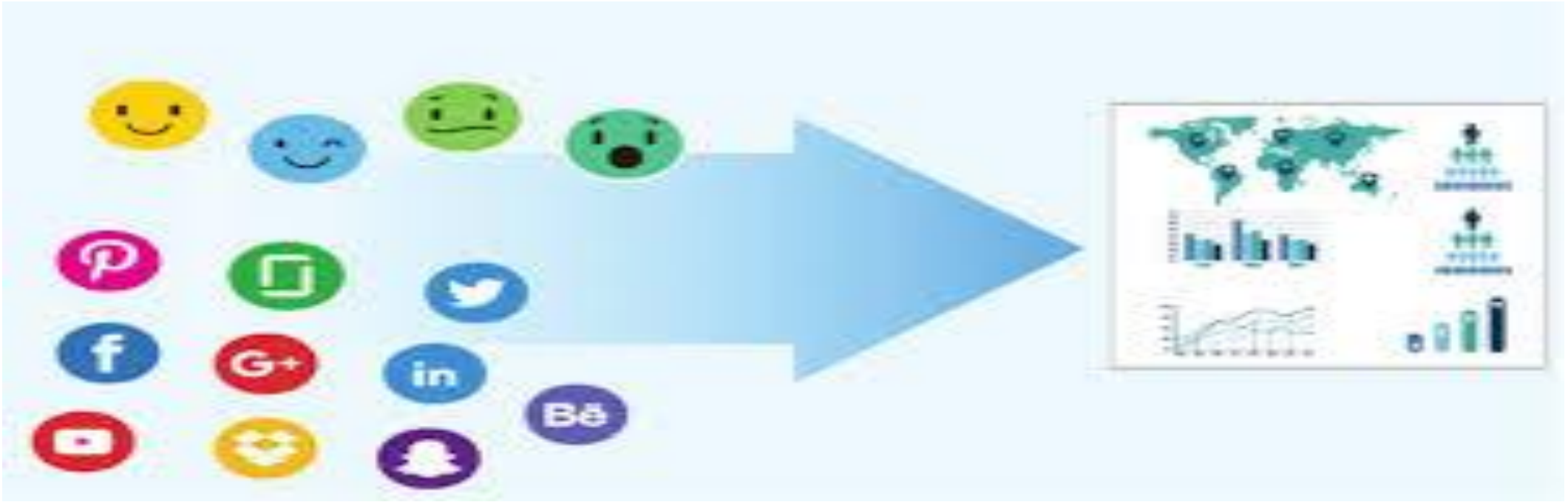
Sentiment Analysis

STUDENT C

What is sentiment analysis?

Sentiment Analysis also known as *Opinion Mining* is a field within Natural Language Processing (NLP) that builds systems that try to identify and extract opinions within text.



Use cases for sentiment analysis

Customer Experience Management (CXM)

reduce churn, increase sales

Workforce Analytics & Voice of Employee

"crazy" good or bad?

Voice of Customer (VoC)

grow market share, build loyalty

Product Management

meet market needs

Social Media Monitoring (SMM)

grow brand awareness

Enterprise Search

turn words into money

Pros of sentiment analysis

Sentiment analysis works best with **large data sets written in the first person, where the nature of the data invites the author to offer a clear opinion**.

- Great for **quickly analyzing** thousands—or even millions—of pieces of data where topic categorization is less important than an overall indication of sentiment.
- Can give you **a starting point** in qualitative data analyis by extracting strongly positive or negative sentences out of documents.
- Works particularly well with data where the author **clearly expresses an opinion** (e.g. app reviews, political views, user feedback).
- Somewhat **context-agnostic** – it doesn't matter if the data is about politics, mobile phone reviews, cooking recipes, or anything.
- Some providers (e.g. Google and Amazon) have support for **multiple languages**.

Cons of sentiment analysis

- **Not a replacement for ML auto-categorization** as it will only categorize text based on its sentiment, not the topic discussed.
- Does not work well on **text written in the third person** (e.g. user testing observations) or where the data is not someone's opinion on a product or service.
- Can struggle with **complex sentences** involving double negatives, sarcasm, adverbials, unknown proper nouns and brand names, and greetings (e.g. "Best wishes!" or "Looking forward to your response" in email signatures).


Text Input
Tokenization
Stop Word Filtering
Negation Handling
Stemming
Classification
Sentiment Class
Sentiment Analysis

The Importance of Good Preprocessing

Methods are discussed in relation to Sentiment Analysis

Procedures and Methods

- Clean up data
- Break up Contractions
- Make text lower case
- Convert numbers to text
- Look at your data and make decisions
- Lemmatization vs. Stemming

Clean Up Data

- Depending on your data source text data can be very difficult to parse.

- Examples:
  - BeautifulSoup is the most popular package for web page scraping
  - Remove Text between square brackets
    - re.sub('\[[^]]*\]', '', text)

Break Up Contractions

- By breaking up these contractions the NN can receive more unified data allowing it to form stronger connections.

- Examples
  - Don't -> Do not
  - Can't -> cannot

Make Words Lowercase

- Words at the start of sentences are capitalized.
- If we do not set all words to lowercase, then we may have 2 separate data points for the same word.

- One downside to this is some words mean different things depending on their capitalization.
  - US to abbreviate United States and the word "us" have very different purposes.

Convert Numbers to Text

A simple technique to maintain consistent data structure.

Python Example Code:

```
def replace_numbers(words):
    p = inflect.engine()
    new_words = []
    for word in words:
        if word.isdigit():
            new_word = p.number_to_words(word)
            new_words.append(new_word)
        else:
            new_words.append(word)
    return new_words
```

Remove Stopwords

This is a common step that is used to reduce the volume of words in the data that do not add value

A list of stop words:

```
a about after all also always am an and any are at be been being but by came can
cant come could did didnt do does doesnt doing dont else for from get give goes
going had happen has have having how i if ill im in into is isnt it its ive just
keep let like made make many may me mean more most much no not now of only or our
really say see some something take tell than that the their them then there they
thing this to try up us use used uses very want was way we what when where which
who why will with without wont you your youre
```

Look at Data and Make Decisions

One method of view your text data is observing the word usage frequency.

Examples of Insights:

- Identify removable words
- Identify skew in data
- Find common misspellings
- Find combinable data points
  - Proper nouns with multiple names

Lemmatization vs Stemming

Stemming is the process of eliminating affixes (suffixed, prefixes, infixes, circumfixes) from a word in order to obtain a word stem.

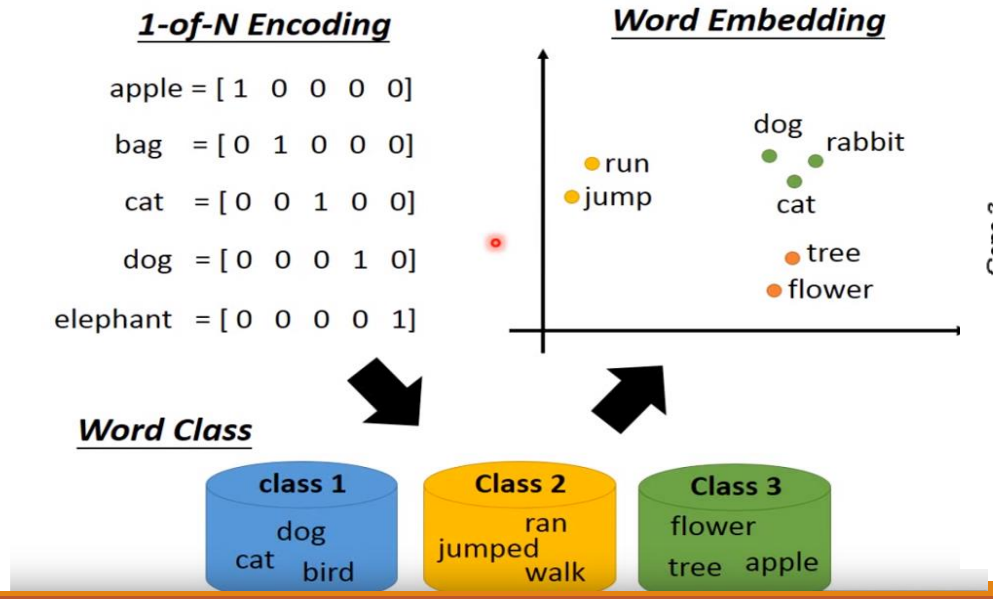
running → run

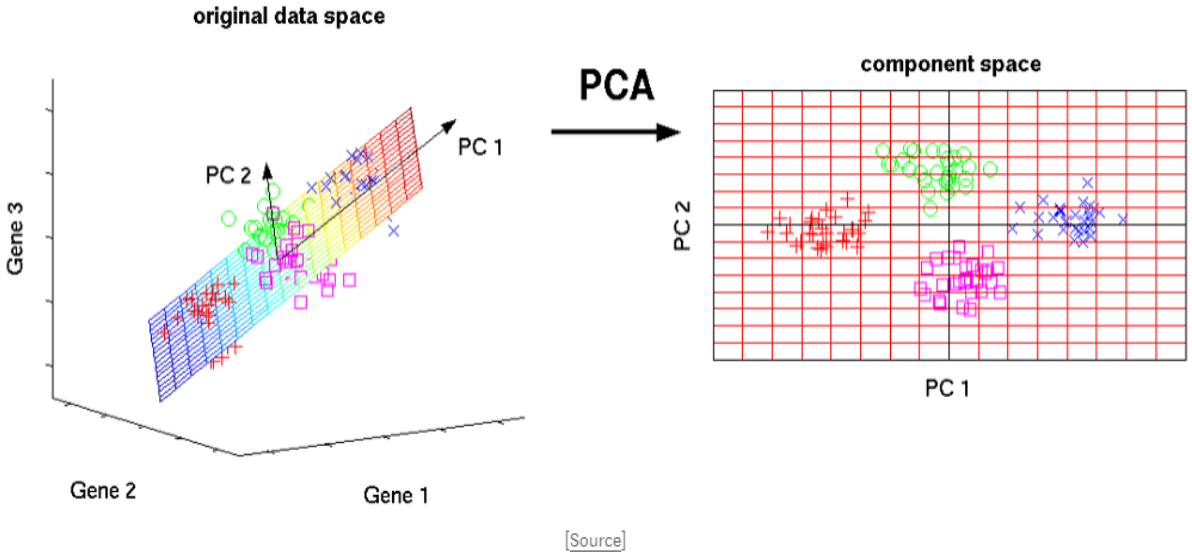
Lemmatization captures canonical forms based on a word's lemma.

better → good

Word Embedding

Word embedding is the collective name for a set of language modeling and feature learning techniques in NLP where words or phrases from the vocabulary are mapped to vectors of real numbers. It involves a mathematical embedding from a space with one dimension per word to a continuous vector space with a much lower dimension. Methods to generate this mapping include neural networks, dimensionality reduction on the word co-occurrence matrix, probabilistic models, etc.

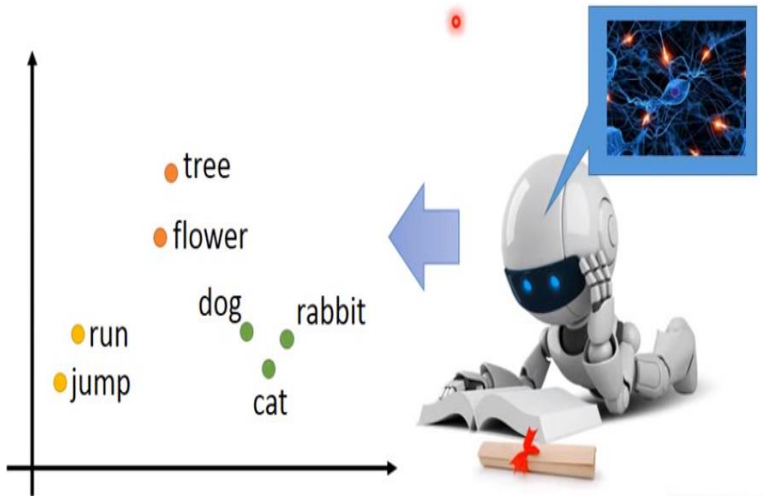




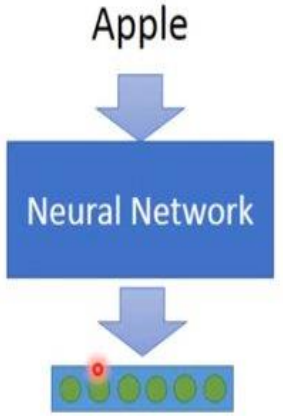
[Source]

Word Embedding

- Machine learn the meaning of words from reading a lot of documents without supervision



- Generating Word Vector is unsupervised



Training data is a lot of text

President Barack Obama **gave an inauguration speech**
President Donald Trump **gave an inauguration speech**

Word Embedding

Word2Vec data generation (skip gram) (window size = 2)

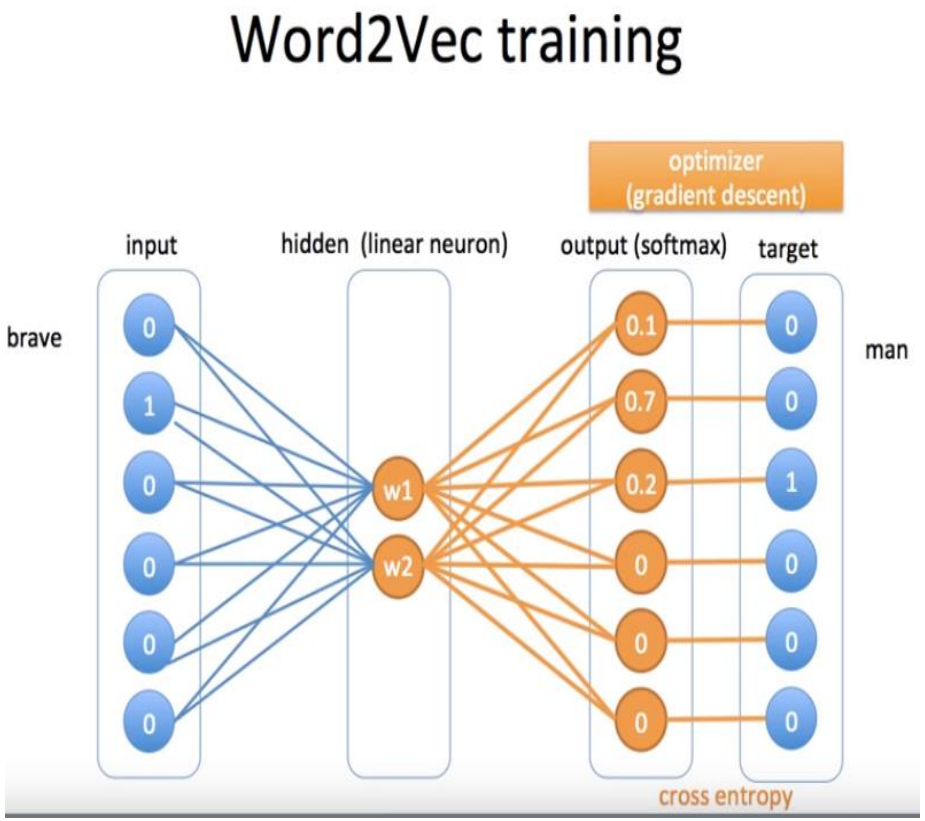
"king brave man"
"queen beautiful woman"

word	neighbor
king	brave
king	man
brave	king
brave	man
man	king
man	brave
queen	beautiful
queen	woman
beautiful	queen
beautiful	woman
woman	queen
woman	beautiful

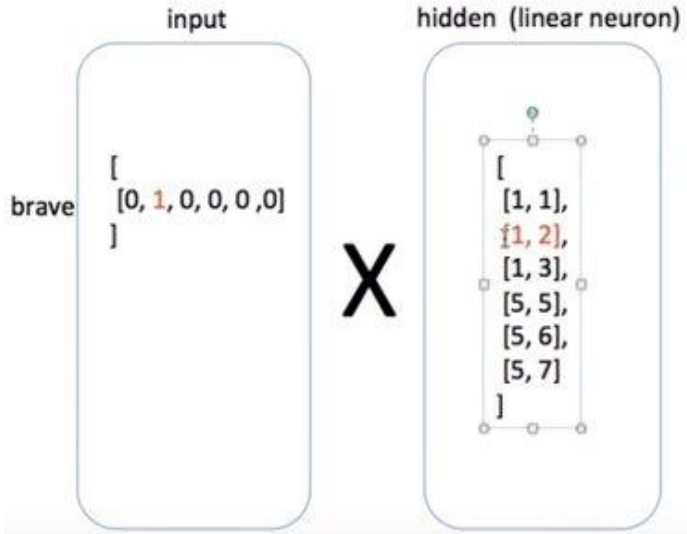
Word2Vec data generation (window size = 2)

word	word one hot encoding	neighbor	neighbor one hot encoding
king	[1, 0, 0, 0, 0, 0]	brave	[0, 1, 0, 0, 0, 0]
king	[1, 0, 0, 0, 0, 0]	man	[0, 0, 1, 0, 0, 0]
brave	[0, 1, 0, 0, 0, 0]	king	[1, 0, 0, 0, 0, 0]
brave	[0, 1, 0, 0, 0, 0]	man	[0, 0, 1, 0, 0, 0]
man	[0, 0, 1, 0, 0, 0]	king	[1, 0, 0, 0, 0, 0]
man	[0, 0, 1, 0, 0, 0]	brave	[0, 1, 0, 0, 0, 0]
queen	[0, 0, 0, 1, 0, 0]	beautiful	[0, 0, 0, 0, 1, 0]
queen	[0, 0, 0, 1, 0, 0]	woman	[0, 0, 0, 0, 0, 1]
beautiful	[0, 0, 0, 0, 1, 0]	queen	[0, 0, 0, 1, 0, 0]
beautiful	[0, 0, 0, 0, 1, 0]	woman	[0, 0, 0, 0, 0, 1]
woman	[0, 0, 0, 0, 0, 1]	queen	[0, 0, 0, 1, 0, 0]
woman	[0, 0, 0, 0, 0, 1]	beautiful	[0, 0, 0, 0, 1, 0]

Word Embedding



because input is one hot encoding,
hidden layer works as lookup table

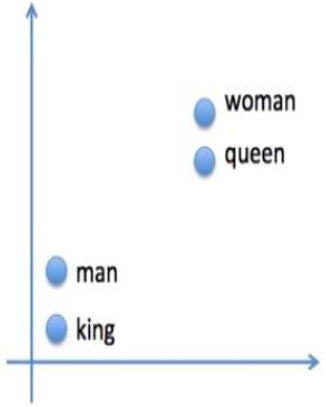


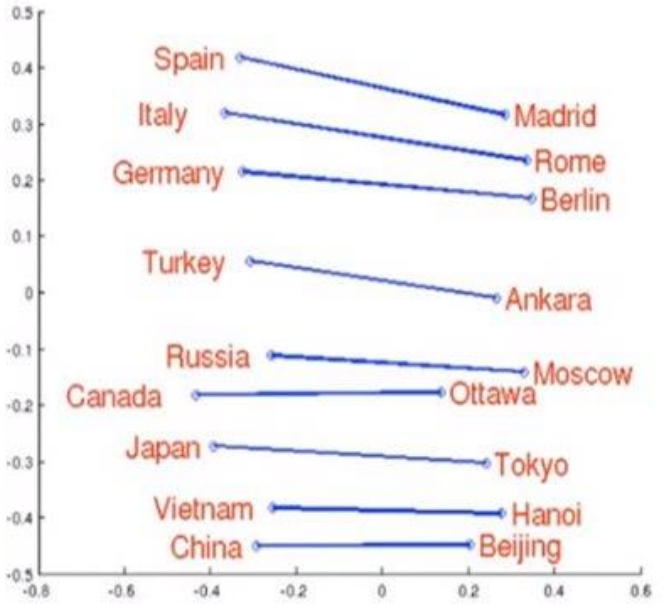
unique word	embedding
king	[1, 1]
brave	[1, 2]
man	[1, 3]
queen	[5, 5]
beautiful	[5, 6]
woman	[5, 7]

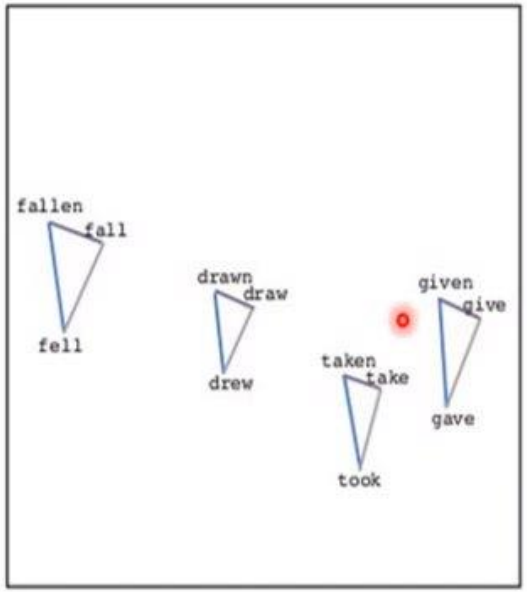
Word Embedding

Word2Vec gives similarity in vector representation

unique word	encoding	word2vec embedding
king	[1, 0, 0, 0, 0, 0]	[1, 1]
man	[0, 0, 0, 1, 0, 0]	[1, 3]
queen	[0, 0, 0, 1, 0, 0]	[5, 5]
woman	[0, 0, 0, 0, 0, 1]	[5, 7]







Comparison of GloVe(Count base) and Word2Vec(Predictive base)
http://clic.cimec.unitn.it/marco/publications/acl2014/baroni-etal-countpredict-acl2014.pdf

Model I

```
#LSTM 2
print("fitting LSTM 1 ...")
model1 = Sequential()
model1.add(Embedding(dictionary_size, 128))
model1.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model1.add(Dense(num_labels, activation='softmax'))

model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model1_result=model1.fit(word_id_train, y_train_enc, nb_epoch=5, batch_size=512, verbose=1,
                         validation_data=(word_id_test, y_test_enc))
```

```
Epoch 1/6
400000/400000 [==============================] - 9186s 23ms/step - loss: 0.8024 - acc: 0.6717
Epoch 2/6
400000/400000 [==============================] - 8991s 22ms/step - loss: 0.6754 - acc: 0.7204
Epoch 3/6
400000/400000 [==============================] - 9074s 23ms/step - loss: 0.6146 - acc: 0.7466
Epoch 4/6
400000/400000 [==============================] - 9058s 23ms/step - loss: 0.5574 - acc: 0.7731
Epoch 5/6
400000/400000 [==============================] - 9071s 23ms/step - loss: 0.4966 - acc: 0.8004
Epoch 6/6
400000/400000 [==============================] - 9079s 23ms/step - loss: 0.4339 - acc: 0.8269

In [9]: score = model.evaluate(word_id_test, y_test_enc)
   ...: print('Test loss:', score[0])
   ...: print('Test accuracy:', score[1])
100000/100000 [==============================] - 236s 2ms/step
Test loss: 0.8124536835813523
Test accuracy: 0.7006
```

Model II

```
embedding = dict();
with open('C:/Users/mtsai1/OneDrive for Business/School/Deep_Learning/Project 2/glove.6B.100d.txt',encoding="utf8") as file:
    for line in file:
        values = line.split()
        word = values[0]
        coef = np.asarray(values[1:], dtype='float32')
        embedding[word] = coef

embedding_m = np.zeros((dictionary_size, 100));
for word, i in dictionary.items():
    embedding_v = embedding.get(word);
    if embedding_v is not None:
        embedding_m[i] = embedding_v;

model3 = Sequential();
model3.add(Embedding(dictionary_size, 100, weights=[embedding_m], trainable=False));
model3.add(LSTM(60, return_sequences=True, recurrent_dropout=0.5));
model3.add(Dropout(0.5))
model3.add(LSTM(60, recurrent_dropout=0.5));
model3.add(Dense(60, activation='relu'));
model3.add(Dense(num_labels, activation='softmax'));
optimizer = optimizers.Adam(lr=0.01, decay=0.001);
model3.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
model3.fit(word_id_train, y_train_enc, batch_size = 512, validation_split=0.2, epochs=2, verbose=1,
           validation_data=(word_id_test, y_test_enc))

score = model3.evaluate(word_id_test, y_test_enc)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Model II

```
In [61]: model3 = Sequential();
    ...: model3.add(Embedding(dictionary_size, 100, weights=[embedding_m], trainable=False));
    ...: model3.add(LSTM(60, return_sequences=True, recurrent_dropout=0.5));
    ...: model3.add(Dropout(0.5))
    ...: model3.add(LSTM(60, recurrent_dropout=0.5));
    ...: model3.add(Dense(60, activation='relu'));
    ...: model3.add(Dense(num_labels, activation='softmax'));
    ...: optimizer = optimizers.Adam(lr=0.01, decay=0.001);
    ...: model3.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    ...: model3.fit(word_id_train, y_train_enc, batch_size = 512, validation_split=0.2, epochs=2, verbose=1,
    ...:             validation_data=(word_id_test, y_test_enc))
Train on 400000 samples, validate on 100000 samples
Epoch 1/2
400000/400000 [==============================] - 6780s 17ms/step - loss: 0.4481 - acc: 0.8000 - val_loss: 0.4444 -
val_acc: 0.8000
Epoch 2/2
400000/400000 [==============================] - 6997s 17ms/step - loss: 0.4459 - acc: 0.8000 - val_loss: 0.4445 -
val_acc: 0.8000
Out[61]: <keras.callbacks.History at 0x1e1ec7465c0>

In [62]: score = model3.evaluate(word_id_test, y_test_enc)
    ...: print('Test loss:', score[0])
    ...: print('Test accuracy:', score[1])
100000/100000 [==============================] - 123s 1ms/step
Test loss: 0.4444561959075928
Test accuracy: 0.800000011920929
```

Save and Retrain Model

```
#LSTM2
model2 = Sequential()
model2.add(Embedding(dictionary_size, 128))
model2.add(LSTM(60, return_sequences=True, recurrent_dropout=0.5))
model2.add(Dropout(0.5))
model2.add(LSTM(60, recurrent_dropout=0.5))
model2.add(Dense(num_labels, activation='softmax'))
optimizer = optimizers.Adam(lr=0.01, decay=0.001);
model2.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])

# Save Model
filepath = "C:/Users/mtsai1/OneDrive for Business/School/Deep_Learning/Project 2/Sen.models.h5"
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, mode='min')
callbacks_list = [checkpoint]

# fit model
model2.fit(word_id_train, y_train_enc, batch_size = 64, validation_split=0.3, epochs=1, verbose=1,
           validation_data=(word_id_test, y_test_enc), callbacks=callbacks_list)

score = model2.evaluate(word_id_test, y_test_enc)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

# load the Model
new_model = load_model("C:/Users/mtsai1/OneDrive for Business/School/Deep_Learning/Project 2/Sen.models.h5")
#test if two numpy arrays are equal (check if the new model is the right one)
assert_allclose(model2.predict(word_id_train),
                new_model.predict(word_id_train),
                1e-5)
# fit the New Model
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, mode='min')
callbacks_list = [checkpoint]
new_model.fit(word_id_train, y_train_enc, epochs=1, batch_size=64, validation_split=0.3,
              validation_data=(word_id_test, y_test_enc), callbacks=callbacks_list)

```

Save and Retrain Model

```
In [68]:

In [68]: model2.fit(word_id_train, y_train_enc, batch_size = 64, validation_split=0.3, epochs=1, verbose=1,
    ...:             validation_data=(word_id_test, y_test_enc), callbacks=callbacks_list)
Train on 400000 samples, validate on 100000 samples
Epoch 1/1
400000/400000 [==============================] - 6064s 15ms/step - loss: 0.2624 - acc: 0.8838 - val_loss: 0.2458 - val_acc: 0.8897

Epoch 00001: loss improved from inf to 0.26240, saving model to C:/Users/mtsai1/OneDrive for Business/School/Deep_Learning/Project 2/
Sen.models.h5
Out[68]: <keras.callbacks.History at 0x1e1b7806d30>
```

```
In [72]:

In [72]: checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, mode='min')
    ...: callbacks_list = [checkpoint]
    ...: new_model.fit(word_id_train, y_train_enc, epochs=1, batch_size=64, validation_split=0.3,
    ...:               validation_data=(word_id_test, y_test_enc), callbacks=callbacks_list)
Train on 400000 samples, validate on 100000 samples
Epoch 1/1
   832/400000 [..............................] - ETA: 2:12:55 - loss: 0.2577 - acc: 0.8868
```