ECE8893B - Spring 2019
Hardware Acceleration for Machine Learning
Lab1A - Running MLP and CNN on CPU

In this lab we will create neural networks to classify handwritten digits from the classic MNIST database. MNIST is a well-known dataset for handwritten digits. The dataset contains 28x28 sized gray scale images with their respective labels. We will use the popular Keras library with Tensorflow backend to build and evaluate our model.

We have two parts in this lab. Part 0 guides us through the process of setting up the environment. In Part 1, we will create and test two models, a multilayer perceptron (MLP) and a small convolution neural network (CNN) running on a CPU. Later, in Lab1B we see run the models created in Lab1A on GPUs and get an appreciation for hardware acceleration.

Part 0: Set the environment for CPU

It is recommended to use the following ECE servers for this exercise.

- ecelinsrvv
- ecelinsrvw

You can access this machine using this example command:

```
ssh <GTid>@ecelinsrvv.ece.gatech.edu
ssh <GTid>@ecelinsrvw.ece.gatech.edu
```

If you wish to use your personal machine, follow Steps 1 and 2. If you use the ECE machines, go to Step 3.

We recommend using bash or any other shell. If you have a linux or mac machine, then the default terminal will do. For windows 10 users, please use bash for windows or windows PowerShell. You can also consider using a virtual machine with ubuntu, however given the amount of computation involved in our exercises, be warned that this will be very slow.

Use the following steps to set up the environment.

1. Ensure that you have python3 installed in the system. You can check if you have one or not using this command:

   ```
   which python3
   ```

   If you don't have python3 you can install it from Python's homepage.

2. Install virtualenv using the following command:

   ```
   pip3 install virtualenv
   ```

3. We will now create a virtual environment so that we do not have to worry about package conflicts with other python packages and versions in your machine
   virtualenv -p `which python3` ~/ece8893b_virtualenv

4. Now we activate the virtual environment and install packages. Please follow these commands:
   source ~/ece8893b_virtualenv/bin/activate
   pip install -r <path_to_lab_folder>/requirements.txt

   This should install tensorflow and keras and we should be good to go.

5. Note: everytime you want to run the lab, you will first need to run
   source ~/ece8893b_virtualenv/bin/activate
   And then run the python code to launch keras.

Part 1: Handwriting classification on CPU

In this part we will construct and train an MLP and a CNN on CPU.
Your job is to fill in the missing code in **mlp_keras.py** and **cnn_keras.py.**

You can see **example_keras.py** for reference.
Use this command to run your keras code: *python <filename>.py*
Eg.

```
python example_keras.py
```

Here is an example of the output that you should see:

```
Using TensorFlow backend.
Training:
Train on 15000 samples, validate on 45000 samples
Epoch 1/10
15000/15000 [==============================] - 2s 116us/step - loss:
2.4698 - acc: 0.8424 - val_loss: 2.3631 - val_acc: 0.8497
Epoch 2/10
15000/15000 [==============================] - 1s 94us/step - loss:
2.2965 - acc: 0.8537 - val_loss: 2.2571 - val_acc: 0.8559
...
Epoch 10/10
15000/15000 [==============================] - 1s 98us/step - loss:
1.3217 - acc: 0.9159 - val_loss: 1.2981 - val_acc: 0.9174
Inference:
10000/10000 [==============================] - 0s 6us/step
Loss and Accuracy on test set: [1.2974647649765014,
0.9175400507926941]
```

The training ran for 10 epochs, with training time for each epoch listed.
The Inference Time was 6us, and the Accuracy was 0.9175 (i.e. 91.75%).

Keras's documentation (https://keras.io/getting-started/sequential-model-guide/) is a great place to get up and running to learn the syntax. **We recommend you take a look before starting.**

a) **Multilayer Perceptron (MLP):** In the lab assignment folder, you will see a file called ***mlp_keras.py***. The file already contains code to fetch the MNIST dataset, split the dataset into training and validation sets and train a given model.
Your task is to build the model in the build_model method.
For this exercise create a simple neural network with one or more hidden layers, all with sigmoid activation. The output classification layer should contain ten neurons (=number of classes) with softmax activation.

Play around with number of neurons in the hidden layers, number of hidden layers etc. and see how it affects the accuracy and training time.
Vary the number of hidden layers from 1 to 4, and in each hidden layer use two neuron numbers – 32 and 1024.
This will give you 30 configurations: <num_layers, num neurons per layer>.
Plot two bar graphs:
1) **Accuracy on the test set for the different configuration.**
2) **Time required to train for 10 epochs for the different configurations.**

Add these graphs in a file called report.pdf

b) **Convolution Neural Networks (CNN):** As you might already be aware of CNNs are great at classifying images. Plus, MNIST dataset is essentially a collection of labelled images. Thus, it makes perfect sense to try out a CNN on this problem.

For this exercise create a simple convolutional neural network with convolution layers with 2x2 max pooling, one fully connected layer and a final classifier layer. Use the ***cnn_keras.py*** provided in the assignment folder for your work.
Use a filter size of 5x5 for the first convolution layer and 3x3 for the rest. You can use any number of filters in these layers. For the fully connected layer you can use 256 units, but feel free to play around and study the effect. Use 'relu' activation for the convolution and fully-connected layers, and 'softmax' for the final classification layer.

Note that MNIST is a fairly easy problem nowadays and accuracies over 95% are normal. If you are getting accuracies less than that, then it's an indication that something needs to be fixed.

As with the previous exercise, feel free to play around with the parameters. See how it affects the training time and accuracies.
For the submission, vary the number of convolution layers from 1 to 4. Add a 2x2 pooling layer after the first and second convolutional layers. Use a filter size of 5x5 for the first convolution layer and 3x3 for the rest. For each convolution layer, try 32 and 128 filters.
This will give you 30 configurations: <num_layers, num filters>
Plot two bar graphs:
3) **Accuracy on the test set for the different configuration.**

Time required to train for 10 epochs for the different configurations
Add these graphs in a file called report.pdf

What to submit:

You need to submit 3 files in total

- report.pdf
- mlp_keras.py
- cnn_keras.py