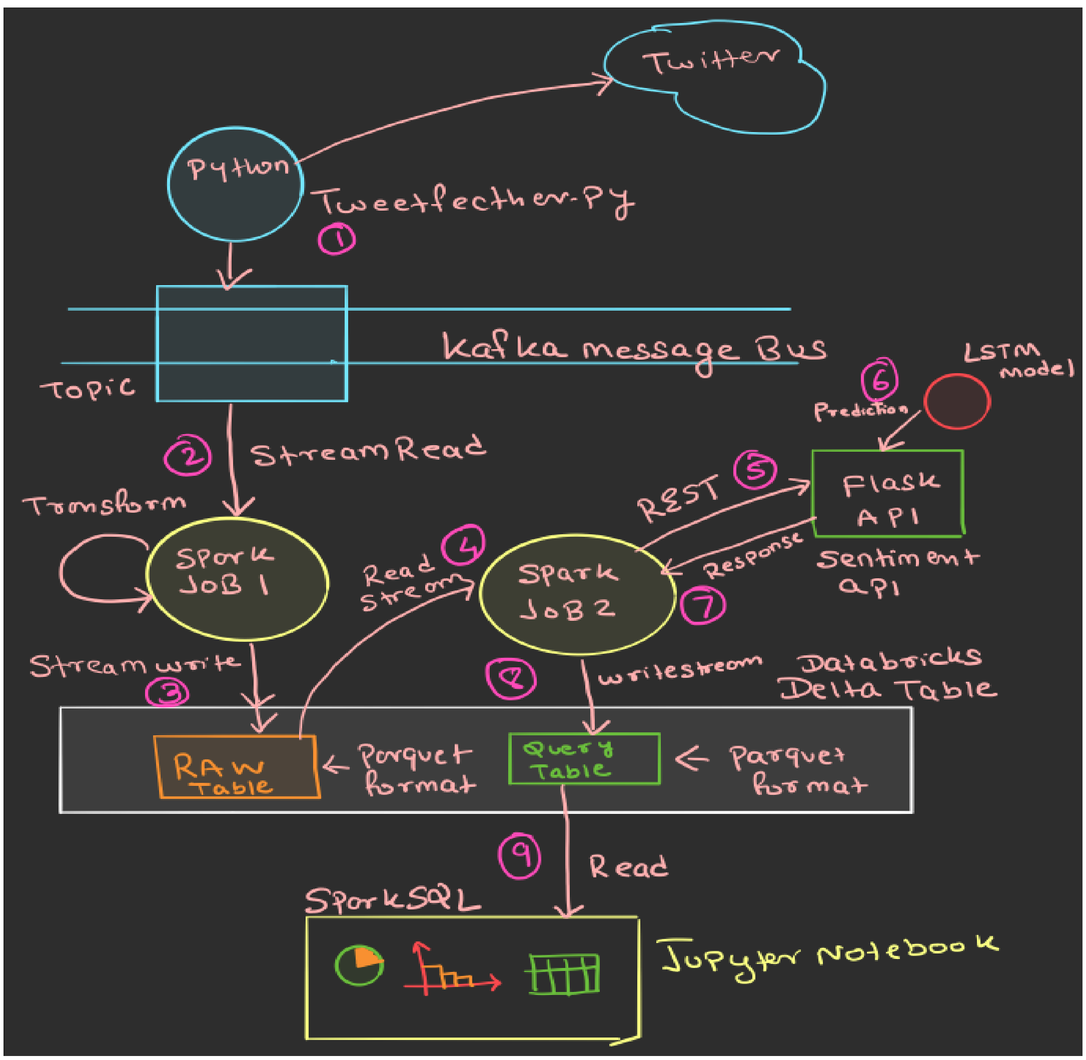
Emotional States Transfers

Introduction:

This project is inspired from the assumption made in the article published in 2014 by Adam D. I. Kramer, Jamie E. Guillory, and Jeffrey T. Hancoc; i.e. **Emotional states can be transferred to others via emotional contagion, leading them to experience the same emotions as those around them** (ref1). This implementation of the idea uses Twitter data to analyze the general emotion on user's timeline and compute the contagion effect of the timeline on the user's future tweet. A tweeter timeline of a user consists of the tweets, retweets from the other users/groups which A follows. Each tweet has some emotion or sentiment associated with it and if the average emotion on user's timeline is negative, it is quite likely that in the near future user may tweet with negative emotion, showing a contagion effect.

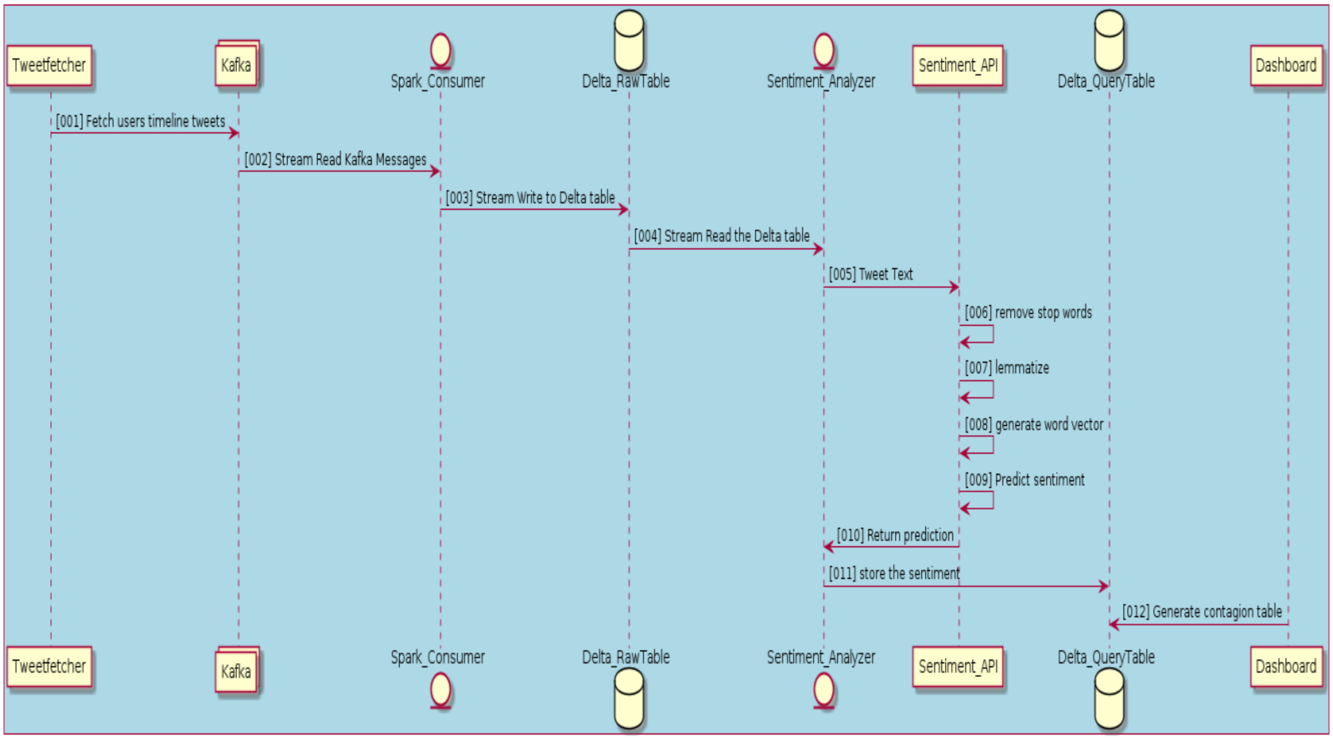
System Design and Components



1. TweetFetcher: A python program executed by a cron job, performs following operations
    1. For a user of interest, find his/hers N friends/group.
    2. For each friend fetch M last tweets.
    3. Fetch tweets sent by the user of interest.
    4. push these tweets in JSON format to Kafka message bus.
2. Kafka Message Bus:
    1. A high throughput distributed streaming platform.
3. Spark Streaming Job 1:
    1. Consumes KAFKA messages, perform some basic transformations and stores the data in Parquet format on disk.
4. Spark Streaming Job 2:
    1. For each record in parquet stream, analyze the emotion of the tweet by making a REST call to a Sentiment analysis service.
    2. The result is stored in parquet table in simplified format.
5. Sentiment Analysis Service:
    1. Is a FLASK based REST api which takes in text of a tweet and returns a sentiment of the tweet using two models
        1. Custom build LSTM based deep learning model.
        2. Off the shelf sentiment analyzer using python nltk library.
6. Dashboard:
    1. A python notebook, uses SparkSQL to transform data and displays historical data of user's timeline and contagion effect.
7. Custom Model:
    1. An LSTM based deep learning model trained on a labeled dataset and detect negative, positive and neutral sentiment.

Data Flow

Data Flow



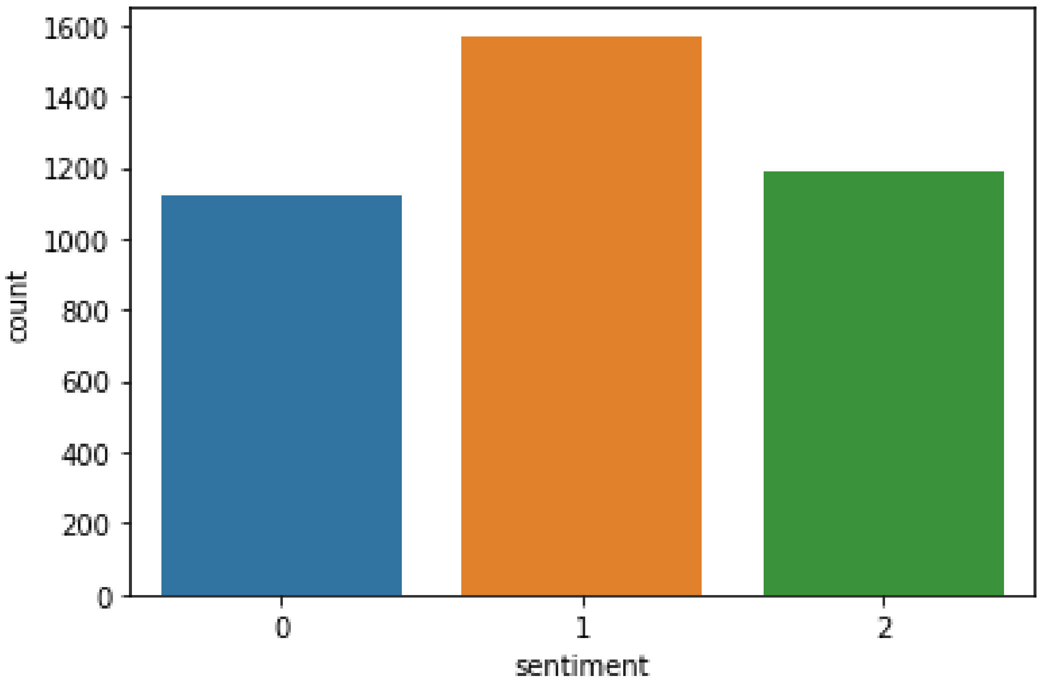
Sentiment Model

Sentiment analyzer can be implemented by multiple ways. Natural language processing is all about creating systems that process or "understand" language in order to perform certain tasks such as sentiment analysis, speech recognition, machine translations etc.

Training Data:

Distribution

The project uses dataset from https://www.kaggle.com/kazanova/sentiment140 which has 1.6 million tweets classified as neutral(1), positive(2) and negative(0).



Word Cloud

Following word cloud is drawn on a sample of tweets. The word and it's co-existence with other words or contexts set the sentiment of the tweet to positive, negative or neutral.

Word Cloud of all tweets

Sample Data

```
In [8]: df.sample(10)
```

Out[8]:

	Tweet	sentiment
2945	Yogi vows to work for all.	1
3409	savours civic win and Congress loss.	1
3736	dog and drinks -- stood out in PM reply in Parliament.	1
395	Happy birthday	2
3474	Dr Kunal Sarkar on how cartels have been ripping off patients in	1
1795	Wth why did 4/20 have to happen during the Easter holidays it's one of the best days in Leeds un...	0
2560	as she opts for over	1
1713	follow mee plss unhappy	0
2256	shouldn't keep Kenyans complaining on	0
3688	State Bank of India Q3 profit more than doubles to Rs 2	1

Data Transformation:

Following steps are performed for data transformations

1. Remove Punctuations
2. Remove Numbers
3. Tokenize
4. Remove stop words.
5. Perform Stemming.
6. Perform Lemmatizing.
7. Transform to word vector.

After cleaning the tweet with above set

Transformation : Step 1-6

In [11]:
```python
def clean_data(data):
    """
     Removes punctuation and return lower case string.
    """
    global ps
    global wn
    global np
    global stopword

    if not isinstance(data, str):
        return ""
    no_punctuation = str(nlp(data).text)
    no_numbers = re.sub('[0-9]+', '', no_punctuation)
    tokenize = re.split('\W+', no_numbers)
    no_stopwords = [str.lower(word) for word in tokenize if word not in stopword]
    stemming = [ps.stem(word) for word in no_stopwords]
    lemmatize = [wn.lemmatize(str(word)) for word in stemming]
    return " ".join(lemmatize)
```

In [12]:
```python
df['clean_tweet'] = df['Tweet'].apply(lambda x: clean_data(x))
df.head()
```

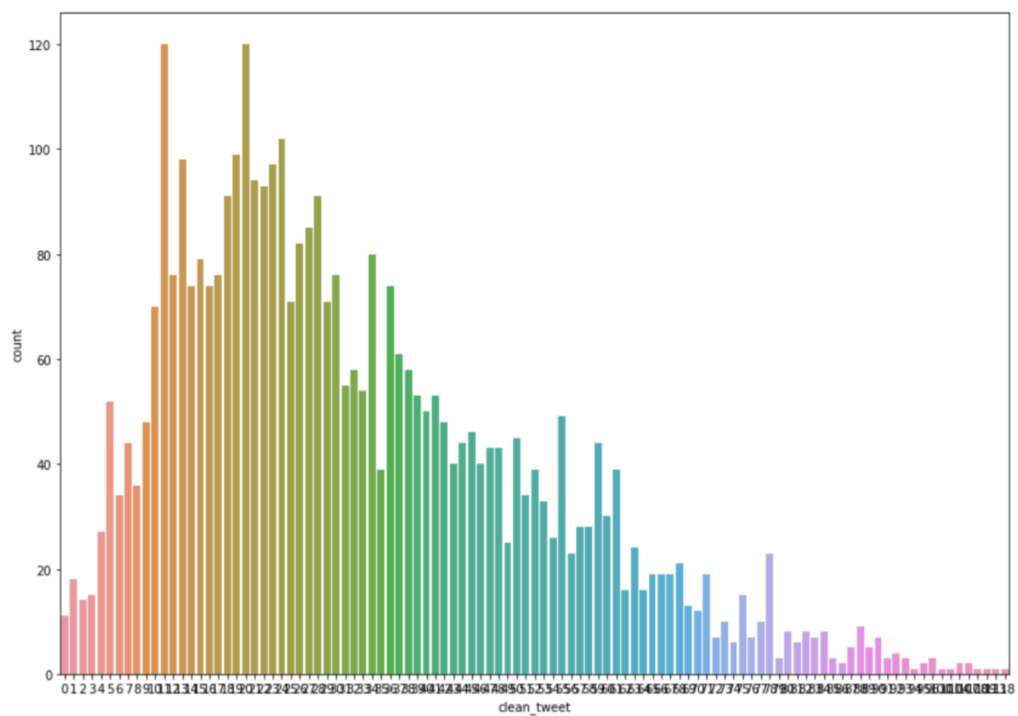
Out[12]:

	Tweet	sentiment	clean_tweet
0	An inspiration in all aspects: Fashion	2	an inspir aspect fashion
1	fitness	2	fit
2	beauty and personality. :)KISSES TheFashionIcon	2	beauti person kiss thefashionicon
3	Apka Apna Awam Ka Channel Frankline Tv Aam Admi Production Please Visit Or Likes Share :)Fb Pag...	2	apka apna awam ka channel franklin tv aam admi product plea visit or like share fb page
4	Beautiful album from the greatest unsung guitar genius of our time - and I've met the great bac...	2	beauti album greatest unsung guitar geniu time i met great backstag

Histogram of length of tweets:

In [35]:
```python
plt.figure(figsize=(14,10))
sns.countplot(df['clean_tweet'].apply(lambda x : len(x)))
```

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x1a61737e90>



Integerized representation

In [36]:

```
def vectorize(tweet):
    global max_tweet_length
    global wordsList

    tweetid =[0] * max_tweet_length
    indexCounter = 0
    for word in tweet.split(" "):
        if indexCounter < max_tweet_length:
            try:
                tweetid[indexCounter] = wordsList.index(word)
            except ValueError:
                tweetid[indexCounter] = 399999 #Vector for unknown words
        indexCounter = indexCounter + 1

    return tweetid
```

In [38]:

```
df['tweet_ids']=df['clean_tweet'].apply(lambda x: vectorize(x))
```

In [40]:

```
df.sample(5)
```

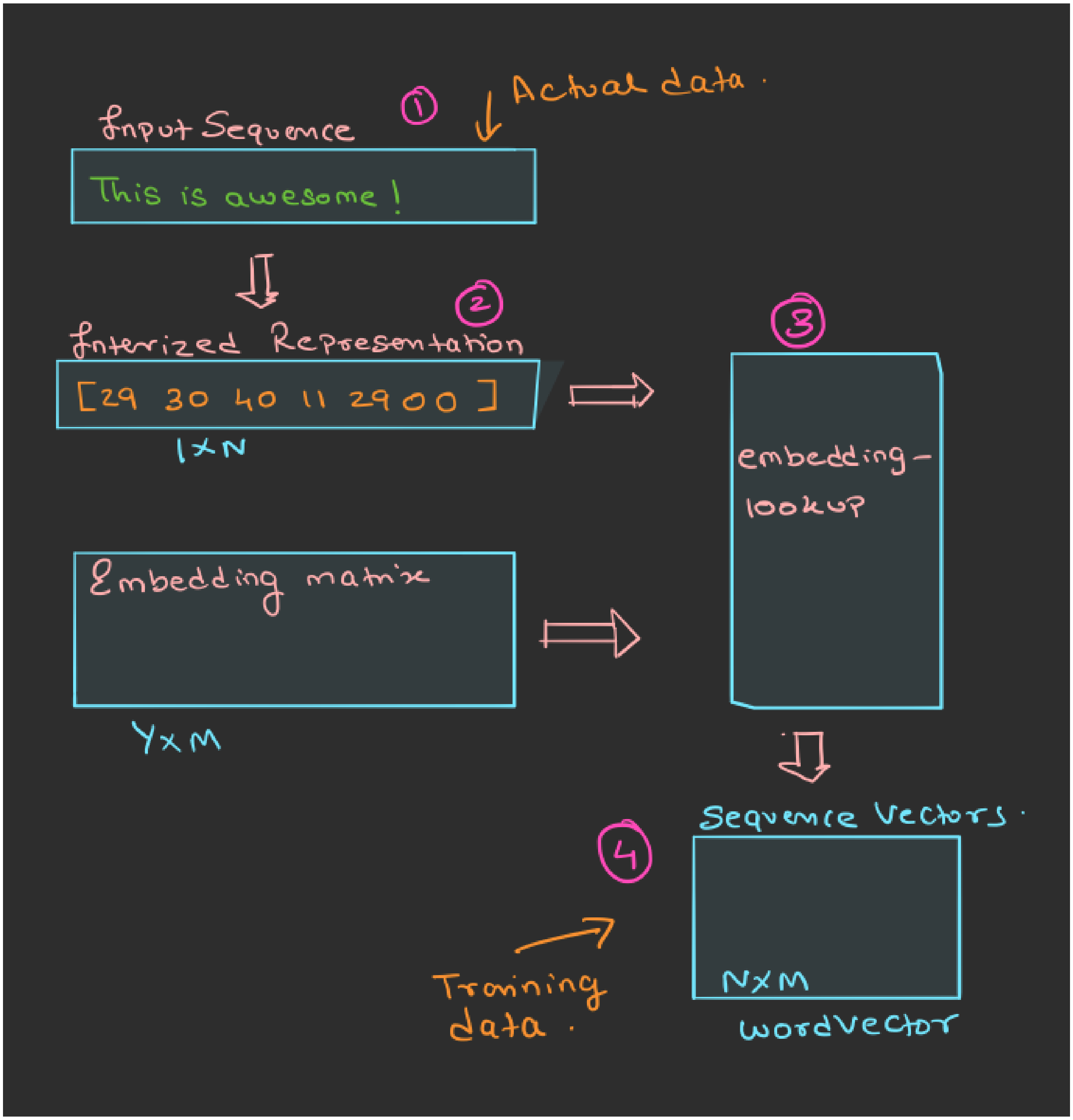
Out[40]:

	Tweet	sentiment	clean_tweet	tweet_ids
2826	made to trek 2km to fetch water for	1	made trek km fetch water	[399999, 116, 9779, 1608, 18184, 430, 399999, 0, 0, 0]
997	Hellooo happy Jackatkinson (jackat13)	2	hellooo happi jackatkinson jackat	[399999, 366139, 399999, 399999, 399999, 0, 0, 0, 0, 0]
2987	How Rahul met HC gives to CBI	1	how rahul met hc give cbi	[197, 20675, 809, 27811, 455, 25782, 0, 0, 0, 0]
1272	crying muh feels	0	cri muh feel	[62558, 13626, 998, 0, 0, 0, 0, 0, 0, 0]
2745	Militant wing welcomes but asks him not to speak against	1	milit wing welcom ask speak	[399999, 1755, 389376, 1712, 2199, 399999, 0, 0, 0, 0]

Word Embedding:

Deep learning algorithms such as RNN, CNN etc. need arrays of scalar values for training. For NLP related task, the text needs to be converted to some numeric format. One way to do it would be use one hot encoding, which would create a huge sparse matrix with the size of the vocabulary of the language. The problem with this approach apart for huge sparse matrix, is the context of words (order) is lost in the transformation.

Word embedding is one of the most popular representation of document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc. Word2Vec is one of the most popular technique to learn word embeddings using shallow neural network. This project pre- trained model GloVe (Global Vector for word representations).GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.The model takes in a large dataset of sentences (English Wikipedia for example) and outputs vectors for each unique word in the corpus. The output of a GloVe model is called an embedding matrix.This embedding matrix will contain vectors for every distinct word in the training corpus. Traditionally, embedding matrices can contain over 3 million word vectors.



Embedding matrix is used as an input to train an a neural network. Each word in a sentence depends greatly on what came before and comes after it. In order to account for this dependency, this project uses a recurrent neural network. The main difference between feedforward neural networks and recurrent ones is the temporal aspect of the latter. In RNNs, each word in an input sequence will be associated with a specific time step. In effect, the number of time steps will be equal to the max sequence length.One of the drawback of RNN is Vanishing Gradient. RNN remembers things for just small durations of time, i.e. if we need the information after a small time it may be reproducible, but once a lot of words are fed in, this information gets lost somewhere. This issue can be resolved by applying a slightly tweaked version of RNNs – the Long Short-Term Memory Networks.Long Short Term Memory Units are modules that you can place inside of recurrent neural network. At a high level, they make sure that the hidden state vector h is able to encapsulate information about long term dependencies in the text. Following is LSTM model built using TensorFlow 2.0

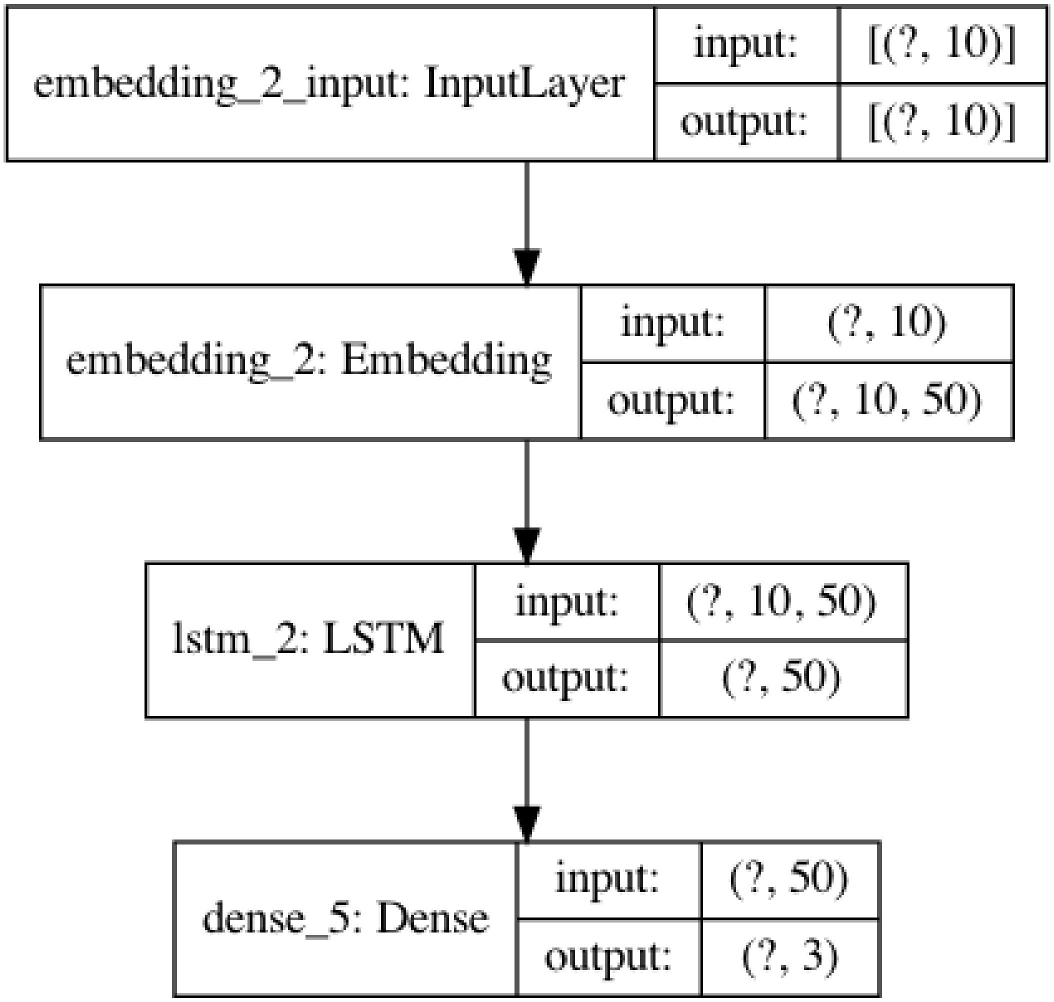
TensorFlow 2.0 provides an export API to export trained model to JSON format. Same mode is used by the flask based REST api to predict the sentiment of the text blob.

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Embedding(vocab_size,
embedding_dim,weights=[wordVectors], \
                                    input_length=max_tweet_length,
trainable=False))
model.add(tf.keras.layers.LSTM(embedding_dim))
tf.keras.layers.Dense(embedding_dim, activation='relu'),
model.add(tf.keras.layers.Dense(numClasses, activation='softmax'))

# compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# summarize the model
print(model.summary())


Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 10, 50)            20000000
_________________________________________________________________
lstm_2 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 153
=================================================================
Total params: 20,020,353
Trainable params: 20,353
Non-trainable params: 20,000,000
_________________________________________________________________
None
```



Accuracy

```
training_hist = model.fit(train_x, train_y, epochs=num_epochs,
validation_split=0.2, verbose=True)
loss, accuracy = model.evaluate(train_x, train_y, verbose=0)
```

In [65]:

```
training_hist_df = pd.DataFrame(training_hist.history)
training_hist_df['epoch'] = training_hist.epoch
training_hist_df.tail()
```

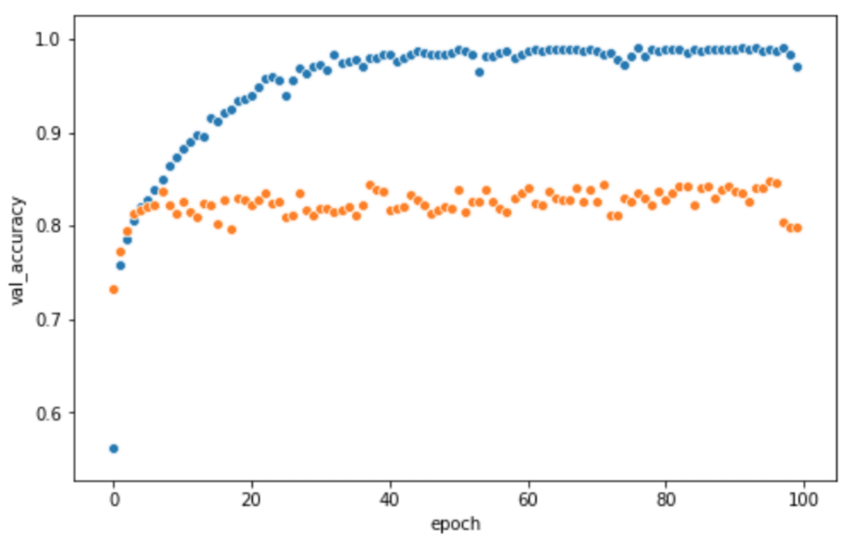
Out[65]:

	loss	accuracy	val_loss	val_accuracy	epoch
95	0.019975	0.988469	1.077181	0.847145	95
96	0.019150	0.987546	1.116785	0.845304	96
97	0.019848	0.989852	1.247327	0.804788	97
98	0.039161	0.983395	1.241796	0.799263	98
99	0.070167	0.970480	1.002536	0.799263	99

In [70]:

```
plt.figure(figsize=(8,5))
sns.scatterplot(training_hist_df.epoch, training_hist_df.accuracy)
sns.scatterplot(training_hist_df.epoch, training_hist_df.val_accuracy)
```

Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x1a63b073d0>



Dashboard Notebook

Jupyter notebook to transform and retrieve contagion table. Following are the steps involved

1. **Load Parquet table schema**

Load Delta Table.

```
spark.sparkContext.setLogLevel("ERROR")
tweets = spark.read.format("delta").load("deltatables/processed_new")
tweets.printSchema()
tweets.createOrReplaceTempView("tweettable")
tweets = spark.sql("select *, to_date(date) as DayofYear from tweettable")
tweets.createOrReplaceTempView("tweettable")
tweets.printSchema()
```

```
root
 |-- tweetid: long (nullable = true)
 |-- friendname: string (nullable = true)
 |-- profilename: string (nullable = true)
 |-- text: string (nullable = true)
 |-- date: string (nullable = true)
 |-- Sentiment: struct (nullable = true)
 |    |-- sentiment: integer (nullable = true)
 |    |-- psentiment: integer (nullable = true)
 |    |-- ngsentiment: integer (nullable = true)
 |    |-- nsentiment: integer (nullable = true)
 |    |-- nltk_sentiment: integer (nullable = true)
 |    |-- nltk_psentiment: integer (nullable = true)
 |    |-- nltk_ngsentiment: integer (nullable = true)
 |    |-- nltk_nsentiment: integer (nullable = true)

root
```

The flask rest api returns two predictions, first predicted by custom model build above and second by off the shelf python nltk module.

2. **Flatten the table and Create new in-memory parquet table**

This table contains tweets sent by friends and filters tweets sent by users.

Create in memory view for tweets sent by 'Friends'.

In [14]:

```
query = "Select Date,\
         ProfileName,Friendname,count(TweetId) as Total,\
         Sum(Nltk_Positive) as Nltk_Positive,\
         Sum(Nltk_Negative) as Nltk_Negative,\
         Sum(Nltk_Neutral) as Nltk_Neutral,\
         Sum(Positive) as Positive,\
         Sum(Negative) as Negative,\
         Sum(Neutral) as Neutral,\
         Avg(Sentiment) as Sent_Avg,\
         Avg(Nltk_Sentiment) as Nltk_Avg\
         from tweettable \
         where Friendname != ProfileName \
         group by ProfileName, Date,Friendname\
         order by Date Desc,ProfileName"
left_df = spark.sql(query)
left_df.show()
left_df.createOrReplaceTempView("timeline")
```

Date	ProfileName	Friendname	Total	Nltk_Positive	Nltk_Negative	Nltk_Neutral	Positive	Negative	Neutral	Sent_Avg	Nltk_Avg
2020-07-10	AskAnshul	earth	2	0	0	2	0	0	2	1.0	1.0
2020-07-10	AskAnshul	AartiTikoo	2	0	0	2	0	2	0	0.0	1.0
2020-07-10	AskAnshul	Captain_Mani72	4	0	0	4	4	0	0	2.0	1.0
2020-07-10	AskAnshul	telegram	4	0	0	4	4	0	0	2.0	1.0
2020-07-10	EnayetSpeaks	teamxecutor	4	0	0	4	0	2	2	0.5	1.0
2020-07-10	EnayetSpeaks	urspessi1	16	0	0	14	10	0	6	1.625	1.125
2020-07-10	EnayetSpeaks	khanumarfa	8	0	0	8	4	0	4	1.5	1.0
2020-07-10	EnayetSpeaks	Tamanna22	6	0	0	6	4	0	2	1.6666666666666667	1.0
2020-07-10	realDonaldTrump	TeamTrump	1	0	0	1	0	1	0	0.0	1.0
2020-07-10	realDonaldTrump	parscale	1	0	0	1	0	1	0	0.0	1.0
2020-07-10	realDonaldTrump	Mike_Pence	4	0	0	4	0	1	3	0.75	1.0
2020-07-10	realDonaldTrump	IngrahamAngle	1	0	0	1	0	1	0	0.0	1.0
2020-07-10	realDonaldTrump	TuckerCarlson	1	0	0	1	1	0	0	2.0	1.0
2020-07-10	realDonaldTrump	DiamondandSilk	5	0	0	5	2	1	2	1.2	1.0
2020-07-10	realDonaldTrump	GOPChairwoman	1	0	0	1	0	0	1	1.0	1.0

3. **Create a in memory view for tweets sent by 'Profile/User'**

Create in memory view for tweet sent by 'Profile'

In [15]:
```
query = "Select Date,\
         ProfileName,Friendname,count(TweetId) as Total,\
         Sum(Nltk_Positive) as Nltk_Positive,\
         Sum(Nltk_Negative) as Nltk_Negative,\
         Sum(Nltk_Neutral) as Nltk_Neutral,\
         Sum(Positive) as Positive,\
         Sum(Negative) as Negative,\
         Sum(Neutral) as Neutral,\
         Avg(Sentiment) as Sent_Avg,\
         Avg(Nltk_Sentiment) as Nltk_Avg\
         from tweettable \
         where Friendname = ProfileName \
         group by ProfileName, Date,Friendname\
         order by Date Desc,ProfileName"
right_df = spark.sql(query)
right_df.show()
right_df.createOrReplaceTempView("profiletweet")
```

Date	ProfileName	Friendname	Total	Nltk_Positive	Nltk_Negative	Nltk_Neutral	Positive	Negative	Neutral	Sent_Avg	Nltk_Avg
2020-07-09	AskAnshul	AskAnshul	10	0	0	10	2	4	4	0.8	1.0
2020-07-09	EnayetSpeaks	EnayetSpeaks	50	0	4	46	18	2	30	1.32	0.92
2020-07-09	realDonaldTrump	realDonaldTrump	20	0	0	20	7	4	9	1.15	1.0
2020-07-08	AskAnshul	AskAnshul	8	0	0	8	2	2	4	1.0	1.0
2020-07-08	realDonaldTrump	realDonaldTrump	5	0	0	5	1	1	3	1.0	1.0
2020-07-07	AskAnshul	AskAnshul	8	0	0	6	2	2	4	1.0	1.25
2020-07-06	AskAnshul	AskAnshul	8	0	0	6	2	2	4	1.0	1.25
2020-07-05	AskAnshul	AskAnshul	8	0	0	8	2	4	2	0.75	1.0
2020-07-04	AskAnshul	AskAnshul	8	0	0	8	6	0	2	1.75	1.0

4. **Aggregate tweets by date for each User.**

Aggreage a in-memory view for tweets sent by 'Profile' group by Profilename

In [16]:
```
query = "Select Date, ProfileName, sum(Total) as Total, \
          sum(Nltk_Positive) as Nltk_Positive , sum(Nltk_Negative) as Nltk_Negative, \
          sum(Nltk_Neutral) as Nltk_Neutral, sum(Positive) as Positive , \
          sum(Negative) as Negative, sum(Neutral) as Neutral, \
          sum(Sent_Avg) as Sent_Avg, sum(Nltk_Avg) as Nltk_Avg\
          from profiletweet group by ProfileName,Date"
agg_profile_df = spark.sql(query)
agg_profile_df.show()
agg_profile_df.createOrReplaceTempView("aggprofile")
```

Date	ProfileName	Total	Nltk_Positive	Nltk_Negative	Nltk_Neutral	Positive	Negative	Neutral	Sent_Avg	Nltk_Avg
2020-07-09	AskAnshul	10	0	0	10	2	4	4	0.8	1.0
2020-07-09	EnayetSpeaks	50	0	4	46	18	2	30	1.32	0.92
2020-07-09	realDonaldTrump	20	0	0	20	7	4	9	1.15	1.0
2020-07-08	AskAnshul	8	0	0	8	2	2	4	1.0	1.0
2020-07-08	realDonaldTrump	5	0	0	5	1	1	3	1.0	1.0
2020-07-07	AskAnshul	8	0	0	6	2	2	4	1.0	1.25
2020-07-06	AskAnshul	8	0	0	6	2	2	4	1.0	1.25
2020-07-05	AskAnshul	8	0	0	8	2	4	2	0.75	1.0
2020-07-04	AskAnshul	8	0	0	8	6	0	2	1.75	1.0

5. **Aggregate Tweets by friends of users and date.**

Aggregate in-memory view for tweets sent by 'Friends' group by Profilename

In [17]:

```
query = "Select Date, ProfileName, sum(Total) as Total, count(Friendname) as Friendname, \
        sum(Nltk_Positive) as Nltk_Positive, sum(Nltk_Negative) as Nltk_Negative, \
        sum(Nltk_Neutral) as Nltk_Neutral, sum(Positive) as Positive, \
        sum(Negative) as Negative, sum(Neutral) as Neutral, \
        Avg(Sent_Avg) as Sent_Avg, Avg(Nltk_Avg) as Nltk_Avg \
        from timeline group by ProfileName,Date"
agg_timeline_df = spark.sql(query)
agg_timeline_df.show()
agg_timeline_df.createOrReplaceTempView("aggtimeline")
```

Date	ProfileName	Total	Friendname	Nltk_Positive	Nltk_Negative	Nltk_Neutral	Positive	Negative	Neutral	Sent_Avg	Nltk_Avg
2020-07-10	AskAnshul	12	4	0	0	12	8	2	2	1.25	1.0
2020-07-10	EnayetSpeaks	34	4	0	0	32	18	2	14	1.3229166666666667	1.03125
2020-07-10	realDonaldTrump	27	11	0	0	24	6	8	13	0.8136363636363636	1.0515151515151517
2020-07-09	AskAnshul	854	40	0	6	794	272	184	398	1.1364458195911111	1.0456313466917013
2020-07-09	EnayetSpeaks	1638	47	0	22	1598	434	220	984	1.1380178939644578	1.0074403610573825
2020-07-09	realDonaldTrump	260	29	0	3	240	76	62	122	1.1441334265653682	1.0359477717281709
2020-07-08	AskAnshul	352	35	0	0	336	112	70	170	1.1358884766027626	1.027845804988662
2020-07-08	EnayetSpeaks	294	19	0	4	266	70	46	178	1.189617838302049	1.0678812415654522
2020-07-08	realDonaldTrump	116	24	0	2	106	29	33	54	0.9220786736411736	1.0742424242424242
2020-07-07	AskAnshul	220	27	0	0	202	66	40	114	1.167577895355673	1.1015873015873014
2020-07-07	EnayetSpeaks	120	13	0	0	114	46	4	70	1.3142857142857143	1.1076923076923078
2020-07-07	realDonaldTrump	70	25	0	1	62	27	20	23	1.088888888888889	1.059111111111111
2020-07-06	AskAnshul	186	20	0	0	166	58	40	88	1.1590025252525251	1.0808964646464645
2020-07-06	EnayetSpeaks	110	8	0	4	106	20	0	90	1.3104166666666668	0.9375
2020-07-06	realDonaldTrump	37	16	0	0	31	16	8	13	1.328125	1.2447916666666667
2020-07-05	AskAnshul	170	19	0	0	160	56	32	82	1.1491228070175439	1.05
2020-07-05	EnayetSpeaks	60	7	0	0	58	28	6	26	1.4219954648526076	1.0158730158730158
2020-07-05	realDonaldTrump	40	14	0	1	33	18	6	16	1.2321428571428572	1.1238095238095238
2020-07-04	AskAnshul	122	16	0	2	112	32	22	68	1.0130208333333333	1.0515625
2020-07-04	EnayetSpeaks	24	4	0	2	22	16	2	6	1.4583333333333333	0.9583333333333334

only showing top 20 rows

6. **Join Friends view and Users view on User sorted by Date.**

```
query = "Select t.*, p.Total as P_Total,\
         p.Nltk_Positive as P_Nltk_Positive,\
         p.Nltk_Negative as P_Nltk_Negative,\
         p.Nltk_Neutral as P_Nltk_Neutral,\
         p.Positive as P_Positive,\
         p.Negative as P_Negative,\
         p.Neutral as P_Neutral,\
         p.Sent_Avg as P_Sent_Avg,\
         p.Nltk_Avg as P_Nltk_Avg\
         from aggtimeline as t\
         inner join aggprofile as p\
         on t.Date = p.Date and t.ProfileName = p.ProfileName"

all_agg_df = spark.sql(query)
all_agg_df.show()
all_agg_df.createOrReplaceTempView("aggall")
```

7. **Check the relations between average sentiment of user's time line vs Tweets sent by the user on the given date.**

Check contingent Effect

[20]:

```
query = "Select Date, ProfileName, Total,P_Total,\
         Sent_Avg,P_Sent_Avg,\
         Nltk_Avg,P_Nltk_Avg\
         from aggall order by ProfileName"
spark.sql(query).show()
```

Date	ProfileName	Total	P_Total	Sent_Avg	P_Sent_Avg	Nltk_Avg	P_Nltk_Avg
2020-07-07	AskAnshul	220	8	1.167577895355673	1.0	1.1015873015873014	1.25
2020-07-09	AskAnshul	854	10	1.1364458195911111	0.8	1.0456313466917013	1.0
2020-07-08	AskAnshul	352	8	1.1358884766027626	1.0	1.027845804988662	1.0
2020-07-06	AskAnshul	186	8	1.1590025252525251	1.0	1.0808964646464645	1.25
2020-07-05	AskAnshul	170	8	1.1491228070175439	0.75	1.05	1.0
2020-07-04	AskAnshul	122	8	1.013020833333333	1.75	1.0515625	1.0
2020-07-09	EnayetSpeaks	1638	50	1.1380178939644578	1.32	1.0074403610573825	0.92
2020-07-08	realDonaldTrump	116	5	0.9220786736411736	1.0	1.0742424242424242	1.0
2020-07-09	realDonaldTrump	260	20	1.1441334265653682	1.15	1.0359477717281709	1.0

ProfileName: User of our interest.

Total : Total tweets on user's timeline on a particular date.

P_Total. : Total tweets sent by user on a particular date.

Sent_Avg. : Average Sentiment on tweets on user's timeline, as computed/predicted by custom LSTM model.

P_Sent_Avg: Average sentiment of tweets sent by user, as computed/predicted by custom LSTM model.

Nltk_Avg. : Average Sentiment on tweets on user's timeline, as computed/predicted by nltk.

P_Nltk_Avg :Average sentiment of tweets sent by user, as computed/predicted by nltk.

Conclusion:

As we see from the inference from the small data set used in the experiment, emotions can spread throughout a network i.e. difference between Sent_Avg, P_Sent_Avg)**.** We can see that there exist some correlation between User's timeline and tweets sent by the user. Online messages influence our experience of emotions, which may affect a variety of offline behaviors.

Reference:

https://www.pnas.org/content/111/24/8788 https://nlp.stanford.edu/projects/glove/

https://www.kaggle.com/kazanova/sentiment140

https://www.oreilly.com/content/perform-sentiment-analysis-with-lstms-using-tensorflow/