In [1]:

```
import numpy as np
import pandas as pd
import re
import os
import sys

from sklearn.metrics import recall_score, precision_score, accuracy_score,
f1_score
from sklearn.feature_extraction.text import CountVectorizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, Dropout
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras import backend as K
```

```
Using TensorFlow backend.
```

In [2]:

```
with open("sent_corpus.csv", "r") as sent_file:
    lines = sent_file.read().split("\n")

rows = [line.split(",") for line in lines if line]
rows = [row[:3] + [",".join(row[3:])] for row in rows]
# remove document start character
rows[0][0] = rows[0][0][1:]
sentDf_cols = ['ItemID', 'Sentiment', 'SentimentSource', 'SentimentText']
sentDf = pd.DataFrame(rows[1:],columns=sentDf_cols)
print(sentDf.columns.values)
```

```
['ItemID' 'Sentiment' 'SentimentSource' 'SentimentText']
```

In [3]:

```
sentDf['SentimentText'] = sentDf['SentimentText'].apply(lambda x: x.lower()
)
sentDf['SentimentText'] = sentDf['SentimentText'].apply((lambda x: re.sub('
[^a-zA-z0-9\s]','',x)))

max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(sentDf['SentimentText'].values)
X = tokenizer.texts_to_sequences(sentDf['SentimentText'].values)
X = pad_sequences(X)
```

In [4]:

```
print(sentDf[ sentDf['Sentiment'] == '1'].size)
print(sentDf[ sentDf['Sentiment'] == '0'].size)
Y = pd.get_dummies(sentDf["Sentiment"]).values
```

```
3160740
3153768
```

In [5]:

```
def f1(y_true, y_pred):
    def recall(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        recall = true_positives / (possible_positives + K.epsilon())
        return recall

    def precision(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision

    precision = precision(y_true, y_pred)
    recall = recall(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall))

def recall(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision
```

In [6]:

```
embed_dim = 128
lstm_out = 196

model = Sequential()
model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1]))
model.add(Dropout(0.2))
model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(2,activation='softmax'))
model.compile(loss = 'binary_crossentropy', optimizer='adam',metrics = ['ac
curacy'])
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 40, 128)           256000    
_________________________________________________________________
dropout_1 (Dropout)          (None, 40, 128)           0         
_________________________________________________________________
lstm_1 (LSTM)                (None, 196)               254800    
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 394       
=================================================================
Total params: 511,194
Trainable params: 511,194
Non-trainable params: 0
_________________________________________________________________
None
```

```
In [7]: X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.33,
        random_state = 42)
        print(X_train.shape,Y_train.shape)
        print(X_test.shape,Y_test.shape)
```

```
(1057680, 40) (1057680, 2)
(520947, 40) (520947, 2)
```

Training the model takes about 22 minutes for each epoch.

```
In [8]: batch_size = 32
        model.fit(X_train, Y_train, initial_epoch=5, batch_size=batch_size, verbose
        =1)
```

```
Epoch 6/10
1057680/1057680 [==============================] - 1348s 1ms/step - loss: 0
.4474 - acc: 0.7890
Epoch 7/10
1057680/1057680 [==============================] - 1346s 1ms/step - loss: 0
.4227 - acc: 0.8035
Epoch 8/10
1057680/1057680 [==============================] - 1347s 1ms/step - loss: 0
.4133 - acc: 0.8094
Epoch 9/10
1057680/1057680 [==============================] - 1349s 1ms/step - loss: 0
.4086 - acc: 0.8120
Epoch 10/10
1057680/1057680 [==============================] - 1354s 1ms/step - loss: 0
.4059 - acc: 0.8136
```

```
Out[8]: <keras.callbacks.History at 0x7f3be6121438>
```

Testing on the test set takes about 3 minutes.

```
In [9]: validation_size = 1500

        X_validate = X_test[-validation_size:]
        Y_validate = Y_test[-validation_size:]
        X_test = X_test[:-validation_size]
        Y_test = Y_test[:-validation_size]
        #metrics = model.evaluate(X_test, Y_test, verbose = 1, batch_size = batch_s
        ize)
```

```
In [10]: yp = model.predict(X_test, batch_size=32, verbose=1)
         ypreds = np.argmax(yp, axis=1)
         print(accuracy_score(Y_test[:,1], ypreds))
         print(precision_score(Y_test[:,1], ypreds))
         print(recall_score(Y_test[:,1], ypreds))
         print(f1_score(Y_test[:,1], ypreds))
```

```
519447/519447 [==============================] - 176s 338us/step
0.810448419184
0.803874201171
0.822235970902
0.812951416989
```

In [11]:

```
pos_cnt, neg_cnt, pos_correct, neg_correct = 0, 0, 0, 0
for x in range(len(X_validate)):

    result = model.predict(X_validate[x].reshape(1,X_test.shape[1]),batch_s
ize=1,verbose = 2)[0]

    if np.argmax(result) == np.argmax(Y_validate[x]):
        if np.argmax(Y_validate[x]) == 0:
            neg_correct += 1
        else:
            pos_correct += 1

    if np.argmax(Y_validate[x]) == 0:
        neg_cnt += 1
    else:
        pos_cnt += 1



print("pos_acc", pos_correct/pos_cnt*100, "%")
print("neg_acc", neg_correct/neg_cnt*100, "%")
```

```
pos_acc 84.40860215053763 %
neg_acc 77.64550264550265 %
```