**UT Arlington / IBM**

**Deep Learning Workshop**

****

**Hands-on TensorFlow Programming**

**Arlington**

**October 22, 2019**

**James Nash**

**jjnash@us.ibm.com**

**A very special thank you to contributors from IBM Research:**

**Ton Ngo** [**ton@us.ibm.com**](mailto:ton@us.ibm.com) **@tango245**

**Paul Van Eck** [**pvaneck@us.ibm.com**](mailto:pvaneck@us.ibm.com) **@pvaneckw**

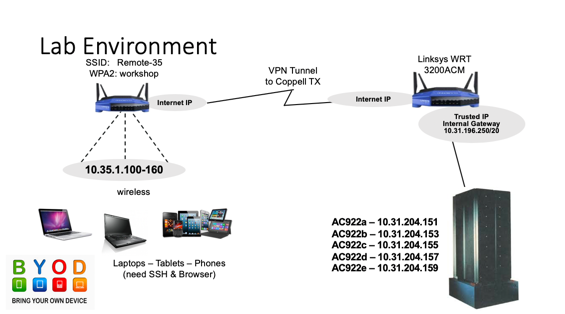
**Winnie Tsang** [**wtsang@us.ibm.com**](mailto:wtsang@us.ibm.com) **@wtsang8**

**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 POWER9 system running Ubuntu 18.04. You will have root access to a container on this system. Within this container we have installed TensorFlow 1.13. While you will walk through the labs on this host, this lab could be performed other places very easily.**

**Note: This system does contain Nvidia Volta V100 GPU devices. As such these tasks could run against GPU.**

****

**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.

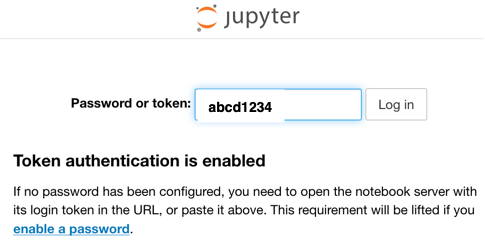
**GOALS:**

* **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)**

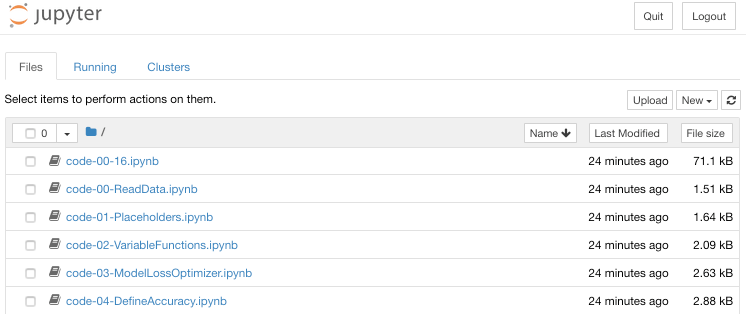


|  |  |
| --- | --- |
| **Team 01** | http://10.31.204.151:**8801** |
| **Team 02** | http://10.31.204.151:**8802** |
| **Team 03** | http://10.31.204.151:**8803** |
| **Team 04** | http://10.31.204.151:**8804** |
| **Team 05** | http://10.31.204.153:**8805** |
| **Team 06** | http://10.31.204.153:**8806** |
| **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** |

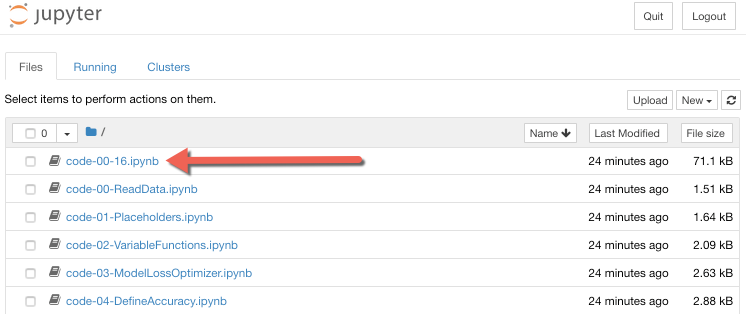
1. Browse to your unique Docker container: [**http://10.31.204.\_\_\_:88##**](http://10.31.204.___:88)
2. Password or Token: **abcd1234**



1. Notice you have several different iPython notebooks (code-00 through code-16). Code-00 is the start of program, code-16 is the complete program. We will use a single notebook that has 17 different code segments (code-00-16.ipynb).



1. Click link to open **code-00-16.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 view them via github:

<https://github.com/jjnash/power-tf>

For additional information on these 'notebook' files, take a look at:

<http://jupyternotebook.readthedocs.io/en/latest/notebook.html>

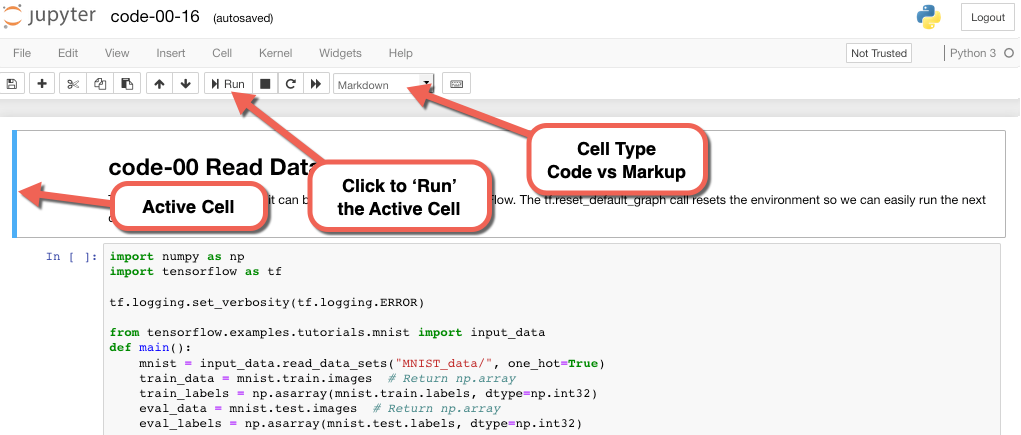
Note, however, that .ipynb files are not typically created by hand; they are created using the Jupyter Notebook.

**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 Read Data:**

**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:**

(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 Variable Fuctions:**

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 Model Loss and Optimizer:**

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 Define Accuracy:**

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 Run Graph with Error:**

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 Working Basic Model:**

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 Increased Batch Size:**

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 File Writer:**

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 Name Scopes:**

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 Images:**

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 Histograms:**

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 One CNN:**

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 Two CNN:**

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 Fully Connected:**

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-15 Drop Out:**

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 Embeddings:**

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()

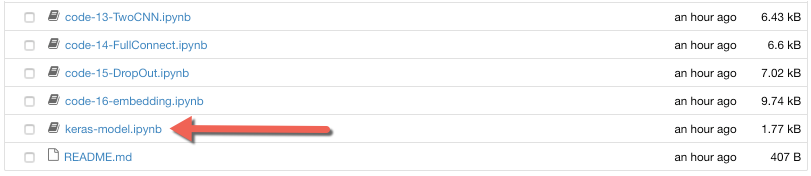
if \_\_name\_\_ == '\_\_main\_\_':

main()

tf.logging.set\_verbosity(old\_v)

**keras-model.ipynb:**

Close and Halt the code-00-16.ipynb notebook. Open the keras-model.ipynb notebook.



import tensorflow as tf

mnist = tf.keras.datasets.mnist

tf.logging.set\_verbosity(tf.logging.ERROR)

(x\_train, y\_train),(x\_test, y\_test) = mnist.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

model = tf.keras.models.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)),

tf.keras.layers.Dense(512, activation=tf.nn.relu),

tf.keras.layers.Dropout(0.2),

tf.keras.layers.Dense(10, activation=tf.nn.softmax)

])

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

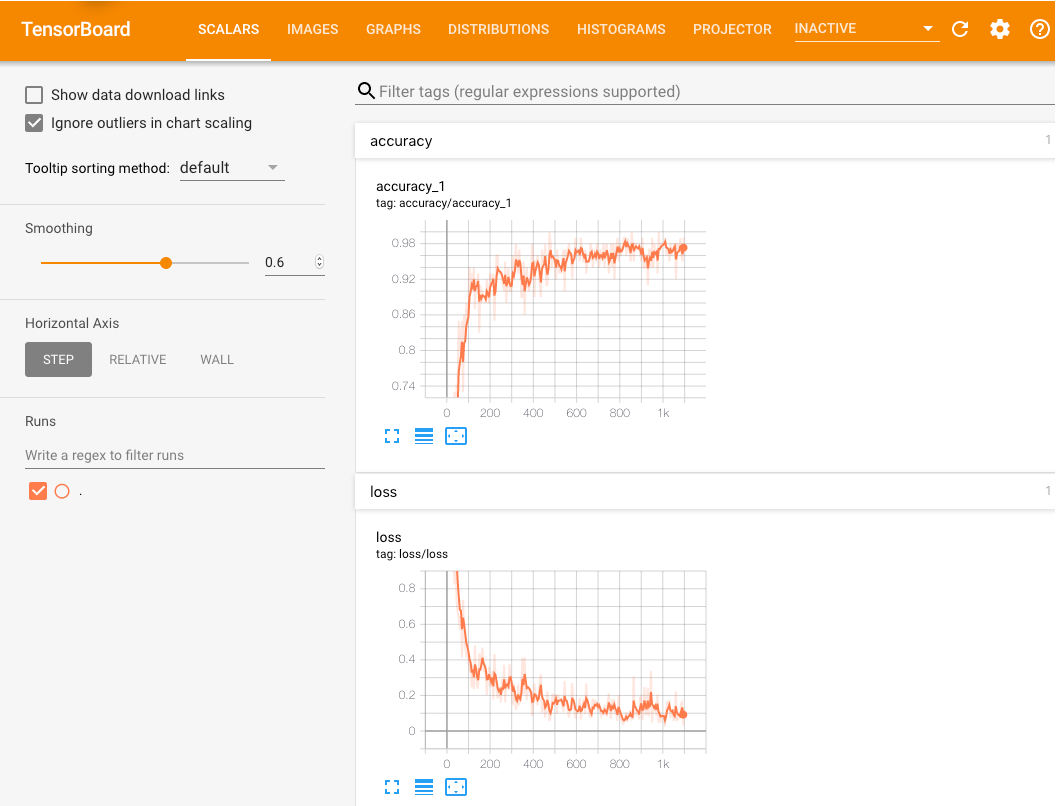
model.fit(x\_train, y\_train, epochs=5)

model.evaluate(x\_test, y\_test)

**TensorBoard – Visualization**

Use a Web browser to connect to TensorBoard:

**http://10.31.204.###:60##**

****

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>



|  |  |
| --- | --- |
| **Team 01** | http://10.31.204.151:**6001** |
| **Team 02** | http://10.31.204.151:**6002** |
| **Team 03** | http://10.31.204.151:**6003** |
| **Team 04** | http://10.31.204.151:**6004** |
| **Team 05** | http://10.31.204.153:**6005** |
| **Team 06** | http://10.31.204.153:**6006** |
| **Team 07** | http://10.31.204.153:**6007** |
| **Team 08** | http://10.31.204.153:**6008** |
| **Team 09** | http://10.31.204.155:**6009** |
| **Team 10** | http://10.31.204.155:**6010** |
| **Team 11** | http://10.31.204.155:**6011** |
| **Team 12** | http://10.31.204.155:**6012** |
| **Team 13** | http://10.31.204.157:**6013** |
| **Team 14** | http://10.31.204.157:**6014** |
| **Team 15** | http://10.31.204.157:**6015** |
| **Team 16** | http://10.31.204.157:**6016** |
| **Team 17** | http://10.31.204.159:**6017** |
| **Team 18** | http://10.31.204.159:**6018** |
| **Team 19** | http://10.31.204.159:**6019** |
| **Team 20** | http://10.31.204.159:**6020** |