Dataset Information

Develop a Deep learning program to identify when an article might be fake news.

Attributes

- id: unique id for a news article
- title: the title of a news article
- author: author of the news article
- text: the text of the article; could be incomplete
- label: a label that marks the article as potentially unreliable
  - 1: unreliable
  - 0: reliable

Import Modules

In [21]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import re
import nltk
import warnings
%matplotlib inline

warnings.filterwarnings('ignore')
```

Loading the Dataset

In [10]:
```
df = pd.read_csv('train.csv')
df.head()
```

Out[10]:

	id	title	author	text	label
0	0	House Dem Aide: We Didn't Even See Comey's Let...	Darrell Lucus	House Dem Aide: We Didn't Even See Comey's Let...	1
1	1	FLYNN: Hillary Clinton, Big Woman on Campus - ...	Daniel J. Flynn	Ever get the feeling your life circles the rou...	0
2	2	Why the Truth Might Get You Fired	Consortiumnews.com	Why the Truth Might Get You Fired October 29, ...	1
3	3	15 Civilians Killed In Single US Airstrike Hav...	Jessica Purkiss	Videos 15 Civilians Killed In Single US Airstr...	1
4	4	Iranian woman jailed for fictional unpublished...	Howard Portnoy	Print \nAn Iranian woman has been sentenced to...	1

In [5]:
```
df['title'][0]
```

Out[5]:
```
'House Dem Aide: We Didn' t Even See Comey' s Letter Until Jason Chaffetz Tweeted It'
```

In [6]:
```
df['text'][0]
```

Out[6]:
```
'House Dem Aide: We Didn't Even See Comey's Letter Until Jason Chaffetz Tweeted It By Darrell Lucus on October 30, 2016 Subscribe Jason Chaffetz on the stump in American Fork, Utah ( image courtesy Michael Jolley, available under a Creative Commons-BY license) \nWith apologies to Keith Olbermann, there is no doubt who the Worst Person in The World is this week - FBI Director James Comey. But according to a House Democratic aide, it looks like we also know who the second-worst person is as well. It turns out that when Comey sent his now-infamous letter announcing that the FBI was looking into emails that may be related to Hillary Clinton's email server, the ranking Democrats on the relevant committees didn't hear about it from Comey. They found out via a tweet from one of the Republican committee chairmen. \nAs we now know, Comey notified the Republican chairmen and Democratic ranking members of the House Intelligence, Judiciary, and Oversight committees that his agency was reviewing emails it had recently discovered in order to see if they contained classified information. Not long after this letter went out, Oversight Committee Chairman Jason Chaffetz set the political world ablaze with this tweet. FBI Dir just informed me, "The FBI has learned of the existence of emails that appear to be pertinent to the investigation." Case reopened \n— Jason Chaffetz (@jasoninthehouse) October 28, 2016 \nOf course, we now know that this was not the case . Comey was actually saying that it was reviewing the emails in light of "an unrelated case" - which we now know to be Anthony Weiner's sexting with a teenager. But apparently such little things as facts didn't matter to Chaffetz. The Utah Republican had already vowed to initiate a raft of investigations if Hillary wins - at least two years' worth, and possibly an entire term's worth of them. Apparently Chaffetz thought the FBI was already doing his work for him - resulting in a tweet that briefly roiled the nation before cooler heads realized it was a dud. \nBut according to a senior House Democratic aide, misreading that letter may have been the least of Chaffetz' sins. That aide told Shareblue that his boss and other Democrats didn't even know about Comey's letter at the time - and only found out when they checked Twitter. "Democratic Ranking Members on the relevant committees didn't receive Comey's letter until after the Republican Chairmen. In fact, the Democratic Ranking Members didn' receive it until after the Chairman of the Oversight and Government Reform Committee, Jason Chaffetz, tweeted it out and made it public." \nSo let's see if we've got this right. The FBI director tells Chaffetz and other GOP committee chairmen about a major development in a potentially politically explosive investigation, and neither Chaffetz nor his other colleagues had the courtesy to let their Democratic counterparts know about it. Instead, according to this aide, he made them find out about it on Twitter. \nThere has already been talk on Daily Kos that Comey himself provided advance notice of this letter to Chaffetz and other Republicans, giving them time to turn on the spin machine. That may make for good theater, but there is nothing so far that even suggests this is the case. After all, there is nothing so far that suggests that Comey was anything other than grossly incompetent and tone-deaf. \nWhat it does suggest, however, is that Chaffetz is acting in a way that makes Dan Burton and Darrell Issa look like models of responsibility and bipartisanship. He didn't even have the decency to notify ranking member Elijah Cummings about something this explosive. If that doesn't trample on basic standards of fairness, I don't know what does. \nGranted, it's not likely that Chaffetz will have to answer for this. He sits in a ridiculously Republican district anchored in Provo and Orem; it has a Cook Partisan Voting Index of R+25, and gave Mitt Romney a punishing 78 percent of the vote in 2012. Moreover, the Republican House leadership has given its full support to Chaffetz' planned fishing expedition. But that doesn't mean we can't turn the hot lights on him. After all, he is a textbook example of what the House has become under Republican control. And he is also the Second Worst Person in the World. About Darrell Lucus \nDarrell is a 30-something graduate of the University of North Carolina who considers himself a journalist of the old school. An attempt to turn him into a member of the religious right in college only succeeded in turning him into the religious right\'s worst nightmare--a charismatic Christian who is an unapologetic liberal. His desire to stand up for those who have been scared into silence only increased when he survived an abusive three-year marriage. You may know him on Daily Kos as Christian Dem in NC . Follow him on Twitter @DarrellLucus or connect with him on Facebook . Click here to buy Darrell a Mello Yello. Connect'
```

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20800 entries, 0 to 20799
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   id      20800 non-null  int64 
 1   title   20242 non-null  object
 2   author  18843 non-null  object
 3   text    20761 non-null  object
 4   label   20800 non-null  int64 
dtypes: int64(2), object(3)
memory usage: 812.6+ KB
```

Data Proprocessing

In [11]:
```
# drop unnecessary columns
df = df.drop(columns=['id', 'title', 'author'], axis=1)
```

In [12]:
```
# drop null values
df = df.dropna(axis=0)
```

In [13]:
```
len(df)
```

Out[13]:
```
20761
```

In [ ]:
```
# remove special characters and punctuations
```

In [14]:
```
df['clean_news'] = df['text'].str.lower()
df['clean_news']
```

Out[14]:
```
0        house dem aide: we didn' t even see comey' s let...
1        ever get the feeling your life circles the rou...
2        why the truth might get you fired october 29, ...
3        videos 15 civilians killed in single us airstr...
4        print \nan iranian woman has been sentenced to...
                               ...
20795    rapper t. i. unloaded on black celebrities who...
20796    when the green bay packers lost to the washing...
20797    the macy' s of today grew from the union of sev...
20798    nato, russia to hold parallel exercises in bal...
20799      david swanson is an author, activist, journa...
Name: clean_news, Length: 20761, dtype: object
```

In [19]:
```
df['clean_news'] = df['clean_news'].str.replace('[^A-Za-z0-9\s]', '')
df['clean_news'] = df['clean_news'].str.replace('\n', '')
df['clean_news'] = df['clean_news'].str.replace('\s+', ' ')
df['clean_news']
```

Out[19]:
```
0        house dem aide we didnt even see comeys letter...
1        ever get the feeling your life circles the rou...
2        why the truth might get you fired october 29 2...
3        videos 15 civilians killed in single us airstr...
4        print an iranian woman has been sentenced to s...
                               ...
20795    rapper t i unloaded on black celebrities who m...
20796    when the green bay packers lost to the washing...
20797    the macys of today grew from the union of seve...
20798    nato russia to hold parallel exercises in balk...
20799     david swanson is an author activist journalis...
Name: clean_news, Length: 20761, dtype: object
```

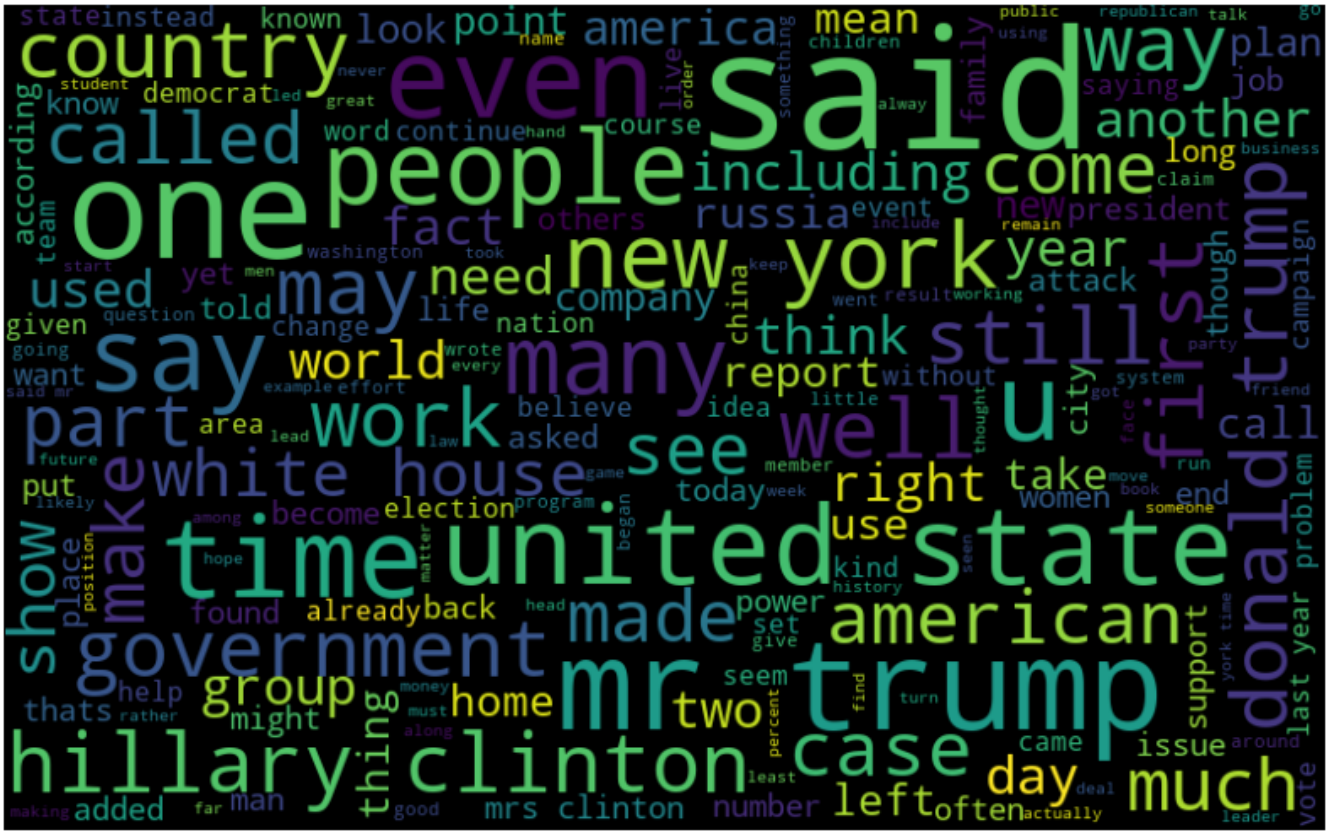
In [20]:
```
# remove stopwords
from nltk.corpus import stopwords
stop = stopwords.words('english')
df['clean_news'] = df['clean_news'].apply(lambda x: " ".join([word for word in x.split() if w
df.head()
```

Out[20]:

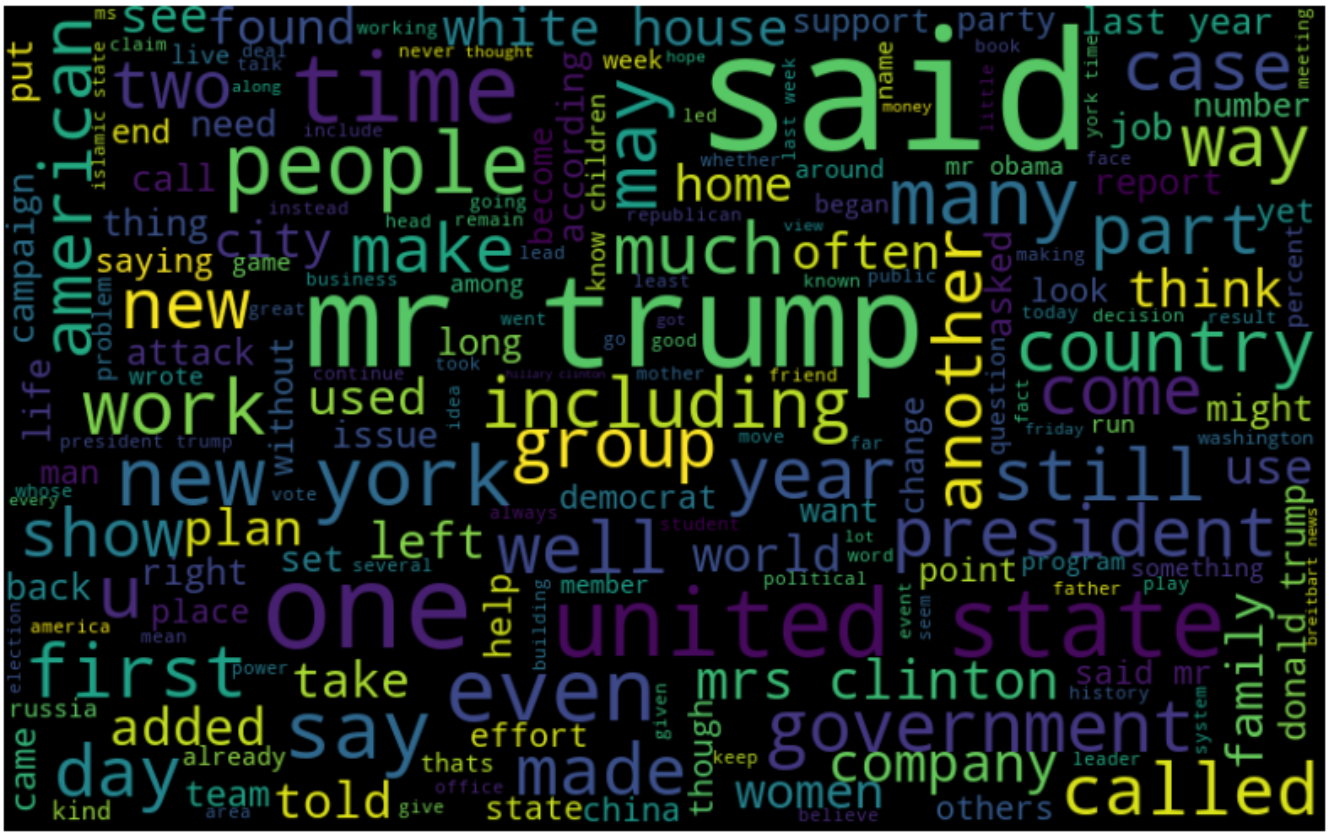
	text	label	clean_news
0	House Dem Aide: We Didn't Even See Comey's Let...	1	house dem aide didnt even see comeys letter ja...
1	Ever get the feeling your life circles the rou...	0	ever get feeling life circles roundabout rathe...
2	Why the Truth Might Get You Fired October 29, ...	1	truth might get fired october 29 2016 tension ...
3	Videos 15 Civilians Killed In Single US Airstr...	1	videos 15 civilians killed single us airstrike...
4	Print \nAn Iranian woman has been sentenced to...	1	print iranian woman sentenced six years prison...

Exploratory Data Analysis

In [22]:
```
# visualize the frequent words
all_words = " ".join([sentence for sentence in df['clean_news']])

wordcloud = WordCloud(width=800, height=500, random_state=42, max_font_size=100).generate(all_

# plot the graph
plt.figure(figsize=(15, 9))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

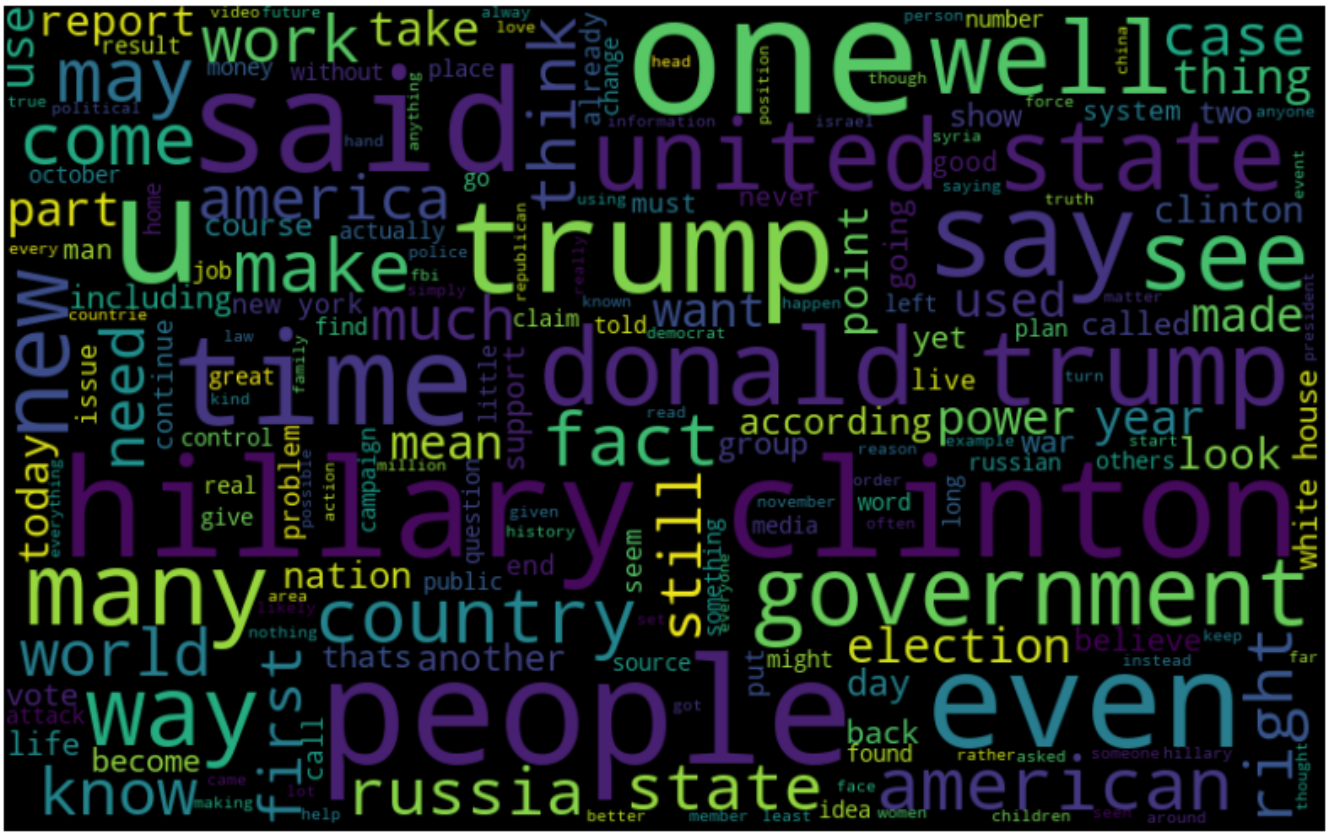


In [23]:
```
# visualize the frequent words for genuine news
all_words = " ".join([sentence for sentence in df['clean_news'][df['label']==0]])

wordcloud = WordCloud(width=800, height=500, random_state=42, max_font_size=100).generate(all_

# plot the graph
plt.figure(figsize=(15, 9))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



In [24]:

```
# visualize the frequent words for fake news
all_words = " ".join([sentence for sentence in df['clean_news'][df['label']==1]])

wordcloud = WordCloud(width=800, height=500, random_state=42, max_font_size=100).generate(all_

# plot the graph
plt.figure(figsize=(15, 9))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



Create Word Embeddings

In [26]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

In [27]:
```
# tokenize text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['clean_news'])
word_index = tokenizer.word_index
vocab_size = len(word_index)
vocab_size
```

Out[27]:
```
199536
```

In [48]:
```
# padding data
sequences = tokenizer.texts_to_sequences(df['clean_news'])
padded_seq = pad_sequences(sequences, maxlen=500, padding='post', truncating='post')
```

In [40]:
```
# create embedding index
embedding_index = {}
with open('glove.6B.100d.txt', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embedding_index[word] = coefs
```

In [42]:
```
# create embedding matrix
embedding_matrix = np.zeros((vocab_size+1, 100))
for word, i in word_index.items():
    embedding_vector = embedding_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

In [45]:
```
embedding_matrix[1]
```

Out[45]:
```
array([-0.13128   , -0.45199999,  0.043399  , -0.99798   , -0.21053   ,
       -0.95867997, -0.24608999,  0.48413   ,  0.18178   ,  0.47499999,
       -0.22305   ,  0.30063999,  0.43496001, -0.36050001,  0.20245001,
       -0.52594   , -0.34707999,  0.0075873 , -1.04970002,  0.18673   ,
        0.57369   ,  0.43814   ,  0.098659  ,  0.38769999, -0.22579999,
        0.41911   ,  0.043602  , -0.73519999, -0.53583002,  0.19276001,
       -0.21961001,  0.42515001, -0.19081999,  0.47187001,  0.18826   ,
        0.13357   ,  0.41839001,  1.31379998,  0.35677999, -0.32172   ,
       -1.22570002, -0.26635   ,  0.36715999, -0.27586001, -0.53245997,
        0.16786   , -0.11253   , -0.99958998, -0.60706002, -0.89270997,
        0.65156001, -0.88783997,  0.049233  ,  0.67110997, -0.27553001,
       -2.40050006, -0.36989   ,  0.29135999,  1.34979999,  1.73529994,
        0.27000001,  0.021299  ,  0.14421999,  0.023784  ,  0.33643001,
       -0.35475999,  1.09210002,  1.48450005,  0.49430001,  0.15688001,
        0.34678999, -0.57221001,  0.12093   , -1.26160002,  1.05410004,
        0.064335  , -0.002732  ,  0.19038001, -1.76429999,  0.055068  ,
        1.47370005, -0.41782001, -0.57341999, -0.12129   , -1.31690001,
       -0.73882997,  0.17682   , -0.019991  , -0.49175999, -0.55247003,
        1.06229997, -0.62879002,  0.29098001,  0.13237999, -0.70414001,
        0.67128003, -0.085462  , -0.30526   , -0.045495  ,  0.56509   ])
```

Input Split

In [49]:
```
padded_seq[1]
```

Out[49]:
```
array([   258,     28,   1557,     92,   4913,  27340,    415,   2246,
         2067,    377,    532,   1558,   5339,     29,     12,    796,
          179,    361,   1917,  17459,    829,  20147,   2990,   2626,
          640,    747,    252,   2025,   3113,  10995,    125,     39,
         2086,  78618,   3022,   3646,   3561,   3113,    835,    153,
         3458,     29,   9775,  51963,   3724,     18,    218,     20,
         3234,  20147,  10024,    625,     11,    481,   2494,   2417,
         8173,    442,    701,    613,    147,     14,  22280,    902,
          324,      8,    164,   3712,     60,  11541,    867,   2644,
           16,    864,   4422,    176,   5305,   2086,   4253,     40,
          257,    835,    192,     10,   2403,     10,   2086,   9775,
           58,   8372,  11246, 104297,  20952,   3713,  20953,  78619,
       104298,   5459,  31169,  25044,   7998,  19120,  65806,   4403,
          168,    261,  25045,   4403,    162,    355,    904,   1581,
          424,   1302,     20,    344,     37,   1963,    187,    394,
           59,   8107,   3658,  18529,    177,   1356,    745,   7401,
         2379,   7787,   1602,   2532,    152,     12,    458,  10153,
        11900,  17701,   8681,    128,    102,  22769,  10582,  10025,
        13518,   9418,    316,      7,    136,    626,    480,    370,
           95,  47538,   2439,  19434,   1139,   9775,   7163,   3591,
         8173,      4,    840,    169,    625,  14079,    414,     51,
          465,    177,      1,    446,   1139,    446,   1078,   1139,
           10,     39,    369,    182,    446,   1139,   8031,     51,
           51,   1557,  30058,   1703,    516,     16,   2633,  19772,
         1139,   8031,    957,  11901,    165,     60,    493,    957,
           16,    588,      6,  19772,  13107,  35329,   1635,   1688,
         3751,   2121,    254,     12,    104,  19772,   1099,    287,
         8032,  12768,   1159,  19121,     52,  14721,   8208,     22,
            6,      3,  20548,   3724,     69,   3241,     69,    292,
          893,   2020,  17201,     37,   1615,    250,    448,   2825,
        14721,     12,    562, 104299,    471,   7358,   1910,   2322,
         1438,   1502,   1212,    592,    448,    674,   1452,     22,
            6,   2420,   1387,    592,    197,  12000,    142,    192,
           42,     49,      6,    102,  14885,   1502,    230,    292,
          973,   1019,    137,    209,    627,    994,  17202,      8,
           15,      6,   4785,   3640,     29,     12,   9944,    907,
           86,   2648,   1521,    229,    176,  13108,   1376,  20147,
          481,     95,     11,    164,   2557,     12,   9203,     70,
          146,    604,   1732,   2688,    263,  25735,  41482,   4166,
           21,  20147,  13639,   4977,    118,     39,     43,   8681,
           86,    320,   2478,    447,   1049,    335,   1304,   1273,
          447,   1049,    247,    891,   1871,    335,    179,    361,
         1917,   4311,    361,     44,     41,   7472,    489,   1464,
           16,    335,   1453,    683,    737,   1032,    169,    934,
           30,   3341,    557,     11,    361,   5797,   7952,  20954,
         6089,    148,  51964,   7203,   1387,    637,    418,   1615,
           37,     53,      8,   1809,  47539,  11442,   3561,     53,
        19773,    981,  12649,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0])
```

In [50]:
```python
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(padded_seq, df['label'], test_size=0.20, ra
```

Model Training

In [63]:
```
from keras.layers import LSTM, Dropout, Dense, Embedding
from keras import Sequential

# model = Sequential([
#     Embedding(vocab_size+1, 100, weights=[embedding_matrix], trainable=False),
#     Dropout(0.2),
#     LSTM(128, return_sequences=True),
#     LSTM(128),
#     Dropout(0.2),
#     Dense(512),
#     Dropout(0.2),
#     Dense(256),
#     Dense(1, activation='sigmoid')
# ])

model = Sequential([
    Embedding(vocab_size+1, 100, weights=[embedding_matrix], trainable=False),
    Dropout(0.2),
    LSTM(128),
    Dropout(0.2),
    Dense(256),
    Dense(1, activation='sigmoid')
])
```

In [64]:
```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics='accuracy')
model.summary()
```

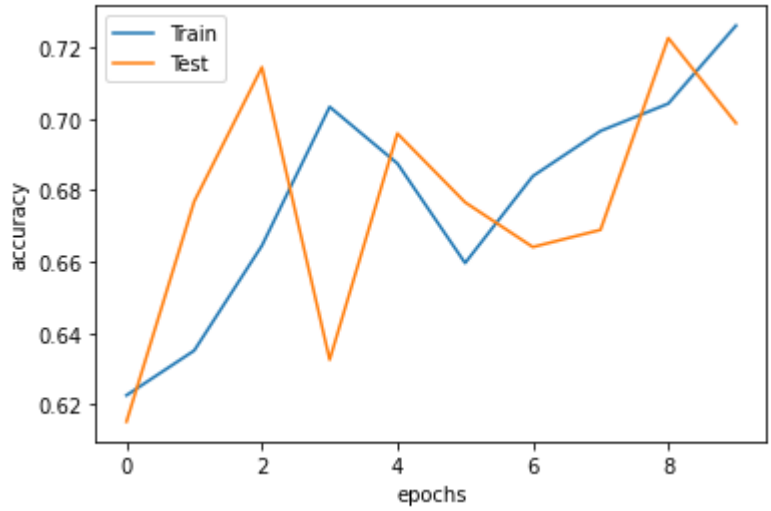
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 100)         19953700
_________________________________________________________________
dropout_5 (Dropout)          (None, None, 100)         0
_________________________________________________________________
lstm_3 (LSTM)                (None, 128)               117248
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 20,071,077
Trainable params: 117,377
Non-trainable params: 19,953,700
_________________________________________________________________
```

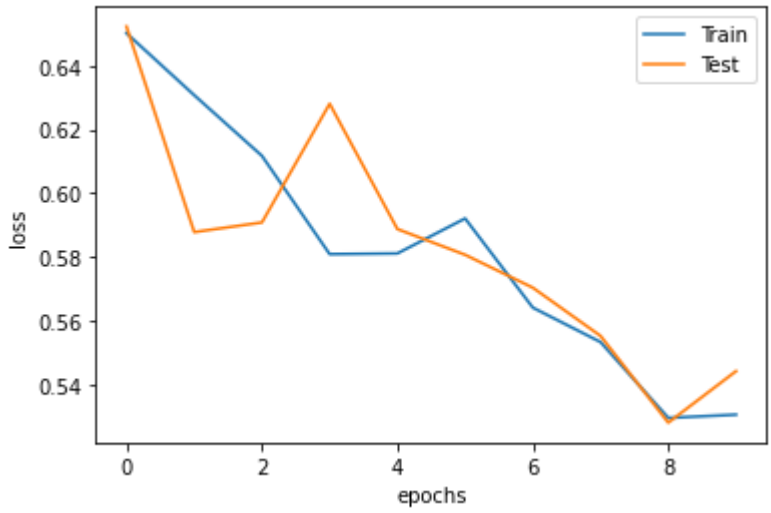
In [61]:
```
# train the model
history = model.fit(x_train, y_train, epochs=10, batch_size=256, validation_data=(x_test, y_te
```

```
Epoch 1/10
65/65 [==============================] - 42s 617ms/step - loss: 0.6541 - accuracy: 0.6098 - val
_loss: 0.6522 - val_accuracy: 0.6152
Epoch 2/10
65/65 [==============================] - 39s 607ms/step - loss: 0.6436 - accuracy: 0.6241 - val
_loss: 0.5878 - val_accuracy: 0.6769
Epoch 3/10
65/65 [==============================] - 40s 611ms/step - loss: 0.6057 - accuracy: 0.6688 - val
_loss: 0.5908 - val_accuracy: 0.7144
Epoch 4/10
65/65 [==============================] - 40s 613ms/step - loss: 0.5693 - accuracy: 0.7239 - val
_loss: 0.6280 - val_accuracy: 0.6326
Epoch 5/10
65/65 [==============================] - 40s 612ms/step - loss: 0.5990 - accuracy: 0.6699 - val
_loss: 0.5887 - val_accuracy: 0.6959
Epoch 6/10
65/65 [==============================] - 40s 614ms/step - loss: 0.6060 - accuracy: 0.6593 - val
_loss: 0.5807 - val_accuracy: 0.6766
Epoch 7/10
65/65 [==============================] - 40s 609ms/step - loss: 0.5546 - accuracy: 0.6906 - val
_loss: 0.5704 - val_accuracy: 0.6641
Epoch 8/10
65/65 [==============================] - 39s 606ms/step - loss: 0.5517 - accuracy: 0.6973 - val
_loss: 0.5553 - val_accuracy: 0.6689
Epoch 9/10
65/65 [==============================] - 33s 508ms/step - loss: 0.5400 - accuracy: 0.6855 - val
_loss: 0.5281 - val_accuracy: 0.7226
Epoch 10/10
65/65 [==============================] - 40s 609ms/step - loss: 0.5244 - accuracy: 0.7236 - val
_loss: 0.5442 - val_accuracy: 0.6988
```

In [62]:

```
# visualize the results
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend(['Train', 'Test'])
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend(['Train', 'Test'])
plt.show()
```





In [ ]: