skipGram

August 17, 2017

1 In this project we will try to build and train a skip gram model to obtain vectors for words in the dataset: (word2vec)

```
In [62]: import pandas as pd
         data = pd.read_csv("../datasets/posts.csv" )

         print data.head(3)

         # lets use sample data
         data = data.iloc[0:100,:]
```

```
                  Title                 Date       Author Categories  \
0  isight vs. firewire   Jan 30 2004 12:00AM  sean bonner
1            idvd help   Jan 30 2004 12:00AM  sean bonner
2    macintosh stories   Jan 30 2004 12:00AM  sean bonner

                                              Post Post_Length  \
0   macfixit has just posted a little  troublesho...         427
1   james over at macmerc has just posted a super...         311
2   folklore.org is collecting stories and anecdo...         272

  No_of_outlinks No_of_inlinks No_of_comments
0              2             0              0
1              3             0             12
2              3             0              0
```

Extract features: (for every word with a window of size 2)

Ex sentence: relationships can be a tressure. (relationships, can) (relationships, be) (can, relationships) (can, be)

```
In [63]: word_tuples = []
         def genTuples(tokens, window_size = 2):
             for index, token in enumerate(tokens):
                 start = index-window_size
                 end = index+window_size+1 if index+window_size+1 <= len(tokens)  else len(token
                 for neighbor in range(start, end):
```

```
                    if(neighbor!= index and neighbor>=0):
                        word_tuples.append((token, tokens[neighbor]))
            return
```

1.1 clean the features:

```
In [64]: # create a dict of unigrams and assign an id
         word_index = {}
         token_unique_id=0


         # tokenize : sentence level(.) and word level(" ")
         from nltk.tokenize import sent_tokenize
         from nltk.tokenize import word_tokenize
         from nltk.corpus import stopwords
         stop_words = set(stopwords.words('english'))

         import re
         import string
         regex = re.compile('[%s]' % re.escape(string.punctuation))


         # tokenize docs and skip unecessary words
         tokenized_docs = []
         exceptions = 0
         for doc in data["Post"]:
             try:
                 for sentence in sent_tokenize(doc):
                     #tokenized_docs.append(map(str.lower, filter(lambda word: word not in stop_

                     # lower strings
                     tokens = map(str.lower, word_tokenize(sentence))

                     # remove stopwords
                     tokens = filter(lambda word: word not in stop_words, tokens)

                     #remove punctuation
                     tokens =  map(lambda token:regex.sub('', token), tokens)

                     #filter empty strings
                     tokens = filter(lambda token: token != '', tokens )


                     #tokenized_docs
                     tokenized_docs.append(tokens)

                     # create unique id for new tokens:
                     for token in tokens:
```

```
                    if token not in word_index.keys():
                        token_unique_id +=1
                        word_index[token] = token_unique_id

                # create word tuples for the sentence: default window size:2
                genTuples(tokens, window_size=2)
            except :
                exceptions=exceptions + 1
                continue
        print "exception occured :", exceptions


        # remove punctuations in words using regular expressions package re:


        # for sentence_index, tokens in enumerate(tokenized_docs):
        #     new_tokens = []
        #     for token in tokens:
        #         new_token = regex.sub('', token)
        #         if not new_token == '':
        #             if new_token not in word_index.keys():
        #                 token_unique_id +=1
        #                 word_index[new_token] = token_unique_id
        #             new_tokens.append(new_token)
        #     tokenized_docs[sentence_index] = new_tokens

        # create word_tuple features (x,y)


        #map(genTuples, tokenized_docs)
        print "total word tuples generated: ", len(word_tuples)
        print word_tuples[0:3], "..."
```

```
exception occured : 0
total word tuples generated:  22004
[('macfixit', 'posted'), ('macfixit', 'little'), ('posted', 'macfixit')] ...
```

1.2 There are four ways tob build a tensorflow computational graph:

1. Using the Tensorflow High Level Api: tf.contrib
2. Extending the TensorFlow : tf.estimator class
3. Using the Tensorflow.Contrib.keras
4. Tensorflow low level API: Tensorflow Core

We will be exploring how to use tensorflow core API to build a skip-gram model

1.3 Using LowLevel TensorFlow Core to build the Skip-Gram Model:

In [65]:
```
import tensorflow as tf
import numpy as np

def feedForward(X, W1, W2):
    """
       compute the forward propgation of the network and,
       return the ouput of the network
    """

    h1 = tf.matmul(X,W1)

    h2 = tf.matmul(h1, W2)

    return tf.nn.softmax(h2)


def word2Vec(word_tuples, word_index, hidden_dim = 300):

    input_dim = len(word_index.keys())
    out_dim = len(word_index.keys())

    print "input dim: ", input_dim
    print "output dim:", out_dim


    # placeholders for the graph
    X = tf.placeholder("float", shape= [None, input_dim])
    Y = tf.placeholder("float", shape= [None, out_dim])



    # Trainable variables of the graph
    # initialize weights for the synapses between input and hidden layer:
    # d * k dimensions for d dimensional input and k hidden neurons

    w1 = tf.Variable(tf.random_normal(shape = [input_dim, hidden_dim]))
    w2 = tf.Variable(tf.random_normal(shape = [hidden_dim, out_dim]))


    # feedForward: mat mul of X*w1*w2
    output = feedForward(X, w1, w2)


    # COST FUNCTION/ LOSS

    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=Y, logits=outp
```

```
    # OPTIMIZER/gradient to update weights:
    optimizer = tf.train.GradientDescentOptimizer(0.05)
    train = optimizer.minimize(loss)


    # run sgd
    sess = tf.Session()
    init = tf.global_variables_initializer()
    sess.run(init)  # all variables:weights will be reset to default/original


    # for each epoch:
    for epoch in range(1,2):

        # iterate for each example:
        for i in range(len(word_tuples)):

            # encode the word_tuple with vectors using the above id(one hot encoding):
            x,y = word_tuples[i]
            x_id = word_index[x]
            y_id = word_index[y]

            train_x = np.zeros(shape=(1,input_dim))
            train_y = np.zeros(shape=(1,out_dim))
            train_x[0][x_id-1] = 1
            train_y[0][y_id-1] = 1

            sess.run(train, {X: train_x, Y: train_y})



        # accuracy/loss for each epoch:
        print "Epoch :", epoch, " loss is",
        #print(tf.reduce_mean(sess.run(loss, {X: train_x, Y: train_y})))
        sum  = sess.run(loss, {X: train_x, Y: train_y})
        print sum
    weights = sess.run(w1)
    return weights
```

```
In [66]: word_weights = word2Vec(word_tuples, word_index, hidden_dim = 100)
```

```
input dim:  2775
output dim: 2775
Epoch : 1  loss is 7.92903
```

```
In [67]: # shape of the learnt weights
```

```
word_weights.shape
```

Out[67]: (2775, 100)

1.4 visualizing the high dimensional word vectors using tsne:

In [75]:
```
# plot using plotly-offline to display in notebooks
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot

from sklearn.manifold import TSNE

# to display plotly images in notebook enable this
init_notebook_mode(connected=True)

# reduce to 2D
tsne = TSNE(n_components=2)

# transform the generated word2vec vectors
X_tsne = tsne.fit_transform(word_weights)

N = len(word_weights)
random_x = X_tsne[:, 0]
random_y = X_tsne[:, 1]


# Plot and embed in ipython notebook!
#iplot({ 'data': [{"x": list(X_tsne[:, 0]), "y": list(X_tsne[:, 1])}] })
#iplot({ 'data': [{"x": list(X_tsne[:, 0]), "y": list(X_tsne[:, 1])}] })

# labels:

labels = [word for word,_ in word_tuples]



iplot([go.Scatter(x=list(X_tsne[:, 0]), y=list(X_tsne[:, 1]), text= labels, mode='marke
# or plot with: plot_url = py.plot(data, filename='basic-line')
```

In [77]:
```
# plot using plotly-offline to display in notebooks
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot

from sklearn.manifold import TSNE

# to display plotly images in notebook enable this
init_notebook_mode(connected=True)
```

```
# reduce to 2D
tsne = TSNE(n_components=3)

# transform the generated word2vec vectors
X_tsne = tsne.fit_transform(word_weights)

N = len(word_weights)
random_x = X_tsne[:, 0]
random_y = X_tsne[:, 1]


# Plot and embed in ipython notebook!
#iplot({ 'data': [{"x": list(X_tsne[:, 0]), "y": list(X_tsne[:, 1])}] })
#iplot({ 'data': [{"x": list(X_tsne[:, 0]), "y": list(X_tsne[:, 1])}] })

# labels:

labels = [word for word,_ in word_tuples]



iplot([go.Scatter3d(x=list(X_tsne[:, 0]), y=list(X_tsne[:, 1]), z=X_tsne[:, 2], text= l
# or plot with: plot_url = py.plot(data, filename='basic-line')
```