



UNIVERSITY OF MISSOURI-KANSAS CITY

Python and Deep Learning

Lab 3 Assignment

Team Members:

Pragathi Thammaneni

Sridevi Mallipudi

Introduction:

The key objective of this assignment is to focus on tensor flow deep learning library.

Objectives:

To code for the 2 questions by implementing the below concepts.

- Installed required packages
- Fitting into data frame
- Labels extraction
- Divided into training and testing
- One hot coding
- Implemented logistic regression
- Optimizer to minimize the loss
- Calculated accuracy
- Implemented the word embeddings

Approaches /Methods:

Using Python 3.6, PyCharm (Community edition)

Workflow &Datasets/Parameters and Evaluation:

The below each question will follow different approaches to solve. Coding is done to perform the evaluation of each individual snippet to execute the datasets which are provided as the input parameters.

Question 1:

1. Implement the Logistic Regression with new data set which is not used in class
Show the graph in TensorBoard
Change the hyperparameter and compare the result

Solution:

This snippet code provides the above implementation which for the logistic regression, have considered iris data and used pandas to set into the data frame and installed sklearn package to load the iris data. After the data framing I have implemented logistic regression and calculated loss function to analyze the accuracy. In this data is divided into training and testing set such as 80% of training data and 20% of testing data. At last tensor flow session is created for the logistic regression model and calculated the accuracy

Data Set: Iris Data set

Iris data set is used to implement the logistic regression model. This dataset has three kinds of colors for petal and sepal length to store in n dimensional array.

Code Snippet:

The below is the code for executing the above workflow

```
103 lines (96 sloc) | 4.02 KB
Raw Blame History
1 import numpy as np
2 import seaborn as sns
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 import tensorflow as tf
6
7 iris = pd.read_csv('dataset.csv')
8 iris.Species = iris.Species.replace(to_replace=['Iris-setosa', 'Iris-versicolor'], value=[0, 1])
9
10 X = iris.drop(labels=['Id', 'Species'], axis=1).values
11 y = iris.Species.values
12
13 # set seed for numpy and tensorflow
14 # set for reproducible results
15 seed = 5
16 np.random.seed(seed)
17 tf.set_random_seed(seed)
18 # dataset segmentation
19 # splitting the dataset as train data and test data
20 train_index = np.random.choice(len(X), round(len(X) * 0.8), replace=False)
21 test_index = np.array(list(set(range(len(X))) - set(train_index)))
22 train_X = X[train_index]
23 train_y = y[train_index]
24 test_X = X[test_index]
25 test_y = y[test_index]
26 # Define the normalized function
27 def min_max_normalized(data):
28     col_max = np.max(data, axis=0)
29     col_min = np.min(data, axis=0)
30     return np.divide(data - col_min, col_max - col_min)
31 # Normalized processing, must be placed after the data set segmentation,
32 # otherwise the test set will be affected by the training set
33 train_X = min_max_normalized(train_X)
34 test_X = min_max_normalized(test_X)
35
36 # Declare the variables that need to be learned and initialization
```

Different optimizer has been implemented

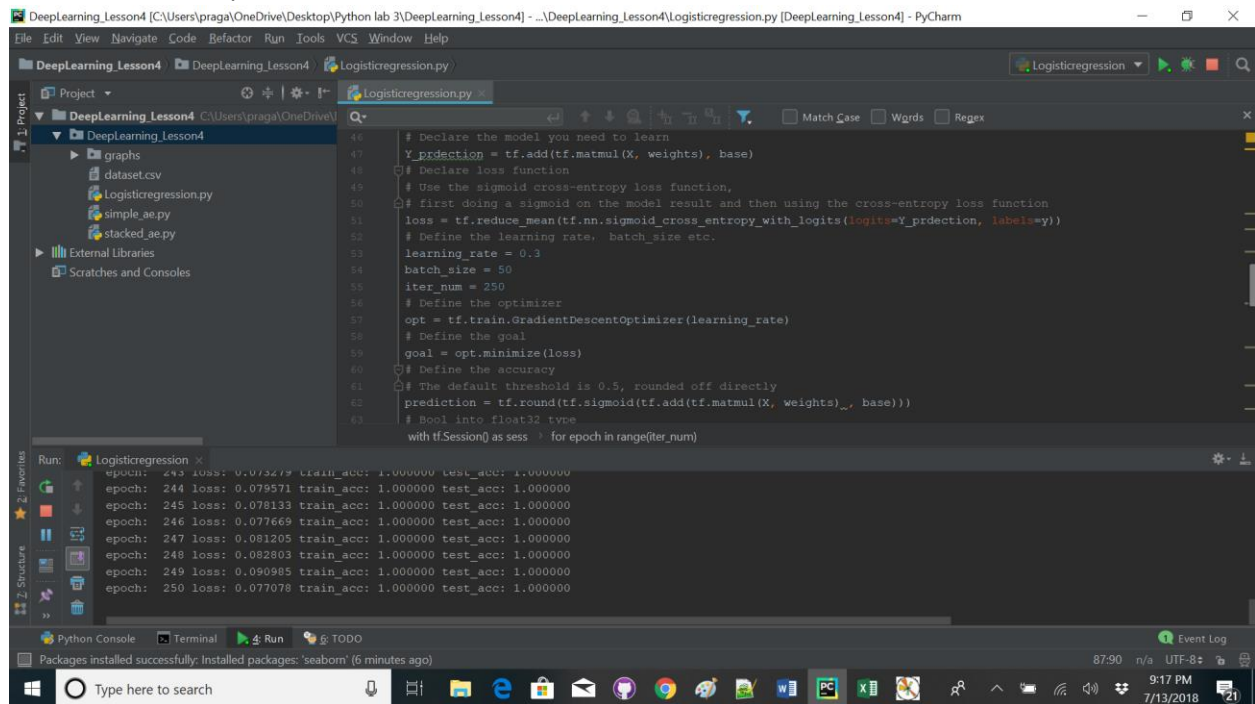
GradientDescentOptimizer, RMSPropOptimizer, AdamOptimizer, AdagradOptimizer with different learning rates. And the loss rate, accuracy and tensor flow is generated for the shown below outputs.

Output Screenshot:

GradientDescentOptimizer

,Tensor

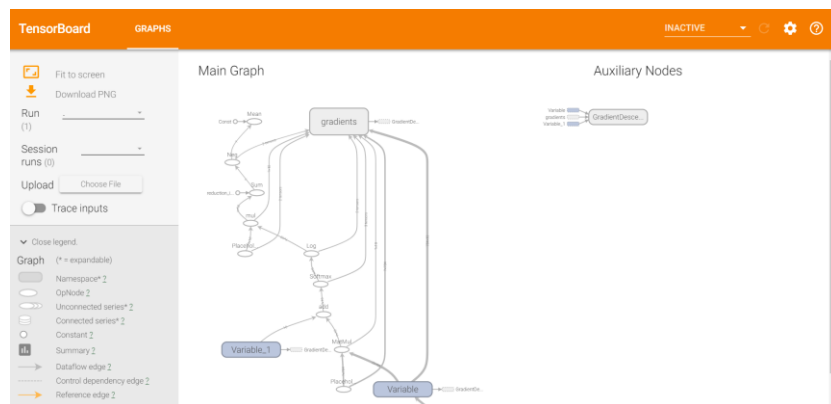
flow:



```
46 # Declare the model you need to learn
47 Y_prediction = tf.add(tf.matmul(X, weights), base)
48 # Declare loss function
49 # Use the sigmoid cross-entropy loss function,
50 # first doing a sigmoid on the model result and then using the cross-entropy loss function
51 loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=Y_prediction, labels=y))
52 # Define the learning rate, batch_size etc.
53 learning_rate = 0.3
54 batch_size = 50
55 iter_num = 250
56 # Define the optimizer
57 opt = tf.train.GradientDescentOptimizer(learning_rate)
58 # Define the goal
59 goal = opt.minimize(loss)
60 # Define the accuracy
61 # The default threshold is 0.5, rounded off directly
62 prediction = tf.round(tf.sigmoid(tf.add(tf.matmul(X, weights), base)))
63 # Bool into float32 type
64 with tf.Session() as sess:
65     for epoch in range(iter_num):
```

Run: LogisticRegression

epoch	loss	train_acc	test_acc
243	0.073279	1.000000	1.000000
244	0.079571	1.000000	1.000000
245	0.078133	1.000000	1.000000
246	0.077669	1.000000	1.000000
247	0.081205	1.000000	1.000000
248	0.082803	1.000000	1.000000
249	0.090985	1.000000	1.000000
250	0.077078	1.000000	1.000000



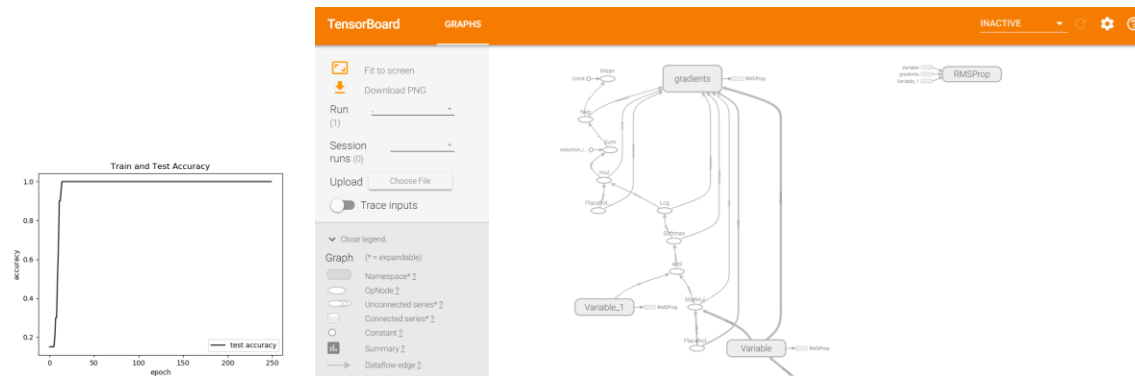
RMSPropOptimizer, Tensor flow:

```
46 # Declare the model you need to learn
47 Y_prediction = tf.add(tf.matmul(X, weights), base)
48 # Declare loss function
49 # Use the sigmoid cross-entropy loss function,
50 # first doing a sigmoid on the model result and then using the cross-entropy loss function
51 loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=Y_prediction, labels=y))
52 # Define the learning rate, batch_size etc.
53 learning_rate = 0.3
54 batch_size = 50
55 iter_num = 250
56 # Define the optimizer
57 opt = tf.train.RMSPropOptimizer(learning_rate)
58 # Define the goal
59 goal = opt.minimize(loss)
60 # Define the accuracy
61 # The default threshold is 0.5, rounded off directly
62 prediction = tf.round(tf.sigmoid(tf.add(tf.matmul(X, weights), base)))
63 # Bool into float32 type
```

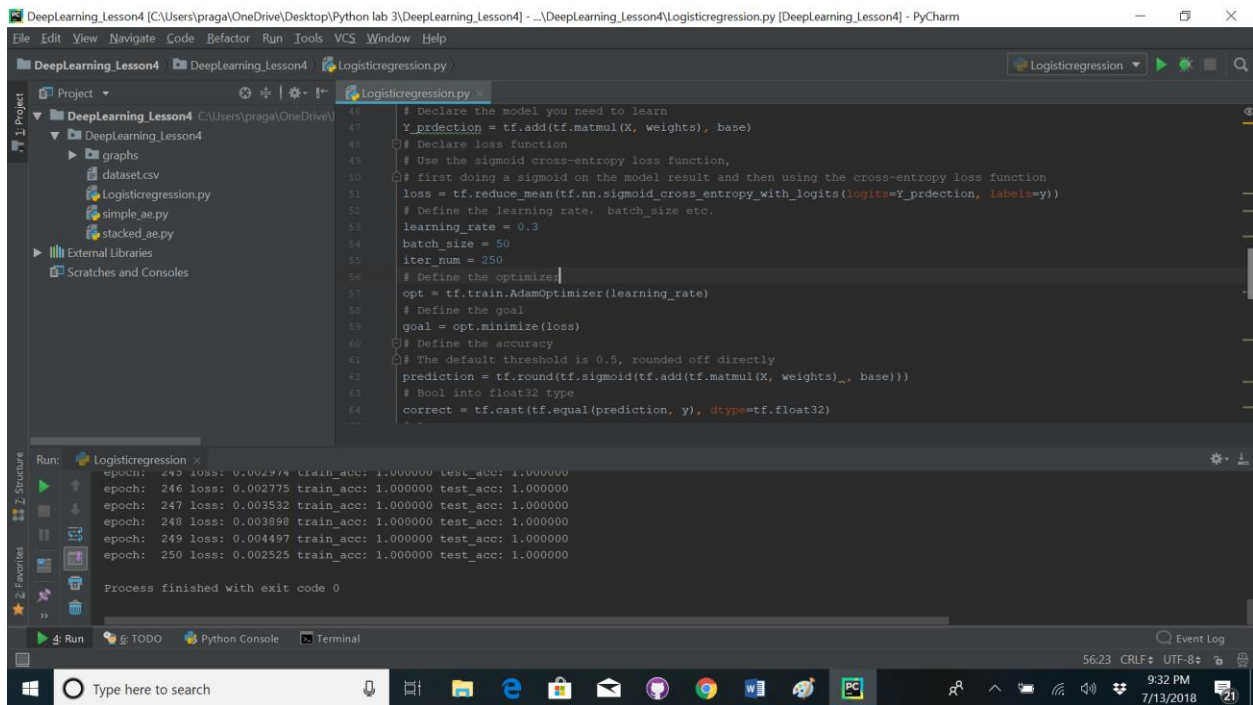
Run: LogisticRegression x LogisticRegression x LogisticRegression x

epoch: 243 loss: 0.000018 train_acc: 1.000000 test_acc: 1.000000
epoch: 244 loss: 0.000013 train_acc: 1.000000 test_acc: 1.000000
epoch: 245 loss: 0.000021 train_acc: 1.000000 test_acc: 1.000000
epoch: 246 loss: 0.000015 train_acc: 1.000000 test_acc: 1.000000
epoch: 247 loss: 0.000015 train_acc: 1.000000 test_acc: 1.000000
epoch: 248 loss: 0.000027 train_acc: 1.000000 test_acc: 1.000000
epoch: 249 loss: 0.000027 train_acc: 1.000000 test_acc: 1.000000
epoch: 250 loss: 0.000009 train_acc: 1.000000 test_acc: 1.000000

Process finished with exit code 0



AdamOptimizer, Tensor flow:



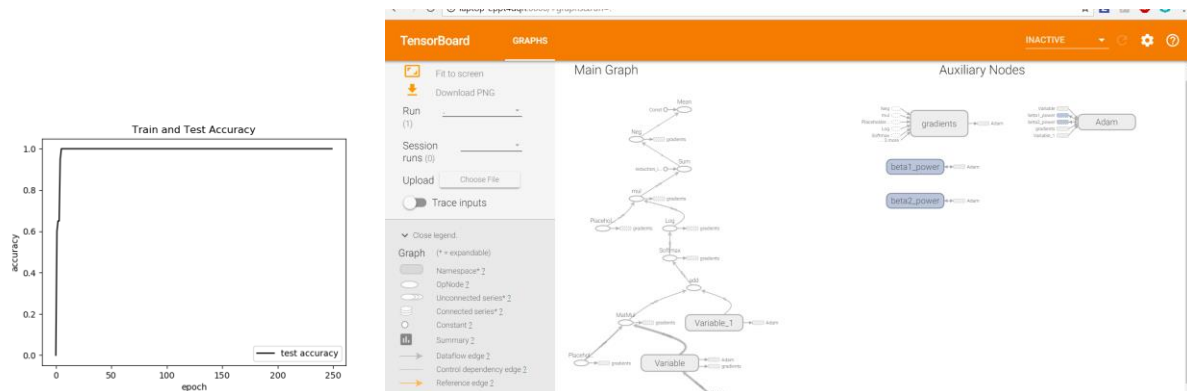
The image shows a PyCharm IDE window titled "DeepLearning_Lesson4 [C:\Users\praga\OneDrive\Desktop\Python lab 3\DeepLearning_Lesson4] - \DeepLearning_Lesson4\logisticregression.py [DeepLearning_Lesson4] - PyCharm". The editor displays a Python script for a logistic regression model using TensorFlow and the AdamOptimizer. The script includes comments and code for loading data, defining the model, training, and evaluating accuracy. The Run window shows the execution output, indicating that the training process finished with an exit code of 0. The output shows the loss and accuracy for each epoch from 245 to 250.

```
46 # Declare the model you need to learn
47 Y_prediction = tf.add(tf.matmul(X, weights), base)
48 # Declare loss function
49 # Use the sigmoid cross-entropy loss function,
50 # first doing a sigmoid on the model result and then using the cross-entropy loss function
51 loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=Y_prediction, labels=y))
52 # Define the learning rate, batch_size etc.
53 learning_rate = 0.3
54 batch_size = 50
55 iter_num = 250
56 # Define the optimizer
57 opt = tf.train.AdamOptimizer(learning_rate)
58 # Define the goal
59 goal = opt.minimize(loss)
60 # Define the accuracy
61 # The default threshold is 0.5, rounded off directly
62 prediction = tf.round(tf.sigmoid(tf.add(tf.matmul(X, weights), base)))
63 # Bool into float32 type
64 correct = tf.cast(tf.equal(prediction, y), dtype=tf.float32)
```

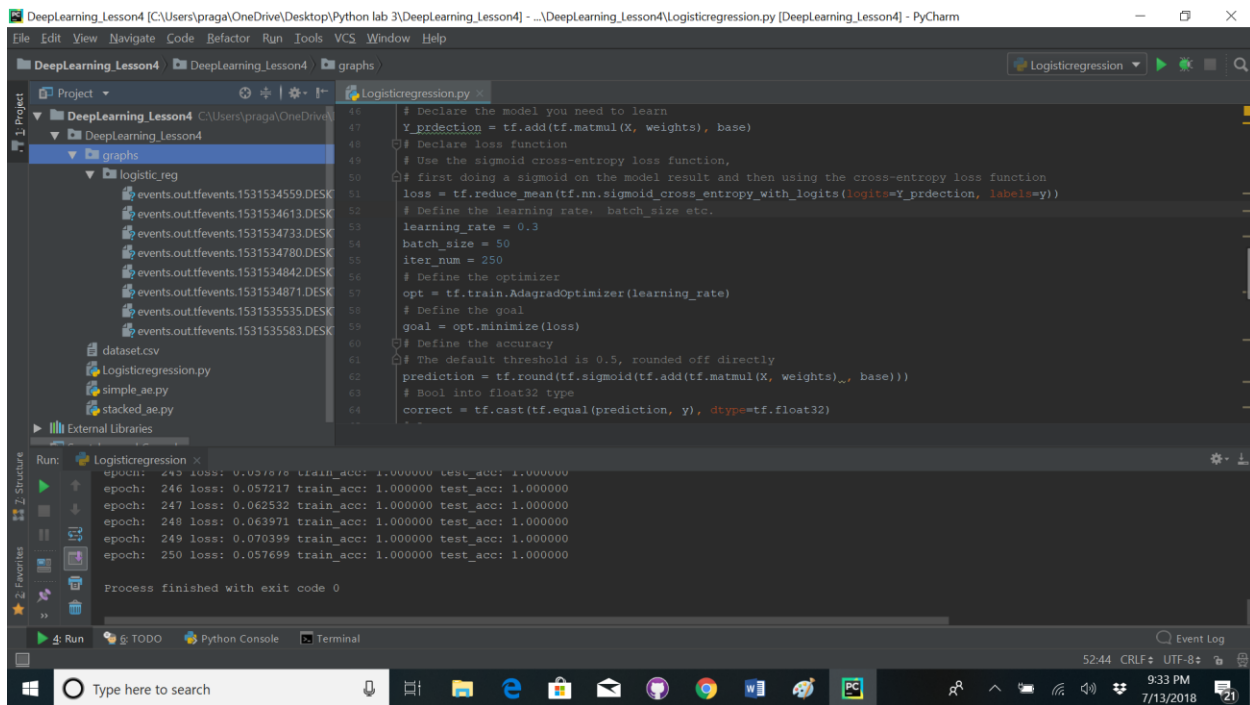
Run: Logisticregression X

epoch: 245 loss: 0.002974 train_acc: 1.000000 test_acc: 1.000000
epoch: 246 loss: 0.002775 train_acc: 1.000000 test_acc: 1.000000
epoch: 247 loss: 0.003532 train_acc: 1.000000 test_acc: 1.000000
epoch: 248 loss: 0.003898 train_acc: 1.000000 test_acc: 1.000000
epoch: 249 loss: 0.004497 train_acc: 1.000000 test_acc: 1.000000
epoch: 250 loss: 0.002525 train_acc: 1.000000 test_acc: 1.000000

Process finished with exit code 0



AdagradOptimizer, Tensor flow:

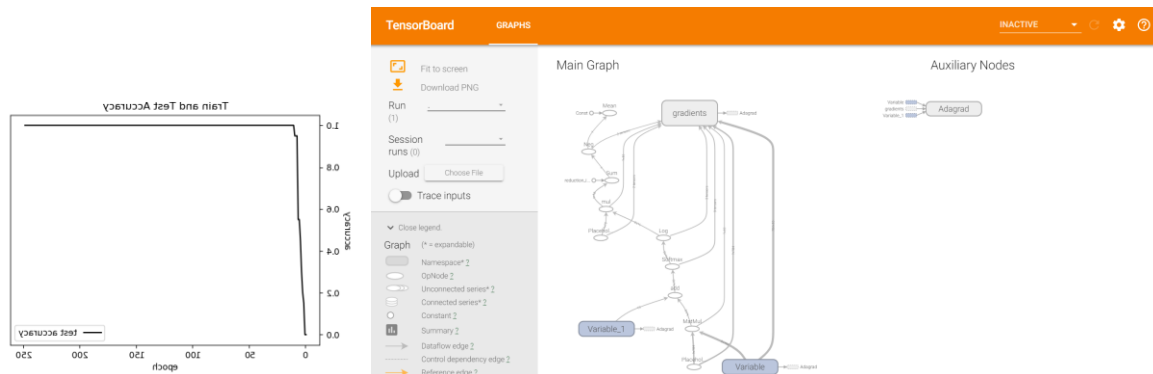


```
# Declare the model you need to learn
Y_prediction = tf.add(tf.matmul(X, weights), base)
# Declare loss function
# Use the sigmoid cross-entropy loss function,
# first doing a sigmoid on the model result and then using the cross-entropy loss function
loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=Y_prediction, labels=y))
# Define the learning rate, batch_size etc.
learning_rate = 0.3
batch_size = 50
iter_num = 250
# Define the optimizer
opt = tf.train.AdagradOptimizer(learning_rate)
# Define the goal
goal = opt.minimize(loss)
# Define the accuracy
# The default threshold is 0.5, rounded off directly
prediction = tf.round(tf.sigmoid(tf.add(tf.matmul(X, weights), base)))
# Bool into float32 type
correct = tf.cast(tf.equal(prediction, y), dtype=tf.float32)
```

Run: LogisticRegression X

epoch	loss	train_acc	test_acc
245	0.057076	1.000000	1.000000
246	0.057217	1.000000	1.000000
247	0.062532	1.000000	1.000000
248	0.063971	1.000000	1.000000
249	0.070399	1.000000	1.000000
250	0.057699	1.000000	1.000000

Process finished with exit code 0



Question 2:

2. Implement the Word Embeddings with new data set which is not used in class
- Show the results in TensorBoard
Change the hyperparameter and compare the result

Solution:

This snippet code provides the above implementation For this task we have chosen the ewik8 dataset in 'http://mattmahoney.net/dc/enwik8.zip' URL. We have changed the different hyper parameters like optimizers, learning rate, number of steps, step size and observed the loss.

- **Data Set : enwik8.zip**

Code Snippet:

The below is the code for executing the above workflow

```

255 lines (187 sloc) | 7.6 KB
Raw Blame History

1  ## Word2Vec
2
3  # The code for this lecture is based off the great tutorial example from tensorflow!
4  # Walk through:
5  # https://www.tensorflow.org/tutorials/word2vec
6  # Raw Code: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/word2vec/word2vec_basic.py
7
8  ## Step 0: Imports
9  import collections
10 import math
11 import os
12 import errno
13 import random
14 import zipfile
15
16 import numpy as np
17 from six.moves import urllib
18 from six.moves import xrange
19 from collections import Counter
20 import tensorflow as tf
21
22 ## Step 1: The data.
23 data_dir = "data"
24 data_url = 'http://mattdahoney.net/dc/enwik8.zip'
25
26
27 def fetch_words_data(url=data_url, words_data=data_dir):
28     # Make the Dir if it does not exist
29     os.makedirs(words_data, exist_ok=True)
30
31     # Path to zip file
32     zip_path = os.path.join(words_data, "enwik8.zip")

```

Analysis depending upon hyperparameters:

Word Embeddings	Model Learning rate	Optimizer	Window Size	Embedding Size	No.of steps	Loss rate
	0.01	Adam Optimizer	100	150	2001	15106.7
	0.02	Adam Optimizer	120	170	5000	2932.53
	0.01	RMS Optimizer	100	100	7000	2428.24

Output Screenshot:

The screenshot shows the PyCharm IDE with a project named 'DeepLearning_Lesson3'. The 'Word2Vec.py' file is open, displaying a snippet of code for training and visualizing word embeddings using T-SNE. The code includes comments about the source (https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding) and the dimensionality reduction process. The 'Run' output window shows the execution progress, including the total length of words (13303079) and the average loss at step 1000 (7520.668123916626). A notification for PyCharm updates is visible in the bottom right corner.

```
238 # * https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding
239 # # Dimensionality reduction to 2-D vectors (down from 150), this takes awhile.
240
241 from sklearn.manifold import TSNE
242
243 tsne = TSNE(perplexity=30, n_components=2, init='pca', n_iter=5000)
244 plot_only = 2000
245 low_dim_embs = tsne.fit_transform(final_embeddings[:plot_only, :])
246
247 labels = [vocabulary[i] for i in range(plot_only)]
248
249 plot_with_labels(low_dim_embs, labels)
250 plt.xlim(-10, 10)
251 plt.ylim(-10, 10)
252 plt.show()
253
254 ## Great Job!
```

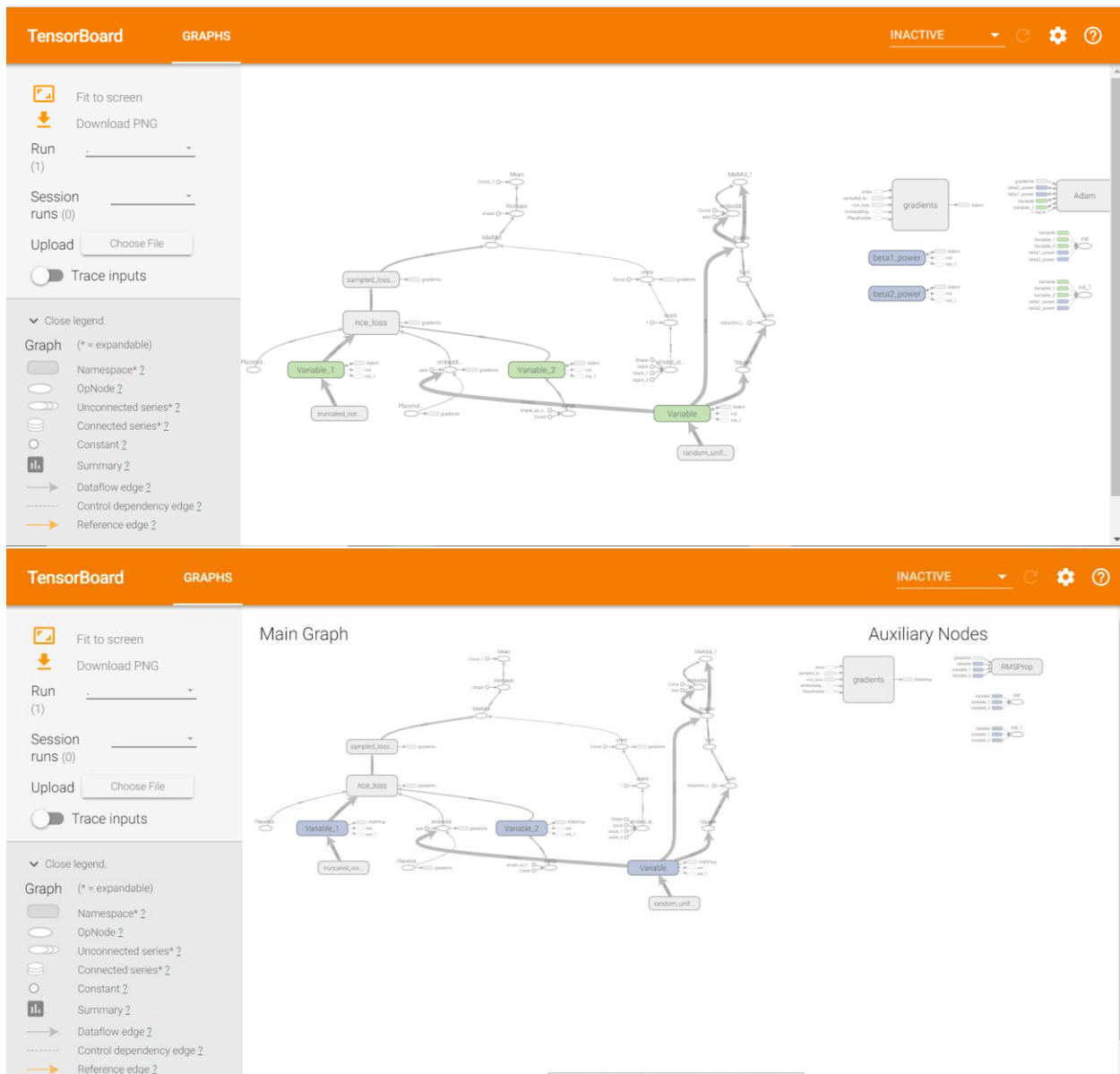
Run: Word2Vec x
"C:\Users\praga\OneDrive\Desktop\Summer Sem\Bigdata\Module 2_curent\sparkInterpeter\Scripts\python.exe" "C:\Users\praga\OneDrive\Desktop\Python lab 3\DeepLearning_Lesson3\Word2Vec.py"
Total length of words is: 13303079
2018-07-13 19:46:49.512568: I T:\src\github\tensorflow\tensorflow\core\platform\cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow binary is not built to use. Please see the readme for supported instructions.
of five and, possibly, other higher [[primate]]s such as adult [[gorilla]]s, [[Common chimpanzee|chimpanzee]]s and [[bonobos]]. Typical 5-year-olds can develop i
Average loss at step 1000 : 7520.668123916626
Average loss at step 2000 : 15106.676568908691

The screenshot shows the PyCharm IDE with the 'Word2Vec.py' file open, displaying a snippet of code for training and visualizing word embeddings using TensorFlow. The code includes comments about the session and the training process. The 'Run' output window shows the execution progress, including the average loss at various steps (4200 to 4900). A notification for PyCharm updates is visible in the bottom right corner.

```
189
190 # # Session
191 # Usually needs to be quite large to get good results,
192 # training takes a long time!
193 num_steps = 5000
194
195 with tf.Session() as sess:
196     sess.run(init)
197     average_loss = 0
198     for step in range(num_steps):
199
200         batch_inputs, batch_labels = generate_batch(batch_size, num_skips, skip_window)
201         feed_dict = {train_inputs: batch_inputs, train_labels: batch_labels}
202
203         # We perform one update step by evaluating the training op (including it
204         # in the list of returned values for session.run()
205         empty, loss_val = sess.run([trainer, loss], feed_dict=feed_dict)
206         average_loss += loss_val
```

Run: Word2Vec x
Average loss at step 4200 : 2929.8875218811036
Average loss at step 4300 : 3573.751421081543
Average loss at step 4400 : 3266.1802450256346
Average loss at step 4500 : 2741.9141599121094
Average loss at step 4600 : 2358.665046508789
Average loss at step 4700 : 2123.9738553466796
Average loss at step 4800 : 3390.033319458008
Average loss at step 4900 : 2932.538003326416

Tensor graph Adam and RMS optimizers:



Final observation:

Depending upon the variation in hyper parameters there is a markable difference in accuracy and loss, and it also depends on the datasets too.

Conclusion: As stated the above workflow with certain set of parameters is followed in solving the execution by implementing the core and basic concepts of the deep learning programming.

Source code link <https://github.com/PragathiThammaneni/Python-and-deep-Learning/tree/master/Labs/Lab%203>

Video Link: <https://youtu.be/1D264Slytfk>

Wiki Link: <https://github.com/PragathiThammaneni/Python-and-deep-Learning/wiki/Lab-3-Assignment>