Import necessary depencencies

In [1]:
```
import pandas as pd
import numpy as np
import text_normalizer as tn
import model_evaluation_utils as meu

np.set_printoptions(precision=2, linewidth=80)
```

Load and normalize data

In [2]:
```
dataset = pd.read_csv(r'movie_reviews_cleaned.csv')

# take a peek at the data
print(dataset.head())
reviews = np.array(dataset['review'])
sentiments = np.array(dataset['sentiment'])

# build train and test datasets
norm_train_reviews = reviews[:35000]
train_sentiments = sentiments[:35000]
norm_test_reviews = reviews[35000:]
test_sentiments = sentiments[35000:]

# normalize datasets
#norm_train_reviews = tn.normalize_corpus(train_reviews)
#norm_test_reviews = tn.normalize_corpus(test_reviews)
```

```
                                              review sentiment
0  not bother think would see movie great supspen...  negative
1  careful one get mitt change way look kung fu f...  positive
2  chili palmer tired movie know want success mus...  negative
3  follow little know 1998 british film make budg...  positive
4  dark angel cross huxley brave new world percys...  positive
```

Tokenize train & test datasets

In [3]:
```
tokenized_train = [tn.tokenizer.tokenize(text) for text in norm_train_reviews
tokenized_test = [tn.tokenizer.tokenize(text) for text in norm_test_reviews]
```

Build Vocabulary Mapping (word to index)

In [4]:

```
from collections import Counter

# build word to index vocabulary
token_counter = Counter([token for review in tokenized_train for token in rev
vocab_map = {item[0]: index+1 for index, item in enumerate(dict(token_counter
max_index = np.max(list(vocab_map.values()))
vocab_map['PAD_INDEX'] = 0
vocab_map['NOT_FOUND_INDEX'] = max_index+1
vocab_size = len(vocab_map)
# view vocabulary size and part of the vocabulary map
print('Vocabulary Size:', vocab_size)
print('Sample slice of vocabulary map:', dict(list(vocab_map.items())[10:20])
```

```
Vocabulary Size: 80004
Sample slice of vocabulary map: {'boring': 11, 'terribly': 12, 'predictabl
e': 13, 'interesting': 14, 'start': 15, 'middle': 16, 'film': 17, 'little':
18, 'social': 19, 'drama': 20}
```

Encode and Pad datasets & Encode prediction class labels

In [5]:

```
from keras.preprocessing import sequence
from sklearn.preprocessing import LabelEncoder

# get max length of train corpus and initialize label encoder
le = LabelEncoder()
num_classes=2 # positive -> 1, negative -> 0
max_len = np.max([len(review) for review in tokenized_train])

## Train reviews data corpus
# Convert tokenized text reviews to numeric vectors
train_X = [[vocab_map[token] for token in tokenized_review] for tokenized_rev
train_X = sequence.pad_sequences(train_X, maxlen=max_len) # pad
## Train prediction class labels
# Convert text sentiment labels (negative\positive) to binary encodings (0/1)
train_y = le.fit_transform(train_sentiments)

## Test reviews data corpus
# Convert tokenized text reviews to numeric vectors
test_X = [[vocab_map[token] if vocab_map.get(token) else vocab_map['NOT_FOUND
           for token in tokenized_review]
              for tokenized_review in tokenized_test]
test_X = sequence.pad_sequences(test_X, maxlen=max_len)
## Test prediction class labels
# Convert text sentiment labels (negative\positive) to binary encodings (0/1)
test_y = le.transform(test_sentiments)

# view vector shapes
print('Max length of train review vectors:', max_len)
print('Train review vectors shape:', train_X.shape, ' Test review vectors sha
```

```
Using TensorFlow backend.

Max length of train review vectors: 1115
Train review vectors shape: (35000, 1115)  Test review vectors shape: (1500
0, 1115)
```

Build the LSTM Model Architecture

In [6]:

```
from keras.models import Sequential
from keras.layers import Dense, Embedding, Dropout, SpatialDropout1D
from keras.layers import LSTM

EMBEDDING_DIM = 128 # dimension for dense embeddings for each token
LSTM_DIM = 64 # total LSTM units

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=EMBEDDING_DIM, input_ler
model.add(SpatialDropout1D(0.2))
model.add(LSTM(LSTM_DIM, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation="sigmoid"))

model.compile(loss="binary_crossentropy", optimizer="adam",
              metrics=["accuracy"])
```

```
WARNING:tensorflow:From c:\users\finan\appdata\local\programs\python\python
37\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: col
ocate_with (from tensorflow.python.framework.ops) is deprecated and will be
removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From c:\users\finan\appdata\local\programs\python\python
37\lib\site-packages\keras\backend\tensorflow_backend.py:3445: calling drop
out (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and wi
ll be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
```

In [7]:

```
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 1115, 128)         10240512  
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 1115, 128)         0         
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                49408     
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65        
=================================================================
Total params: 10,289,985
Trainable params: 10,289,985
Non-trainable params: 0
_________________________________________________________________
None
```

Visualize model architecture

In [ ]:
```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

SVG(model_to_dot(model, show_shapes=True, show_layer_names=False,
                 rankdir='LR').create(prog='dot', format='svg'))
```

Train the model

In [10]:
```
batch_size = 100
model.fit(train_X, train_y, epochs=5, batch_size=batch_size,
          shuffle=True, validation_split=0.1, verbose=1)
```

```
1:09 - loss: 0.0975 - acc: 0.96 - ETA: 11:03 - loss: 0.0973 - acc: 0.96 -
ETA: 10:58 - loss: 0.0978 - acc: 0.96 - ETA: 10:52 - loss: 0.0982 - acc:
0.96 - ETA: 10:47 - loss: 0.0982 - acc: 0.96 - ETA: 10:41 - loss: 0.0982
- acc: 0.96 - ETA: 10:36 - loss: 0.0986 - acc: 0.96 - ETA: 10:30 - loss:
0.0989 - acc: 0.96 - ETA: 10:25 - loss: 0.0988 - acc: 0.96 - ETA: 10:20 -
loss: 0.0989 - acc: 0.96 - ETA: 10:14 - loss: 0.0991 - acc: 0.9661



31500/31500 [==============================] - ETA: 10:09 - loss: 0.0991
- acc: 0.96 - ETA: 10:04 - loss: 0.0992 - acc: 0.96 - ETA: 9:58 - loss:
0.0994 - acc: 0.9660 - ETA: 9:53 - loss: 0.0994 - acc: 0.966 - ETA: 9:47
- loss: 0.0993 - acc: 0.966 - ETA: 9:42 - loss: 0.0992 - acc: 0.966 - ET
A: 9:37 - loss: 0.0994 - acc: 0.966 - ETA: 9:31 - loss: 0.0992 - acc: 0.9
66 - ETA: 9:26 - loss: 0.0992 - acc: 0.966 - ETA: 9:20 - loss: 0.0993 - a
cc: 0.966 - ETA: 9:15 - loss: 0.0992 - acc: 0.966 - ETA: 9:10 - loss: 0.0
990 - acc: 0.966 - ETA: 9:04 - loss: 0.0994 - acc: 0.966 - ETA: 8:59 - lo
ss: 0.0995 - acc: 0.966 - ETA: 8:54 - loss: 0.0994 - acc: 0.965 - ETA: 8:
49 - loss: 0.0994 - acc: 0.965 - ETA: 8:43 - loss: 0.0993 - acc: 0.965 -
ETA: 8:38 - loss: 0.0992 - acc: 0.965 - ETA: 8:33 - loss: 0.0995 - acc:
```

Predict and Evaluate Model Performance

In [11]:
```
pred_test = model.predict_classes(test_X)
predictions = le.inverse_transform(pred_test.flatten())
```

In [12]:
```
meu.display_model_performance_metrics(true_labels=test_sentiments, predicted_
                                      classes=['positive', 'negative'])
```

```
Model Performance metrics:
------------------------------
Accuracy: 0.8777
Precision: 0.8778
Recall: 0.8777
F1 Score: 0.8777

Model Classification report:
------------------------------
              precision    recall  f1-score   support

    positive       0.88      0.87      0.88      7587
    negative       0.87      0.88      0.88      7413

   micro avg       0.88      0.88      0.88     15000
   macro avg       0.88      0.88      0.88     15000
weighted avg       0.88      0.88      0.88     15000


Prediction Confusion Matrix:
------------------------------

C:\Users\finan\model_evaluation_utils.py:62: FutureWarning: the 'labels' ke
yword is deprecated, use 'codes' instead
  labels=level_labels),
C:\Users\finan\model_evaluation_utils.py:64: FutureWarning: the 'labels' ke
yword is deprecated, use 'codes' instead
  labels=level_labels))

                 Predicted:
                   positive negative
Actual: positive       6615      972
        negative        863     6550
```

In [ ]:

Introduction

In this project, I classify Yelp round-10 review datasets. The reviews contain a lot of metadata that can be mined and used to infer meaning, business attributes, and sentiment. For simplicity, I classify the review comments into two class: either as positive or negative. Reviews that have star higher than three are regarded as positive while the reviews with star less than or equal to 3 are negative. Therefore, the problem is a supervised learning. To build and train the model, I first tokenize the text and convert them to sequences. Each review comment is limited to 50 words. As a result, short texts less than 50 words are padded with zeros, and long ones are truncated. After processing the review comments, I trained three model in three different ways:

- Model-1: In this model, a neural network with LSTM and a single embedding layer were used.
- Model-2: In Model-1, an extra 1D convolutional layer has been added on top of LSTM layer to reduce the training time.
- Model-3: In this model, I use the same network architecture as Model-2, but use the pre-trained glove 100 dimension word embeddings as initial input.

Since there are about 1.6 million input comments, it takes a while to train the models. To reduce the training time step, I limit the training epoch to three. After three epochs, it is evident that Model-2 is better regarding both training time and validation accuracy.

Project Outline

In this project I will cover the follwouings :

- Download data from yelp and process them
- Build neural network with LSTM
- Build neural network with LSTM and CNN
- Use pre-trained GloVe word embeddings
- Word Embeddings from Word2Vec

Import libraries

In [1]:

```
# Keras
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Flatten, LSTM, Conv1D, MaxPooling1D, Dropout,
from keras.layers.embeddings import Embedding

## Plot
import plotly.offline as py
import plotly.graph_objs as go
py.init_notebook_mode(connected=True)
import matplotlib as plt

# NLTK
import nltk
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer

# Other
import re
import string
import numpy as np
import pandas as pd
from sklearn.manifold import TSNE
```

```
Using TensorFlow backend.
```

Data Processing

In [2]:

```
df = pd.read_csv('yelp.csv', names = ['stars', 'text'], error_bad_lines=False
```

In [3]:

```
df= df.dropna()
df = df[df.stars.apply(lambda x: x.isnumeric())]
df = df[df.stars.apply(lambda x: x !="")]
df = df[df.text.apply(lambda x: x !="")]
```

In [4]:

```
df.describe()
```

Out[4]:

	stars	text
count	10000	10000
unique	28	29
top	0	0
freq	4130	7013

In [5]:
```
df.head()
```

Out[5]:

9yKzy9PApeiPPOUJEtnvkg	2011-01-26	fWKvX83p0-ka4JS3dc6E5A	5	My wife took me here on my birthday for breakfast and it was excellent. The weather was perfect which made sitting outside overlooking their grounds an absolute pleasure. Our waitress was excellent and our food arrived quickly on the semi-busy Saturday morning. It looked like the place fills up pretty quickly so the earlier you get here the better.\n\nDo yourself a favor and get their Bloody Mary. It was phenomenal and simply the best I've ever had. I'm pretty sure they only use ingredients from their garden and blend them fresh when you order it. It was amazing.\n\nWhile EVERYTHING on the menu looks excellent, I had the white truffle scrambled eggs vegetable skillet and it was tasty and delicious. It came with 2 pieces of their griddled bread with was amazing and it absolutely made the meal complete. It was the best "toast" I've ever had.\n\nAnyway, I can't wait to go back!	rev

ZRJwVLyzEJq1VAihDhYiow	2011-07-27	IjZ33sJrzXqU-0X6U8NwyA	5	I have no idea why some people give bad reviews about this place. It goes to show you, you can please everyone. They are probably griping about something that their own fault...there are many people like that.\n\nIn any case, my friend and I arrived at about 5:50 PM this past Sunday. It was pretty crowded, more than I thought for a Sunday evening and thought we would have to wait forever to get a seat but they said we'll be seated when the girl comes back from seating someone else. We were seated at 5:52 and the waiter came and got our drink orders. Everyone was very pleasant from the host that seated us to the waiter to the server. The prices were very good as well. We placed our orders once we decided what we wanted at 6:02. We shared the baked spaghetti calzone and the small "Here's The Beef" pizza so we can both try them. The calzone was huge and we got the smallest one (personal) and got the small 11" pizza. Both were awesome! My friend liked the pizza better and I liked the calzone better. The calzone does have a sweetish sauce but that's how I like my sauce!\n\nWe had to box part of the pizza to take it home and we were out the door by 6:42. So, everything was great and not like these bad reviewers. That goes to show you that you have to try these things yourself because all these bad reviewers have some serious issues.	rev

6oRAC4uyJCsJl1X0WZpVSA	2012-06-14	IESLBzqUCLdSzSqm0eCSxQ	4	love the gyro plate. Rice is so good and I also dig their candy selection :)	rev
_1QQZuf4zZOyFCvXc0o6Vg	2010-05-27	G-WvGaISbqqaMHlNnByodA	5	Rosie, Dakota, and I LOVE Chaparral Dog Park!!! It's very convenient and surrounded by a lot of paths, a desert xeriscape, baseball fields, ballparks, and a lake with ducks.\n\nThe Scottsdale Park and Rec Dept. does a wonderful job of keeping the park clean and shaded. You can find trash cans and poopy-pick up mitts located all over the park and paths.\n\nThe fenced in area is huge to let the dogs run, play, and sniff!	rev
6ozycU1RpktNG2-1BroVtw	2012-01-05	1uJFq2r5QfJG_6ExMRCaGw	5	General Manager Scott Petello is a good egg!!! Not to go into detail, but let me assure you if you have any issues (albeit rare) speak with Scott and treat the guy with some respect as you state your case and I'd be surprised if you don't walk out totally satisfied as I just did. Like I always say..... "Mistakes are inevitable, it's how we recover from them that is important"!!!\n\nThanks to Scott and his awesome staff. You've got a customer for life!! .... :^)	rev

Convert five classes into two classes (positive = 1 and negative = 0)

Since the main purpose is to identify positive or negative comments, I convert five class star category into two classes:

- (1) Positive: comments with stars > 3 and
- (2) Negative: comments with stars <= 3

In [6]:
```
labels = df['stars'].map(lambda x : 1 if int(x) > 3 else 0)
```

Tokenize text data

Because of the computational expenses, I use the top 20000 unique words. First, tokenize the comments then convert those into sequences. I keep 50 words to limit the number of words in each comment.

In [7]:
```
def clean_text(text):

    ## Remove puncuation
    text = text.translate(string.punctuation)

    ## Convert words to lower case and split them
    text = text.lower().split()

    ## Remove stop words
    stops = set(stopwords.words("english"))
    text = [w for w in text if not w in stops and len(w) >= 3]

    text = " ".join(text)

    # Clean the text
    text = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", text)
    text = re.sub(r"what's", "what is ", text)
    text = re.sub(r"\'s", " ", text)
    text = re.sub(r"\'ve", " have ", text)
    text = re.sub(r"n't", " not ", text)
    text = re.sub(r"i'm", "i am ", text)
    text = re.sub(r"\'re", " are ", text)
    text = re.sub(r"\'d", " would ", text)
    text = re.sub(r"\'ll", " will ", text)
    text = re.sub(r",", " ", text)
    text = re.sub(r"\.", " ", text)
    text = re.sub(r"!", " ! ", text)
    text = re.sub(r"\/", " ", text)
    text = re.sub(r"\^", " ^ ", text)
    text = re.sub(r"\+", " + ", text)
    text = re.sub(r"\-", " - ", text)
    text = re.sub(r"\=", " = ", text)
    text = re.sub(r"'", " ", text)
    text = re.sub(r"(\d+)(k)", r"\g<1>000", text)
    text = re.sub(r":", " : ", text)
    text = re.sub(r" e g ", " eg ", text)
    text = re.sub(r" b g ", " bg ", text)
    text = re.sub(r" u s ", " american ", text)
    text = re.sub(r"\0s", "0", text)
    text = re.sub(r" 9 11 ", "911", text)
    text = re.sub(r"e - mail", "email", text)
    text = re.sub(r"j k", "jk", text)
    text = re.sub(r"\s{2,}", " ", text)

    text = text.split()
    stemmer = SnowballStemmer('english')
    stemmed_words = [stemmer.stem(word) for word in text]
    text = " ".join(stemmed_words)

    return text
```

In [8]:
```
df['text'] = df['text'].map(lambda x: clean_text(x))
```

In [9]:
```
df.head(10)
```

Out[9]:

9yKzy9PApeiPPOUJEtnvkg	2011-01-26	fWKvX83p0-ka4JS3dc6E5A	5	My wife took me here birthday for breakfast was excellent. The w was perfect which sitting outside overl their grounds an ab pleasure. Our waitre excellent and our food quickly on the sem Saturday morning. It like the place fills up quickly so the earlier y here the better.\n\nDo y a favor and get their Mary. It was phenomer simply the best I've eve I'm pretty sure they or ingredients from their and blend them fresh

In [10]:
```
vocabulary_size = 20000
tokenizer = Tokenizer(num_words= vocabulary_size)
tokenizer.fit_on_texts(df['text'])

sequences = tokenizer.texts_to_sequences(df['text'])
data = pad_sequences(sequences, maxlen=50)
```

In [11]:
```
print(data.shape)
```

```
(10000, 50)
```

Build neural network with LSTM

Network Architechture

The network starts with an embedding layer. The layer lets the system expand each token to a more massive vector, allowing the network to represent a word in a meaningful way. The layer takes 20000 as the first argument, which is the size of our vocabulary, and 100 as the second input parameter, which is the dimension of the embeddings. The third parameter is the input_length of 50, which is the length of each comment sequence.

In [12]:
```
model_lstm = Sequential()
model_lstm.add(Embedding(20000, 100, input_length=50))
model_lstm.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model_lstm.add(Dense(1, activation='sigmoid'))
model_lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['ac
```

```
WARNING:tensorflow:From c:\users\finan\appdata\local\programs\python\python
37\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: col
ocate_with (from tensorflow.python.framework.ops) is deprecated and will be
removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From c:\users\finan\appdata\local\programs\python\python
37\lib\site-packages\keras\backend\tensorflow_backend.py:3445: calling drop
out (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and wi
ll be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
```

Train the network

There are about 1.6 million comments, and it takes a while to train the model in a MacBook Pro. To save time I have used only three epochs. GPU machines can be used to accelerate the training with more time steps. I split the whole datasets as 60% for training and 40% for validation.

In [13]:
```
model_lstm.fit(data, np.array(labels), validation_split=0.4, epochs=3)
```

```
WARNING:tensorflow:From c:\users\finan\appdata\local\programs\python\python
37\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a futu
re version.
Instructions for updating:
Use tf.cast instead.
Train on 6000 samples, validate on 4000 samples
Epoch 1/3
6000/6000 [==============================] - 15s 2ms/step - loss: 0.3463 -
acc: 0.8957 - val_loss: 0.3159 - val_acc: 0.9058
Epoch 2/3
6000/6000 [==============================] - 13s 2ms/step - loss: 0.3306 -
acc: 0.8982 - val_loss: 0.3137 - val_acc: 0.9058
Epoch 3/3
6000/6000 [==============================] - 13s 2ms/step - loss: 0.3316 -
acc: 0.8982 - val_loss: 0.3123 - val_acc: 0.9058
```

Out[13]:
```
<keras.callbacks.History at 0x201de768eb8>
```

Build neural network with LSTM and CNN

The LSTM model worked well. However, it takes forever to train three epochs. One way to speed up the training time is to improve the network adding “Convolutional” layer. Convolutional Neural Networks (CNN) come from image processing. They pass a “filter” over the data and calculate a higher-level representation. They have been shown to work surprisingly well for text, even though they have none of the sequence processing ability of LSTMs.

In [14]:
```
def create_conv_model():
    model_conv = Sequential()
    model_conv.add(Embedding(vocabulary_size, 100, input_length=50))
    model_conv.add(Dropout(0.2))
    model_conv.add(Conv1D(64, 5, activation='relu'))
    model_conv.add(MaxPooling1D(pool_size=4))
    model_conv.add(LSTM(100))
    model_conv.add(Dense(1, activation='sigmoid'))
    model_conv.compile(loss='binary_crossentropy', optimizer='adam', metrics=
    return model_conv
```

In [15]:
```
model_conv = create_conv_model()
model_conv.fit(data, np.array(labels), validation_split=0.4, epochs = 3)
```

```
Train on 6000 samples, validate on 4000 samples
Epoch 1/3
6000/6000 [==============================] - 12s 2ms/step - loss: 0.3458 -
acc: 0.8938 - val_loss: 0.3141 - val_acc: 0.9058
Epoch 2/3
6000/6000 [==============================] - 11s 2ms/step - loss: 0.3305 -
acc: 0.8982 - val_loss: 0.3125 - val_acc: 0.9058
Epoch 3/3
6000/6000 [==============================] - 11s 2ms/step - loss: 0.3300 -
acc: 0.8982 - val_loss: 0.3132 - val_acc: 0.9058
```

Out[15]:
```
<keras.callbacks.History at 0x201dff0e400>
```

Save processed Data

In [16]:
```
df_save = pd.DataFrame(data)
df_label = pd.DataFrame(np.array(labels))
```

In [17]:
```
result = pd.concat([df_save, df_label], axis = 1)
```

In [18]:
```
result.to_csv('train_dense_word_vectors.csv', index=False)
```

Use pre-trained Glove word embeddings

In this subsection, I want to use word embeddings from pre-trained Glove. It was trained on a dataset of one billion tokens (words) with a vocabulary of 400 thousand words. The glove has embedding vector sizes, including 50, 100, 200 and 300 dimensions. I chose the 100-dimensional version. I also want to see the model behavior in case the learned word weights do not get updated. I, therefore, set the trainable attribute for the model to be False.

Get embeddings from Glove

In [19]:
```
embeddings_index = dict()
f = open('glove.6B.100d.txt', encoding="utf8")
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))
```

```
Loaded 400000 word vectors.
```

In [20]:
```
# create a weight matrix for words in training docs
embedding_matrix = np.zeros((vocabulary_size, 100))
for word, index in tokenizer.word_index.items():
    if index > vocabulary_size - 1:
        break
    else:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[index] = embedding_vector
```

Develop model

I use the same model architecture with a convolutional layer on top of the LSTM layer.

In [21]:
```
model_glove = Sequential()
model_glove.add(Embedding(vocabulary_size, 100, input_length=50, weights=[emb
model_glove.add(Dropout(0.2))
model_glove.add(Conv1D(64, 5, activation='relu'))
model_glove.add(MaxPooling1D(pool_size=4))
model_glove.add(LSTM(100))
model_glove.add(Dense(1, activation='sigmoid'))
model_glove.compile(loss='binary_crossentropy', optimizer='adam', metrics=['a
```

In [22]:
```
model_glove.fit(data, np.array(labels), validation_split=0.4, epochs = 3)
```

```
Train on 6000 samples, validate on 4000 samples
Epoch 1/3
6000/6000 [==============================] - 4s 730us/step - loss: 0.3607 -
acc: 0.8982 - val_loss: 0.3143 - val_acc: 0.9058
Epoch 2/3
6000/6000 [==============================] - 3s 581us/step - loss: 0.3340 -
acc: 0.8982 - val_loss: 0.3140 - val_acc: 0.9058
Epoch 3/3
6000/6000 [==============================] - 4s 597us/step - loss: 0.3321 -
acc: 0.8982 - val_loss: 0.3160 - val_acc: 0.9058
```

Out[22]:
```
<keras.callbacks.History at 0x201e8ebf208>
```

Word embedding visialization

In this subsection, I want to visualize word embedding weights obtained from trained models. Word embeddings with 100 dimensions are first reduced to 2 dimensions using t-SNE. Tensorflow has an excellent tool to visualize the embeddings in a great way, but here I just want to visualize the word relationship.

Get embedding weights from glove

In [23]:
```
lstm_embds = model_lstm.layers[0].get_weights()[0]
```

In [24]:
```
conv_embds = model_conv.layers[0].get_weights()[0]
```

In [25]:
```
glove_emds = model_glove.layers[0].get_weights()[0]
```

Get word list

In [26]:
```
word_list = []
for word, i in tokenizer.word_index.items():
    word_list.append(word)
```

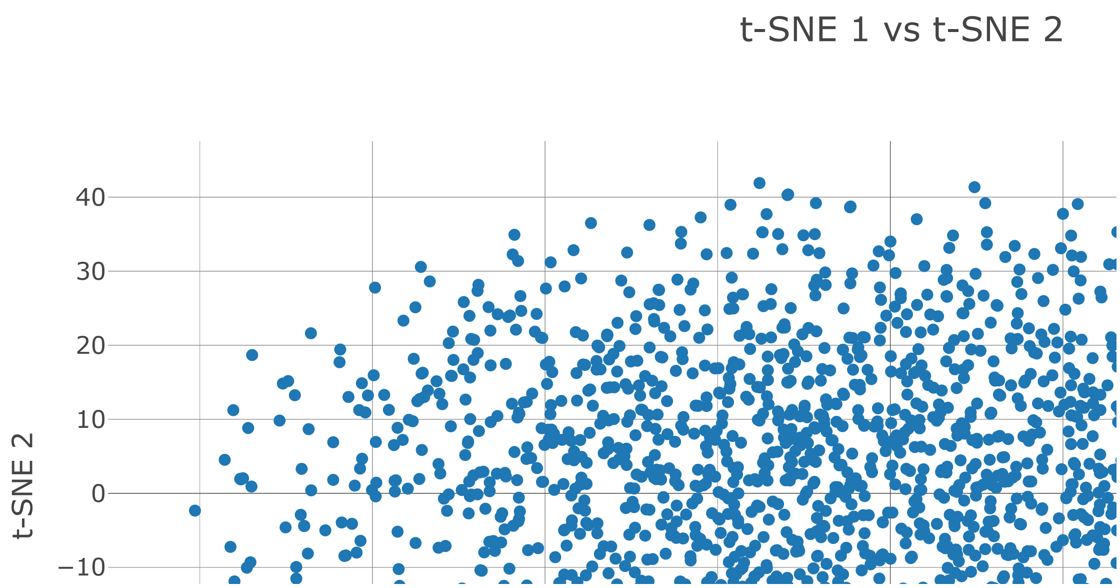
Scatter plot of first two components of TSNE

In [27]:
```
def plot_words(data, start, stop, step):
    trace = go.Scatter(
        x = data[start:stop:step,0],
        y = data[start:stop:step, 1],
        mode = 'markers',
        text= word_list[start:stop:step]
    )
    layout = dict(title= 't-SNE 1 vs t-SNE 2',
                  yaxis = dict(title='t-SNE 2'),
                  xaxis = dict(title='t-SNE 1'),
                  hovermode= 'closest')
    fig = dict(data = [trace], layout= layout)
    py.iplot(fig)
```

1. LSTM

In [28]:
```
lstm_tsne_embds = TSNE(n_components=2).fit_transform(lstm_embds)
```

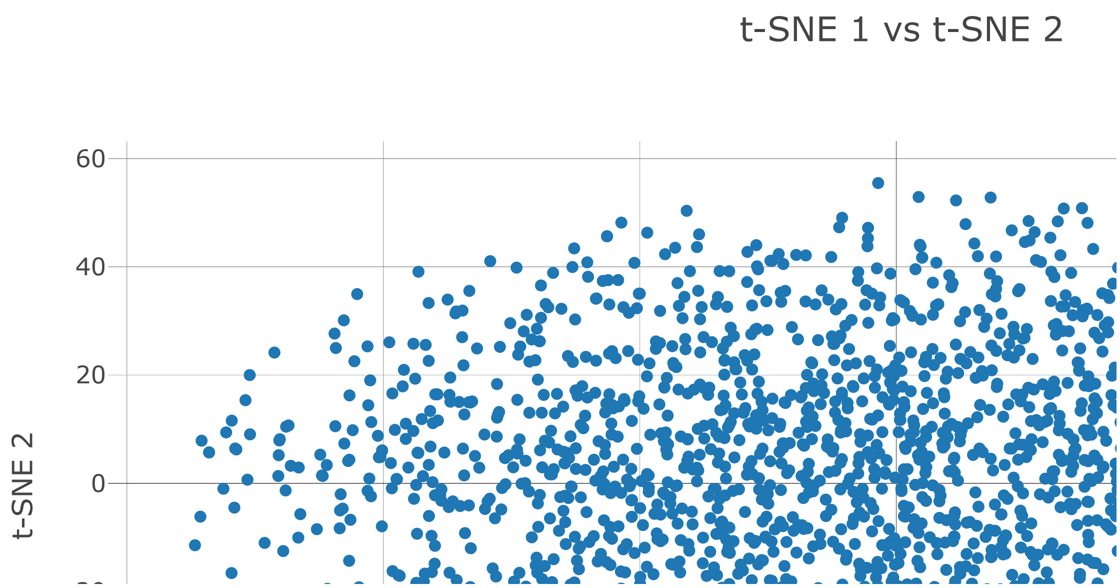
In [29]:
```
plot_words(lstm_tsne_embds, 0, 2000, 1)
```



2. CNN + LSTM

In [30]:
```
conv_tsne_embds = TSNE(n_components=2).fit_transform(conv_embds)
```

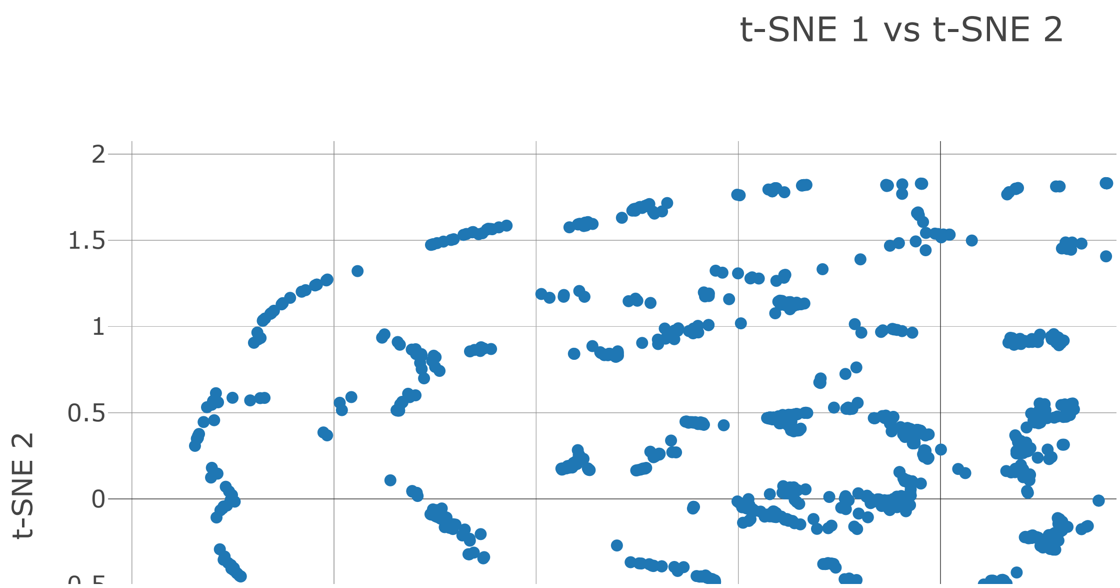
In [31]:
```
plot_words(conv_tsne_embds, 0, 2000, 1)
```



3. Glove

In [32]:
```
glove_tsne_embds = TSNE(n_components=2).fit_transform(glove_emds)
```

In [33]:
```
plot_words(glove_tsne_embds, 0, 2000, 1)
```



Word Embeddings from Word2Vec

In this subsection, I use word2vec to create word embeddings from the review comments. Word2vec is one algorithm for learning a word embedding from a text corpus.

In [34]:
```
from gensim.models import Word2Vec
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\finan\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[34]: True

Tokenize the reviews coments.

In [35]:
```
df['tokenized'] = df.apply(lambda row : nltk.word_tokenize(row['text']), axis
```

In [36]:
```
df.head()
```

small 11" pizza. Both were awesome! My friend liked the pizza better and I liked the calzone better. The calzone does have a sweetish sauce but that's how I like my sauce!\n\nWe had to box part of the pizza to take it home and we were out the door by 6:42. So, everything was great and not like these bad reviewers. That goes to show you that you have to try these things yourself because all these bad reviewers have some serious issues.

Train word2vec model

In [ ]:
```
model_w2v = Word2Vec(df['tokenized'], size=100)
model_w2v.build_vocab(text)
```

In [ ]:
```
X = model_w2v[model_w2v.wv.vocab]
```

Plot Word Vectors Using PCA

In [38]:
```
from sklearn.decomposition import TruncatedSVD
```

In [ ]:
```
tsvd = TruncatedSVD(n_components=5, n_iter=10)
result = tsvd.fit_transform(X)
```

In [ ]:
```
result.shape
```

In [ ]:

```
tsvd_word_list = []
words = list(model_w2v.wv.vocab)
for i, word in enumerate(words):
    tsvd_word_list.append(word)

trace = go.Scatter(
    x = result[0:number_of_words, 0],
    y = result[0:number_of_words, 1],
    mode = 'markers',
    text= tsvd_word_list[0:number_of_words]
)

layout = dict(title= 'SVD 1 vs SVD 2',
              yaxis = dict(title='SVD 2'),
              xaxis = dict(title='SVD 1'),
              hovermode= 'closest')

fig = dict(data = [trace], layout= layout)
py.iplot(fig)
```

In [ ]:

```

```