## HU Extension Assignment 10 E63 Big Data Analytics

### Handed out: 04/08/2017 Due by 9:30AM EST on Saturday, 04/15/2017

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

All my comments and code are in blue color

**Problem 1.** Please use tf\_upgrade.py utility, which you could find on the TensorFlow GitHub site to upgrade attached Python script vectorized\_graph.py to TensorFlow 1.x. Demonstrate that upgraded script will run and produce TensorBoard graph and summaries. Provide working upgraded script and images of your graphs and calculated summaries. **(25%)**

Upgrade the script:

|  |
| --- |
| sudo python tf\_upgrade.py --infile vectorized\_graph.py --outfile vectorized\_graph\_updated.py |

After upgrading the script by using upgrade.py utility, the script remains throwing the following error:

|  |
| --- |
| TypeError Traceback (most recent call last)  <ipython-input-1-5a6453c5eccf> in <module>()  38 # Summary Operations  39 with tf.name\_scope("summaries"):  ---> 40 tf.summary.scalar(b'output', output, name="output\_summary") # Creates summary for output node  41 tf.summary.scalar(b'product of inputs', b, name="prod\_summary")  42 tf.summary.scalar(b'sum of inputs', c, name="sum\_summary")  TypeError: scalar() got multiple values for keyword argument 'name' |

I modified the script according to what in TensorFlow API:

|  |
| --- |
| tf.summary.scalar(tensor= output, name="output\_summary") # Creates summary for output node  tf.summary.scalar(tensor = b, name="prod\_summary")  tf.summary.scalar(tensor = c, name="sum\_summary") |

Now the script works without error and produces the below graph:

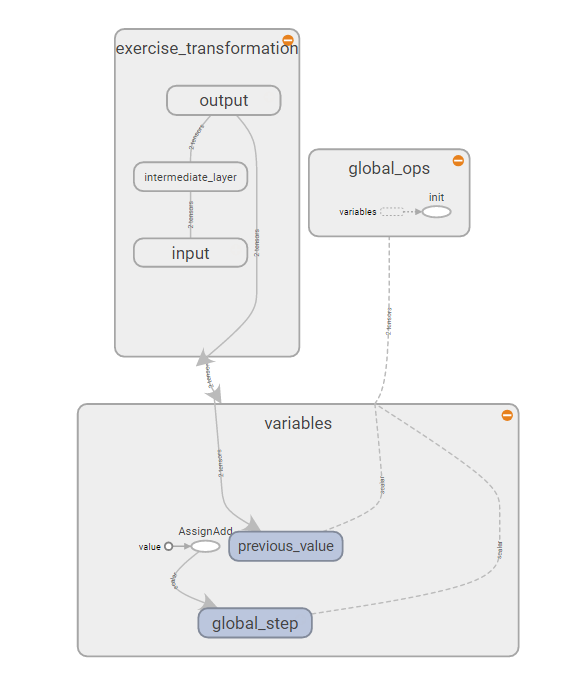


Figure 1 - Graph result of the script execution

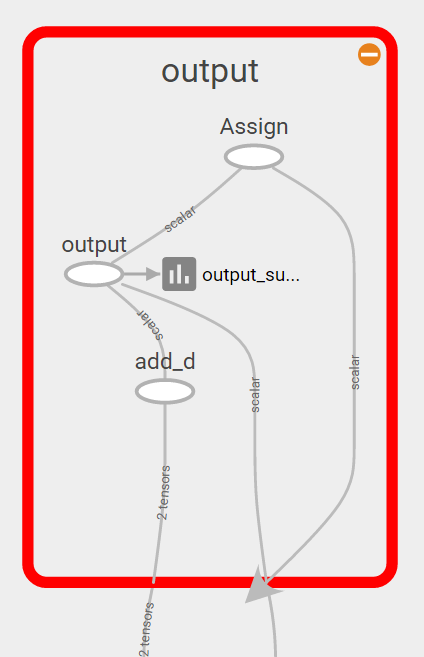


Figure 2 - Detail of the previous graph

And this scalars:

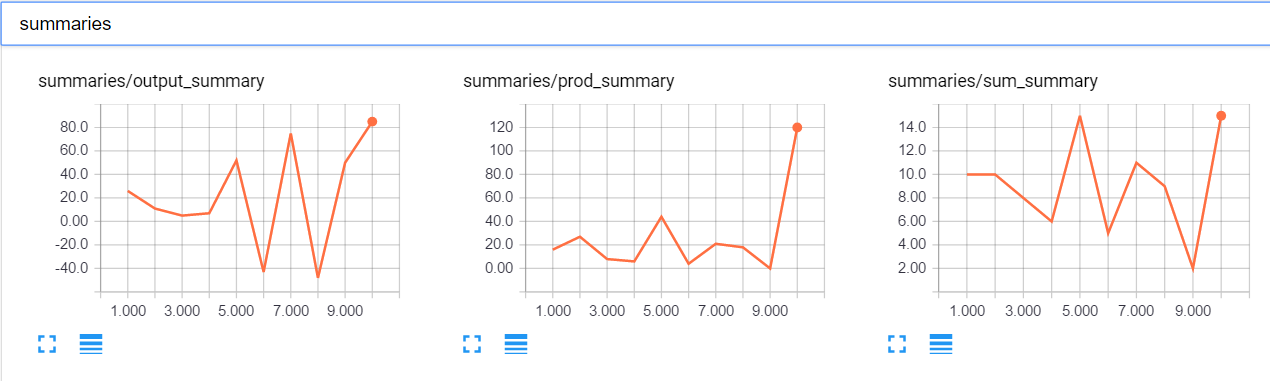


Figure 3 - Summary scalar with the requested results

And summary results:

SUM:

|  |  |
| --- | --- |
| Step | Value |
| 1 | 10 |
| 2 | 10 |
| 3 | 8 |
| 4 | 6 |
| 5 | 15 |
| 6 | 5 |
| 7 | 11 |
| 8 | 9 |
| 9 | 2 |
| 10 | 15 |

PRODUCT:

|  |  |
| --- | --- |
| Step | Value |
| 1 | 16 |
| 2 | 27 |
| 3 | 8 |
| 4 | 6 |
| 5 | 44 |
| 6 | 4 |
| 7 | 21 |
| 8 | 18 |
| 9 | 0 |
| 10 | 120 |

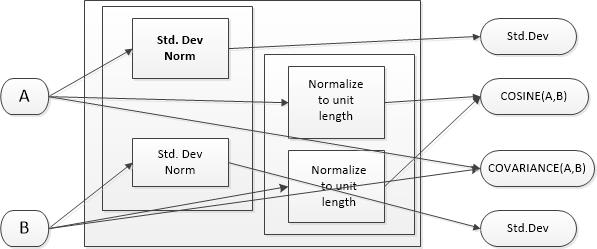
OUTPUT:

|  |  |
| --- | --- |
| Step | Value |
| 1 | 26 |
| 2 | 11 |
| 3 | 5 |
| 4 | 7 |
| 5 | 52 |
| 6 | -43 |
| 7 | 75 |
| 8 | -48 |
| 9 | 50 |
| 10 | 85 |

This is the complete working 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")    # Increments the above `global\_step` Variable, should be run whenever the graph is run  increment\_step = global\_step.assign\_add(1)    # Variable that keeps track of previous output value:  previous\_value = tf.Variable(0.0, dtype=tf.float32, name="previous\_value")    # Primary transformation Operations  with tf.name\_scope("exercise\_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"):  d = tf.add(b, c, name="add\_d")  output = tf.subtract(d, previous\_value, name="output")  update\_prev = previous\_value.assign(output)    # Summary Operations  with tf.name\_scope("summaries"):  tf.summary.scalar(tensor= output, name="output\_summary") # Creates summary for output node  tf.summary.scalar(tensor = b, name="prod\_summary")  tf.summary.scalar(tensor = c, name="sum\_summary")    # Global Variables and Operations  with tf.name\_scope("global\_ops"):  # Initialization Op  init = tf.global\_variables\_initializer()  # Collect all summary Ops in graph  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('./improved\_graph', 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}  output, summary, step = sess.run([update\_prev, merged\_summaries, increment\_step], feed\_dict=feed\_dict)  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])  # Writes the summaries to disk  writer.flush()  # Flushes the summaries to disk and closes the SummaryWriter  writer.close()  # Close the session  sess.close()  # To start TensorBoard after running this file, execute the following command:  # $ tensorboard --logdir='./improved\_graph' |

**Problem 2.** Please construct a graph that will accept as inputs two vectors of equal length (tensors of dimension 1) and perform the operations on those vectors as depicted in the drawing bellow. Organize your variables and operations in nested namespaces as suggested by the nested boxes in the same graph. Organize your program in such a way that it repeats calculations in the graphs for 8 vectors of different lengths and element values. Collect and display in the TensorBoard as summaries the results on the right. **(25%)**



According to instructions this is the python code to generate the requested graph.

Code included comments about how calculations are done, but in brief:

* Standard deviation is the square root of the variance. TensorFlow provides variance.
* Norm of a vector is its Euclidian length. TensorFlow provides it.
* Normalization to unit length, is to convert a vector to a length of 1 preserving its direction. So dividing it by its norm
* Cosine similarity measures the dregree of alignment of 2 vector, obtained by calculating the dot product of its unit vector representations. Same direction 1, opposite -1, orthogonal 0
* Covariance, measure the level of linear correlation between the 2 vectors. The formula is defines by the result of adding the multiplication of the difference of each dimension of the vectors and its average, divided by the number of dimensions.

All the previous operations are implemented according to TensorFlow’s API, resulting in the following Python code:

|  |
| --- |
| import tensorflow as tf  # Explicitly create a Graph object  graph = tf.Graph()  with graph.as\_default():    # Variable to keep track of how many times the graph has been run  global\_step = tf.Variable(0, dtype=tf.int32, name="global\_step")    # Increments the above `global\_step` Variable, should be run whenever the graph is run  increment\_step = global\_step.assign\_add(1)    # 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")  b = tf.placeholder(tf.float32, shape=[None], name="input\_placeholder\_b")    # Operations  with tf.name\_scope("operations"):    # Standard Deviation Operations and Norm  #for SD tensorflow returns variance, and SD is the square root of variance  #a norm of a vector is its euclidian lenght  with tf.name\_scope("std\_deviation\_norm"):    with tf.name\_scope("std\_deviation\_norm\_a"):  \_,var\_a = tf.nn.moments(a, axes=[0], name="var\_a")  std\_dev\_a = tf.sqrt(var\_a, name="std\_dev\_a")  norm\_a = tf.norm(a, name = "norm\_a")    with tf.name\_scope("std\_deviation\_norm\_b"):  \_,var\_b = tf.nn.moments(b, axes=[0], name="var\_b")  std\_dev\_b = tf.sqrt(var\_b, name="std\_dev\_b")  norm\_b = tf.norm(b, name = "norm\_b")    # Normalization to unit lenght  # a vector is normalized to unit lenght dividing it by its norm, by doing that we have  # a vector with exact same direction, but with lenght 1  with tf.name\_scope("normalize\_to\_unit\_length"):    with tf.name\_scope("normalize\_to\_unit\_length\_a"):  norm\_unit\_a = tf.div(a, norm\_a, "norm\_unit\_a")      with tf.name\_scope("normalize\_to\_unit\_length\_b"):  norm\_unit\_b = tf.div(b, norm\_b, "norm\_unit\_b")    #cosine similarity of 2 vectors of unit lenght is  #the dot product a x b reduced to a scalar by adding result vector's dimensions  num\_cosine = tf.multiply(norm\_unit\_a, norm\_unit\_b)  cosine\_a\_b = tf.reduce\_sum(tf.multiply(norm\_unit\_a, norm\_unit\_b), name = "cosine\_a\_b")    #covariance is sum((x-mean x) x (y - mean y))/n  covariance\_a\_b = tf.reduce\_sum(tf.div((tf.multiply (tf.subtract (a, tf.reduce\_mean(a)), tf.subtract (b, tf.reduce\_mean(b)))),  tf.cast(tf.shape(a), dtype = tf.float32 )))      with tf.name\_scope("summaries"):  tf.summary.scalar(tensor = std\_dev\_a, name="std\_dev\_a\_summary")  tf.summary.scalar(tensor = std\_dev\_b, name="std\_dev\_b\_summary")  tf.summary.scalar(tensor = cosine\_a\_b, name="cosine\_a\_b\_summary")  tf.summary.scalar(tensor = covariance\_a\_b, name="covariance\_a\_b\_summary")      # Global Variables and Operations  with tf.name\_scope("global\_ops"):  # Initialization Op  init = tf.global\_variables\_initializer()  # Collect all summary Ops in graph  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('./problem2\_graph', graph)  # Initialize Variables  sess.run(init)  def run\_graph(input\_tensor\_a, input\_tensor\_b):  """  Helper function; runs the graph with given input tensor and saves summaries  """  feed\_dict = {a: input\_tensor\_a, b: input\_tensor\_b}  summary, step, cos, cov = sess.run([merged\_summaries, increment\_step, cosine\_a\_b, covariance\_a\_b], feed\_dict=feed\_dict)  writer.add\_summary(summary, global\_step=step)    print "Cosine similarity:", cos  print "Covariance:", cov  # Run the graph with various 8 pairs of 2 vectors  #cosine zero  run\_graph([32,12],[-6,16])  #cosine 1  run\_graph([32,12,33],[16, 6, 16.5])  #random vectors  run\_graph([7, 14, 3, 2, -7],[16, 6, 16.5, 14, -18])  run\_graph([1, -3, 12, 4, -77, -1000],[1, -16, 9, 1444, -1844, 5])  run\_graph([1, -1, 3, 44, -77,-120, 121],[22, -44, 94, 144, -14, 4, 5])  run\_graph([7, 3, 2, -7],[16, 6, 13.5, 14])  run\_graph([7, 14, -7],[-35, -6, 16])  run\_graph([7, 22],[16555, 67666])  # Writes the summaries to disk  writer.flush()  # Flushes the summaries to disk and closes the SummaryWriter  writer.close()  # Close the session  sess.close() |

And below is the graph organized as per the above scope definitions:

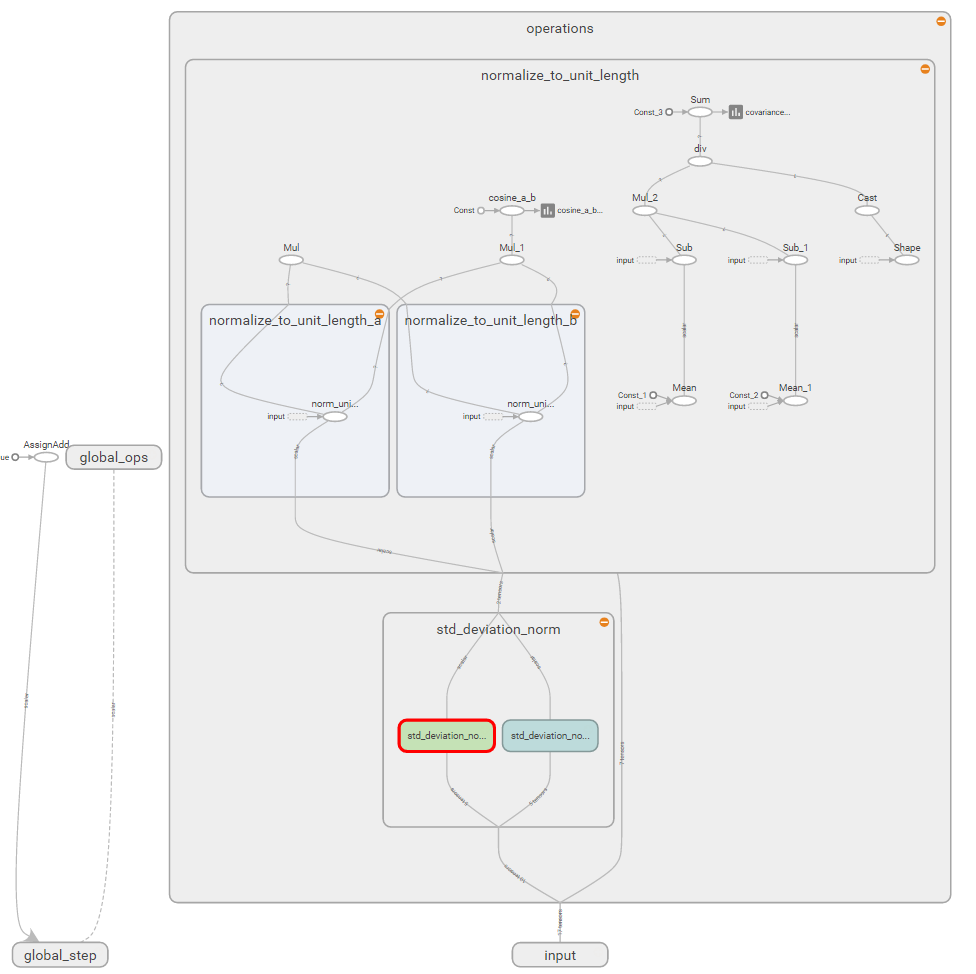


Figure 4 - TensorBoard graph according to the provided image

And the corresponding summary scalars:

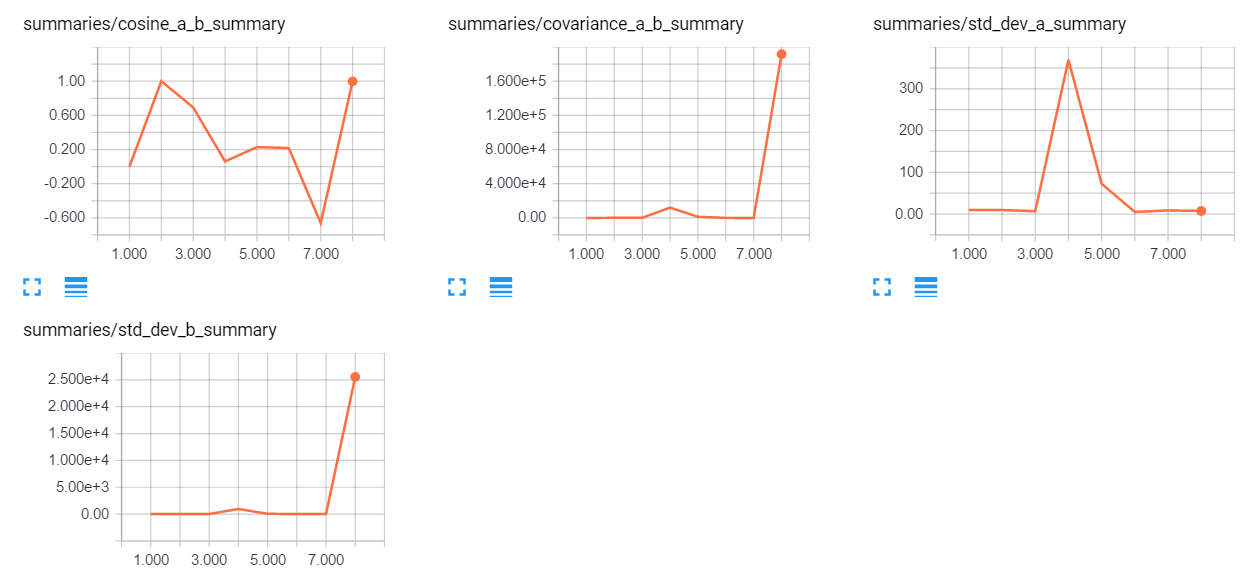


Figure 5 - Summary scalar detail in TensorBoard

**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.

Script is upgraded to work by typing:

|  |
| --- |
| sudo python tf\_upgrade.py --infile softmax\_irises.py --outfile softmax\_irises\_updated.py |

Then, I copy the updated script code to a Jupyter notebook, and modified one line of code to solve an issue related to the iris dataset path:

|  |
| --- |
| filename\_queue = tf.train.string\_input\_producer([file\_name]) |

Finally I run the script and get the following result:

|  |
| --- |
| loss: [1.0896934]  loss: [1.0061879]  loss: [0.97894365]  loss: [0.9068445]  loss: [0.85181707]  loss: [0.83601087]  loss: [0.80641389]  loss: [0.76214647]  loss: [0.78893083]  loss: [0.70025045]  loss: [0.68283963]  loss: [0.74291307]  loss: [0.6730445]  loss: [0.67512876]  loss: [0.61769885]  loss: [0.60832554]  loss: [0.65671563]  loss: [0.59194702]  loss: [0.61474001]  loss: [0.61037767]  loss: [0.55162543]  loss: [0.60921144]  loss: [0.54208702]  loss: [0.56770253]  loss: [0.59888864]  loss: [0.50387788]  loss: [0.55702192]  loss: [0.52349859]  loss: [0.52094871]  loss: [0.55458236]  loss: [0.49814498]  loss: [0.49707109]  loss: [0.49208054]  loss: [0.46489203]  loss: [0.51364684]  loss: [0.49815392]  loss: [0.45370093]  loss: [0.50520921]  loss: [0.45822796]  loss: [0.46332872]  loss: [0.52214724]  loss: [0.4447538]  loss: [0.48600951]  loss: [0.44634551]  loss: [0.45368484]  loss: [0.46611488]  loss: [0.44179681]  loss: [0.46290612]  loss: [0.45916215]  loss: [0.41356313]  loss: [0.46329907]  loss: [0.45766294]  loss: [0.45202416]  loss: [0.44644538]  loss: [0.43983734]  loss: [0.46171224]  loss: [0.42937583]  loss: [0.40804115]  loss: [0.45691359]  loss: [0.36660129]  loss: [0.41976902]  loss: [0.44575694]  loss: [0.4221881]  loss: [0.45848542]  loss: [0.35023916]  loss: [0.40067983]  loss: [0.43158707]  loss: [0.40783203]  loss: [0.429975]  loss: [0.41308051]  loss: [0.37447643]  loss: [0.43695316]  loss: [0.3874836]  loss: [0.41202065]  loss: [0.38481766]  loss: [0.36227006]  loss: [0.42550334]  loss: [0.36900973]  loss: [0.35596657]  loss: [0.4264766]  loss: [0.39321449]  loss: [0.4216916]  loss: [0.39022842]  loss: [0.38414264]  loss: [0.38933221]  loss: [0.36237854]  loss: [0.3700974]  loss: [0.38558677]  loss: [0.32251602]  loss: [0.40302742]  loss: [0.36852616]  loss: [0.39615235]  loss: [0.39633191]  loss: [0.35642219]  loss: [0.34791866]  loss: [0.38070679]  loss: [0.3480576]  loss: [0.40163025]  loss: [0.35717356]  loss: [0.35555917]  1.0 |

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. **(25%)**

I have modified the script to include the scalar that shows loss function variation along the training process. Following to images present the graph and the scalar.

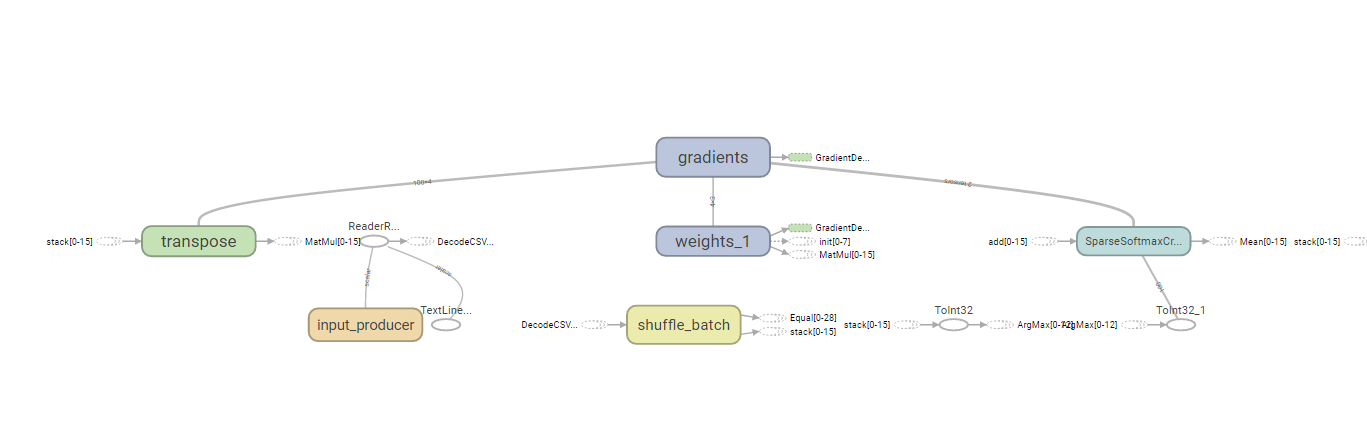


Figure 6 - Detail of the graph generated

And the result of the loss function summary presented from TensorBoard, where is presented in a graphical way what already obtained through the print sentence included in the original python code:

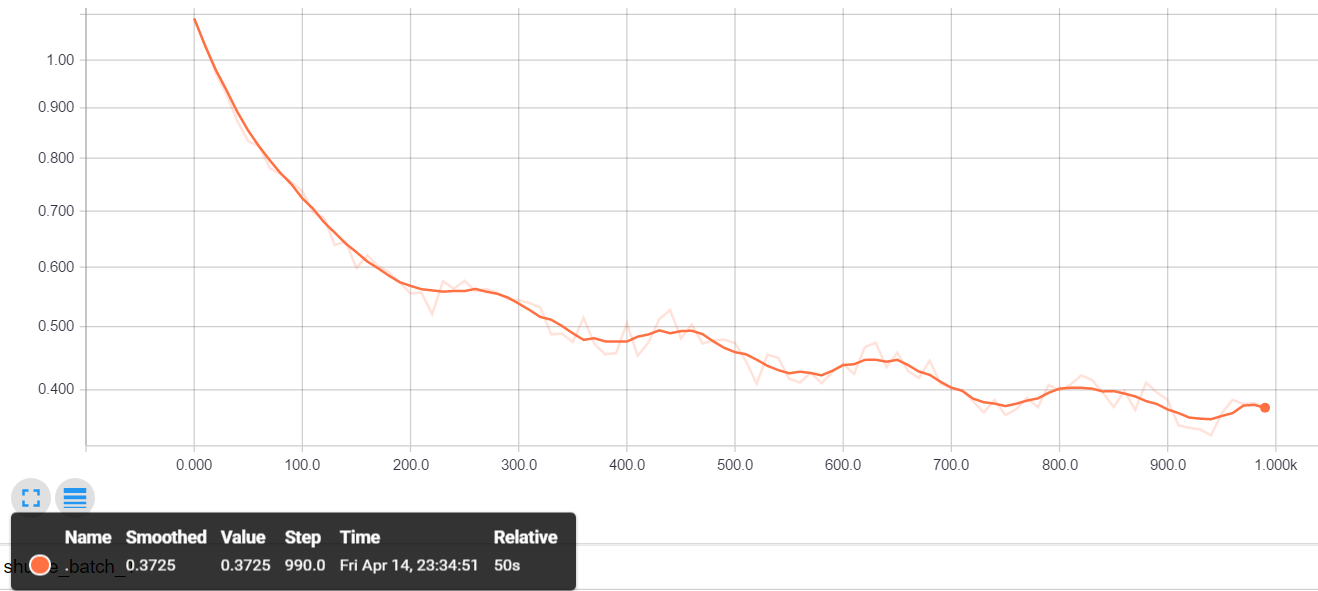
****

Figure 7 - 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

First thing done by the script is to read data from iris.data CSV file. In this file explanatory variables correspond to the 4 first column sepal\_length, sepal\_width, petal\_length, petal\_width, and response or label variable is the type of iris that is the last column of the file. This data needs to be converted in a format suitable for the regression model.

inputs and read\_csv functions take care of these tasks:

Once the file is read, we have X as a 1 x 100 matrix with the explanatory variables. Y is a 1 x 100 matrix with the response variable values. IRIS file is read through the read\_csv function that creates four 100 x 1 tensors with the explanatory variables reads and another 100 x 1 with the dependent variable. Value of 100 is defined by batch\_size parameter. Initial dependent variable tensor contains text descriptions for the 3 species of irises, and is converted into a 0 1 2 valued tensor that is what actually corresponds to Y tensor.

Detail of 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 as read from the file, corresponds to label tensor

|  |
| --- |
| ['Iris-setosa' 'Iris-setosa' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'  'Iris-setosa' 'Iris-setosa' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'  'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-versicolor'  'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica'  'Iris-setosa' 'Iris-setosa' 'Iris-versicolor' 'Iris-setosa'  'Iris-virginica' 'Iris-virginica' 'Iris-setosa' 'Iris-setosa'  'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor'  'Iris-setosa' 'Iris-setosa' 'Iris-versicolor' 'Iris-setosa'  'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'  'Iris-virginica' 'Iris-virginica' 'Iris-versicolor' 'Iris-virginica'  'Iris-virginica' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'  'Iris-virginica' 'Iris-setosa' 'Iris-versicolor' 'Iris-setosa'  'Iris-versicolor' 'Iris-setosa' 'Iris-virginica' 'Iris-setosa'  'Iris-virginica' 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa'  'Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica'  'Iris-virginica' 'Iris-virginica' 'Iris-setosa' 'Iris-virginica'  'Iris-virginica' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'  'Iris-versicolor' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'  'Iris-versicolor' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'  'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'  'Iris-virginica' 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa'  'Iris-virginica' 'Iris-virginica' 'Iris-versicolor' '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 specie and are the actual values that compound Y tensor:

|  |
| --- |
| [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] |

Within read\_csv function, **decode\_csv()** will convert the text line read from the file into a tensor, those tensor are put together in a tensor/matrix representing explanatory variables by **train.shuffle\_batch()** function. Afterwards this tensor is transposed by using **transpose()** function ton obtain the final 100 x 4 explanatory variables tensor.

With **label\_number,** dependent variable is converted to a numeric representation from 0 to 2 representing the 3 possible different species. This is the final Y tensor listed above.

Then we start the modelling process. We implement our model by using **softmax** **regression**. Softmax regression is modelled in the form of tf.matmul(X, W) + b. We construct this expression by using **combine\_inputs ()** function that allows us to deal with a 2D tensor with multiple inputs, like is the case of our X tensor

Loss defines the goodness or our model, so through training we try to make its value as low as possible. Loss is determined through **the cross entropy** function, by means of **sparse\_softmax\_cross\_entropy\_with\_logits** tensorflow's function. The purpose is to minimize this value. To achieve it we optimize the values of our regression by means of the **GradientDescentOptimizer.** We train the model 5,000 times

Finally, once trained the model, we want to determine how good our model is. For it, we have the **evaluate ()** function, that gives a result of the accuracy of the model after the 5,000 training steps. Evaluate() give us the number of predictions that are actually correct and with that provide us with us actual model’s accuracy in percentage, that means the number of actual dependent variable values that were correctly predicted by the model.

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)**

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.