HU Extension Assignment 11 E63 Big Data Analytics

Handed out: 11/10/2017 Due by 4:00 PM EST on Saturday, 11/18/2017

You are welcome to implement TensorFlow problems in this problem set in any of supported languages.

**Problem 1. Consider provided Jupyter notebook Summaries and NameScopes.ipynb. Add one more output summary. For example, calculate the rolling mean of all one-dimensional tensors passed as arguments of run_graph function. Provide working notebook and images of your graphs and calculated summaries. In the Word document presented as your solution provide snippets of additional or modified code. (15%)**

Code

```
import tensorflow as tf
import numpy as np

# Explicitly create a Graph object
graph = tf.Graph()

with graph.as_default():

  with tf.name_scope("variables"):
    # Variable to keep track of how many times the graph has been run
    global_step = tf.Variable(0, dtype=tf.int32, name="global_step")

    # Variable that keeps track of the sum of all output values over time:
    total_output = tf.Variable(0.0, dtype=tf.float32, name="total_output")

    avgs = tf.Variable(0, dtype=tf.float32, name="avgs")

  # Primary transformation Operations
  with tf.name_scope("transformation"):

    # Separate input layer
    with tf.name_scope("input"):
      # Create input placeholder- takes in a Vector
      a = tf.placeholder(tf.float32, shape=[None],
name="input_placeholder_a")
```

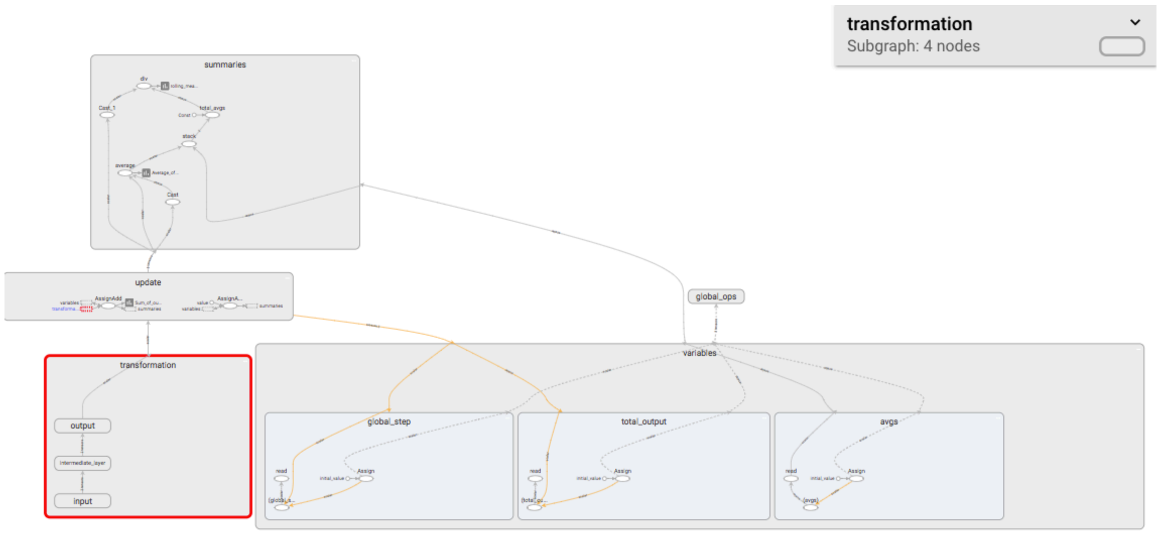
```
        # Separate middle layer
        with tf.name_scope("intermediate_layer"):
            b = tf.reduce_prod(a, name="product_b")
            c = tf.reduce_sum(a, name="sum_c")

        # Separate output layer
        with tf.name_scope("output"):
            output = tf.add(b, c, name="output")

    with tf.name_scope("update"):
        # Increments the total_output Variable by the latest output
        update_total = total_output.assign_add(output)

        # Increments the above `global_step` Variable, should be run whenever
the graph is run
        increment_step = global_step.assign_add(1)

    # Summary Operations
    with tf.name_scope("summaries"):
        avg = tf.div(update_total, tf.cast(increment_step, tf.float32),
name="average")
        avgs = tf.stack([avgs, avg], axis=0)
        total_avgs = tf.reduce_sum(avgs, name="total_avgs")
        rolling_mean = total_avgs / tf.cast(increment_step, tf.float32)

        # Creates summaries for output node
        tf.summary.scalar('Output', output)
        tf.summary.scalar('Sum_of_outputs_over_time', update_total)
        tf.summary.scalar('Average_of_outputs_over_time', avg)
        tf.summary.scalar('rolling_mean_of_outputs_over_time', rolling_mean)
#        tf.summary.scalar('max_number_over_time', update_max)

    # Global Variables and Operations
    with tf.name_scope("global_ops"):
        # Initialization Op
        init = tf.global_variables_initializer()
        # Merge all summaries into one Operation
        merged_summaries = tf.summary.merge_all()
```

```
# Start a Session, using the explicitly created Graph
sess = tf.Session(graph=graph)

# Open a SummaryWriter to save summaries
writer = tf.summary.FileWriter('problem-2', graph)

# Initialize Variables
sess.run(init)

def run_graph(input_tensor):
    """
    Helper function; runs the graph with given input tensor and saves
summaries
    """
    feed_dict = {a: input_tensor}
    out, step, summary, rolling_meany, so_avg = sess.run([output,
increment_step, merged_summaries, rolling_mean, avg],
feed_dict=feed_dict)
    print("avg: {0}, rolling_mean: {1}".format(so_avg, rolling_meany))
    writer.add_summary(summary, global_step=step)

# Run the graph with various inputs
run_graph([2,8])
run_graph([3,1,3,3])
run_graph([8])
run_graph([1,2,3])
run_graph([11,4])
run_graph([4,1])
run_graph([7,3,1])
run_graph([6,3])
run_graph([0,2])
run_graph([4,5,6])

# Write the summaries to disk
writer.flush()

# Close the tf.summary.FileWriter
writer.close()

# Close the session
```

```
sess.close()
```

OUTPUT

```
avg: 26.0, rolling_mean: 26.0
avg: 31.5, rolling_mean: 15.75
avg: 26.3333339691, rolling_mean: 8.77777767181
avg: 22.75, rolling_mean: 5.6875
avg: 30.0, rolling_mean: 6.0
avg: 26.5, rolling_mean: 4.41666650772
avg: 27.2857151031, rolling_mean: 3.89795923233
avg: 27.25, rolling_mean: 3.40625
avg: 24.4444446564, rolling_mean: 2.71604943275
avg: 35.5, rolling_mean: 3.54999995232
```



**Problem 2. Consider the attached file logistic_regression_mnist.py. Search through TensorFlow API documentation and the Internet and describe for us what is the meaning and purpose of functions used in step 5 and step 6.**

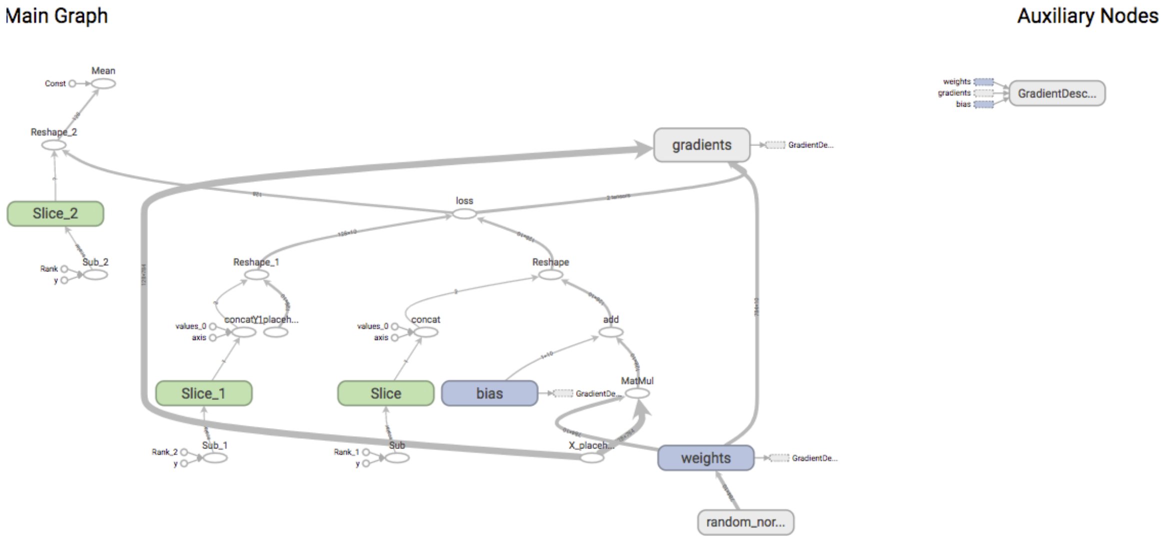
- Step 5:

    - Function computes a softmax that is cross entopy between logits and labels. The function can also be used at the end of a NN for classification.
    - Used as a measure for probability error in discrete classification tasks, and the classes are mutually exclusive.
    - Each image is labeled with only one digit.
    - Cross entropy is used to make the loss function
- Step 6:
    - Function trains data in batches using gradient decent optimizer on loss function.
    - Tests model accuracy

**Demonstrate that you can run the code successfully.**

```
Average loss epoch 0: 1.2855992295
Average loss epoch 1: 0.731785832724
Average loss epoch 2: 0.599681908087
Average loss epoch 3: 0.536343993354
Average loss epoch 4: 0.497743486386
Average loss epoch 5: 0.47123347657
Average loss epoch 6: 0.450513010606
Average loss epoch 7: 0.436465035021
Average loss epoch 8: 0.424002825033
Average loss epoch 9: 0.413197082031
Average loss epoch 10: 0.404840141257
Average loss epoch 11: 0.396882115683
Average loss epoch 12: 0.389814734667
Average loss epoch 13: 0.384264811804
Average loss epoch 14: 0.380483167954
Average loss epoch 15: 0.373798934417
Average loss epoch 16: 0.370727893807
Average loss epoch 17: 0.366061177322
Average loss epoch 18: 0.363209333141
Average loss epoch 19: 0.35967902537
Average loss epoch 20: 0.35831703072
Average loss epoch 21: 0.353276430623
Average loss epoch 22: 0.350656741913
Average loss epoch 23: 0.349562785634
Average loss epoch 24: 0.346703727172
Average loss epoch 25: 0.345034455016
Average loss epoch 26: 0.341071036451
Average loss epoch 27: 0.341392980207
Average loss epoch 28: 0.33764902306
Average loss epoch 29: 0.335006513553
Total time: 12.2428109646 seconds
Optimization Finished!
Accuracy 0.9123
```

**Fetch for us the TensorBoard Graph.**



**Vary parameter `batch_size` through values: 8, 64, 128, 256 and report and plot changes in the execution time and accuracy.**

**Keep other parameters the same as in the original program. Similarly, vary parameter `learning_rate` through values 0.001, 0.005, 0.01, 0.02 and 0.05.**

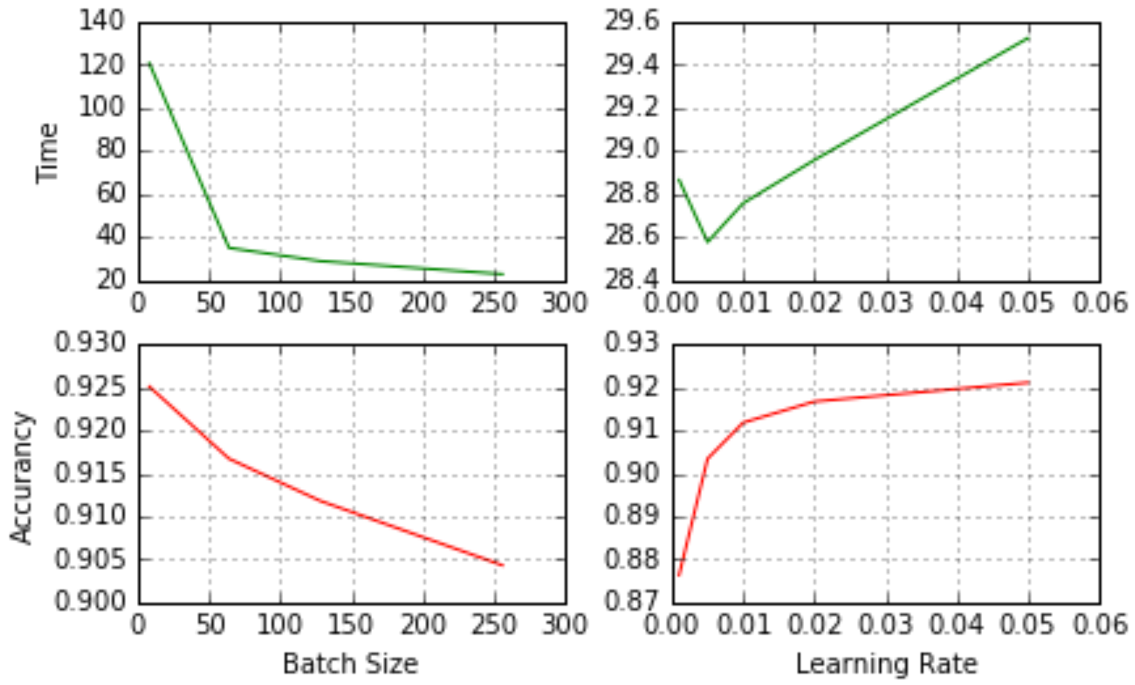
**Report and plot changes in the execution time and accuracy. (25%)**

This part was a little ambiguous for me of what was being asked of me. Typically I would save results and then iterate over results to plot, but I am running out of time.

I saved a sample of times and accuracy results (see below), and plotted it in matplotlib. Judging from other classmates in Piazza this was the best course of action.

```
# input test results
batch_size = [8, 64, 128, 256]
time_1 = [121.864, 34.887, 28.756, 22.691]
```

```
accuracy_1 = [0.9251, 0.9167, 0.9118, 0.9043]

learning_rate = [0.001, 0.005, 0.01, 0.02, 0.05]
time_2 = [28.8654, 28.575, 28.756, 28.957, 29.525]
accuracy_2 = [0.8763, 0.9034, 0.9118, 0.9167, 0.9211]

# plot with various axes scales
plt.figure(1)

# test1
plt.subplot(221)
plt.plot(batch_size, time_1, "g")
plt.ylabel('Time')
plt.grid(True)

# test2
plt.subplot(222)
plt.plot(learning_rate, time_2, "g")
plt.grid(True)

# test3
plt.subplot(223)
plt.plot(batch_size, accuracy_1, "r")
plt.ylabel('Accurancy')
plt.xlabel('Batch Size')
plt.grid(True)

# test4
plt.subplot(224)
plt.plot(learning_rate, accuracy_2, "r")
plt.xlabel('Learning Rate')
plt.grid(True)

plt.subplots_adjust(top=0.92, bottom=0.15, left=0.1, right=0.95, hspace=0.25,
                wspace=0.25)

plt.show()
```



Shown above:

1. Execution time goes down as batch size gets bigger, and goes up as learning rate goes up
2. Model accuracy goes down as batch size gets bigger, and goes up as learning rate also goes up. I

Other Thoughts:
Bigger batch sizes improve training speed at the cost of model accuracy, while learning rate it is vice versa.

**Problem 3. Fetch Iris Dataset from https://archive.ics.uci.edu/ml/datasets/Iris and make attached Python script, `softmax_irises.py` work. You might have to upgrade the script to TF 1.x API. Generate TensorBoard graph of the process and use scalar summary to presenting variation of the loss function during the training process. Report the results of the evaluation process. (35%)**

**See problem-3.ipynb**

**Upgrade Script to TF 1.x**

```
# https://www.tensorflow.org/install/migration
# https://github.com/tensorflow/tensorflow/issues/11217
python2.7 tf_upgrade.py --infile softmax_irises-1.py  --outfile softmax_irises-2.py
```

**Changes after upgrade**

```
def read_csv(batch_size, file_name, record_defaults):
    filename_queue = tf.train.string_input_producer([file_name])
    …
```

**Ran into issues for running script**

```
INFO:tensorflow:Error reported to Coordinator: <class 'tensorflow.py
thon.framework.errors_impl.InvalidArgumentError'>, Expect 5 fields b
ut have 0 in record 0        [[Node: DecodeCSV = DecodeCSV[OUT_TYPE
=[DT_FLOAT, DT_FLOAT, DT_FLOAT, DT_FLOAT, DT_STRING], field_delim=",
", na_value="", use_quote_delim=true, _device="/job:localhost/replic
a:0/task:0/device:CPU:0"](ReaderReadV2:1, DecodeCSV/record_defaults_
0, DecodeCSV/record_defaults_0, DecodeCSV/record_defaults_0, DecodeC
SV/record_defaults_0, DecodeCSV/record_defaults_4)]]
```

## Fix for the "Expect 5 fields but have 0 in record 0" Error

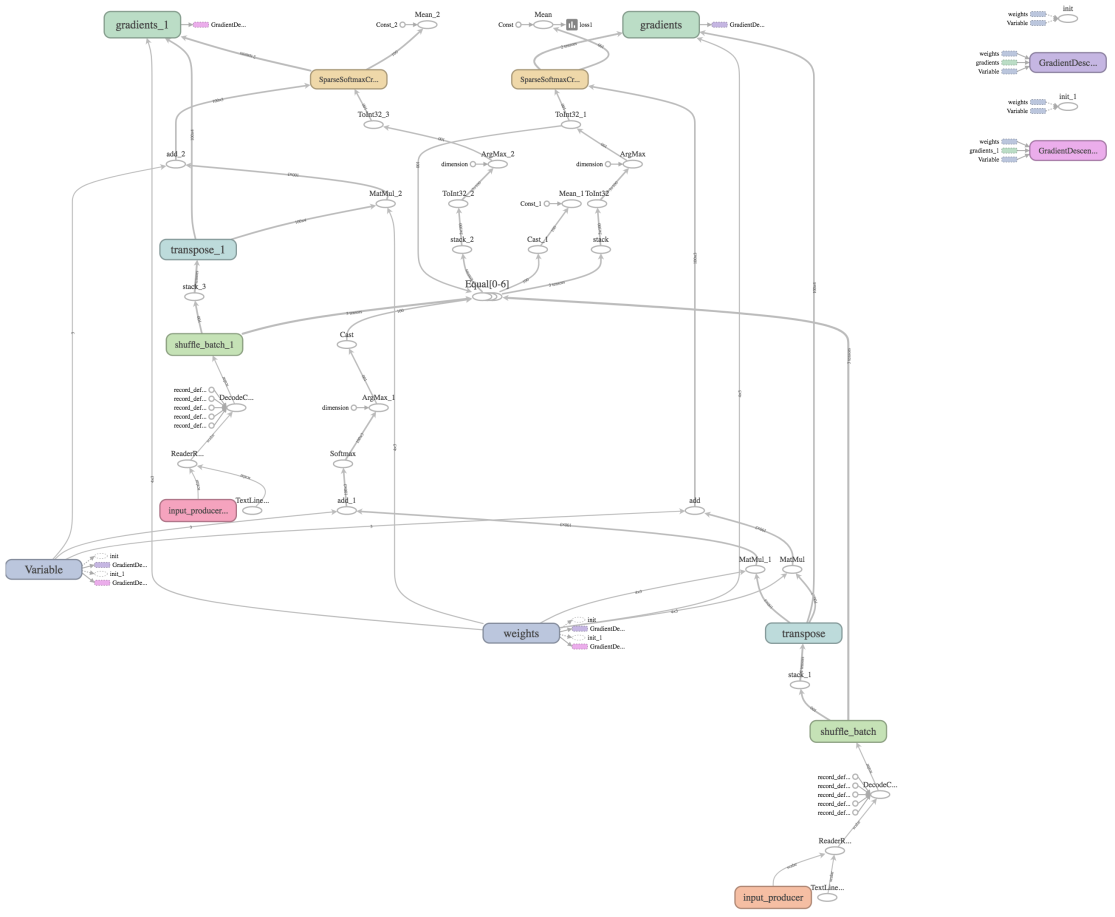
This ended up being extra \n lines in the iris.data file, and not my code.
https://stackoverflow.com/questions/43781143/tensorflow-decode-csv-expect-3-fields-but-have-5-in-record-0-when-given-5-de

```
('Step 0, loss: ', [1.0906187])
('Step 10, loss: ', [1.036504])
('Step 20, loss: ', [0.97362268])
('Step 30, loss: ', [0.93095422])
('Step 40, loss: ', [0.89822221])
('Step 50, loss: ', [0.85084516])
('Step 60, loss: ', [0.82227331])
('Step 70, loss: ', [0.78220087])
('Step 80, loss: ', [0.7724067])
('Step 90, loss: ', [0.73348314])
('Step 100, loss: ', [0.73416811])
('Step 110, loss: ', [0.69817078])
('Step 120, loss: ', [0.66958982])
('Step 130, loss: ', [0.64547896])
('Step 140, loss: ', [0.61703444])
('Step 150, loss: ', [0.59696203])
('Step 160, loss: ', [0.6037131])
('Step 170, loss: ', [0.60522115])
('Step 180, loss: ', [0.58061081])
('Step 190, loss: ', [0.55284351])
('Step 200, loss: ', [0.55683333])
('Step 210, loss: ', [0.5960651])
('Step 220, loss: ', [0.58864295])
…
…
…
('Step 930, loss: ', [0.34087875])
('Step 940, loss: ', [0.31053558])
('Step 950, loss: ', [0.3879489])
('Step 960, loss: ', [0.34680954])
```

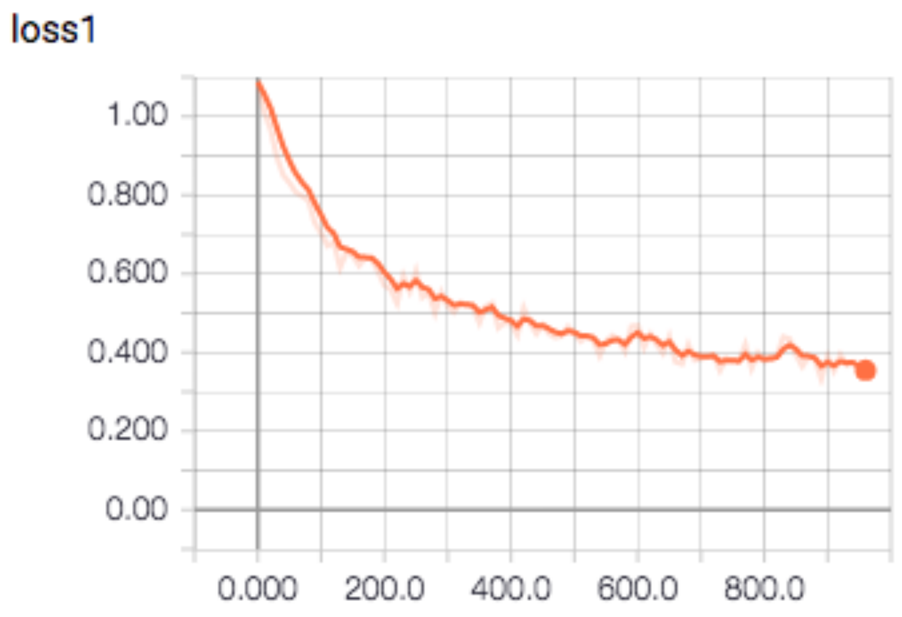
```
('Step 970, loss: ', [0.34077659])
('Step 980, loss: ', [0.35206902])
('Step 990, loss: ', [0.39931166])

0.97
```

**Generate TensorBoard graph of the process and use scalar summary to presenting variation of the loss function during the training process. Report the results of the evaluation process. (35%)**

## Tensorboard Graph



## Loss Function Summary for 1000 iterations



**Problem 4. Analyze all relevant and non-obvious individual steps in the script, `softwmax_irises.py` by examining their inputs and outputs. When convenient, use existing Iris Dataset. When convenient, you are welcome to provide your own inputs. Please examine and describe actions of functions and operations within those functions:**

- `combine_inputs(), line 13`
- `inference(), line 17`
- `read_csv(), line 25`
    - `decode_csv() line 34`
    - `train.shuffle_batch(), line 37`
- `inputs(), line 43`
    - `label_number = tf.to_int32(…), line 49`
    - `features = tf.transpose(..), line 57`
- `evaluate(), line 67`
    - `predicted = tf.cast(tf.arg_max(inference(X), 1)..,` `line 69`
    - `tf.reduce_mean(tf.cast(tf.equal(predicted,Y),.,line 71`
- `threads = tf.train.start_queue_runners(sess=sess,` `coord=coord).., line 85`

**Please describe the effect of every function or command by providing an illustrative input and output set of values and well as a brief narrative. Please rely on TensorFlow API as much as possible. (%25)**

1. Script reads data from iris.data file
2. Explanatory variables map to first 4 columns [sepal_length, sepal_width, petal_length, petal_width]
3. Response or label variable is type of iris is last column of file
4. Data needs to be converted to a dtype suitable for regression model
5. Inputs and read_csv function do the following tasks:
    a. Read file
    b. X is a 1 x 100 matrix with explanatory variables
    c. Y is a 1 X 100 matrix with explanatory variables
    d. Iris file is read into read_csv function
        i. Function creates four 100 X 1 tensors with the above explanatory variables
        ii. Function reads another 100 X 1 dependent variable
        iii. Value of 100 defined by batch_size parameter.
    e. Initial dependent variable tensor has text descriptions for three species of irises, and is converted into a 0 1 2 valued tensor

X tensor (3 first rows):

```
[ 5.4000001   3.70000005  1.5         0.2       ]
 [ 4.5999999   3.20000005  1.39999998  0.2       ]
 [ 5.4000001   3.          4.5         1.5       ]
```

Dependent variable, corresponds to label tensor

```
['Iris-setosa' 'Iris-setosa' 'Iris-versicolor' 'Iris-setosa' 'Iris
-setosa'
…
…
…
 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris
-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor' 'Iris
-setosa']
```

Label tensor is converted to numeric values 0, 1, 2 for each species:

```
[0 0 1 0 0 0 0 1 0 0 1 1 0 1 1 1 1 2 0 0 1 0 2 2 0 0 1 2 1 1 0 0 1
0 2 1 1
 1 2 2 1 2 2 0 0 0 2 0 1 0 1 0 2 0 2 2 1 0 2 1 0 2 2 2 0 2 2 0 0 0
1 0 0 0
 1 2 2 2 2 1 1 0 2 2 1 0 2 2 1 0 1 0 0 0 0 0 0 0 1 0]
```

1. decode_csv()
    a. Converts the text line read from the file into a tensor

b. Thetensor are put together in a tensor/matrix
c. This represents explanatory variables by **train.shuffle_batch()** function.
d. Tensor is then transposed by using **transpose()** function
e. Obtains the final 100 x 4 explanatory variables tensor

2. Input()
   a. label_number
      i. Dependent variable is converted to numeric representation from 0 to 2 representing the 3 possible different species.
      ii. This is the final Y tensor listed above

Model Process

- Uses softmax regression
  - Softmax regression uses tf.matmul(X, W) + b to model
  - A expression is created by using combine_inputs ()
    - This function allows work with a 2D tensor with multiple inputs

Loss

- Is determined through the cross entropy function
- Uses the sparse_softmax_cross_entropy_with_logits function
  - The purpose is to minimize this value
  - Optimizes the values of our regression with GradientDescentOptimizer**.**
  - We train the model 1,000 times

To determine model success the evaluate() function is used

Please, describe every step of your work and present all intermediate and final results in a Word document. Please, copy past text version of all essential command and snippets of results into the Word document with explanations of the purpose of those commands. We cannot retype text that is in JPG images. Please, always submit a separate copy of the original, working scripts and/or class files you used. Sometimes we need to run your code and retyping is too costly. Please include in your MS Word document only relevant portions of the console output or output files. Sometime either console output or the result file is too long and including it into the MS Word document makes that document too hard to read. PLEASE DO NOT EMBED files into your MS Word document. For issues and comments visit the class Discussion Board.