so that they range from 0 to 1 (This can easily be done by dividing each pixel value by 255) to avoid any numerical overflow issues. Each pixel is exactly 1 byte.

Data format

The testing and training data are in the following format:

train-images-idx3-ubyte.gz: 16 byte header (which you can ignore) followed by 60,000 training images. A training example consists of 784 single-byte integers (from 0-255) which represent pixel intensities. You will want to read the 16 byte header and then save the rest of the data as the actual training inputs.

train-labels-idx1-ubyte.gz: 8 byte header (which you can ignore) followed by 60,000 training labels. A training label consists of single-byte integers from 0-9 representing the class label. You will want to read the 8 byte header and then save the rest of the data as the actual training labels.

t10k-images-idx3-ubyte.gz: 16 byte header (which you can ignore) followed by 10,000 testing images. You will want to read the 16 byte header and then save the rest of the data as the actual testing inputs.

t10k-labels-idx1-ubyte.gz: 8 byte header (which you can ignore) followed by 10,000 testing labels. You will want to read the 8 byte header and then save the rest of the data as the actual training labels.

Note: You can use the data type np.uint8 for single-byte (or 8-bit) integers. You may have to convert to np.float32 when preprocessing to normalize the data.

Visualizing Results

We've provided the visualizeresults(imagedata, probabilities, imagelabels) method for you to visualize your predictions against the true labels using matplotlib, a useful Python library for plotting graphs. This method is currently written with the labels having a shape of [number of images, 1]. DO NOT EDIT THIS FUNCTION. You should call this function after training and testing on a set of 10 test images. This should result in a visual of 10 images with your predictions and the actual label written above so you can compare your results! You should do this last, after you are sure you have met the benchmark for test accuracy.

Part 3: Conceptual Questions

Please edit and submit the README.txt file with answers to these conceptual questions. You are encouraged to discuss the problems and approaches but you should NOT share any written information or show your written responses to any student.

Q1: Is there anything we need to know to get your code to work? If your code doesn't work or is an incomplete solution please let us know what you did complete (0-4 sentences)
Q2: Why do we normalize our pixel values between 0-1? (1-3 sentences)
Q3: Why do we use a bias vector in our forward pass? (1-3 sentences)
Q4: Why do we separate the functions for the gradient descent update from the calculation of the gradient in back propagation? (2-4 sentences)
Q5: What are some qualities of MNIST that make it a "good" dataset for a classification problem? (2-3 sentences)
Q6: Suppose you are an administrator of the NZ Health Service (CDHB or similar). What positive and/or negative effects would result from deploying an MNIST-trained neural network to recognize numerical codes on forms that are completed by hand by a patient when arriving for a health service appointment? (2-4 sentences)

Grading

Code: You will be primarily graded on functionality, correctness, and proper numerical coding practices. *If you are looping over weights, pixels, or batch records (images) then your program will lose points.* Try to look at high level functions such as np.dot, np.sum, broadcasting, masking (for example, inputs[labels == 0] gives all input values where the expected label is 0), and using operands directly on numpy arrays.

Your model should have an accuracy that is at least greater than 80% on the testing data.

While there is no strict time limit for the running this assignment, it should take less than a minute. If it takes more than >5 minutes, you are probably doing something incorrectly.

We will test by running your Model class against our test suite (we will not be running your main function, we will be running your train and preprocessing functions directly). This means that your program has to be able to run without any arguments from the command line.

Handing in

Please zip and upload your assignment to learn. It should include at least python_numpy.ipynb, assignment.py, preprocess.py, and README.txt.

This assignment is adapted from CS1470/2470 at Brown University with permission and dedicated to Alex and Katie, two awesome retired Deep Learning HTAs.