Introduction

Implementing the Text Classification with CNN model with new dataset and displaying the graphs in the tensor board.

Objectives

To perform the text classification with Convolutional Neural Networks model and display the results in tensor board.

Approaches/Methods

In the beginning, I choose the consumer complaints as the input file. Here we will develop a model of multilayers where the layers perform the convolution of embedded words then the convolution layer is converted into a long feature vector. Then the feature vector is used from the max pooling by solving the matrix multiplication and choose the class with more score.

Workflow

1-> Select the dataset

2-> Import the dataset

3-> Convert the characters to strings

4-> Collect the most frequent words and build the vocabulary and store every word as key value pair

5-> Build the model and loss function which is optimized by using the Gradient Descent Optimizer

6-> Train the model and plot the graph on tensor board

Dataset

Consumer complaints Dataset

Parameters

- Vocabulary size
- Number of classes
- Embedding size
- Filter sizes
- Number of filters
- R2 lambda value and classify the model into eleven classes

Evaluation

```
import os
import sys
import json
import time
import logging
import dataHelpers
import numpy as np
import tensorflow as tf
from textCNN import TextCNN
from tensorflow.contrib import learn
from sklearn.model_selection import train_test_split

logging.getLogger().setLevel(logging.INFO)

def train_cnn():
    """Step 0: load sentences, labels, and training parameters"""
    #train_file = sys.argv[1]
    input_file = 'C:\Users\sai smaran chinthala\Desktop\Spring-2018\Python\DL\CNN\consumer_complaints.csv.zip'
    x_raw, y_raw, df, labels = dataHelpers.load_data_and_labels(input_file)

    #parameter_file = sys.argv[0]
    #paramater_file='C:/Users/sai smaran chinthala/Desktop/New folder/parameters.json'
    # Model Hyper parameters
    tf.flags.DEFINE_integer("embedding_dim", 40, "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", 32, "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 ", 1, "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")

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

    #params = json.loads(open(parameter_file).read())

    """Step 1: pad each sentence to the same length and map each word to an id"""
    max_document_length = max([len(x.split(' ')) for x in x_raw])
    logging.info('The maximum length of all sentences: {}'.format(max_document_length))
    vocab_processor = learn.preprocessing.VocabularyProcessor(max_document_length)
    x = np.array(list(vocab_processor.fit_transform(x_raw)))
    y = np.array(y_raw)

    """Step 2: split the original dataset into train and test sets"""
    x_, x_test, y_, y_test = train_test_split(x, y, test_size=0.1, random_state=42)



    """Step 3: shuffle the train set and split the train set into train and dev sets"""
    shuffle_indices = np.random.permutation(np.arange(len(y_)))
    x_shuffled = x_[shuffle_indices]
    y_shuffled = y_[shuffle_indices]
    x_train, x_dev, y_train, y_dev = train_test_split(x_shuffled, y_shuffled, test_size=0.1)

    """Step 4: save the labels into labels.json since predict.py needs it"""
    #with open('./labels.json', 'w') as outfile:
    #    json.dump(labels, outfile, indent=4)

    logging.info('x_train: {}, x_dev: {}, x_test: {}'.format(len(x_train), len(x_dev), len(x_test)))
    logging.info('y_train: {}, y_dev: {}, y_test: {}'.format(len(y_train), len(y_dev), len(y_test)))

    """Step 5: build a graph and cnn object"""
    graph = tf.Graph()
    with graph.as_default():
        session_conf = tf.ConfigProto(allow_soft_placement=True, log_device_placement=False)
        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)
```

```
        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)

        timestamp = str(int(time.time()))
        out_dir = os.path.abspath(os.path.join(os.path.curdir, "trained_model_" + timestamp))

        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())

        # One training step: train the model with one batch
        def train_step(x_batch, y_batch):
            feed_dict = {
                cnn.input_x: x_batch,
                cnn.input_y: y_batch,
                cnn.dropout_keep_prob: FLAGS.dropout_keep_prob}
            _, step, loss, acc = sess.run([train_op, global_step, cnn.loss, cnn.accuracy], feed_dict)

        # One evaluation step: evaluate the model with one batch
        def dev_step(x_batch, y_batch):
            feed_dict = {cnn.input_x: x_batch, cnn.input_y: y_batch, cnn.dropout_keep_prob: 1.0}
            step, loss, acc, num_correct = sess.run([global_step, cnn.loss, cnn.accuracy, cnn.num_correct], feed_dict
            return num_correct


        # Save the word_to_id map since predict.py needs it
        vocab_processor.save(os.path.join(out_dir, "vocab.pickle"))
        #sess.run(tf.initialize_all_variables())
        sess.run(tf.global_variables_initializer())

        # Training starts here
        train_batches = dataHelpers.batch_iter(list(zip(x_train, y_train)),FLAGS.batch_size,FLAGS.num_epochs)
        best_accuracy, best_at_step = 0, 0

        """Step 6: train the cnn model with x_train and y_train (batch by batch)"""
        for train_batch in train_batches:
            x_train_batch, y_train_batch = zip(*train_batch)
            train_step(x_train_batch, y_train_batch)
            current_step = tf.train.global_step(sess, global_step)

            """Step 6.1: evaluate the model with x_dev and y_dev (batch by batch)"""
            if current_step % FLAGS.evaluate_every == 0:
                dev_batches = dataHelpers.batch_iter(list(zip(x_dev, y_dev)), FLAGS.batch_size, 1)
                total_dev_correct = 0
                for dev_batch in dev_batches:
                    x_dev_batch, y_dev_batch = zip(*dev_batch)
                    num_dev_correct = dev_step(x_dev_batch, y_dev_batch)
                    total_dev_correct += num_dev_correct

                dev_accuracy = float(total_dev_correct) / len(y_dev)
                logging.critical('Accuracy on dev set: {}'.format(dev_accuracy))
```

```
                """Step 6.2: save the model if it is the best based on accuracy on dev set"""
                if dev_accuracy >= best_accuracy:
                    best_accuracy, best_at_step = dev_accuracy, current_step
                    path = saver.save(sess, checkpoint_prefix, global_step=current_step)
                    logging.critical('Saved model at {} at step {}'.format(path, best_at_step))
                    logging.critical('Best accuracy is {} at step {}'.format(best_accuracy, best_at_step))

        """Step 7: predict x_test (batch by batch)"""
        test_batches = dataHelpers.batch_iter(list(zip(x_test, y_test)),FLAGS.batch_size, 1)
        total_test_correct = 0
        for test_batch in test_batches:
            x_test_batch, y_test_batch = zip(*test_batch)
            num_test_correct = dev_step(x_test_batch, y_test_batch)
            total_test_correct += num_test_correct

        test_accuracy = float(total_test_correct) / len(y_test)
        logging.critical('Accuracy on test set is {} based on the best model {}'.format(test_accuracy, path))
        logging.critical('The training is complete')

if __name__ == '__main__':
    # python3 train
    train_cnn()
```

Predicting the CNN:

```
import os
import sys
import json
import logging
import dataHelpers
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.contrib import learn

logging.getLogger().setLevel(logging.INFO)

def predict_unseen_data():
    """Step 0: load trained model and parameters"""
    params = json.loads(open('./parameters.json').read())
    checkpoint_dir = 'C:\Users\sai smaran chinthala\Desktop\Spring-2018\Python\DL\CN/trained_model_1510877046'
    if not checkpoint_dir.endswith('/'):
        checkpoint_dir += '/'
    checkpoint_file = tf.train.latest_checkpoint(checkpoint_dir + 'checkpoints')
    logging.critical('Loaded the trained model: {}'.format(checkpoint_file))

    """Step 1: load data for prediction"""
    test_file = '/Users/bhavyateja/Masters/PythonDeepLearning/DL_4/New folder/small_samples.json'
    test_examples = json.loads(open(test_file).read())

    # labels.json was saved during training, and it has to be loaded during prediction
    labels = json.loads(open('./labels.json').read())
    one_hot = np.zeros((len(labels), len(labels)), int)
    np.fill_diagonal(one_hot, 1)
    label_dict = dict(zip(labels, one_hot))
```

```
    x_raw = [example['consumer_complaint_narrative'] for example in test_examples]
    x_test = [dataHelpers.clean_str(x) for x in x_raw]
    logging.info('The number of x_test: {}'.format(len(x_test)))

    y_test = None
    if 'product' in test_examples[0]:
        y_raw = [example['product'] for example in test_examples]
        y_test = [label_dict[y] for y in y_raw]
        logging.info('The number of y_test: {}'.format(len(y_test)))

    vocab_path = os.path.join(checkpoint_dir, "vocab.pickle")
    vocab_processor = learn.preprocessing.VocabularyProcessor.restore(vocab_path)
    x_test = np.array(list(vocab_processor.transform(x_test)))

    """Step 2: compute the predictions"""
    graph = tf.Graph()
    with graph.as_default():
        session_conf = tf.ConfigProto(allow_soft_placement=True, log_device_placement=False)
        sess = tf.Session(config=session_conf)

        with sess.as_default():
            saver = tf.train.import_meta_graph("{}.meta".format(checkpoint_file))
            saver.restore(sess, checkpoint_file)

            input_x = graph.get_operation_by_name("input_x").outputs[0]
            dropout_keep_prob = graph.get_operation_by_name("dropout_keep_prob").outputs[0]
            predictions = graph.get_operation_by_name("output/predictions").outputs[0]


            batches = dataHelpers.batch_iter(list(x_test), params['batch_size'], 1, shuffle=False)
            all_predictions = []
            for x_test_batch in batches:
                batch_predictions = sess.run(predictions, {input_x: x_test_batch, dropout_keep_prob: 1.0})
                all_predictions = np.concatenate([all_predictions, batch_predictions])

    if y_test is not None:
        y_test = np.argmax(y_test, axis=1)
        correct_predictions = sum(all_predictions == y_test)
        logging.critical('The accuracy is: {}'.format(correct_predictions / float(len(y_test))))
        logging.critical('The prediction is complete')

if __name__ == '__main__':

    predict_unseen_data()
```

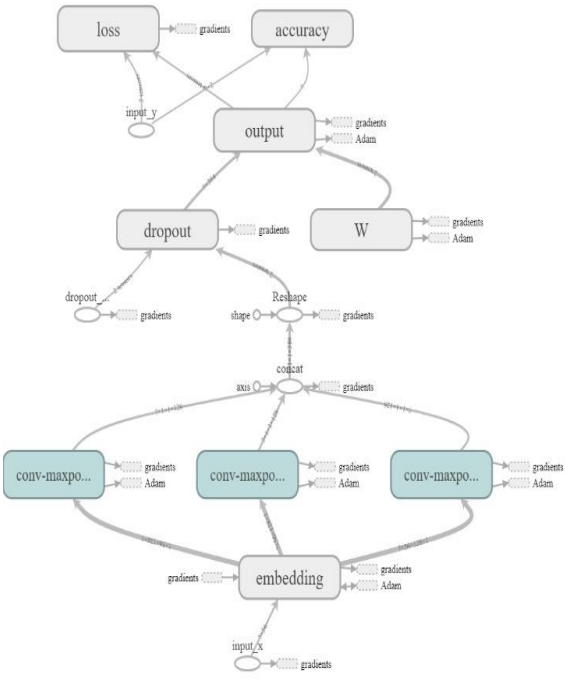
DataHelpers:

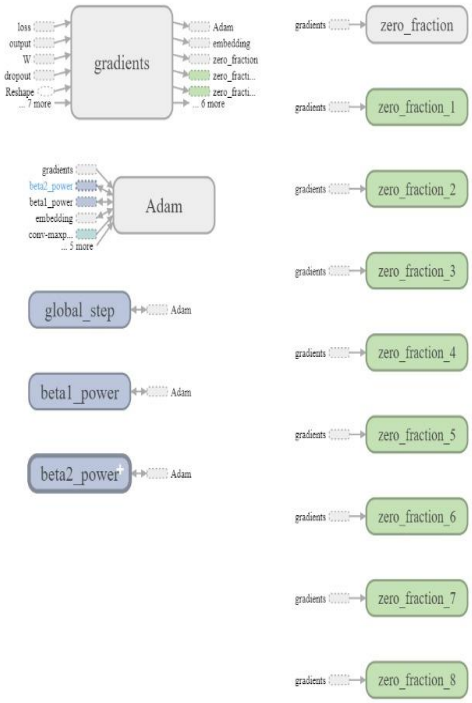
```
import re
import logging
import numpy as np
import pandas as pd
from collections import Counter

def clean_str(s):
    """Clean sentence"""
    s = re.sub(r"[^A-Za-z0-9(),!?\'\`]", " ", s)
    s = re.sub(r"\'s", " \'s", s)
    s = re.sub(r"\'ve", " \'ve", s)
    s = re.sub(r"n\'t", " n\'t", s)
    s = re.sub(r"\'re", " \'re", s)
    s = re.sub(r"\'d", " \'d", s)
    s = re.sub(r"\'ll", " \'ll", s)
    s = re.sub(r",", " , ", s)
    s = re.sub(r"!", " ! ", s)
    s = re.sub(r"\(", " \( ", s)
    s = re.sub(r"\)", " \) ", s)
    s = re.sub(r"\?", " \? ", s)
    s = re.sub(r"\s{2,}", " ", s)
    s = re.sub(r'\S*(x{2,}|X{2,})\S*',"xxx", s)
    s = re.sub(r'[^\x00-\x7F]+', "", s)
    return s.strip().lower()

def load_data_and_labels(filename):
    """Load sentences and labels"""
    df = pd.read_csv(filename, compression='zip', dtype={'consumer_complaint_narrative': object})
    selected = ['product', 'consumer_complaint_narrative']
    non_selected = list(set(df.columns) - set(selected))


    df = df.drop(non_selected, axis=1) # Drop non selected columns
    df = df.dropna(axis=0, how='any', subset=selected) # Drop null rows
    df = df.reindex(np.random.permutation(df.index)) # Shuffle the dataframe

    # Map the actual labels to one hot labels
    labels = sorted(list(set(df[selected[0]].tolist())))
    one_hot = np.zeros((len(labels), len(labels)), int)
    np.fill_diagonal(one_hot, 1)
    label_dict = dict(zip(labels, one_hot))

    x_raw = df[selected[1]].apply(lambda x: clean_str(x)).tolist()
    y_raw = df[selected[0]].apply(lambda y: label_dict[y]).tolist()
    return x_raw, y_raw, df, labels

def batch_iter(data, batch_size, num_epochs, shuffle=True):
    """Iterate the data batch by batch"""
    data = np.array(data)
    data_size = len(data)
    num_batches_per_epoch = int(data_size / batch_size) + 1

    for epoch in range(num_epochs):
        if shuffle:
            shuffle_indices = np.random.permutation(np.arange(data_size))
            shuffled_data = data[shuffle_indices]
        else:
            shuffled_data = data
```

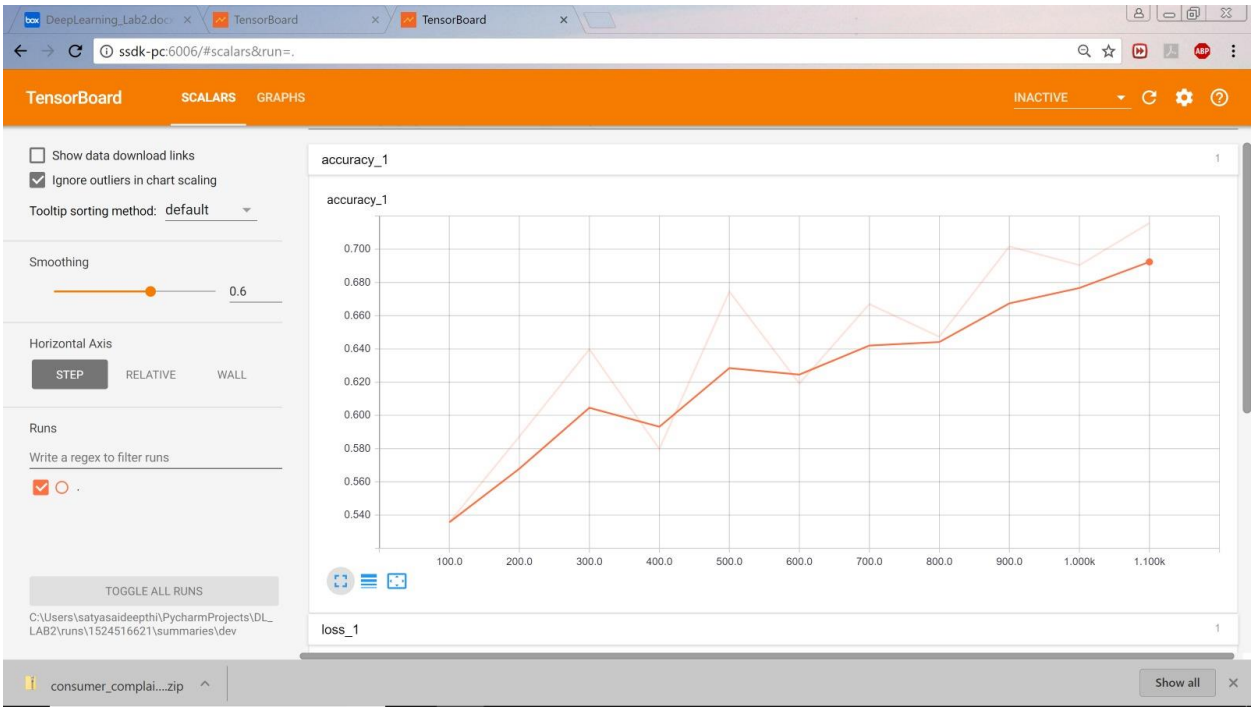
```
        for batch_num in range(num_batches_per_epoch):
            start_index = batch_num * batch_size
            end_index = min((batch_num + 1) * batch_size, data_size)
            yield shuffled_data[start_index:end_index]

if __name__ == '__main__':
    input_file = 'C:\Users\sai smaran chinthala\Desktop\Spring-2018\Python\DL\CNN\consumer_complaints.csv.zip'
    load_data_and_labels(input_file)


import numpy as np
import tensorflow as tf

class TextCNN(object):
    def __init__(self, sequence_length, num_classes, vocab_size, embedding_size, filter_sizes, num_filters, l2_reg_lam
        # Placeholders for input, output and dropout
        self.input_x = tf.placeholder(tf.int32, [None, sequence_length], name='input_x')
        self.input_y = tf.placeholder(tf.float32, [None, num_classes], name='input_y')
        self.dropout_keep_prob = tf.placeholder(tf.float32, name='dropout_keep_prob')

        # Keeping track of l2 regularization loss (optional)
        l2_loss = tf.constant(0.0)

        # Embedding layer
        with tf.device('/cpu:0'), tf.name_scope('embedding'):
            W = tf.Variable(tf.random_uniform([vocab_size, embedding_size], -1.0, 1.0), name='W')
            self.embedded_chars = tf.nn.embedding_lookup(W, self.input_x)
            self.embedded_chars_expanded = tf.expand_dims(self.embedded_chars, -1)

        # Create a convolution + maxpool layer for each filter size
        pooled_outputs = []
        for i, filter_size in enumerate(filter_sizes):
            with tf.name_scope('conv-maxpool-%s' % filter_size):
                # Convolution Layer
                filter_shape = [filter_size, embedding_size, 1, num_filters]
                W = tf.Variable(tf.truncated_normal(filter_shape, stddev=0.1), name='W')
                b = tf.Variable(tf.constant(0.1, shape=[num_filters]), name='b')
                conv = tf.nn.conv2d(
                    self.embedded_chars_expanded,
                    W,
                    strides=[1, 1, 1, 1],
                    padding='VALID',
                    name='conv')

                # Apply nonlinearity
                h = tf.nn.relu(tf.nn.bias_add(conv, b), name='relu')

                # Maxpooling over the outputs
                pooled = tf.nn.max_pool(
                    h,
                    ksize=[1, sequence_length - filter_size + 1, 1, 1],
                    strides=[1, 1, 1, 1],
                    padding='VALID',
                    name='pool')
                pooled_outputs.append(pooled)

        # Combine all the pooled features
        num_filters_total = num_filters * len(filter_sizes)
        self.h_pool = tf.concat(pooled_outputs,3)
        self.h_pool_flat = tf.reshape(self.h_pool, [-1, num_filters_total])

        # Add dropout
        with tf.name_scope('dropout'):
            self.h_drop = tf.nn.dropout(self.h_pool_flat, self.dropout_keep_prob)
```

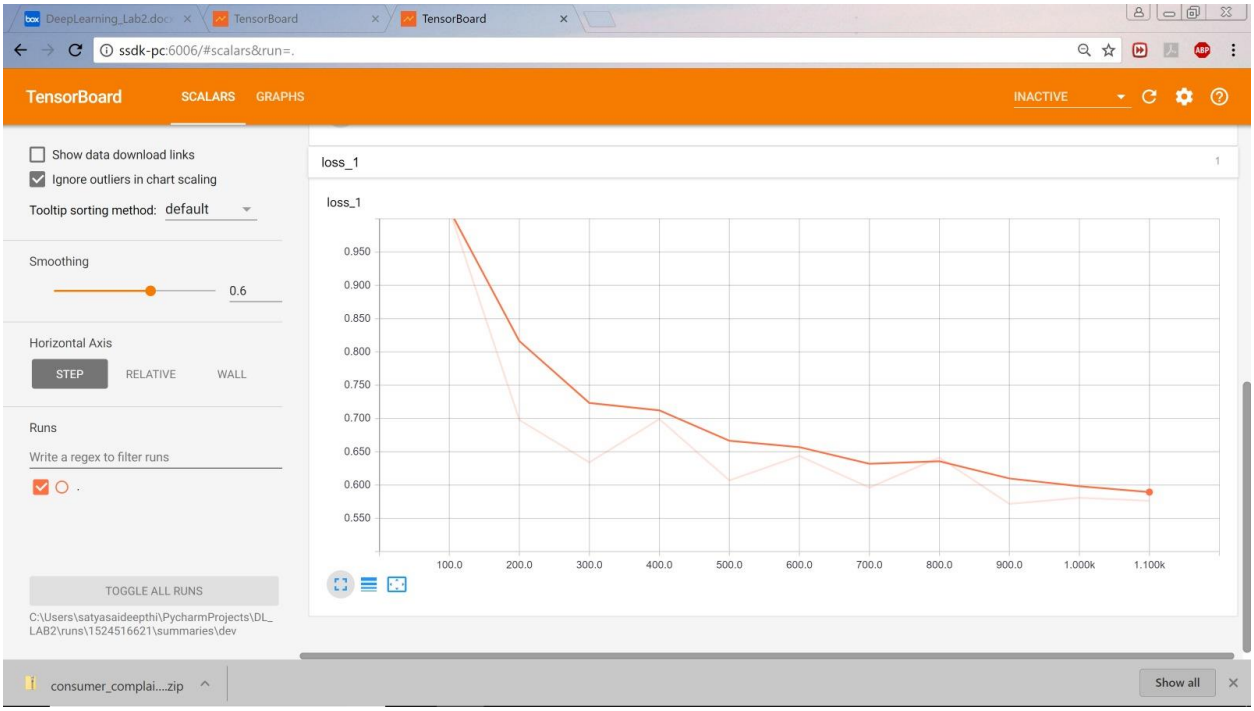
```
        # Final (unnormalized) scores and predictions
        with tf.name_scope('output'):
            W = tf.get_variable(
                'W',
                shape=[num_filters_total, num_classes],
                initializer=tf.contrib.layers.xavier_initializer())
            b = tf.Variable(tf.constant(0.1, shape=[num_classes]), name='b')
            l2_loss += tf.nn.l2_loss(W)
            l2_loss += tf.nn.l2_loss(b)
            self.scores = tf.nn.xw_plus_b(self.h_drop, W, b, name='scores')
            self.predictions = tf.argmax(self.scores, 1, name='predictions')

        # Calculate mean cross-entropy loss
        with tf.name_scope('loss'):
            losses = tf.nn.softmax_cross_entropy_with_logits(labels = self.input_y, logits = self.scores) #  only named
            self.loss = tf.reduce_mean(losses) + l2_reg_lambda * l2_loss

        # Accuracy
        with tf.name_scope('accuracy'):
            correct_predictions = tf.equal(self.predictions, tf.argmax(self.input_y, 1))
            self.accuracy = tf.reduce_mean(tf.cast(correct_predictions, 'float'), name='accuracy')

        with tf.name_scope('num_correct'):
            correct_predictions = tf.equal(self.predictions, tf.argmax(self.input_y, 1))
            self.num_correct = tf.reduce_sum(tf.cast(correct_predictions, 'float'), name='num_correct')
```

Graph in Tensor Board:






DeepLearning_Lab2.doc
TensorBoard
TensorBoard
ssdk-pc:6006/#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_LAB2\runs\1524516621\summaries\dev
accuracy_1
accuracy_1
0.700
0.680
0.660
0.640
0.620
0.600
0.580
0.560
0.540
100.0
200.0
300.0
400.0
500.0
600.0
700.0
800.0
900.0
1.000k
1.100k
loss_1
consumer_complai....zip
Show all


DeepLearning_Lab2.doc
TensorBoard
TensorBoard
ssdk-pc:6006/#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_LAB2\runs\1524516621\summaries\dev
loss_1
loss_1
0.950
0.900
0.850
0.800
0.750
0.700
0.650
0.600
0.550
100.0
200.0
300.0
400.0
500.0
600.0
700.0
800.0
900.0
1.000k
1.100k
consumer_complai....zip
Show all

Output:

```
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
```

Conclusion

We conclude that metrices are not smooth because we have used small batch sizes for training