

FINAL REPORT

RECURRENT NEURAL NETWORK ANALYSIS

Submitted by

SAYAN DEY
M.E Software Systems



Birla Institute of Technology and Science
Pilani, Pilani Campus, Rajasthan (India)

COURSE DETAILS

Research Area / Title: Recurrent Neural Networks (Deep Learning)

Duration: March 2021 – June 2021

Abstract: This report is based on the learning outcomes of Deep Learning in the area of Recurrent Neural Networks. Several observations have been done throughout the given period of time to complete the course work, with better understanding of the topics and the proof of concepts. Work has been proposed with maximum novelty in whatever format requested for demonstrating the knowledge on particular concept assigned for a particular duration. Learning Outcomes are displayed using problem formulation and solutions, given in each sections of report.

Confidentiality: The report is submitted by Sayan Dey (M.E Software Systems) for the completion of course work of Research Practice in Recurrent Neural Networks. It may not be used for any other purposes, reproduced in whole or in part, nor passed to any commercial organization or person without the specific permission in writing of BITS Pilani or participants concerning this project.

Author Details:

Sayan Dey, M.E Software Systems

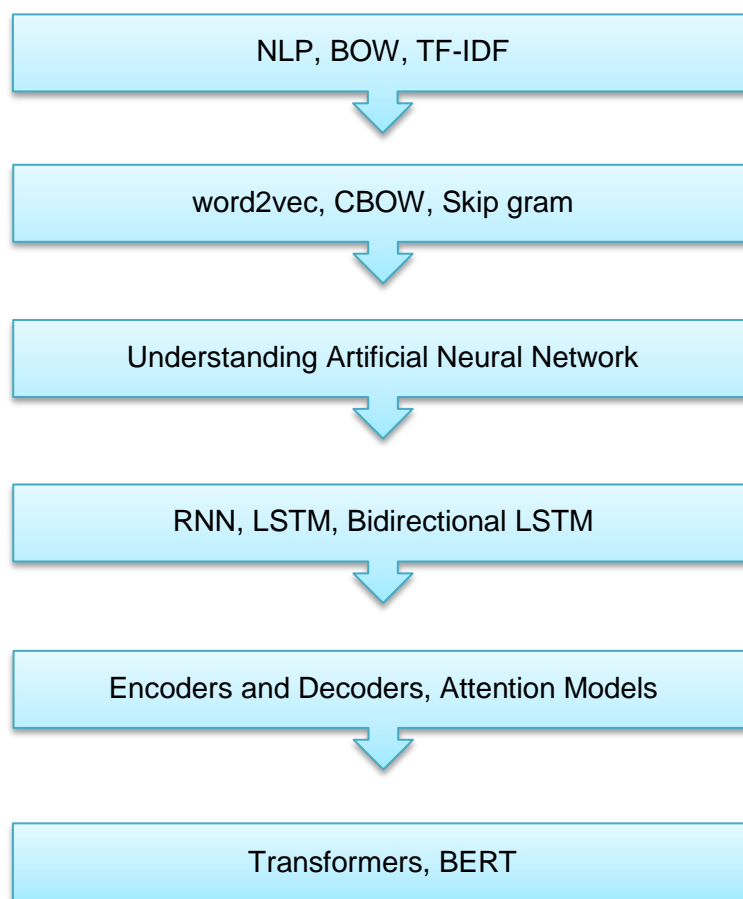
BITS ID: 2020H1120273P

TABLE OF CONTENTS

1 Introduction	05
2 NLP – Preprocessing Techniques	06
2.1 Bag of Words (BOW)	06
2.2 TF-IDF	06
2.3 Case Study	08
2.4 Problems with BOW and TFIDF	10
3 Word2vec	10
3.1 Training word2vec model	11
3.2 One Hot Representation	11
3.3 CBOW and Skip gram	12
3.4 Case Study	12
4 RNN, LSTM, Bi-LSTM	16
4.1 RNN Architecture	16
4.2 Drawback in Traditional RNN	18
4.2 LSTM and Bi-LSTM	18
4.2 Case Study I	20
4.3 Activation Functions	23
4.3 Case Study II	24
5 Encoder Decoder.....	26
5.1 BLEU Score	26
6 Conclusion	27
Notebook References	

1 Introduction

The main objective is to study about Recurrent Neural Network which is based upon the principle of sequential data as an input. Before proceeding to RNN, one should have a good idea on Natural language processing (NLP) which deals upon mainly on textual data. The data from surrounding can be interpreted as texts so that with the help NLP techniques, data can be converted to the machine understandable format, upon training the machine starts mimicking the human brain. There are different levels of NLP techniques through which text processing can be done, for converting words into vectors. After the text processing, we focus on different RNN techniques which helps in training series of texts with the use of input, output and hidden layers in neural network exhibiting different performance, efficiency and accuracy and in different use cases. Given below is the flowchart is depicting the series of learning undertaken from top to bottom for mastering RNN.



2 NLP – Preprocessing Techniques

Two popular preprocessing techniques are discussed in this section – Bag of Words (BOW), TF-IDF. When we are dealing with the text data, it is important to convert to other format so that the machine can interpret the text. These are two popular frequency based approach for word embedding where frequency of words in a corpus is taken a consideration to find the importance of it.

2.1 Bag of Words (BOW)

Represent a sentence as a bag of words vector (a string of numbers).

Review 1: This movie is very scary and long
Review 2: This movie is not scary and is slow
Review 3: This movie is spooky and good

	1 This	2 movie	3 is	4 very	5 scary	6 and	7 long	8 not	9 slow	10 spooky	11 good	Length of the review(in words)
Review 1	1	1	1	1	1	1	1	0	0	0	0	7
Review 2	1	1	2	0	0	1	1	0	1	0	0	8
Review 3	1	1	1	0	0	0	1	0	0	1	1	6

Library:

```
From sklearn.feature_extraction.text import CountVectorizer  
cv=CountVectorizer(max_features='..', ngram_range='..');
```

2.2 Term Frequency – Inverse Document Frequency (TF-IDF)

Term Frequency: It is a measure of how frequently a word appears in a sentence.

TF = (No. of repetition of words in sentence / No. of words in a sentence)

Review: This movie is not scary and is slow

- Number of words in Review = 8
- TF for the word 'this' = (number of times 'this' appears in review 2)/(number of terms in review 2) = 1/8

Term	Review 1	Review 2	Review 3	TF (Review 1)	TF (Review 2)	TF (Review 3)
This	1	1	1	1/7	1/8	1/6
movie	1	1	1	1/7	1/8	1/6
is	1	2	1	1/7	1/4	1/6
very	1	0	0	1/7	0	0
scary	1	1	0	1/7	1/8	0
and	1	1	1	1/7	1/8	1/6
long	1	0	0	1/7	0	0
not	0	1	0	0	1/8	0
slow	0	1	0	0	1/8	0
spooky	0	0	1	0	0	1/6
good	0	0	1	0	0	1/6

IDF is a measure of how important a word is.

IDF= log (No. of sentences / No. of sentences containing the word)

Term	Review 1	Review 2	Review 3	IDF
This	1	1	1	0.00
movie	1	1	1	0.00
is	1	2	1	0.00
very	1	0	0	0.48
scary	1	1	0	0.18
and	1	1	1	0.00
long	1	0	0	0.48
not	0	1	0	0.48
slow	0	1	0	0.48
spooky	0	0	1	0.48
good	0	0	1	0.48

After calculating the TF-IDF score for every word in each sentence,

Term	Review 1	Review 2	Review 3	IDF	TF-IDF (Review 1)	TF-IDF (Review 2)	TF-IDF (Review 3)
This	1	1	1	0.00	0.000	0.000	0.000
movie	1	1	1	0.00	0.000	0.000	0.000
is	1	2	1	0.00	0.000	0.000	0.000
very	1	0	0	0.48	0.068	0.000	0.000
scary	1	1	0	0.18	0.025	0.022	0.000
and	1	1	1	0.00	0.000	0.000	0.000
long	1	0	0	0.48	0.068	0.000	0.000
not	0	1	0	0.48	0.000	0.060	0.000
slow	0	1	0	0.48	0.000	0.060	0.000
spooky	0	0	1	0.48	0.000	0.000	0.080
good	0	0	1	0.48	0.000	0.000	0.080

Library:

```
From sklearn.feature_extraction.text import TfidfVectorizer
tfidf=TfidfVectorizer();
```

2.3 Case Study

❖ Problem Formulation

The problem deals with Stock sentiment analysis using news headlines. The dataset in consideration is a combination of the world news and stock price shifts. There are 25 columns of top news headlines for each day in the data frame. Class label '1' indicates increase in stock price, whereas Class label '0' indicates decrease or same stock price.

Objective: Using NLP, predict whether the stock will increase or decrease. Analysing the performance of BOW and TF-IDF with respect to accuracy.

❖ Analysis

1. Creating BOW and TF-IDF model.

```
[ ] # Creating the Bag of Words model
    from sklearn.feature_extraction.text import CountVectorizer
    from sklearn.ensemble import RandomForestClassifier
```

```
[ ] ## implement BAG OF WORDS
    countvector=CountVectorizer(ngram_range=(2,3))
    cv_train=countvector.fit_transform(headlines)
```

```
[ ] # Creating the TF-IDF model
    from sklearn.feature_extraction.text import TfidfVectorizer
    #from sklearn.ensemble import RandomForestClassifier
```

```
[ ] #implement TF-IDF
    tfvector=TfidfVectorizer(ngram_range=(2,3))
    tfd_train=tfvector.fit_transform(headlines)
```

2. Training both models using Random Forest Classifier.

```
[ ] # implement RandomForest Classifier
    randomclassifier=RandomForestClassifier(n_estimators=200,criterion='entropy')
    randomclassifier.fit(cv_train,train['Label'])
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                        criterion='entropy', max_depth=None, max_features='auto',
                        max_leaf_nodes=None, max_samples=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=200,
                        n_jobs=None, oob_score=False, random_state=None,
                        verbose=0, warm_start=False)
```

```
[ ] cv_test = countvector.transform(test_transform)
      predictions_cv = randomclassifier.predict(cv_test)
```

```
▶ ## Import library to check accuracy for BAG OF WORDS
  from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

  matrix_cv=confusion_matrix(test['Label'], predictions_cv)
  print(matrix_cv)
  score_cv=accuracy_score(test['Label'], predictions_cv)
  print(score_cv)
  report_cv=classification_report(test['Label'], predictions_cv)
  print(report_cv)
```

```
↳ [[135  51]
    [  1 191]]
0.8624338624338624
```

	precision	recall	f1-score	support
0	0.99	0.73	0.84	186
1	0.79	0.99	0.88	192
accuracy			0.86	378
macro avg	0.89	0.86	0.86	378
weighted avg	0.89	0.86	0.86	378

86% accuracy for Bag of Words using Random Forest Classifier.

```
[ ] tfd_test = tfvector.transform(test_transform)
      predictions_tfd = randomclassifier.predict(tfd_test)
```

```
▶ ## Import library to check accuracy for BAG OF WORDS
  from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

  matrix_tfd=confusion_matrix(test['Label'], predictions_tfd)
  print(matrix_tfd)
  score_tfd=accuracy_score(test['Label'], predictions_tfd)
  print(score_tfd)
  report_tfd=classification_report(test['Label'], predictions_tfd)
  print(report_tfd)
```

```
↳ [[148  38]
    [ 23 169]]
0.8386243386243386
```

	precision	recall	f1-score	support
0	0.87	0.80	0.83	186
1	0.82	0.88	0.85	192
accuracy			0.84	378
macro avg	0.84	0.84	0.84	378
weighted avg	0.84	0.84	0.84	378

84% accuracy for TF-IDF model using Random Forest Classifier.

After training the Count Vectorizer and tfidf Vectorizer with Random Forest Classifier, the accuracy is almost similar after prediction with test data for the given Stock sentiment dataset.

2.4 Problems with BOW and TF-IDF

1. If the new sentences contain new words, then the vocabulary size would increase and thereby, the length of the vectors would increase too. If the vectors contain many 0s, it will result in a sparse matrix.
2. Problem with detecting the similarity between the words, or translation of words into another language requires more information.
3. Semantic information is not retained nor the ordering of the words in the text. So it is not useful for sentiment analysis.

3 Word2vec

Word2Vec is a shallow, two-layer neural networks which is trained to reconstruct linguistic contexts of words.

It takes as its input a large corpus of words and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space.

Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space. So, semantic information and relation between different words is also preserved.

It is computationally-efficient predictive model for learning word embeddings from raw text.

It comes in two flavors, the Continuous Bag-of-Words (CBOW) model and the Skip-Gram model.

Popular Applications:

1. Sentiment analysis
2. Recommender systems
3. Tags in Quora: Given a tag, we want to find similar tags in order to recommend them to a user for exploration. We can do this by treating each set of co-occurring tags as a "sentence" and train a Word2Vec model on this data.

3.1 Training word2vec model

Steps for training:

1. Do basic preprocessing tasks.
2. Represent each word in One-Hot Representation
3. Generating the model where Dimensionality reduction will take place after specifying fixed no. of features. Word2vec object 'sg' can be used to identify the training algorithm, Skip gram or CBOW.

```
model = gensim.models.Word2Vec(sentence, vector_size=150, window=10, min_count=2, sg=1, workers=10)
```

vector_size = Dimensionality of word vectors

window = Maximum distance between the target word and the neighbouring word

min_count = Ignores all words with total frequency lower than this

workers = No. of worker threads used to train the model

4. Train the model and find correlation among words.

```
model.train(sentences=documents, total_examples=len(documents), epochs=model.iter)
```

3.2 One Hot Representation

Considering the following example,

- Agama is a reptile.
- Snake is a reptile.
- Reptile is a cold-blooded animal.

As there are 7 unique words, the length of the vector is 7.

```
["agama", "reptile", "snake", "a", "is", "cold-blooded", "animal"]
```

```
"Agama" => [1, 0, 0, 0, 0, 0, 0]
```

```
"Reptile" => [0, 1, 0, 0, 0, 0, 0]
```

3.3 CBOW and Skip gram

CBOW	Skip gram
CBOW predicts target words (e.g. 'mat') from the surrounding context words ('the cat sits on the').	Skip-gram predicts surrounding context words from the target words (inverse of CBOW).
Statistically, it has the effect that CBOW smoothes over a lot of the distributional information (by treating an entire context as one observation). For the most part, this turns out to be a useful thing for smaller datasets.	Statistically, skip-gram treats each context-target pair as a new observation, and this tends to do better when we have larger datasets.
Computationally moderate	Computationally expensive

3.4 Case Study

❖ Problem Formulation

The problem deals with the Sentiment Analysis of Flight data. The concerned data set is extracted from Twitter where tweets have been expressed in positive, negative and neutral sentiments for the U.S Airlines on the month of February 2015. Classification will be done upon the attribute "Airline Sentiment".

Objective: The objective is to train the text of different tweets using word2vec and find the similar words related to a negative or positive word. Also, compare the performance of Skip gram and CBOW for the given dataset.

1. In the dataset, there are some attributes which have lots of null values. So we can remove those attributes which will not add value for the sentiment analysis.

```
df.isnull().sum()

tweet_id      0
airline_sentiment      0
airline_sentiment_confidence      0
negativereason      5462
negativereason_confidence      4118
airline      0
airline_sentiment_gold      14600
name      0
negativereason_gold      14608
retweet_count      0
text      0
tweet_coord      13621
tweet_created      0
tweet_location      4733
user_timezone      4820
dtype: int64

df.shape

(14640, 15)
```

```
df.drop(['negativereason', 'negativereason_confidence', 'airline_sentiment_gold', \
        'negativereason_gold', 'tweet_coord', 'tweet_location', 'user_timezone'], axis='columns', inplace=True)
```

```
df.shape

(14640, 8)
```

2. For binary classification, neutral and positive can be mapped as 1.

```
df.airline_sentiment=df.airline_sentiment.map({'neutral' : 1, 'positive':1, 'negative':0})
df.airline_sentiment[0:10]

0    1
1    1
2    1
3    0
4    0
5    0
6    1
7    1
8    1
9    1
Name: airline_sentiment, dtype: int64
```

❖ Analysis

1. Creating the word2vec model using CBOW (sg=0), and finding the top 15 similar words to “inconvenience”.

```
#Using CBOW
model1 = word2vec.Word2Vec(wt, workers=num_workers, size=num_features, \
                           min_count = min_word_count, window = context, \
                           sg=0, sample = downsampling)
```

```

▶ model1.most_similar('inconvenience',topn =15)

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher
"""Entry point for launching an IPython kernel.
[('wifi', 0.9999246597290039),
 ('many', 0.9999239444732666),
 ('club', 0.9999225735664368),
 ('ask', 0.999921441078186),
 ('sense', 0.9999125003814697),
 ('case', 0.9999110698699951),
 ('clothes', 0.9999085664749146),
 ('2015', 0.9999064207077026),
 ('big', 0.9999056458473206),
 ('anywhere', 0.999905526638031),
 ('needs', 0.9999048113822937),
 ('mail', 0.9999028444290161),
 ('say', 0.9999017715454102),
 ('internet', 0.9999011158943176),
 ('show', 0.9998999238014221)]

```

2. Creating the word2vec model using Skip gram (sg=1), and finding the top 15 similar words to “inconvenience”.

```

[ ] #Using Skipgram
model2 = word2vec.Word2Vec(wt, workers=num_workers, size=num_features,\
                           min_count = min_word_count, window = context, \
                           sg=1, sample = downsampling)

```

```

▶ model2.most_similar('inconvenience',topn =15)

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher
"""Entry point for launching an IPython kernel.
[('reliable', 0.9963783621788025),
 ('hoped', 0.9963176250457764),
 ('confusing', 0.9960012435913086),
 ('refundable', 0.9956945180892944),
 ('harbor', 0.9956585764884949),
 ('exception', 0.9955346584320068),
 ('proving', 0.9950238466262817),
 ('expenses', 0.9950082302093506),
 ('sincere', 0.9949765205383301),
 ('notimpressed', 0.9949107766151428),
 ('nonprofit', 0.9947516918182373),
 ('manch', 0.9947259426116943),
 ('opinion', 0.9946929216384888),
 ('getittogether', 0.9946870803833008),
 ('rectify', 0.9946756362915039)]

```

3. Plotting the output of both models using t-SNE graphic applied on vocabulary of 500 words of approx. 4400 words. Hence, some words are not included in the plot, but it will give approximate estimate of distribution of words in two dimension.

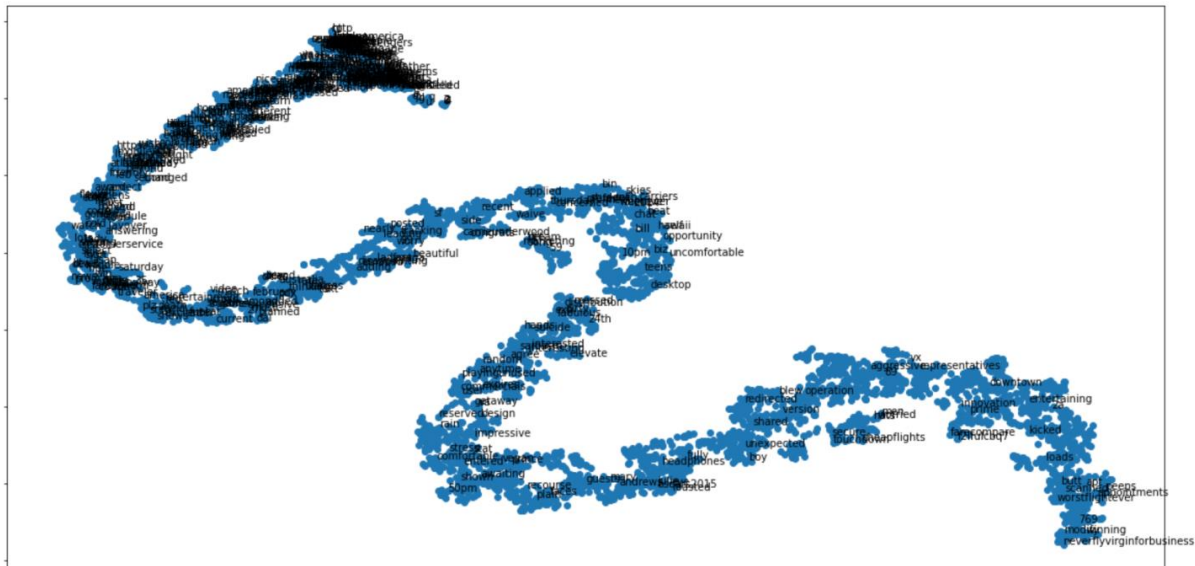


Fig: Output of t-SNE plot using CBOW

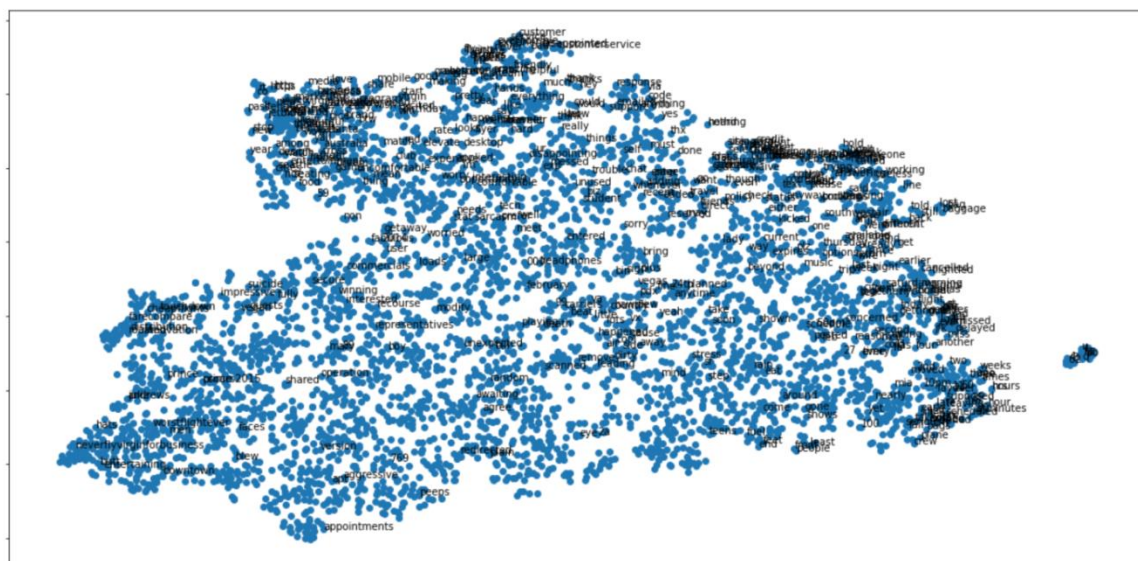


Fig: Output of t-SNE plot using Skip gram

In the given output, we can see that we are getting skinny cluster in case of CBOW, and a globular or dense cluster in case of Skip gram.

As CBOW tries to predict the words with the help of neighbours, it forms the similarity cluster in a line, whereas Skip gram predicts the neighbor of a word after studying different context. So it is computationally expensive, but tends to give a better output.

4 RNN, LSTM, Bi-LSTM

Using the NLP Text Processing Techniques, we are able to convert words into vectors with varying importance, but the sequence information gets discarded. Hence to control sequence information, we use RNN.

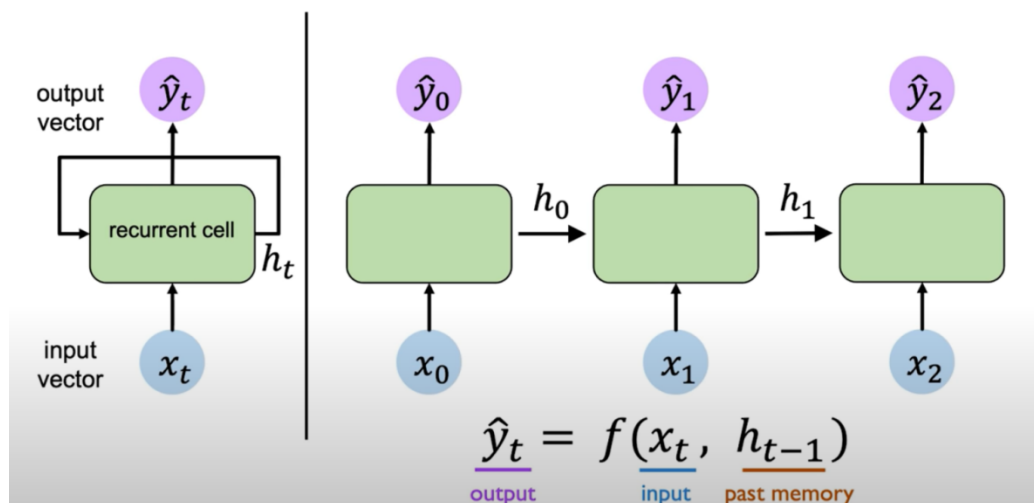
So we use RNN to preserve the sequential information, do the sequence modeling to predict the sequential output.

Popular Applications:

1. Time Series data
2. Stock price prediction
3. Genomic sequence data
4. Voice Assistant
5. Image captioning
6. Language Translator
7. Sentiment Analysis

4.1 RNN Architecture

Individual text/words can be thought of time step in the sequence.



Apply the given recurrence relation at every time step,

$$h_t = f_w(x_t, h_{t-1})$$

RNNs have state h_t , that is updated at each time as sequence is processed. f_w is a function of weights, and x_t is an input at time t .

We input a sentence into the RNN Network, each word is considered as individual time step.

Output of the first word will be sent as an input to the second word in the hidden layer which forms the basis of RNN.

For example, first output can be written as,

$$O_1 = f(x_1 * W) + O_0 w'$$

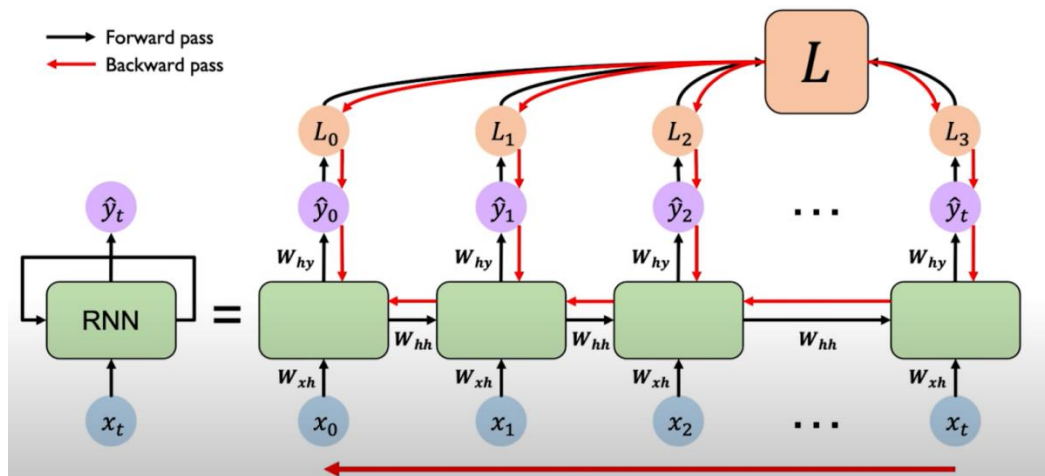
$$O_2 = f(x_2 * W) + O_1 w'$$

$$O_3 = f(x_3 * W) + O_2 w' \text{ and so on....}$$

Due to this process, the sequence information is maintained.

Backpropagation:

After calculating the loss function from the output at the end of the sequence, backpropagation is done in order to adjust the weights so that the loss can be minimized.



For updating weights while backpropagating:

1. Calculate Derivative of Loss with respect to y . (DL/Dy)
2. Calculating derivative of Loss with respect to previous weight,

$$DL/Dw'' = (DL/Dy)(Dy/Dw'') \quad (\text{Using chain rule})$$

$$w'' = w'' - (DL/Dw'') \quad (\text{Updating weights})$$

3. Considering $(O_4 * w)$ was the input to activation function for getting the final output as y , and O_4 depending on previous weight w ,

$$DL/Dw = (DL/Dy)(Dy/DO_4)(DO_4/Dw) \quad (\text{Using chain rule})$$

$$w = w - (DL/Dw)$$

After some number of iterations, we will reach the global minima where there will be minimum loss, and weights have been learned accurately in optimum time.

4.2 Drawback in Traditional RNN (Long Term Dependencies)

1. Vanishing Gradient problem

Repeatedly multiply smaller numbers while apply chain rule of derivatives across every time step (Slow convergence). Usually occurs when sigmoid activation function is used.

2. Exploding Gradient problem

If weights are higher and derivatives are greater than 1 (if ReLu is used). Then (DL/Dw) is very large.

Unfortunately, as that gap grows, RNNs are unable to learn to connect the information.

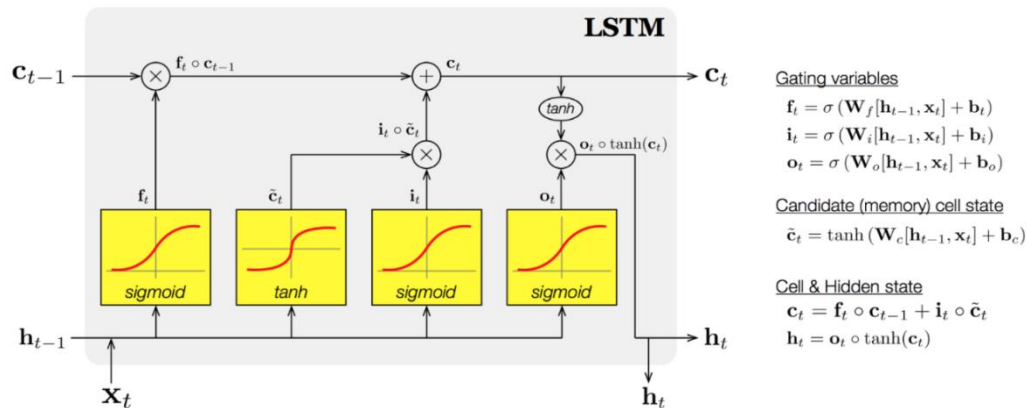
4.3 LSTM and Bi-LSTM

Introducing a more complex RNN unit with gates which can track the long term dependencies by intuitively controlling the information passed through cell which we call gated cells. It tracks information throughout many time steps. They are specially developed to deal with exploding and vanishing gradient problem when training traditional RNNs. LSTM is one such popular gated unit architecture which is used in many deep learning applications nowadays.

LSTM unit is composed of a cell, an **input gate**, an **output gate** and a **forget gate**. The cell is responsible for "remembering" values over arbitrary time intervals, hence the word "memory" in LSTM. Each of the three gates can be thought of as a "conventional" artificial neuron, as in a multi-layer (or feed forward) neural network, that is, they compute an activation (using an activation function) of a weighted sum.

Long short-term refers to the fact that LSTM is a model for the short-term memory which can last for a long period of time. An LSTM is well-suited to classify, process and predict time series given time lags of unknown size and duration between important events.

LSTM Architecture:



- Forget Gate “f” (a neural network with sigmoid)
- Candidate layer “C”(a NN with Tanh)
- Input Gate “I” (a NN with sigmoid)
- Output Gate “O”(a NN with sigmoid)
- Hidden state “H” (a vector)
- Memory state “C” (a vector)
- Inputs to the LSTM cell at any step are x_t (current input) , h_{t-1} (previous hidden state) and c_{t-1} (previous memory state).
- Outputs from the LSTM cell are h_t (current hidden state) and c_t (current memory state).

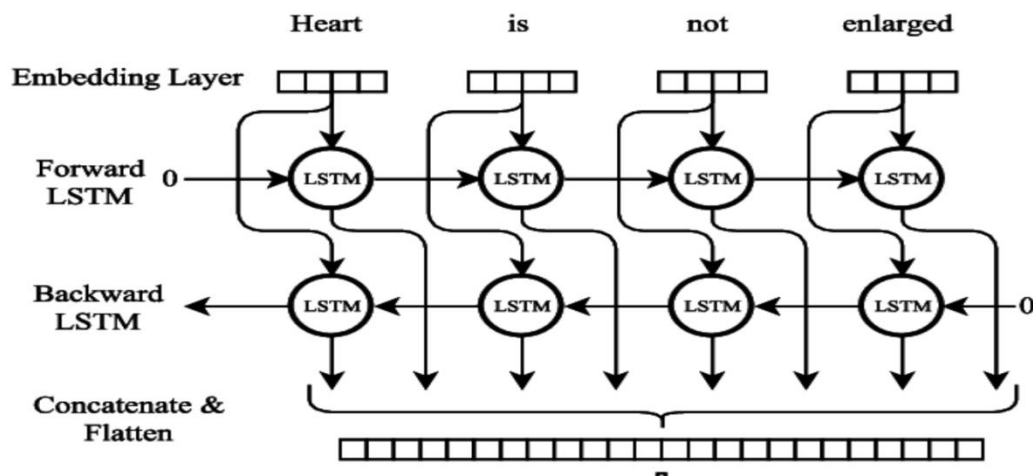
Bidirectional LSTM Architecture:

With respect to context of LSTM, RNN at a particular time step will not only get the information about the words in the previous time steps, but also have the information about the words in future time steps. That is why the architecture is named bidirectional.

The idea is to split the state neurons of a regular RNN in a part that is responsible for the positive time direction (forward states) and a part for the negative time direction (backward states).

It works well when we have sequence of words in the form of sentences is already given. In case of Speech recognition, the future words can't be available, so bidirectional LSTM will not work well.

It is slower than LSTM, but may provide better accuracy for dataset having long text.



4.4 Case Study I

❖ Problem Formulation

Sentiment analysis data has been considered mentioned in the Section 3.4. Non useful attribute has been dropped for further analysis.

```
tweet.head()
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	airline	name	retweet_count	text	tweet_created
0	570306133677760513	neutral	1.0000	Virgin America	cairdin	0	@VirginAmerica What @dhepburn said.	2015-02-24 11:35:52 -0800
1	570301130888122368	positive	0.3486	Virgin America	jnardino	0	@VirginAmerica plus you've added commercials t...	2015-02-24 11:15:59 -0800
2	570301083672813571	neutral	0.6837	Virgin America	yvonnalynn	0	@VirginAmerica I didn't today... Must mean I n...	2015-02-24 11:15:48 -0800
3	570301031407624196	negative	1.0000	Virgin America	jnardino	0	@VirginAmerica it's really aggressive to blast...	2015-02-24 11:15:36 -0800
4	570300817074462722	negative	1.0000	Virgin America	jnardino	0	@VirginAmerica and it's a really big bad thing...	2015-02-24 11:14:45 -0800

```
[ ] tweet.shape
```

```
(14640, 8)
```

Objective: The objective is to compare the performance of LSTM and Bi-LSTM for the given airline tweets dataset.

❖ Analysis

1. After preprocessing the tweets, each unique words is representation using one hot encoding.

```
▶ onehot_rep=[one_hot(words,voc_size)for words in corpus]
onehot_rep
```

```
▶ sent_length=30
embedded_docs=pad_sequences(onehot_rep,padding='pre',maxlen=sent_length)
embedded_docs
```

```
↳ array([[ 0,  0,  0, ..., 3657, 1859, 1209],
        [ 0,  0,  0, ...,  43,  906, 1767],
        [ 0,  0,  0, ..., 4303, 1685, 3753],
        ...,
        [ 0,  0,  0, ..., 1881, 4841, 4890],
        [ 0,  0,  0, ..., 3392,  22, 3672],
        [ 0,  0,  0, ..., 2642, 4873, 1198]], dtype=int32)
```

2. Creating LSTM and Bi-LSTM model, using 100 neurons in the hidden layer.

```
▶ ## Creating LSTM model
embedding_vector_features=40
model1=Sequential()
model1.add(Embedding(voc_size,embedding_vector_features,input_length=sent_length))
model1.add(LSTM(100))
model1.add(Dense(1,activation='sigmoid'))
model1.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
print(model1.summary())
```

↳ Model: "sequential_3"

Layer (type)	Output Shape	Param #
=====		
embedding_3 (Embedding)	(None, 30, 40)	200000

lstm_2 (LSTM)	(None, 100)	56400

dense_2 (Dense)	(None, 1)	101
=====		
Total params: 256,501		
Trainable params: 256,501		
Non-trainable params: 0		

None		

```
▶ ## Creating BiLSTM model
embedding_vector_features=40
model2=Sequential()
model2.add(Embedding(voc_size,embedding_vector_features,input_length=sent_length))
model2.add(Bidirectional(LSTM(100)))
model2.add(Dropout(0.3))
model2.add(Dense(1,activation='sigmoid'))
model2.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
print(model2.summary())
```

↳ Model: "sequential_4"

Layer (type)	Output Shape	Param #
=====		
embedding_4 (Embedding)	(None, 30, 40)	200000

bidirectional_1 (Bidirection	(None, 200)	112800

dropout_1 (Dropout)	(None, 200)	0

dense_3 (Dense)	(None, 1)	201
=====		
Total params: 313,001		
Trainable params: 313,001		
Non-trainable params: 0		

None		

3. Training both the models after splitting the input into Train and Test.

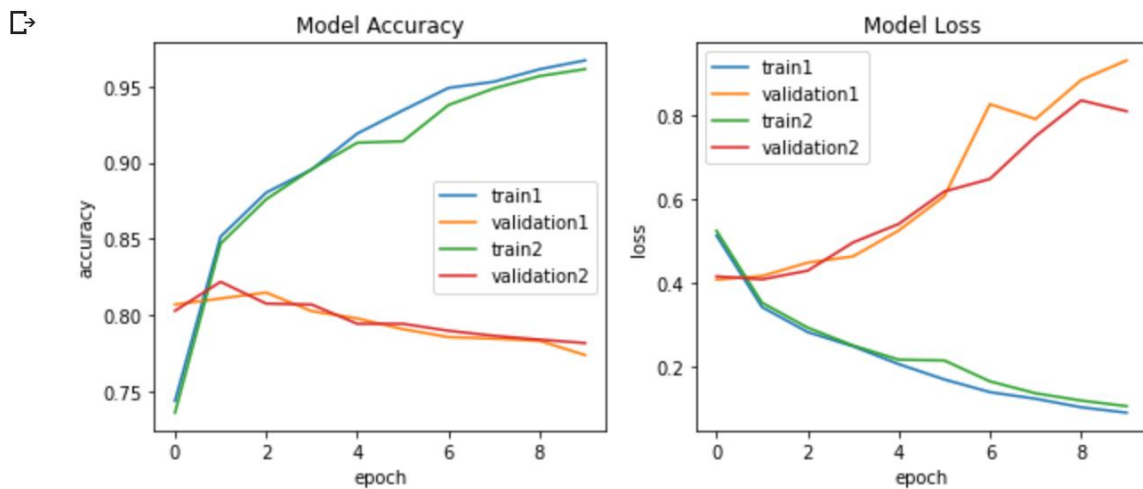
```
[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X_final, y_final, test_size=0.33, random_state=42)
```

```
▶ ##Training of LSTM Model
  stat1=model1.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=10,batch_size=64)
```

```
[ ] ##Training of BiLSTM Model
  stat2=model2.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=10,batch_size=64)
```

4. In the given plot,

- Training and Validation accuracy has been plotted for LSTM (blue, orange) and Bi-LSTM (green, red).
- Training and Validation loss has been plotted for LSTM (blue, orange) and Bi-LSTM (green, red).

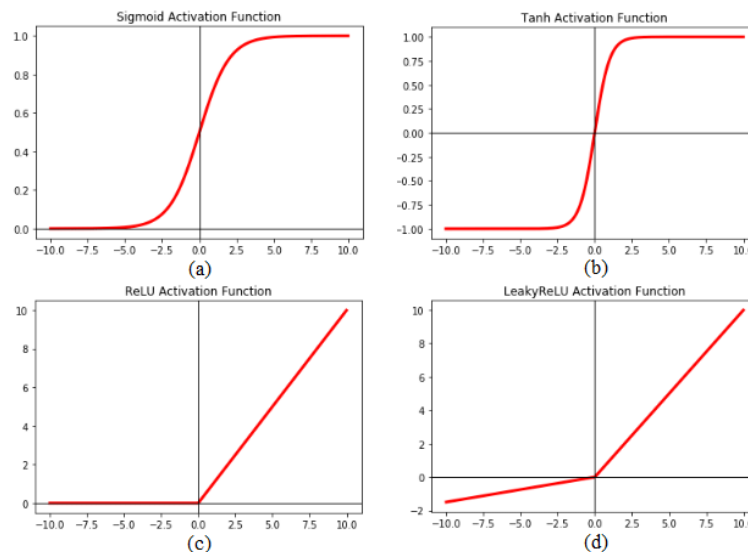


For the given dataset, LSTM and Bi-LSTM seems to work in a similar way. Each tweet should be long enough to see significant changes in Bi-LSTM. In that case higher accuracy could have been achieved earlier in Bi-LSTM.

According to the observation, validation accuracy and loss are good and there were no overfitting.

4.5 Activation Functions

Here in this stage, Activation Function becomes relevant to discuss to check the performance of different activation function in the hidden layer of RNN. We have discussed some drawback of RNN in previous section. Vanishing gradient problem occurs which leads to slower convergence, reaching global minima in more time. Same problem occurs with the Exploding gradient problem. Now we will discuss the characteristics of each and every activation function to see how they behave with LSTM.



Function name	Function	Derivative	Issues
sigmoid	$\epsilon(z) = 1/(1+e^{-z})$ $z=0, \epsilon(z)=0.5$ $z>0, 0.5<\epsilon(z)\leq 1$ $z<0, 0\leq\epsilon(z)<0.5$	$0\leq d\epsilon(z)/dz\leq 0.25$	Vanishing gradient problem.
tanh	$\tanh(z) = (e^z - e^{-z}) / (e^z + e^{-z})$ $z=0, \tanh(z)=0$ $z>0, 0<\tanh(z)\leq 1$ $z<0, -1\leq\tanh(z)<0$	$0\leq d \tanh(z)/dz\leq 1$	Data is zero centric, but relatively better than sigmoid.
ReLU	$\text{ReLU}(z) = \max(0, z)$ $z\leq 0, \text{ReLU}=0$ $z>0, \text{ReLU}=z$	$z\leq 0, d\text{ReLU}(z)/dz = 0$ $z>0, d\text{ReLU}(z)/dz = 1$	Solves Vanishing gradient but suffers from Exploding gradient; Dead neuron problem
Leaky ReLU	$\text{LReLU}(z) = \max(0.01z, z)$ $z\leq 0, \text{LReLU}=0.01z$ $z>0, \text{LReLU}=z$	$z\leq 0, d\text{LReLU}(z)/dz = 0.01$ $z>0, d\text{LReLU}(z)/dz = 1$	Solves Dying ReLU problem

4.6 Case Study II

❖ Problem Formulation

Considering the word embedding formed in section 4.4 in the tweets of airline sentiment data, our goal is to check performance of each of the activation function in the hidden layer. Loss function “binary_crossentropy” has been considered due to binary classification of the sentiments.

❖ Analysis

1. Model for sigmoid activation function in each layer.

```
#Input Layer
model3.add(Embedding(voc_size,embedding_vector_features,input_length=sent_length))
#First Hidden Layer
model3.add(LSTM(100))
#Second Hidden Layer
model3.add(Dense(100,activation='sigmoid'))
#Output Layer
model3.add(Dense(1,activation='sigmoid'))
model3.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

2. Model for tanh activation function in hidden layer.

```
#Input Layer
model4.add(Embedding(voc_size,embedding_vector_features,input_length=sent_length))
#First Hidden Layer
model4.add(LSTM(100))
#Second Hidden Layer
model4.add(Dense(100,activation='tanh'))
#Output Layer
model4.add(Dense(1,activation='sigmoid'))
model4.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

3. Model for ReLu activation function in hidden layer.

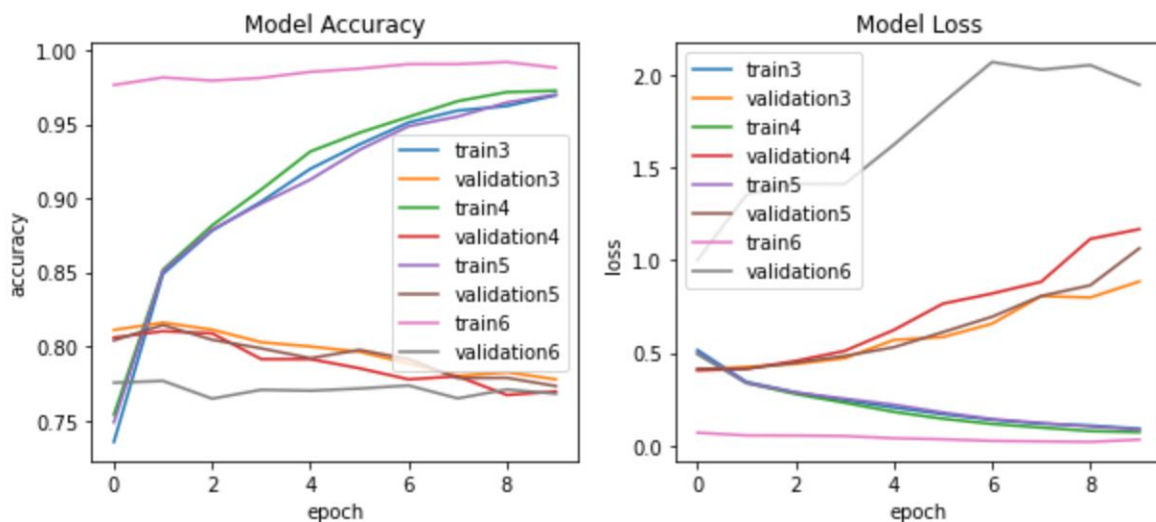
```
#Input Layer
model5.add(Embedding(voc_size,embedding_vector_features,input_length=sent_length))
#First Hidden Layer
model5.add(LSTM(100))
#Second Hidden Layer
model5.add(Dense(100,activation='relu'))
#Output Layer
model5.add(Dense(1,activation='sigmoid'))
model5.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

4. Model for Leaky ReLu activation function in hidden layer.

```
#Input Layer
model6.add(Embedding(voc_size,embedding_vector_features,input_length=sent_length))
#First Hidden Layer
model6.add(LSTM(100))
#Second Hidden Layer
model6.add(Dense(100,activation='LeakyReLU'))
#Output Layer
model6.add(Dense(1,activation='sigmoid'))
model6.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

5. In the given plot,

- Training and Validation accuracy has been plotted for sigmoid (blue, orange), tanh (green, red), ReLU (magenta, brown) and Leaky Relu (pink, gray).
- Training and Validation loss has been plotted for sigmoid (blue, orange), tanh (green, red), ReLU (magenta, brown) and Leaky Relu (pink, gray).



LeakyRelu attains the maximum accuracy at early stage. Whereas, tanh performs better than ReLu (suffers from dead neuron) and sigmoid (suffers from vanishing gradient).

Hence, we can say we overcome lot of the traditional RNN problems using LeakyReLU in hidden layer, but overfitting may occur. So, tanh can be used conveniently for LSTM.

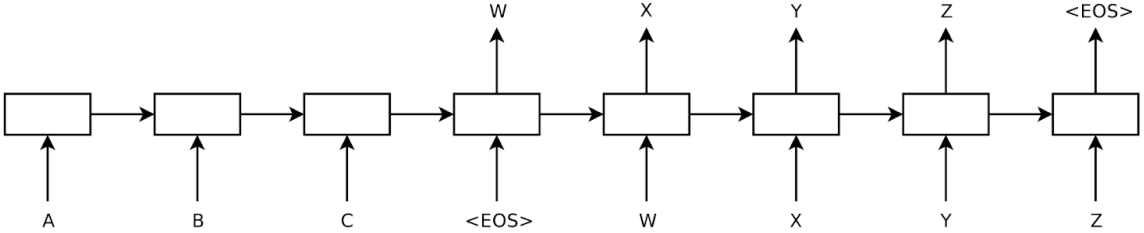
5 Encoder Decoder

Encoder Decoder Architecture basically focuses on converting sequence to sequence. It works on the principle of LSTM where output of the input vector will be available only at the end of the time step (Encoder). Then the context vector will be passed to the decoder and the output of a given time step will be passed as an input to succeeding time steps.

Popular applications:

1. Language translation
2. Automated reply in whatsapp, linkedin
3. Image Captioning (Advanced CNN techniques used in place of Encoder)

Encoder	Decoder
Output will be available only at the last timestamp or end of the sequence in the form of context vector.	After the context vector is received at first time step of Decoder, output is passed to the succeeding time steps on so on till the end of the sequence.
Context vector is passed to Decoder.	Input sequence (Encoder size) and Output sequence (Decoder size) may be different.



5.1 BLEU Score

The major problem with Encoder Decoder is that the accuracy is not good for long sentences. It has been observed that as the sentence length increases with respect to the Encoders and Decoders, the accuracy decreases. That is why the metric BLEU (Bilingual Evaluation Understudy Score) is used for evaluating a generated sentence to a reference sentence. A perfect match results in a score of 1.0, whereas a perfect mismatch results in a score of 0.0.

6 Conclusion

The main focus topics which are discussed on the report were limited to Encoder Decoder. A significant amount of learning is done for understanding the concepts in depth and doing the appropriate analysis in the same. It has been shown that RNN is performing better than NLP when the output is pertaining to the sequence of input data. But with different kinds of datasets, results may vary in LSTM and Bi-LSTM. Future work can be extended by doing comparable analysis of discussed architecture in the report with the Attention models, Transformers and BERT, with several case studies of sequence to sequence, vector to sequence translation.

Notebook References

Section	Name	Data Set	Code
2.3	BOW TFIDF	StockData.csv	StockPredict_BOW_TFIDF.ipynb
3.4	Word2vec	Tweets.csv	RNN_analysis.ipynb
4.4	LSTM Bi-LSTM	Tweets.csv	RNN_analysis.ipynb
4.6	Activation Functions	Tweets.csv	RNN_analysis.ipynb