ECE1513H - Winter 2020 **Assignment 4**
University of Toronto - LEC0101 (Prof. Papernot) **Due Mar 2**

This assignment contains 9 questions worth a total of 11 points. Because assignments will help you learn things that are complementary to what we covered in class, they should be completed on your own. Otherwise, you will not learn from taking this course and you are harming yourself.

This assignment comes with a starter notebook here:

`https://colab.research.google.com/drive/10sKNckPeUM_BjUDjDIJ19l-RhSOOYS_B`

It is **mandatory** to hand in a Python script (i.e., file with extension `.py`) with this assignment, **otherwise it will be considered late**. You can generate a Python script automatically through the `File > Download .py` menu in Colab. Submitting a link to a Colab notebook does **not** constitute a valid submission.

Recall that you can attach a free GPU to your Colab notebook and accelerate your computations. Go to `Runtime > Change runtime type`, and choose GPU under "hardware accelerator".

Problem 1 The starter notebook includes boilerplate code to train a fully-connected neural network (i.e., multilayer perceptron) with JAX. In this problem, we will use MNIST to illustrate concepts from recent lectures.

1. (1 point) Complete the cell which computes the neural network's prediction through the function `predict`.
2. (1 point) Complete the cell which computes the neural network's loss through the function `loss`.
3. (1 point) Complete the cell which defines mini-batch stochastic gradient descent through the function `update`.
4. (0 points) Make sure you are able to train the neural network to an accuracy of about 97%.
5. (1.5 points) Modify the learning rate of your mini-batch SGD and report (a) a value that results in slow convergence, (b) a value that results in oscillations but still converges, and (c) a value that results to instabilities and diverges. For each value, copy-paste the training log (i.e., the list of accuracies achieved after each epoch) in the PDF you hand in on Quercus.
6. (.5 point) Modify the number of neurons and/or number of layers that make up the architecture of your neural network. You can do so by modifying the list `layer_sizes`. Report a list of integers that results in a neural network that underfits the data.
7. (1 point) Find a setting in which your neural network overfits. To this end, modify the set of hyperparameters discussed so far, i.e., learning rate and architecture, as well as the number of epochs if that's necessary. You may also find it useful to set `create_outliers` to `True` and reload the MNIST data by executing `mnist()` again. This will mislabel half of the training data, which makes it easier to find a setting in which the model overfits. Report the set of hyperparameters that result in a neural network that overfits the data.

Problem 2 We will now modify the model architecture to train a convnet. Here again, boilerplate is provided in the starter notebook linked to above in the instructions.

1. (2 points) Update the cell which defines a `stax.serial` model to replace some of the fully-connected layers (they are called `Dense` layers in `stax`) by the `conv+maxpool` layer pairs we studied in class. A convolutional layer is defined using `stax.Conv` and a maxpool layer using `stax.MaxPool`. You will also need to insert a ReLU non-linearity with `stax.Relu`.

2. (3 points) Report an architecture (you can copy-paste the `stax.serial` model definition in the PDF you hand in on Quercus) and set of hyperparameters (learning rate, batch size, number of epochs) that allow you to train a convnet with at least 99% test accuracy. Also report the exact accuracy you achieve (mean and standard deviation over 5 runs).

*
* *