# 1. Text Summariazation of News Articles

Lets start the work, and a short note the work is based on the colab platform so please make necessary changes when using any other platform

First task is to know the dependencies, Following are the list of dependencies

1. Levenshtein : The Levenshtein Python C extension module contains functions for fast computation of
    - Levenshtein (edit) distance(minimum number of edits)
    - string similarity
    - approximate median strings, and generally string averaging
    - string sequence and set similarity
2. Tensorflow and Keras with tensorflow backend

In [0]:

```
!pip install python-levenshtein
!pip install tensorflow==2.0.0-alpha0
!pip install keras
```

In [0]:

```
#Linking the Google colab to google drive
from google.colab import drive
drive.mount('/content/drive/')
```

## 2. DataSet Infromation

The next step is to download the dataset, the dataset that we are using for this project is the **"Signal One Million News Articles Dataset"**, which has one million data items, where each data item consist of the headline and the corresponding news article.

The link for the dataset is as follows : Signal Reasearch data set

## Pre-processing the data

After downloading the dataset, we can see that the format of the dataset is json, then we have to convert it to pickle triplet for easier and better use

for this purpose install following libraries **pickle and jsonlines** using pip.

In [0]:

```
import _pickle as pickle
import jsonlines

heads = []
descs = []
keywords = []

count = 0
with jsonlines.open('./sample-1M' + '.jsonl','r') as reader:   #file location
    i = 0
    for obj in reader:
        if i < 10000:
            i += 1
            head = obj["title"]
            desc = [s.strip() for s in obj["content"].splitlines()]
            desc = ' '.join(desc)
            heads.append(head)
            descs.append(desc)
        else:
            break
print(count)

# print(heads, descs, keywords)
with open('./tokens_10000.pkl', 'wb') as f:   #output file name and location
    pickle.dump((heads,descs,keywords),f)
```

## 3. Read tokenized headlines and descriptions

In [0]:

```
filename = '/content/drive/My Drive/Summarizer_dataset/tokens_10000.pkl'    #File Location to load
after pre-processing
```

In [0]:

```
FN = 'vocabulary-embedding'
seed=42
vocab_size = 40000
embedding_dim = 100
lower = False # dont lower case the text
```

In [0]:

```
import _pickle as pickle
```

In [0]:

```
with open(filename, 'rb') as fp:
    heads, desc, keywords = pickle.load(fp) # keywords are not used in this project
```

In [0]:

```
if lower:
    heads = [h.lower() + ' ' for h in heads]
#print(heads[0])


if lower:
    desc = [h.lower() for h in desc]
# print(desc[0])
```

## 4. Build vocabulary

In [0]:

```
from collections import Counter
from itertools import chain
import operator
def get_vocab(lst):
    vocabcount = Counter(w for txt in lst for w in txt.split())
    lst = dict(vocabcount)
    lst = sorted(lst.items(), key = operator.itemgetter(1), reverse = True)
    rst = []
    for i in lst:
      rst.append(i[0])
    vocab = map(lambda x: x[0], sorted(vocabcount.items(), key=lambda x: -x[1]))
    return rst, vocabcount
```

The **get_vocab** function takes the list head + desc as input and outputs a list and counter as output , the counter variable is used to count the occurence of each word in the list and store it. The list rst stores the words in sorted order (descendig) based on the count value of each word. The list rst is stored in vocab and vocabcounter stores the counter of the vocabulary.

```
vocab, vocabcount = get_vocab(heads + desc)
```

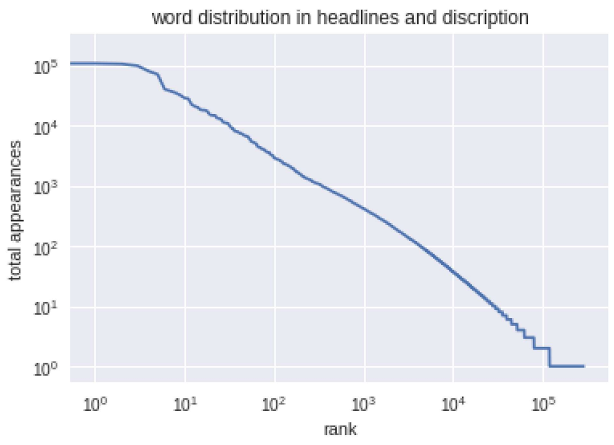
**Plotting the vocabulary and the corresponding rank**

In [0]:

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([vocabcount[w] for w in vocab]);
plt.gca().set_xscale("log", nonposx='clip')
plt.gca().set_yscale("log", nonposy='clip')
plt.title('word distribution in headlines and discription')
plt.xlabel('rank')
plt.ylabel('total appearances')
```

Out[0]:

```
Text(0, 0.5, 'total appearances')
```



## 5. Indexing the word

In [0]:

```
empty = 0 # RNN mask of no data
eos = 1   # end of sentence
start_idx = eos+1 # first real word
```

The method **get_idx** creates two dictionaries/maps one which maps a word to an index and an another which maps an index to a word.

In [0]:

```
def get_idx(vocab, vocabcount):
    word2idx = dict((word, idx+start_idx) for idx,word in enumerate(vocab))
    word2idx['<empty>'] = empty
    word2idx['<eos>'] = eos

    idx2word = dict((idx,word) for word,idx in word2idx.items())

    return word2idx, idx2word
```

In [0]:

```
word2idx, idx2word = get_idx(vocab, vocabcount)
```

## 6. Word Embedding

Below code is for downloading the Glove from the nlp server and unzipping it into the drive

In [0]:

```
fname = 'glove.6B.%dd.txt'%embedding_dim
from keras.utils import get_file
path = 'glove.6B.zip'
path = get_file(path, origin="http://nlp.stanford.edu/data/glove.6B.zip",
cache_dir='/content/drive/My\ Drive/Summarizer_dataset')
!unzip {'/content/drive/My\ Drive/Summarizer_dataset'}/{'glove.6B.zip'}
```

Below code checks the size of the downloaded Glove directory

In [0]:

```
glove_name = './' + fname
glove_n_symbols = !wc -l {glove_name}
glove_n_symbols = int(glove_n_symbols[0].split()[0])
```

Below code creates a dictionaries/maps **glove_index_dict** which maps the words in the Glove to the index and a numpy array **glove_embedding_weights** which stores the glove vector corresponding to each word in the Glove, **glove_embedding_weights** is a array of dimension (4,00,000 x 100) where each row index corresponds to a word whose mapping is provided by the dictionary **glove_index_dict.**

In [0]:

```
import numpy as np
glove_index_dict = {}
glove_embedding_weights = np.empty((glove_n_symbols, embedding_dim))
globale_scale=.1
with open(glove_name, 'r') as fp:
    i = 0
    for l in fp:
        l = l.strip().split()
        w = l[0]
        glove_index_dict[w] = i
        glove_embedding_weights[i,:] = list(map(float,l[1:]))
        i += 1
glove_embedding_weights *= globale_scale
```

Converting all words to lower cases within the glove

In [0]:

```
for w,i in glove_index_dict.items():
    w = w.lower()
    if w not in glove_index_dict:
        glove_index_dict[w] = i
```

## 7. Embedding Matrix

Use GloVe to initialize embedding matrix. We generate random embedding with same scale as glove. Then we copy from glove weights of words that appear in our short vocabulary (idx2word) from the glove to our embedding array.

In [78]:

```
# generate random embedding with same scale as glove
np.random.seed(seed)
shape = (vocab_size, embedding_dim)
scale = glove_embedding_weights.std()*np.sqrt(12)/2 # uniform and not normal
embedding = np.random.uniform(low=-scale, high=scale, size=shape)
print('random-embedding/glove scale', scale, 'std', embedding.std())

# copy from glove weights of words that appear in our short vocabulary (idx2word)
```

```
for i in range(vocab_size):
    w = idx2word[i]
    g = glove_index_dict.get(w, glove_index_dict.get(w.lower()))
    if g is None and w.startswith('#'): # glove has no hastags (I think...)
        w = w[1:]
        g = glove_index_dict.get(w, glove_index_dict.get(w.lower()))
    if g is not None:
        embedding[i,:] = glove_embedding_weights[g,:]
        c+=1
print('number of tokens, in small vocab, found in glove and copied to embedding',
c,c/float(vocab_size))
```

```
random-embedding/glove scale 0.0706949139514209 std 0.04081382495746382
number of tokens, in small vocab, found in glove and copied to embedding 27322 0.68305
```

**Note** : Lots of word in the full vocabulary (word2idx) are outside vocab_size. Build an alterantive which will map them to their closest match in glove but only if the match is good enough (cos distance above glove_thr)

In [0]:

```
glove_thr = 0.5
```

We now maintain a new dictionary **word2glove** which keeps the mapping from each word to its glove counterpart

In [0]:

```
word2glove = {}
for w in word2idx:
    if w in glove_index_dict:
        g = w
    elif w.lower() in glove_index_dict:
        g = w.lower()
    elif w.startswith('#') and w[1:] in glove_index_dict:
        g = w[1:]
    elif w.startswith('#') and w[1:].lower() in glove_index_dict:
        g = w[1:].lower()
    else:
        continue
    word2glove[w] = g
```

1. For every word outside the embedding matrix find the closest word inside the mebedding matrix. Use cos distance of GloVe vectors for this purpose. Cos distance is the angle formed between the vectors.

1. Allow for the last nb_unknown_words words inside the embedding matrix to be considered to be outside. Dont accept distances below glove_thr.

In [82]:

```
normed_embedding = embedding/np.array([np.sqrt(np.dot(gweight,gweight)) for gweight in embedding])
[:,None]

nb_unknown_words = 100

glove_match = []
for w,idx in word2idx.items():
    if idx >= vocab_size-nb_unknown_words and w.isalpha() and w in word2glove:
        gidx = glove_index_dict[word2glove[w]]
        gweight = glove_embedding_weights[gidx,:].copy()
        # find row in embedding that has the highest cos score with gweight
        gweight /= np.sqrt(np.dot(gweight,gweight))
        score = np.dot(normed_embedding[:vocab_size-nb_unknown_words], gweight)
        while True:
            embedding_idx = score.argmax()
            s = score[embedding_idx]
            if s < glove_thr:
                break
            if idx2word[embedding_idx] in word2glove :
                glove_match.append((w, embedding_idx, s))
                break
```

```
print('# of glove substitutes found', len(glove_match))
```

```
# of glove substitutes found 48240
```

Finally we build a lookup table of index of outside words to index of inside words.

In [0]:

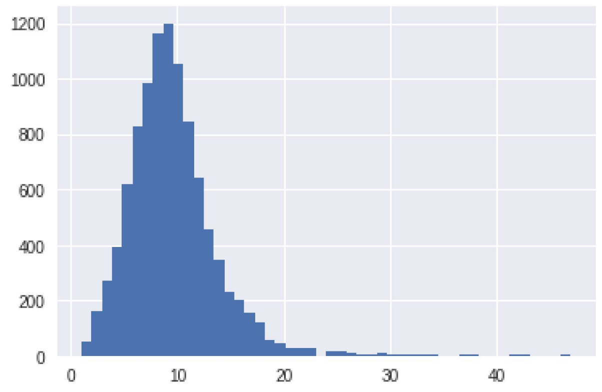
```
glove_idx2idx = dict((word2idx[w],embedding_idx) for  w, embedding_idx, _ in glove_match)
```

## 8. Output of the Vocabulary Build phase

**`Plot of the headline(target) vocabulary.`**

In [94]:

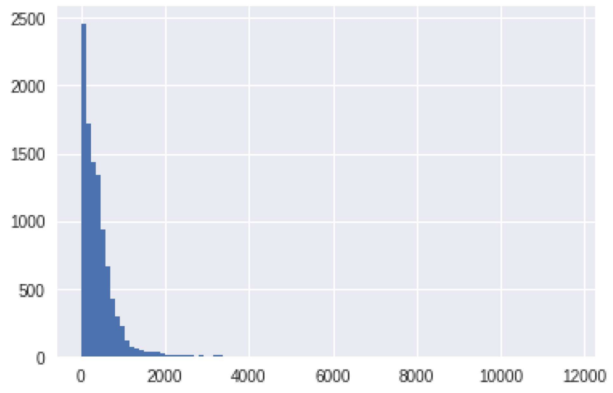
```
Y = [[word2idx[token] for token in headline.split()] for headline in heads]
plt.hist(list(map(len,Y)),bins=48);
```



**`Plot of the Description(Input) vocabulary.`**

In [98]:

```
X = [[word2idx[token] for token in d.split()] for d in desc]
plt.hist(list(map(len,X)),bins=100);
```



**Saving the files for next phase**

In [0]:

```
with open('/content/drive/My Drive/Summarizer_dataset/%s.pkl'%FN,'wb') as fp:
    pickle.dump((embedding, idx2word, word2idx, glove_idx2idx),fp,-1)
with open('/content/drive/My Drive/Summarizer_dataset/%s.data.pkl'%FN,'wb') as fp:
```