CSEE 5590 0001
Special Topics SPRING 2018

DEEP LEARNING
LAB ASSIGNMENT - 2

Submitted On 4/23/2018

Name: Satya Sai Deepthi Katta

Class ID: 21

UMKC

Department of Computer Science and Electrical Engineering

DOCUMENT CONTENTS

Introduction

Text classification is the classification of text documents into certain categories using machine learning to automate the tasks which makes whole process fast and efficient. The CNN model i.e.; convolutional neural networks gives good classification performance and also establishes a standard baseline for the new text architectures.

Objective

The objective behind the lab assignment is to create exposure to machine learning concepts listed below:

- Text classification
- Convolutional Neural Networks
- Generating work flow graph in Tensor Board.
- Changing random hyper parameters like learning rate to observe the results.

Approach

The approach for the assignment can be defined as simple steps given below:

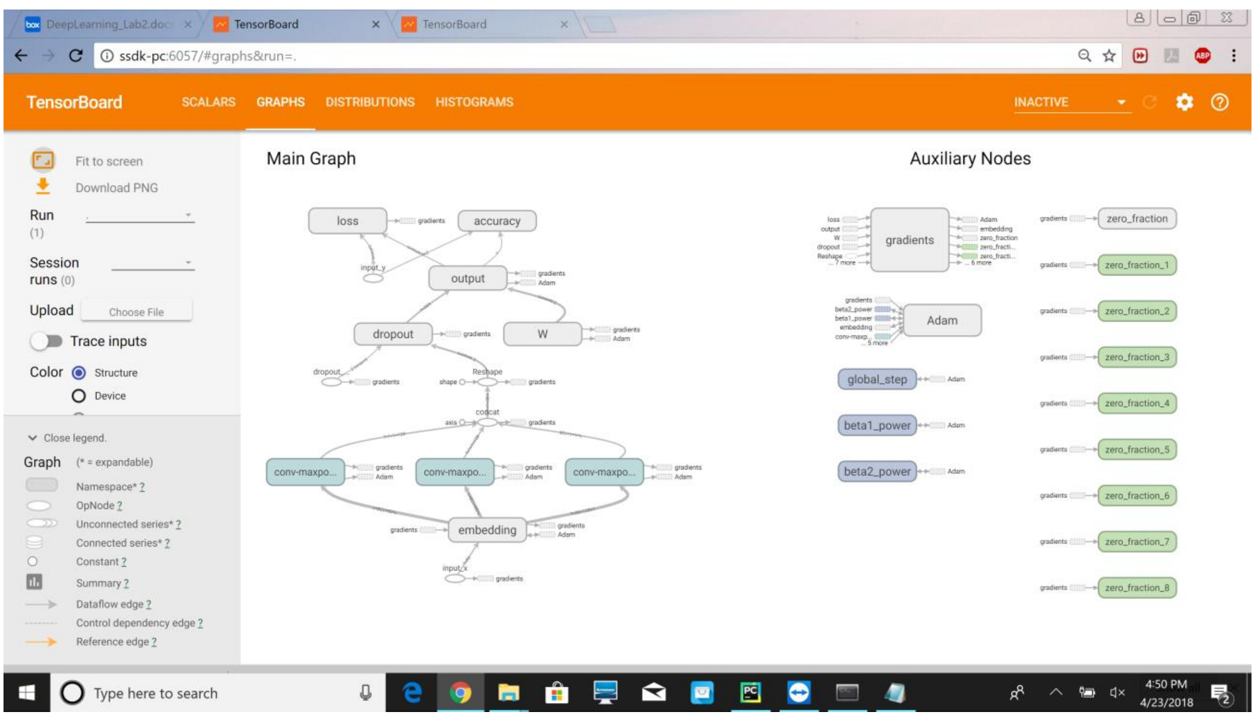
- Importing Data from Dataset
- Assigning X and Y Placeholders
- Variable Weights and Bias Collection
- Construction of prediction model
- Optimize model for less errors
- Train model for the training data
- Compare the prediction and actual model variables
- Compute the accuracy of the model
- Change hyper parameters to get the results

Parameters

Below are the parameters set for the assignment:
ALLOW_SOFT_PLACEMENT=True
BATCH_SIZE=64
CHECKPOINT_EVERY=100
DEV_SAMPLE_PERCENTAGE=0.1
DROPOUT_KEEP_PROB=0.5
EMBEDDING_DIM=128
EVALUATE_EVERY=100
FILTER_SIZES=3,4,5
L2_REG_LAMBDA=0.0
LOG_DEVICE_PLACEMENT=False
NUM_CHECKPOINTS=5
NUM_EPOCHS=200
NUM_FILTERS=128

Workflow

The workflow for the text classification of Convolutional neural networks performed on Customer Finance Complaints Dataset is shown in Tensor Board.



Dataset

The dataset used is ***Kaggle Consumer Finance Complaint Dataset***.

The dataset contains of complaints given by everyone represented in rows.

The CSV file contains 17 columns listing the reasons for the complaints from about 5 lakh customers to the Financial sector.

The figure below shows the preview of the dataset obtained from the website.

Preview (first 100 rows) Column Metadata

date_received	product	sub_product	issue	sub_issue	cc
08/30/2013	Mortgage	Other mortgage	Loan modification,collection,foreclosure		
08/30/2013	Mortgage	Other mortgage	Loan servicing, payments, escrow account		
08/30/2013	Credit reporting		Incorrect information on credit report	Account status	
08/30/2013	Student loan	Non-federal student loan	Repaying your loan	Repaying your loan	
08/30/2013	Debt collection	Credit card	False statements or representation	Attempted to collect wrong amount	
08/30/2013	Credit card		Application processing delay		
08/30/2013	Credit card		Credit line increase/decrease		
08/30/2013	Bank account or service	Checking account	Deposits and withdrawals		
08/30/2013	Bank account or service	Checking account	Deposits and withdrawals		
09/17/2013	Mortgage	Conventional adjustable mortgage (ARM)	Loan modification,collection,foreclosure		

Configuration

For executing the tasks given in the lab assignment, advanced version of **Python 3.6.4** is used and the code is built in **PYCHARM** Software.

Evaluation & Discussion

The code snippets are provided below evaluating the performance of text classification on Kaggle Consumer Finance Complaint Dataset.

Cnn.py

CNN model class is created which defines the training model for the text classification. The model is set by initializing input X & Y using place holders.

```
import tensorflow as tf
import numpy as np


class TextCNN(object):
    """
    A CNN for text classification.
    Uses an embedding layer, followed by a convolutional, max-pooling and softmax layer.
    """
    def __init__(
      self, sequence_length, num_classes, vocab_size,
      embedding_size, filter_sizes, num_filters, l2_reg_lambda=0.0):

        # 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"):
            self.W = tf.Variable(
                tf.random_uniform([vocab_size, embedding_size], -1.0, 1.0),
                name="W")
            self.embedded_chars = tf.nn.embedding_lookup(self.W, self.input_x)
            self.embedded_chars_expanded = tf.expand_dims(self.embedded_chars, -1)
```

A convolution maxpool layer is created for each filter size. Non linearity is applied after the layer is described in the model.

```
# 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)
```

Combining all the filters, final predictions are made. Cross-entropy loss is created by taking the difference of actual values and predicted values. Accuracy is calculated using the argmax of all values.

```python
        # 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(logits=self.scores, labels=self.input_y)
            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")
```

The import statements are given and defining the parameters using flags Definition.

```python
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
import tensorflow as tf
import numpy as np
import os
import time
import datetime
import data_helper
from cnn import TextCNN
from tensorflow.contrib import learn
import pandas as pd

# Parameters
# ==================================================
# Data loading params
tf.flags.DEFINE_float("dev_sample_percentage", .1, "Percentage of the training data to use for validation")
input_file = pd.read_csv(r"C:\Users\satyasaideepthi\Downloads\consumer_complaints.csv.zip")
# 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")
```

Model is trained by shuffling the datasets and taking the sample indexes from the shuffled data.

```
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_helper.load_data_and_labels(FLAGS.positive_data_file, FLAGS.negative_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]

# 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:]
```

Graph sessions is created to get the tensor board workflow and the training procedure is defined by initializing global variable. Optimizers are used for reducing the loss.

```
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)
```

Accuracy and Loss are given to train and dev sections which give the summary of all accuracies in train section and average is given in dev section. Checkpoints are defined at each step after which takes all the meta data and vocab points.

```python
# 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)

# 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"))
```

```python
# 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)
```

Batches are generated for every 100 steps, evaluating at each checkpoint and storing the data at 100th step.

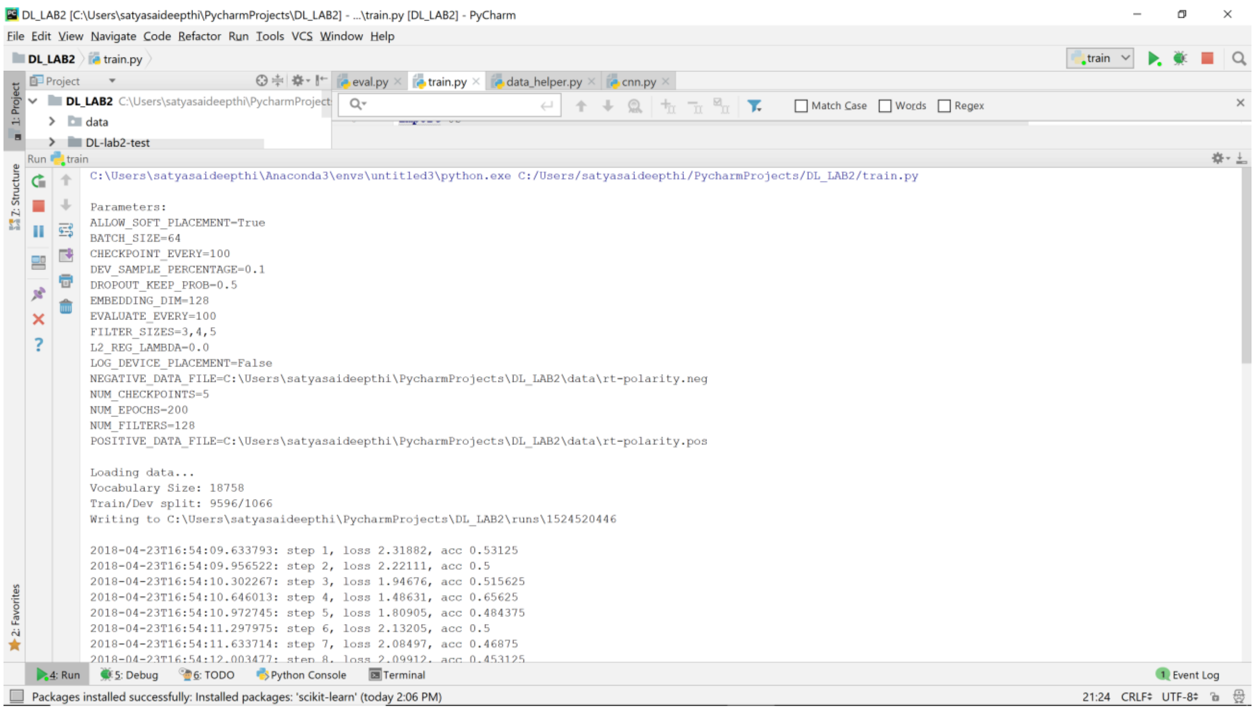
```python
# Generate batches
batches = data_helper.batch_iter(
    list(zip(x_train, y_train)), FLAGS.batch_size, FLAGS.num_epochs)
# Training loop. For each batch...
for batch in batches:
    x_batch, y_batch = zip(*batch)
    train_step(x_batch, y_batch)
    current_step = tf.train.global_step(sess, 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

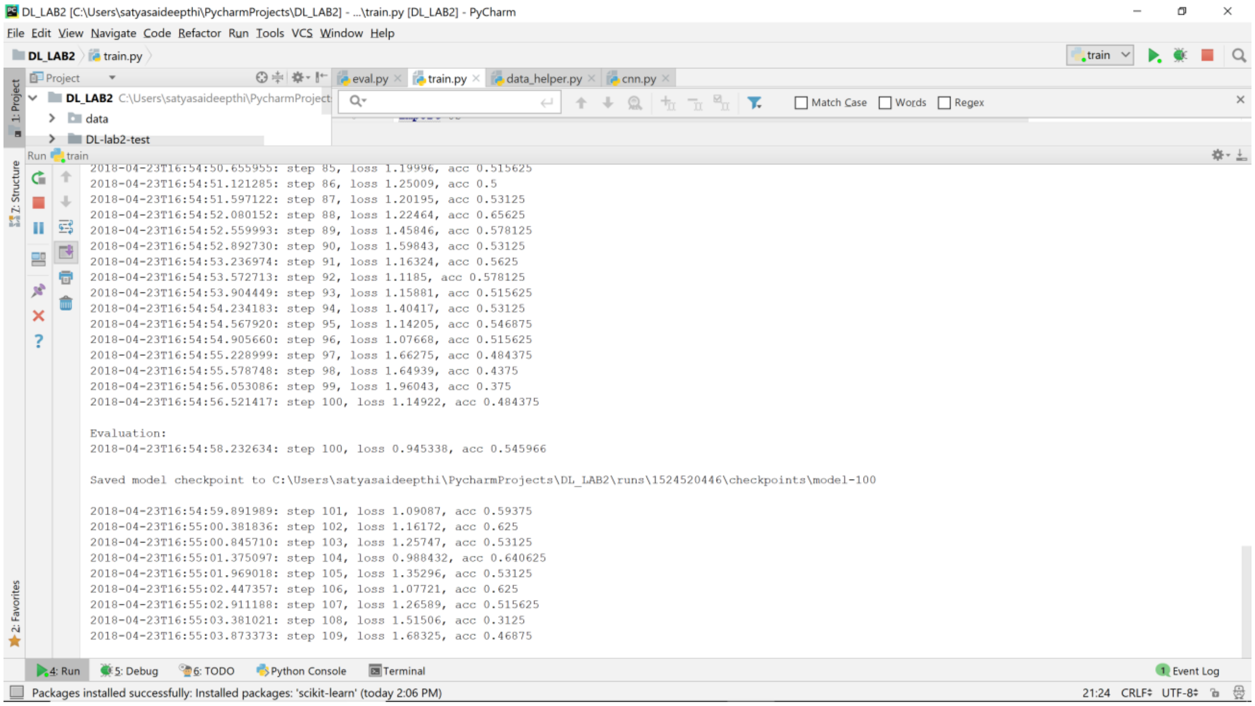
Changing the Hyper Parameter Learning Rate i.e; sample percentage in this code gives the results of accuracy and loss of the text classification.

Sample Percentage: 0.1

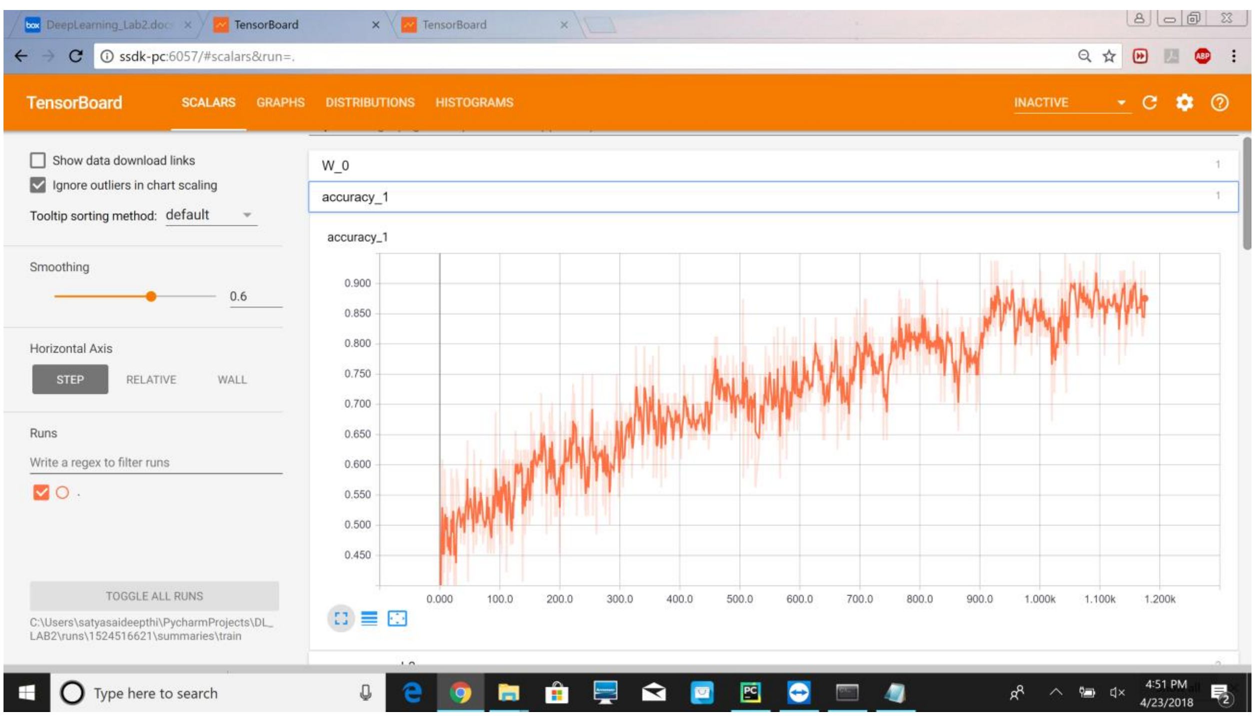
Loading data starts with setting the vocabulary data followed by loss and accuracy at each step.



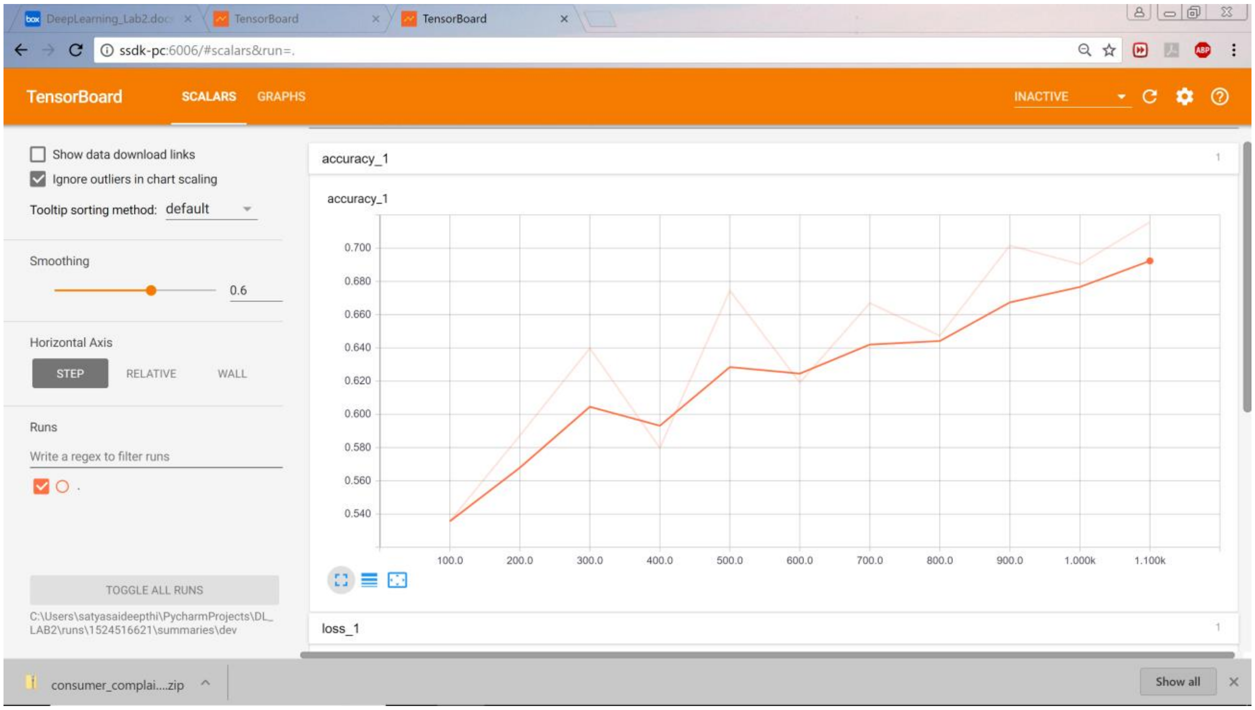
Evaluation report is stored at each step generating the average of loss and accuracy at 100^{th} step.



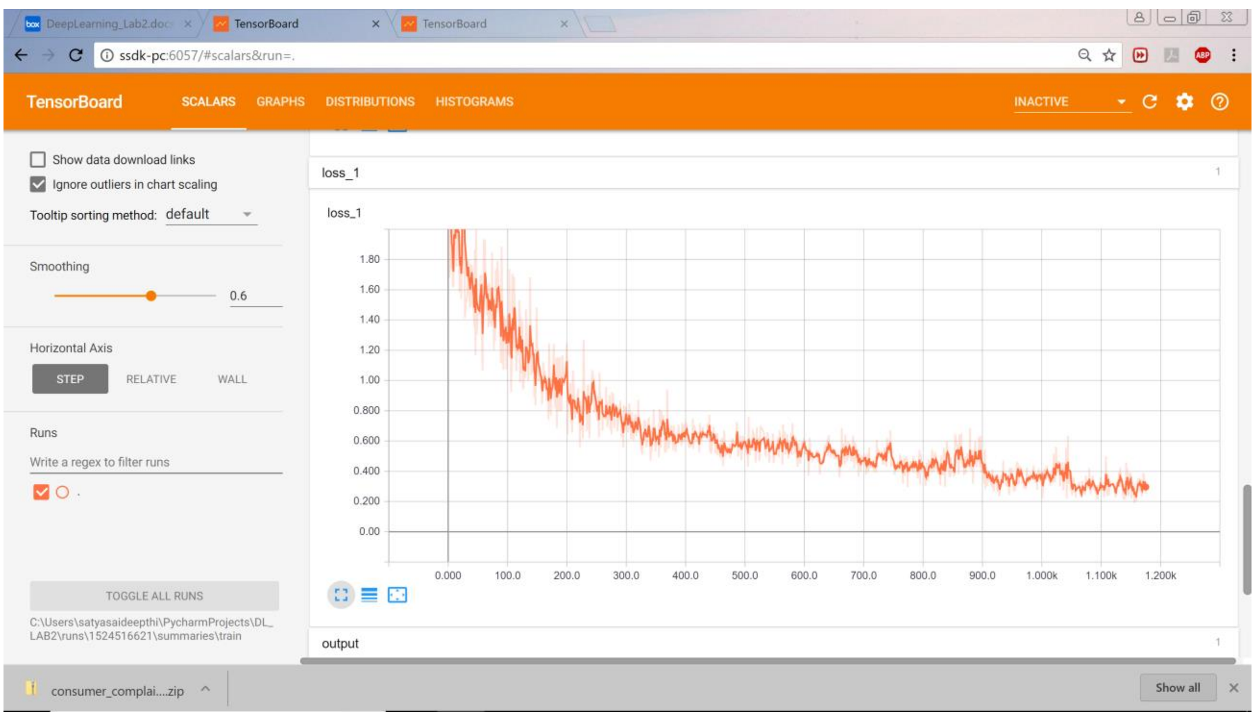
The accuracy for sample percentage = 0.1 for 1000 steps is given below:



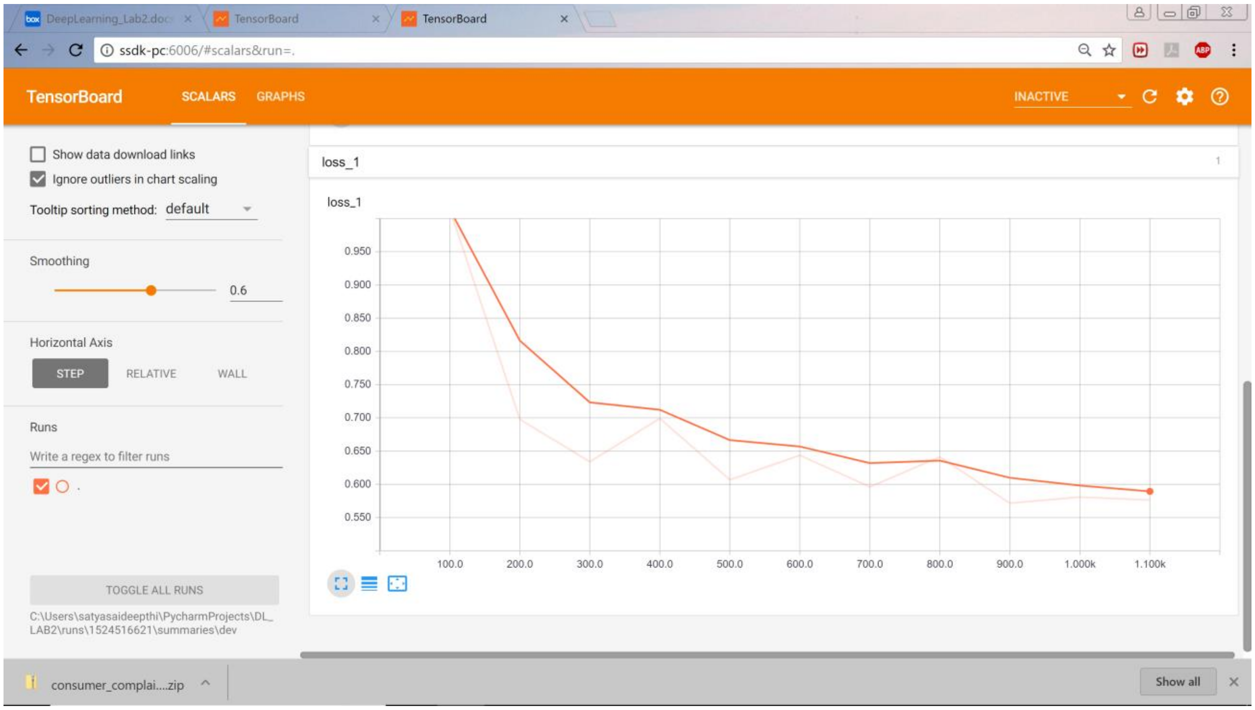
The average accuracy at each 100^{th} step is given below:



The loss for sample percentage = 0.1 for 1000 steps is given below:

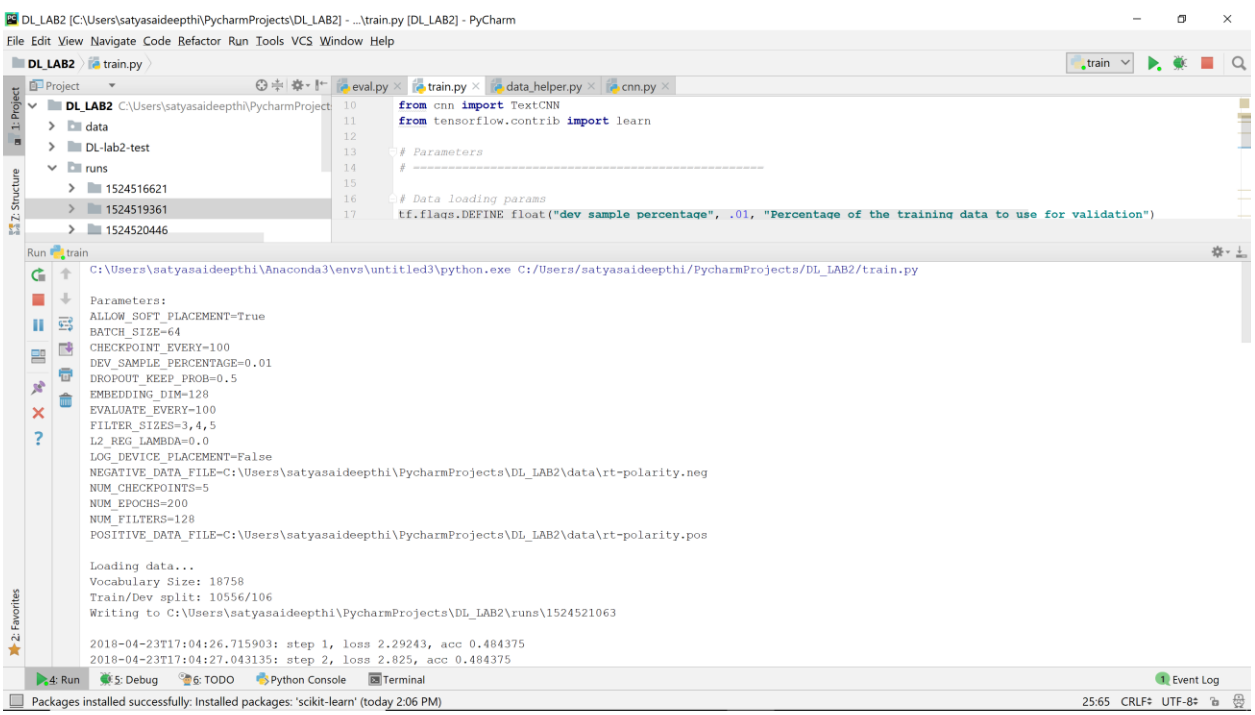


The average loss at each step 100^{th} step is below:

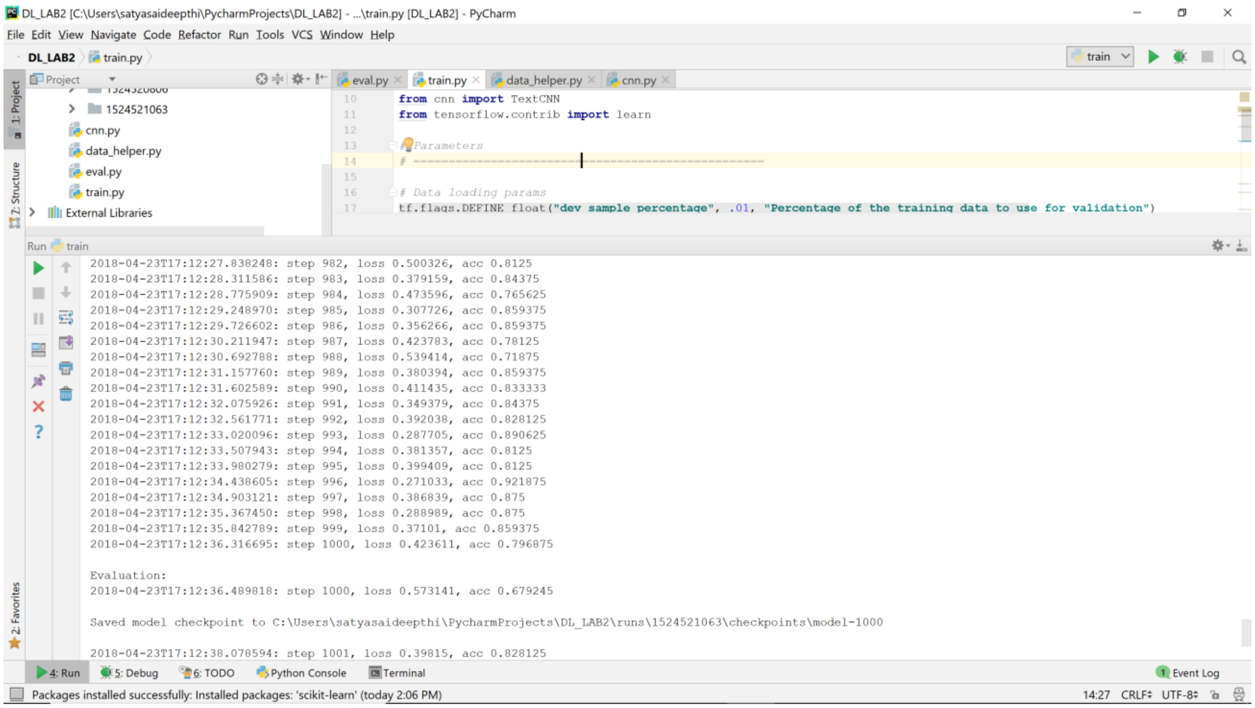


Sample Percentage = 0.01

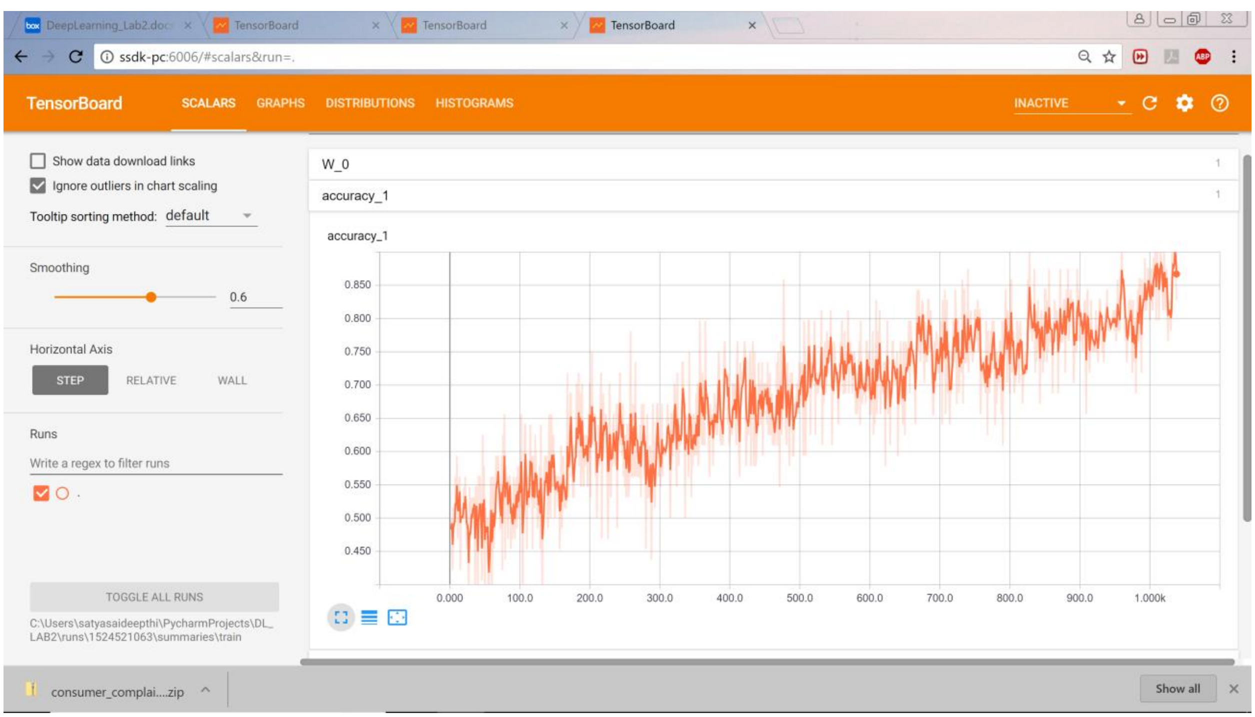
Loading data starts with setting the vocabulary data followed by loss and accuracy at each step.



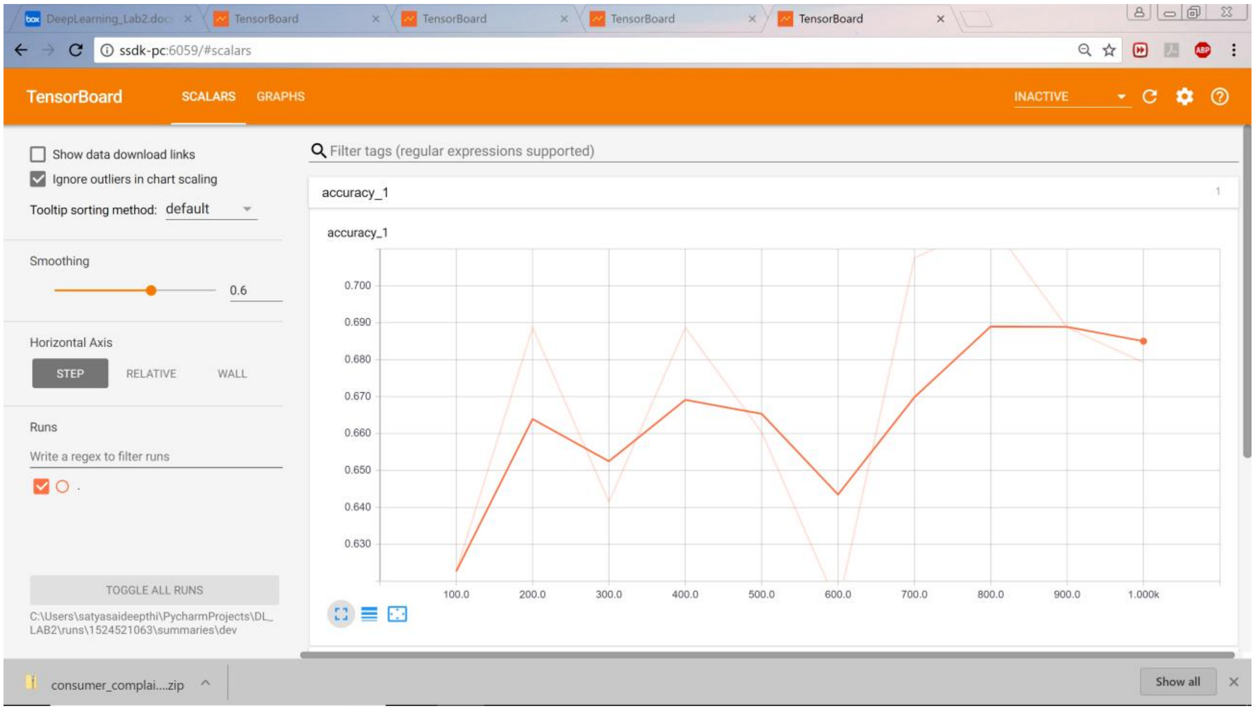
Evaluation report is stored at each step generating the average of loss and accuracy at 100th step.



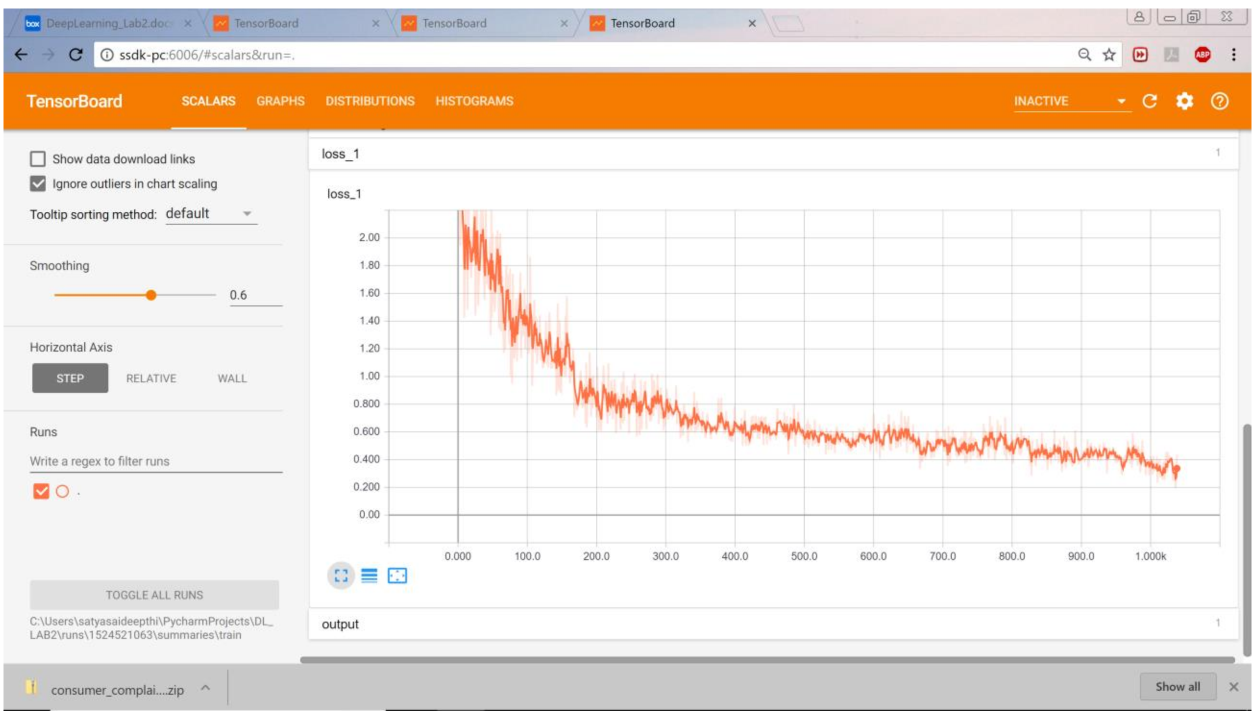
The accuracy for sample percentage = 0.01 for 1000 steps is given below:



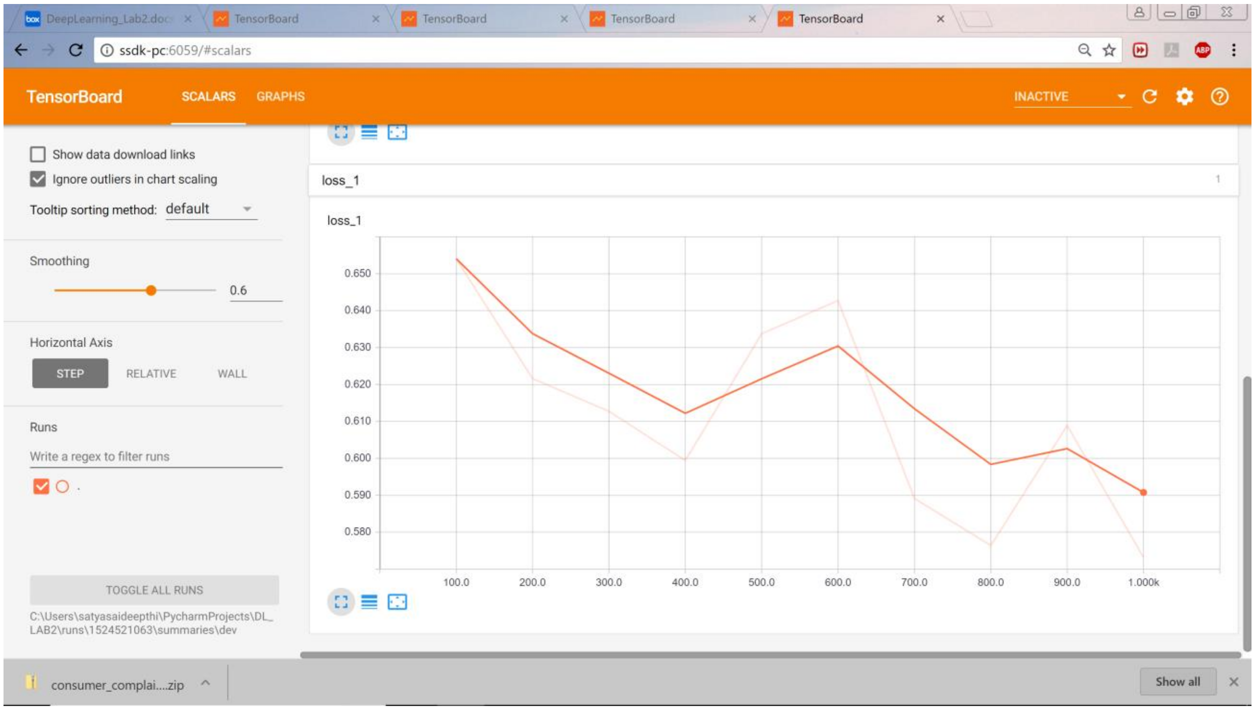
The average accuracy at each 100th step is given below:



The loss for sample percentage = 0.01 for 1000 steps is given below:



The average loss at each step 100th step is below:



Conclusion

Performing Text classification on Kaggle Consumer Finance Complaints dataset gives the following conclusions:

- By increasing the sample percentage, the accuracy value increases.
- By increasing the sample percentage, the cross-entropy loss value decreases.