In [1]:

```
import numpy as np
import pandas as pd
import bcolz as bz
import pickle
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.nn.utils.rnn import pack_padded_sequence, pad_packed_sequence
from torch.autograd import Variable
import time
from sklearn import metrics
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
```

In [2]:

```
def divide_string(sentence):
    s_l = sentence.split()
    r_l = []
    punc = [ '?', '!', '*']
    delete = ['%','^','&','_','>','<','\\','$', '+', '~','`', ',', '.', ';',
              '#', '@', '(', ')', '[' , ']', '/', '|', '{', '}', '"', '-', '=']
    for i in range(len(s_l)):
        tmp = 0
        flag = True;
        for j in range(len(s_l[i])):
            if s_l[i][j] in punc:
                if(flag):
                    r_l.append(s_l[i][tmp : j])
                tmp = j
                flag = True
            if s_l[i][j] in delete:
                if(flag):
                    r_l.append(s_l[i][tmp : j])
                tmp = j
                flag = False
            else:
                if(not flag):
                    tmp = tmp + 1;
                flag = True
        if(flag):
            r_l.append(s_l[i][ tmp : len(s_l[i]) ] )
    str_list = [x for x in r_l if x != '']
    return str_list
```

In [3]:

```
a = "THIS is a good day!=== don't}}} know ====================================="
b = divide_string(a)
print(len(b))
print(b)
```

```
8
['THIS', 'is', 'a', 'good', 'day', '!', "don't", 'know']
```

In [4]:

```
vectors = bz.open(r"C:\Users\mul02\Desktop\Course\LIGN 167\Final Project\glove\27B.100.dat")[:]
words = pickle.load(open(r"C:\Users\mul02\Desktop\Course\LIGN 167\Final
Project\glove\27B.100_words.pkl", 'rb'))
word2idx = pickle.load(open(r"C:\Users\mul02\Desktop\Course\LIGN 167\Final
Project\glove\27B.100_idx.pkl", 'rb'))

glove = {w: vectors[word2idx[w]] for w in words}
print(len(glove))
```

```
1193514
```

In [5]:

```
def remove_infrequent_words(sents):
    word_counts = {}
    divide_sentence = []
    #divide each sentence first
    for s in sents:
        tmp = divide_string(s)
        divide_sentence.append(tmp)

    # count the word
    for s in divide_sentence:
        for w in s:
            if w in word_counts:
                word_counts[w] += 1
            else:
                word_counts[w] = 1

    threshold = 2
    filtered_sents = []
    for s in divide_sentence:
        new_s = []
        for w in s:
            if word_counts[w] < threshold:
                new_s.append('<UNKOWN>')
            else:
                new_s.append(w)
        filtered_sents.append(new_s)
    return filtered_sents
```

In [6]:

```
s = ["This is a good time!!", "You are pretty@$&^$@ happy", "test", "test*%& test"]
def word_embedding_glove_total(sentence):
    #create the total size of weight matrix
    weights_matrix = np.zeros((100,len(sentence), 103))


    processed_sentence = remove_infrequent_words(sentence)

    for k in range(len(processed_sentence)):
        target_vocab = processed_sentence[k]
        emb_dim = 100

        # enumerate the word, break at position 100
        for i, word in enumerate(target_vocab):
            if i == 100:
                break;
            try:
                tmp_vec = glove[word]
            except KeyError:
                tmp_vec = np.random.normal(scale=0.6, size=(emb_dim, ))
                tmp_dic = {word:tmp_vec}
                glove.update(tmp_dic)
            #write something to change it to 103
            ind = 0
            if word.isupper():
                tmp_vec = np.append(tmp_vec, [1])
            else:
                tmp_vec = np.append(tmp_vec, [0])
                ind = ind + 1

            if word.islower():
                tmp_vec = np.append(tmp_vec, [1])
            else:
                tmp_vec = np.append(tmp_vec, [0])
                ind = ind + 1

            if ind == 2:
                tmp_vec = np.append(tmp_vec, [1])
            else:
                tmp_vec = np.append(tmp_vec, [0])

            weights_matrix[i][k] = tmp_vec;
    return weights_matrix
```

```
test =  word_embedding_glove_total(s)
print(test.shape)
sentences = remove_infrequent_words(s)
print(sentences)
```

```
(100, 4, 103)
[['<UNKOWN>', '<UNKOWN>', '<UNKOWN>', '<UNKOWN>', '<UNKOWN>', '!', '!'], ['<UNKOWN>', '<UNKOWN>',
'<UNKOWN>', '<UNKOWN>'], ['test'], ['test', '<UNKOWN>', 'test']]
```

In [7]:

```
s = ["This is a good time!!", "You are pretty@$&^$@ happy", "test", "test*%& test"]
def base_glove_total(sentence):
    #create the total size of weight matrix
    weights_matrix = np.zeros((100,len(sentence), 103))


    for k in range(len(sentence)):
        target_vocab = sentence[k].split()
        emb_dim = 100

        # enumerate the word, break at position 100
        for i in range(len(target_vocab)):
            if i == 100:
                break;
            try:
                word = target_vocab[i]
                tmp_vec = glove[target_vocab[i]]
            except KeyError:
                tmp_vec = np.random.normal(scale=0.6, size=(emb_dim, ))

            #write something to change it to 103
            ind = 0
            if word.isupper():
                tmp_vec = np.append(tmp_vec, [1])
            else:
                tmp_vec = np.append(tmp_vec, [0])
                ind = ind + 1

            if word.islower():
                tmp_vec = np.append(tmp_vec, [1])
            else:
                tmp_vec = np.append(tmp_vec, [0])
                ind = ind + 1

            if ind == 2:
                tmp_vec = np.append(tmp_vec, [1])
            else:
                tmp_vec = np.append(tmp_vec, [0])

            weights_matrix[i][k] = tmp_vec;
    return weights_matrix

test =  base_glove_total(s)
print(test.shape)
```

```
(100, 4, 103)
```

In [8]:

```
def create_emb_layer(weights_matrix, non_trainable=False):
    num_embeddings, embedding_dim = weights_matrix.shape
    emb_layer = nn.Embedding(num_embeddings, embedding_dim)
    emb_layer.load_state_dict({'weight': weights_matrix})
    if non_trainable:
        emb_layer.weight.requires_grad = False

    return emb_layer, num_embeddings, embedding_dim


class tcLSTM(nn.Module):
    def __init__(self, input_dim, hidden_size, target_size, batch_size):
```

```
        super(tcLSTM, self).__init__()
        self.embedding_dim = input_dim
        self.hidden_size = hidden_size
        self.batch_size = batch_size
        self.lstm = nn.LSTM(self.embedding_dim, hidden_size)

        # The linear layer that maps from hidden state space to target space
        self.hidden2tar = nn.Linear(self.hidden_size, target_size)
        self.hidden = self.init_hidden()

    def forward(self, weights_matrix):
        #didn't use batch, may need to add batch size in the middle
        lstm_out, self.hidden = self.lstm(weights_matrix)

        y_pred = self.hidden2tar(lstm_out[-1])
        return torch.sigmoid(y_pred)


    def init_hidden(self):
        return (Variable(torch.zeros(1, self.batch_size, self.hidden_size)),
                Variable(torch.zeros(1, self.batch_size, self.hidden_size)))
```

In [9]:

```
data = pd.read_csv(r"C:\Users\mul02\Desktop\Course\LIGN 167\Final Project\data\train.csv",
encoding = "ISO-8859-1")
#data.head()
```

In [10]:

```
x_train = data['comment_text']
y_train = data.iloc[:,2:8]
print(x_train.shape)
print(y_train.shape)
```

```
(159571,)
(159571, 6)
```

In [11]:

```
data = pd.read_csv(r"C:\Users\mul02\Desktop\Course\LIGN 167\Final Project\data\test.csv", encoding
= "ISO-8859-1")
x_test = data['comment_text']
print(x_test.shape)
data = pd.read_csv(r"C:\Users\mul02\Desktop\Course\LIGN 167\Final Project\data\test_labels.csv", e
ncoding = "ISO-8859-1")
y_test = data.iloc[:,1:7]
print(y_test.shape)
```

```
(153164,)
(153164, 6)
```

In [12]:

```
#the embedding
train_set = word_embedding_glove_total(x_train)
#train_set = base_glove_total(x_train)
```

In [13]:

```
print(train_set.shape)
w = train_set[:,512:1024,:]
print(w.shape)
c = y_train.values
c = c.reshape((1,159571,6))
print(c.shape)
```

```
(100, 159571, 103)
(100, 512, 103)
(1, 159571, 6)
```

In [14]:

```
data = pd.read_csv(r"C:\Users\mul02\Desktop\Course\LIGN 167\Final Project\data\test.csv", encoding
= "ISO-8859-1")
data2 = pd.read_csv(r"C:\Users\mul02\Desktop\Course\LIGN 167\Final Project\data\test_labels.csv",
encoding = "ISO-8859-1")
df = pd.concat([data.reset_index(drop=True), data2.reset_index(drop=True)], axis=1)
df = df.drop(['id', 'id'],axis=1)
df = df[df.toxic != -1]
print(data.shape)
print(data2.shape)
print(df.shape)
x_test = df.reset_index()['comment_text']
y_test = df.reset_index().iloc[:, 2:8]
print(x_test.shape)
print(y_test.shape)
```

```
(153164, 2)
(153164, 7)
(63978, 7)
(63978,)
(63978, 6)
```

In [15]:

```
batch_size = 512
epoch = 100
lr = 0.005
hidden_state = 24
output_dim = 6
input_dim = 103
train_size = len(x_train)
train_output = y_train.values


model = tcLSTM(input_dim=input_dim, hidden_size=hidden_state, target_size=output_dim, batch_size =
batch_size)
model = model.cuda()
loss_fn = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```

In [16]:

```
start = time.time()

#Training
for i in range(epoch):  # again, normally you would NOT do 300 epochs, it is toy data
    j = 0
    total_loss = 0
    while(j < train_size ):
        k = j + 512;
        if k > train_size:
            break

        #prepare input and output
        x_t = train_set[:,j:k,:]
        x_t = torch.cuda.FloatTensor(x_t)
        y_t = train_output[j:k,:]
        #y_t = y_t.sum(axis=1)
        y_t = torch.cuda.FloatTensor(y_t)

        # Step 1. Remember that Pytorch accumulates gradients.
        # We need to clear them out before each instance
        model.zero_grad()

        # Also, we need to clear out the hidden state of the LSTM,
        # detaching it from its history on the last instance.
        model.hidden = model.init_hidden()


        # Step 2. Run our forward pass.
        y_pred = model(x_t)
        y_pred = y_pred.reshape((512,6))
```

```
        y_pred = y_pred.reshape((512,))


        # Step 4. Compute the loss, gradients, and update the parameters by
        #  calling optimizer.step()
        loss = loss_fn(y_pred, y_t)
        total_loss += loss
        loss.backward()
        optimizer.step()
        j = k

    print(total_loss)



end = time.time()
print(end - start)
```

```
tensor(50.9755, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(26.5789, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(20.6462, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(18.2622, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(17.1125, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(16.5756, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(16.0052, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(15.6461, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(15.2870, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(15.0594, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(14.8109, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(14.5929, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(14.4486, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(14.2640, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(14.1223, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(14.0939, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(13.9003, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(13.8095, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(13.8289, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(13.5723, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(13.6184, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(13.6492, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(13.7871, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(13.3744, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(13.1718, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(13.0925, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(13.0016, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.8202, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.6674, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.7638, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.8251, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.6669, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.6374, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.5004, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.4183, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.3117, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.4911, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.3462, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.2893, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.2929, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.3215, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.1323, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.1822, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.0597, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.9271, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.9397, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.8751, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.9765, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.9381, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.0447, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.8232, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.7424, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.7341, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.8392, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.7028, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.7153, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.7565, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.6475, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.6694, device='cuda:0', grad_fn=<ThAddBackward>)
```

```
tensor(11.5199, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.5948, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.5233, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.5300, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.5061, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.3988, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.3948, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.4354, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.2962, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.4013, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.4855, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.3449, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.2600, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.2647, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.2138, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.4501, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.1831, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.0584, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(13.1655, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.6704, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.8622, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.6644, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.3593, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.2854, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.3863, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.3569, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.2513, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.0933, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.2250, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.1404, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.1831, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.0189, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(10.9212, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.0028, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(10.9860, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(10.8652, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(10.9190, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.0704, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(12.0151, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.7395, device='cuda:0', grad_fn=<ThAddBackward>)
tensor(11.3191, device='cuda:0', grad_fn=<ThAddBackward>)
797.7614636421204
```

In [17]:

```
#test set
total = 10000
test_set_d = x_test[0:total]
test_output = y_test.values[0:total, :]
test_set = word_embedding_glove_total(test_set_d)
#test_set = base_glove_total(test_set_d)
print(len(glove))
```

```
1270947
```

In [18]:

```
#then do the testing
#first test on 10000 set
correct = 0


with torch.no_grad():
    x_t = torch.cuda.FloatTensor(test_set)
    y_t = torch.cuda.FloatTensor(test_output)
    y_pred = model(x_t)
    y_pred = y_pred.reshape((total, 6))
    y_pred_c = y_pred > 0.6
    y_pred_c = y_pred_c.type(torch.cuda.FloatTensor)
    for i in range(total):
        if torch.equal(y_pred_c[i], y_t[i]):
            correct = correct + 1
    print(correct)
```

```
8669
```

In [ ]:

```

```

In [ ]:

```

```