Introduction:

Text is one of the most widespread forms of sequence data. It can be understood as either a sequence of charactors or a sequence of words, but it's most common to work at level of words. Text-sequence processing includes following applications:

Applications of deep learning for text data:

Document classification Articles lebelling Sentiment analysis Author identification Question-answering Language detection Translation Tasks In true sense deep learning models map the statistical structure of written language, which is sufficient to solve many simple textual tasks and problems. Deep learning for natural-language processing is pattern recognition applied to words, sequence, and paragraphs, in much similar way that computer vision is pattern recognition applied to pixels. Deep-learning models don't take input as text like other models they only work with numeric tensors Three techniques to vectorize the text data:

Segment text into words, and convert word into a vector Segment text into charactors, and transform each chractors into a vector. Extract n-grams of words, and transform each n-grams into a vector. There are many ways one can convert text to vector and it depends on what models one is using along with time or resources utilization. Typical workflow to prepare text data for machine learning models:

Tokenization One-Hot encoding or word indexing Pad sequencing Embedding layer (Word2Vec) Corresponding word vector In this notebook, we are going to explore and solve news classification problem to classify 41 types of news headlines and news descriptions. Use-case: Such text classification models are used in News Apps or by reporter to classify news topics for better reach to right audience.

Problem-statement: Build news classification model using deep learning teechniques and deploy model for reporters to classify and label news articles.

Reference: Deep learning with python by Francois Chollet

Table of contents:

1. Importing libraries
2. Data exploration and pre-processing
    - Profiling report of dataset
    - Text data visualization
    - Text data pre-processing
3. Tokenization and vectorization
4. What are the word embeddings?
5. Model training using EmbeddingLayer,RNNs and LSTMs
    - Baseline model 1
    - What are the RNNs?
    - Model training using LSTM
    - Loss and accuracy curve
6. Conclusion

Importing libraries

```
!pip install pandas-profiling
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pandas-profiling
  Downloading pandas_profiling-3.6.6-py2.py3-none-any.whl (324 kB)
                                           324.4/324.4 kB 14.7 MB/s eta 0:00:00
Collecting ydata-profiling (from pandas-profiling)
  Downloading ydata_profiling-4.1.2-py2.py3-none-any.whl (345 kB)
                                           345.9/345.9 kB 25.6 MB/s eta 0:00:00
Collecting scipy<1.10,>=1.4.1 (from ydata-profiling->pandas-profiling)
  Downloading scipy-1.9.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (33.7 MB)
                                           33.7/33.7 MB 24.4 MB/s eta 0:00:00
Requirement already satisfied: pandas!=1.4.0,<1.6,>1.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling) (1.5.3)
Collecting matplotlib<3.7,>=3.2 (from ydata-profiling->pandas-profiling)
  Downloading matplotlib-3.6.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.8 MB)
                                           11.8/11.8 MB 119.1 MB/s eta 0:00:00
Requirement already satisfied: pydantic<1.11,>=1.8.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling) (1.10.7)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling) (6.0)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling) (3.1.2)
Collecting visions[type_image_path]==0.7.5 (from ydata-profiling->pandas-profiling)
  Downloading visions-0.7.5-py3-none-any.whl (102 kB)
                                           102.7/102.7 kB 10.5 MB/s eta 0:00:00
Requirement already satisfied: numpy<1.24,>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling) (1.22.4)
Collecting htmlmin==0.1.12 (from ydata profiling->pandas profiling)
  Downloading htmlmin-0.1.12.tar.gz (19 kB)
  Preparing metadata (setup.py) ... done
Collecting phik<0.13,>=0.11.1 (from ydata-profiling->pandas-profiling)
  Downloading phik-0.12.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (679 kB)
                                           679.5/679.5 kB 62.4 MB/s eta 0:00:00
Requirement already satisfied: requests<2.29,>=2.24.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling) (2.27.1)
Collecting tqdm<4.65,>=4.48.2 (from ydata-profiling->pandas-profiling)
  Downloading tqdm-4.64.1-py2.py3-none-any.whl (78 kB)
                                           78.5/78.5 kB 10.5 MB/s eta 0:00:00
Requirement already satisfied: seaborn<0.13,>=0.10.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling) (0.12.2)
Collecting multimethod<1.10,>=1.4 (from ydata-profiling->pandas-profiling)
  Downloading multimethod-1.9.1-py3-none-any.whl (10 kB)
Requirement already satisfied: statsmodels<0.14,>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling) (0.13.5)
Collecting typeguard<2.14,>=2.13.2 (from ydata-profiling->pandas-profiling)
  Downloading typeguard-2.13.3-py3-none-any.whl (17 kB)
Collecting imagehash==4.3.1 (from ydata-profiling->pandas-profiling)
  Downloading ImageHash-4.3.1-py2.py3-none-any.whl (296 kB)
                                           296.5/296.5 kB 35.7 MB/s eta 0:00:00
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.10/dist-packages (from imagehash==4.3.1->ydata-profiling->pandas-profiling) (1.4.1)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from imagehash==4.3.1->ydata-profiling->pandas-profiling) (8.4.0)
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.10/dist-packages (from visions[type_image_path]==0.7.5->ydata-profiling->pandas-profiling) (23.1.0)
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.10/dist-packages (from visions[type_image_path]==0.7.5->ydata-profiling->pandas-profiling) (3.1)
Collecting tangled-up-in-unicode>=0.0.4 (from visions[type_image_path]==0.7.5->ydata-profiling->pandas-profiling)
  Downloading tangled_up_in_unicode-0.2.0-py3-none-any.whl (4.7 MB)
                                           4.7/4.7 MB 128.3 MB/s eta 0:00:00
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2<3.2,>=2.11.1->ydata-profiling->pandas-profiling) (2.1.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (1.0.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (4.39.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (23.1)
Requirement already satisfied: pyparsing>=2.2.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.7,>=3.2->ydata-profiling->pandas-profiling) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas!=1.4.0,<1.6,>1.1->ydata-profiling->pandas-profiling) (2022.7.1)
```

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import re
import nltk
import pandas_profiling as pp
from markupsafe import escape
from wordcloud import WordCloud, STOPWORDS
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
import tensorflow as tf
from tensorflow import keras
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.text import Tokenizer
from sklearn.model_selection import train_test_split, cross_val_score
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

# model building imports
from keras.layers import Embedding, Flatten, Dense, Dropout
from keras.layers import Conv1D, SimpleRNN, Bidirectional, MaxPooling1D, GlobalMaxPool1D, LSTM, GRU
from keras.models import Sequential
from keras.regularizers import L1L2

%matplotlib inline

# matplotlib defaults
plt.style.use("ggplot")
plt.rc("figure", autolayout=True)
plt.rc(
    "axes",
    labelweight="bold",
    labelsize="large",
    titleweight="bold",
    titlesize=14,
    titlepad=10,
)

import warnings
warnings.filterwarnings('ignore')
nltk.download('omw-1.4')
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

```
<ipython-input-2-a536228eef5d>:8: DeprecationWarning: `import pandas_profiling` is going to be deprecated by April 1st. Please use `import ydata_profiling` instead.
  import pandas_profiling as pp
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
```

Data exploration and pre-processing

```python
df = pd.read_json('https://raw.githubusercontent.com/Sagor96/ds/main/dataset/News_Category_Dataset_v3.json', lines=True)
df.head()
```

	link	headline	category	short_description	authors	date
0	https://www.huffpost.com/entry/covid-boosters-...	Over 4 Million Americans Roll Up Sleeves For O...	U.S. NEWS	Health experts said it is too early to predict...	Carla K. Johnson, AP	2022-09-23
1	https://www.huffpost.com/entry/american-airlin...	American Airlines Flyer Charged, Banned For Li...	U.S. NEWS	He was subdued by passengers and crew when he ...	Mary Papenfuss	2022-09-23
2	https://www.huffpost.com/entry/funniest-tweets...	23 Of The Funniest Tweets About Cats And Dogs ...	COMEDY	"Until you have a dog you don't understand wha...	Elyse Wanshel	2022-09-23
3	https://www.huffpost.com/entry/funniest-parent...	The Funniest Tweets From Parents This Week (Se...	PARENTING	"Accidentally put grown-up toothpaste on my to...	Caroline Bologna	2022-09-23
4	https://www.huffpost.com/entry/amy-cooper-lose...	Woman Who Called Cops On Black Bird-Watcher Lo...	U.S. NEWS	Amy Cooper accused investment firm Franklin Te...	Nina Golgowski	2022-09-22

```python
# shape of the dataset
print(df.shape)
# total number of unique categories
print("Unique categories:",df['category'].nunique())
print("-------------------------------------------------")
# information about metadata
df.info()
```

```
(209527, 6)
Unique categories: 42
-------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209527 entries, 0 to 209526
Data columns (total 6 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   link               209527 non-null  object
 1   headline           209527 non-null  object
 2   category           209527 non-null  object
 3   short_description  209527 non-null  object
 4   authors            209527 non-null  object
 5   date               209527 non-null  datetime64[ns]
dtypes: datetime64[ns](1), object(5)
memory usage: 9.6+ MB
```

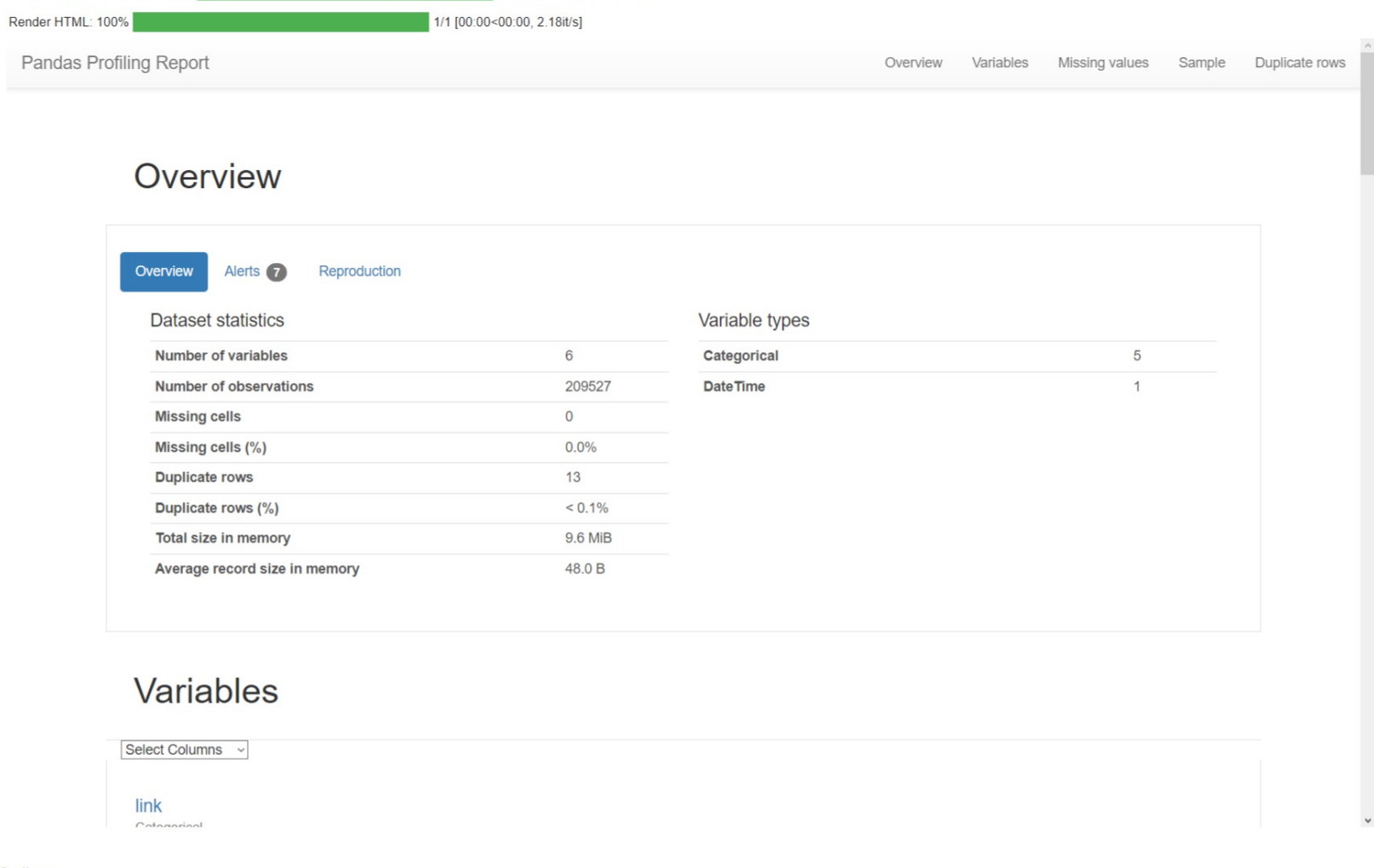
```python
# general descrption of dataset
df.describe().style.set_properties(**{'background-color': '#F0F0F0',
                                     'color': '#222222',
                                     'border': '1.5px  solid black'})
```

	link	headline	category	short_description	authors	date
count	209527	209527	209527	209527	209527	209527
unique	209486	207996	42	187022	29169	3890
top	https://www.huffingtonpost.comhttps://www.washingtonpost.com/politics/divisions-within-gop-over-trumps-candidacy-are-growing/2016/02/28/97b16010-de3a-11e5-8d98-4b3d9215ade1_story.html	Sunday Roundup	POLITICS			2014-03-25 00:00:00
freq	2	90	35602	19712	37418	100
first	nan	nan	nan	nan	nan	2012-01-28 00:00:00
last	nan	nan	nan	nan	nan	2022-09-23 00:00:00

Profiling report of dataset

```python
# produce pandas profiling report
df.profile_report()
```

```
Summarize dataset: 100% 15/15 [00:25<00:00, 1.12it/s, Completed]
Generate report structure: 100% 1/1 [00:02<00:00, 2.81s/it]
Render HTML: 100% 1/1 [00:00<00:00, 2.18s/it]
```



Pandas Profiling Report

Overview

Dataset statistics

Number of variables	6
Number of observations	209527
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	13
Duplicate rows (%)	< 0.1%
Total size in memory	9.6 MiB
Average record size in memory	48.0 B

Variable types

Categorical	5
DateTime	1

Variables

Select Columns

link

Key findings:

Dataset has total 42 distinct categories of news articles 'Politics' is the most common category of news in our dataset. We have total of 29169 unique authors who have written various news articles Maximum length of headline is 320 while median length is around 152. Maximum length of description is 1472 while median length is around 120.

Text data visualization

```python
# drop columns like authors, links and date as they are irrelevent to our problem.
new_df = df.drop(columns=['authors','link','date'])
new_df.head()
```

	headline	category	short_description
0	Over 4 Million Americans Roll Up Sleeves For O...	U.S. NEWS	Health experts said it is too early to predict...
1	American Airlines Flyer Charged, Banned For Li...	U.S. NEWS	He was subdued by passengers and crew when he ...
2	23 Of The Funniest Tweets About Cats And Dogs ...	COMEDY	"Until you have a dog you don't understand wha...
3	The Funniest Tweets From Parents This Week (Se...	PARENTING	"Accidentally put grown-up toothpaste on my to...
4	Woman Who Called Cops On Black Bird-Watcher Lo...	U.S. NEWS	Amy Cooper accused investment firm Franklin Te...

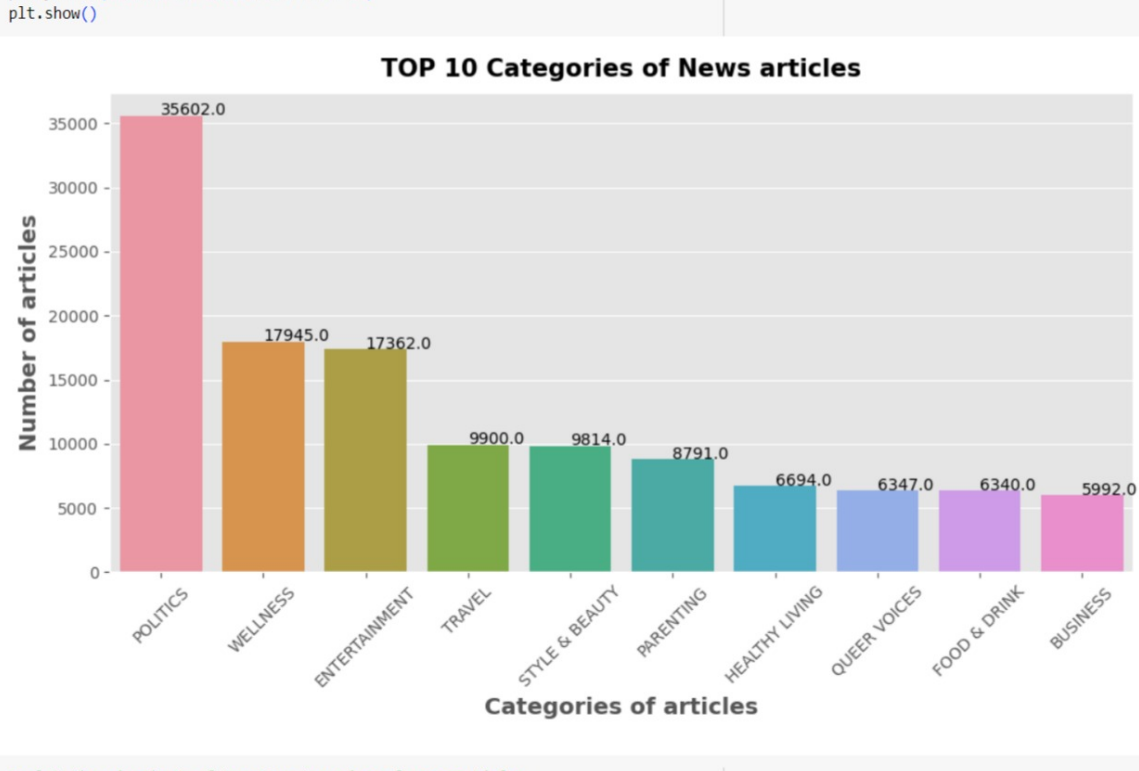
TOP 10 Categories of News Articles

```python
# Converting the json to data frame
base_df = pd.read_json('https://raw.githubusercontent.com/Sagor96/ds/main/dataset/News_Category_Dataset_v3.json', lines=True)
base_df.head(5)

# Dropping the other not relevant data
base_df = base_df.drop(['authors','link','date','short_description'], axis = 1)
base_df.head(3)
```

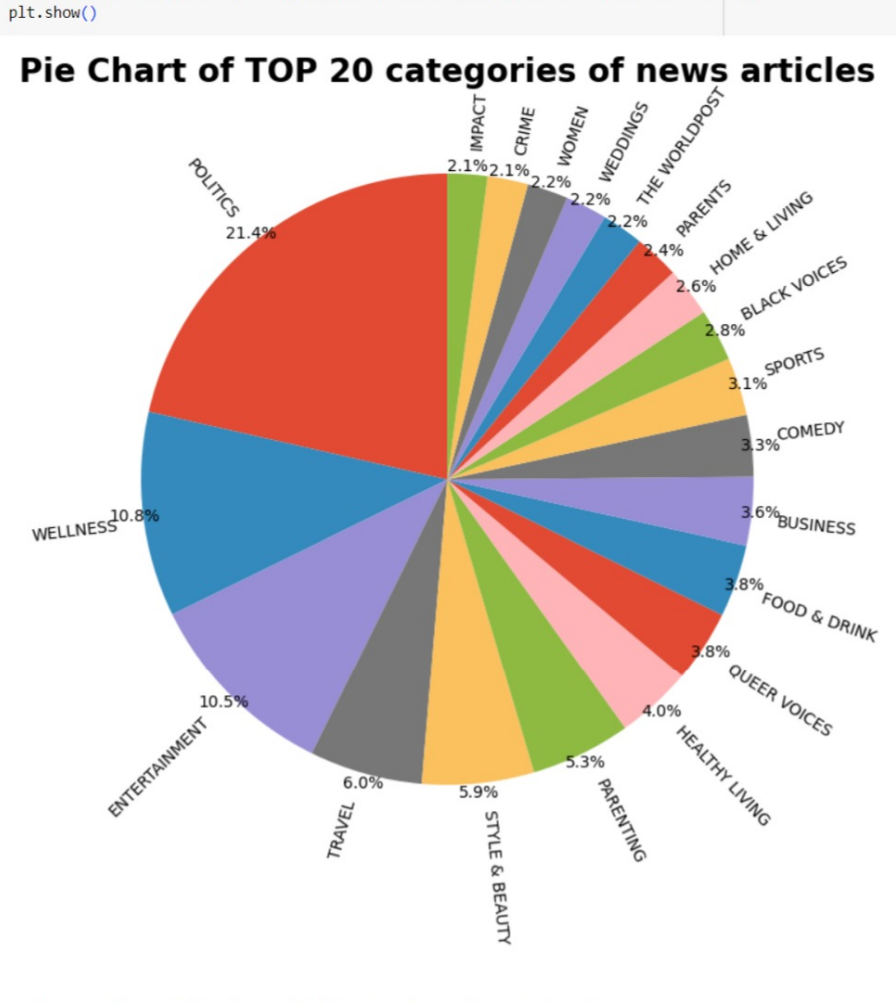
	headline	category
0	Over 4 Million Americans Roll Up Sleeves For O...	U.S. NEWS
1	American Airlines Flyer Charged, Banned For Li...	U.S. NEWS
2	23 Of The Funniest Tweets About Cats And Dogs ...	COMEDY

```python
# Create a DataFrame with category counts
cat_df = pd.DataFrame(new_df['category'].value_counts()).reset_index()
cat_df.rename(columns={'index':'news_classes','category':'numcat'}, inplace=True)
```

```python
# Visualize top 10 categories and proportion of each category in the dataset
plt.figure(figsize=(10,6))
ax = sns.barplot(x=cat_df.news_classes[:10], y=cat_df.numcat[:10])
for p in ax.patches:
    ax.annotate(p.get_height(), (p.get_x()+0.4, p.get_height() + 50))
plt.title("TOP 10 Categories of News articles", size=15)
plt.xlabel("Categories of articles", size=14)
plt.xticks(rotation=45)
plt.ylabel("Number of articles", size=14)
plt.show()
```



```python
# plot the pie chart of top 20 categories of news articles
fig = plt.figure(figsize=(12,8))
A = plt.pie(cat_df['numcat'][:20],
            labels=cat_df['news_classes'][:20],
            autopct='%1.1f%%',
            startangle=90,
            labeldistance=1.08,
            pctdistance=1.03,
            rotatelabels=45
            )

plt.title("Pie Chart of TOP 20 categories of news articles", size=20, weight='bold')
plt.show()
```



Lengths of 'headline' and 'short_description' of each category

```python
# create final dataframe of combined headline and short_description
final_df = new_df.copy()
final_df['length_of_news'] = final_df['headline'] + final_df['short_description']
final_df.drop(['headline','short_description'], inplace=True, axis=1)
final_df['len_news'] = final_df['length_of_news'].map(lambda x: len(x))
final_df.head()
```

	category	length_of_news	len_news
0	U.S. NEWS	Over 4 Million Americans Roll Up Sleeves For O...	230
1	U.S. NEWS	American Airlines Flyer Charged, Banned For Li...	248
2	COMEDY	23 Of The Funniest Tweets About Cats And Dogs ...	133
3	PARENTING	The Funniest Tweets From Parents This Week (Se...	215
4	U.S. NEWS	Woman Who Called Cops On Black Bird-Watcher Lo...	233

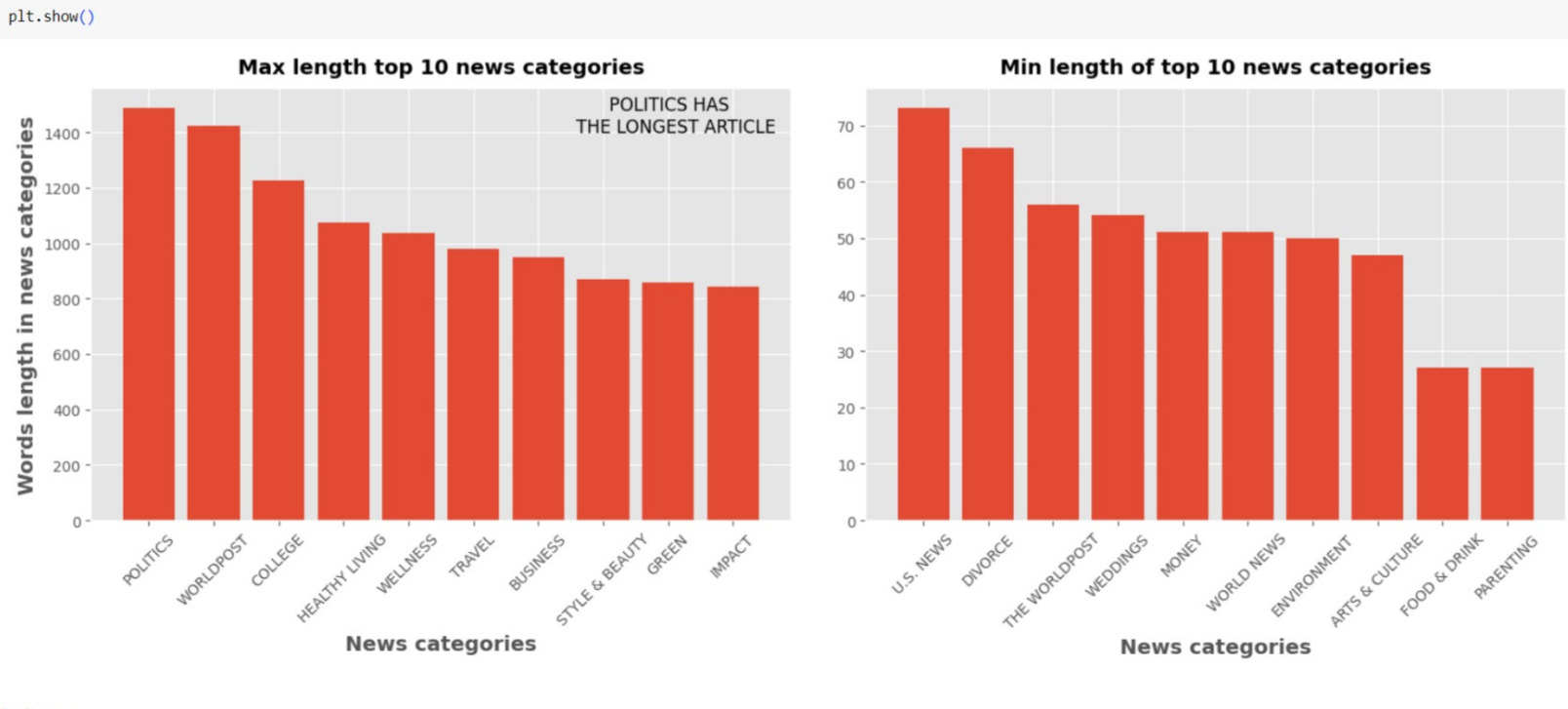
```python
# maximum length of news in each category
lenmax_df = final_df.groupby('category')['len_news'].max().reset_index().sort_values(by='len_news',ascending=False)
lenmax_df.head()
```

	category	len_news
24	POLITICS	1486
41	WORLDPOST	1424
4	COLLEGE	1225
16	HEALTHY LIVING	1073
38	WELLNESS	1036

```python
# minimum lenght of news in each category
lenmin_df = final_df.groupby('category')['len_news'].min().reset_index().sort_values(by='len_news',ascending=False)
lenmin_df.head()
```

	category	len_news
35	U.S. NEWS	73
8	DIVORCE	66
33	THE WORLDPOST	56
36	WEDDINGS	54
21	MONEY	51

```python
# plot the bar plots of max and min length of news articles
plt.figure(figsize=(15,6))

plt.subplot(121)
plt.bar(lenmax_df['category'][:10],lenmax_df['len_news'][:10])
plt.xticks(rotation=45)
plt.ylabel("Words length in news categories", size=14)
plt.xlabel("News categories",  size=14)
plt.text(6.5,1400, '''       POLITICS HAS
 THE LONGEST ARTICLE''', fontsize=12)
plt.title("Max length top 10 news categories", size=14)

plt.subplot(122)
plt.bar(lenmin_df['category'][:10], lenmin_df['len_news'][:10])
plt.xticks(rotation=45)
plt.xlabel("News categories", size=14)
plt.title("Min length of top 10 news categories", size=14)

plt.show()
```



Key findings:

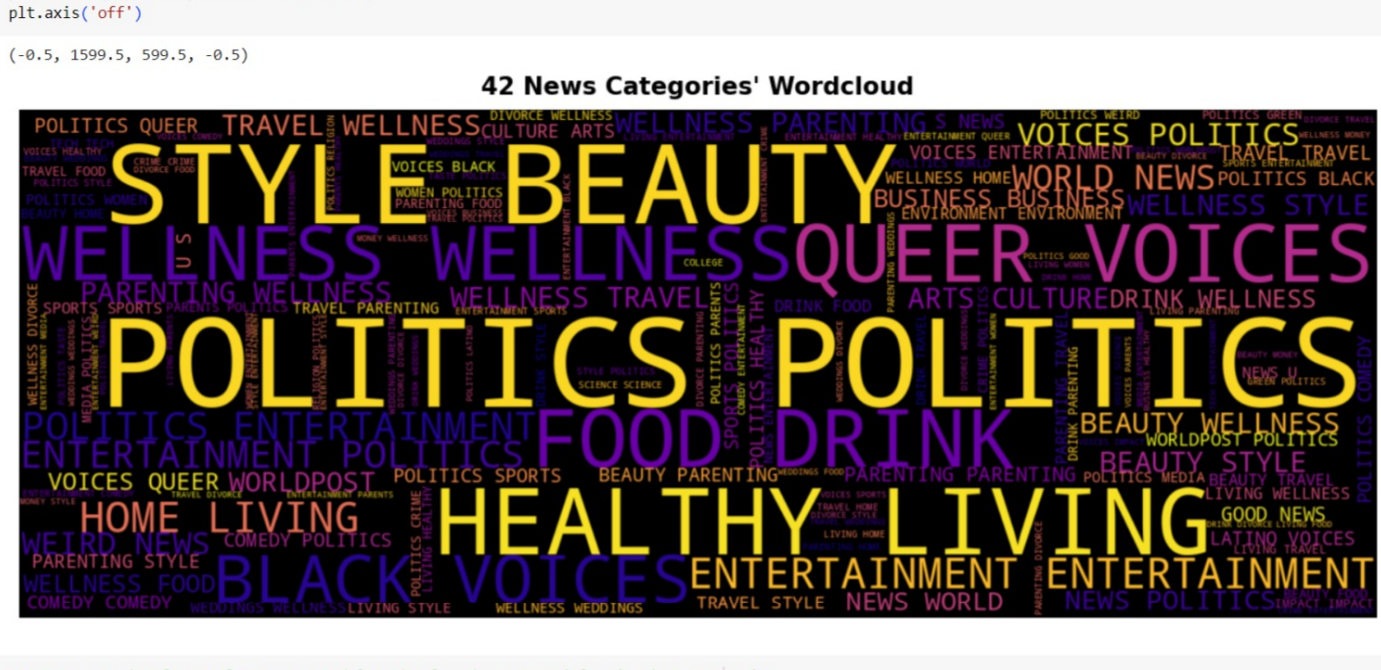
From all the categories of news POLITICS has the most number of articles as well as length of characters in the article (headlines and short-description) After 'POLITICS' category, 'WORLDPOST', 'COLLEGE', 'HEALTHY LIVING','WELLNESS' categories are in top 5 most length of articles 'POLITICS', 'WELLNESS' and 'ENTERTAINMENT' are top 3 categories having most number of articles in out dataset.

Wordclouds of categories and news articles

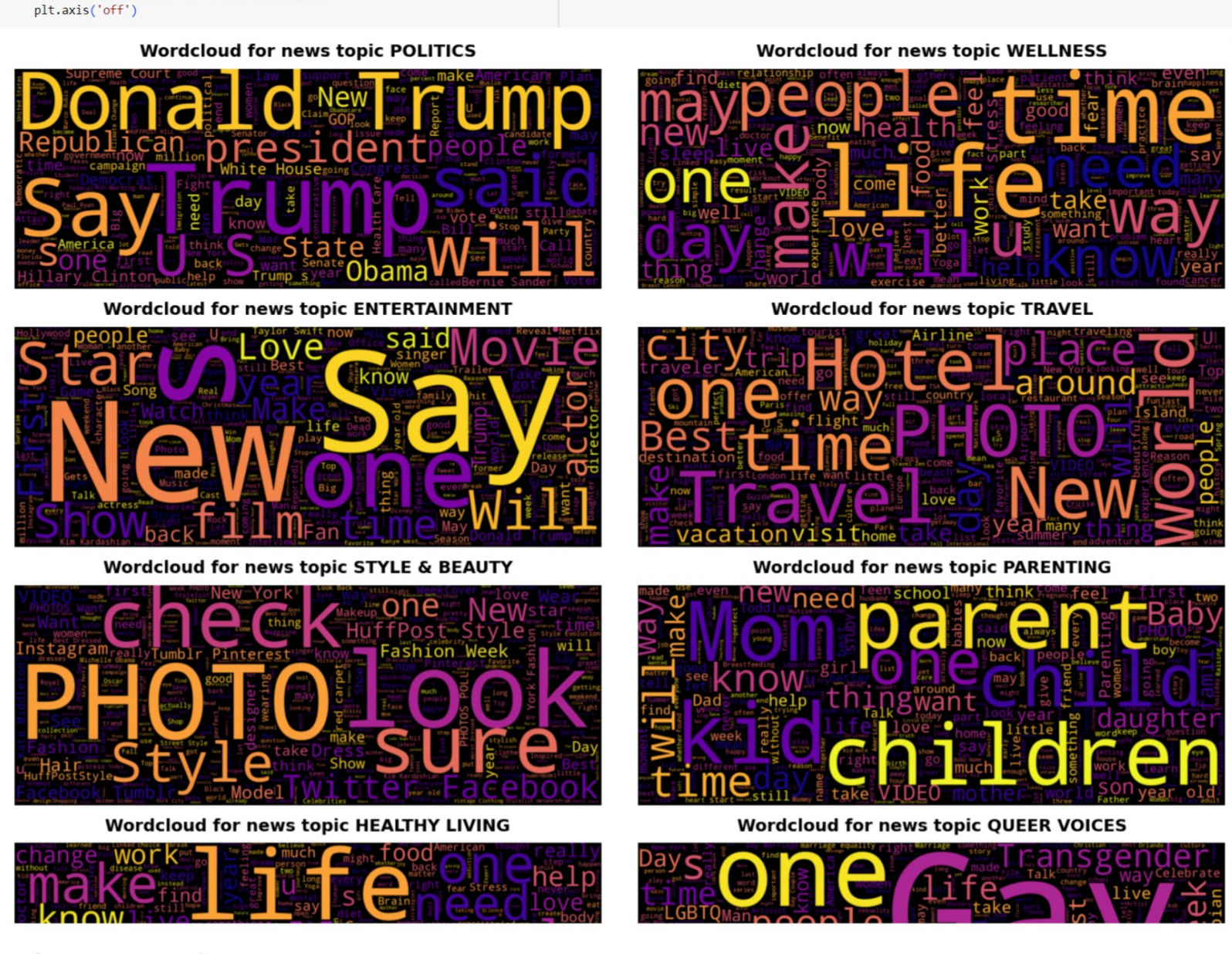
```python
# wordcloud of categories of news articles in our dataset
plt.figure(figsize=(12,12))
wc = WordCloud(max_words=1000,
               min_font_size=10,
               height=600,
               width=1600,
               background_color='black',
               contour_color='black',
               colormap='plasma',
               repeat=False,
               stopwords=STOPWORDS).generate(' '.join(final_df.category))

plt.title("42 News Categories' Wordcloud", size=15, weight='bold')
plt.imshow(wc, interpolation= "bilinear")
plt.axis('off')
```

```
(-0.5, 1599.5, 599.5, -0.5)
```

42 News Categories' Wordcloud



```python
# create new dataframe of category and length of each news articles in that categories
ndf = final_df.copy()
ndf.drop('len_news', inplace=True, axis=1)

# list of top 10  categories in out dataset
categories = cat_df['news_classes'][:10].to_list()

# list of news articles of each top 10 categories list
articles_list = []

for i in categories:
    cat_ndf = ndf[ndf['category'] == i]
    cat_array = cat_ndf['length_of_news'].values  # array of news articles text in each category
    articles_list.append(cat_array)

# create a wordcloud instance
wc1 = WordCloud(max_words=1000,
               min_font_size=10,
               height=600,
               width=1600,
               background_color='black',
               contour_color='black',
               colormap='plasma',
               repeat=True,
               stopwords=STOPWORDS)

# plot the figure of 10 wordcloud from out dataset
plt.figure(figsize=(15,15))

for idx, j in enumerate(categories):
    plt.subplot(5,2,idx+1)
    cloud = wc1.generate(' '.join(articles_list[idx]))
    plt.imshow(cloud, interpolation= "bilinear")
    plt.title(f"Wordcloud for news topic {j}")
    plt.axis('off')
```



Wordcloud for news topic POLITICS

Wordcloud for news topic WELLNESS

Wordcloud for news topic ENTERTAINMENT

Wordcloud for news topic TRAVEL

Wordcloud for news topic STYLE & BEAUTY

Wordcloud for news topic PARENTING

Wordcloud for news topic HEALTHY LIVING

Wordcloud for news topic QUEER VOICES

Text-data Preprocessing

```python
# start with removing some data from politics dataframe ndf
politics_list = list(ndf[ndf['category'] == 'POLITICS'].index)
list_16 = politics_list[:16000] # list of 16000 row labels of POLITICS category

# drop these 16000 labels from the dataset
ndf2 = ndf.copy()
ndf2.drop(list_16, axis=0, inplace=True)
```

```python
# clean the text data using regex and data cleaning function
def datacleaning(text):
    whitespace = re.compile(r"\s+")
    user = re.compile(r"(?i)@[a-z0-9_]+")
    text = whitespace.sub(' ', text)
    text = user.sub('', text)
    text = re.sub(r"\[[^()]*\]","", text)
    text = re.sub("\d+", "", text)
    text = re.sub(r'[^\w\s]','',text)
    text = re.sub(r"(?:@\S*|#\S*|http(?=.*://)\S*)", "", text)
    text = text.lower()

    # removing stop-words
    text = [word for word in text.split() if word not in list(STOPWORDS)]

    # word lemmatization
    sentence = []
    for word in text:
        lemmatizer = WordNetLemmatizer()
        sentence.append(lemmatizer.lemmatize(word,'v'))

    return ' '.join(sentence)
```

```python
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```python
# Example of pre-processing using above function
print("Text sentence before pre-processing:\n",ndf2['length_of_news'][0])
print("---"*35)
print("Text sentence after pre-processing:\n",datacleaning(ndf2['length_of_news'][0]))
```

```
Text sentence before pre-processing:
 Over 4 Million Americans Roll Up Sleeves For Omicron-Targeted COVID BoostersHealth experts said it is too early to predict whether demand would match up with the 171 million doses
---------------------------------------------------------------------------------------------------------
Text sentence after pre-processing:
 million americans roll sleeves omicrontargeted covid boostershealth experts say early predict whether demand match million dose new boosters us order fall
```

```python
# apply datacleaning function to column 'length_of_news'
ndf2['length_of_news'] = ndf2['length_of_news'].apply(datacleaning)
```

```python
# length of total characters before and after cleaning text data
old_length = ndf['length_of_news'].apply(len).sum()
new_length = ndf2['length_of_news'].apply(len).sum()
print(f"Old chracters length of text data: {old_length}")
print(f"New chracters length of text data: {new_length}")
```

```
Old chracters length of text data: 36169394
New chracters length of text data: 23045855
```

```python
# length of total words before and after cleaning text data
old_text = ndf['length_of_news'].apply(lambda x: len([i for i in x.split()])).sum()
new_text = ndf2['length_of_news'].apply(lambda x: len([i for i in x.split()])).sum()
print(f"Old word length of text data: {old_text}")
print(f"New word length of text data: {new_text}")
```

```
Old word length of text data: 5942993
New word length of text data: 3290751
```

Tokenization and vectorization

Some of the most common NLP terminologies

1. Document

Each and every training example used in text dataset known as a Document.

2. Corpus

Collections of documents called as a corpus of text data

3. Vocabulary(BoW)

Vocabulary or Bag-of-words is nothing but number of unique words are present in text corpus.

4. Stop words

Stop words are those used most commonly in any language, e.g. 'the', 'a', etc. they do not form any meaning to the context of the text

5. N-grams

N-grams is text representation in form of N words sequences to extract meaning and context out of each sentences or paragraphs.

6. Tokenization

It's an early step in NLP process to split text sentences into smaller words or tokens.

7. Vectorization

Machine do not understand text or words, so text data or tokens must be converted to corrospinding word index or word vectors in order process text and build models. process of converting tokenized words into numerical vectors called as a vectorization.

One-hot encoding and word-indexing example on chunk of data

```python
# tokenization and vectorization workflow
# word level one-hot encoding for sample data

samples = list(ndf2['length_of_news'][:5].values)  # samples of first 5 documents of out dataset

token_index = {}  # builds an index of tokens in the data
for sample in samples:
    for word in sample.split():
        if word not in token_index:
            token_index[word] = len(token_index) + 1 # assigning unique index to each unique words

max_length = 15

results = np.zeros(shape=(len(samples),   # results will be stored in this array
                          max_length,
                          max(token_index.values()) +1))

print("Shape of stored results array:", results.shape)
print("Token index of unique words: \n", token_index)

for i, sample in enumerate(samples):
    for j, word in list(enumerate(sample.split()))[:max_length]:
        index = token_index.get(word)
        results[i,j,index] = 1
```

```
Shape of stored results array: (5, 15, 91)
Token index of unique words:
 {'million': 1, 'americans': 2, 'roll': 3, 'sleeves': 4, 'omicrontargeted': 5, 'covid': 6, 'boostershealth': 7, 'experts': 8, 'say': 9, 'early': 10, 'predict': 11, 'whether': 12,
```

One-hot encoding and indexing of train and test data

```python
# one hot encoding using keras tokenizer and pad sequencing
X = ndf2['length_of_news']
encoder = LabelEncoder()
y = encoder.fit_transform(ndf2['category'])
print("shape of input data: ", X.shape)
print("shape of target variable: ", y.shape)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

tokenizer = Tokenizer(num_words=100000, oov_token='<00V>')
tokenizer.fit_on_texts(X_train) # build the word index
# padding X_train text input data
train_seq = tokenizer.texts_to_sequences(X_train) # converts strinfs into integer lists
train_padseq = pad_sequences(train_seq, maxlen=20) # pads the integer lists to 2D integer tensor

# padding X_test text input data
test_seq = tokenizer.texts_to_sequences(X_test)
test_padseq = pad_sequences(test_seq, maxlen=20)

word_index = tokenizer.word_index
max_words = 150000  # total number of words to consider in embedding layer
total_words = len(word_index)
maxlen = 130 # max length of sequence
y_train = to_categorical(y_train, num_classes=42)
y_test = to_categorical(y_test, num_classes=42)
print("Length of word index:", total_words)
```

```
shape of input data:  (193527,)
shape of target variable:  (193527,)
Length of word index: 180960
```

What are the Wordembeddings?

A word embedding is a learned representation for text where words that have the same meaning and save similar representation

Reference:- Machinelearningmastery

This approach to representing words and documents that may be considered one of the key breakthroughs of deep learning on chllengin NLP problems Word embeddings are alternative to one-hot encoding along with dimensionality reduction One-hot word vectors - Sparse, High-dimensional and Hard-coded

Word embeddings - Dense, Lower-Dimensional and Learned from the data

Keras library has embeddings layer which does word representation of given text corpus tf.keras.layers.Embedding(input_dim, output_dim, embeddings_initializer='uniform', embeddings_regularizer=None, activity_regularizer=None, embeddings_constraint=None, mask_zero=False, input_length=None, kwargs)**

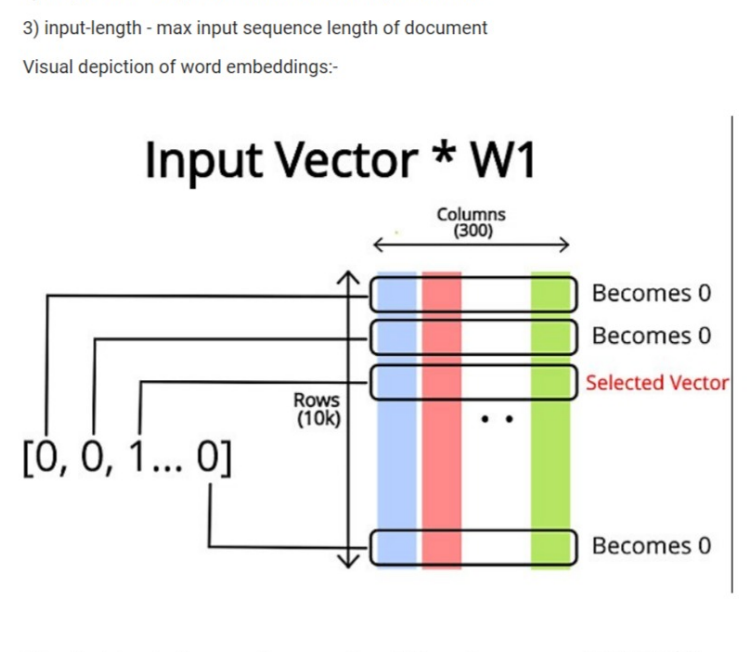
Key Arguments:

1) input_dim - Size of vocabulary - length of word index

2) output_dim - Output dimention of word representation

3) input-length - max input sequence length of document

Visual depiction of word embeddings:-



Model training using embedding layer and RNN (Baseline)

Baseline model 1 with SimpleRNN and embedding layer

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, SimpleRNN, Dropout, Dense

# Define the model
model = Sequential()
model.add(Embedding(total_words, 70, input_length=maxlen))
model.add(Bidirectional(SimpleRNN(64, dropout=0.1, recurrent_dropout=0.20, activation='tanh', return_sequences=True)))
model.add(Bidirectional(SimpleRNN(64, dropout=0.1, recurrent_dropout=0.30, activation='tanh', return_sequences=True)))
model.add(SimpleRNN(32, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(42, activation='softmax'))

# Print the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 130, 70)           12667200

 bidirectional (Bidirectiona  (None, 130, 128)         17280
 l)

 bidirectional_1 (Bidirectio  (None, 130, 128)         24704
 nal)

 simple_rnn_2 (SimpleRNN)    (None, 32)                5152

 dropout (Dropout)           (None, 32)                0

 dense (Dense)               (None, 42)                1386

=================================================================
Total params: 12,715,722
Trainable params: 12,715,722
Non-trainable params: 0
_________________________________________________________________
```

What are the Recurrent Neural Networks??

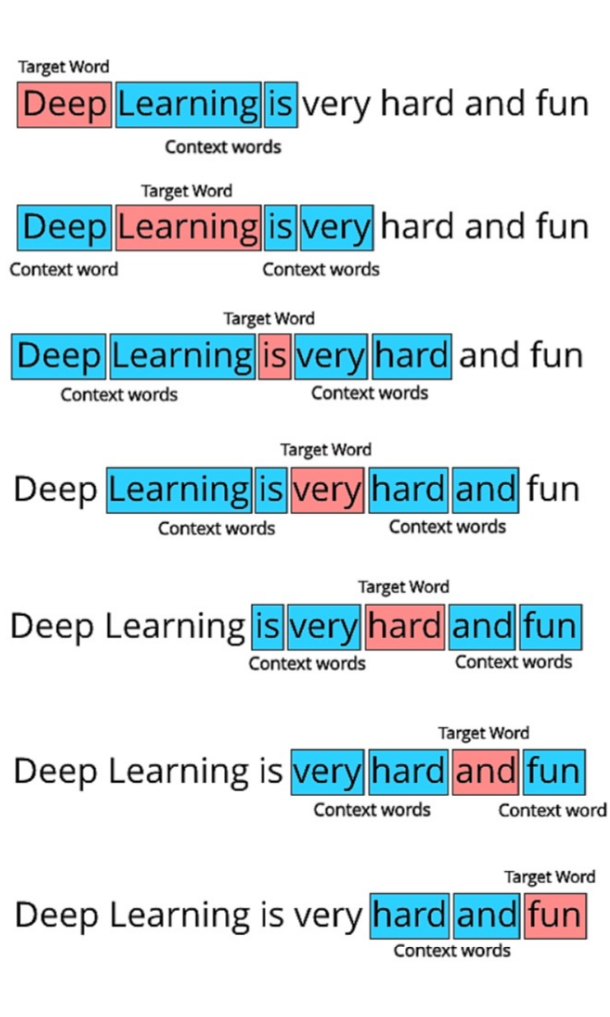
A major difference between densely connected neural network and recurrent nueral network, is that fully connected networks have no memory in units of each layer. while recurrent nueral networks do store state of previous timestep or sequence while assigning weights to current input.

In RNNs, we process inputs word by word or eye saccade but eye saccade - while keeping memories of what came before in each cells. this gives fluid representation of sequences and gives nueral network a ability to capture context of sequence rather than absolute representation of words.

"Recurrent nueral network processes sequences by iterating through the sequence elements and maintaining a a state containing information relative to what it has seen so far. In effect, an RNN is a type of nueral network that has an internal loop."

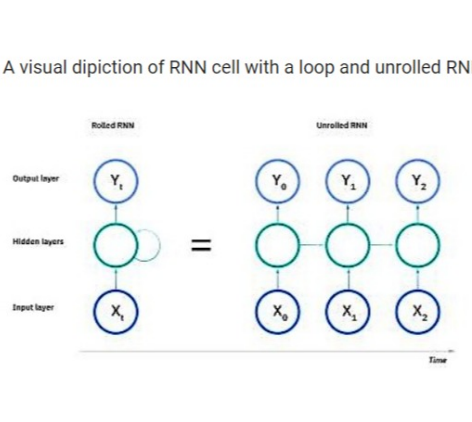
-6.2 Understanding recurrent nueral network, Deep learning using python by chollat

See the below depiction of how RNNs learns the context of sequences.



Target Word
Deep Learning is very hard and fun
Context words
Target Word
Deep Learning is very hard and fun
Context word
Context words
Target Word
Deep Learning is very hard and fun
Context words
Context words
Target Word
Deep Learning is very hard and fun
Context words
Context words
Target Word
Deep Learning is very hard and fun
Context words
Context words
Target Word
Deep Learning is very hard and fun
Context words
Context word
Target Word
Deep Learning is very hard and fun
Context words

A visual dipiction of RNN cell with a loop and unrolled RNN cell.



```python
X_train.shape
```

```
(154821,)
```

```python
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Pad sequences to the desired length
train_padseq = pad_sequences(train_padseq, maxlen=130)
```

```python
model.compile(optimizer='rmsprop',
          loss='categorical_crossentropy',
          metrics=['accuracy']
          )
# SETUP A EARLY STOPPING CALL and model check point API
earlystopping = keras.callbacks.EarlyStopping(monitor='accuracy',
                                              patience=5,
                                              verbose=1,
                                              mode='min'
                                              )
checkpointer = ModelCheckpoint(filepath='bestvalue',moniter='val_loss', verbose=0, save_best_only=True)
callback_list = [checkpointer, earlystopping]

# fit model to the data
history = model.fit(train_padseq, y_train,
                   batch_size=42,
                    epochs=15,
                    validation_split=0.2
                   )

# evalute the model
test_loss, test_acc = model.evaluate(test_padseq, y_test, verbose=0)
print("test loss and accuracy:", test_loss, test_acc)
```

```
Epoch 1/15
2949/2949 [==============================] - 713s 240ms/step - loss: 3.1150 - accuracy: 0.1972 - val_loss: 2.7261 - val_accuracy: 0.2898
Epoch 2/15
2949/2949 [==============================] - 721s 245ms/step - loss: 2.6441 - accuracy: 0.3140 - val_loss: 2.4588 - val_accuracy: 0.3556
Epoch 3/15
2949/2949 [==============================] - 736s 250ms/step - loss: 2.3800 - accuracy: 0.3798 - val_loss: 2.3340 - val_accuracy: 0.3940
Epoch 4/15
2949/2949 [==============================] - 725s 246ms/step - loss: 2.2461 - accuracy: 0.4194 - val_loss: 2.2524 - val_accuracy: 0.4267
Epoch 5/15
2949/2949 [==============================] - 721s 244ms/step - loss: 2.1680 - accuracy: 0.4422 - val_loss: 2.2035 - val_accuracy: 0.4365
Epoch 6/15
2949/2949 [==============================] - 711s 241ms/step - loss: 2.1102 - accuracy: 0.4584 - val_loss: 2.1824 - val_accuracy: 0.4462
Epoch 7/15
2949/2949 [==============================] - 714s 242ms/step - loss: 2.0725 - accuracy: 0.4657 - val_loss: 2.1372 - val_accuracy: 0.4547
Epoch 8/15
2949/2949 [==============================] - 714s 242ms/step - loss: 2.0309 - accuracy: 0.4762 - val_loss: 2.1821 - val_accuracy: 0.4462
Epoch 9/15
2949/2949 [==============================] - 712s 241ms/step - loss: 2.0068 - accuracy: 0.4828 - val_loss: 2.1510 - val_accuracy: 0.4548
Epoch 10/15
2949/2949 [==============================] - 735s 249ms/step - loss: 1.9900 - accuracy: 0.4857 - val_loss: 2.1340 - val_accuracy: 0.4557
Epoch 11/15
2949/2949 [==============================] - 714s 242ms/step - loss: 1.9588 - accuracy: 0.4937 - val_loss: 2.1689 - val_accuracy: 0.4529
Epoch 12/15
2949/2949 [==============================] - 708s 240ms/step - loss: 1.9319 - accuracy: 0.5009 - val_loss: 2.1349 - val_accuracy: 0.4591
Epoch 13/15
2949/2949 [==============================] - 730s 248ms/step - loss: 1.9139 - accuracy: 0.5060 - val_loss: 2.1393 - val_accuracy: 0.4587
Epoch 14/15
2949/2949 [==============================] - 709s 240ms/step - loss: 1.9089 - accuracy: 0.5067 - val_loss: 2.1350 - val_accuracy: 0.4607
Epoch 15/15
2949/2949 [==============================] - 727s 246ms/step - loss: 1.8991 - accuracy: 0.5110 - val_loss: 2.1390 - val_accuracy: 0.4550
test loss and accuracy: 2.1493613719940186 0.4507569968700409
```

Notes:

Experiment 1: Parameters: max_words=100000, output_dim=50, maxlen=50, epoch=10. Model is highly overfitting as training accuracy comes around 84% while test accuracy is barely 40%.

Experiment 2: Params: max_words=150000, out_dim=50, maxlen=80, epoch=10, added droput layers in RNN Model do not overfit but underfit and biased. val_acc comes again almost 39.3% while train accuracy is only 42%. model needs more parameters with more epochs

Experiment 3: Params: max_words=total_words, out_dim=70, maxlen=100, epoch=15, addding bidirection layer over both dropout RNN an improvement in train accuracy to 63% and val_accuracy to 49% max at epoch 13. after epoch 13 model started overfitting. (next step) model can be more complex with regularization, model is forgetting information after 3rd RNN layer due to lower number of units, this needs to be solved.

Model 2, training using Conv1D, Bi-directional RNN, LSTMs and GRU layer

```python
model2 = Sequential()
model2.add(Embedding(total_words, 1000, input_length=maxlen))
model2.add(Bidirectional(LSTM(64, dropout=0.1, recurrent_dropout=0.10, activation='tanh', return_sequences=True)))
model2.add(Bidirectional(LSTM(64, dropout=0.2, recurrent_dropout=0.20, activation='tanh', return_sequences=True)))
model2.add(Bidirectional(SimpleRNN(64, dropout=0.2, recurrent_dropout=0.20, activation='tanh', return_sequences=True)))
model2.add(Conv1D(72, 3, activation='relu'))
model2.add(MaxPooling1D(2))
model2.add(SimpleRNN(64, activation='tanh', dropout=0.2, recurrent_dropout=0.20, return_sequences=True))
model2.add(GRU(64, recurrent_dropout=0.20, recurrent_regularizer='l1_l2'))
model2.add(Dropout(0.2))
model2.add(Dense(42, activation='softmax'))
model2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 130, 1000)         180960000

 bidirectional_5 (Bidirectio  (None, 130, 128)         545280
 nal)

 bidirectional_6 (Bidirectio  (None, 130, 128)         98816
 nal)

 bidirectional_7 (Bidirectio  (None, 130, 128)         24704
 nal)

 conv1d_1 (Conv1D)           (None, 128, 72)           27720

 max_pooling1d_1 (MaxPooling  (None, 64, 72)           0
 1D)

 simple_rnn_6 (SimpleRNN)    (None, 64, 64)            8768

 gru_1 (GRU)                 (None, 64)                24960

 dropout_2 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 42)                2730

=================================================================
Total params: 181,692,978
Trainable params: 181,692,978
Non-trainable params: 0
_________________________________________________________________
```

```python
model2.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy']
              )
# SETUP A EARLY STOPPING CALL and model check point API
earlystopping = keras.callbacks.EarlyStopping(monitor='accuracy',
                                              patience=5,
                                              verbose=1,
                                              mode='min'
                                              )
checkpointer = ModelCheckpoint(filepath='bestvalue1',moniter='val_loss', verbose=0, save_best_only=True)
callback_list = [checkpointer, earlystopping]

# fit model to the data
history2 = model2.fit(train_padseq, y_train,
                      batch_size=130,
                      epochs=15,
                      validation_split=0.2,
                      shuffle=True
                     )


# evalute the model
test_loss2, test_acc2 = model2.evaluate(test_padseq, y_test, verbose=0)
print("test loss and accuracy:", test_loss2, test_acc2)
```

```
120/953 [==>...........................] - ETA: 1:03:53 - loss: 1.8158 - accuracy: 0.5351
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-38-65f21910211d> in <cell line: 15>()
     13
     14 # fit model to the data
---> 15 history2 = model2.fit(train_padseq, y_train,
     16                       batch_size=130,
     17                       epochs=15,

                                ⌃ 8 frames
/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/execute.py in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     50   try:
     51     ctx.ensure_initialized()
---> 52     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     53                                         inputs, attrs, num_outputs)
     54   except core._NotOkStatusException as e:

KeyboardInterrupt:

SEARCH STACK OVERFLOW
```

Notes:

Experiment 4: params: max_words=total_words, out_dim=70, maxlen=100, epoch=15, in this case out train and test accuracy improved but model is again overfitting as train accuracy is at 74% while test accuray is at 49%. to avoid overfitting we need ot increase maxlen and output dimensions of vector and also adding CNN1D layer after 3rd RNN with gloablmax pooling with regularization should help, also GRU can be considered

Experiment 5: params: max_words=total_words, out_dim=100, maxlen=130, epoch=15, train accuracy=71%, test accuracy=49%, not any significant difference than previous model, (next up)add GRU, cnn1d with globalmaxpooling at the end, with shuffling of train data and run the notebook again

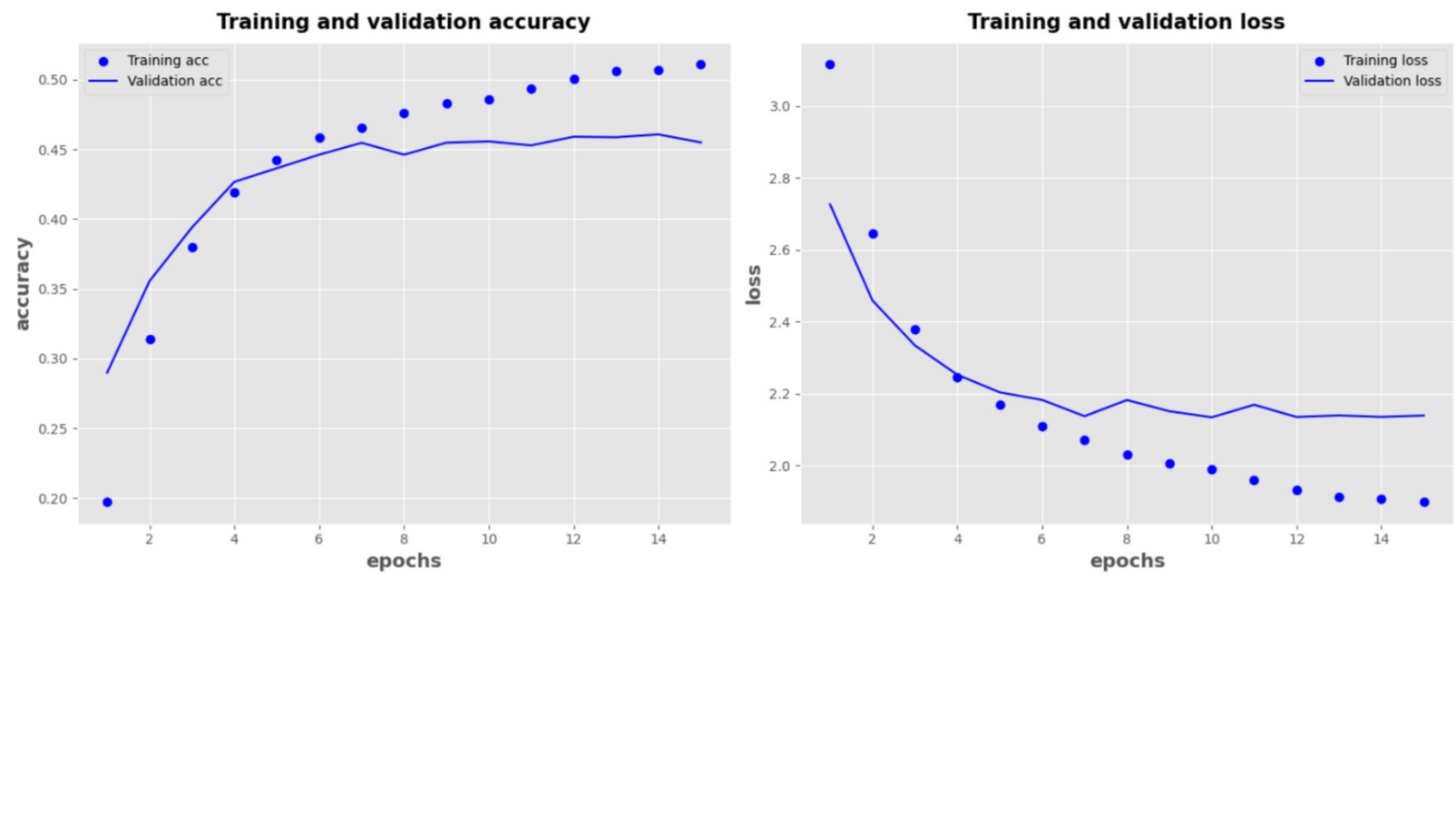
Experiment 6: params: max_words=total_words, out_dim=100, maxlen=130, epoch=15, train accuracy=85%, test accuracy=50%, we used LSTMs and GRU to process long sequences and retain previous inputs at perticuler input. while train accuracy improved significantly, test accuracy is still at 50%, a marginal improvement. Text data is always hard problem owing distribution on inputs. NIP models generally requires huge data to gain maximum accuracy on testing data. pre-trained models like BERT, Roberta can be used to overcome these challanges.\

```python
model.save("new_clf_exp06.h5")
```

learning curve of model 2

```python
def plot_loss_accuracy(history):

    # create object of arrays of accuracy and loss
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    # number of epochs in our model
    epochs = range(1 ,len(acc) + 1)

    # call matplolib figure object and plot loss and accuracy curves
    plt.figure(figsize=(15,6))

    plt.subplot(121)
    plt.plot(epochs, acc, 'bo', label='Training acc')
    plt.plot(epochs, val_acc, 'b', label='Validation acc')
    plt.title("Training and validation accuracy", fontsize=15)
    plt.xlabel('epochs', fontsize=14)
    plt.ylabel("accuracy", fontsize=14)
    plt.legend()

    plt.subplot(122)
    plt.plot(epochs, loss, 'bo', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title("Training and validation loss", fontsize=15)
    plt.xlabel('epochs', fontsize=14)
    plt.ylabel("loss", fontsize=14)
    plt.legend()

    plt.show()
```

```python
plot_loss_accuracy(history)
```



Colab paid products - Cancel contracts here