LAB 7 | CE 74

In [1]:

```
# import libraries
import nltk
import re
import string
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from nltk.corpus import twitter_samples
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import TweetTokenizer
from __future__ import absolute_import, division, print_function
```

In [2]:

```
# download twitter_samples and stopwords dataset
nltk.download('twitter_samples')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package twitter_samples to /root/nltk_data...
[nltk_data]   Package twitter_samples is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[2]:

```
True
```

In [3]:

```
# function for preprocessing task and set train data
def process_tweet(tweet):
  stemmer = PorterStemmer()
  stopwords_english = stopwords.words('english')
  tweet = re.sub(r'\$\w*', '', tweet)
  tweet = re.sub(r'^RT[\s]+', '', tweet)
  tweet = re.sub(r'https?:\/\/.*[\r\n]*', '', tweet)
  tweet = re.sub(r'#', '', tweet)

  tokenizer = TweetTokenizer(preserve_case=False,strip_handles=True,reduce_len
=True)
  tweet_tokens = tokenizer.tokenize(tweet)
  tweets_clean = []
  for word in tweet_tokens:
    if (word not in stopwords_english and word not in string.punctuation):
      stem_word = stemmer.stem(word)
      tweets_clean.append(stem_word)
  return tweets_clean

def build_freqs(tweets, ys):
  yslist = np.squeeze(ys).tolist()
  freqs = {}
  for y, tweet in zip(yslist, tweets):
    for word in process_tweet(tweet):
      pair = (word, y)
      if pair in freqs:
        freqs[pair]+=1
      else:
        freqs[pair]=1
  return freqs

def extract_features(tweet, freqs):
  word_list = process_tweet(tweet)
  x = np.zeros((1,2), dtype=np.float32)
  for word in word_list:
    if (word,1) in freqs:
      x[0,0]+=freqs[word,1]
    if (word,0) in freqs:
      x[0,1]+=freqs[word,0]
  assert(x.shape==(1,2))
  return x
```

In [4]:

```
# sample of preprocessed tweet
processed_tweet = process_tweet("@Amazon is always #good company")
print(processed_tweet)
```

```
['alway', 'good', 'compani']
```

In [5]:

```
# Get the positive and negative tweets and create dataset
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')

test_pos = all_positive_tweets[3000:]
train_pos = all_positive_tweets[:3000]
test_neg = all_negative_tweets[3000:]
train_neg = all_negative_tweets[:3000]

train_x = train_pos + train_neg
test_x = test_pos + test_neg
train_y = np.append(np.ones((len(train_pos), 1),np.int64), np.zeros((len(train
_neg), 1),np.int64), axis=0)
test_y = np.append(np.ones((len(test_pos), 1),np.int64), np.zeros((len(test_ne
g), 1),np.int64), axis=0)
```

In [6]:

```
# Get word frequencies for positive and negative sentiment
freqs = build_freqs(train_x,train_y)
print("type(freqs) = " + str(type(freqs)))
print("len(freqs) = " + str(len(freqs.keys())))
```

```
type(freqs) = <class 'dict'>
len(freqs) = 9326
```

In [7]:

```
# Define parameters
num_classes = 2 # 1 or 0
num_features = 2 # positive and negative freqs
learning_rate =  0.001
training_steps = 1000
batch_size = 256
display_step = 50
```

In [8]:

```
# Get the frequencies of positive and negative word for 2 samples
sample_1 = extract_features(train_x[0], freqs)
print("sample 1 : ", sample_1)
sample_2 = extract_features(train_x[4010], freqs)
print("sample 2 : ", sample_2)
```

```
sample 1 :  [[2276.   47.]]
sample 2 :  [[  45. 2822.]]
```

In [9]:

```
# Format X_train and X_test
X_train = np.zeros((len(train_x),2),dtype=np.float32)
X_test = np.zeros((len(test_x),2),dtype=np.float32)
for i in range(len(train_x)):
  X_train[i,:] = extract_features(train_x[i],freqs)
for i in range(len(test_x)):
  X_test[i,:] = extract_features(test_x[i],freqs)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
Y_train = train_y
Y_test = test_y
print("Train sample : ",X_train[0],Y_train[0])
print("Test sample : ",X_test[1500],Y_test[1500])
```

```
Train sample :  [ 1.0975696  -0.91117305] [1]
Test sample :  [ 1.2294407  -0.74473023] [1]
```

In [10]:

```
# Intialize weight and bias
W = tf.Variable(tf.ones([num_features, num_classes]), name="weight")
b = tf.Variable(tf.zeros([num_classes]), name="bias")

# Use tf.data API to shuffle and batch data.
train_data=tf.data.Dataset.from_tensor_slices((X_train,Y_train))
train_data=train_data.repeat().shuffle(5000).batch(batch_size).prefetch(1)
```

In [11]:

```
# Main function for perform logistic regression
def logistic_regression(x,W,b):
  return tf.nn.sigmoid(tf.matmul(x,W) + b)

def cross_entropy(y_pred,y_true):
  y_true = tf.one_hot(y_true, depth=num_classes)
  y_pred = tf.clip_by_value(y_pred,1e-9,1.)
  return tf.reduce_mean(-tf.reduce_sum(y_true*tf.math.log(y_pred)))

def accuracy(y_pred, y_true):
  correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.cast(y_true, tf.int64
))
  return tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

def run_optimization(x,y):
  with tf.GradientTape() as g:
    pred = logistic_regression(x,W,b)
    loss = cross_entropy(pred,y)
  gradients = g.gradient(loss,[W,b])
  optimizer = tf.optimizers.SGD(learning_rate)
  optimizer.apply_gradients(zip(gradients, [W,b]))
```

In [12]:

```
# Train model for given number of step
for step, (batch_x, batch_y) in enumerate(train_data.take(training_steps), 1):
  run_optimization(batch_x, batch_y)
  if step % display_step == 0:
        pred = logistic_regression(batch_x,W,b)
        loss = cross_entropy(pred, batch_y)
        acc = accuracy(pred, batch_y)
        print("step: %i, loss: %f, accuracy: %f" % (step, loss, acc))
```

```
step: 50, loss: 0.118538, accuracy: 0.356598
step: 100, loss: 0.059395, accuracy: 0.406250
step: 150, loss: 0.074945, accuracy: 0.519531
step: 200, loss: 1.319319, accuracy: 0.550781
step: 250, loss: 0.029314, accuracy: 0.574219
step: 300, loss: 0.029319, accuracy: 0.578125
step: 350, loss: 0.757784, accuracy: 0.472656
step: 400, loss: 0.044673, accuracy: 0.429688
step: 450, loss: 0.046378, accuracy: 0.386719
step: 500, loss: 0.572788, accuracy: 0.468750
step: 550, loss: 0.099040, accuracy: 0.585938
step: 600, loss: 0.027436, accuracy: 0.578125
step: 650, loss: 0.027968, accuracy: 0.621094
step: 700, loss: 0.037113, accuracy: 0.437500
step: 750, loss: 0.058728, accuracy: 0.421875
step: 800, loss: 0.113535, accuracy: 0.417969
step: 850, loss: 0.499733, accuracy: 0.449615
step: 900, loss: 0.033708, accuracy: 0.488281
step: 950, loss: 0.389323, accuracy: 0.515503
step: 1000, loss: 0.026424, accuracy: 0.625000
```

In [13]:

```
#Final weight
print("Weight : ")
print(W)
#Final bias
print("Bias : ")
print(b)
```

```
Weight :
<tf.Variable 'weight:0' shape=(2, 2) dtype=float32, numpy=
array([[-0.65007293, -1.4276    ],
       [-0.7790809 , -1.7711275 ]], dtype=float32)>
Bias :
<tf.Variable 'bias:0' shape=(2,) dtype=float32, numpy=array([14.012577, 22.61
566 ], dtype=float32)>
```

In [14]:

```
pred = logistic_regression(X_test,W,b)
print("Test accuracy: %f" % accuracy(pred,Y_test))
```

```
Test accuracy: 0.500000
```