CSEE 5590 0001
Special Topics SPRING 2018

DEEP LEARNING
LAB ASSIGNMENT - 3

Submitted On 5/9/2018

Name: Satya Sai Deepthi Katta

Class ID: 21

UMKC

Department of Computer Science and Electrical Engineering

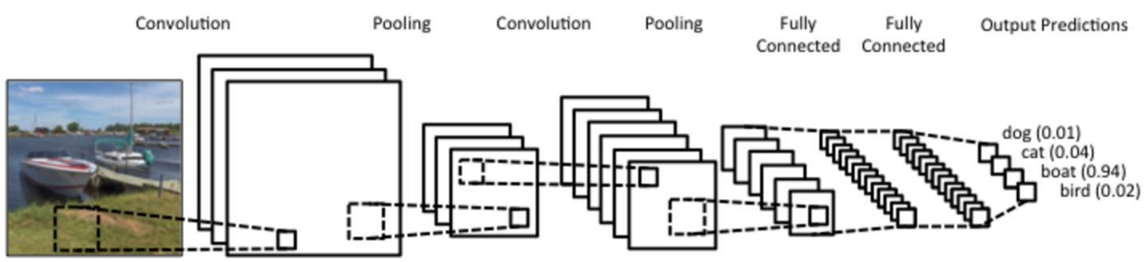
INTRODUCTION:

Text Classification is classifying the data based on some feature like email-spam filtering i.e, spam vs ham, sentiment analysis i.e; positives vs negatives. For most of the real world applications, text classification is used in many new stories developed by topics, product categories tags etc.

Three main techniques are used in classifying the text:

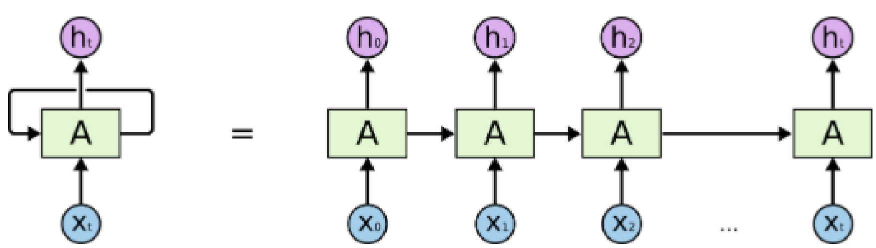
CNN – Convolutional Neural Network

This also known as ConvNet is used in analyzing visual imagery. The is class of deep learning technique part of feed forward neural network which doesn't contain any loop in the network.



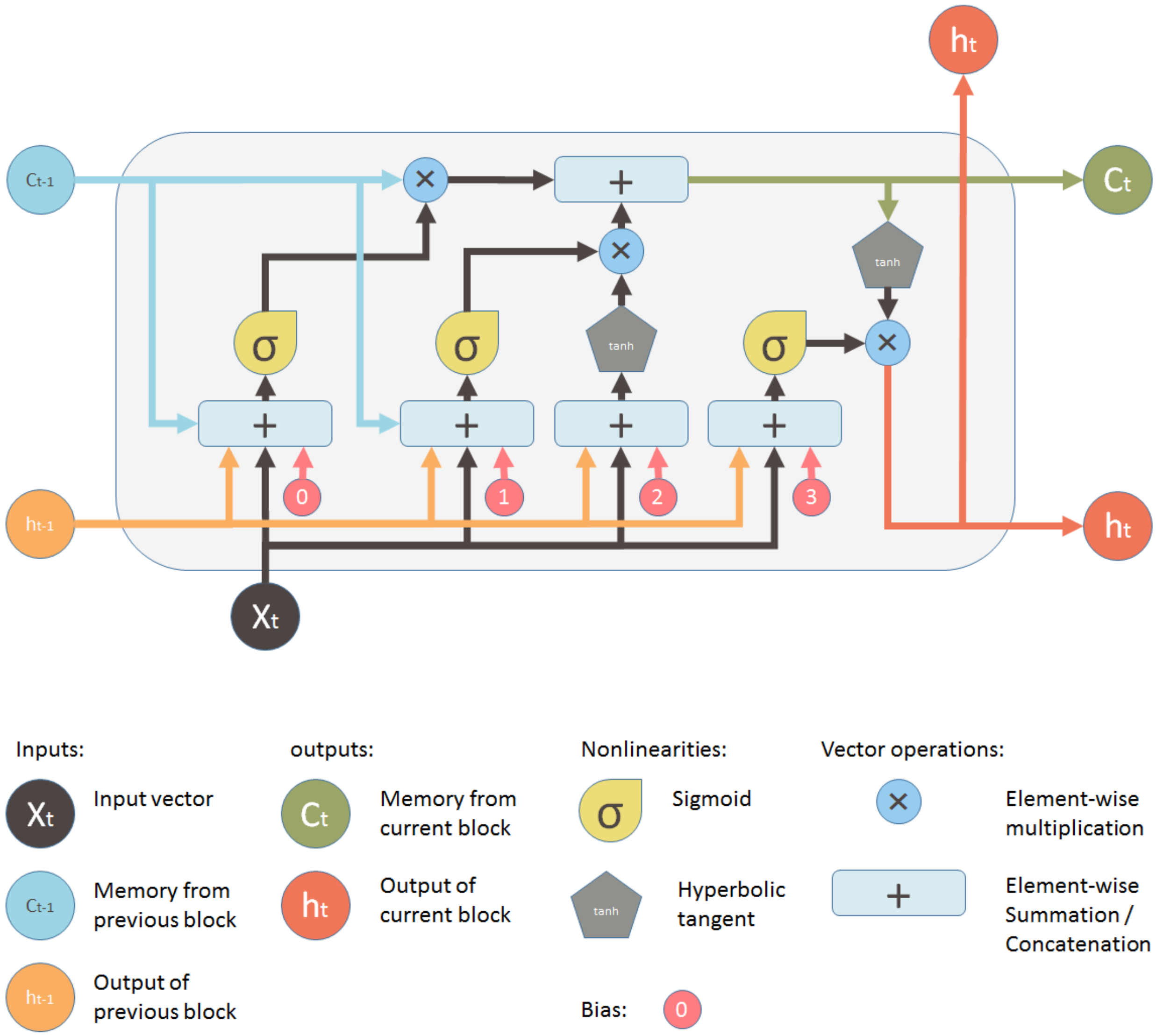
RNN – Recurrent Neural Network

This is class of artificial neural network which follows a directed graph in a time sequence. Unlike feed forward network, these have loops which direct from output of input. RNN is used for unsegmented data like handwriting or speech recognition.



LSTM – Long Short-Term Memory

These are the unit blocks of the Recurrent neural networks which helps in building layers. These have cell, input, output and forget gates in the model. The cell usually has the remembering values which contribute to the "memory" segment of the RNN.



OBJECTIVE:

The object of the assignment is to give basic understanding behind the difference in using different text classification on a new dataset which is not used in the class.

And also, to compare the results plotting the workflows and graphs for scalars like accuracy and loss in tensor board.

APPROACH:

The approach is explained in simple steps below:

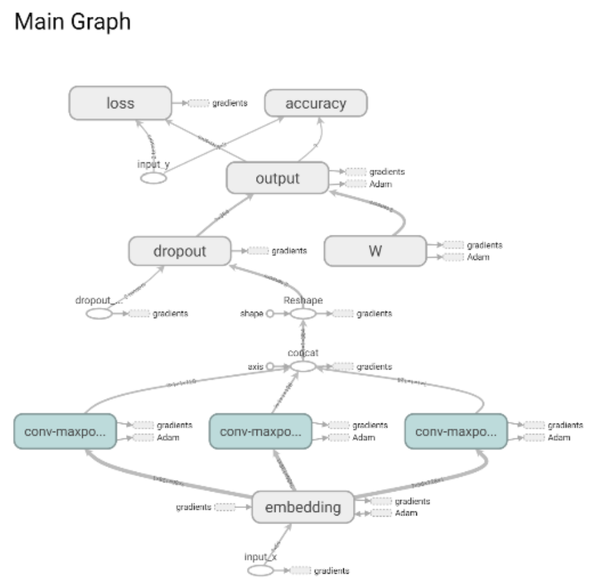
- Extract data from the dataset
- Weights and Bias assigned by Variables
- Inputs are given by Placeholders
- Prediction Model is built
- Trained using training data
- Optimizer used in reducing loss
- Loss and Accuracy Graphs Plotted
- Comparisons made using results

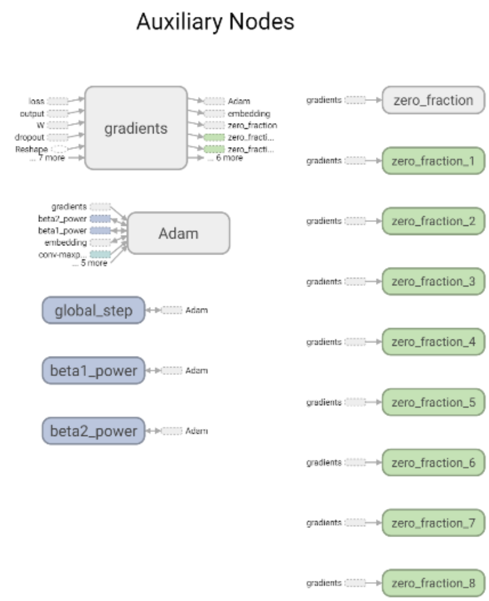
PARAMETERS:

Hyper parameters used for comparing the results by changing the values are listed below:
Learning Rate = 0.001
Training Iterations = 10000
Displaying step = 1000
No. of inputs = 2
No. of hidden cells = 10

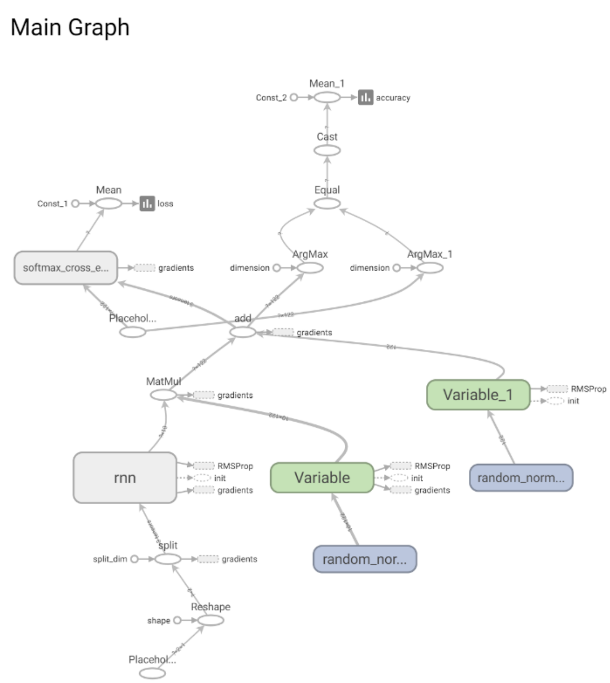
WORKFLOWS:

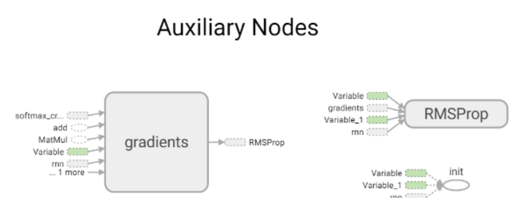
CNN





RNN & LSTM





DATASET

The dataset used here is few lines of text downloaded from GOOGLE Wikipedia. The text contains 500 words separated into 15 lines. The first paragraph of Google Wikipedia explaining company's history is considered as the dataset for text classification.

CONFIGURATION:

Code is built on PYCHARM Software using the advanced version of python programming language v3.6.4.

EVALUATION & DISCUSSION

Code snippets are provided below:

RNN

Importing required functions for the code. A start time is initiated as soon as the game begins.

```
from __future__ import print_function

import numpy as np
import tensorflow as tf
from tensorflow.contrib import rnn
import random
import collections
import time

start_time = time.time()


def elapsed(sec):
    if sec < 60:
        return str(sec) + " sec"
    elif sec < (60 * 60):
        return str(sec / 60) + " min"
    else:
        return str(sec / (60 * 60)) + " hr"
```

A training file is given as input where the text data is present. Read Data method helps to extract the data from the document.

```
# Text file containing words for training
training_file = 'google.txt'


def read_data(fname):
    with open(fname) as f:
        content = f.readlines()
    content = [x.strip() for x in content]
    content = [content[i].split() for i in range(len(content))]
    content = np.array(content)
    content = np.reshape(content, [-1, ])
    return content


training_data = read_data(training_file)
print("Loaded training data...")
```

The dataset is built into set of words storing in the dictionaries. Parameters are initialized which are used for changing for getting the outputs.

```
def build_dataset(words):
    count = collections.Counter(words).most_common()
    dictionary = dict()
    for word, _ in count:
        dictionary[word] = len(dictionary)
    reverse_dictionary = dict(zip(dictionary.values(), dictionary.keys()))
    return dictionary, reverse_dictionary


dictionary, reverse_dictionary = build_dataset(training_data)
vocab_size = len(dictionary)

# Parameters
learning_rate = 0.001
training_iters = 10000
display_step = 1000
n_input = 2

# number of units in RNN cell
n_hidden = 10
```

Inputs are initialized using place holders, weights and bias are given initialized using variables.

```
# tf Graph input
x = tf.placeholder("float", [None, n_input, 1])
y = tf.placeholder("float", [None, vocab_size])

# RNN output node weights and biases
weights = {
    'out': tf.Variable(tf.random_normal([n_hidden, vocab_size]))
}
biases = {
    'out': tf.Variable(tf.random_normal([vocab_size]))
}
```

RNN function is used in defining the layers based on the no. of hidden layers initiated in the code.

```
def RNN(x, weights, biases):
    # reshape to [1, n_input]
    x = tf.reshape(x, [-1, n_input])

    # Generate a n_input-element sequence of inputs
    # (eg. [had] [a] [general] -> [20] [6] [33])
    x = tf.split(x, n_input, 1)

    # 2-layer LSTM, each layer has n_hidden units.
    # Average Accuracy= 95.20% at 50k iter
    # rnn_cell = rnn.MultiRNNCell([rnn.BasicLSTMCell(n_hidden),rnn.BasicLSTMCell(n_hidden)])

    # 1-layer LSTM with n_hidden units but with lower accuracy.
    # Average Accuracy= 90.60% 50k iter
    # Uncomment line below to test but comment out the 2-layer rnn.MultiRNNCell above
    rnn_cell = rnn.BasicLSTMCell(n_hidden)

    # generate prediction
    outputs, states = rnn.static_rnn(rnn_cell, x, dtype=tf.float32)

    # there are n_input outputs but
    # we only want the last output
    return tf.matmul(outputs[-1], weights['out']) + biases['out']
```

Loss and optimizer are varied based on the learning rate and RMS propagation optimizer gives best optimizing results.

```
pred = RNN(x, weights, biases)

# Loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
optimizer = tf.train.RMSPropOptimizer(learning_rate=learning_rate).minimize(cost)

# Model evaluation
correct_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initializing the variables
init = tf.global_variables_initializer()
loss_summary=tf.summary.scalar('loss',cost)
accuracy_summary=tf.summary.scalar('accuracy',accuracy)
merged= tf.summary.merge_all()
```

A session is created to store instances for every iteration.

```
# Launch the graph
with tf.Session() as session:
    session.run(init)

    writer = tf.summary.FileWriter('./graphs/rnn', session.graph)

    step = 0
    offset = random.randint(0, n_input + 1)
    end_offset = n_input + 1
    acc_total = 0
    loss_total = 0

    while step < training_iters:
        # Generate a minibatch. Add some randomness on selection process.

        if offset > (len(training_data) - end_offset):
            offset = random.randint(0, n_input + 1)

        symbols_in_keys = [[dictionary[str(training_data[i])]] for i in range(offset, offset + n_input)]
        symbols_in_keys = np.reshape(np.array(symbols_in_keys), [-1, n_input, 1])

        symbols_out_onehot = np.zeros([vocab_size], dtype=float)
        symbols_out_onehot[dictionary[str(training_data[offset + n_input])]] = 1.0
        symbols_out_onehot = np.reshape(symbols_out_onehot, [1, -1])

        _, acc, loss, onehot_pred,summary = session.run([optimizer, accuracy, cost, pred,merged], \
                                                feed_dict={x: symbols_in_keys, y: symbols_out_onehot})
        loss_total += loss
        acc_total += acc
```

The loss and accuracy are displayed for every 100th step.

```
        if (step + 1) % display_step == 0:
            print("Iter= " + str(step + 1) + ", Average Loss= " + \
                  "{:.6f}".format(loss_total / display_step) + ", Average Accuracy= " + \
                  "{:.2f}%".format(100 * acc_total / display_step))
            acc_total = 0
            loss_total = 0
            writer.add_summary(summary,step)

            symbols_in = [training_data[i] for i in range(offset, offset + n_input)]
            symbols_out = training_data[offset + n_input]
            symbols_out_pred = reverse_dictionary[int(tf.argmax(onehot_pred, 1).eval())]
            print("%s - [%s] vs [%s]" % (symbols_in, symbols_out, symbols_out_pred))
        step += 1
        offset += (n_input + 1)
    print("Optimization Finished!")
    print("Elapsed time: ", elapsed(time.time() - start_time))
    print("Run on command line.")
```

If the given words are present in the dictionary then rest of sentence is printed. If the words are not present then it shows an error message that words are not present.

```
    while True:
        prompt = "%s words: " % n_input
        sentence = input(prompt)
        sentence = sentence.strip()
        words = sentence.split(' ')
        if len(words) != n_input:
            continue
        try:
            symbols_in_keys = [dictionary[str(words[i])] for i in range(len(words))]
            for i in range(32):
                keys = np.reshape(np.array(symbols_in_keys), [-1, n_input, 1])
                onehot_pred = session.run(pred, feed_dict={x: keys})
                onehot_pred_index = int(tf.argmax(onehot_pred, 1).eval())
                sentence = "%s %s" % (sentence, reverse_dictionary[onehot_pred_index])
                symbols_in_keys = symbols_in_keys[1:]
                symbols_in_keys.append(onehot_pred_index)
            print(sentence)
        except:
            print("Word not in dictionary")
```

CNN

Initiate code mu importing required functions. Parameters are defined for the dataset taken to display the results.

```
import tensorflow as tf
import numpy as np
import os
import time
import datetime
import data_helpers
from cnn import TextCNN
from tensorflow.contrib import learn
```

```
# Parameters
# ==================================================

# Data loading params
tf.flags.DEFINE_float("data_sample", .1, "Percentage of the training data to use for validation")
tf.flags.DEFINE_string("google_story", "./data/google.txt", "Data source for the google_story data.")
tf.flags.DEFINE_string("google_data_file", "./data/google_analysis.txt", "Data source for the google data.")

# Model Hyperparameters
tf.flags.DEFINE_integer("embedding_dim", 128, "Dimensionality of character embedding (default: 128)")
tf.flags.DEFINE_string("filter_sizes", "3,4,5", "Comma-separated filter sizes (default: '3,4,5')")
tf.flags.DEFINE_integer("num_filters", 128, "Number of filters per filter size (default: 128)")
tf.flags.DEFINE_float("dropout_keep_prob", 0.5, "Dropout keep probability (default: 0.5)")
tf.flags.DEFINE_float("l2_reg_lambda", 0.0, "L2 regularization lambda (default: 0.0)")

# Training parameters
tf.flags.DEFINE_integer("batch_size", 64, "Batch Size (default: 64)")
tf.flags.DEFINE_integer("num_epochs", 200, "Number of training epochs (default: 200)")
tf.flags.DEFINE_integer("evaluate_every", 100, "Evaluate model on dev set after this many steps (default: 100)")
tf.flags.DEFINE_integer("checkpoint_every", 100, "Save model after this many steps (default: 100)")
tf.flags.DEFINE_integer("num_checkpoints", 5, "Number of checkpoints to store (default: 5)")
# Misc Parameters
tf.flags.DEFINE_boolean("allow_soft_placement", True, "Allow device soft device placement")
tf.flags.DEFINE_boolean("log_device_placement", False, "Log placement of ops on devices")
```

```python
FLAGS = tf.flags.FLAGS
FLAGS._parse_flags()
print("\nParameters:")
for attr, value in sorted(FLAGS.__flags.items()):
    print("{}={}".format(attr.upper(), value))
print("")


# Data Preparation
# ==================================================

# Load data
print("Loading data...")
x_text, y = data_helpers.load_data_and_labels(FLAGS.google_story, FLAGS.google_data_file)

# Build vocabulary
max_document_length = max([len(x.split(" ")) for x in x_text])
vocab_processor = learn.preprocessing.VocabularyProcessor(max_document_length)
x = np.array(list(vocab_processor.fit_transform(x_text)))

# Randomly shuffle data
np.random.seed(10)
shuffle_indices = np.random.permutation(np.arange(len(y)))
x_shuffled = x[shuffle_indices]
y_shuffled = y[shuffle_indices]
```

Dataset is split into training and testing data.

```python
# Split train/test set
# TODO: This is very crude, should use cross-validation
dev_sample_index = -1 * int(FLAGS.dev_sample_percentage * float(len(y)))
x_train, x_dev = x_shuffled[:dev_sample_index], x_shuffled[dev_sample_index:]
y_train, y_dev = y_shuffled[:dev_sample_index], y_shuffled[dev_sample_index:]
print("Vocabulary Size: {:d}".format(len(vocab_processor.vocabulary_)))
print("Train/Dev split: {:d}/{:d}".format(len(y_train), len(y_dev)))
```

Graph sessions are taken to plot loss and accuracy for 10000 epochs displaying at each 1000 steps.

```
# Training
# ==================================================

with tf.Graph().as_default():
    session_conf = tf.ConfigProto(
      allow_soft_placement=FLAGS.allow_soft_placement,
      log_device_placement=FLAGS.log_device_placement)
    sess = tf.Session(config=session_conf)
    with sess.as_default():
        cnn = TextCNN(
            sequence_length=x_train.shape[1],
            num_classes=y_train.shape[1],
            vocab_size=len(vocab_processor.vocabulary_),
            embedding_size=FLAGS.embedding_dim,
            filter_sizes=list(map(int, FLAGS.filter_sizes.split(","))),
            num_filters=FLAGS.num_filters,
            l2_reg_lambda=FLAGS.l2_reg_lambda)

        # Define Training procedure
        global_step = tf.Variable(0, name="global_step", trainable=False)
        optimizer = tf.train.AdamOptimizer(1e-3)
        grads_and_vars = optimizer.compute_gradients(cnn.loss)
        train_op = optimizer.apply_gradients(grads_and_vars, global_step=global_step)
```

```
# Keep track of gradient values and sparsity (optional)
grad_summaries = []
for g, v in grads_and_vars:
    if g is not None:
        grad_hist_summary = tf.summary.histogram("{}/grad/hist".format(v.name), g)
        sparsity_summary = tf.summary.scalar("{}/grad/sparsity".format(v.name), tf.nn.zero_fraction(g))
        grad_summaries.append(grad_hist_summary)
        grad_summaries.append(sparsity_summary)
grad_summaries_merged = tf.summary.merge(grad_summaries)

# Output directory for models and summaries
timestamp = str(int(time.time()))
out_dir = os.path.abspath(os.path.join(os.path.curdir, "runs", timestamp))
print("Writing to {}\n".format(out_dir))

# Summaries for loss and accuracy
loss_summary = tf.summary.scalar("loss", cnn.loss)
acc_summary = tf.summary.scalar("accuracy", cnn.accuracy)

# Train Summaries
train_summary_op = tf.summary.merge([loss_summary, acc_summary, grad_summaries_merged])
train_summary_dir = os.path.join(out_dir, "summaries", "train")
train_summary_writer = tf.summary.FileWriter(train_summary_dir, sess.graph)

# Dev summaries
dev_summary_op = tf.summary.merge([loss_summary, acc_summary])
dev_summary_dir = os.path.join(out_dir, "summaries", "dev")
dev_summary_writer = tf.summary.FileWriter(dev_summary_dir, sess.graph)
```

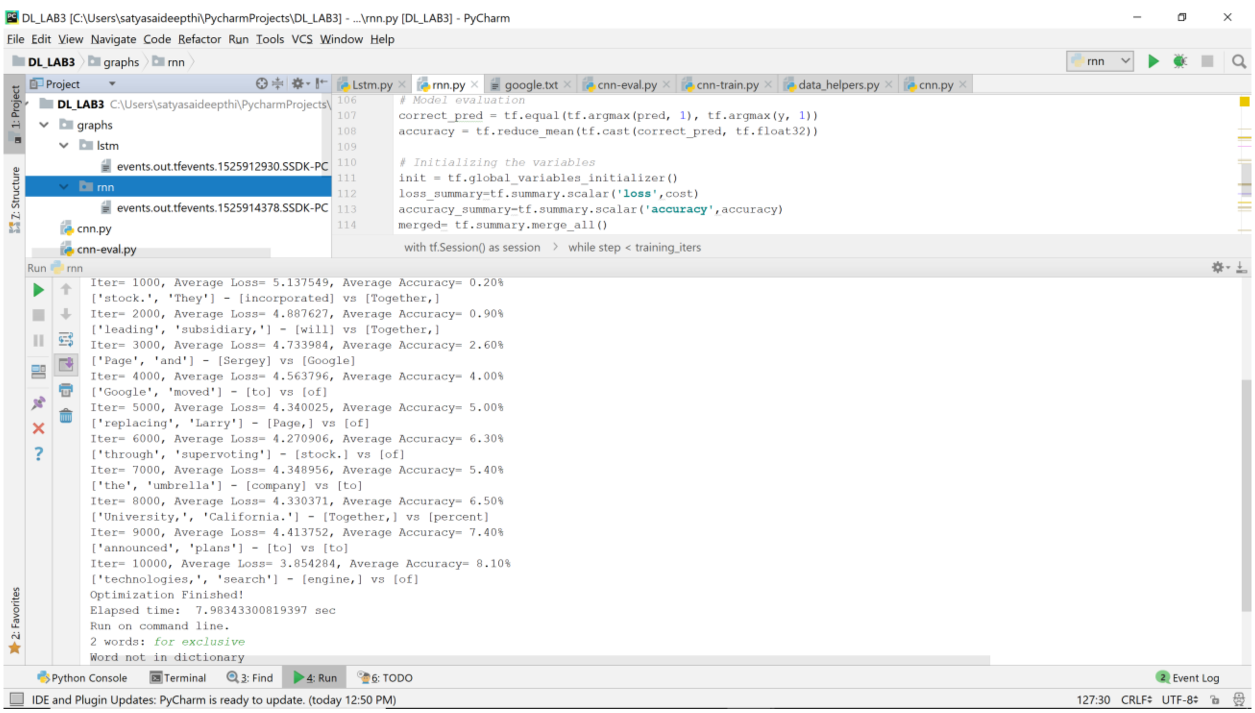
```python
# Checkpoint directory. Tensorflow assumes this directory already exists so we need to create it
checkpoint_dir = os.path.abspath(os.path.join(out_dir, "checkpoints"))
checkpoint_prefix = os.path.join(checkpoint_dir, "model")
if not os.path.exists(checkpoint_dir):
    os.makedirs(checkpoint_dir)
saver = tf.train.Saver(tf.global_variables(), max_to_keep=FLAGS.num_checkpoints)

# Write vocabulary
vocab_processor.save(os.path.join(out_dir, "vocab"))

# Initialize all variables
sess.run(tf.global_variables_initializer())

def train_step(x_batch, y_batch):
    """
    A single training step
    """
    feed_dict = {
      cnn.input_x: x_batch,
      cnn.input_y: y_batch,
      cnn.dropout_keep_prob: FLAGS.dropout_keep_prob
    }
    _, step, summaries, loss, accuracy = sess.run(
        [train_op, global_step, train_summary_op, cnn.loss, cnn.accuracy],
        feed_dict)
    time_str = datetime.datetime.now().isoformat()
    print("{}: step {}, loss {:g}, acc {:g}".format(time_str, step, loss, accuracy))
    train_summary_writer.add_summary(summaries, step)
```

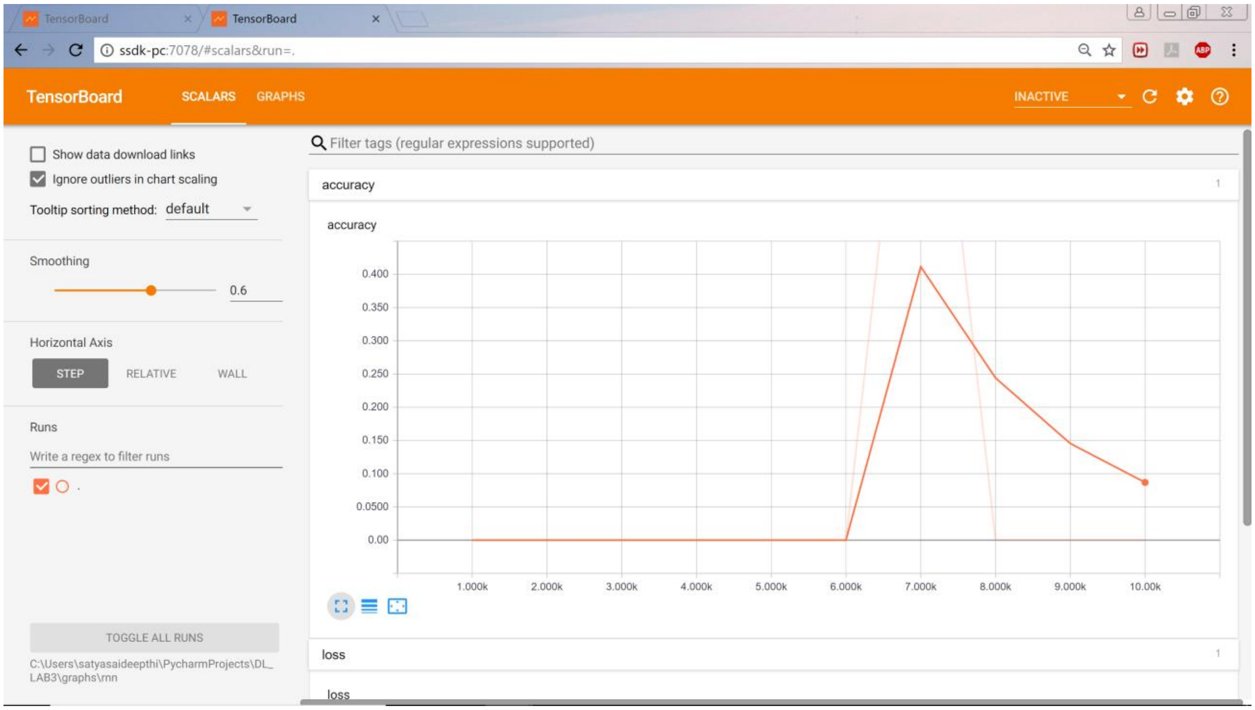
```python
def dev_step(x_batch, y_batch, writer=None):
    """
    Evaluates model on a dev set
    """
    feed_dict = {
      cnn.input_x: x_batch,
      cnn.input_y: y_batch,
      cnn.dropout_keep_prob: 1.0
    }
    step, summaries, loss, accuracy = sess.run(
        [global_step, dev_summary_op, cnn.loss, cnn.accuracy],
        feed_dict)
    time_str = datetime.datetime.now().isoformat()
    print("{}: step {}, loss {:g}, acc {:g}".format(time_str, step, loss, accuracy))
    if writer:
        writer.add_summary(summaries, step)
```

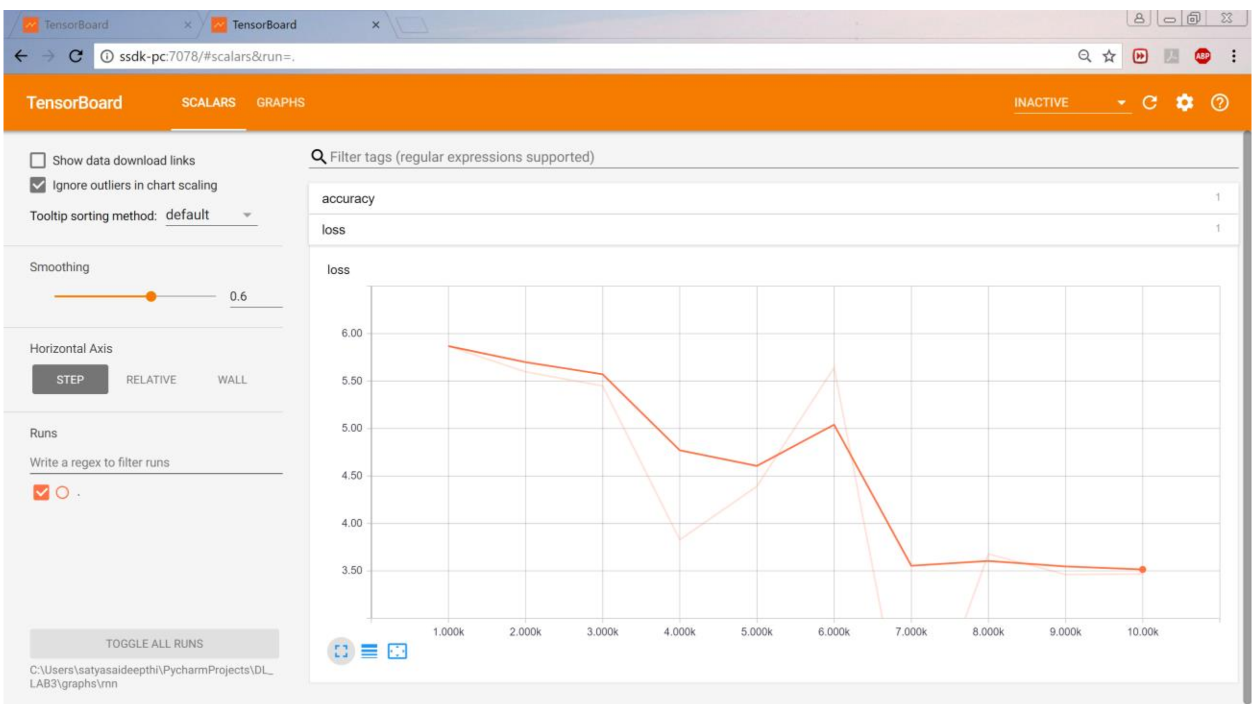
```python
    # Generate batches
    batches = data_helpers.batch_iter(
        list(zip(x_train, y_train)), FLAGS.batch_size, FLAGS.num_epochs)
    # Training loop. For each batch...
    for batch in batches:
        x_batch, y_batch = zip(*batch)
        train_step(x_batch, y_batch)
        current_step = tf.train.global_step(sess, global_step)
        if current_step % FLAGS.evaluate_every == 0:
            print("\nEvaluation:")
            dev_step(x_dev, y_dev, writer=dev_summary_writer)
            print("")
        if current_step % FLAGS.checkpoint_every == 0:
            path = saver.save(sess, checkpoint_prefix, global_step=current_step)
            print("Saved model checkpoint to {}\n".format(path))
```

RESULTS:

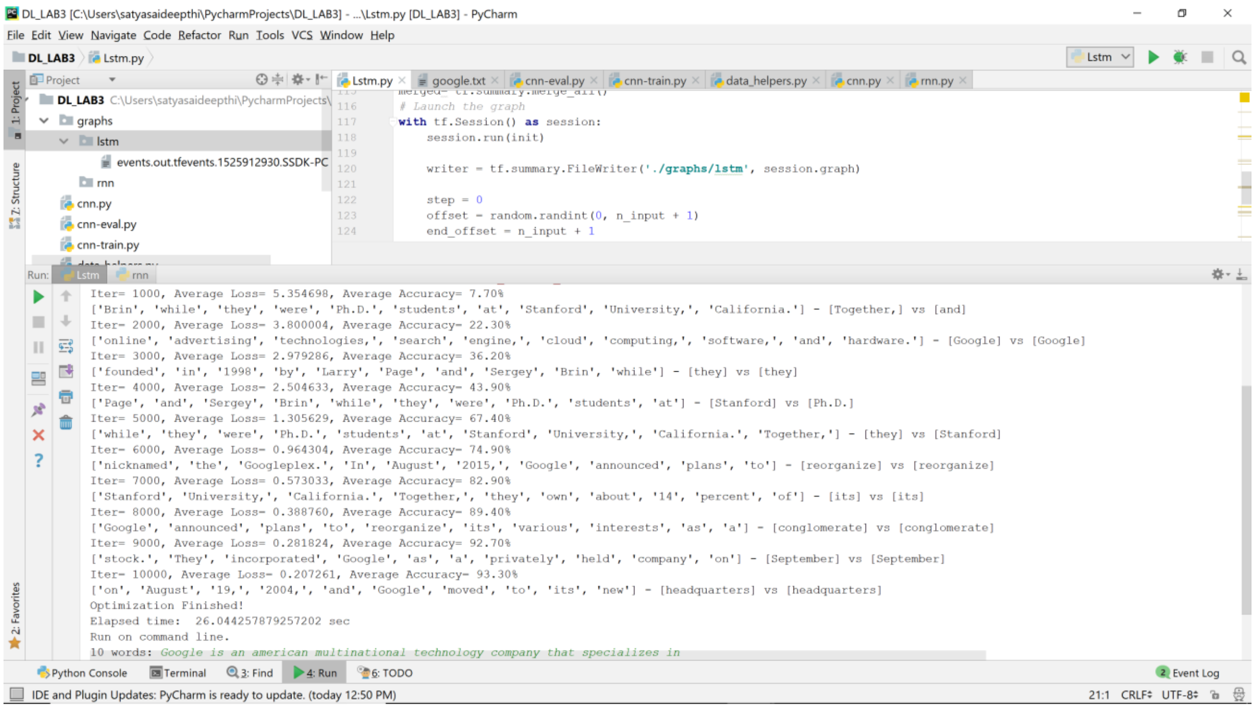
Code is deployed on pycharm console showing accuracy and loss for every 1000 iteration.

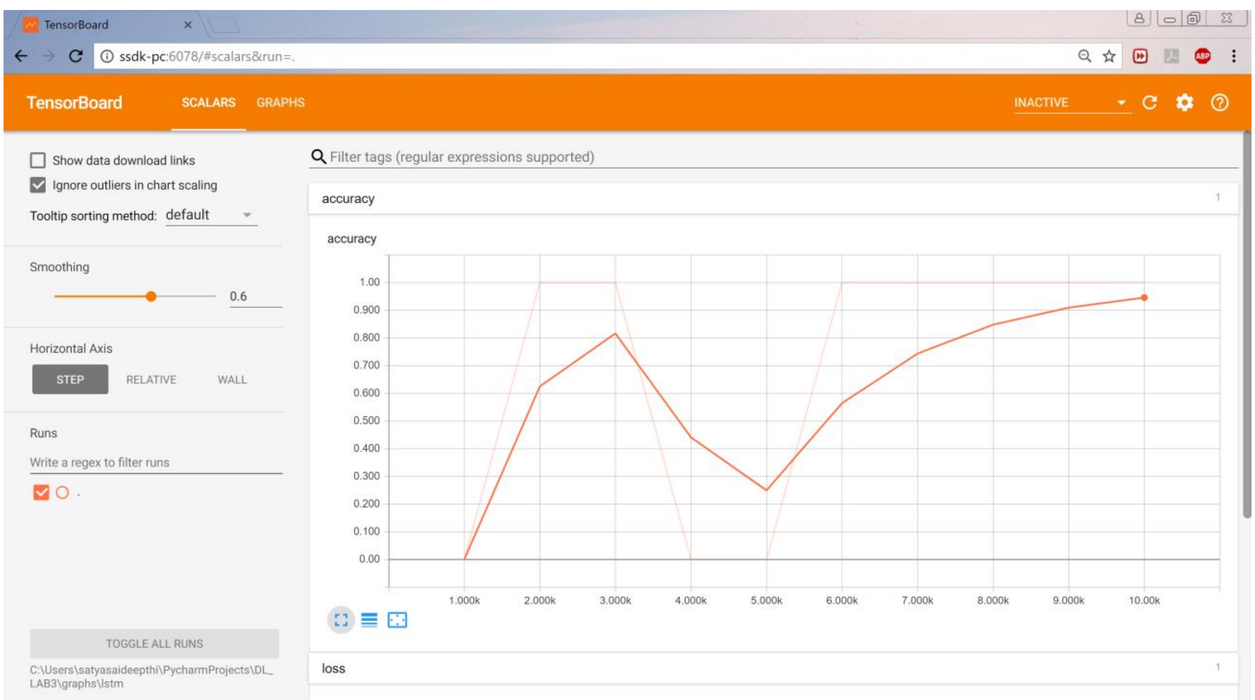


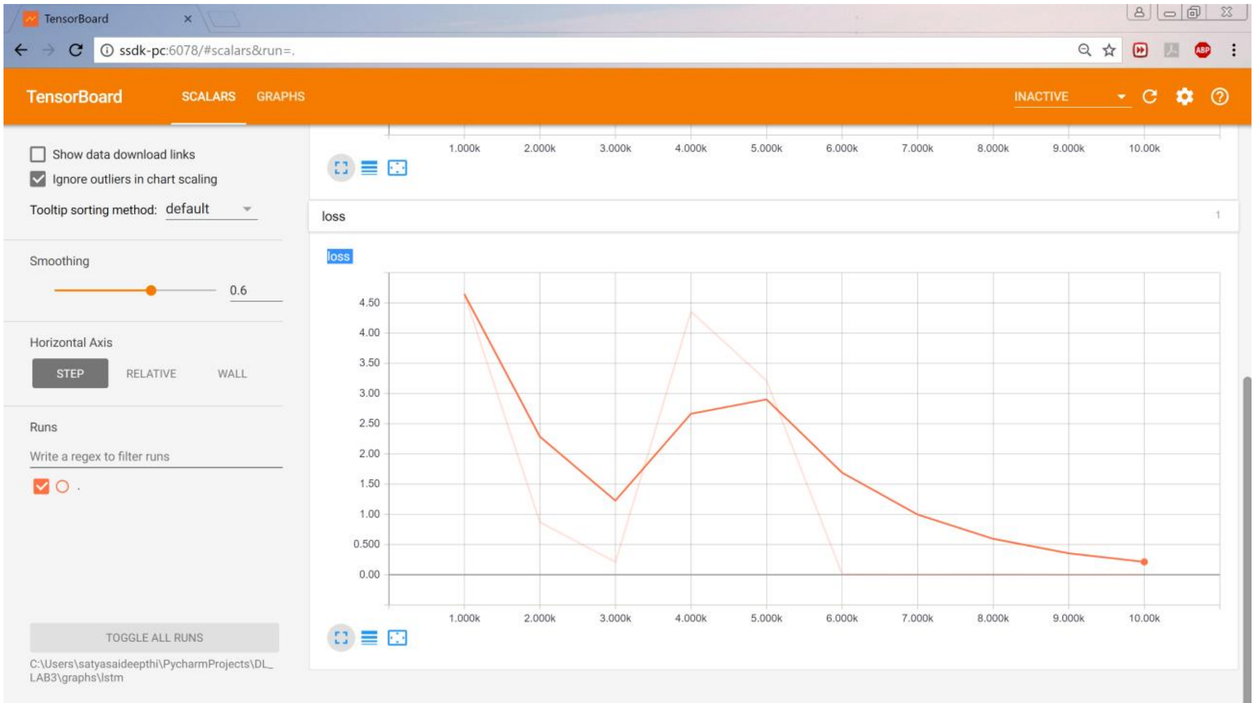

TensorBoard
SCALARS
GRAPHS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
C:\Users\satyasaideepthi\PycharmProjects\DL_LAB3\graphs\rnn
Filter tags (regular expressions supported)
accuracy
accuracy
loss
loss


TensorBoard
SCALARS
GRAPHS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
C:\Users\satyasaideepthi\PycharmProjects\DL_LAB3\graphs\rnn
Filter tags (regular expressions supported)
accuracy
loss
loss

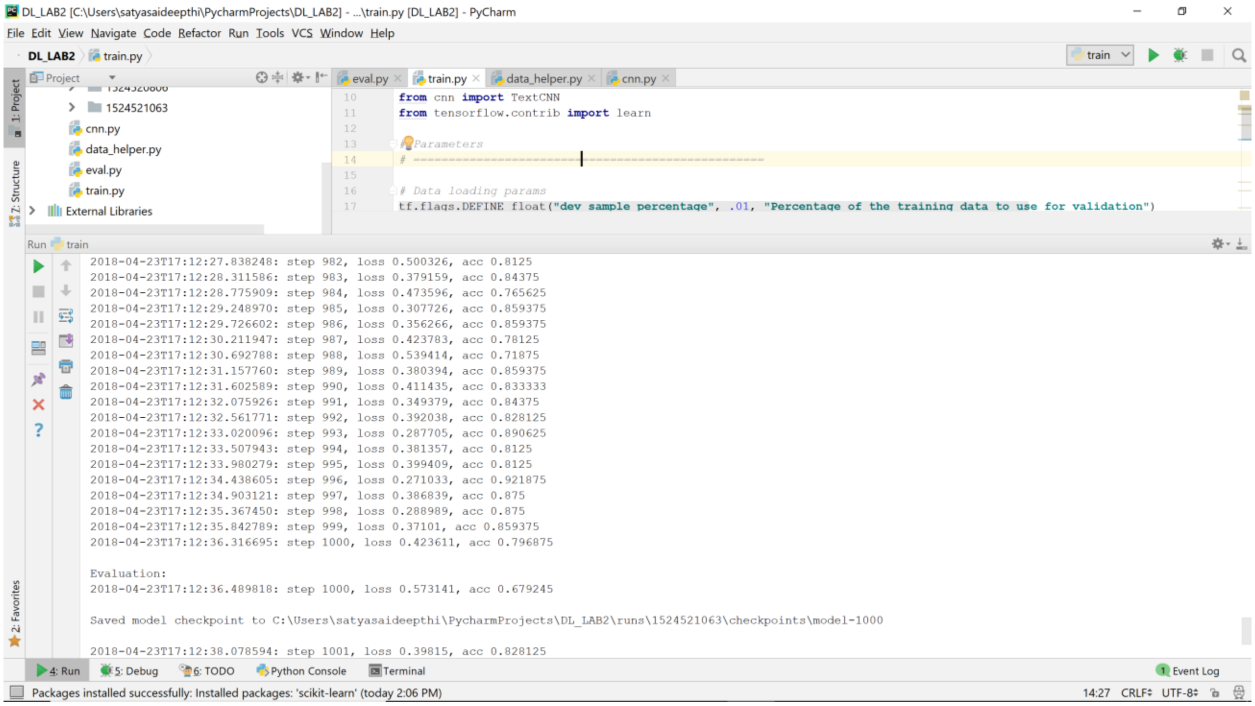
LSTM

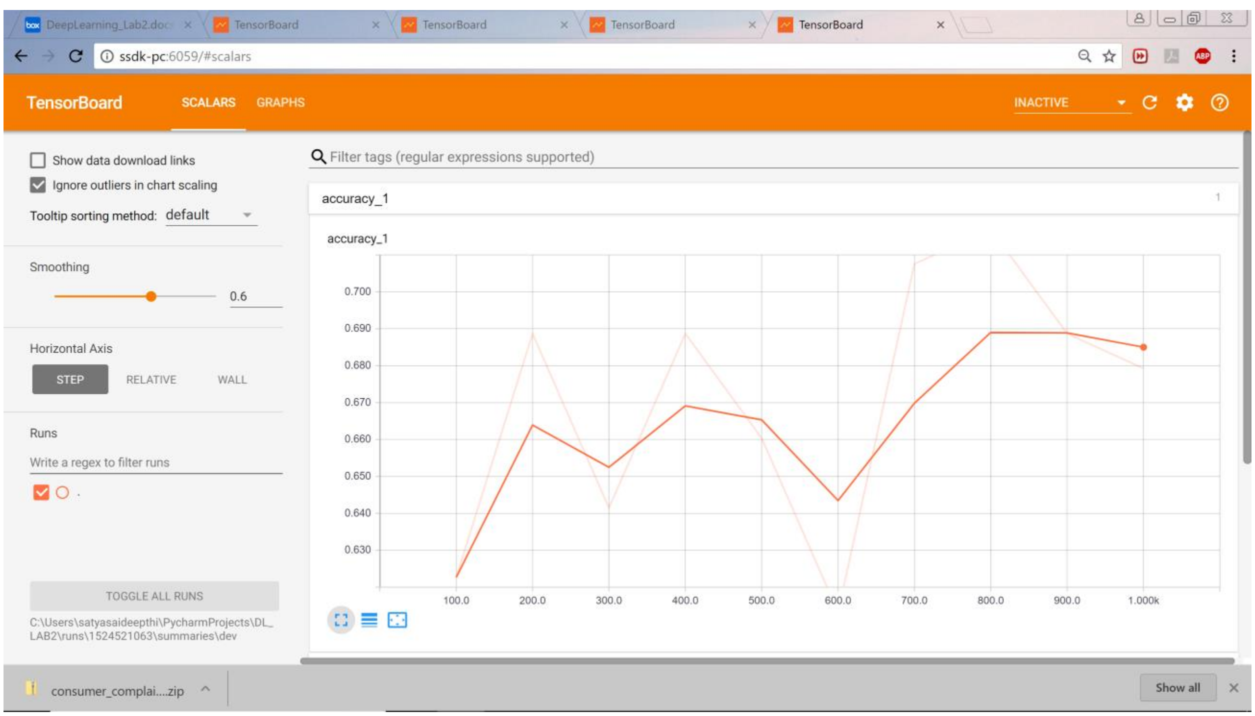

DL_LAB3 [C:\Users\satyasaideepthi\PycharmProjects\DL_LAB3] - ...\Lstm.py [DL_LAB3] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
DL_LAB3 Lstm.py
Lstm.py google.txt cnn-eval.py cnn-train.py data_helpers.py cnn.py rnn.py
DL_LAB3 C:\Users\satyasaideepthi\PycharmProjects\
graphs
lstm
events.out.tfevents.1525912930.SSDK-PC
rnn
cnn.py
cnn-eval.py
cnn-train.py
Launch the graph
with tf.Session() as session:
session.run(init)
writer = tf.summary.FileWriter('./graphs/lstm', session.graph)
step = 0
offset = random.randint(0, n_input + 1)
end_offset = n_input + 1
Run: Lstm rnn
Iter= 1000, Average Loss= 5.354698, Average Accuracy= 7.70%
['Brin', 'while', 'they', 'were', 'Ph.D.', 'students', 'at', 'Stanford', 'University,', 'California.'] - [Together,] vs [and]
Iter= 2000, Average Loss= 3.800004, Average Accuracy= 22.30%
['online', 'advertising', 'technologies,', 'search', 'engine,', 'cloud', 'computing,', 'software,', 'and', 'hardware.'] - [Google] vs [Google]
Iter= 3000, Average Loss= 2.979286, Average Accuracy= 36.20%
['founded', 'in', '1998', 'by', 'Larry', 'Page', 'and', 'Sergey', 'Brin', 'while'] - [they] vs [they]
Iter= 4000, Average Loss= 2.504633, Average Accuracy= 43.90%
['Page', 'and', 'Sergey', 'Brin', 'while', 'they', 'were', 'Ph.D.', 'students', 'at'] - [Stanford] vs [Ph.D.]
Iter= 5000, Average Loss= 1.305629, Average Accuracy= 67.40%
['while', 'they', 'were', 'Ph.D.', 'students', 'at', 'Stanford', 'University,', 'California.', 'Together,'] - [they] vs [Stanford]
Iter= 6000, Average Loss= 0.964304, Average Accuracy= 74.90%
['nicknamed', 'the', 'Googleplex.', 'In', 'August', '2015,', 'Google', 'announced', 'plans', 'to'] - [reorganize] vs [reorganize]
Iter= 7000, Average Loss= 0.573033, Average Accuracy= 82.90%
['Stanford', 'University,', 'California.', 'Together,', 'they', 'own', 'about', '14', 'percent', 'of'] - [its] vs [its]
Iter= 8000, Average Loss= 0.388760, Average Accuracy= 89.40%
['Google', 'announced', 'plans', 'to', 'reorganize', 'its', 'various', 'interests', 'as', 'a'] - [conglomerate] vs [conglomerate]
Iter= 9000, Average Loss= 0.281824, Average Accuracy= 92.70%
['stock.', 'They', 'incorporated', 'Google', 'as', 'a', 'privately', 'held', 'company', 'on'] - [September] vs [September]
Iter= 10000, Average Loss= 0.207261, Average Accuracy= 93.30%
['on', 'August', '19,', '2004,', 'and', 'Google', 'moved', 'to', 'its', 'new'] - [headquarters] vs [headquarters]
Optimization Finished!
Elapsed time: 26.044257879257202 sec
Run on command line.
10 words: Google is an american multinational technology company that specializes in
Python Console Terminal 3: Find 4: Run 6: TODO Event Log
IDE and Plugin Updates: PyCharm is ready to update. (today 12:50 PM)
21:1 CRLF UTF-8

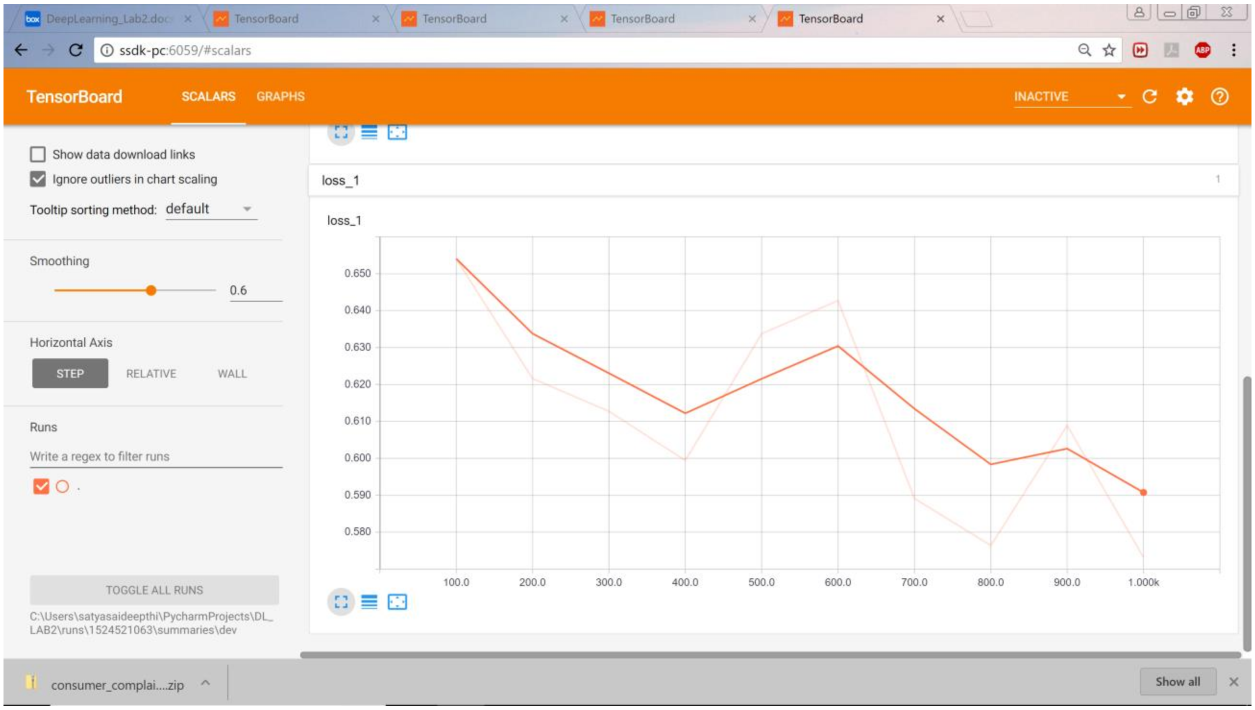

TensorBoard
ssdk-pc:6078/#scalars&run=.
TensorBoard SCALARS GRAPHS INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP RELATIVE WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
C:\Users\satyasaideepthi\PycharmProjects\DL_LAB3\graphs\lstm
Filter tags (regular expressions supported)
accuracy
accuracy
1.00
0.900
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.00
1.000k
2.000k
3.000k
4.000k
5.000k
6.000k
7.000k
8.000k
9.000k
10.00k
loss


TensorBoard
ssdk-pc:6078/#scalars&run=.
TensorBoard
SCALARS
GRAPHS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
C:\Users\satyasaideepthi\PycharmProjects\DL_LAB3\graphs\lstm
loss
loss
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.500
0.00
1.000k
2.000k
3.000k
4.000k
5.000k
6.000k
7.000k
8.000k
9.000k
10.00k

CNN


DL_LAB2 [C:\Users\satyasaideepthi\PycharmProjects\DL_LAB2] - ...\train.py [DL_LAB2] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
DL_LAB2 train.py
train
Project
1524521063
cnn.py
data_helper.py
eval.py
train.py
External Libraries
eval.py train.py data_helper.py cnn.py
from cnn import TextCNN
from tensorflow.contrib import learn
#Parameters
# Data loading params
tf.flags.DEFINE_float("dev_sample_percentage", .01, "Percentage of the training data to use for validation")
Run train
2018-04-23T17:12:27.838248: step 982, loss 0.500326, acc 0.8125
2018-04-23T17:12:28.311586: step 983, loss 0.379159, acc 0.84375
2018-04-23T17:12:28.775909: step 984, loss 0.473596, acc 0.765625
2018-04-23T17:12:29.248970: step 985, loss 0.307726, acc 0.859375
2018-04-23T17:12:29.726602: step 986, loss 0.356266, acc 0.859375
2018-04-23T17:12:30.211947: step 987, loss 0.423783, acc 0.78125
2018-04-23T17:12:30.692788: step 988, loss 0.539414, acc 0.71875
2018-04-23T17:12:31.157760: step 989, loss 0.380394, acc 0.859375
2018-04-23T17:12:31.602589: step 990, loss 0.411435, acc 0.833333
2018-04-23T17:12:32.075926: step 991, loss 0.349379, acc 0.84375
2018-04-23T17:12:32.561771: step 992, loss 0.392038, acc 0.828125
2018-04-23T17:12:33.020096: step 993, loss 0.287705, acc 0.890625
2018-04-23T17:12:33.507943: step 994, loss 0.381357, acc 0.8125
2018-04-23T17:12:33.980279: step 995, loss 0.399409, acc 0.8125
2018-04-23T17:12:34.438605: step 996, loss 0.271033, acc 0.921875
2018-04-23T17:12:34.903121: step 997, loss 0.386839, acc 0.875
2018-04-23T17:12:35.367450: step 998, loss 0.288989, acc 0.875
2018-04-23T17:12:35.842789: step 999, loss 0.37101, acc 0.859375
2018-04-23T17:12:36.316695: step 1000, loss 0.423611, acc 0.796875
Evaluation:
2018-04-23T17:12:36.489818: step 1000, loss 0.573141, acc 0.679245
Saved model checkpoint to C:\Users\satyasaideepthi\PycharmProjects\DL_LAB2\runs\1524521063\checkpoints\model-1000
2018-04-23T17:12:38.078594: step 1001, loss 0.39815, acc 0.828125
4: Run 5: Debug 6: TODO Python Console Terminal Event Log
Packages installed successfully: Installed packages: 'scikit-learn' (today 2:06 PM)
14:27 CRLF UTF-8


DeepLearning_Lab2.docx TensorBoard TensorBoard TensorBoard TensorBoard
ssdk-pc:6059/#scalars
TensorBoard SCALARS GRAPHS INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP RELATIVE WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
C:\Users\satyasaideepthi\PycharmProjects\DL_LAB2\runs\1524521063\summaries\dev
Filter tags (regular expressions supported)
accuracy_1
accuracy_1
0.700
0.690
0.680
0.670
0.660
0.650
0.640
0.630
100.0 200.0 300.0 400.0 500.0 600.0 700.0 800.0 900.0 1.000k
consumer_complai....zip
Show all



CONCLUSION:

We observe that the text is short and sentimental analysis is less but the text considered is about the company, RNN gives best results. When the text classification is based on the feature detection like giving sentimental analysis example angry, sadness etc CNN gives more accuracy than recurrent networks.