Name: John Moran
Email: jfmoran2@illinois.edu
Class: CS 498, Applied Machine Learning, Spring 2018
Assignment: Homework 9
Due Date: Thursday, May 10, 2018

Sources used in this programming assignment:

- Algorithms and ideas taken from Professor Forysth's book:
  - http://luthuli.cs.uiuc.edu/~daf/courses/AML-18/learning-book-19-April-18.pdf
- Ideas and code samples/examples taken from:
  - https://www.tensorflow.org/versions/r1.1/get_started/mnist/pros#build_a_multilayer_convolutional_network
  - https://www.tensorflow.org/tutorials/deep_cnn
  - https://github.com/ddigiorg/AI-TensorFlow/blob/master/CNN-MNIST/CNN-MNIST.py
  - https://cambridgespark.com/content/tutorials/neural-networks-tuning-techniques/index.html
  - https://www.tensorflow.org/versions/r1.1/get_started/mnist/beginners
  - https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/mnist/mnist_with_summaries.py

Files contained in "Moran_HW9.zip"

1. HW9_mnist_unimproved.py
2. HW9_mnist_improved.py
3. cifar10_train_eval.py
4. cifar10_input.py
5. cifar10.py
6. cifar10_train_eval_BN.py
7. cifar10_input_BN.py
8. cifar10_BN.py
9. part1_a.png
10. part1_b.png
11. part2_a.png
12. part2_b.png

Problem 1.a – MNIST Unimproved Tensorflow/Tensorboard

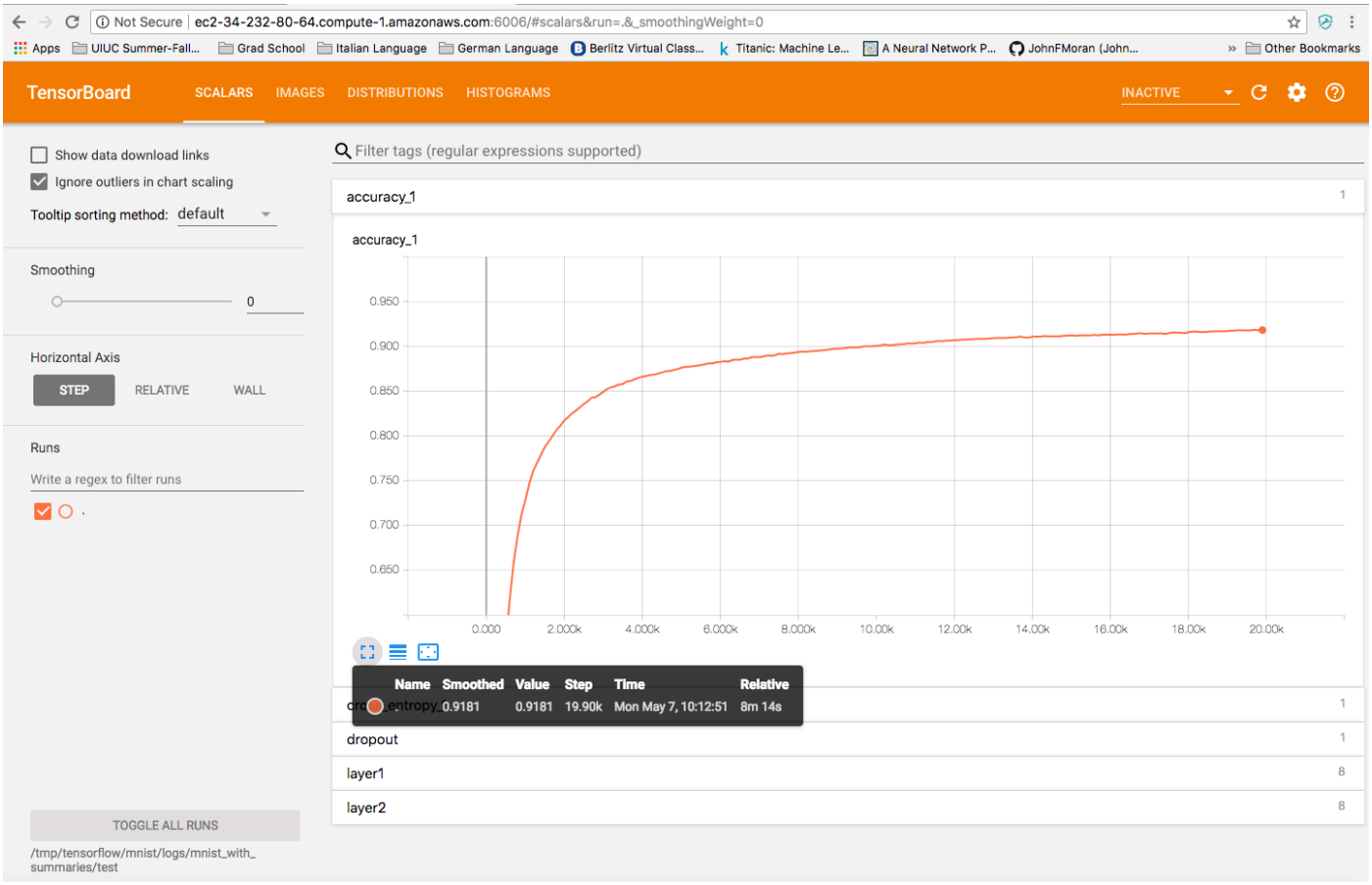
This first version of the MNIST tensorflow code was taken from:

https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/mnist/mnist_with_summaries.py

It was modified slightly to change the optimizer to use Gradient Descent as the first pass attempt since that is what the original beginner model used.

The resulting accuracy was **91.8%**

The resulting tensorboard graph is shown below (and included as a separate file named **part1_a.png**)



Problem 1.b – MNIST Improved Tensorflow/Tensorboard

Code ideas and samples for improving the MNIST program were taken primarily from:

https://github.com/ddigiorg/AI-TensorFlow/blob/master/CNN-MNIST/CNN-MNIST.py
but the code was modified and integrated into the original unimproved tensorflow program in Problem 1.a.

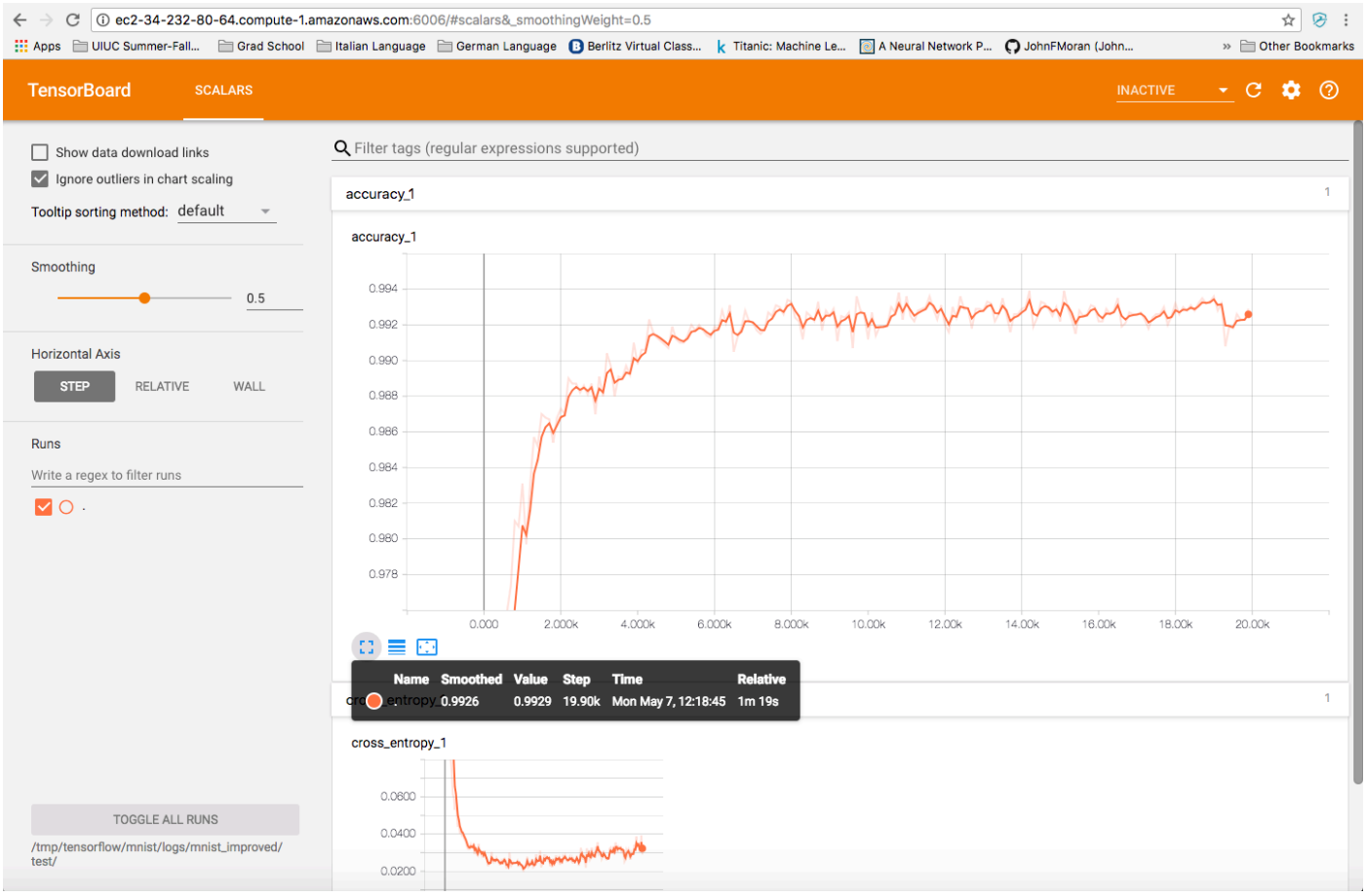
The **modified architecture** was as follows—

1) Delete the simple NN in ***HW9_mnist_unimproved.py***
2) Change the optimizer from *GradientDescentOptimizer* to *AdamOptimizer*
3) Change the model from the deleted simple NN to the following:
    a. Convolution layer, compute 32 features for each 5x5 patch
    b. Apply ReLU function
    c. Max pooling 2x2 to reduce image size to 14x14
    d. Convolution layer, compute 64 features for each 5x5 patch
    e. Apply ReLU function
    f. Max pooling 2x2 again
    g. Apply Dropout (0.5)

The program was run with various option values and the best results were with a Learning Rate of 0.0003 and a dropout of 0.5, run for 20,000 steps.

The resulted in an accuracy of **99.3%**, a significant improvement.

The resulting tensorboard graph is shown below (and included as a separate file named **part1_b.png**)



Problem 2.a – CIFAR10 Unimproved Tensorflow/Tensorboard

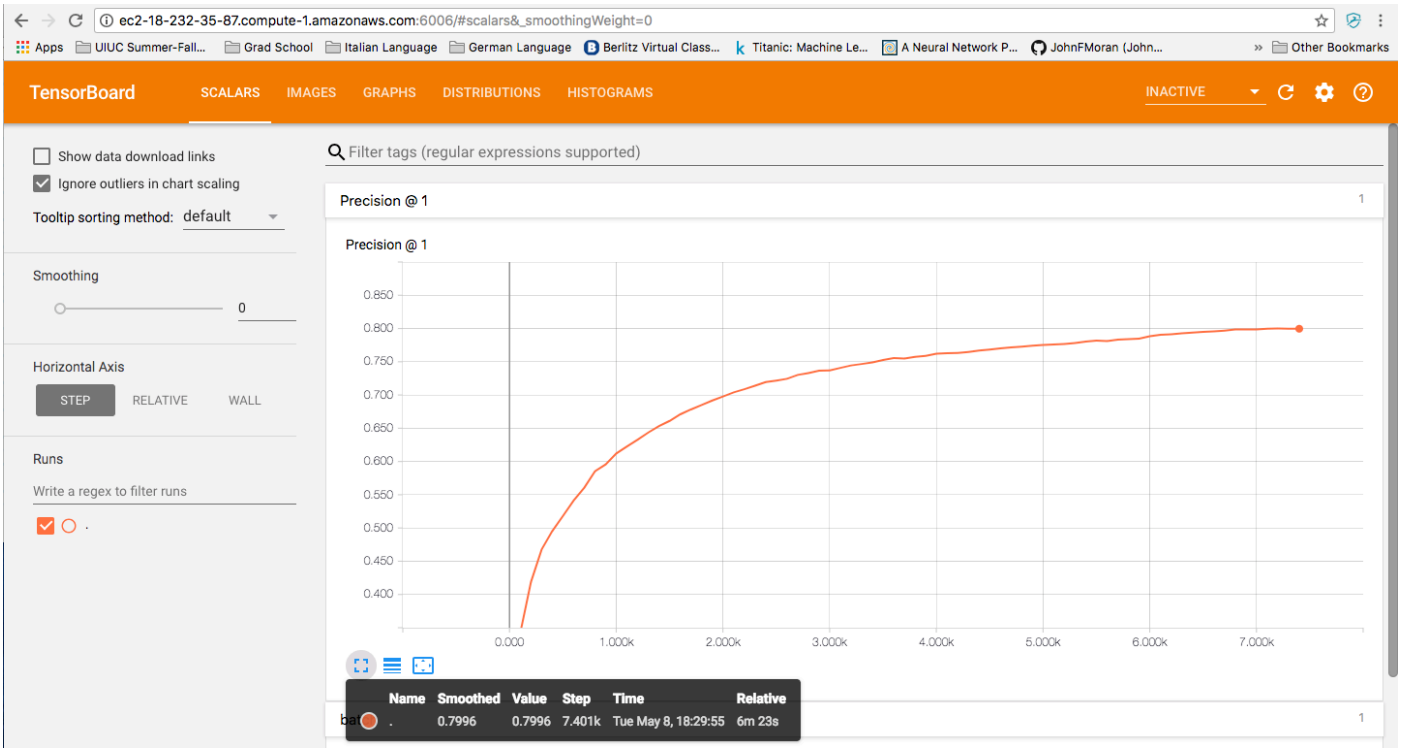
For the first part of the problem, to get tensorboard to show the model accuracy every 100 steps, instead of having two asynchronous programs (train and eval), I merged them into a single file, "**cifar10_train_eval.py**" and called "evaluate()" every 100 steps. I also modified *MonitoredTrainingSession* to add:

save_checkpoint_steps = 100

this required upgrading to *Tensorflow 1.8*.

The accuracy for running 7,500 steps using the default model (though merged) was **80.0%**

The resulting tensorboard graph is shown below (and included as a separate file named **part2_a.png**)



Problem 2.b – CIFAR10 Improved Tensorflow/Tensorboard

To improve on 80% in the same 7,500 steps proved difficult. I attempted multiple approaches including:

1) Modifying hyperparameters
2) Deleting convolution layers
3) Adding convolution layers
4) Changing pooling parameters
5) Adding batch normalization

In the end, there were two changes that proved fruitful. The resulting **modified architecture** was as follows—

1) Changing the image size from 24 to 32
2) Adding batch normalization to normalize the inputs to the ReLU calls in the convolution layers.

These two architectural changes allowed the system to reach **83.5% accuracy** in the same 7,500 steps, which is an absolute **3.5% performance improvement**.

The tensorflow python code for this section are the same filenames as the previous section, but with **_BN** added for "batch normalization".

The resulting tensorboard graph is shown below (and included as a separate file named **part2_b.png**)