NLP Lab -6-

Using Word Embedding and CNN to Build a Classifier

*Problem: classify news articles into pre-defined 20 categories

- Given a dataset of news article, the objective is to build a CNN classifer to predict the class of a given news article, Glove word embedding model will be used to numerically represent the articles' text
- This script loads pre-trained word embeddings (GloVe embeddings) into a frozen Keras Embedding layer, and uses it to train a text classification model on the 20 Newsgroup dataset
- GloVe embedding data can be found at: http://nlp.stanford.edu/data/glove.6B.zip (http://nlp.stanford.edu/data/glove.6B.zip) (source page: http://nlp.stanford.edu/projects/glove/ (http://nlp.stanford.edu/projects/glove/))
- 20 Newsgroup data can be found at: http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/news20.html (http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/news20.html)
- reference:https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html (https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html)

(1) Imports

In [2]:

```
import os
import sys
import numpy as np
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.layers import Embedding
from keras.layers import Dense, Input, GlobalMaxPooling1D
from keras.layers import Conv1D, MaxPooling1D, Embedding, Flatten
from keras.models import Model
```

```
c:\users\sarahhassan\appdata\local\programs\python\python35\lib\site-packa
ges\h5py\__init__.py:36: FutureWarning: Conversion of the second argument
of issubdtype from `float` to `np.floating` is deprecated. In future, it w
ill be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
c:\users\sarahhassan\appdata\local\programs\python\python35\lib\site-packa
ges\requests\__init__.py:80: RequestsDependencyWarning: urllib3 (1.22) or
chardet (2.3.0) doesn't match a supported version!
  RequestsDependencyWarning)
```

(2) Load dataset

prepare text samples and their labels.

- Input: path to a folder containing news files. the dataset folder contain 20 folder each representing one class. each folder contains samples of news articles from each class. each article is written in a separate text file.
- output:

1) list of all text samples

2) list of the label ID for each text sample

3) dictionary mapping label ID to label name

In [3]:

```
print('Processing text dataset')

TEXT_DATA_DIR = '20_newsgroup'
texts = []  # list of text samples
labels_index = {}  # dictionary mapping label name to numeric id
labels = []  # list of label ids
for name in sorted(os.listdir(TEXT_DATA_DIR)):
    path = os.path.join(TEXT_DATA_DIR, name)
    if os.path.isdir(path):
        label_id = len(labels_index)
        labels_index[name] = label_id
        for fname in sorted(os.listdir(path)):
            if fname.isdigit():
                fpath = os.path.join(path, fname)
                args = {} if sys.version_info < (3,) else {'encoding': 'latin-1'}
                with open(fpath, **args) as f:
                    t = f.read()
                    i = t.find('\n\n')  # skip header
                    if 0 < i:
                        t = t[i:]
                    texts.append(t)
                labels.append(label_id)
```

```
Processing text dataset
```

In [4]:

```
print('Found %s texts.' % len(texts))
print('nLabels = ', len(labels))
print('Classes are:\n ')
for key in labels_index:
        print (key)
```

```
Found 19997 texts.
nLabels =  19997
Classes are:

alt.atheism
soc.religion.christian
comp.windows.x
sci.crypt
comp.sys.ibm.pc.hardware
sci.med
comp.os.ms-windows.misc
rec.motorcycles
comp.sys.mac.hardware
rec.sport.hockey
talk.religion.misc
comp.graphics
misc.forsale
sci.electronics
sci.space
talk.politics.mideast
rec.sport.baseball
rec.autos
talk.politics.guns
talk.politics.misc
```

(3) Process Text samples

The steps to process text samples include:

- tokenize samples to words and keep top 20,000 most commonly occuring words in the dataset and neglect the others
- form a dictionary of all words in the dataset.. the dictionary maps a word to a word ID
- replace each word by its ID
- truncate/pad the sequences to a maximum length of 1000 words

* *Keras tokenizer object lowers all characters and removes special characters: '!"#$%&()*+,-./:;<=>?@[\]^_`{|}~\t\n'

=> Each news article will be a vector of words IDS

In [5]:

```
MAX_SEQUENCE_LENGTH = 1000
MAX_NUM_WORDS = 20000
```

In [7]:

```
tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index # the dictionary
print('Found %s unique tokens.' % len(word_index)) #only top MAX_NUM_WORDS will be used to generate the sequences
data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)
print('Shape of samples:', data.shape)
print('Sampele:(the zeros at the begining are for padding text to max length)')
print(data[2])
```

```
Found 174074 unique tokens.
Shape of samples: (19997, 1000)
Sampele:(the zeros at the begining are for padding text to max length)
[    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     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     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     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     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     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
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     6    54
 10754  1731    26  8090   201  1731    26  9652  8455    43   107   316
    52     4   346   251    17  1341    68  4299    39     8    58   362
     9    51   200    18   575  3020     5     9     1  2694     3    89
    25   575     2   166    24    17     6     4   648    14     8     4
   108   610     2    57   105    13  6174     4   246     1   610   654
    21  1878    89  2518  1304     6   316    24    17     6     1   521
   120    27    70     6  1456    22  3020   240    27    19   817     2
    16   597   262     2   180     2   883     3    89    21   135     5
    17    32  1629   251   432  1507     9    51  1183  3020    38  1032
   265   166   154  3020     5    38   252   362    12     1  2201  2400
     7    65    84    14  1878    23    13   179     1   134  3231    91
  3020   182    18   575   316    24   316     8    32  1056     6   282
   238   713   599     3    93   146    61    19   107   567     1  1259
     3 18377    46    19   599   799     3  4089   146    17     1  1190
     3     1  3187     1  2106  2288    12     1    93   329     3   429
  1456     5  3020    88    17   559    71  9483  5626    11   289  1456
    52   205     1   666     3   429     5  3020     9     3  1456     5
   255     9     3   429    21    11     8  2860     9   316  1183     1
  2694     3 15003   666     1   362    12     9     8    17  6175  7221
    28  1351 11222     3   316    18    83  2270   692   142     5    61
    80    19    27  8593    80   309    19    27  1140     7   690   216
     1   414    31   501    11    25     7    84     6     1  1185 15004
    22   729     5    11    25  1370    21   309    21  9190  1015    67
    27    19    29    44  1015   121    42    27  7403   834    15     1
   329    15  1677   316   154     1   687     3     1   342     7    65
   116    53   180    11     8   615    35   802   145    27    81    50
   126   501   883     3     1  1143     5    11  1264     9    31   258
   123     3    77   151  1950  4089    21     1  1019     3    71  1295
     5   114     1  2366     3  3020     2   543    22   429     2   870
    19    17  1202  1687   107     4  6764  1231     3   870    38    16
   849   262   513    11   203    32   485   861    20  1495   344     3
  5562     5  4194    38    16   637    48   827     9  1025     2   130
   112     7  1859   159     5     7    65    84     4    40  1491  2604
   432     8     9   349    23  3020     5   429     5  1456  2422    71
  2309  1043    29    91     7   157   419   163     6     1   576     3
   397 10568    29   320     2  3189     9    58    66 15005     6   256
  1168    40  1202   773    50   184   185    82    14     5    23   316
     8     4    91   452   290    68     2    16   145     9   316    25
     4 13456     8    17  1202  1687     1  1922     5  1031   861    20
     1  4299    19   661   151    21     1   610     9   316    25     4
 13456 12815    15     1   476     6     1  3189     3   316   667     8
    39    55    75   362    12    11    40  1491     5    40  2031  2604
     8   349   114     6   138   387  2051     9     2  1168     3   597
  1143     6   138  1719     6   521 13457  6762    17   114     2   840
   174     9  4570    19    17  1341     1   259   835    11     8    86
  1264     9   429    52    83  5766    80   309    19     1 10755 11222
     2    33   883    61    34    16     1 10755     8   346   154    55
     3   100  6680     5    11     8    17   114   945     1    76   580
  5715     8   168     3     1  6176     5    96  1350     4 19951  4472
     1 10755  1680   252     4   858   579     3     2     1   180   142
     1 17068  1680   391     2  1111   174     6     1   134  1264  1255
     8     9    31    58   180   230     3     1   666    25   750     1
  1307    29  3083   128    15     8   457   917   538     5   253     2
  3105     4     3     1   305     3  3020     6    75   560    40    88
    17    74    35     1   485     3   429   139   221    64     5  1403
   323    15   429    19   457  2298    24    80     8     9  2069     2
     4     3   316  5816]
```

(4) format output of the CNN (the shape of labels)

- Each label will be a binary vector of size 20

In [8]:

```
labels_matrix = to_categorical(np.asarray(labels))
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels_matrix.shape)
print('Sample label:\n',labels_matrix[1590])
```

```
Shape of data tensor: (19997, 1000)
Shape of label tensor: (19997, 20)
Sample label:
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

(5) Split samples and labels to training and testing sets

In [9]:

```
VALIDATION_SPLIT = 0.2

indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data_shuffled = data[indices]
labels_shuffled = labels_matrix[indices]
nb_validation_samples = int(VALIDATION_SPLIT * data_shuffled.shape[0])

x_train = data_shuffled[:-nb_validation_samples]
y_train = labels_shuffled[:-nb_validation_samples]
x_val = data_shuffled[-nb_validation_samples:]
y_val = labels_shuffled[-nb_validation_samples:]
print('Shape of training data: ',x_train.shape)
print('Shape of testing data: ',x_val.shape)
```

```
Shape of training data:  (15998, 1000)
Shape of testing data:  (3999, 1000)
```

(6) Read Glove Word Embeddings

Build a dictionary mapping words in the embeddings set to their embedding vector

In [10]:

```
EMBEDDING_DIM = 100

print('Indexing word vectors.')

embeddings_index = {}

with open('glove.6B.100d.txt') as f:
    for line in f:
        values = line.split(sep=' ')
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

print('Found %s word vectors.' % len(embeddings_index))
```

```
Indexing word vectors.
Found 400000 word vectors.
```

(7) Map the dataset dictionary of (words,IDs) to a matrix of the embeddings of each word in the dictionary

the words in the dataset that don't exist in Glove's dictionary will get a zeroes vector these words already have id 0 so there zeros embedding ector will be at index zero in the new matrix

In [11]:

```
embedding_matrix = np.zeros((len(word_index) + 1, EMBEDDING_DIM))#+1 to include the zerors vector for non-existing words
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
print ('Shape of Embedding Matrix: ',embedding_matrix.shape)
```

```
Shape of Embedding Matrix:  (174075, 100)
```

(8) Build the Deep NN:

(8.1) Embedding Layer

responsible of converting a paded sequence of words IDs to a sequence of words embeddings

In [12]:

```
embedding_layer = Embedding(len(word_index) + 1, #vocab size
                            EMBEDDING_DIM,       #embedding vector size
                            weights=[embedding_matrix], #weights matrix
                            input_length=MAX_SEQUENCE_LENGTH, #padded sequence length
                            trainable=False)
```

(8.2) Build 1D CNN Layers

In [13]:

```
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
x = Conv1D(128, 5, activation='relu')(embedded_sequences)
x = MaxPooling1D(5)(x)
x = Conv1D(128, 5, activation='relu')(x)
x = MaxPooling1D(5)(x)
x = Conv1D(128, 5, activation='relu')(x)
x = MaxPooling1D(35)(x)  # global max pooling
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
preds = Dense(len(labels_index), activation='softmax')(x)
```

(8.3) Build, Compile, and Run the model

In [14]:

```
model = Model(sequence_input, preds)
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])

# happy learning!
model.fit(x_train, y_train, validation_data=(x_val, y_val),
          epochs=5, batch_size=128)
```

```
Train on 15998 samples, validate on 3999 samples
Epoch 1/5
15998/15998 [==============================] - 189s 12ms/step - loss: 2.43
14 - acc: 0.2108 - val_loss: 1.8208 - val_acc: 0.3576
Epoch 2/5
15998/15998 [==============================] - 183s 11ms/step - loss: 1.54
58 - acc: 0.4616 - val_loss: 1.3524 - val_acc: 0.5279
Epoch 3/5
15998/15998 [==============================] - 183s 11ms/step - loss: 1.19
46 - acc: 0.5871 - val_loss: 1.1309 - val_acc: 0.6107
Epoch 4/5
15998/15998 [==============================] - 182s 11ms/step - loss: 0.97
26 - acc: 0.6671 - val_loss: 1.0934 - val_acc: 0.6297
Epoch 5/5
15998/15998 [==============================] - 182s 11ms/step - loss: 0.83
66 - acc: 0.7165 - val_loss: 0.9568 - val_acc: 0.6734
```

Out[14]:

```
<keras.callbacks.History at 0x375c8a20>
```

(8.4) Evaluate the model

In [15]:

```
print('Acuracy on testing set:')
model.evaluate(x_val,y_val)
```

```
Acuracy on testing set:
3999/3999 [==============================] - 18s 5ms/step
```

Out[15]:

```
[0.9568204918066303, 0.6734183546631716]
```

(9) Use the model for prediction

In [16]:

```
model.predict(x_val)
```

Out[16]:

```
array([[1.0327732e-02, 2.9313486e-04, 2.4011679e-06, ..., 1.7724369e-05,
        1.8570792e-03, 7.6327776e-04],
       [1.2563624e-02, 1.9062838e-02, 8.7171635e-03, ..., 8.0603240e-03,
        2.9240904e-02, 5.6615467e-03],
       [5.5559494e-06, 4.2651324e-03, 4.3965750e-03, ..., 2.5074318e-04,
        1.2648354e-03, 9.5650757e-06],
       ...,
       [1.1672964e-03, 2.3448658e-05, 2.9504277e-05, ..., 1.2243649e-03,
        3.6598616e-03, 3.0622948e-03],
       [4.8667334e-06, 7.7692151e-02, 5.7424837e-01, ..., 6.6595958e-06,
        6.7728670e-06, 1.7917351e-05],
       [2.5362926e-06, 2.0805377e-01, 3.7774213e-02, ..., 6.3202569e-06,
        1.6121081e-05, 2.8711088e-06]], dtype=float32)
```

In [18]:

```
sample = 1
label_vec = model.predict(data[sample].reshape(1,-1))
label_id = np.argmax(label_vec)
label_name = ''
for name, ID in labels_index.items():    # for name, age in dictionary.iteritems():  (f
or Python 2.x)
    if label_id == ID:
        label_name = name
        break
print ('The category of article no %s is %s' %(sample ,label_name))
```

```
The category of article no 1 is soc.religion.christian
```