In [3]:

```
import os
import io
import pandas as pd
import numpy as np
import json
from collections import defaultdict
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
import seaborn as sns

plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12

import re
import urllib.parse
from PIL import Image, ImageFilter
from IPython.display import display
import zipfile
import cv2
```

In [2]:

```
import tensorflow as tf
import time
```

In [3]:

```
                   Name  Dimension                     Corpus VocabularySize  \
2          fastText(en)        300                  Wikipedia           2.5M
11         GloVe.6B.50d         50  Wikipedia+Gigaword 5 (6B)           400K
12         GloVe.6B.100d        100  Wikipedia+Gigaword 5 (6B)           400K
13         GloVe.6B.200d        200  Wikipedia+Gigaword 5 (6B)           400K
14         GloVe.6B.300d        300  Wikipedia+Gigaword 5 (6B)           400K
15         GloVe.42B.300d        300          Common Crawl(42B)           1.9M
16         GloVe.840B.300d        300         Common Crawl(840B)
           2.2M
17         GloVe.Twitter.25d         25                Twitter(27B)
           1.2M
18         GloVe.Twitter.50d         50                Twitter(27B)
           1.2M
19         GloVe.Twitter.100d        100                Twitter(27B)
           1.2M
20         GloVe.Twitter.200d        200                Twitter(27B)
           1.2M
21         word2vec.GoogleNews         300           Google News(100B)
           3.0M

      Method Language    Author
2   fastText  English  Facebook
11     GloVe  English  Stanford
12     GloVe  English  Stanford
13     GloVe  English  Stanford
14     GloVe  English  Stanford
15     GloVe  English  Stanford
16     GloVe  English  Stanford
17     GloVe  English  Stanford
18     GloVe  English  Stanford
19     GloVe  English  Stanford
20     GloVe  English  Stanford
```

```
21     word2vec  English    Google
```

Glove6B - 50D

In [9]:

```
 CHAKIN_INDEX = 11
NUMBER_OF_DIMENSIONS = 50
SUBFOLDER_NAME = "gloVe.6B"

DATA_FOLDER = "embeddings"
ZIP_FILE = os.path.join(DATA_FOLDER, "{}.zip".format(SUBFOLDER_NAME))
ZIP_FILE_ALT = "glove" + ZIP_FILE[5:]
UNZIP_FOLDER = os.path.join(DATA_FOLDER, SUBFOLDER_NAME) if
SUBFOLDER_NAME[-1] == "d":
    GLOVE_FILENAME = os.path.join(
        UNZIP_FOLDER, "{}.txt".format(SUBFOLDER_NAME)) else:
    GLOVE_FILENAME = os.path.join(UNZIP_FOLDER, "{}.{}d.txt".format(
        SUBFOLDER_NAME, NUMBER_OF_DIMENSIONS))


if not os.path.exists(ZIP_FILE) and not os.path.exists(UNZIP_FOLDER):
    print("Downloading embeddings to '{}'".format(ZIP_FILE))
    chakin.download(number=CHAKIN_INDEX, save_dir='./{}'.format(DATA_FOLDER))
else:
    print("Embeddings already downloaded.")

if not os.path.exists(UNZIP_FOLDER):
    import zipfile if not os.path.exists(ZIP_FILE) and
    os.path.exists(ZIP_FILE_ALT):
        ZIP_FILE = ZIP_FILE_ALT with
    zipfile.ZipFile(ZIP_FILE, "r") as zip_ref:
        print("Extracting embeddings to '{}'".format(UNZIP_FOLDER))
zip_ref.extractall(UNZIP_FOLDER) else:
    print("Embeddings already extracted.")
```

```
Embeddings already downloaded. Embeddings
already extracted.
```

In

```
Run complete
```

In [42]:

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import numpy as np

import os
import os.path

import re

from collections import defaultdict

import nltk
from nltk.tokenize import TreebankWordTokenizer

import tensorflow as tf

RANDOM_SEED = 9999
```

[43]:

```
def reset_graph(seed= RANDOM_SEED):
    tf.reset_default_graph()
    tf.set_random_seed(seed)
    np.random.seed(seed)

REMOVE_STOPWORDS = False
```

In [44]:

```
embeddings_directory = 'embeddings/gloVe.6B'
filename = 'glove.6B.50d.txt'
```

In [46]:

```
    def load_embedding_from_disks(embeddings_filename, with_indexes=True):
    """
    Read a embeddings txt file. If `with_indexes=True`,
we return a tuple of two dictionnaries
    `(word_to_index_dict, index_to_embedding_array)`,
otherwise we return only a direct
    `word_to_embedding_dict` dictionnary mapping
from a string to a numpy array.
    """ if
    with_indexes:
        word_to_index_dict = dict() index_to_embedding_array
        = []

    else:
        word_to_embedding_dict = dict()

      with open(embeddings_filename, 'r', encoding='utf-8') as embeddings_file:
        for (i, line) in enumerate(embeddings_file):

            split = line.split(' ')

            word = split[0]

            representation = split[1:]
            representation = np.array(
                [float(val) for val in representation]
            )

            if with_indexes:
                word_to_index_dict[word] = i
                index_to_embedding_array.append(representation)
            else:
                word_to_embedding_dict[word] = representation
    _WORD_NOT_FOUND = [0.0] * len(representation) if
    with_indexes: _LAST_INDEX = i + 1
    word_to_index_dict = defaultdict( lambda:
    _LAST_INDEX, word_to_index_dict)
        index_to_embedding_array = np.array( index_to_embedding_array
            + [_WORD_NOT_FOUND])
        return word_to_index_dict, index_to_embedding_array
    else:
        word_to_embedding_dict = defaultdict(lambda: _WORD_NOT_FOUND) return
        word_to_embedding_dict

print('\nLoading embeddings from', embeddings_filename) word_to_index,
index_to_embedding = \ load_embedding_from_disks(embeddings_filename,
with_indexes=True)
print("Embedding loaded from disks.")
```

```
Loading embeddings from embeddings/gloVe.6B\glove.6B.50d.txt Embedding
loaded from disks.
```

[47]:

```
vocab_size, embedding_dim = index_to_embedding.shape print("Embedding
is of shape: {}".format(index_to_embedding.shape)) print("This means
(number of words, number of dimensions per word)\n") print("The first
words are words that tend occur more often.")

print("Note: for unknown words, the representation is an empty vector,\n"
      "and the index is the last one. The dictionnary has a limit:") print("
{} --> {} --> {}".format("A word", "Index in embedding",
      "Representation")) word = "worsdfkljsdf" idx =
word_to_index[word] complete_vocabulary_size = idx embd =
```

In

```
list(np.array(index_to_embedding[idx], dtype=int)) print("
{} --> {} --> {}".format(word, idx, embd)) word = "the"
idx = word_to_index[word] embd =
list(index_to_embedding[idx]) print("    {} --> {} -->
{}".format(word, idx, embd))

a_typing_test_sentence = 'The quick brown fox jumps over the lazy dog'
print('\nTest sentence: ', a_typing_test_sentence, '\n') words_in_test_sentence
= a_typing_test_sentence.split()

print('Test sentence embeddings from complete vocabulary of',
      complete_vocabulary_size, 'words:\n')
for word in words_in_test_sentence:
    word_ = word.lower()
    embedding = index_to_embedding[word_to_index[word_]] print(word_
    + ": ", embedding)
```

```
Embedding is of shape: (400001, 50)
This means (number of words, number of dimensions per word)

The first words are words that tend occur more often.
Note: for unknown words, the representation is an empty vector,
and the index is the last one. The dictionnary has a limit:
A word --> Index in embedding --> Representation
    worsdfkljsdf --> 400000 --> [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0]
    the --> 0 --> [0.418, 0.24968, -0.41242, 0.1217, 0.34527, -0.044457, -0.4968
8, -0.17862, -0.00066023, -0.6566, 0.27843, -0.14767, -0.55677, 0.14658, -0.0095
095, 0.011658, 0.10204, -0.12792, -0.8443, -0.12181, -0.016801, -0.33279, -0.155
2, -0.23131, -0.19181, -1.8823, -0.76746, 0.099051, -0.42125, -0.19526, 4.0071,
-0.18594, -0.52287, -0.31681, 0.00059213, 0.0074449, 0.17778, -0.15897, 0.01204
1, -0.054223, -0.29871, -0.15749, -0.34758, -0.045637, -0.44251, 0.18785, 0.0027
849, -0.18411, -0.11514, -0.78581]

Test sentence:  The quick brown fox jumps over the lazy dog Test

sentence embeddings from complete vocabulary of 400000 words:

the:  [ 4.1800e-01  2.4968e-01 -4.1242e-01  1.2170e-01  3.4527e-01 -4.4457e-02
 -4.9688e-01 -1.7862e-01 -6.6023e-04 -6.5660e-01  2.7843e-01 -1.4767e-01
 -5.5677e-01  1.4658e-01 -9.5095e-03  1.1658e-02  1.0204e-01 -1.2792e-01
 -8.4430e-01 -1.2181e-01 -1.6801e-02 -3.3279e-01 -1.5520e-01 -2.3131e-01
 -1.9181e-01 -1.8823e+00 -7.6746e-01  9.9051e-02 -4.2125e-01 -1.9526e-01
  4.0071e+00 -1.8594e-01 -5.2287e-01 -3.1681e-01  5.9213e-04  7.4449e-03
1.7778e-01 -1.5897e-01  1.2041e-02 -5.4223e-02 -2.9871e-01 -1.5749e-01  -3.4758e-
01 -4.5637e-02 -4.4251e-01  1.8785e-01  2.7849e-03 -1.8411e-01
 -1.1514e-01 -7.8581e-01]
quick:  [ 0.13967   -0.53798   -0.18047   -0.25142    0.16203   -0.13868
 -0.24637    0.75111    0.27264    0.61035   -0.82548    0.038647
 -0.32361    0.30373   -0.14598   -0.23551    0.39267   -1.1287
 -0.23636   -1.0629     0.046277   0.29143   -0.25819   -0.094902
```

[48]:

```
def default_factory():
    return EVOCABSIZE
limited_word_to_index = defaultdict(default_factory, \
    {k: v for k, v in word_to_index.items() if v < EVOCABSIZE})

limited_index_to_embedding = index_to_embedding[0:EVOCABSIZE,:]
limited_index_to_embedding = np.append(limited_index_to_embedding,
index_to_embedding[index_to_embedding.shape[0] - 1, :].\
reshape(1,embedding_dim), axis = 0)
```

In

```
del index_to_embedding print('\nTest sentence embeddings from vocabulary of',
EVOCABSIZE, 'words:\n') for word in words_in_test_sentence:
    word_ = word.lower() embedding =
    limited_index_to_embedding[limited_word_to_index[word_]]
```

```
Test sentence embeddings from vocabulary of 10000 words:

the:  [ 4.1800e-01  2.4968e-01 -4.1242e-01  1.2170e-01  3.4527e-01 -4.4457e-02
 -4.9688e-01 -1.7862e-01 -6.6023e-04 -6.5660e-01  2.7843e-01 -1.4767e-01
 -5.5677e-01  1.4658e-01 -9.5095e-03  1.1658e-02  1.0204e-01 -1.2792e-01
 -8.4430e-01 -1.2181e-01 -1.6801e-02 -3.3279e-01 -1.5520e-01 -2.3131e-01
 -1.9181e-01 -1.8823e+00 -7.6746e-01  9.9051e-02 -4.2125e-01 -1.9526e-01
  4.0071e+00 -1.8594e-01 -5.2287e-01 -3.1681e-01  5.9213e-04  7.4449e-03
1.7778e-01 -1.5897e-01  1.2041e-02 -5.4223e-02 -2.9871e-01 -1.5749e-01  -3.4758e-
01 -4.5637e-02 -4.4251e-01  1.8785e-01  2.7849e-03 -1.8411e-01
 -1.1514e-01 -7.8581e-01]
quick:  [ 0.13967   -0.53798   -0.18047   -0.25142    0.16203   -0.13868
 -0.24637    0.75111    0.27264    0.61035   -0.82548    0.038647
 -0.32361    0.30373   -0.14598   -0.23551    0.39267   -1.1287
 -0.23636   -1.0629     0.046277   0.29143   -0.25819   -0.094902
  0.79478   -1.2095     -0.01039   -0.092086   0.84322   -0.11061
3.0096      0.51652    -0.76986    0.51074    0.37508    0.12156
0.082794   0.43605   -0.1584     -0.61048    0.35006    0.52465   -
0.51747    0.0034705  0.73625    0.16252    0.85279    0.85268
  0.57892    0.64483   ]
brown:  [-0.88497   0.71685  -0.40379  -0.10698   0.81457   1.0258   -1.2698
 -0.49382  -0.27839  -0.92251  -0.49409   0.78942  -0.20066  -0.057371
  0.060682  0.30746   0.13441  -0.49376  -0.54788  -0.81912  -0.45394
  0.52098   1.0325   -0.8584   -0.65848  -1.2736    0.23616   1.0486
  0.18442  -0.3901    2.1385   -0.45301  -0.16911  -0.46737   0.15938
-0.095071 -0.26512  -0.056479  0.63849  -1.0494    0.037507  0.76434  -
0.6412   -0.59594   0.46589   0.31494  -0.34072  -0.59167  -0.31057
  0.73274 ]
fox:  [ 0.44206   0.059552  0.15861   0.92777   0.1876    0.24256  -1.593   -
0.79847  -0.34099  -0.24021  -0.32756   0.43639  -0.11057   0.50472
  0.43853   0.19738  -0.1498   -0.046979 -0.83286   0.39878   0.062174
  0.28803   0.79134   0.31798  -0.21933  -1.1015   -0.080309  0.39122
0.19503  -0.5936    1.7921    0.3826   -0.30509  -0.58686  -0.76935  -
0.61914  -0.61771  -0.68484  -0.67919  -0.74626  -0.036646  0.78251  -
1.0072   -0.59057  -0.7849   -0.39113  -0.49727  -0.4283   -0.15204
  1.5064  ] jumps:  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0.
0.     0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0.]
over:  [ 0.12972    0.088073   0.24375    0.078102  -0.12783    0.27831
 -0.48693    0.19649   -0.39558   -0.28362   -0.47425   -0.59317
 -0.58804   -0.31702    0.49593    0.0087594  0.039613  -0.42495
 -0.97641   -0.46534    0.020675   0.086042   0.39317   -0.51255
 -0.17913   -1.8333     0.5622     0.41626    0.075127   0.02189
  3.784      0.71067   -0.073943   0.15373   -0.3853    -0.070163
 -0.35374    0.074501  -0.084228  -0.45548   -0.081068   0.39157
  0.173      0.2254    -0.12836    0.40951   -0.26079    0.090912
```

```
In
   [49]:
          def listdir_no_hidden(path):
              start_list = os.listdir(path)
              end_list = []
              for file in start_list:
                  if (not file.startswith('.')):
                      end_list.append(file)
              return(end_list)

          codelist = ['\r', '\n', '\t']

          if REMOVE_STOPWORDS :
```

```
In [54]:  import nltk

          [nltk_data] Downloading package stopwords to
          [nltk_data]     C:\Users\upsto\AppData\Roaming\nltk_data...
          [nltk_data]   Unzipping corpora\stopwords.zip. Out[54]:

    True
```

```
In [55]:   more_stop_words = ['cant','didnt','doesnt','dont','goes','isnt','hes',\
                  'shes','thats','theres','theyre','wont','youll','youre','youve', 'br'\
                  've', 're', 'vs']

          some_proper_nouns_to_remove = ['dick','ginger','hollywood','jack',\
                  'jill','john','karloff','kudrow','orson','peter','tcm','tom',\
              'toni','welles','william','wolheim','nikita'] plist =
             nltk.corpus.stopwords.words('english') + more_stop_words +\
```

```
In [56]:  def text_parse(string):

              temp_string = re.sub('[^a-zA-Z]', '  ', string)     for
              i in range(len(codelist)):
                  stopstring = ' ' + codelist[i] + '  '
                  temp_string = re.sub(stopstring, '  ', temp_string)
              temp_string = re.sub('\s.\s', ' ', temp_string)
              temp_string = temp_string.lower()    if
              REMOVE_STOPWORDS:
                  for i in range(len(stoplist)):
                       stopstring = ' ' + str(stoplist[i]) + ' '
                      temp_string = re.sub(stopstring, ' ', temp_string)
              temp_string = re.sub('\s+', ' ', temp_string)    return(temp_string)
```

```
In [65]:    dir_name = 'C:/Users/upsto/Downloads/movie-reviews-negative/movie-reviews-negative

          filenames = listdir_no_hidden(path=dir_name) num_files
          = len(filenames)

          for i in range(len(filenames)): file_exists =
              os.path.isfile(os.path.join(dir_name, filenames[i]))

          print('\nDirectory:',dir_name)    print('%d
          files found' % len(filenames))

          Directory: C:/Users/upsto/Downloads/movie-reviews-negative/movie-reviews-negativ e
          500 files found
```

In [67]:

```
def read_data(filename):

  with open(filename, encoding='utf-8') as f:
    data = tf.compat.as_str(f.read())
    data = data.lower()
    data = text_parse(data)
    data = TreebankWordTokenizer().tokenize(data)   # The Penn Treebank

  return data

negative_documents = []

print('\nProcessing document files under', dir_name)
for i in range(num_files):
    ## print(' ', filenames[i])

    words = read_data(os.path.join(dir_name, filenames[i]))

    negative_documents.append(words)
    print('Data size (Characters) (Document %d) %d' %(i,len(words)))
    print('Sample string (Document %d) %s' %(i,words[:50]))
```

```
Processing document files under C:/Users/upsto/Downloads/movie-reviews-negative/
movie-reviews-negative
Data size (Characters) (Document 0) 105
Sample string (Document 0) ['story', 'of', 'man', 'who', 'has', 'unnatural', 'fe
elings', 'for', 'pig', 'starts', 'out', 'with', 'opening', 'scene', 'that', 'is',
'terrific', 'example', 'of', 'absurd', 'comedy', 'formal', 'orchestra', 'audi
ence', 'is', 'turned', 'into', 'an', 'insane', 'violent', 'mob', 'by', 'the', 'c
razy', 'chantings', 'of', 'it', 'singers', 'unfortunately', 'it', 'stays', 'absu
rd', 'the', 'whole', 'time', 'with', 'no', 'general', 'narrative', 'eventually']
Data size (Characters) (Document 1) 114
Sample string (Document 1) ['ok', 'its', 'not', 'the', 'best', 'film', 've', 'ev
er', 'seen', 'but', 'at', 'the', 'same', 'time', 've', 'been', 'able', 'to', 'si
t', 'and', 'watch', 'it', 'twice', 'story', 'line', 'was', 'pretty', 'awful', 'a
nd', 'during', 'the', 'first', 'part', 'of', 'the', 'first', 'short', 'story', '
wondered', 'what', 'the', 'hell', 'was', 'watching', 'but', 'at', 'the', 'same',
'time', 'it']
Data size (Characters) (Document 2) 223
Sample string (Document 2) ['amateur', 'no', 'budget', 'films', 'can', 'be', 'su
rprisingly', 'good', 'this', 'however', 'is', 'not', 'one', 'of', 'them', 'br',
```

In [68]:

```
dir_name = 'C:/Users/upsto/Downloads/movie-reviews-positive/movie-reviews-positive
filenames = listdir_no_hidden(path=dir_name)
num_files = len(filenames)

for i in range(len(filenames)):
    file_exists = os.path.isfile(os.path.join(dir_name, filenames[i]))
    assert file_exists
print('\nDirectory:',dir_name)
print('%d files found' % len(filenames))


def read_data(filename):

  with open(filename, encoding='utf-8') as f:
    data = tf.compat.as_str(f.read())
    data = data.lower()
    data = text_parse(data)
    data = TreebankWordTokenizer().tokenize(data)

  return data

positive_documents = []

print('\nProcessing document files under' , dir_name)
for i in range(num_files):

    words = read_data(os.path.join(dir_name, filenames[i]))

    positive_documents .append(words)
```

```
Directory: C:/Users/upsto/Downloads/movie-reviews-positive/movie-reviews-positiv e
500 files found

Processing document files under C:/Users/upsto/Downloads/movie-reviews-positive/
movie-reviews-positive
```

[69]:

```
max_review_length = 0 # initialize for doc in
negative_documents: max_review_length =
max(max_review_length, len(doc))
for doc in positive_documents:
    max_review_length = max(max_review_length, len(doc))
print('max_review_length:', max_review_length)

min_review_length = max_review_length # initialize for
doc in negative_documents: min_review_length =
min(min_review_length, len(doc))
for doc in positive_documents:
    min_review_length = min(min_review_length, len(doc))
print('min_review_length:', min_review_length)

# construct list of 1000 lists with 40 words in each list
from itertools import chain
documents = [] for doc in
negative_documents:
    doc_begin = doc[0:20]
    doc_end = doc[len(doc) - 20: len(doc)]
    documents.append(list(chain(*[doc_begin, doc_end])))
for doc in positive_documents:
    doc_begin = doc[0:20]
    doc_end = doc[len(doc) - 20: len(doc)]
```

In

```
max_review_length: 1052 min_review_length:
22
```

In [70]:

```
embeddings = []
for doc in documents:
embedding = [] for
word in doc:
         embedding.append(limited_index_to_embedding[limited_word_to_index[word]])
```

In [71]:

```
# Show the first word in the first document test_word
= documents[0][0]
print('First word in first document:', test_word)    print('Embedding
for this word:\n',
limited_index_to_embedding[limited_word_to_index[test_word]])
print('Corresponding embedding from embeddings list of list of lists\n',
      embeddings[0][0][:])
```

```
First word in first document: story
Embedding for this word:
 [ 0.48251    0.87746   -0.23455    0.0262     0.79691    0.43102
 -0.60902   -0.60764   -0.42812   -0.012523  -1.2894     0.52656
 -0.82763    0.30689    1.1972    -0.47674   -0.46885   -0.19524
 -0.28403    0.35237    0.45536    0.76853    0.0062157  0.55421
  1.0006    -1.3973    -1.6894     0.30003    0.60678   -0.46044
2.5961    -1.2178     0.28747   -0.46175   -0.25943    0.38209   -
0.28312   -0.47642   -0.059444  -0.59202    0.25613    0.21306
 -0.016129  -0.29873   -0.19468    0.53611    0.75459   -0.4112
  0.23625    0.26451   ]
Corresponding embedding from embeddings list of list of lists
 [ 0.48251    0.87746   -0.23455    0.0262     0.79691    0.43102
 -0.60902   -0.60764   -0.42812   -0.012523  -1.2894     0.52656
 -0.82763    0.30689    1.1972    -0.47674   -0.46885   -0.19524
 -0.28403    0.35237    0.45536    0.76853    0.0062157  0.55421
  1.0006    -1.3973    -1.6894     0.30003    0.60678   -0.46044
2.5961    -1.2178     0.28747   -0.46175   -0.25943    0.38209   -
0.28312   -0.47642   -0.059444  -0.59202    0.25613    0.21306
 -0.016129  -0.29873   -0.19468    0.53611    0.75459   -0.4112
0.23625    0.26451   ]
```

[72]:

```
# Show the seventh word in the tenth document test_word
= documents[6][9]
print('First word in first document:', test_word)    print('Embedding
for this word:\n',
limited_index_to_embedding[limited_word_to_index[test_word]])
print('Corresponding embedding from embeddings list of list of lists\n',
      embeddings[6][9][:])
```

```
First word in first document: but
Embedding for this word:
 [ 0.35934   -0.2657    -0.046477  -0.2496     0.54676    0.25924
 -0.64458    0.1736    -0.53056    0.13942    0.062324   0.18459
 -0.75495   -0.19569    0.70799    0.44759    0.27031   -0.32885
 -0.38891   -0.61606   -0.484      0.41703    0.34794   -0.19706
  0.40734   -2.1488    -0.24284    0.33809    0.43993   -0.21616
3.7635      0.19002   -0.12503   -0.38228    0.12944   -0.18272
0.076803   0.51579    0.0072516 -0.29192   -0.27523    0.40593   -
0.040394   0.28353   -0.024724   0.10563   -0.32879    0.10673
 -0.11503    0.074678 ]
Corresponding embedding from embeddings list of list of lists
 [ 0.35934   -0.2657    -0.046477  -0.2496     0.54676    0.25924
 -0.64458    0.1736    -0.53056    0.13942    0.062324   0.18459
 -0.75495   -0.19569    0.70799    0.44759    0.27031   -0.32885
 -0.38891   -0.61606   -0.484      0.41703    0.34794   -0.19706
```

In

```
  0.40734   -2.1488    -0.24284    0.33809    0.43993   -0.21616
3.7635      0.19002   -0.12503   -0.38228    0.12944   -0.18272
0.076803   0.51579    0.0072516 -0.29192   -0.27523    0.40593   -
0.040394   0.28353   -0.024724   0.10563   -0.32879    0.10673   -
0.11503    0.074678 ]
```

In [73]:

```
# Show the last word in the last document test_word
= documents[999][39]
print('First word in first document:', test_word)    print('Embedding
for this word:\n',
limited_index_to_embedding[limited_word_to_index[test_word]])
print('Corresponding embedding from embeddings list of list of lists\n',
      embeddings[999][39][:])
```

```
First word in first document: from
Embedding for this word:
 [ 0.41037   0.11342   0.051524 -0.53833  -0.12913   0.22247  -0.9494
 -0.18963  -0.36623  -0.067011  0.19356  -0.33044   0.11615  -0.58585
  0.36106   0.12555  -0.3581    -0.023201 -1.2319    0.23383   0.71256
  0.14824   0.50874  -0.12313  -0.20353  -1.82      0.22291   0.020291
 -0.081743 -0.27481   3.7343    -0.01874  -0.084522 -0.30364   0.27959
  0.043328 -0.24621   0.015373  0.49751   0.15108  -0.01619   0.40132
  0.23067  -0.10743  -0.36625  -0.051135  0.041474 -0.36064  -0.19616
 -0.81066 ]
Corresponding embedding from embeddings list of list of lists
 [ 0.41037   0.11342   0.051524 -0.53833  -0.12913   0.22247  -0.9494
 -0.18963  -0.36623  -0.067011  0.19356  -0.33044   0.11615  -0.58585
  0.36106   0.12555  -0.3581    -0.023201 -1.2319    0.23383   0.71256
  0.14824   0.50874  -0.12313  -0.20353  -1.82      0.22291   0.020291
 -0.081743 -0.27481   3.7343    -0.01874  -0.084522 -0.30364   0.27959
  0.043328 -0.24621   0.015373  0.49751   0.15108  -0.01619   0.40132
  0.23067  -0.10743  -0.36625  -0.051135  0.041474 -0.36064  -0.19616   -
0.81066 ]
```

[74]:

```
embeddings_array = np.array(embeddings) thumbs_down_up =
np.concatenate((np.zeros((500), dtype = np.int32), np.ones((500),
dtype = np.int32)), axis = 0)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = \ train_test_split(embeddings_array,
    thumbs_down_up, test_size=0.20, =
```

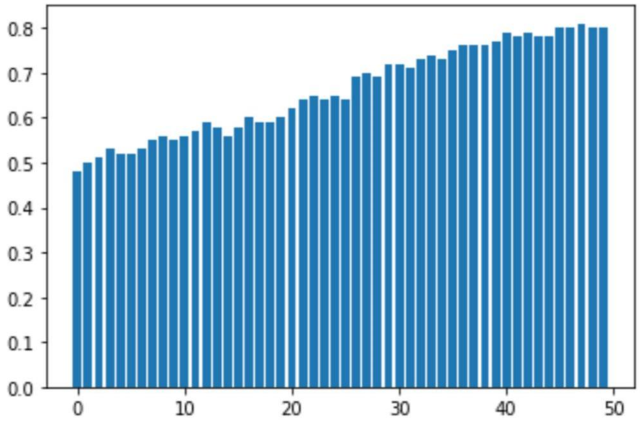
In [75]:

```
reset_graph()

n_steps = embeddings_array.shape[1]  # number of words per document n_inputs
= embeddings_array.shape[2]  # dimension of  pre-trained embeddings n_neurons
= 20 # analyst specified number of neurons n_outputs = 2 # thumbs-down or
thumbs-up learning_rate = 0.001

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs]) y
= tf.placeholder(tf.int32, [None])

basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons) outputs,
states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32)

logits = tf.layers.dense(states, n_outputs)
xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y, logits=logits)
loss = tf.reduce_mean(xentropy)
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss) correct =
tf.nn.in_top_k(logits, y, 1) accuracy =
tf.reduce_mean(tf.cast(correct, tf.float32)) init =
tf.global_variables_initializer()

n_epochs = 50
batch_size = 100

with tf.Session() as sess:
    init.run() for epoch in
    range(n_epochs):
        print('\n  ---- Epoch ', epoch, ' ----\n') for iteration in
        range(y_train.shape[0] // batch_size):          X_batch =
        X_train[iteration*batch_size:(iteration + 1)*batch_size,:] y_batch =
        y_train[iteration*batch_size:(iteration + 1)*batch_size] print('  Batch
        ', iteration, ' training observations from ',  iteration*batch_size, ' to
        ', (iteration + 1)*batch_size-1,)
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_test = accuracy.eval(feed_dict={X: X_test, y: y_test}) print('\n
        Train accuracy:', acc_train, 'Test accuracy:', acc_test)
```

```
WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
*      https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-
su nset.md (https://github.com/tensorflow/community/blob/master/rfcs/20180907-
contr ib-sunset.md)
*      https://github.com/tensorflow/addons (https://github.com/tensorflow/addons)
* https://github.com/tensorflow/io (https://github.com/tensorflow/io) (for I/O
related ops)
If you depend on functionality not listed there, please file an issue.

WARNING:tensorflow:From <ipython-input-75-78523554dc7a>:13: BasicRNNCell.__init_ _
(from tensorflow.python.ops.rnn_cell_impl) is deprecated and will be removed i n a
future version. Instructions for updating:
This class is equivalent as tf.keras.layers.SimpleRNNCell, and will be replaced by
that in Tensorflow 2.0.
WARNING:tensorflow:From <ipython-input-75-78523554dc7a>:14: dynamic_rnn (from te
nsorflow.python.ops.rnn) is deprecated and will be removed in a future version.
Instructions for updating:
```

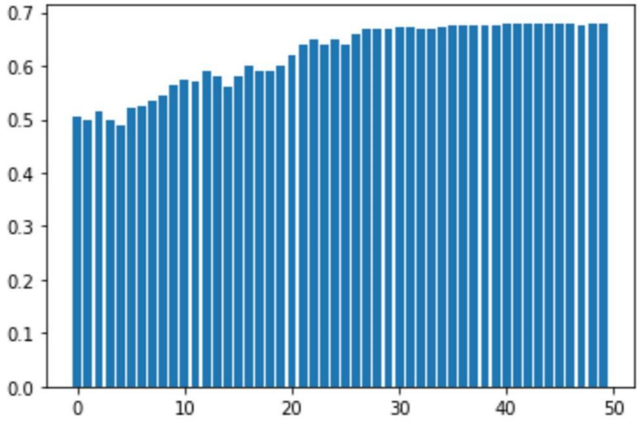
In [14]:

```
import matplotlib.pyplot as plt

#train data
plt.bar(epoch,train_acc)
```



In [16]:

```
import matplotlib.pyplot as plt

#train data
plt.bar(epoch,test_acc)
plt.
```



In [44]:

Out[44]:

```
0.6613999999999999
```

In [45]:

Out[45]:

```
0.6213599999999999
```

Glove6b-100D

```
[4]:CHAKIN_INDEX = 12
NUMBER_OF_DIMENSIONS = 100
SUBFOLDER_NAME = "gloVe.6B"

DATA_FOLDER = "embeddings"
ZIP_FILE = os.path.join(DATA_FOLDER, "{}.zip".format(SUBFOLDER_NAME))
ZIP_FILE_ALT = "glove" + ZIP_FILE[5:]
UNZIP_FOLDER = os.path.join(DATA_FOLDER, SUBFOLDER_NAME) if
SUBFOLDER_NAME[-1] == "d":
    GLOVE_FILENAME = os.path.join(
        UNZIP_FOLDER, "{}.txt".format(SUBFOLDER_NAME)) else:
    GLOVE_FILENAME = os.path.join(UNZIP_FOLDER, "{}.{}d.txt".format(
        SUBFOLDER_NAME, NUMBER_OF_DIMENSIONS))


if not os.path.exists(ZIP_FILE) and not os.path.exists(UNZIP_FOLDER):
    print("Downloading embeddings to '{}'".format(ZIP_FILE))
    chakin.download(number=CHAKIN_INDEX, save_dir='./{}'.format(DATA_FOLDER))
else:
    print("Embeddings already downloaded.")

if not os.path.exists(UNZIP_FOLDER):
    import zipfile if not os.path.exists(ZIP_FILE) and
    os.path.exists(ZIP_FILE_ALT):
        ZIP_FILE = ZIP_FILE_ALT with
    zipfile.ZipFile(ZIP_FILE, "r") as zip_ref:
        print("Extracting embeddings to '{}'".format(UNZIP_FOLDER))
        zip_ref.extractall(UNZIP_FOLDER)
else:
    print("Embeddings already extracted.")
```

```
Embeddings already downloaded. Embeddings
already extracted.

Run complete
```

In [ ]:

```
def reset_graph(seed= RANDOM_SEED):
    tf.reset_default_graph()
    tf.set_random_seed(seed)
    np.random.seed(seed)

REMOVE_STOPWORDS = False
```

In [ ]:

```
embeddings_directory = 'embeddings/gloVe.6B'
filename = 'glove.6B.100d.txt'
```

```
In [
     ]: def load_embedding_from_disks(embeddings_filename,
     with_indexes=True): """
    Read a embeddings txt file. If `with_indexes=True`,
we return a tuple of two dictionaries
    `(word_to_index_dict, index_to_embedding_array)`,
otherwise we return only a direct
```

In

```
    `word_to_embedding_dict` dictionnary mapping
from a string to a numpy array.
    """ if
    with_indexes:
        word_to_index_dict = dict() index_to_embedding_array
        = []

    else:
        word_to_embedding_dict = dict()

      with open(embeddings_filename, 'r', encoding='utf-8') as embeddings_file:
        for (i, line) in enumerate(embeddings_file):

            split = line.split(' ')

            word = split[0]

            representation = split[1:]
            representation = np.array(
                [float(val) for val in representation]
            )

            if with_indexes:
                 word_to_index_dict[word] = i
                index_to_embedding_array.append(representation)
            else:
                word_to_embedding_dict[word] = representation
    _WORD_NOT_FOUND = [0.0] * len(representation) if
    with_indexes: _LAST_INDEX = i + 1
    word_to_index_dict = defaultdict( lambda:
    _LAST_INDEX, word_to_index_dict)
        index_to_embedding_array = np.array( index_to_embedding_array
            + [_WORD_NOT_FOUND])
        return word_to_index_dict, index_to_embedding_array
    else:
        word_to_embedding_dict = defaultdict(lambda: _WORD_NOT_FOUND) return
        word_to_embedding_dict

word_to_index, index_to_embedding = \
    load_embedding_from_disks(embeddings_filename, with_indexes=True)
```

In [ ]:

```
embeddings_array = np.array(embeddings) thumbs_down_up =
np.concatenate((np.zeros((500), dtype = np.int32), np.ones((500),
dtype = np.int32)), axis = 0)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = \ train_test_split(embeddings_array,
    thumbs_down_up, test_size=0.20,
```

[5]:

```
reset_graph()

n_steps = embeddings_array.shape[1]  # number of words per document n_inputs =
embeddings_array.shape[2]  # dimension of  pre-trained embeddings n_neurons =
20 # analyst specified number of neurons n_outputs = 2 # thumbs-down or
thumbs-up learning_rate = 0.001

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs]) y
= tf.placeholder(tf.int32, [None])

basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons) outputs,
states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32)

logits = tf.layers.dense(states, n_outputs)
xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y, logits=logits)
loss = tf.reduce_mean(xentropy)
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss) correct =
tf.nn.in_top_k(logits, y, 1) accuracy =
tf.reduce_mean(tf.cast(correct, tf.float32)) init =
tf.global_variables_initializer()

n_epochs = 50
batch_size = 100

with tf.Session() as sess:
    init.run() for epoch in
    range(n_epochs):
        print('\n  ---- Epoch ', epoch, ' ----\n') for iteration in
        range(y_train.shape[0] // batch_size):          X_batch =
        X_train[iteration*batch_size:(iteration + 1)*batch_size,:] y_batch =
        y_train[iteration*batch_size:(iteration + 1)*batch_size] print('  Batch
        ', iteration, ' training observations from ',  iteration*batch_size, ' to
        ', (iteration + 1)*batch_size-1,)
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_test = accuracy.eval(feed_dict={X: X_test, y: y_test}) print('\n
        Train accuracy:', acc_train, 'Test accuracy:', acc_test)
```

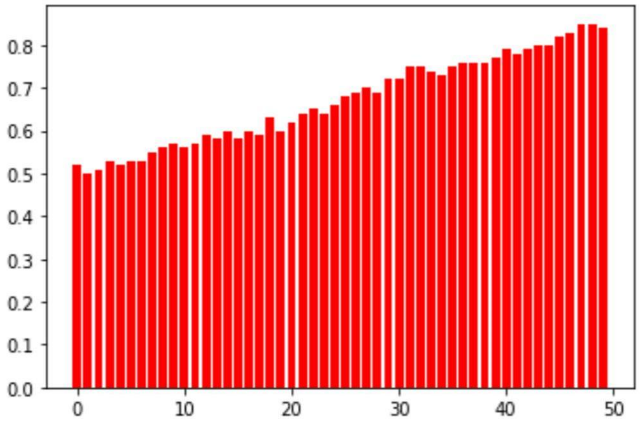
```
WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
*      https://github.com/tensorflow/community/blob/master/rfcs/20180907-
contrib-sunset.md
(https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-
sunset.md)
*      https://github.com/tensorflow/addons
(https://github.com/tensorflow/addons)   * https://github.com/tensorflow/io
(https://github.com/tensorflow/io) (for I/O related ops)
If you depend on functionality not listed there, please file an issue.

WARNING:tensorflow:From <ipython-input-75-78523554dc7a>:13: BasicRNNCell.__init_
_ (from tensorflow.python.ops.rnn_cell_impl) is deprecated and will be removed
in a future version. Instructions for updating:
This class is equivalent as tf.keras.layers.SimpleRNNCell, and will be replaced
by that in Tensorflow 2.0.
```

In

```
WARNING:tensorflow:From <ipython-input-75-78523554dc7a>:14: dynamic_rnn (from te
nsorflow.python.ops.rnn) is deprecated and will be removed in a future version.
Instructions for updating:
```

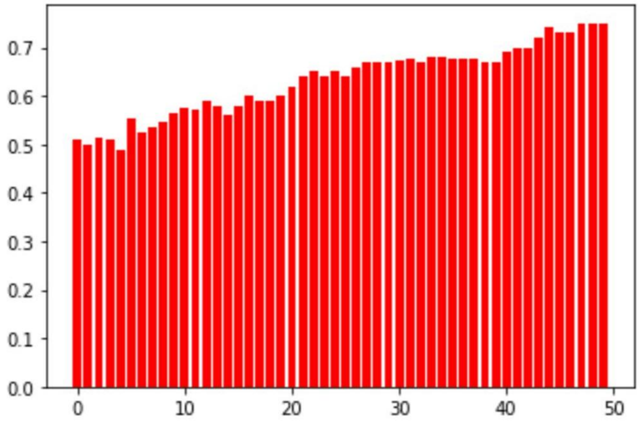
In [19]:

```
import matplotlib.pyplot as plt

#train data
plt.bar(epoch,train_acc,color='red')
```



In [20]:

```
#test data
plt.bar(epoch,test_acc,color='red')
```



In [41]:

Out[41]:

```
0.671
```

In [42]:

Out[42]:

```
0.6319400000000001
```

Fasttext-300D

[6]:

```
 CHAKIN_INDEX = 2
NUMBER_OF_DIMENSIONS = 300
SUBFOLDER_NAME = "fastText"

DATA_FOLDER = "embeddings"
ZIP_FILE = os.path.join(DATA_FOLDER, "{}.zip".format(SUBFOLDER_NAME))
ZIP_FILE_ALT = "fastText" + ZIP_FILE[5:]
UNZIP_FOLDER = os.path.join(DATA_FOLDER, SUBFOLDER_NAME) if
SUBFOLDER_NAME[-1] == "d":
    GLOVE_FILENAME = os.path.join(
        UNZIP_FOLDER, "{}.txt".format(SUBFOLDER_NAME)) else:
    GLOVE_FILENAME = os.path.join(UNZIP_FOLDER, "{}.{}d.txt".format(
        SUBFOLDER_NAME, NUMBER_OF_DIMENSIONS))


if not os.path.exists(ZIP_FILE) and not os.path.exists(UNZIP_FOLDER):
    print("Downloading embeddings to '{}'".format(ZIP_FILE))
    chakin.download(number=CHAKIN_INDEX, save_dir='./{}'.format(DATA_FOLDER))
else:
    print("Embeddings already downloaded.")

if not os.path.exists(UNZIP_FOLDER):
    import zipfile if not os.path.exists(ZIP_FILE) and
    os.path.exists(ZIP_FILE_ALT):
        ZIP_FILE = ZIP_FILE_ALT with
    zipfile.ZipFile(ZIP_FILE, "r") as zip_ref:
        print("Extracting embeddings to '{}'".format(UNZIP_FOLDER))
        zip_ref.extractall(UNZIP_FOLDER)
else:
    print("Embeddings already extracted.")
```

```
Embeddings already downloaded. Embeddings
already extracted.

Run complete
```

In [ ]:

```
def reset_graph(seed= RANDOM_SEED):
    tf.reset_default_graph()
    tf.set_random_seed(seed)
    np.random.seed(seed)

REMOVE_STOPWORDS = False
```

In [ ]:

```
embeddings_directory = 'embeddings/fastText'
filename = 'fastText.300d.txt'
```

In [ ]:

```
    def load_embedding_from_disks(embeddings_filename, with_indexes=True):
    """
    Read a embeddings txt file. If `with_indexes=True`,
we return a tuple of two dictionnaries
    `(word_to_index_dict, index_to_embedding_array)`,
otherwise we return only a direct
    `word_to_embedding_dict` dictionnary mapping
from a string to a numpy array.
    """ if
    with_indexes:
        word_to_index_dict = dict() index_to_embedding_array
        = []

    else:
        word_to_embedding_dict = dict()

      with open(embeddings_filename, 'r', encoding='utf-8') as embeddings_file:
        for (i, line) in enumerate(embeddings_file):

            split = line.split(' ')

            word = split[0]

            representation = split[1:]
            representation = np.array(
                [float(val) for val in representation]
            )

            if with_indexes:
                word_to_index_dict[word] = i
                index_to_embedding_array.append(representation)
            else:
                word_to_embedding_dict[word] = representation
    _WORD_NOT_FOUND = [0.0] * len(representation) if
    with_indexes: _LAST_INDEX = i + 1
    word_to_index_dict = defaultdict( lambda:
    _LAST_INDEX, word_to_index_dict)
        index_to_embedding_array = np.array( index_to_embedding_array
            + [_WORD_NOT_FOUND])
        return word_to_index_dict, index_to_embedding_array
    else:
        word_to_embedding_dict = defaultdict(lambda: _WORD_NOT_FOUND) return
        word_to_embedding_dict

word_to_index, index_to_embedding = \
    load_embedding_from_disks(embeddings_filename, with_indexes=True)
```

[7]:

```
reset_graph()

n_steps = embeddings_array.shape[1]  # number of words per document n_inputs =
embeddings_array.shape[2]  # dimension of  pre-trained embeddings n_neurons =
    20 # analyst specified number of neurons n_outputs = 2 # thumbs-down or
    thumbs-up learning_rate = 0.001

    X = tf.placeholder(tf.float32, [None, n_steps, n_inputs]) y
    = tf.placeholder(tf.int32, [None])

    basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons) outputs,
    states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32)

    logits = tf.layers.dense(states, n_outputs)
xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y, logits=logits)
    loss = tf.reduce_mean(xentropy)
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
    training_op = optimizer.minimize(loss) correct =
    tf.nn.in_top_k(logits, y, 1) accuracy =
    tf.reduce_mean(tf.cast(correct, tf.float32)) init =
    tf.global_variables_initializer()

    n_epochs = 50
    batch_size = 100

    with tf.Session() as sess:
        init.run() for epoch in
        range(n_epochs):
            print('\n  ---- Epoch ', epoch, ' ----\n') for iteration in
            range(y_train.shape[0] // batch_size):          X_batch =
            X_train[iteration*batch_size:(iteration + 1)*batch_size,:] y_batch =
            y_train[iteration*batch_size:(iteration + 1)*batch_size] print('  Batch
            ', iteration, ' training observations from ',  iteration*batch_size, ' to
            ', (iteration + 1)*batch_size-1,)
                sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
            acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch}) acc_test
            = accuracy.eval(feed_dict={X: X_test, y: y_test})
```

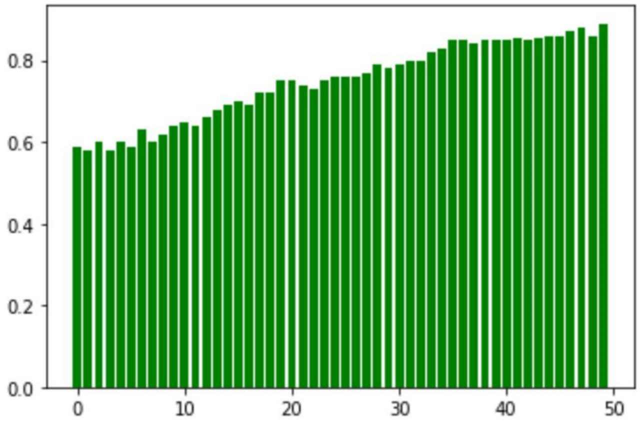
```
WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
*      https://github.com/tensorflow/community/blob/master/rfcs/20180907-
contrib-su nset.md
(https://github.com/tensorflow/community/blob/master/rfcs/20180907-contr ib-
sunset.md)
*      https://github.com/tensorflow/addons
(https://github.com/tensorflow/addons)    * https://github.com/tensorflow/io
(https://github.com/tensorflow/io) (for I/O related ops)
If you depend on functionality not listed there, please file an issue.

WARNING:tensorflow:From <ipython-input-75-78523554dc7a>:13: BasicRNNCell.__init_
_ (from tensorflow.python.ops.rnn_cell_impl) is deprecated and will be removed
i n a future version. Instructions for updating:
This class is equivalent as tf.keras.layers.SimpleRNNCell, and will be replaced
by that in Tensorflow 2.0.
```

In

```
WARNING:tensorflow:From <ipython-input-75-78523554dc7a>:14: dynamic_rnn (from te
nsorflow.python.ops.rnn) is deprecated and will be removed in a future version.
Instructions for updating:
```

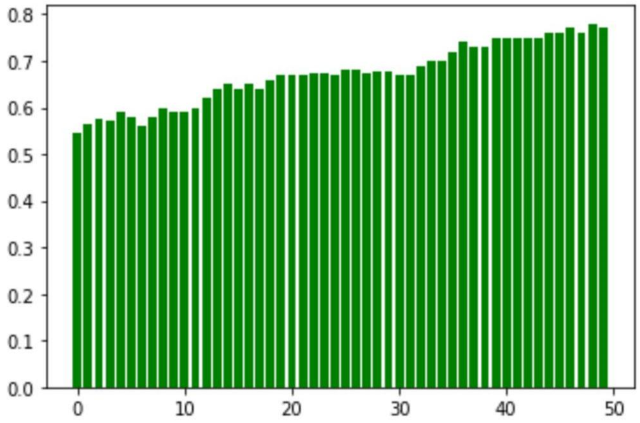
In [23]:

```
#train data
plt.bar(epoch,train_acc,color='green')
```



In [25]:

```
#test data
plt.bar(epoch,test_acc,color='green')
```



In [39]:

Out[39]:

```
0.74862
```

In [36]:

Out[36]:

```
0.6711800000000001
```

word2vec.Googlenews-300d

In

```
[10]:CHAKIN_INDEX = 21
NUMBER_OF_DIMENSIONS = 300
SUBFOLDER_NAME = "word2vec.GoogleNews"

DATA_FOLDER = "embeddings"
ZIP_FILE = os.path.join(DATA_FOLDER, "{}.zip".format(SUBFOLDER_NAME))
ZIP_FILE_ALT = "word2vec" + ZIP_FILE[5:]
UNZIP_FOLDER = os.path.join(DATA_FOLDER, SUBFOLDER_NAME) if
SUBFOLDER_NAME[-1] == "d":
    GLOVE_FILENAME = os.path.join(
        UNZIP_FOLDER, "{}.txt".format(SUBFOLDER_NAME)) else:
    GLOVE_FILENAME = os.path.join(UNZIP_FOLDER, "{}.{}d.txt".format(
        SUBFOLDER_NAME, NUMBER_OF_DIMENSIONS))


if not os.path.exists(ZIP_FILE) and not os.path.exists(UNZIP_FOLDER):
    print("Downloading embeddings to '{}'".format(ZIP_FILE))
    chakin.download(number=CHAKIN_INDEX, save_dir='./{}'.format(DATA_FOLDER))
else:
    print("Embeddings already downloaded.")

if not os.path.exists(UNZIP_FOLDER):
    import zipfile if not os.path.exists(ZIP_FILE) and
    os.path.exists(ZIP_FILE_ALT):
        ZIP_FILE = ZIP_FILE_ALT with
    zipfile.ZipFile(ZIP_FILE, "r") as zip_ref:
        print("Extracting embeddings to '{}'".format(UNZIP_FOLDER))
zip_ref.extractall(UNZIP_FOLDER) else:
    print("Embeddings already extracted.")
```

```
Embeddings already downloaded. Embeddings
already extracted.
```

```
Run complete
```

```
def reset_graph(seed= RANDOM_SEED):
    tf.reset_default_graph()
    tf.set_random_seed(seed)
    np.random.seed(seed)

REMOVE_STOPWORDS  = False
```

In [ ]:

In [ ]:

```
        def load_embedding_from_disks(embeddings_filename, with_indexes=True):
        """
        Read a embeddings txt file. If `with_indexes=True`,
we return a tuple of two dictionnaries
    `(word_to_index_dict, index_to_embedding_array)`,
otherwise we return only a direct
    `word_to_embedding_dict` dictionnary mapping
from a string to a numpy array.
```

```
    """ if
    with_indexes:
        word_to_index_dict = dict() index_to_embedding_array
        = []

    else:
        word_to_embedding_dict = dict()

      with open(embeddings_filename, 'r', encoding='utf-8') as embeddings_file:
        for (i, line) in enumerate(embeddings_file):

            split = line.split(' ')

            word = split[0]

            representation = split[1:]
            representation = np.array(
                [float(val) for val in representation]
            )

            if with_indexes:
                word_to_index_dict[word] = i
                index_to_embedding_array.append(representation)
            else:
                word_to_embedding_dict[word] = representation
    _WORD_NOT_FOUND = [0.0] * len(representation) if
    with_indexes: _LAST_INDEX = i + 1
    word_to_index_dict = defaultdict( lambda:
    _LAST_INDEX, word_to_index_dict)
        index_to_embedding_array = np.array( index_to_embedding_array
            + [_WORD_NOT_FOUND])
        return word_to_index_dict, index_to_embedding_array
    else:
        word_to_embedding_dict = defaultdict(lambda: _WORD_NOT_FOUND) return
        word_to_embedding_dict

word_to_index, index_to_embedding = \
    load_embedding_from_disks(embeddings_filename, with_indexes=True)
```

In [26]:

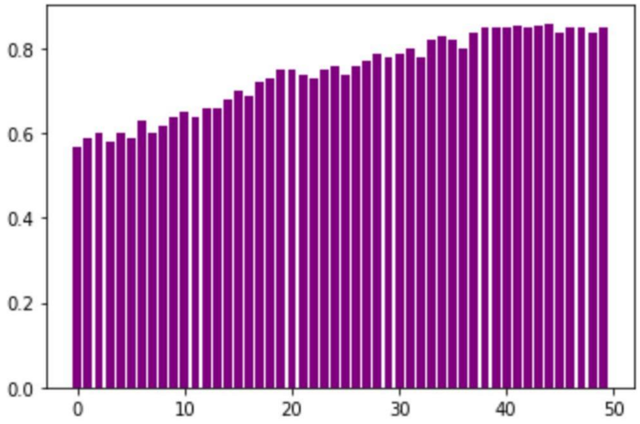
```
reset_graph()

n_steps = embeddings_array.shape[1]  # number of words per document n_inputs =
embeddings_array.shape[2]  # dimension of  pre-trained embeddings n_neurons =
20 # analyst specified number of neurons n_outputs = 2 # thumbs-down or
thumbs-up learning_rate = 0.001

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs]) y
= tf.placeholder(tf.int32, [None])

basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons) outputs,
states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32)

logits = tf.layers.dense(states, n_outputs)
```

```
xentropy     = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y, logits=logits)
            loss = tf.reduce_mean(xentropy)
            optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
            training_op = optimizer.minimize(loss) correct =
            tf.nn.in_top_k(logits, y, 1) accuracy =
            tf.reduce_mean(tf.cast(correct, tf.float32)) init =
            tf.global_variables_initializer()

            n_epochs = 50
            batch_size = 100

            with tf.Session() as sess:
                init.run() for epoch in
                range(n_epochs):
                    print('\n  ---- Epoch ', epoch, ' ----\n') for iteration in
                    range(y_train.shape[0] // batch_size):          X_batch =
                    X_train[iteration*batch_size:(iteration + 1)*batch_size,:] y_batch =
                    y_train[iteration*batch_size:(iteration + 1)*batch_size] print('  Batch
                    ', iteration, ' training observations from ',  iteration*batch_size, ' to
                    ', (iteration + 1)*batch_size-1,)
                        sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
                    acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch}) acc_test
                    = accuracy.eval(feed_dict={X: X_test, y: y_test})
```

```
WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
*      https://github.com/tensorflow/community/blob/master/rfcs/20180907-
contrib-su nset.md
(https://github.com/tensorflow/community/blob/master/rfcs/20180907-contr ib-
sunset.md)
*      https://github.com/tensorflow/addons
(https://github.com/tensorflow/addons)   * https://github.com/tensorflow/io
(https://github.com/tensorflow/io) (for I/O related ops)
If you depend on functionality not listed there, please file an issue.

WARNING:tensorflow:From <ipython-input-75-78523554dc7a>:13: BasicRNNCell.__init_
_ (from tensorflow.python.ops.rnn_cell_impl) is deprecated and will be removed
i n a future version. Instructions for updating:
This class is equivalent as tf.keras.layers.SimpleRNNCell, and will be replaced
by that in Tensorflow 2.0.
WARNING:tensorflow:From <ipython-input-75-78523554dc7a>:14: dynamic_rnn (from te
nsorflow.python.ops.rnn) is deprecated and will be removed in a future version.
Instructions for updating:
```

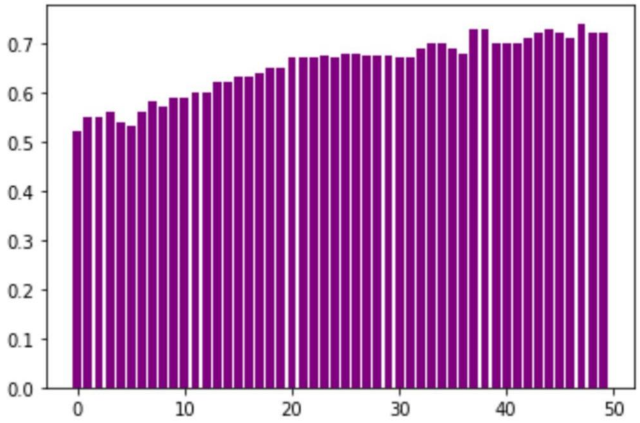
In [30]:

```
#train data
plt.bar(epoch,train_acc,color='purple')
```



In [29]:

```
#test data
plt.bar(epoch,test_acc,color='purple')
```



In [32]:

```
Out[32]: 0.74302
```

In [33]:

```
Out[33]: 0.6530799999999999
```

In [ ]: