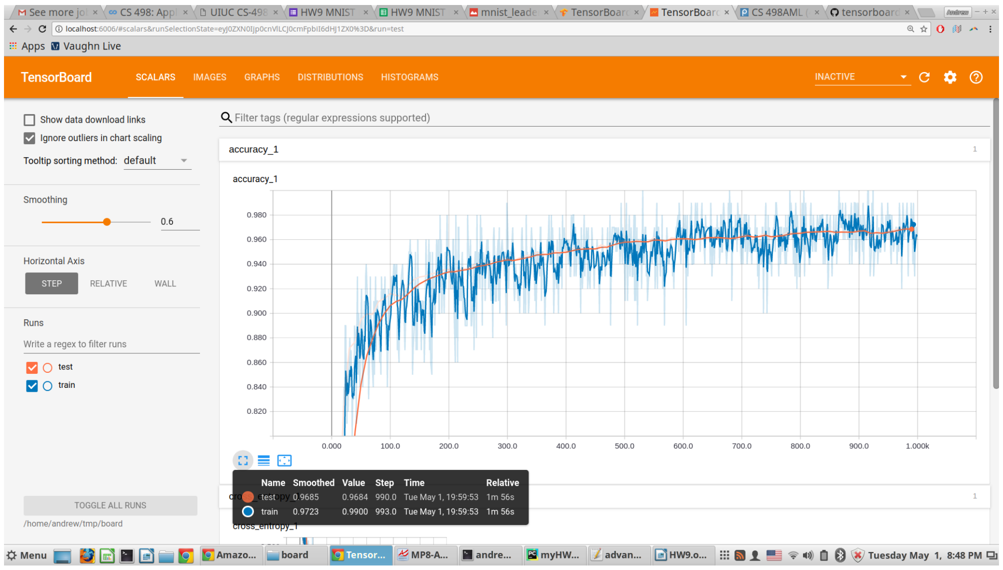
1. MNIST

1.1 Preliminary results for MNIST

These are results from running the two unmodified tutorials

MNIST tutorial
Tutorial Title: *TF Layers: Building a CNN MNIST Classifier*
File path in my submission: other/cnn_mnist_unmodified.py
Test accuracy: 0.9699
Global_step: 20000
Loss: 0.097417705
It took me several hours to run this script locally on my machine

Tensorboard tutorial
Tutorial Title: *TensorBoard: Visualizing Learning*
File path in my submission: other/mnist_with_summaries_unmodified.py
Test accuracy: 0.9683



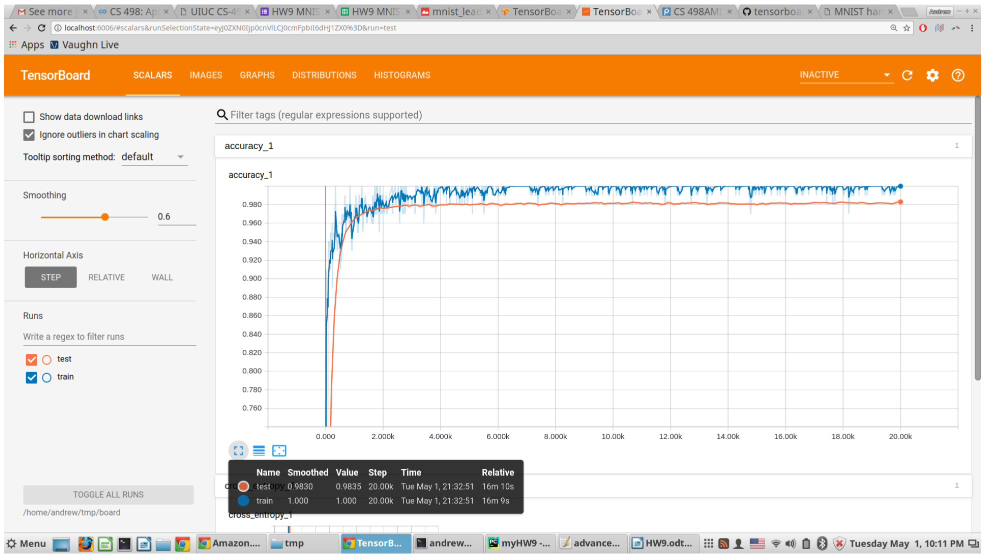
localhost:6006/#scalars&runSelectionState=eyJ0ZXN0Ijp0cnVlLCJ0cmFpbiI6dHJ1ZX0%3D&run=test

☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing: 0.6
Horizontal Axis: [✓ STEP] RELATIVE WALL
Runs: ☑ test ☑ train
/home/andrew/tmp/board

	Name	Smoothed	Value	Step	Time	Relative
●	test	0.9685	0.9684	990.0	Tue May 1, 19:59:53	1m 56s
○	train	0.9723	0.9900	993.0	Tue May 1, 19:59:53	1m 56s

As this was just a tutorial, I ran it for 1000 batches only just to see how it performs.

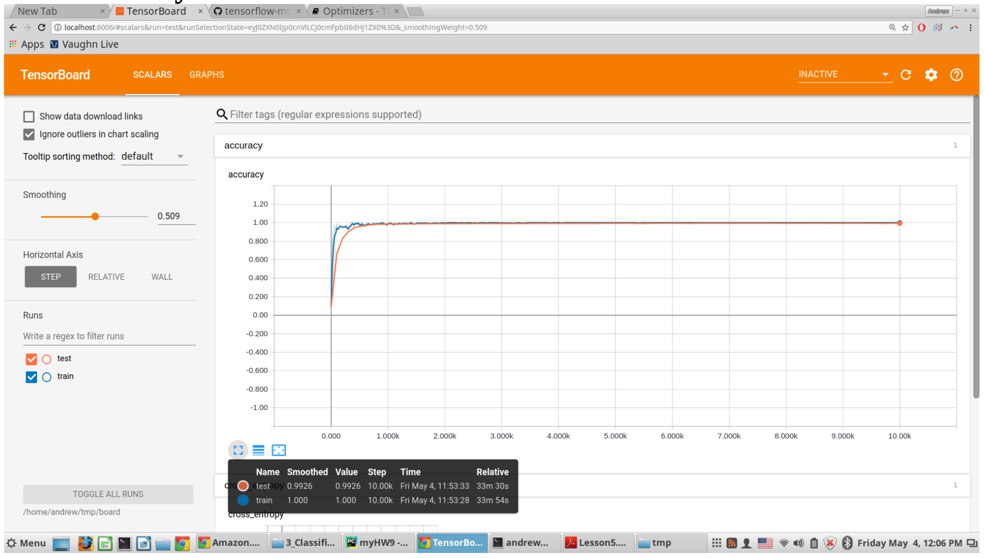
Tensorboard tutorial with modifications
Modifications: log the accuracy on tensorboard every 100 batches, for 20,000 batches
File path in my submission: other/mnist_with_summaries_modified.py
Test accuracy: 0.983



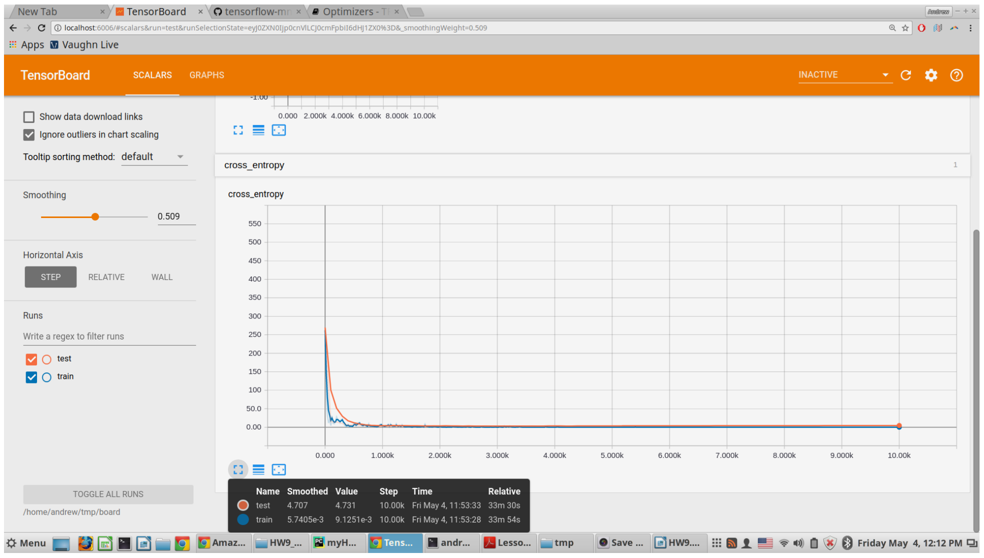
Name	Smoothed	Value	Step	Time	Relative
test	0.9830	0.9835	20.00k	Tue May 1, 21:32:51	16m 10s
train	1.000	1.000	20.00k	Tue May 1, 21:32:51	16m 9s

1.2 Advanced results for MNIST

My main solution is based on the Tensorboard tutorial and reference [2]
Modifications: introduce a convolutional neural network with 3 layers, add Tensorboard functionality (scalars and summaries) in order to visualize results (every 100 batches for 10,000 batches, number of batches was decreased from 20,000 for speed as no significant changes occur after 10,000 in any of the performed runs)
File path in my submission: part1.py
Test accuracy: 0.9926



	Name	Smoothed	Value	Step	Time	Relative
🟠	test	0.9926	0.9926	10.00k	Fri May 4, 11:53:33	33m 30s
🔵	train	1.000	1.000	10.00k	Fri May 4, 11:53:28	33m 54s



	Name	Smoothed	Value	Step	Time	Relative
🟠	test	4.707	4.731	10.00k	Fri May 4, 11:53:33	33m 30s
🔵	train	5.7405e-3	9.1251e-3	10.00k	Fri May 4, 11:53:28	33m 54s

Considering that all these numbers are very close to 100%, the accuracy of 0.9926 is a significant improvement compared to the previous results. It took me several days of test and trial to achieve this locally.
I also tried other approaches similar to reference 2 (https://github.com/hwalsuklee/tensorflow-mnist-cnn/blob/master/mnist_cnn_train.py), but they were very resource-consuming.

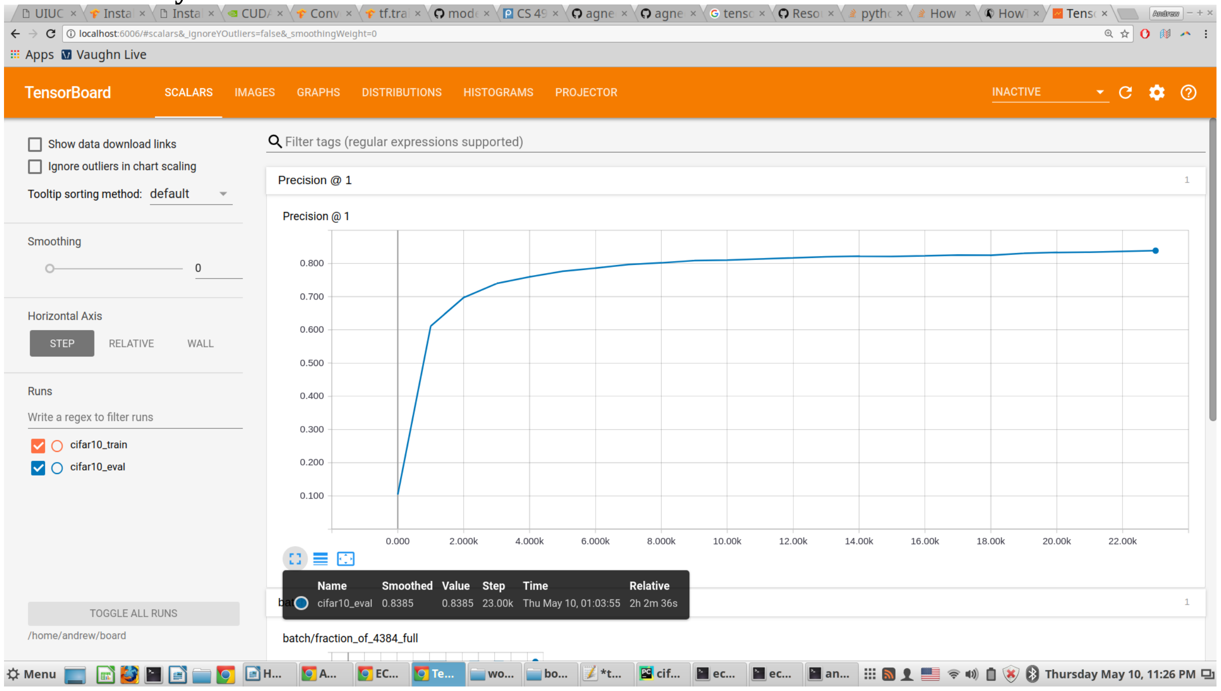
2. CIFAR-10

2.1 Preliminary results for CIFAR-10

As required by the assignment, I made sure I can run the CIFAR-10 tutorial. I did it on a GPU-enabled machine at AWS this time, and it was considerably faster. To expedite the implementation, I decreased the *maximum number of steps* from 1,000,000 to 60,000 in cifar10_multi_gpu_train.py and decreased *min_fraction_of_examples_in_queue* from 0.4 to 0.2 in cifar10_input.py, as it was advised in the CIFAR10 tutorial.
The cifar10_multi_gpu_train.py file has code to capture training statistics every 100 batches. In addition, I changed the parameter eval_interval_secs in cifar10_eval.py from 60*5 to 27 seconds because the average *seconds_per_batch* parameter (as displayed in every line output every 10 seconds during training) is 0.27, and this change allowed me to log the eval accuracy for tensorboard for approximately every 100 batches. As I have mentioned, the maximum number of examples was set to 60,000 (instead of the minimum required 2000), so this gave me plenty of data to log on tensorboard. The training script saves a model checkpoint only every 1000 steps which doesn't happen too often, so there is really no need to have the eval script perform evaluation too often. To have more diverse data I changed the point at which model ckeckpoints are saved from 1000 to 500 in the training script (cifar10_multi_gpu_train.py). With this setup, the training and evaluation scripts ran simultaneously (despite the warning in the tutorial) for about **four hours** on AWS's g2.2xlarge instance. Combined, the train and eval event.output files used to construct the below accuracy chart in Tensorboard are about 20 MB in size, therefore I am not attaching them, but they can be provided if needed.

File path in my submission: folder cifar_baseline

Test accuracy: 0.838



localhost:6006/#scalars&_ignoreYOutliers=false&_smoothingWeight=0

☐ Show data download links
☐ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0
Horizontal Axis: STEP
Runs: ☑ cifar10_train ☑ cifar10_eval
/home/andrew/board

```
Name         Smoothed  Value   Step    Time                      Relative
cifar10_eval 0.8385    0.8385  23.00k  Thu May 10, 01:03:55      2h 2m 36s
```

Thursday May 10, 11:26 PM

1.2 Advanced results for MNIST

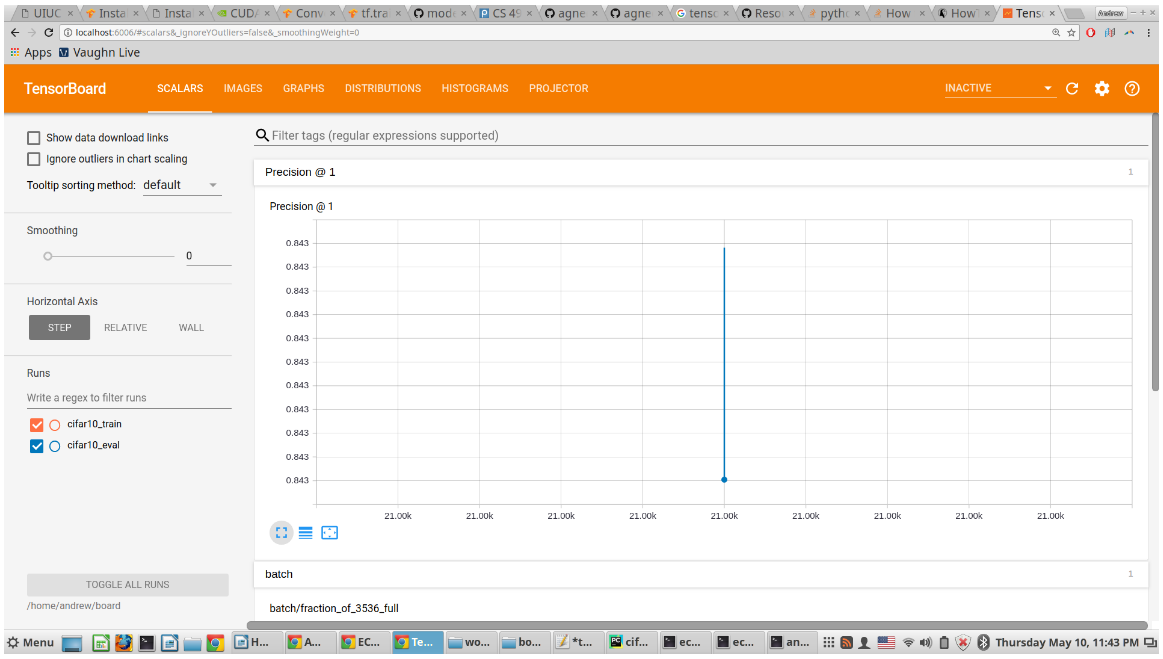
When making test runs with the number of batches=2000, I noticed that initial accuracies were pretty high if I changed the batch_size or the GradientDecentOptimizer to AdamOptimizer. I decided to check the performance on 10,000 batches (this number was selected in the interest of time as the deadline was already approaching) of the following cases:
Case 1: AdamOptimizer with the original learning rate = 0.1
batch_size = 512
One more (third) convolution layer identical to conv2: max_pool and norm were swapped in conv2 to match conv21
Accuracy on 10,000 batches: very poor results

Case 2: AdamOptimizer with the initial learning rate = 0.001 and the learning rate decay factor = 0.001 (trick I learned from reference 1, because the original learning rate gave poor performance)
batch_size = 128
Accuracy on 10,000 batches: 0.815

Case 3: AdamOptimizer with the initial learning rate = 0.001 and the learning rate decay factor = 0.001,
batch_size = 512
Accuracy on 10,000 batches: 0.828

Case 4: GradientDecentOptimizer with the original initial learning rate,
batch_size = 512
Accuracy on 10,000 batches: 0.843

As you can see, the best accuracy is in Case 4, but the problem was that the eval script gave me error and did not run together with the train script because of the lack of memory. Therefore, I was able to record only the last final accuracy with the eval script (see below).
File path in my submission: folder cifar_improved


localhost:6006/#scalars&_ignoreYOutliers=false&_smoothingWeight=0
TensorBoard
SCALARS
IMAGES
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
PROJECTOR
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
cifar10_train
cifar10_eval
TOGGLE ALL RUNS
/home/andrew/board
Filter tags (regular expressions supported)
Precision @ 1
0.843
21.00k
batch
batch/fraction_of_3536_full
Thursday May 10, 11:43 PM

This is a marginal improvement, but I tried spending days to achieve it. Each run on 10,000 batches took several hours even on the AWS instance. I even rented a more powerful one in the end, but did not help to speed things up significantly.
The next natural step would be to change the CNN architecture. I tried this in Case 1, but was confused by the results which were due to the incorrect learning rate for the AdamOptimizer, as I found out later.
To be continued.

REFERENCES
1. Learning rate schedules. https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1
2. Tensorflow CNN MNIST tutorial. https://github.com/martin-gorner/tensorflow-mnist-tutorial