Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

Checklist

MNIST Softmax Regression
  [X] Included commented code
  [X] Included good answers for Questions 1.1 to 1.3

MNIST Convolution
  [X] Included 2 versions of commented code
  [X] Included good answers for Questions 2.1 to 2.12
  [X] Included 2 requested graph images.

Vector representations of words
  [X] Included commented code
  [X] Included requested outputs
  [X] Included good answers for Questions 3.1 to 3.5

[X] Did NOT include the MNIST data with my submission

Part 1: Install Tensor flow

```
loop-depaulsecure-252-197:~ Bru$ sudo easy_install pip
Password:
Searching for pip
Reading https://pypi.python.org/simple/pip/
Best match: pip 9.0.1
Downloading https://pypi.python.org/packages/11/b6/abcb525026a4be042b486df43905d6893fb04f05aac21c32c638e939e447/pip-9.0.1.tar.gz#md5=35f01da33009719497f01a4ba69d63c9
Processing pip-9.0.1.tar.gz
Writing /tmp/easy_install-qn5jDH/pip-9.0.1/setup.cfg
Running pip-9.0.1/setup.py -q bdist_egg --dist-dir /tmp/easy_install-qn5jDH/pip-9.0.1/egg-dist-tmp-TjBCBq
/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/distutils/dist.py:267: UserWarning: Unknown distribution option: 'python_requires'
  warnings.warn(msg)
warning: no previously-included files found matching '.coveragerc'
warning: no previously-included files found matching '.mailmap'
warning: no previously-included files found matching '.travis.yml'
warning: no previously-included files found matching '.landscape.yml'
warning: no previously-included files found matching 'pip/_vendor/Makefile'
warning: no previously-included files found matching 'tox.ini'
warning: no previously-included files found matching 'dev-requirements.txt'
warning: no previously-included files found matching 'appveyor.yml'
no previously-included directories found matching '.github'
no previously-included directories found matching '.travis'
no previously-included directories found matching 'docs/_build'
no previously-included directories found matching 'contrib'
no previously-included directories found matching 'tasks'
no previously-included directories found matching 'tests'
creating /Library/Python/2.7/site-packages/pip-9.0.1-py2.7.egg
Extracting pip-9.0.1-py2.7.egg to /Library/Python/2.7/site-packages
Adding pip 9.0.1 to easy-install.pth file
Installing pip script to /usr/local/bin
Installing pip2.7 script to /usr/local/bin
Installing pip2 script to /usr/local/bin

Installed /Library/Python/2.7/site-packages/pip-9.0.1-py2.7.egg
Processing dependencies for pip
Finished processing dependencies for pip
loop-depaulsecure-252-197:~ Bru$ sudo pip install --upgrade virtualenv
The directory '/Users/Bru/Library/Caches/pip/http' or its parent directory is not owned by the current user and the cache has been disabled. Please check the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.
The directory '/Users/Bru/Library/Caches/pip' or its parent directory is not owned by the current user and caching wheels has been disabled. check the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.
Collecting virtualenv
  Downloading virtualenv-15.1.0-py2.py3-none-any.whl (1.8MB)
    100% |████████████████████████████████| 1.8MB 709kB/s
Installing collected packages: virtualenv
Successfully installed virtualenv-15.1.0
loop-depaulsecure-252-197:~ Bru$
```

Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

```
Last login: Mon Oct 30 14:55:25 on ttys000
Brundas-MacBook-Pro:~ Bru$ pwd
/Users/Bru
Brundas-MacBook-Pro:~ Bru$ sudo easy_install pip
Password:
Searching for pip
Best match: pip 9.0.1
Processing pip-9.0.1-py2.7.egg
pip 9.0.1 is already the active version in easy-install.pth
Installing pip script to /usr/local/bin
Installing pip2.7 script to /usr/local/bin
Installing pip2 script to /usr/local/bin

Using /Library/Python/2.7/site-packages/pip-9.0.1-py2.7.egg
Processing dependencies for pip
Finished processing dependencies for pip
Brundas-MacBook-Pro:~ Bru$ pip install --upgrade virtualenv
Requirement already up-to-date: virtualenv in /Library/Python/2.7/site-packages
Brundas-MacBook-Pro:~ Bru$ virtualenv --system-site-packages /Users/Bru/tensorflow
New python executable in /Users/Bru/tensorflow/bin/python
Installing setuptools, pip, wheel...done.
Brundas-MacBook-Pro:~ Bru$ source /Users/Bru/tensorflow/bin/activate
(tensorflow) Brundas-MacBook-Pro:~ Bru$ easy_install -U pip
Searching for pip
Reading https://pypi.python.org/simple/pip/
Downloading https://pypi.python.org/packages/11/b6/abcb525026a4be042b486df43905d6893fb04f05aac21c32c638e939e447/pip-9.0.1.tar.gz#md5=35f01da33009719497f01a4ba69d63c9
Best match: pip 9.0.1
Processing pip-9.0.1.tar.gz
Writing /var/folders/pl/790qvyz94hn8lg4thdts5zhh0000gn/T/easy_install-HBmS5k/pip-9.0.1/setup.cfg
Running pip-9.0.1/setup.py -q bdist_egg --dist-dir /var/folders/pl/790qvyz94hn8lg4thdts5zhh0000gn/T/easy_install-HBmS5k/pip-9.0.1/egg-dist-tmp-NKHMNV
warning: no previously-included files found matching '.coveragerc'
warning: no previously-included files found matching '.mailmap'
warning: no previously-included files found matching '.travis.yml'
warning: no previously-included files found matching '.landscape.yml'
warning: no previously-included files found matching 'pip/_vendor/Makefile'
warning: no previously-included files found matching 'tox.ini'
warning: no previously-included files found matching 'dev-requirements.txt'
warning: no previously-included files found matching 'appveyor.yml'
no previously-included directories found matching '.github'
no previously-included directories found matching '.travis'
no previously-included directories found matching 'docs/_build'
no previously-included directories found matching 'contrib'
no previously-included directories found matching 'tasks'
no previously-included directories found matching 'tests'
creating /Users/Bru/tensorflow/lib/python2.7/site-packages/pip-9.0.1-py2.7.egg
Extracting pip-9.0.1-py2.7.egg to /Users/Bru/tensorflow/lib/python2.7/site-packages
Adding pip 9.0.1 to easy-install.pth file
Installing pip script to /Users/Bru/tensorflow/bin
Installing pip2.7 script to /Users/Bru/tensorflow/bin
Installing pip2 script to /Users/Bru/tensorflow/bin

Installed /Users/Bru/tensorflow/lib/python2.7/site-packages/pip-9.0.1-py2.7.egg
Processing dependencies for pip
Finished processing dependencies for pip
(tensorflow) Brundas-MacBook-Pro:~ Bru$
```

```
(tensorflow) Brundas-MacBook-Pro:~ Bru$ pip install --upgrade tensorflow
Collecting tensorflow
  Downloading tensorflow-1.4.0-cp27-cp27m-macosx_10_11_x86_64.whl (38.8MB)
    100% |████████████████████████████████| 38.9MB 33kB/s
Collecting enum34>=1.1.6 (from tensorflow)
  Downloading enum34-1.1.6-py2-none-any.whl
Collecting six>=1.10.0 (from tensorflow)
  Downloading six-1.11.0-py2.py3-none-any.whl
Collecting protobuf>=3.3.0 (from tensorflow)
  Downloading protobuf-3.4.0-py2.py3-none-any.whl (375kB)
    100% |████████████████████████████████| 378kB 2.6MB/s
Collecting numpy>=1.12.1 (from tensorflow)
  Downloading numpy-1.13.3-cp27-cp27m-macosx_10_6_intel.macosx_10_9_intel.macosx_10_9_x86_64.macosx_10_10_intel.macosx_10_10_x86_64.whl (4.6MB)
    100% |████████████████████████████████| 4.6MB 244kB/s
Requirement already up-to-date: wheel in ./tensorflow/lib/python2.7/site-packages (from tensorflow)
Collecting backports.weakref>=1.0rc1 (from tensorflow)
  Downloading backports.weakref-1.0.post1-py2.py3-none-any.whl
Collecting tensorflow-tensorboard<0.5.0,>=0.4.0rc1 (from tensorflow)
  Downloading tensorflow_tensorboard-0.4.0rc2-py2-none-any.whl (1.7MB)
    100% |████████████████████████████████| 1.7MB 783kB/s
Collecting mock>=2.0.0 (from tensorflow)
  Downloading mock-2.0.0-py2.py3-none-any.whl (56kB)
    100% |████████████████████████████████| 61kB 3.7MB/s
Requirement already up-to-date: setuptools in ./tensorflow/lib/python2.7/site-packages (from protobuf>=3.3.0->tensorflow)
Collecting futures>=3.1.1; python_version < "3.2" (from tensorflow-tensorboard<0.5.0,>=0.4.0rc1->tensorflow)
  Downloading futures-3.1.1-py2-none-any.whl
Collecting werkzeug>=0.11.10 (from tensorflow-tensorboard<0.5.0,>=0.4.0rc1->tensorflow)
  Downloading Werkzeug-0.12.2-py2.py3-none-any.whl (312kB)
    100% |████████████████████████████████| 317kB 2.0MB/s
Collecting html5lib==0.9999999 (from tensorflow-tensorboard<0.5.0,>=0.4.0rc1->tensorflow)
  Downloading html5lib-0.9999999.tar.gz (889kB)
    100% |████████████████████████████████| 890kB 1.3MB/s
Collecting markdown>=2.6.8 (from tensorflow-tensorboard<0.5.0,>=0.4.0rc1->tensorflow)
  Downloading Markdown-2.6.9.tar.gz (271kB)
    100% |████████████████████████████████| 276kB 2.8MB/s
Collecting bleach==1.5.0 (from tensorflow-tensorboard<0.5.0,>=0.4.0rc1->tensorflow)
  Downloading bleach-1.5.0-py2.py3-none-any.whl
Collecting funcsigs>=1; python_version < "3.3" (from mock>=2.0.0->tensorflow)
  Downloading funcsigs-1.0.2-py2.py3-none-any.whl
Collecting pbr>=0.11 (from mock>=2.0.0->tensorflow)
  Downloading pbr-3.1.1-py2.py3-none-any.whl (99kB)
    100% |████████████████████████████████| 102kB 5.0MB/s
Building wheels for collected packages: html5lib, markdown
  Running setup.py bdist_wheel for html5lib ... done
  Stored in directory: /Users/Bru/Library/Caches/pip/wheels/6f/85/6c/56b8e1292c6214c4eb73b9dda50f53e8e977bf65989373c962
  Running setup.py bdist_wheel for markdown ... done
  Stored in directory: /Users/Bru/Library/Caches/pip/wheels/bf/46/10/c93e17ae86ae3b3a919c7b39dad3b5ccf09aeb066419e5c1e5
Successfully built html5lib markdown
Installing collected packages: enum34, six, protobuf, numpy, backports.weakref, futures, werkzeug, html5lib, markdown, bleach, tensorflow-tensorboard, funcsigs, pbr, mock, tensorflow
  Found existing installation: six 1.4.1
    Not uninstalling six at /System/Library/Frameworks/Python.framework/Versions/2.7/Extras/lib/python, outside environment /Users/Bru/tensorflow
  Found existing installation: numpy 1.8.0rc1
    Not uninstalling numpy at /System/Library/Frameworks/Python.framework/Versions/2.7/Extras/lib/python, outside environment /Users/Bru/tensorflow
Successfully installed backports.weakref-1.0.post1 bleach-1.5.0 enum34-1.1.6 funcsigs-1.0.2 futures-3.1.1 html5lib-0.9999999 markdown-2.6.9 mock-2.0.0 numpy-1.13.3 pbr-3.1.1 protobuf-3.4.0 six-1.11.0 tens
orflow-1.4.0 tensorflow-tensorboard-0.4.0rc2 werkzeug-0.12.2
```

Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

Part 2: MNIST Softmax Regression

Code from the tutorial:

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

#Import tensor flow and other required libraries
import argparse
import sys
import tensorflow as tf

#import mnist data from tensorflow examples
from tensorflow.examples.tutorials.mnist import input_data
FLAGS = None

def main(_):
  # Load mnist data
  mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)


  # Create the model

  #Create a placeholder for each input image of size 28*28 pixels
  x = tf.placeholder(tf.float32, [None, 784])
  #Create a variable to hold weights for the data and initializing W as tensors full of zeros
 # W is a 784x10 matrix – 780 input features and 10 outputs

  W = tf.Variable(tf.zeros([784, 10]))

  # Create a variable to hold biases and initialize b as tensors full of zeros
  # b is a 10-dimensional vector
  b = tf.Variable(tf.zeros([10]))

  #Multiply the input vectorized images by the weights matrix and add the bias          component b
  y = tf.matmul(x, W) + b

  # Define loss and optimizer
  # Create a placeholder for each of target output where each row is a one-hot 10-dimensional vector
indicating which digit class (zero through nine) the corresponding MNIST image belongs to.
  y_ = tf.placeholder(tf.float32, [None, 10])

  #cross entropy is the loss function and softmax is the activation function applied to the model's prediction
  # Applies the softmax on the model's unnormalized model prediction and sums across all classes, and
tf.reduce_mean takes the average over these sums
  cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))

 #Train the model - uses gradient descent with a step length of 0.5 to descend the cross entropy
  train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)


 # Create a tensor flow interactive session to handle computational graph objects
```

```
        sess = tf.InteractiveSession()
        tf.global_variables_initializer().run()

       # Train the model for 1000 epochs
        for _ in range(1000):
          batch_xs, batch_ys = mnist.train.next_batch(100) #Load 100 instances of the training set for each
iteration
       # Run the train step repeatedly. feed_dict is used to replace the placeholder tensors x and y_ with the
training examples
          sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})


        # Evaluating the trained model
        # Calculate the number of correct predictions – using tf.equal to check if our prediction matches the actual
true label. Outputs a list of booleans
        correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))

        #Calculate the percentage of correct predictions
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

        # Test and print accuracy of the test data
        print(sess.run(accuracy, feed_dict={x: mnist.test.images,
                                            y_: mnist.test.labels}))


      if __name__ == '__main__':
        parser = argparse.ArgumentParser()
        parser.add_argument('--data_dir', type=str, default='/tmp/tensorflow/mnist/input_data',
                            help='Directory for storing input data')
        FLAGS, unparsed = parser.parse_known_args()
        tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

Question 1.1: What was the final accuracy?

→ 0.9186 or 91.86%

```
>>> from tensorflow.examples.tutorials.mnist import input_data
>>> mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Extracting MNIST_data/train-images-idx3-ubyte.gz
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Extracting MNIST_data/train-labels-idx1-ubyte.gz

Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
[Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
[>>>
>>> import tensorflow as tf
>>> sess = tf.InteractiveSession()
>>> x = tf.placeholder(tf.float32, shape=[None, 784])
>>> y_ = tf.placeholder(tf.float32, shape=[None, 10])
>>> W = tf.Variable(tf.zeros([784,10]))
>>> b = tf.Variable(tf.zeros([10]))
>>> sess.run(tf.global_variables_initializer())
[>>> y = tf.matmul(x,W) + b
>>> cross_entropy = tf.reduce_mean(
[...     tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
>>> train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
[>>> for _ in range(1000):
[...   batch = mnist.train.next_batch(100)
[...   train_step.run(feed_dict={x: batch[0], y_: batch[1]})
...
[>>> correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
[>>> accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
>>> print(accuracy.eval(feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
0.9186
[>>>
[
```

Change the code to make it run for 10,000 epochs and run it again.

Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

Question 1.2: What was the final accuracy after this modification
→ 0.9215 or 92.15%

```
>>> for _ in range(10000):
...   batch = mnist.train.next_batch(100)
...   train_step.run(feed_dict={x: batch[0], y_: batch[1]})
...

>>>
>>> correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
>>> accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
>>> print(accuracy.eval(feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
0.9215
>>>
```

Question 1.3: Was the difference surprising or not? Explain.
→ The difference in accuracy is not very huge considering it is 10000 epochs i.e. 10 times the initial number of epochs. However, considering this is a one layer regression I did think the difference was surprising.

Part 3: MNIST Multilayer Convolutional Network

VERSION 1:
Code from the tutorial:

```
"""
Spyder Editor
Brunda Chouthoy
Mnist deep convolutional network
"""
#import mnist data from tensorflow examples
from tensorflow.examples.tutorials.mnist import input_data
#Load mnist data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)

#Import tensor flow and other required libraries
import tensorflow as tf

#Placeholders
#Create a placeholder for each input image of size 28*28 pixels
x = tf.placeholder(tf.float32, shape=[None, 784])
'''Create a placeholder for each of target output where each row is a one-hot
10-dimensional vector indicating which digit class the corresponding
MNIST image belongs to.'''
y_ = tf.placeholder(tf.float32, shape=[None, 10])

#Weight initialization
```

Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

```
'''Function initilizes weights with a small amount of noise
to prevent 0 gradients
Input argument: shape
Returns the weight variable'''
def weight_variable(shape):
  #intilize weights with a std deviation of 0.1
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)

'''Function initilizes with a slightly positive initial bias
to avoid dead neurons
Input argument: shape
Returns the bias variable'''
def bias_variable(shape):
  #initialize a positive bias to avoid dead neurons
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)

#Convolution and Pooling
''' Function applies convolution to layers in the network
Input arguments: x - input images of size 28*28 pixels and
W - weights corresponding to the input matrix'''
def conv2d(x, W):
  #Strides window shifts by 1 in all dimensions and zero padding so that output
  #is same size as output
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

''' Function applies pooling to layers in the network
Input argument: x - input images of size 28*28 pixels'''
def max_pool_2x2(x):
  '''max_pool method takes the parameters ksize i.e kernel size with a 2*2 window,
  strides window shifts
  zero padding so that output is same as input
  max pooling over 2x2 blocks'''
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')

#Implementing the First layer - Convolution computes 32 features
#weight tensor of shape 5*5 patch, 1 input channel and 32 output channels
W_conv1 = weight_variable([5, 5, 1, 32])
#bias vector with a component for each output channel
b_conv1 = bias_variable([32])

#Reshaping the input to 28*28 pixels and 1 color channel
x_image = tf.reshape(x, [-1, 28, 28, 1])

#For the first convolutional layer - convolve the reshaped input with weight tensor,
#apply the relu activation function and add the bias vector
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
```

Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

```
#Invoke the max pool method to reduce the image size to 14*14
h_pool1 = max_pool_2x2(h_conv1)

#Second convolutional layer will have 64 features for each 5x5 patch
#weight tensor with 5*5 patch, 32 input channels and 64 output channels
W_conv2 = weight_variable([5, 5, 32, 64])
#bias vector with a component for each output channel
b_conv2 = bias_variable([64])

#For the second convolutional layer - convolve the reshaped input with weight tensor,
#apply the relu activation function and add the bias vector
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
#Invoke the max pool method to reduce the image size
h_pool2 = max_pool_2x2(h_conv2)

#Densely connected layer - add a fully-connected layer with 1024 neurons to
#allow processing on the entire image
W_fc1 = weight_variable([7 * 7 * 64, 1024])
#bias vector with a component for each of the 1024 output channels
b_fc1 = bias_variable([1024])

#Reshaping the tensor from the pooling layer into a batch of vectors
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
#Multiply the reshaped tensor with the weight matrix, add the bias component
#and apply the relu activation function
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

#Dropout
#create a placeholder for the probability that a neuron's output is kept during dropout
keep_prob = tf.placeholder(tf.float32)
#tf.nn.dropout op handles scaling neuron outputs in addition to masking them,
#so dropout should work without any additional scaling
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

#Readout layer
#Weight matrix for the final layer with shape - 1024*10
W_fc2 = weight_variable([1024, 10])
#bias vector for the readout layer
b_fc2 = bias_variable([10])

#Multiply the dropout output with weight matrix and add the bias component
y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2

#Train and evaluate the model
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))
#Using Adam optimizer for gradient descent to descend entropy
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
```

Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

```
#Calculate the number of correct predictions – using tf.equal to check if our prediction matches the actual true label.
#Outputs a list of booleans
correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
#Calculate the percentage of correct predictions
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

#Using tf.sessionto separate Model specification and the process of evaluating
#the graph
with tf.Session() as sess:
  #initialize all the variables in the session created
  sess.run(tf.global_variables_initializer())
  #20000 training iterations or epochs
  for i in range(20000):
    #loading 50 training instances for each iteration
    batch = mnist.train.next_batch(50)
    #For 100 steps
    if i % 100 == 0:
      #computing training accuracy for each input image and output class
      #keep_prob is set to 1 to control dropout rate
      train_accuracy = accuracy.eval(feed_dict={
          x: batch[0], y_: batch[1], keep_prob: 1.0})
      #Printing Step number and training accuract for each step
      print('step %d, training accuracy %g' % (i, train_accuracy))
    #computing training accuracy for each input image and output class
    #keep_prob is set to 0.5 to control dropout rate
    train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})

  #Compute and print test accuracy for the test data
  print('test accuracy %g' % accuracy.eval(feed_dict={
      x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
```

Question 2.1: What was the final accuracy (on the test set, of course!) for the convolutional model?
→ 0.9917 or 99.17%

```
(tensorflow) Brundas-MacBook-Pro:Desktop Bru$ python DeepCNN_mnist.py
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Extracting MNIST_data/train-images-idx3-ubyte.gz
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
2017-11-03 14:48:46.404209: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports instructions that this TensorFlow binary was not compiled to use: SSE4.1 SSE4.2
 AVX AVX2 FMA
step 0, training accuracy 0.08
step 100, training accuracy 0.88
step 200, training accuracy 0.92
step 300, training accuracy 0.84
step 400, training accuracy 0.94
step 500, training accuracy 0.96
step 600, training accuracy 0.92
step 700, training accuracy 0.98
step 800, training accuracy 1
step 900, training accuracy 1
step 1000, training accuracy 0.92
step 1100, training accuracy 0.98
step 1200, training accuracy 1
step 1300, training accuracy 1
step 1400, training accuracy 0.94
step 1500, training accuracy 1
step 1600, training accuracy 0.96
step 1700, training accuracy 0.98
step 1800, training accuracy 0.98
```

```
step 17000, training accuracy 1
step 17100, training accuracy 1
step 17200, training accuracy 1
step 17300, training accuracy 1
step 17400, training accuracy 1
step 17500, training accuracy 1
step 17600, training accuracy 1
step 17700, training accuracy 1
step 17800, training accuracy 1
step 17900, training accuracy 1
step 18000, training accuracy 1
step 18100, training accuracy 1
step 18200, training accuracy 1
step 18300, training accuracy 0.98
step 18400, training accuracy 1
step 18500, training accuracy 1
step 18600, training accuracy 1
step 18700, training accuracy 0.98
step 18800, training accuracy 1
step 18900, training accuracy 1
step 19000, training accuracy 1
step 19100, training accuracy 1
step 19200, training accuracy 1
step 19300, training accuracy 1
step 19400, training accuracy 1
step 19500, training accuracy 1
step 19600, training accuracy 1
step 19700, training accuracy 1
step 19800, training accuracy 1
step 19900, training accuracy 1
test accuracy 0.9917
(tensorflow) Brundas-MacBook-Pro:Desktop Bru$
```

VERSION 2: MNIST Multilayer Convolutional Network with variable summaries

Code with variable summaries:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Nov  3 14:55:54 2017
Brunda Chouthoy
CSC 578
Project 3: MultiLayer Convolutional Neural network with Summaries
```

Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

```
"""

#import mnist data from tensorflow examples
from tensorflow.examples.tutorials.mnist import input_data
#Load mnist data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)

#Import tensor flow and other required libraries
import tensorflow as tf

sess = tf.InteractiveSession()

#Placeholders
#Create a placeholder for each input image of size 28*28 pixels
x = tf.placeholder(tf.float32, shape=[None, 784])
#Create a placeholder for each of target output where each row is a one-hot
#10-dimensional vector indicating which digit class the corresponding
#MNIST image belongs to.
y_ = tf.placeholder(tf.float32, shape=[None, 10])


# tell it where to write info
summaries_dir = '/tmp/mnist_logs'

# Function to create variable summaries.
# Input arguments: variable, name
# Creates a scope, and then summaries for mean, sd, max, min and histogram.
def variable_summaries(var,name):
  '''Attach a lot of summaries to a Tensor (for TensorBoard visualization)'''
  with tf.name_scope('summaries'):
    mean = tf.reduce_mean(var)
    tf.summary.scalar('mean' + name, mean)
    with tf.name_scope('stddev'):
      stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
    tf.summary.scalar('stddev' + name, stddev)
    tf.summary.scalar('max' + name, tf.reduce_max(var))
    tf.summary.scalar('min' + name, tf.reduce_min(var))
    tf.summary.histogram('histogram' + name, var)


#Weight initialization
#Function initilizes weights with a small amount of noise
#to prevent 0 gradients
#Input argument: shape
#Returns the weight variable
def weight_variable(shape):
  #intilize weights with a std deviation of 0.1
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)
```

Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

```
#Function initilizes with a slightly positive initial bias
#to avoid dead neurons
#Input argument: shape
#Returns the bias variable
def bias_variable(shape):
  #initialize a positive bias to avoid dead neurons
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)

#Convolution and Pooling
#Function applies convolution to layers in the network
#Input arguments: x - input images of size 28*28 pixels and
#W - weights corresponding to the input matrix'''
def conv2d(x, W):
  #Strides window shifts by 1 in all dimensions and zero padding so that output
  #is same size as output
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

#Function applies pooling to layers in the network
#Input argument: x - input images of size 28*28 pixels
def max_pool_2x2(x):
  #max_pool method takes the parameters ksize i.e kernel size with a 2*2 window,
  #strides window shifts and zero padding so that output is same as input
  #max pooling over 2x2 blocks'''
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')


# Creating the naming scope for the first convolutional layer.
# Adding a name scope will ensure layers are grouped together in the graph logically
with tf.name_scope('Conv1'):
    #Implementing the First convolutional layer - Convolution computes 32 feautres
    #weight tensor of shape 5*5 patch, 1 input channel and 32 output channels
    W_conv1 = weight_variable([5, 5, 1, 32])
    #Invokes the variable_summaries function to create the variables with summaries
    variable_summaries(W_conv1, 'Conv1/weights')

    #bias vector with a component for each output channel
    b_conv1 = bias_variable([32])
    variable_summaries(b_conv1, 'Conv1/biases')

    #Reshaping the input to 28*28 pixels and 1 color channel
    x_image = tf.reshape(x, [-1, 28, 28, 1])

    #For the first convolutional layer - convolve the reshaped input with weight tensor,
    #apply the relu activation function and add the bias vector
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
    #Invoke the max pool method to reduce the image size to 14*14
```

```
    h_pool1 = max_pool_2x2(h_conv1)

with tf.name_scope('Conv2'):
    #Second convolutional layer will have 64 features for each 5x5 patch
    #weight tensor with 5*5 patch, 32 input channels and 64 output channels
    W_conv2 = weight_variable([5, 5, 32, 64])
    variable_summaries(W_conv2, 'Conv2/weights')

    #bias vector with a component for each output channel
    b_conv2 = bias_variable([64])
    variable_summaries(b_conv2, 'Conv2/biases')

    #For the second convolutional layer - convolve the reshaped input with weight tensor,
    #apply the relu activation function and add the bias vector
    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
    #Invoke the max pool method to reduce the image size
    h_pool2 = max_pool_2x2(h_conv2)

with tf.name_scope('fc1'):
    #Densely connected layer - add a fully-connected layer with 1024 neurons to
    #allow processing on the entire image
    W_fc1 = weight_variable([7 * 7 * 64, 1024])
    variable_summaries(W_fc1, 'FulCon1/weights')

    #bias vector with a component for each of the 1024 output channels
    b_fc1 = bias_variable([1024])
    variable_summaries(b_fc1, 'FulCon1/biases')

    #Reshaping the tensor from the pooling layer into a batch of vectors
    h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
    #Multiply the reshaped tensor with the weight matrix, add the bias component
    #and apply the relu activation function
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

    #Dropout
    #create a placeholder for the probability that a neuron's output is kept during dropout
    keep_prob = tf.placeholder(tf.float32)
    #tf.nn.dropout op handles scaling neuron outputs in addition to masking them,
    #so dropout should work without any additional scaling
    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

with tf.name_scope('fc2'):
    #Readout layer
    #Weight matrix for the final layer with shape - 1024*10
    W_fc2 = weight_variable([1024, 10])
    variable_summaries(W_fc2, 'FulCon2/weights')

    #bias vector for the readout layer
    b_fc2 = bias_variable([10])
```

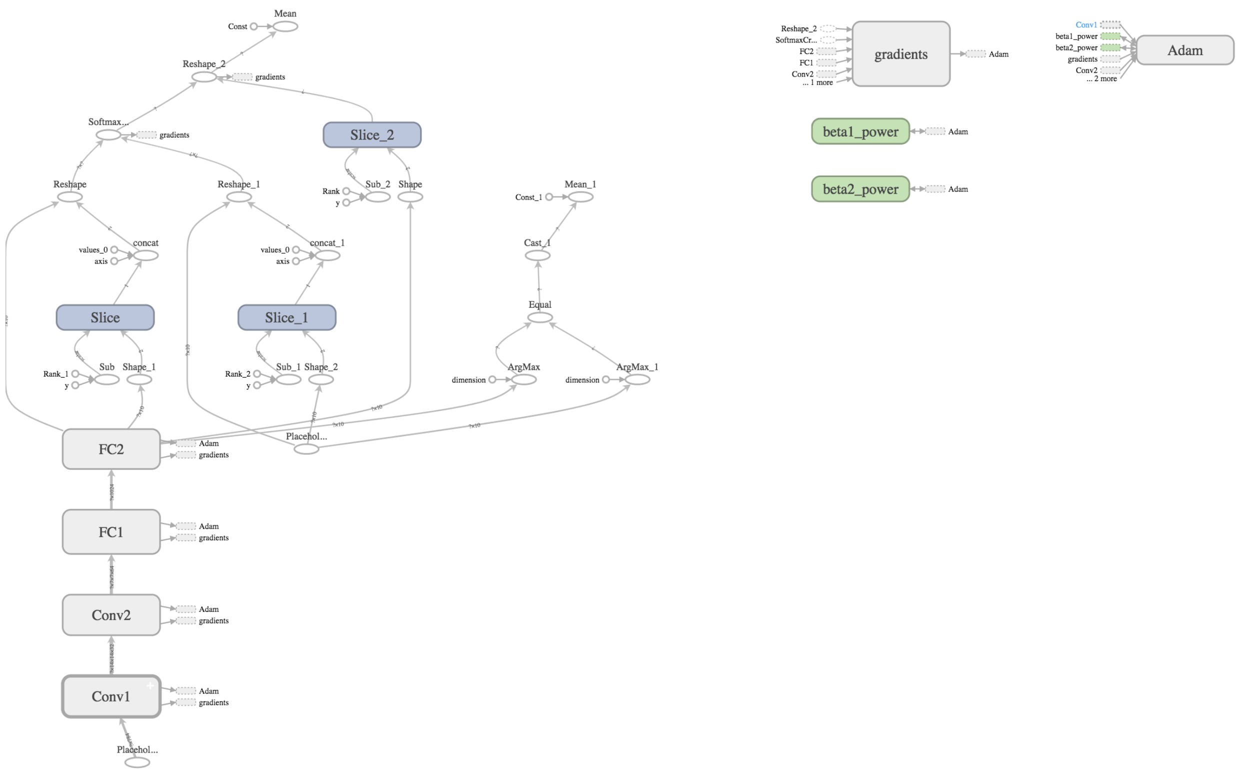
```
    variable_summaries(b_fc2, 'FulCon2/biases')

    #Multiply the dropout output with weight matrix and add the bias component
    y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2

#Train and evaluate the model
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))
#Using Adam optimizer for gradient descent to descend entropy
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
#Calculate the number of correct predictions – using tf.equal to check if our prediction matches the actual
true label.
#Outputs a list of booleans
correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
#Calculate the percentage of correct predictions
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

#Merge all the summaries and write them out to /tmp/mnist_logs
merged = tf.summary.merge_all()
#Using FileWriter to add summaries as we train the model
train_writer = tf.summary.FileWriter(summaries_dir + '/train',sess.graph)
#Using FileWriter to add summaries as we test the model
test_writer = tf.summary.FileWriter(summaries_dir + '/test')
tf.global_variables_initializer().run()

#Function allows to easily switch between feeding in training or testing instances
def feed_dict(train):
    '''Make a TensorFlow feed_dict: maps data onto Tensor placeholders.'''
    if train:
        xs, ys = mnist.train.next_batch(50, False)
        k = 0.5 # Orig value
    else:
        xs, ys = mnist.test.images, mnist.test.labels
        k = 1.0
    return {x: xs, y_: ys, keep_prob: k}

#Train the model and also write summaries.
#For 20000 training iterations or epochs
for i in range(20000):
    # Every 10th step, measure test-set accuracy, and write test summaries
    if i%10 == 0:
        summary, acc = sess.run([merged, accuracy], feed_dict=feed_dict(False))
        # adding summaries for test
        test_writer.add_summary(summary, i)
        print('Accuracy at step %s: %s' % (i, acc))
    else: #All other steps, run train_step on training data & add training summaries
        #Code will emit runtime statistics for every 100th step starting at step 99
        if i%100 == 99:
            run_options = tf.RunOptions(trace_level=tf.RunOptions.FULL_TRACE)
```

```
            run_metadata = tf.RunMetadata()
            # summary is returned by running the session for a train step
            summary, _ = sess.run([merged, train_step],feed_dict=feed_dict(True),
                              options=run_options,run_metadata=run_metadata)
            # adding metadata about the run
            train_writer.add_run_metadata(run_metadata, 'step%03d' % i)
            # adding training summaries
            train_writer.add_summary(summary, i)
            print('Adding run metadata for', i)
        else: # Record a summary
            summary, _ = sess.run([merged, train_step], feed_dict=feed_dict(True))
            # add summary of regular step
            train_writer.add_summary(summary, i)

#closing the file writers
train_writer.close()
test_writer.close()

#final test accuracy
print('test accuracy %g' %accuracy.eval(feed_dict={x: mnist.test.images, y_: mnist.test.labels, keep_prob:
1.0}))
```

Visualize your model:
Graph

Examine the GRAPHS tab. You should see a graph which shows the high-level structure of the tensorflow computation graph you've created, that is, the different layers and how they are connected. Download a PNG of this graph (click on the button on the GUI) so you can include it in your report.

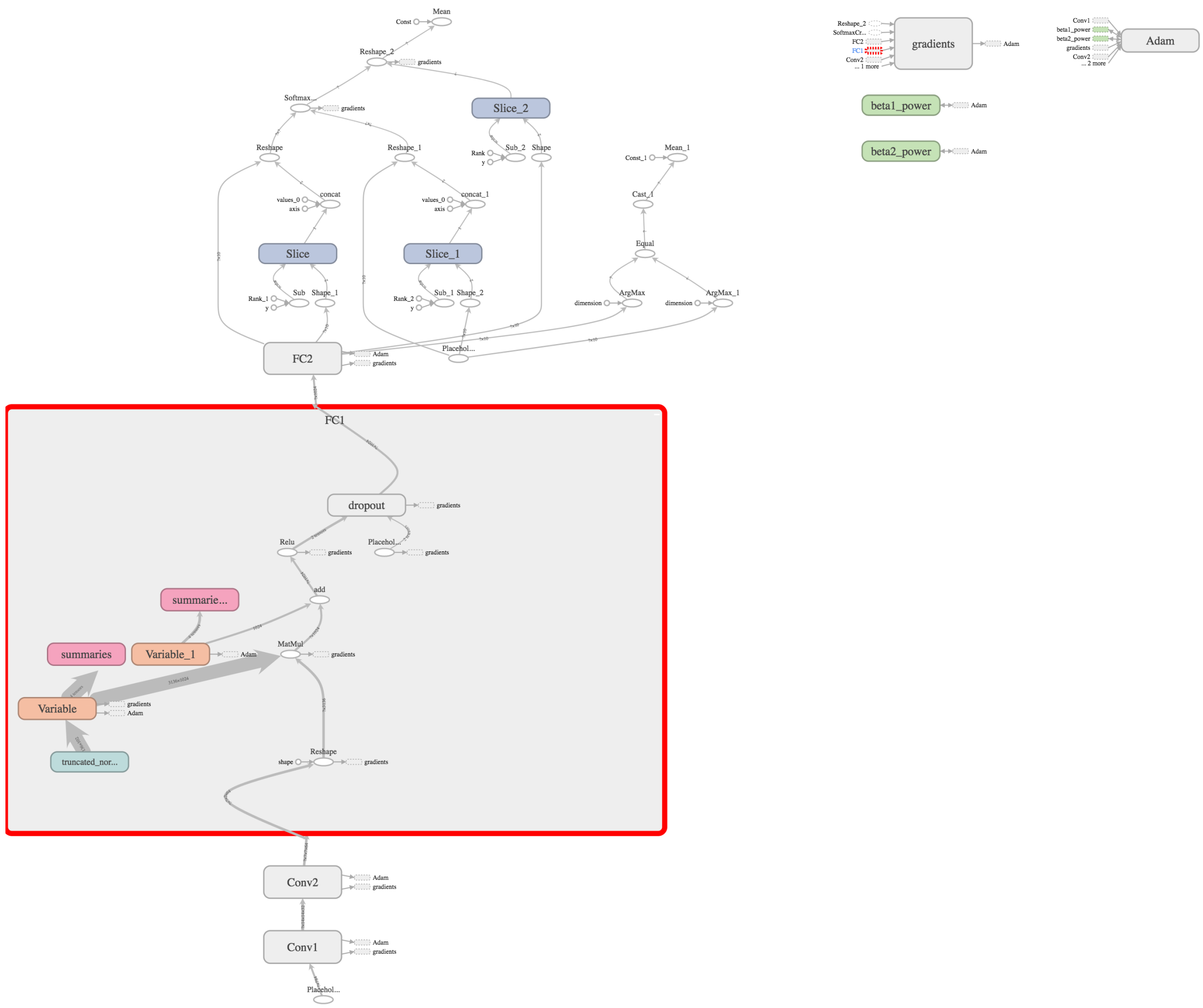
Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow



Expand the node for the first fully connected layer. Download another PNG with it included.

Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

Explore the graph.



Question 2.2: What information can you get out of it?
→ The graph depicts all the network layers and flow of data from input to output. Each node layer (high level node) can be expanded to get the detailed view of the variables and summaries involved. It groups all the layers within each name scope in a meaningful way.

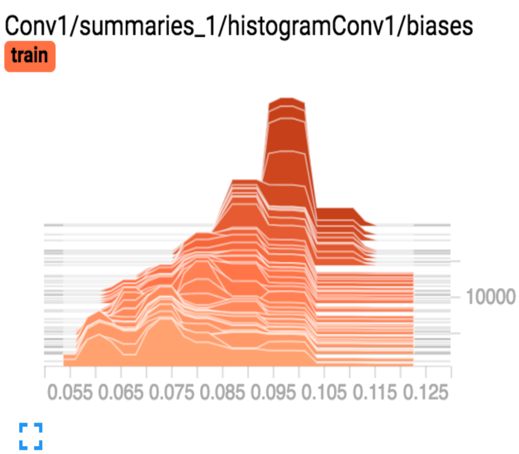
Question 2.3: What do the nodes in the graph tell you?
→ The visualization graph uses special icons for constants and summary nodes. An oval shaped rectangle node like Conv1 represent a high-level node representing a name scope. A circle represents a constant, a stack of ovals represents a sequence of numbered nodes that are connected to each other etc.

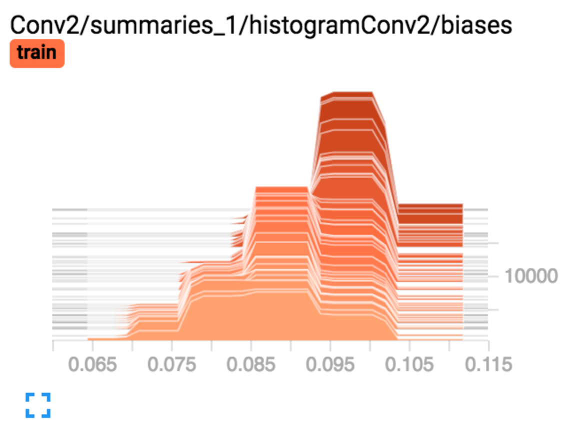
Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

Histograms: After you've run the model for at least a couple hundred epochs, open the HISTOGRAMS tab

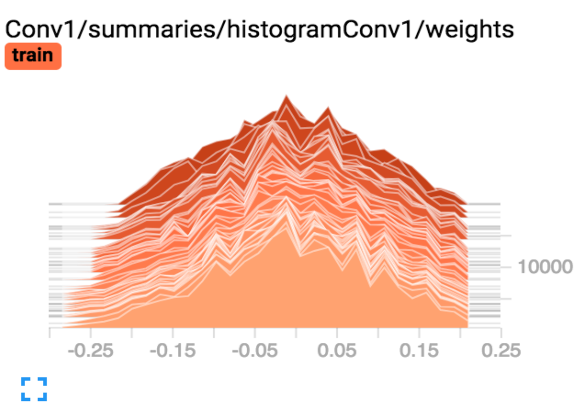
Convolutional layers

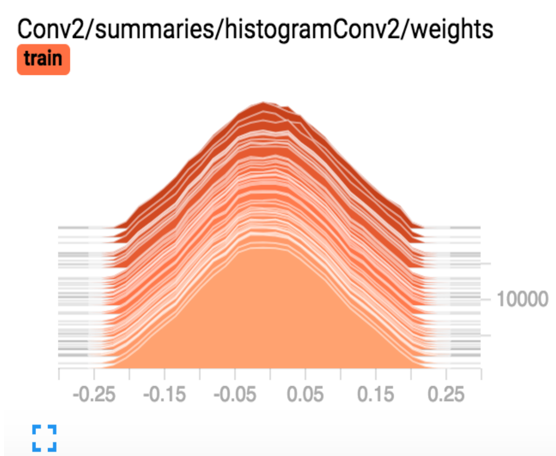
Question 2.4: What do the histograms for the biases show about their distributions?
→ For each of the epochs, histograms are plotted for biases w.r.t frequency values. The biases for conv1 and conv2 are shown below. Looks like the biases uniformly distributed only for a small number of values. Conv2 histogram looks more uniform across iterations/epochs.





Question 2.5: What do the histograms for the weights show about their distributions?
→ The weights look approximately normally distributed across all epochs. The graph for conv2 looks smooth and normally distributed and uniform. For conv1, some spikes and changes in frequencies can be observed.





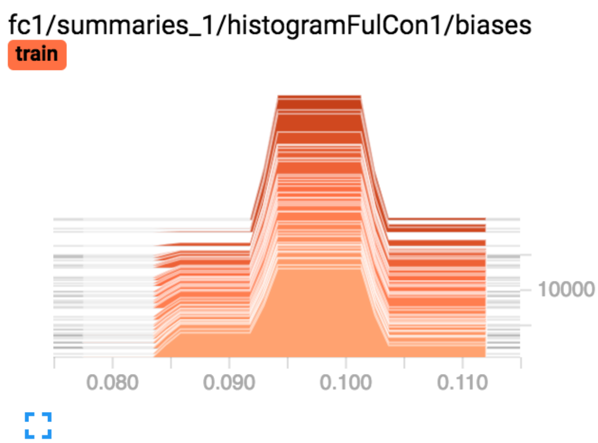
Question 2.6: Anything different about the weights near 0?

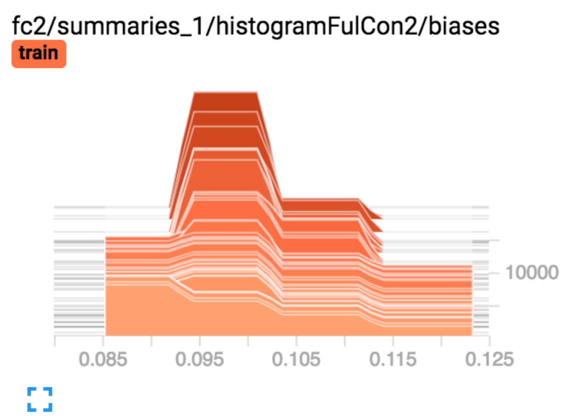
Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

→ For conv1 histogram, there's a dip in the number of weights that can be observed.
For conv2, there is nothing different about the weights near 0, it has a symmetric normal distribution at mean 0.

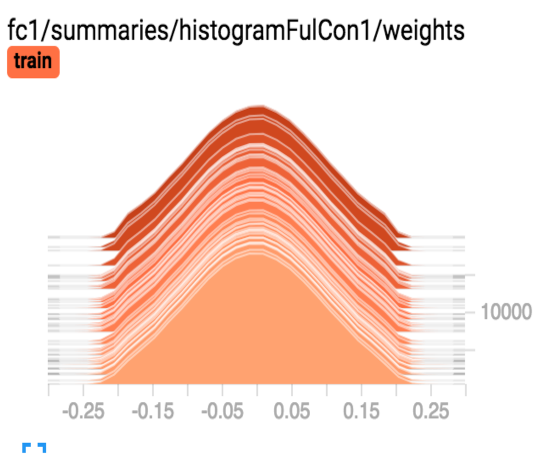
Question 2.7: What is your interpretation of this?
→ For conv2, the peak is at its highest at mean 0 and it depicts a normal distribution. For conv1, the weights show a dip which could mean weights are not constant and are more volatile with respect to frequencies.

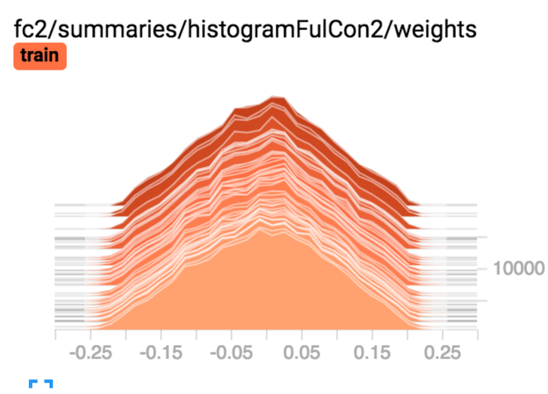
Fully connected layers
Question 2.8: What do the bias histograms show about their distributions?
→fc1 layer looks uniform across all the epochs.
For fc2 layer, the histogram for the bias appears to be right skewed, most of the values fall to right of the curve. Doesn't look uniformly distributed across all the epochs.





Question 2.9: What do the weight histograms show about their distributions?
→ The histogram for weights for both the fully connected layers look normally distributed and uniform. The range of values span from -0.25 to 0.25 with a mean of 0.



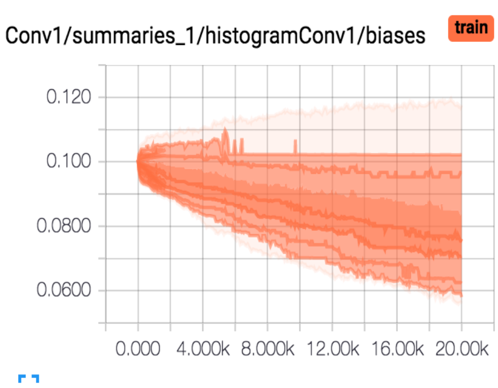


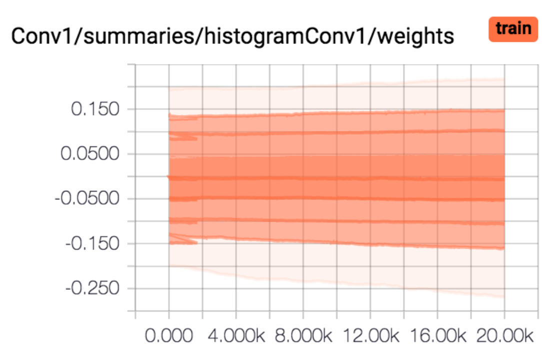
Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

Question 2.10: Anything different about weights near 0?
→ Nothing different about the weights near 0, the distribution is symmetrical at mean 0 and has a normal distribution.

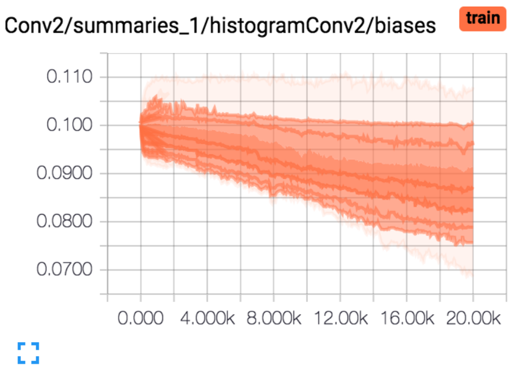
Question 2.11: What is your interpretation of this?
→ The weights and biases appear to be more normal than the previous layers.

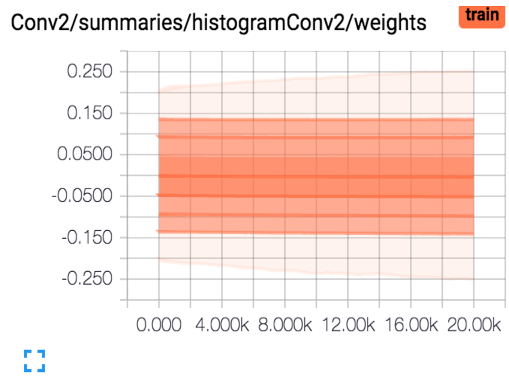
Distributions:
Question 2.12: Can you figure out what is displayed under the DISTRIBUTIONS tab? (I can't, but it looks cool.)
The graphs in the distributions tab depict the distribution of values over the number of epochs/iterations. The graphs below show the weights and biases for both conv1 and conv2 layers for 20000 epochs. X axis represents the epochs and values are on the Y axis. The distribution of weights for both layers look very uniform. The biases are more volatile and are varying with the number of iterations.





Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow





Part 4: Vector Representations of Words

Code from the tutorial:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
'''
Brunda Chouthoy
CSC 578: Project 3
Basic word2vec code from Tensorboard Tutorial'''

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

#Import all the necessary libraries
import collections
import math
import os
import random
from tempfile import gettempdir
import zipfile

import numpy as np
from six.moves import urllib
from six.moves import xrange  # pylint: disable=redefined-builtin
import tensorflow as tf

# Step 1: Download the data from the webURL
url = 'http://mattmahoney.net/dc/'

#Function downloads and retrieves a file from the URL
#Input arguments - filename and size
#Returns name of the file downloaded
```

Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

```
def maybe_download(filename, expected_bytes):
  """Download a file if not present, and make sure it's the right size."""
  local_filename = os.path.join(gettempdir(), filename)
  #Check if the filepath exists locally.
  #If path doesn't exist, download the file
  if not os.path.exists(local_filename):
    local_filename, _ = urllib.request.urlretrieve(url + filename,
                                                   local_filename)
    statinfo = os.stat(local_filename)
  #Check if the filesize is same as the expected size
  if statinfo.st_size == expected_bytes:
    print('Found and verified', filename)
  else: #If not, raise an exception
    print(statinfo.st_size)
    raise Exception('Failed to verify ' + local_filename +
                    '. Can you get to it with a browser?')
  return local_filename

#Invoking the maybe_download method to retrieve filename from the URL
filename = maybe_download('text8.zip', 31344016)

#Function reads the data into a list of strings.
#Input argument: filename
#Returns the data as a list of strings
def read_data(filename):
  """Extract the first file enclosed in a zip file as a list of words."""
  with zipfile.ZipFile(filename) as f:
    #Converts the input as a list of strings
    #Splits each sentence into a list of word strings
    data = tf.compat.as_str(f.read(f.namelist()[0])).split()
  return data

#Invoking the read_data method to get a list of strings
vocabulary = read_data(filename)
#Print the length of all of words
print('Data size', len(vocabulary))

# Step 2: Build the dictionary and replace rare words with UNK token.
vocabulary_size = 50000

#Function processes inputs/words into a dataset and returns a dictionary of key/value pairs
#Input arguments: words from the input file and number of words
#Returns data list, count list and dictionaries with key/word pairs
def build_dataset(words, n_words):
  """Process raw inputs into a dataset."""
  #create a list for counts
  count = [['UNK', -1]]
  #Counter tracks the number of times most common words are added and appends to the count list
  count.extend(collections.Counter(words).most_common(n_words - 1))
```

Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

```
  #Creates a dict object
  dictionary = dict()

  #For each word in the count list
  for word, _ in count:
    #assign size to that particular word
    dictionary[word] = len(dictionary)
  #creates a data list
  data = list()
  unk_count = 0
  #for each word
  for word in words:
    index = dictionary.get(word, 0)
    #If the word not present in the dictionary
    if index == 0:  # dictionary['UNK']
      unk_count += 1 # Assign it to UNK and increase the counter value
    #append the index to the data list
    data.append(index)
  #update the count values
  count[0][1] = unk_count
  #create a dictionary with value(word) and key pairs
  reversed_dictionary = dict(zip(dictionary.values(), dictionary.keys()))
  return data, count, dictionary, reversed_dictionary

# Filling 4 global variables:
# data - list of codes (integers from 0 to vocabulary_size-1).
# This is the original text but words are replaced by their codes
# count - map of words(strings) to count of occurrences
# dictionary - map of words(strings) to their codes(integers)
# reverse_dictionary - maps codes(integers) to words(strings)

#Invoking the build_dataset function
data, count, dictionary, reverse_dictionary = build_dataset(vocabulary,
                                                            vocabulary_size)
del vocabulary  # Hint to reduce memory.
#Prints the 5 most common words
print('Most common words (+UNK)', count[:5])
#Prints the first 10 words and key pairs
print('Sample data', data[:10], [reverse_dictionary[i] for i in data[:10]])

#Initialize data_index to 0
data_index = 0

#Step 3: Function to generate a training batch for the skip-gram model.
#Input arguments: batch size, no of skips - the no of times to reuse the input to generate a label
#and skip window - the window to consider to the left and right of the target
#Returns batch array and labels
def generate_batch(batch_size, num_skips, skip_window):
  global data_index
```

```
  assert batch_size % num_skips == 0
  assert num_skips <= 2 * skip_window
  #Creates an n-dimensional integer array for batch and labels with shape set to batchsize
  batch = np.ndarray(shape=(batch_size), dtype=np.int32)
  labels = np.ndarray(shape=(batch_size, 1), dtype=np.int32)
  #the span for skip window
  span = 2 * skip_window + 1  # [ skip_window target skip_window ]
  buffer = collections.deque(maxlen=span)
  if data_index + span > len(data):
    data_index = 0
  buffer.extend(data[data_index:data_index + span])
  data_index += span
  #Loop by batch_size // num_skips
  for i in range(batch_size // num_skips):
    context_words = [w for w in range(span) if w != skip_window]
    words_to_use = random.sample(context_words, num_skips)
    for j, context_word in enumerate(words_to_use):
      batch[i * num_skips + j] = buffer[skip_window]
      labels[i * num_skips + j, 0] = buffer[context_word]
    if data_index == len(data):
      buffer[:] = data[:span]
      data_index = span
    else:
      buffer.append(data[data_index])
      data_index += 1
  # Backtrack a little bit to avoid skipping words in the end of a batch
  data_index = (data_index + len(data) - span) % len(data)
  return batch, labels

#Invokes the generate_batch function to get the batch and labels
batch, labels = generate_batch(batch_size=8, num_skips=2, skip_window=1)
#Prints the 8 items
for i in range(8):
  print(batch[i], reverse_dictionary[batch[i]],
        '->', labels[i, 0], reverse_dictionary[labels[i, 0]])

# Step 4: Build and train a skip-gram model.

batch_size = 128
embedding_size = 128  # Dimension of the embedding vector.
skip_window = 1       # How many words to consider left and right.
num_skips = 2         # How many times to reuse an input to generate a label.
num_sampled = 64      # Number of negative examples to sample.

# We pick a random validation set to sample nearest neighbors. Here we limit the
# validation samples to the words that have a low numeric ID, which by
# construction are also the most frequent. These 3 variables are used only for
# displaying model accuracy, they don't affect calculation.
valid_size = 16     # Random set of words to evaluate similarity on.
```

Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

```
valid_window = 100  # Only pick dev samples in the head of the distribution.
valid_examples = np.random.choice(valid_window, valid_size, replace=False)


#create a new graph
graph = tf.Graph()

with graph.as_default():

  # Input data.
  #Creates a placeholder for train inputs and labels
  train_inputs = tf.placeholder(tf.int32, shape=[batch_size])
  train_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])
  #creates a constant for valid_examples
  valid_dataset = tf.constant(valid_examples, dtype=tf.int32)

  # Ops and variables pinned to the CPU because of missing GPU implementation
  with tf.device('/cpu:0'):
    # Look up embeddings for inputs.
    #populate the embeddings variable
    embeddings = tf.Variable(
        tf.random_uniform([vocabulary_size, embedding_size], -1.0, 1.0))
    #embedding lookup
    embed = tf.nn.embedding_lookup(embeddings, train_inputs)

    # Construct the variables for the NCE loss
    nce_weights = tf.Variable(
        tf.truncated_normal([vocabulary_size, embedding_size],
                            stddev=1.0 / math.sqrt(embedding_size)))
    nce_biases = tf.Variable(tf.zeros([vocabulary_size]))

  # Compute the average NCE loss for the batch.
  # tf.nce_loss automatically draws a new sample of the negative labels each
  # time we evaluate the loss.
  # Explanation of the meaning of NCE loss:
  #   http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

  #Invokes the nce_loss function after setting all the params
  #NCE or noise contrastive estimation loss is computed
  loss = tf.reduce_mean(
      tf.nn.nce_loss(weights=nce_weights,
                     biases=nce_biases,
                     labels=train_labels,
                     inputs=embed,
                     num_sampled=num_sampled,
                     num_classes=vocabulary_size))

  # Construct the SGD optimizer using a learning rate of 1.0.
  # Minimizing the loss with gradient descent
```

```
  optimizer = tf.train.GradientDescentOptimizer(1.0).minimize(loss)

  # Compute the cosine similarity between minibatch examples and all embeddings.
  norm = tf.sqrt(tf.reduce_sum(tf.square(embeddings), 1, keep_dims=True))
  normalized_embeddings = embeddings / norm
  #Look up valid_embeddings
  valid_embeddings = tf.nn.embedding_lookup(
      normalized_embeddings, valid_dataset)
  #compute similarity
  similarity = tf.matmul(
      valid_embeddings, normalized_embeddings, transpose_b=True)

  # Add variable initializer.
  init = tf.global_variables_initializer()

# Step 5: Begin training.
num_steps = 100001

#Run the session
with tf.Session(graph=graph) as session:
  # We must initialize all variables before we use them.
  init.run()
  print('Initialized')

  #initialize the average loss variable
  average_loss = 0
  #iterating through number of steps
  for step in xrange(num_steps):
    #Invoke generate_batch and feed the input
    batch_inputs, batch_labels = generate_batch(
        batch_size, num_skips, skip_window)
    #call the feed_dict method
    feed_dict = {train_inputs: batch_inputs, train_labels: batch_labels}

    # We perform one update step by evaluating the optimizer op (including it
    # in the list of returned values for session.run()
    _, loss_val = session.run([optimizer, loss], feed_dict=feed_dict)
    average_loss += loss_val

    #Every 2000th step, report the average loss over the last 2000 steps
    if step % 2000 == 0:
      if step > 0:
        average_loss /= 2000
      # The average loss is an estimate of the loss over the last 2000 batches.
      print('Average loss at step ', step, ': ', average_loss)
      average_loss = 0

    # Note that this is expensive (~20% slowdown if computed every 500 steps)
    if step % 10000 == 0:
```

```
      sim = similarity.eval()
      for i in xrange(valid_size):
        valid_word = reverse_dictionary[valid_examples[i]]
        top_k = 8  # number of nearest neighbors
        #Finding the nearest neighbors and writing the log
        nearest = (-sim[i, :]).argsort()[1:top_k + 1]
        log_str = 'Nearest to %s:' % valid_word
        for k in xrange(top_k):
          close_word = reverse_dictionary[nearest[k]]
          #Printing the log string
          log_str = '%s %s,' % (log_str, close_word)
        print(log_str)
  #Evaluate the final embeddings
  final_embeddings = normalized_embeddings.eval()

# Step 6: Visualize the embeddings.

# pylint: disable=missing-docstring
# Function to draw visualization of distance between embeddings.
#Plots a graph with embeddings
def plot_with_labels(low_dim_embs, labels, filename):
  assert low_dim_embs.shape[0] >= len(labels), 'More labels than embeddings'
  plt.figure(figsize=(18, 18))  # in inches
  for i, label in enumerate(labels):
    x, y = low_dim_embs[i, :]
    plt.scatter(x, y)
    plt.annotate(label,
                 xy=(x, y),
                 xytext=(5, 2),
                 textcoords='offset points',
                 ha='right',
                 va='bottom')

  plt.savefig(filename)

#Try block plots the graph for visualizing embeddings
try:
  # pylint: disable=g-import-not-at-top
  from sklearn.manifold import TSNE
  import matplotlib.pyplot as plt

  #visualize high dimensionality data
  #perplexity is the number of nearest neighbors, 5000 iterations, and pca embedding
  tsne = TSNE(perplexity=30, n_components=2, init='pca', n_iter=5000, method='exact')
  plot_only = 500 #plotting only for 500 words
  low_dim_embs = tsne.fit_transform(final_embeddings[:plot_only, :])
  #Generates labels from the word and key pairs
  labels = [reverse_dictionary[i] for i in xrange(plot_only)]
  #invokes the plot function above
```

Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

```
  plot_with_labels(low_dim_embs, labels, os.path.join(gettempdir(), 'tsne.png'))

except ImportError as ex:
  #Raise exception if the required libraries are not imported
  print ('Please install sklearn, matplotlib, and scipy to show embeddings.')
  print(ex)
```

OUTPUT: Run the program 3 times, and save the final results (the average loss at the end, and the final list of nearest neighbors) to include in your project report.

1st Run output:

Average loss at step 92000 : 4.66407999277
Average loss at step 94000 : 4.72407329929
Average loss at step 96000 : 4.68305543387
Average loss at step 98000 : 4.59174929279
Average loss at step 100000 : 4.69754930568
Nearest to its: their, his, the, her, ambush, wishes, gollancz, mico,
Nearest to were: are, have, was, had, be, is, frau, circ,
Nearest to only: but, dasyprocta, three, baralong, birkenau, operatorname, ursus, thaler,
Nearest to three: five, four, six, seven, two, eight, operatorname, dasyprocta,
Nearest to about: sq, sponsors, busan, three, ursus, barrage, mangeshkar, mcduck,
Nearest to by: during, with, be, ursus, as, was, including, microsite,
Nearest to more: less, most, very, musketeers, excellent, greater, gb, yhwh,
Nearest to in: within, at, on, circ, busan, thaler, during, from,
Nearest to one: six, two, seven, four, five, three, eight, microcebus,
Nearest to system: systems, flight, commenced, apocalyptic, mangeshkar, counsellors, space, bokassa,
Nearest to often: usually, generally, commonly, now, also, not, sometimes, still,
Nearest to known: used, such, leuven, uncreated, latinus, seen, well, connected,
Nearest to nine: eight, seven, six, zero, five, four, three, ursus,
Nearest to if: when, though, microcebus, scrip, since, normally, circ, before,
Nearest to it: he, this, there, she, which, they, itself, thaler,
Nearest to than: or, but, dasyprocta, questions, microcebus, no, selfless, seven,

2nd Run output:

Average loss at step 92000 : 4.67041185999
Average loss at step 94000 : 4.72161608398
Average loss at step 96000 : 4.70110002899
Average loss at step 98000 : 4.60581782198
Average loss at step 100000 : 4.71346759093
Nearest to after: during, before, while, when, from, in, but, chymotrypsin,

Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

Nearest to only: pulau, michelob, kilometers, operatorname, but, abakan, cebus, agouti,
Nearest to system: capitolina, pulau, libra, gigantopithecus, systems, dasyprocta, angola, groin,
Nearest to to: michelob, will, would, ursus, not, nine, through, must,
Nearest to such: well, these, many, some, dmd, cartilaginous, other, known,
Nearest to th: six, seven, nine, three, eight, aslan, ulyanov, jackal,
Nearest to world: blues, yakovlev, hardback, scarcity, agouti, gollancz, taira, chymotrypsin,
Nearest to people: michelob, liao, players, iit, critics, agouti, mitral, zero,
Nearest to new: dasyprocta, braveheart, dissertation, nutcracker, second, aveiro, thaler, langle,
Nearest to that: which, however, michelob, this, hbox, but, agouti, upanija,
Nearest to are: were, is, have, cebus, be, do, microcebus, agouti,
Nearest to war: eschatology, wildlife, iit, annexed, projective, trieste, akita, archie,
Nearest to one: two, four, six, three, seven, five, eight, agouti,
Nearest to from: into, in, through, ursus, after, despite, under, by,
Nearest to while: when, but, however, cebus, and, after, though, agouti,
Nearest to he: it, she, they, there, who, never, we, but,

3rd Run output:

Average loss at step 92000 : 4.67366855431
Average loss at step 94000 : 4.70975905395
Average loss at step 96000 : 4.69951351905
Average loss at step 98000 : 4.59363963997
Average loss at step 100000 : 4.6890027945
Nearest to five: four, three, seven, eight, six, two, zero, nine,
Nearest to eight: seven, six, nine, five, four, three, zero, two,
Nearest to than: or, much, and, upanija, but, morisot, albury, pressburg,
Nearest to so: thibetanus, while, michelob, affricate, participated, leontopithecus, upanija, kapoor
Nearest to from: into, in, during, dasyprocta, through, by, somehow, eight,
Nearest to for: kapoor, during, in, of, with, leontopithecus, agouti, stenella,
Nearest to six: seven, eight, five, four, nine, three, two, zero,
Nearest to new: arin, pancreas, ashford, margrave, albury, bath, shipyard, landmarks,
Nearest to which: that, this, but, however, also, it, what, abet,
Nearest to there: they, it, he, which, acapulco, generally, now, aveiro,
Nearest to four: five, three, seven, six, two, eight, zero, nine,
Nearest to used: known, referred, pontificia, circ, leontopithecus, agave, pulau, considered,
Nearest to can: may, will, would, could, must, might, should, cannot,
Nearest to more: less, most, very, rather, filed, basins, thaler, poppies,
Nearest to into: from, through, plastics, between, with, accentual, in, back,
Nearest to zero: eight, five, seven, four, nine, six, three, upanija,

Output:
Question 3.1: In about a paragraph, describe (in your own words, of course) this task. What is the purpose, the data, the general idea for the learning approach?

Brunda Chouthoy
Neural Networks and Deep learning
CSC 578, Project 3: Digging in to Tensor Flow

→ Word2vec is a 2-layer neural network with a text corpus as an input and a set of vectors as the output. Text is converted into a numerical form that a deep neural network can understand. The purpose is to learn word representations in a high dimensional space. Words that occur near each other are considered to have similar meanings are computed using cosine similarity.

Question 3.2: How did it determine which words to find the neighbors of?
→ The program determines the neighbors within the consecutive 3 words that occur in the text corpus. It is predicted by using the skip-gram model to frame a window to determine the relationship between the words in the sentence and finding the cosine similarity of the words from the label.

Question 3.3: Were there any surprises in the results (good or bad)? Give example outputs.
→ Yes. Some of the good surprises –
Nearest to eight: seven, six, nine, five, four, three, zero, two,
Nearest to five: four, three, seven, eight, six, two, zero, nine,
Nearest to he: it, she, they, there, who, never, we, but,
Almost all neighbors are relevant and not much noise in the data.

Some of the bad surprises –
Nearest to so: thibetanus, while, michelob, affricate, participated, leontopithecus, upanija, kapoor,
Nearest to only: pulau, michelob, kilometers, operatorname, but, abakan, cebus, agouti,
Totally unrelated and contains noise.

Question 3.4: What could this be useful for (in your own words)?
→ Word vectors are more powerful and can be used for many applications like word prediction, sentiment analysis and translation. One of the nice applications for word2vec could be for item recommendations like recommending movies, products etc. These can also be used as input to other applications.

Question 3.5: What is noise-contrastive training?
→ Noise-contrastive estimation (NCE) is an estimation principle for unnormalized statistical models. It estimates an unknown distribution Pd by comparing it to a known distribution Pn. The main advantage of NCE is that it allows us to fit models that are not explicitly normalized making the training time effectively independent of the vocabulary size. In the word2vec example, it helps to distinguish target words from noise words.