Vanderbilt / IBM
Deep Learning Workshop

Hands-on TensorFlow Programming
Vanderbilt
Feb. 6, 2019

James Nash
jjnash@us.ibm.com

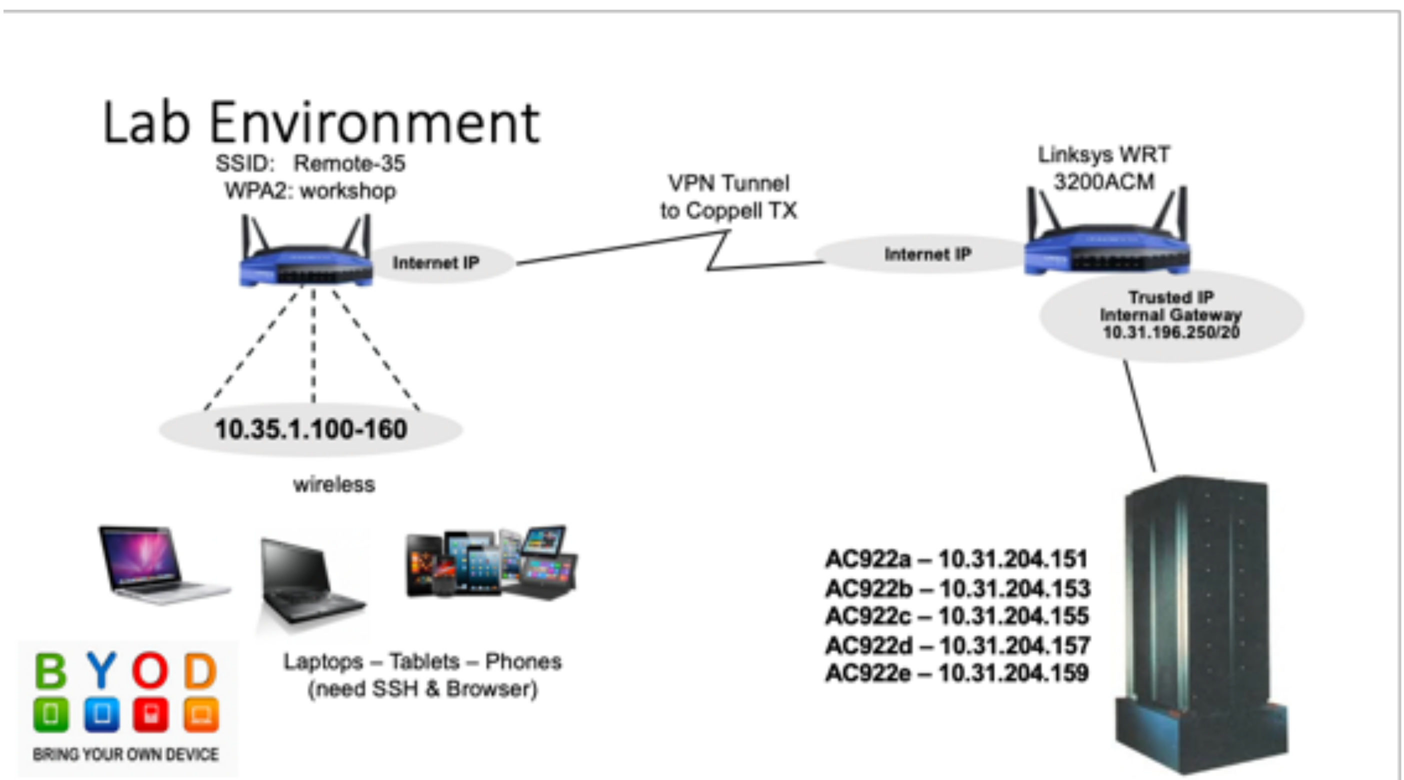
A very special <u>thank you</u> to contributors from IBM Research:

Ton Ngo	**ton@us.ibm.com**	**@tango245**
Paul Van Eck	**pvaneck@us.ibm.com**	**@pvaneckw**
Winnie Tsang	**wtsang@us.ibm.com**	**@wtsang8**

This session is an adaptation of a lab developed and presented by IBM Research. It has been modified slightly to fit within the time constraints for this event.

Logistics:
You will access a POWER8 system running Ubuntu 16.04.3. You will have root access to a container on this system. Within this container we have installed TensorFlow 1.1.0-4. While you will walk through the labs on this host, this lab could be performed other places very easily.

Note: This system does not contain any GPU devices. As such these tasks will run against GPU. It would be ideal to show this lab running in a GPU-based environment to help show additional value.



Using MNIST and TensorFlow
This lab is intended to give users an example of normal (albiet simple) process by which one could use TensorFlow to help identify handwritten characters. This will also expose you to TensorBoard which could be used to visualize or troubleshoot potential problems.

MNIST is a simple computer vision dataset. It consists of images of handwritten digits like these:

It also includes labels for each image, telling us which digit it is. For example, the labels for the images above are 5, 0, 4, and 1.

In this tutorial, we're going to train a model to look at images and predict what digits they are. Our goal isn't to train a really elaborate model that achieves state-of-the-art performance -- although we'll give you code to do that later! -- but rather to dip a toe into using TensorFlow. As such, we're going to start with a very simple model, called a Softmax Regression.

The actual code for this tutorial is very short, and all the interesting stuff happens in just three lines. However, it is very important to understand the ideas behind it: both how TensorFlow works and the core machine learning concepts. Because of this, we are going to very carefully work through the code.

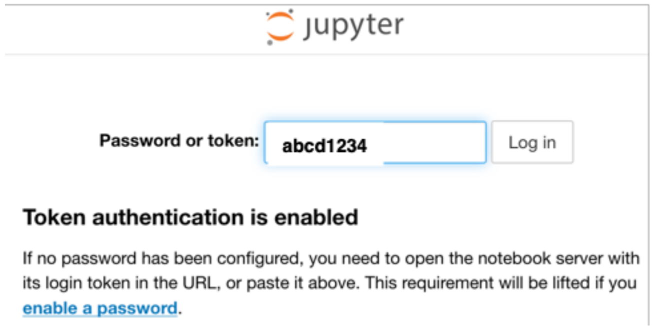
GOALS:

- **Learn about the MNIST data and softmax regressions**
- **Create a function that is a model for recognizing digits, based on looking at every pixel in the image**
- **Use TensorFlow to train the model to recognize digits by having it "look" at thousands of examples (and run our first TensorFlow session to do so)**
- **Check the model's accuracy with our test data**
- **Understand how this could run on a CPU (40x faster on a Volta V100)**

Team 01	http://10.31.204.151:**8801**
Team 02	http://10.31.204.151:**8802**
Team 03	http://10.31.204.151:**8803**
Team 04	http://10.31.204.151:**8804**
Team 05	http://10.31.204.153:**8805**
Team 06	http://10.31.204.153:**8806**
Team 07	http://10.31.204.153:**8807**
Team 08	http://10.31.204.153:**8808**
Team 09	http://10.31.204.155:**8809**
Team 10	http://10.31.204.155:**8810**
Team 11	http://10.31.204.155:**8811**
Team 12	http://10.31.204.155:**8812**
Team 13	http://10.31.204.157:**8813**
Team 14	http://10.31.204.157:**8814**
Team 15	http://10.31.204.157:**8815**
Team 16	http://10.31.204.157:**8816**
Team 17	http://10.31.204.159:**8817**
Team 18	http://10.31.204.159:**8818**
Team 19	http://10.31.204.159:**8819**
Team 20	http://10.31.204.159:**8820**

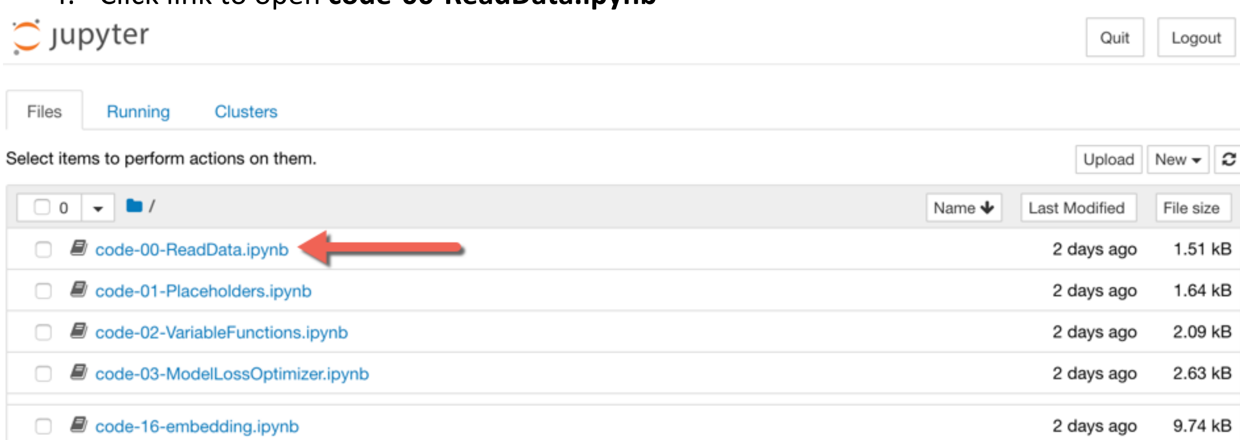
1. Browse to your unique Docker container: **http://10.31.204.___:88##**

2. Password or Token: **abcd1234**



3. Notice you have 17 iPython notebooks (code-00 through code-16). Code-00 is the start of program, code-16 is the complete program. We will walk through each step to discuss code snippets that have been progressively added.

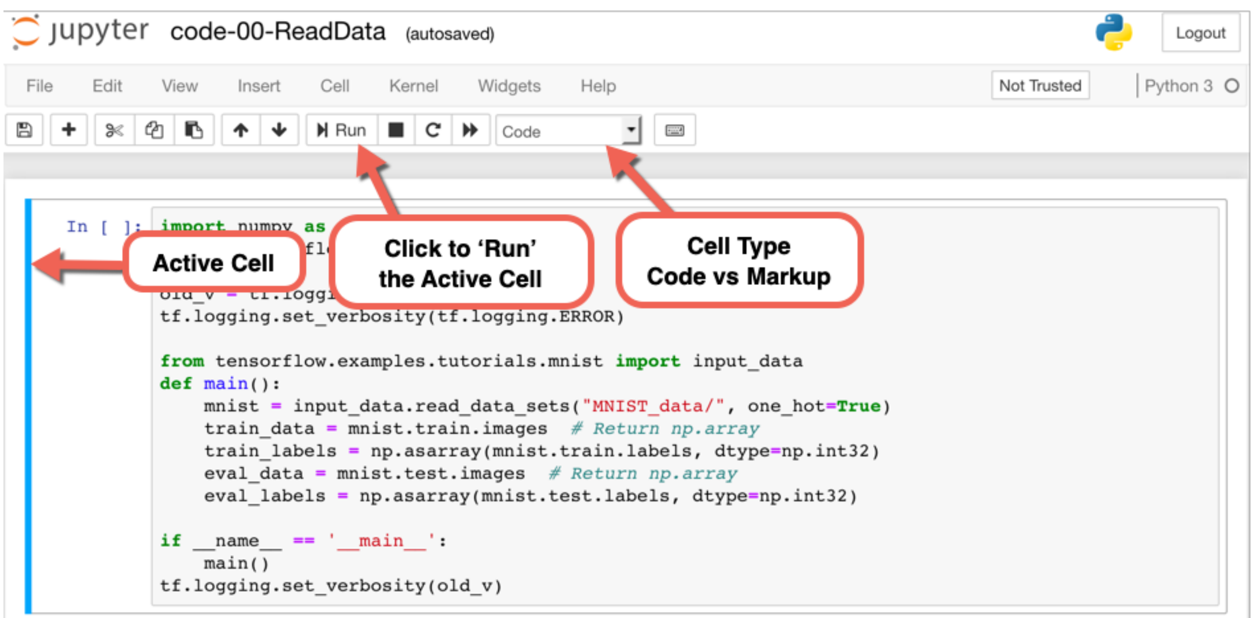
4. Click link to open **code-00-ReadData.ipynb**



Brief details explaining Jupyer:
The ipynb files in /opt/DL/tensorflow are iPython Notebook files created for use by Jupyter notebooks. These notebook files are in JSON format to be created, viewed, edited, manipulated, and run in the Jupyter client within a web browser. Notebook files can contain markup code as well as python commands.

If you're interested in what ipynb files look like, you can open another PuTTY ssh session to your container and inspect 00-classification.ipynb. For additional information on these 'notebook' files, take a look at:
http://jupyternotebook.readthedocs.io/en/latest/notebook.html
Note, however, that .ipynb files are not typically created by hand; they are created using the Jupyter Notebook.

Continuation of Step 4:
In the screen shot below, the blue bar on the left highlights the active cell you are in, and to the right of the control buttons along the top, you can see that this particular cell is a 'Markdown' cell, which is really just a text cell used to document things in the notebook. The run button will step through code, section by section.



code-00-ReadData.ipynb:

Click “Run” in the Jupyter Notebook

```
import numpy as np
import tensorflow as tf

old_v = tf.logging.get_verbosity()
tf.logging.set_verbosity(tf.logging.ERROR)

from tensorflow.examples.tutorials.mnist import input_data
def main():
    mnist = input_data.read_data_sets("MNIST_data/",
one_hot=True)
    train_data = mnist.train.images  # Return np.array
    train_labels = np.asarray(mnist.train.labels,
dtype=np.int32)
    eval_data = mnist.test.images  # Return np.array
    eval_labels = np.asarray(mnist.test.labels,
dtype=np.int32)

if __name__ == '__main__':
    main()
tf.logging.set_verbosity(old_v)
```

This imports MNIST data so it can be accessed and used by TensorFlow.

code-01-Placeholders.ipynb:

(New code added is displayed in **BOLD**)

This new code defines a Placeholder for input: image and label

```
import numpy as np
import tensorflow as tf

old_v = tf.logging.get_verbosity()
tf.logging.set_verbosity(tf.logging.ERROR)

from tensorflow.examples.tutorials.mnist import input_data
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
    train_data = mnist.train.images  # Return np.array
    train_labels = np.asarray(mnist.train.labels, dtype=np.int32)
    eval_data = mnist.test.images  # Return np.array
    eval_labels = np.asarray(mnist.test.labels, dtype=np.int32)

    # Placeholder that will be fed image data.
    x = tf.placeholder(tf.float32, [None, 784])
    # Placeholder that will be fed the correct labels.
    y_ = tf.placeholder(tf.float32, [None, 10])

if __name__ == '__main__':
    main()
tf.logging.set_verbosity(old_v)
```

code-02-VariableFuctions.ipynb:

Define Variables for model – weight and bias:

```
import numpy as np
import tensorflow as tf


def weight_variable(shape):
  """Generates a weight variable of a given shape."""
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)


def bias_variable(shape):
  """Generates a bias variable of a given shape."""
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)


old_v = tf.logging.get_verbosity()
tf.logging.set_verbosity(tf.logging.ERROR)

from tensorflow.examples.tutorials.mnist import input_data
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
    train_data = mnist.train.images  # Return np.array
    train_labels = np.asarray(mnist.train.labels, dtype=np.int32)
    eval_data = mnist.test.images  # Return np.array
    eval_labels = np.asarray(mnist.test.labels, dtype=np.int32)

    # Placeholder that will be fed image data.
    x = tf.placeholder(tf.float32, [None, 784])
    # Placeholder that will be fed the correct labels.
    y_ = tf.placeholder(tf.float32, [None, 10])

if __name__ == '__main__':
    main()
tf.logging.set_verbosity(old_v)
```

code-03-ModelLossOptimizer.ipynb:

Define Loss and Optimzer functions.

```
import numpy as np
import tensorflow as tf


def weight_variable(shape):
  """Generates a weight variable of a given shape."""
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)


def bias_variable(shape):
  """Generates a bias variable of a given shape."""
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)


old_v = tf.logging.get_verbosity()
tf.logging.set_verbosity(tf.logging.ERROR)

from tensorflow.examples.tutorials.mnist import input_data
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
    train_data = mnist.train.images  # Return np.array
    train_labels = np.asarray(mnist.train.labels, dtype=np.int32)
    eval_data = mnist.test.images  # Return np.array
    eval_labels = np.asarray(mnist.test.labels, dtype=np.int32)

    # Placeholder that will be fed image data.
    x = tf.placeholder(tf.float32, [None, 784])
    # Placeholder that will be fed the correct labels.
    y_ = tf.placeholder(tf.float32, [None, 10])
    # Define weight and bias.
    W = weight_variable([784, 10])
    b = bias_variable([10])

    # Here we define our model which utilizes the softmax regression.
    y = tf.nn.softmax(tf.matmul(x, W) + b)

    # Define our loss.
    cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
reduction_indices=[1]))

    # Define our optimizer.
    train_step =
tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

if __name__ == '__main__':
    main()
tf.logging.set_verbosity(old_v)
```

code-04-DefineAccuracy.ipynb:

Add Accuracy Calculation.

```
import numpy as np
import tensorflow as tf

...

def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
    train_data = mnist.train.images  # Return np.array
    train_labels = np.asarray(mnist.train.labels, dtype=np.int32)
    eval_data = mnist.test.images  # Return np.array
    eval_labels = np.asarray(mnist.test.labels, dtype=np.int32)

    # Placeholder that will be fed image data.
    x = tf.placeholder(tf.float32, [None, 784])
    # Placeholder that will be fed the correct labels.
    y_ = tf.placeholder(tf.float32, [None, 10])
    # Define weight and bias.
    W = weight_variable([784, 10])
    b = bias_variable([10])

    # Here we define our model which utilizes the softmax regression.
    y = tf.nn.softmax(tf.matmul(x, W) + b)

    # Define our loss.
    cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
reduction_indices=[1]))

    # Define our optimizer.
    train_step =
tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

    # Define accuracy.
    correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
    correct_prediction = tf.cast(correct_prediction, tf.float32)
    accuracy = tf.reduce_mean(correct_prediction)

if __name__ == '__main__':
    main()
tf.logging.set_verbosity(old_v)
```

code-05-RunGraphWithError.ipynb:

Connect to runtime and run a training graph.

```
import numpy as np
import tensorflow as tf
...

def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

    # Define accuracy.
    correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
    correct_prediction = tf.cast(correct_prediction, tf.float32)
    accuracy = tf.reduce_mean(correct_prediction)

    # Launch session.
    sess = tf.InteractiveSession()

    # Do the training.
    for i in range(1100):
        batch = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch[0], y_: batch[1]})

    # See how model did.
    print("Test Accuracy %g" % sess.run(accuracy, feed_dict=
                                 {x: mnist.test.images,
                                 y_: mnist.test.labels}))

if __name__ == '__main__':
    main()
tf.logging.set_verbosity(old_v)
```

code-06-WorkingBasic.ipynb:

Fix error: initialize variables

```
import numpy as np
import tensorflow as tf
...

def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

    # Launch session.
    sess = tf.InteractiveSession()

    # Initialize variables.
    tf.global_variables_initializer().run()

    # Do the training.
    for i in range(1100):
        batch = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch[0], y_: batch[1]})

    # See how model did.
    print("Test Accuracy %g" % sess.run(accuracy, feed_dict={x:
mnist.test.images,
                                                              y_:
mnist.test.labels}))

if __name__ == '__main__':
    main()
tf.logging.set_verbosity(old_v)
```

code-07-IncreasedBatch.ipynb:

Try a larger batch of images.

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

    # Launch session.
    sess = tf.InteractiveSession()

    # Initialize variables.
    tf.global_variables_initializer().run()

    # Do the training.
    for i in range(1100):
        batch = mnist.train.next_batch(100)
        if i % 100 == 0:
            train_accuracy = sess.run(accuracy, feed_dict={x:batch[0], y_:
batch[1]})
            print("Step %d, Training Accuracy %g" % (i,
float(train_accuracy)))
        sess.run(train_step, feed_dict={x: batch[0], y_: batch[1]})

    # See how model did.
    print("Test Accuracy %g" % sess.run(accuracy, feed_dict={x:
mnist.test.images,
                                                                 y_:
mnist.test.labels}))

if __name__ == '__main__':
    main()
tf.logging.set_verbosity(old_v)
```

code-08-FileWriter.ipynb:

Add FileWriter to visualize with TensorBoard

```
import numpy as np
import tensorflow as tf

LOGDIR = './tensorflow_logs/mnist_deep'
...

def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

    # Launch session.
    sess = tf.InteractiveSession()

    # Initialize variables.
    tf.global_variables_initializer().run()

    # Create summary writer
    writer = tf.summary.FileWriter(LOGDIR, sess.graph)

    # Do the training.
    for i in range(1100):
        batch = mnist.train.next_batch(100)
        if i % 100 == 0:
            train_accuracy = sess.run(accuracy, feed_dict={x:batch[0], y_:
batch[1]})
            print("Step %d, Training Accuracy %g" % (i,
float(train_accuracy)))
        sess.run(train_step, feed_dict={x: batch[0], y_: batch[1]})

    # See how model did.
    print("Test Accuracy %g" % sess.run(accuracy, feed_dict={x:
mnist.test.images, y_: mnist.test.labels}))

    # Close summary writer
    writer.close()


if __name__ == '__main__':
    main()
tf.logging.set_verbosity(old_v)
```

code-09-NameScopes.ipynb:

Add names and name scope to make it easier to read the graph.

```
...
def weight_variable(shape):
  """Generates a weight variable of a given shape."""
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial, name='weight')


def bias_variable(shape):
  """Generates a bias variable of a given shape."""
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial, name='bias')


def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

    # Placeholder that will be fed image data.
    x = tf.placeholder(tf.float32, [None, 784], name='x')
    # Placeholder that will be fed the correct labels.
    y_ = tf.placeholder(tf.float32, [None, 10], name='labels')

    # Define weight and bias.
    W = weight_variable([784, 10])
    b = bias_variable([10])

    # Here we define our model which utilizes the softmax regression.
    with tf.name_scope('softmax'):
        y = tf.nn.softmax(tf.matmul(x, W) + b, name='y')

    # Define our loss.
    with tf.name_scope('loss'):
        cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
reduction_indices=[1]), name='cross_entropy')

    # Define our optimizer.
    with tf.name_scope('optimizer'):
        train_step =
tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy,
name='train_step')

    # Define accuracy.
    with tf.name_scope('accuracy'):
        correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
        correct_prediction = tf.cast(correct_prediction, tf.float32,
name='correct_prediction')
        accuracy = tf.reduce_mean(correct_prediction, name='accuracy')
...
```

code-10-image.ipynb:

Viewing images in TensorBoard

```
import numpy as np
import tensorflow as tf
...

def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

    # Define weight and bias.
    W = weight_variable([784, 10])
    b = bias_variable([10])

    with tf.name_scope('reshape'):
        x_image = tf.reshape(x, [-1, 28, 28, 1])
        tf.summary.image('input', x_image, 4)

    # Here we define our model which utilizes the softmax regression.
    with tf.name_scope('softmax'):
        y = tf.nn.softmax(tf.matmul(x, W) + b, name='y')

...
# Initialize variables.
    tf.global_variables_initializer().run()

    # Merge all the summary data
    merged = tf.summary.merge_all()

    # Create summary writer
    writer = tf.summary.FileWriter(LOGDIR, sess.graph)

    # Do the training.
    for i in range(1100):
        batch = mnist.train.next_batch(100)
        if i % 5 == 0:
            summary = sess.run(merged, feed_dict={x: batch[0], y_: batch[1]})
            writer.add_summary(summary, i)
        if i % 100 == 0:
...
```

code-11-histogram.ipynb:

View line graphs and histograms of variables

```
import numpy as np
import tensorflow as tf

...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

    # Define weight and bias.
    W = weight_variable([784, 10])
    tf.summary.histogram(‘weight’, W)
    b = bias_variable([10])
    tf.summary.histogram(‘bias’, b)
...

    # Here we define our model which utilizes the softmax regression.
    with tf.name_scope('softmax'):
        y = tf.nn.softmax(tf.matmul(x, W) + b, name='y')
        tf.summary.histogram('softmax', y)

    # Define our loss.
    with tf.name_scope('loss'):
        cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
reduction_indices=[1]), name='cross_entropy')
        tf.summary.scalar('loss', cross_entropy)
...

    # Define accuracy.
    with tf.name_scope('accuracy'):
        correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
        correct_prediction = tf.cast(correct_prediction, tf.float32,
name='correct_prediction')
        accuracy = tf.reduce_mean(correct_prediction, name='accuracy')
        tf.summary.scalar('accuracy', accuracy)

    # Launch session.
    sess = tf.InteractiveSession()
...
```

code-12-OneCNN.ipynb:

Create the first Convolutional Layer in the Neural Network (CNN)

```
import numpy as np
import tensorflow as tf

...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

    # Placeholder that will be fed image data.
    x = tf.placeholder(tf.float32, [None, 784], name='x')
    # Placeholder that will be fed the correct labels.
    y_ = tf.placeholder(tf.float32, [None, 10], name='labels')

    # Define weight and bias.
    W = weight_variable([784, 10])
    tf.summary.histogram('weight', W)
    b = bias_variable([10])
    tf.summary.histogram('bias', b)

    # Reshape to use within a convolutional neural net.
    # Last dimension is for "features" - there is only one here, since images 
are
    # grayscale -- it would be 3 for an RGB image, 4 for RGBA, etc.
    with tf.name_scope('reshape'):
        x_image = tf.reshape(x, [-1, 28, 28, 1])
        tf.summary.image('input', x_image, 4)

    # Convolutional layer - maps one grayscale image to 32 features.
    with tf.name_scope('conv1'):
        W_conv1 = weight_variable([5, 5, 1, 32])
        b_conv1 = bias_variable([32])
        x_conv1 = tf.nn.conv2d(x_image, W_conv1, strides=[1, 1, 1, 1], 
padding='SAME')
        h_conv1 = tf.nn.relu(x_conv1 + b_conv1)

    # Pooling layer - downsamples by 2X.
    with tf.name_scope('pool1'):
        h_pool1 = tf.nn.max_pool(h_conv1, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')

    # After downsampling, our 28x28 image is now 14x14
    # with 32 feature maps.
    with tf.name_scope('flatten'):
        h_pool_flat = tf.reshape(h_pool1, [-1, 14*14*32])

    # Map the features to 10 classes, one for each digit
    with tf.name_scope('fc-classify'):

        W_fc2 = weight_variable([14*14*32, 10])
        b_fc2 = bias_variable([10])
        y = tf.matmul(h_pool_flat, W_fc2) + b_fc2
```

```
    # Here we define our model which utilizes the softmax regression.
    with tf.name_scope('softmax'):
        y = tf.nn.softmax(tf.matmul(x, W) + b, name='y')
        tf.summary.histogram('softmax', y)

    ###########################
    # Define our loss.
    with tf.name_scope('loss'):
        # Use more numerically stable cross entropy.
        cross_entropy = tf.reduce_mean(
            tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y),
            name='cross_entropy'
        )
        tf.summary.scalar('loss', cross_entropy)

    # Define our optimizer.
    with tf.name_scope('optimizer'):
        train_step =
tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy,
name='train_step')

    # Define accuracy.
    with tf.name_scope('accuracy'):
        correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
        correct_prediction = tf.cast(correct_prediction, tf.float32,
name='correct_prediction')
        accuracy = tf.reduce_mean(correct_prediction, name='accuracy')
        tf.summary.scalar('accuracy', accuracy)

...
```

code-13-TwoCNN.ipynb:

Create the second Convolution Layer in the Neural Network (CNN)

```
import numpy as np
import tensorflow as tf

...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

    # Pooling layer - downsamples by 2X.
    with tf.name_scope('pool1'):
        h_pool1 = tf.nn.max_pool(h_conv1, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')

    # Second convolutional layer -- maps 32 feature maps to 64.
    with tf.name_scope('conv2'):
        W_conv2 = weight_variable([5, 5, 32, 64])
        b_conv2 = bias_variable([64])
        x_conv2 = tf.nn.conv2d(h_pool1, W_conv2, strides=[1, 1, 1, 1],
padding='SAME')
        h_conv2 = tf.nn.relu(x_conv2 + b_conv2)

    # Second pooling layer.
    with tf.name_scope('pool2'):
        h_pool2 = tf.nn.max_pool(h_conv2, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')

    # After 2 rounds of downsampling, our 28x28 image
    # is down to 7x7 with 64 feature maps.
    with tf.name_scope('flatten'):
        h_pool_flat = tf.reshape(h_pool2, [-1, 7*7*64])

    # Map the features to 10 classes, one for each digit
    with tf.name_scope('fc-classify'):
        W_fc2 = weight_variable([7*7*64, 10])
        b_fc2 = bias_variable([10])
        y = tf.matmul(h_pool_flat, W_fc2) + b_fc2

# Define our optimizer.
    with tf.name_scope('optimizer'):
        train_step = tf.train.AdamOptimizer(0.0001).minimize(cross_entropy,
name='train_step')
...
```

code-14-FullConnect.ipynb:

Create the Fully connected layer in the Neural Network

```
import numpy as np
import tensorflow as tf

...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

    # After 2 rounds of downsampling, our 28x28 image
    # is down to 7x7 with 64 feature maps.
    with tf.name_scope('fc1'):
        h_pool_flat = tf.reshape(h_pool2, [-1, 7*7*64])
        W_fc1 = weight_variable([7*7*64, 1024])
        b_fc1 = bias_variable([1024])
        h_fc1 = tf.nn.relu(tf.matmul(h_pool_flat, W_fc1) + b_fc1)

    # Map the features to 10 classes, one for each digit
    with tf.name_scope('fc-classify'):
        W_fc2 = weight_variable([1024, 10])
        b_fc2 = bias_variable([10])
        y = tf.matmul(h_fc1, W_fc2) + b_fc2
...
```

code-15DropOut.ipynb:

Add the dropout layer in the neural network to control overfitting.

```
import numpy as np
import tensorflow as tf

...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

# After 2 rounds of downsampling, our 28x28 image
    # is down to 7x7 with 64 feature maps.
    with tf.name_scope('fc1'):
        h_pool_flat = tf.reshape(h_pool2, [-1, 7*7*64])
        W_fc1 = weight_variable([7*7*64, 1024])
        b_fc1 = bias_variable([1024])
        h_fc1 = tf.nn.relu(tf.matmul(h_pool_flat, W_fc1) + b_fc1)

    # Dropout - controls the complexity of the model, prevents co-adaptation
of
    # features.
    with tf.name_scope('dropout'):
        keep_prob = tf.placeholder(tf.float32)
        h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

    # Map the features to 10 classes, one for each digit
    with tf.name_scope('fc-classify'):
        W_fc2 = weight_variable([1024, 10])
        b_fc2 = bias_variable([10])
        y = tf.matmul(h_fc1_drop, W_fc2) + b_fc2

...

    # Do the training.
    for i in range(1100):
        batch = mnist.train.next_batch(100)
        if i % 5 == 0:
            summary = sess.run(merged, feed_dict={x: batch[0], y_: batch[1],
keep_prob: 1.0})
            writer.add_summary(summary, i)
        if i % 100 == 0:
            train_accuracy = sess.run(accuracy, feed_dict={x:batch[0], y_:
batch[1], keep_prob: 1.0})
            print("Step %d, Training Accuracy %g" % (i,
float(train_accuracy)))
        sess.run(train_step, feed_dict={x: batch[0], y_: batch[1], keep_prob:
0.5})

    # See how model did.
```

```
    print("Test Accuracy %g" % sess.run(accuracy, feed_dict={x:
mnist.test.images,
                                                               y_:
mnist.test.labels,
                                                               keep_prob:
1.0}))
...
```

code-16-embedding.ipynb:

Add full visualization for all the layers.

```
import os
import numpy as np
import tensorflow as tf
import sys
import urllib.request

if sys.version_info[0] >= 3:
  from urllib.request import urlretrieve
else:
  from urllib import urlretrieve

LOGDIR = './tensorflow_logs/mnist_deep'
_deep'
...

def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

# Convolutional layer - maps one grayscale image to 32 features.
    with tf.name_scope('conv1'):
        W_conv1 = weight_variable([5, 5, 1, 32])
        b_conv1 = bias_variable([32])
        x_conv1 = tf.nn.conv2d(x_image, W_conv1, strides=[1, 1, 1, 1],
padding='SAME')
        h_conv1 = tf.nn.relu(x_conv1 + b_conv1)
        tf.summary.histogram("weights", W_conv1)
        tf.summary.histogram("biases", b_conv1)
        tf.summary.histogram("activations", h_conv1)

    # Pooling layer - downsamples by 2X.
    with tf.name_scope('pool1'):
        h_pool1 = tf.nn.max_pool(h_conv1, ksize=[1, 2, 2, 1],
                                strides=[1, 2, 2, 1], padding='SAME')
        # Display the image after max pooling on tensorboard
        h_pool1_image = tf.reshape(h_pool1, [-1, 14, 14, 1])
        tf.summary.image('conv1', h_pool1_image, 4)

    # Second convolutional layer -- maps 32 feature maps to 64.
    with tf.name_scope('conv2'):
        W_conv2 = weight_variable([5, 5, 32, 64])
        b_conv2 = bias_variable([64])
        x_conv2 = tf.nn.conv2d(h_pool1, W_conv2, strides=[1, 1, 1, 1],
padding='SAME')
        h_conv2 = tf.nn.relu(x_conv2 + b_conv2)
        tf.summary.histogram("weights", W_conv2)
        tf.summary.histogram("biases", b_conv2)
        tf.summary.histogram("activations", h_conv2)

    # Second pooling layer.
```

```
    with tf.name_scope('pool2'):
        h_pool2 = tf.nn.max_pool(h_conv2, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
        # Display the image after max pooling on tensorboard
        h_pool2_image = tf.reshape(h_pool2, [-1, 7, 7, 1])
        tf.summary.image('conv2', h_pool2_image, 4)

    # After 2 rounds of downsampling, our 28x28 image
    # is down to 7x7 with 64 feature maps.
    with tf.name_scope('fc1'):
        h_pool_flat = tf.reshape(h_pool2, [-1, 7*7*64])
        W_fc1 = weight_variable([7*7*64, 1024])
        b_fc1 = bias_variable([1024])
        h_fc1 = tf.nn.relu(tf.matmul(h_pool_flat, W_fc1) + b_fc1)
        tf.summary.histogram("weights", W_fc1)
        tf.summary.histogram("biases", b_fc1)
        tf.summary.histogram("activations", h_fc1)

...
    # Create summary writer
    writer = tf.summary.FileWriter(LOGDIR, sess.graph)

    # Get sprite and labels file for the embedding projector
    GITHUB_URL ='https://raw.githubusercontent.com/mamcgrath/TensorBoard-TF-
Dev-Summit-Tutorial/master/'
    urlretrieve(GITHUB_URL + 'labels_1024.tsv', os.path.join(LOGDIR,
'labels_1024.tsv'))
    urlretrieve(GITHUB_URL + 'sprite_1024.png', os.path.join(LOGDIR,
'sprite_1024.png'))

    # Setup embedding visualization
    embedding = tf.Variable(tf.zeros([1024, 1024]), name="test_embedding")
    assignment = embedding.assign(h_fc1_drop)
    saver = tf.train.Saver()

    config = tf.contrib.tensorboard.plugins.projector.ProjectorConfig()
    embedding_config = config.embeddings.add()
    embedding_config.tensor_name = embedding.name
    embedding_config.sprite.image_path = 'sprite_1024.png'
    embedding_config.metadata_path = 'labels_1024.tsv'
    # Specify the width and height of a single thumbnail.
    embedding_config.sprite.single_image_dim.extend([28, 28])
    tf.contrib.tensorboard.plugins.projector.visualize_embeddings(writer,
config)

    # Do the training.
    for i in range(1100):
        batch = mnist.train.next_batch(100)
        if i % 5 == 0:
            summary = sess.run(merged, feed_dict={x: batch[0], y_: batch[1],
keep_prob: 1.0})
            writer.add_summary(summary, i)
        if i % 100 == 0:
            train_accuracy = sess.run(accuracy, feed_dict={x:batch[0], y_:
batch[1], keep_prob: 1.0})
            print("Step %d, Training Accuracy %g" % (i,
float(train_accuracy)))
```

```
        if i % 500 == 0:
            sess.run(assignment, feed_dict={x: mnist.test.images[:1024], y_:
mnist.test.labels[:1024], keep_prob: 1.0
})
            saver.save(sess, os.path.join(LOGDIR, "model.ckpt"), i)
        sess.run(train_step, feed_dict={x: batch[0], y_: batch[1], keep_prob:
0.5})

    # See how model did.
    print("Test Accuracy %g" % sess.run(accuracy, feed_dict={x:
mnist.test.images,
                                                              y_:
mnist.test.labels,
                                                              keep_prob:
1.0}))

    # Close summary writer
    writer.close()

if __name__ == '__main__':
    main()
tf.logging.set_verbosity(old_v)
```

This concludes your hands-on lab. We will give a brief tour of TensorBoard as time permits. You can also get much more detail here:

https://www.tensorflow.org/get_started/summaries_and_tensorboard