# STOCKGRAM : DEEP LEARNING MODEL FOR DIGITIZING FINANCIAL COMMUNICATIONS VIA NATURAL LANGUAGE GENERATION

Purva Singh

School of Computer Science and Engineering, VIT University, Vellore

## *ABSTRACT*

*This paper proposes a deep learning model, StockGram, to automate financial communications via natural language generation. StockGram is a seq2seq model that generates short and coherent versions of financial news reports based on the client's point of interest from numerous pools of verified resources. The proposed model is developed to mitigate the pain points of advisors who invest numerous hours while scanning through these news reports manually. StockGram leverages bi-directional LSTM cells that allows a recurrent system to make its prediction based on both past and future word sequences and hence predicts the next word in the sequence more precisely. The proposed model utilizes custom word-embeddings, GloVe, which incorporates global statistics to generate vector representations of news articles in an unsupervised manner and allows the model to converge faster. StockGram is evaluated based on the semantic closeness of the generated report to the provided prime words.*

## *KEYWORDS*

*Natural language generation, bi-directional LSTM networks, language modelling, GloVe embeddings*

## 1. INTRODUCTION

In the past years, Artificial Intelligence has seen rapid improvements in the field of natural language and we can see various sub-domains evolving at the intersection of AI and linguistics. The tendency to establish a two-way communication with computers has given rise to a distinguished sub-category of tasks that is concerned with generating (quasi)-natural speech. This category is referred to as Natural Language Generation (NLG). Artificial Intelligence : Natural Language Processing Fundamentals [2] defines Natural Language Generation as "process of producing meaningful phrases and sentences in the form of natural language". Major applications of NLG include generating financial reports, meeting agendas, chatbots, text summarization and many more. The goal of NLG is to predict the next word, given a sentence as input. This task of selecting the next probable word amongst a lot of possibilities is performed by *language models*. The construction of these language models can be at character level, bi-gram level, n-gram level, sentence and even paragraph level. The usage of feed-forward networks as language models is not encouraged as an appropriate model for natural language generation. The main reason as to why feed-forward neural networks are not considered optimal for NLG is because they do not take into account the order of inputs. In natural language generation, the sequence of encoding is important [3]. This structure is analogous to time-series data and word order is of cardinal importance. Hence, utilizing feed-forward networks for text generation would lead to information loss and generated sentences might not make sense and could be grammatically incorrect. In order to take into account both time and sequence, Recurrent Neural Networks (RNNs) are used for natural language processing related tasks.

This paper introduces StockGram, a deep learning model to automatize financial communication via natural language generation. StockGram is a bi-directional LSTM network that captures both the backward and forward information about the sequence at every time-step. In order to better capture the semantic information of words, StockGram utilizes pre-trained word-embeddings known as GloVe for a better vectorized representation of the words. The model is trained on latest financial tweets and news reports on stocks, which is pre-processed before feeding into the model. StockGram takes a set of prime words as input and generates a concise report which is syntactically inline with those prime words. In comparison to a vanilla RNN model for natural language generation, the StockGram model converges faster and generates concise reports which are more syntactically inline with the prime words. This difference in model's performance is due to the structure of bidirectional LSTMs, where the model gets to make a prediction of the next likely word based on both past and future sequences and hence predicts the next word in the sequence more precisely.

The main purpose of StockGram is to mitigate the pain points of advisors who scan through multiple financial reports, decks, news articles and tweets to collate meeting points for the clients. The proposed model takes a set of prime words which are related to the client's point of interest and based on the pre-processed financial data fed into the model, StockGram generates a set of consolidated financial reports which can save hours of the advisor's efforts to scan through numerous financial resources.

This paper is divided into the following sections: Section 2 explains the reasoning behind choosing bi-directional LSTM model with pre-trained word embeddings. Section 3 defines related work done in the domain of natural language generation followed by Section 4 that describes the problem statement we have addressed in the paper, followed by Section 5 that defines the dataset and methodology adopted for text generation. Section 6 shows the experimental results conducted for achieving the optimal model for natural language generation. The paper concludes by discussing conclusions and mentioning future work to be done on the proposed model.

## 2. MODEL DEFINITION

### 2.1. Bidirectional LSTM Units

Bi-directional LSTMs are an extension to traditional LSTM which tends to improvise on the model's performance in the text generation domain. In scenarios where, all timesteps of input sequence are provided, bi-directional LSTMs put together two independent LSTMs instead of one LSTM and train them on the input. Figure 6 shows the structure of a bi-directional LSTM. This structure allows the model to have both past and future information about input sequences. In the two networks, one runs from past to future and another one from future to past. The first on the input sequence as-is and the second on a reversed copy of the input sequence [5]. Due to this nature of bi-directional LSTM, the model gets additional context about the sentence and it converges faster as compared to vanilla uni-directional LSTM. For example, if we consider a scenario were sequence of words is as follows :

"*Due to the negative impact of the corona pandemic, the oil prices have__*."

Since a typical vanilla RNN retains only short-term information, it will assign the probability of next word based on "oil prices have" and can predict words such as "risen", "improved", "increasing". But an LSTM model will retain the long-term information and take into account

"negative impact of coronavirus" and hence would predict the word "plunged". It is this nature of bidirectional LSTM that StockGram utilizes in order to generate reports that are syntactically inline with the prime words provided as input to the model.

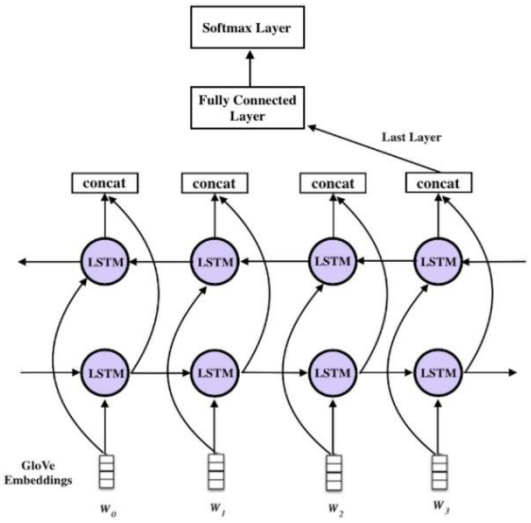


Figure 1: General architecture of bidirectional LSTM that uses pre-trained GloVe embeddings.

## 2.2. Pre-trained Word Embeddings

Pre-trained word embeddings are the embeddings that are trained on huge datasets and learned on one task and those learnings are then used for solving another analogous task. These pre-trained word embeddings can be thought of as a form of Transfer Learning where learnings of one task are transferred to another task. StockGram uses pre-trained word embeddings because they capture the semantic and syntactic meaning of a word as they are trained on large datasets. These word embeddings boost the performance of our NLP model. One of the reasons for not choosing StockGram's own embeddings is because the model takes into account numerous financial reports, news and tweets as its training data which comprises large volumes of rare corpus. The StockGram embeddings that would be learned from this training data won't be able to arrive at the correct vector representation of the word. Secondly, if the model learns embeddings from scratch, the number of training parameters would increase which would eventually lead to a slower training process. StockGram leverages Stanford's pre-trained word embeddings called GloVe which relies on both local statistics (local context information of words, word-based models) and global statistics (word co-occurrence, count-based models) in order to obtain vectorized representation of words [7]. GloVe optimizes our model one count at a time which leads to faster model training. The GloVe model is scalable, captures the semantics irrespective of word frequency and performs efficiently on both large and small text corpus.

# 3. LITERATURE SURVEY

Dipti Pawade et. al [9] have developed a LSTM based Story Scrambler that takes into input different stories. These stories can have different or same storyline. After pre-processing of input text, the pre-processed words are one-hot encoded before feeding them into the model. Batches of pre-processed data based on sequence length are then fed into the model. Next word is then selected with the highest probability and the window is shifted by one word. This process continues till the desired number of words required in the story is reached. They have trained the

model by performing various hyper-parameter tunings by varying the RNN size, number of layers, batch size and sequence length. The accuracy of the generated text by their model w.r.t grammar and correctness of sentences was validated by seven different people and their best model has achieved 63% accuracy as evaluated by humans. This model can be further improved by using bi-directional LSTM instead of uni-directional LSTM model structure and by using pre-trained word embeddings such as GloVe or word2vec instead of one-hot encodings which are computationally inefficient.

Gregory Luppescu et al. [10] presents a LSTM based text-generation model to predict the next word in the sequence. They have mentioned two separate tasks for text generation purposes: per-author and mix-of-authors text generation that determines the source of text from a single author or different authors respectively. To minimize the errors in text generation, they have used perplexity, which is the inverse probability of the correct word. Hence, perplexity is inversely proportional to the correctness of generated text. They have used pre-trained GloVe embeddings for text-generation and a dynamic learning rate for the model. The test perplexity for most of the authors lied between 100-200, thus proving high accuracy of the model but with few outliers. The model can be further improved by utilizing bi-directional LSTM for text generation.

Nallapati R. et. al have proposed an abstractive text summarization model using encoder and decoder, that was originally designed for machine translation, along with attention and have achieved state-of-the-art performance on two different corpora [14]. They have taken a baseline model for text generation consisting of bi-directional GRU-RNN encoder and uni-directional decoder and have proposed several models to tackle specific drawbacks of the baseline model. They have used the Large Vocabulary Trick (LVT) to speed up the convergence by reducing the size of decoder's softmax layer which is a computationally inefficient operation In order to capture key features of a document, they have added additional look-up based embedding matrices for the vocabulary of each tag-type, similar to the embeddings for words [14]. In regards to handling out-of-vocabulary (OOV) words, instead of emitting UNK token, their new switching decoder/pointer architecture where if the switch is on, decoder continues to generate

words in a normal manner, otherwise, decoder generates pointer to a word present in source which is then included in the document summary. For very long documents, where not only key words but key sentences also play an important role in generating text summarization, their model assigns two bi-directional RNNs, one at word level and one at sentence level, while attention mechanism operates at both levels simultaneously.

Santhanam, S. has proposed a context based text generation model based on bi-directional LSTM networks and uses one-hot encoding for vectorized representation of words. He has proposed two methods for generating context vectors that capture the semantics of the sentence[13]. First using the word importance method, where based on the frequency of the word in a sentence and frequency of that word in the document, the importance of each word is calculated. Based on this tf-idf method, the word with highest importance is calculated and is predicted as the next word. Second, using the word clustering method to extract context-vectors to exploit two properties : semantic similarity and relationships. This model states that context of a sentence is dependent on context vector present in word vector space in such a way that semantic similarity between context vector and accumulated sentence vector is the highest [13]. This model can be further improved by using pre-trained word-embeddings such as GloVe or FastText.

## 4. PROBLEM STATEMENT

The main idea behind StockGram is to reduce the pain points of advisors who invest hours of manual effort to gain insights from latest financial news articles, reports and tweets to present to

the clients. In order to reduce this manual effort, we propose a natural language generation model that takes the latest financial articles, reports and tweets on stocks as its training data and generates a consolidated report based on the client's sector of interest.

In order to generate reports that are syntactically and semantically aligned with client's interest, Stockgram leverages bidirectional LSTM which helps the model to have a complete picture of the past and the future words around the prime word to generate a sequence of words that are relevant to the prime word. For StockGram, we also need a better vectorized representation of words that can greatly capture their semantic information and to increase the speed of training the model. Keeping these properties in mind, the model proposed in this paper is a bi-directional LSTM network that captures both the backward and forward information about the sequence at every time-step. In order to better capture the semantic information of words, I have used pre-trained word-embeddings called GloVe for a better vectorized representation of the words. This language model can be applied for text generation tasks, where, given a set of prime words, the model generates syntactically meaningful sentences.

Text generation models are usually evaluated based on whether the generated sentences are syntactically strong and whether the model is not generating duplicate sentences [13]. One such measure is Perplexity. In this paper, the proposed model is evaluated based on how semantically correct sentences are and evaluated the semantic closeness of the generated text with provided prime words.

## 5. DATASET

The proposed model was made keeping in mind to perform text generation on financial news reports related to stock news. The proposed model is expected to perform neural text generation task on an input of combined financial news on stock market from various verified sources in the form of plain text file and based on these news reports, the model generates a concise, holistic and latest report which are semantically related to words of interest for example IT sector, oil prices, DJIA etc. Since manually scraping news articles or subscribing to historical news for high quality financial news on stocks can be economically inefficient and a very tedious process, hence, I have taken a pre-scrapped financial news article, from benzinga.com, a dataset present in Kaggle. Name of the dataset is "*Daily Financial News for 6000+ Stocks*" which contains 6 columns that define : headline (article release headline, string), URL (url of the article, string), publisher (name of the person who wrote the article, string), date (date of article's release, DateTime) and stock (stock ticker , abbreviation used to uniquely identify publicly traded shares of a particular stock on a particular stock market (NYSE /NASDAQ/ AMEX only). The only column in which we are interested in is the "Article" column that contains news headlines related to latest financial news on stocks. These articles are then combined together in a simple text file, pre-processed and then fed into the model.

## 6. METHODOLOGY

### 6.1. Pre-processing input data

The first step in a text generation task is to pre-process the input data. In the proposed framework, we are using pre-scrapped Kaggle dataset and are only interested in the "Article" column of that dataset which contains financial news headlines related to stocks. We parse the entries in the "Article" column and store them in a list. Then, we open a sample text file, 'finance_news.txt' and join all the entries in our list and write them in the text file.

Now, before we proceed to feed input to the neural network, we need to pre-process the data. Here I have followed the following steps as mentioned below:

1. Remove html tags (<href></href>, <a></a>), emoticons (:heart, :laugh), punctuations ( !"#$%&')and stopwords ( "you've", "you'll", "you'd" etc)

2. Convert the pre-processed text into lowercase.

3. Perform tokenization on the input text. Tokenization is the process where given a sequence of characters tokenizing the sequence would imply chopping the sequence into pieces known as tokens and also getting rid of punctuations [15].

After we have performed the above three steps, we are almost ready for feeding the input to the model. After pre-processing the tokenized input data, we need to convert them into numbers. For this, we need to create a vocabulary-to-integer and integer-to-vocabulary dictionary. There are many ways to perform this task, but in the proposed model, I am assigning integers (indices) to words in the inverse order of their frequencies. For example if the most common term is "*the*", then word "*the*" gets the index 0 and the next most frequent word gets the index 1 and so on.

## 6.2. Batching pre-processed data

The pre-processed input data needs to be fed in the model as small batches of data instead of feeding the entire input at once. The figure 10 below defines how we can batch our input data. Say for example we have a sequence [1, 2, 3, 4, 5, 6, 7, 8, 9 , 10, 11, 12] and we want a batch size of two. In this scenario, our batching data function should split our word sequence into two separate sequences for example batched data 1 can contain [1, 2, 3, 4, 5, 6] and batched data 2 can contain [7, 8, 9, 10, 11, 12]. Then, we specify the sequence length that tells us how big we want our sequences to be. Figure 10 illustrated below shows the resultant data that is generated after batching a simple sequence and how a batch size of 2 and sequence length of 3 would look like.

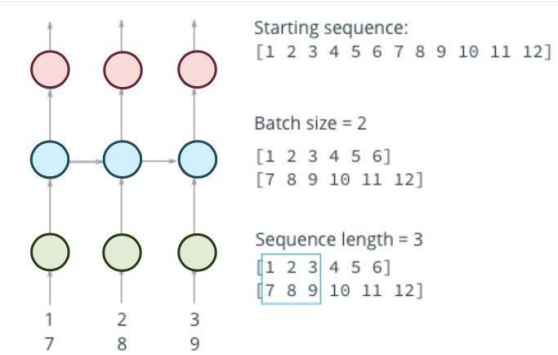


Figure 2: Batching of a sample pre-processed data

## 6.3. Model architecture

Firstly, we provide our model with a sequence of prime words. These words will give our bi-directional LSTM model, a clue regarding which word should be predicted next. For example, given the following prime words ["corona", "djia", "oil", "unemployment"], the output of the proposed model would be a matrix providing the probability for each word from the dictionary to be the next one of the given sequence [16]. Based on the news articles provided to the model, it

can predict that the next word in the sequence might be “plunged”. Now once the next word is predicted, we append it in a common list and iterate the entire process.

Now in the proposed model, the batched data is first fed through the GloVe embedding layer, where each word is converted into its vectorized representation. If the word present in an input news article is also present as a key in GloVe matrix, then we take the corresponding GloVe embedding of that word, otherwise, we randomly assign a numerical vector to that particular word. The embedding matrix is of dimension V, where V is the size of our vocabulary. After the embedding layer, the vectorized input is then passed through bi-directional LSTM with 256 hidden units and a softmax activation function for the outer layer. The learning rate for the model is kept fixed throughout the training process. I have used Adam as the optimizer and loss calculation is done on cross entropy. Model has been trained using the PyTorch library. Since this model has to predict the embedding representation of the target word, that is, predicting the next probable word in the sequence, using softmax activation function seems valid to determine

the maximum probability of the word. After training the model for ten epochs, the prediction part requires the end-user to provide two inputs. The first input is a sequence of prime words. These prime words are opening words that provide our model a seed for it to start the text generation process. One can provide any number of prime words. The next input that end-use has to provide is the number of words to generate at a time. If the number of words to be generated is n and the number of prime words provided by the user is p, then for every prime word, n words will be generated, i.e., a total of np words will be generated in the end. Once the next word is predicted, the first word from the sequence is removed and the predicted word is appended. With the newly formed input sequence, the next word is predicted. This process is iterated till the desired number of words is reached.

## 7. RESULTS

Before looking at the text generated by the proposed framework, given a set of prime words, we need to understand how the model learned over time by observing its training loss. Figure 11 shows a comparison of training loss of StockGram (bidirectional LSTM model with pre-trained word embeddings) v/s basic RNN model (unidirectional model with 2 contiguous LSTM layers and one-hot encoding) for natural language generation.

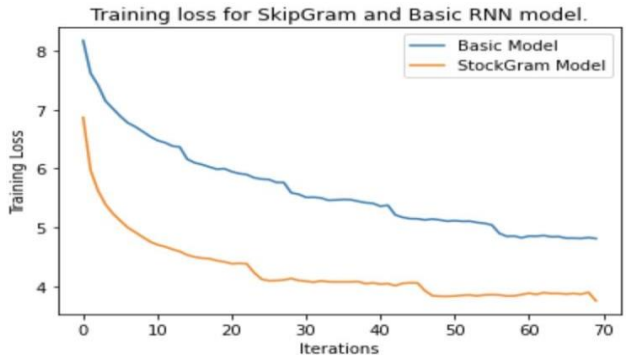


Figure 3: Comparison of training loss of basic RNN model v/s StockGram model

There is no point of taking validation dataset into account, since, for text generation models, there is no particular point where the training accuracy increases and validation accuracy decreases that the training model should be stopped. [13]. As is it clear from the training graph, the StockGram model has converged faster than the basic RNN model for text generation for the same number of

iterations. For text generation models, perplexity is a fairly good evaluation metric. It tests the syntactical integrity of the generated text. Table below shows a summarised financial report generated after experimenting with various combinations of prime words for both the basic RNN model and Stock Gram model.

Table 1. Financial reports generated by a basic RNN model (2 contiguous LSTM layers with one-hot encoding) based on various combination of prime-words

| **Prime words** | **Report generated** |
|---|---|
| coronavirus, s&p, 500, oil, prices, | ***coronavirus*** outbreak**,** which was first detected in China, ***s&p 500*** spasm 'new $180 peco ekspress brief? China international group holdings posts fy net loss of hk$11. 2 mln brief? china international group holdings posts fy net loss of hk$11. 2 mln u. s. judge says he will not be a purge of 500 furry furry furry new ceo china says it to meet u. s. tariffs over tariffs brief? china international group holdings posts fy net loss of hk$11.***oil prices*** 'new newark peco furry 'new report for first quarter earnings release and conference call on april 24, 2018 ***oil prices*** new newark peco furry 'new report for first quarter earnings release and conference call on april 24, 2018 brief? china international group holdings posts fy net loss of hk$11.brief? china international group holdings posts fy net loss of hk$11.brief? china international group holdings posts fy net loss of hk$11. |

Table 2. Financial reports generated by **StockGram** (bidirectional model with pre-trained word embeddings) based on various combination of prime-words

| **Prime words** | **Report generated** |
|---|---|
| coronavirus, pandemic, s&p, 500, oil, prices, unemployement | since ***coronavirus*** outbreak states reporting significant spikes following attempts to ease lockdown restrictions.related: bulls beware: a forecasting model model nor has been edited due to coronavirus as the ***s&p 500*** index rates caused the world's largest single day at a collapse in the globe are warning that the ***pandemic*** is nowhere near over, encouraging individuals to settle near 9. the current scenario is different, though, from the chicago board of exchange, volatility index rates, things don't implement to the ***s&p 500*** triggered demand dissipated, energy demand dissipated, and ***oil prices*** outbreak in a shorter period of time, there is not able to work as much as 30% as the ***s&p 500*** index has been expanded as planned . the magnitude of how damaged the energy industry is came into full view on april 20 when the benchmark price of us ***unemployment*** rates, it is useful to recall that the market has had been from laying off people in bulk |

By observing the reports generated by StockGram and basic RNN models, we can see that the basic RNN model tends to repeat a lot of sentences. Also, the report generated is not in accordance with the list of prime words provided. Whereas, some of the semantic content of the report generated by StockGram is in line with the prime words provided. For instance, the prime word “crude” generated phrases such as crude prices went negative for the first time in history,

prime word “covid” generated results around topic such as how covid19 is impacting oil prices negatively, topics related to how covid19 is causing lay-offs in various companies and how covid19 is affecting s&p 500 and DJIA. The StockGram model still needs more training as for some of the prime words, the reports generated by StockGram were not syntactically correct.

# 8. DISCUSSION

This paper proposes a natural language generation model, StockGram, that is developed keeping in mind to mitigate the pain points of advisors who put in hours of manual effort to scan through financial news articles in order to generate concise reports based on client's preference (prime words). In order to generate reports that are semantically and syntactically inline with prime words, StockGram utilizes bidirectional LSTM with pre-trained word embeddings. In order to evaluate the performance of StockGram, the paper compares the proposed model with a vanilla RNN model for natural language generation. Upon comparison, it is observed that StockGram generates reports that are more syntactically inline with the prime words and converges faster than basic RNN models. The proposed model can also be extended to the following areas:

1. Question-answering framework, where the question can be a sequence of prime words and based on those prime words, the model generates the answer

2. To generate a consolidated report on stock news from various verified sources based on a sequence of prime words. This will save a lot of time for advisors who go through a lot of news articles online to get a complete picture of the latest trends in the financial domain.

3. Since this model has been trained on financial news related to stocks, this model can be extended to serve as a financial assistant chatbot, where the model answers questions related to latest trends in the stock market.

## 8.2. Future Work

1. The proposed model can leverage attention based models such as Transformers instead of LSTMs.

2. More robust evaluation metrics such as BLEURT, BERTScore can be considered for StockGram.

3. Convolutional neural networks can be leveraged instead of memory based networks such as RNNs, to encapsulate dependencies between words.

## REFERENCES

[1] Evolution of Natural Language Generation. https://medium.com/sfu-cspmp/evolution-of-natural-language-generation-c5d7295d6517

[2] A comprehensive guide to natural language generation. https://medium.com/sciforce/a-comprehensive-guide-to-natural-language-generation-dd63a4b6e548

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[4] Develop bidirectional lstm sequence classification using python keras. https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/

[5] NLP 101 : Word2vec, skip gram and CBOW. https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314

[6] Negative sampling and GloVe. https://towardsdatascience.com/nlp-101-negative-sampling-and-glove-936c88f3bc68

[7] GloVe embeddings by Stanford. https://nlp.stanford.edu/projects/glove/

[8] Dipti Pawade, Avani Sakhapara, Mansi Jain, Neha Jain, Krushi Gada, "Story Scrambler – Automatic Text Generation Using Word Level RNN-LSTM", International Journal of Information Technology and Computer Science(IJITCS), Vol.10, No.6, pp.44-53, 2018. DOI: 10.5815/ijitcs.2018.06.05

[9] Luppescu, G., & Romero, F. (2017). Comparing Deep Learning and Conventional Machine Learning for Authorship Attribution and Text Generation.

[10] Yao, L., Peng, N., Weischedel, R., Knight, K., Zhao, D., & Yan, R. (2019, July). Plan-and-write: Towards better automatic storytelling. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 33, pp. 7378-7385).

[11] Semeniuta, S., Severyn, A., & Barth, E. (2017). A hybrid convolutional variational autoencoder for text generation. arXiv preprint arXiv:1702.02390 .

[12] Santhanam, S. (2020). Context based Text-generation using LSTM networks. arXiv preprint arXiv:2005.00048 .

[13] Nallapati, R., Zhou, B., Gulcehre, C., & Xiang, B. (2016). Abstractive text summarization using sequence-to-sequence rnns and beyond. arXiv preprint arXiv:1602.06023 .

[14] Tokenization.https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html

[15] Text generation using bidirectional LSTM and doc2vec models.

[16] PyTorch’s official documentation.

AUTHOR

Purva Singh, has completed her B.Tech in Computer Science and Engineering from Vellore Institute of Technology, Vellore. She has been exploring the field of Artificial Intelligence and Deep Learning for the past two years. She is passionate about Natural Language Processing, Deep Reinforcement Learning and Big Data Analytics.