COGS185 Final Project
Student: Xinyao Yi
PID: A59019592

Generate Sherlock Holmes Text using Recurrent Neural Network (RNN)

Abstract.

This study aims to explore the performance of Recurrent Neural Networks (RNNs) in text generation with the training data of the novel *Sherlock Holmes*. Three types of RNN models, including Simple RNN, LSTM, and GRU, are trained on the text data. The models generate excerpts of text based on user input. After comprehensive hyper-parameter tuning, experimental results show that LSTM performs the best among the three, achieving an accuracy of 0.82 after 50 epochs. This study provides insights into the practical performance of RNNs in text generation and highlights the effectiveness of LSTM for capturing the Sherlock Holmes writing style.

1 Introduction

Text generation has been a captivating research area, aiming to develop algorithms that can generate coherent and contextually relevant text[1]. Recurrent Neural Networks (RNN) have shown promise in addressing sequential problems due to their ability to capture long-term dependencies[2, 3]. While Artificial Neural Networks (ANN) struggle with sequential data, RNN excels in tasks involving natural language processing (NLP) and speech recognition[4]. This study aims to explore the effectiveness of RNN in generating text.

Though RNNs have been widely used in NLP tasks, their training complexity has raised concerns. However, recent advancements in optimization algorithms and network architectures have alleviated some of these challenges[5]. This study seeks to unravel the true potential of RNNs in text generation, shedding light on their performance and feasibility.

In this study, we leverage the rich text data from the *Sherlock Holmes* novels. By training the RNN models on this corpus, we aim to generate meaningful excerpts of text based on user-provided input. The generated text will be evaluated for its coherence and adherence to the Sherlock Holmes writing style.

Additionally, to explore the performance differences among RNN variants, we employ three different types of RNN models: Simple RNN, LSTM (Long Short-Term Memory), and GRU (Gated Recurrent Unit). By comparing their performance in generating Sherlock Holmes text, we seek to identify the most effective model for this specific task.

2 Method

2.1 Model: Recurrent Neural Network (RNN)

The method of this study is using Recurrent Neural Network (RNN) to train the data. RNN is a type of Artificial Neural Network (ANN) specifically used for sequential data or time-series data[6]. Compared to ANN, RNN processes the data by retaining and utilizing information from previous steps. It is a particular type which helps to do some common research including the data with sequence or time dimension, such as natural language processing, speech recognition, language translation, and time series analysis.

When coming to analyzing the sequential data, RNN can solve the problem which ANN could not regarding undefined variable size of input/output neurons and no parameter sharing. The key feature of RNN is its ability to maintain a memory which enables it to process sequential inputs. By introducing recurrent connections, RNN can allow information to flow from one step of the sequence to the next. Regarding how this process actually works, RNN takes an input and combines it with the previous hidden state to produce an output and updates its current hidden state at each step. The math function for RNN can be represented as follows:

$$h_t = f(W_x \cdot x_t + W_h \cdot h_{t-1} + b)$$

In this function, h_t is the hidden state at time step t, x_t is the input at time step t, f is the activation function, W_x and W_h are weight matrices, and b is the bias term.

By processing the inputs sequentially, RNN can learn to capture dependencies and patterns in the data, making it suitable for tasks that require context or temporal information.

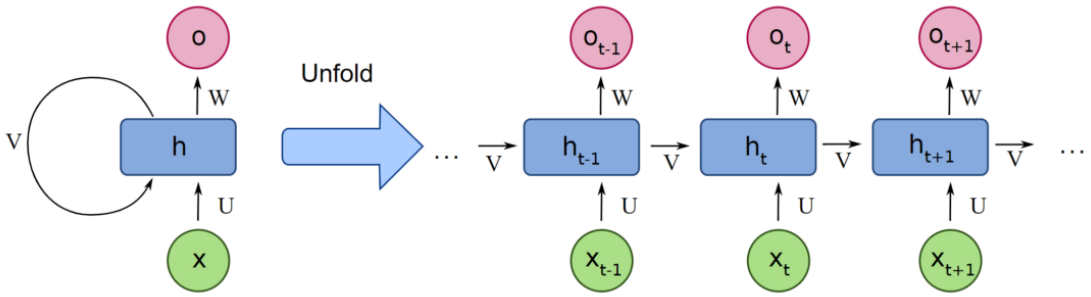


Fig 1. A visualization of RNN architecture

2.2 RNN Achitecture Types

2.2.1 Simple RNN

Simple RNN is a basic type of recurrent neural network architecture. It is the simplest form of RNN and is primarily used for modeling sequential data. The achitecture of Simple RNN is the same as RNN above.

Though Simple RNN is handy to perform, the main issue it suffers from is the vanishing/exploding gradient problem. Due to the repeated multiplication of weight matrices in the recurrent connection, gradients can either vanish or explode as they propagate backward during training. This limits the SimpleRNN's ability to capture long-range dependencies in sequences as it could not hold the relevance of information from distant time steps.

To address the vanishing/exploding gradient problem, more sophisticated RNN variants like LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) have been developed.

2.2.2 Long Short-Term Memory (LSTM)

LSTM is a type of neural network that can understand and process sequences of data, such as text or time series. It can address the problem of capturing long-term dependencies that traditional neural networks struggle with.

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. A cell can remember the values. An input gate can decide which pieces of new information to be stored in the current state. An output gate can control which pieces of information to be displayed. A forget gate can decide which type of information to be discarded. Making predictions by outputting selectively information from the current state is necessary and practical for those research with time series, both in current and future time-steps.

In brief, with the three gates implemented, LSTM is able to selectively store, forget, and output information based on the context and relevance. This structure enables LSTM to capture long-term dependencies in sequences. In this way, LSTM is an ideal method to be applied in NLP, speech recognition, time series analysis, and even music composing the video analysis.

2.2.3 Gated Recurrent Unit (GRU)

GRU is another type of neural network that is also designed to solve the vanishing/exploding gradient problem. It has two gates in its structure: the reset gate and the update gate. The reset gate determines how much of the previous hidden state information should be forgotten. The update gate controls how much of the new information should be incorporated into the current hidden state. The architecture and computing process of these two gates allows GRU to capture long-term dependencies in sequences, remembering important details from the past and adapting to new input. Though GRU is a simpler version of LSTM, it still performs well in tasks that require understanding and modeling of sequential data.

In this study, the data will be respectively trained on Simple RNN, LSTM and GRU to see which architecture performs better. Within the architecture, other hyper-parameter tuning will also be performed.

2.3 Regularization Techniques

2.3.1 Dropout

Dropout is a regularization technique used in neural networks to prevent overfitting. It drops out neurons to prevent the network from relying heavily on specific neurons or complex interdependencies between them. In this way, the network is forced to learn more robust features.

Here is the process of how Dropout works. During each training iteration, a fraction of neurons in a certain layer is randomly selected and dropped out. The dropped-out neurons are temporarily removed from the network for that iteration. Their outputs are set to zero, and their connections are ignored during both the forward pass and the backward pass.

2.3.2 Batch Normalization

Batch normalization is a technique used in neural networks to standardize the inputs of each layer. It aims to improve the training process and the overall performance of the network by normalizing the activations within mini-batches of training data.

Here is the process of how Batch Normalization works. During the forward pass of training, the activations of a particular layer are computed for each mini-batch. The mean and standard deviation of the activations within the mini-batch are calculated, and then activations are normalized by subtracting the batch mean and dividing by the batch standard deviation.

Batch Normalization is useful in improving the training speed, providing better generalization and increasing robustness to hyperparameters and network architecture.

In this study, both techniques of dropout and batch normalization are used in model construction to prevent overfitting as well as making the network architecture more robust. The rate of a dropout layer was set to be 0.1.

2.4 Optimization Methods

2.4.1 Stocahstic Gradient Descent (SGD)

SGD is a widely used optimization algorithm for training machine learning models, including neural networks. The difference between SGD and gradient descent is that SGD only takes small random chunks of the training data to teach the model how to improve. More specifically, SGD makes the parameters of

the network update based on the gradients of the loss function only with respect to a small randomly selected subset of training examples (i.e., a mini-batch).

However, though SGD is efficient and flexible, the optimization process can fluctuate, since the use of mini-batches could introduce noise in parameter updates, and also, the performance of SGD depends a lot on learning rate selection.

2.4.2 Adaptive Moment Estimation (Adam)

Adam is an optimization algorithm commonly used in neural networks. It adjusts the learning rate for each parameter individually based on the estimated first and second moments. It adapts the learning rate based on the magnitude of the gradients, providing larger updates for infrequent and important features and smaller updates for frequent and less important features. The main strength of Adam lies in offering adaptive learning rates that efficiently update model parameters, leading to faster convergence and improved optimization performance. Adam could perform well on a wide range of problems and often converges faster than traditional optimization algorithms and is becoming more and more popular as an optimization tool in neural networks.

3 Experiments

3.1 Data and Research Purpose

This data is the text data of the novel *Sherlock Holmes* composed by Arthur Conan Doyle. The whole content is available here[2]. It is of the English language and contains 576,452 words in all. Since it is a relatively large text data set, it will consume more time while training.

To pre-process the data, a vocabulary of unique characters was first created from the text. In this text, we had 91 unique characters in all. All of them were then transformed to numerical forms. In this way, the whole dataset was transformed into a numerical dataset and was stored in a NumPy array. To prepare the data for training, the dataset was first divided into sequences of a specified length of 100. Then to improve the randomness of the training data, the dataset was shuffled with a buffer size of 10000. Finally, the dataset was batched with a batch size of 64 and stored as the final dataset for training.

The purpose of this study is to train a text generation model using RNN architectures. The model is designed to generate a piece of “Sherlock Holmes” story text based on an input string given by the user.

The study aims to explore and compare the accuracy and effectiveness of the chosen architectures and training parameters in generating coherent and meaningful text output.

3.2 Model Implementation/Achitecture

In this study, RNN models are implemented using the Tensorflow API. Three architectures under RNN - Simple RNN, LSTM, GRU were implemented respectively on the data set by using the tf.keras library.

The models follow the main structure as follows. The first layer is an Embedding layer that maps the vocabulary size to an embedding dimension of 256. This layer converts each word in the text into a dense vector representation. The hidden layer(s) in the middle is(are) of Simple RNN or LSTM or GRU architecture with certain units, and regularization layers are tuned to be added or not. The final layer was completed with a Dense layer that has the same size as the vocabulary. The first Embedding layer and the final Dense layer are the same in all models, but the hidden layers and regularization layers in the middle vary. Below is the detailed information of how the middle layers were constructed.

In the Simple RNN architecture, 2 models in all were performed. Model 1 contained 2 layers of Simple RNN, with regularization layers dropout and batch normalization respectively added after each Simple RNN layer. The optimization method was Adam optimization, and the epoch iteration number was set to 50. Model 2 had the same architecture as Model 1 except for the epoch iteration was set to 25.

In the LSTM architecture, 7 models in all were performed. Model 1 contained 2 layers of LSTM, with regularization layers dropout and batch normalization respectively added after each LSTM layer. The optimization method was Adam optimization, and the epochs iterated was set to 50. Model 2 was quite different from Model 1: only 1 layer of LSTM was implemented in this model, and no regularization layers were added. The optimization method was still Adam and the epoch was set to 50. Model 3 contained 1 LSTM layer, regularization methods (dropout and batch normalization) were included - where different from Model 2, with Adam optimization and epoch was set to 50. Model 4 contained 1 LSTM layer, regularization methods (dropout and batch normalization) were included, SGD optimization - where different from Model 3, epoch was set to 50. Model 5 contained 1 LSTM layer, regularization methods (dropout and batch normalization) were included, with Adam optimization but the epoch number was set to 25. Model 6 was of the same structure as Model 1 (i.e. 2 LSTM layers, regularization included, Adam optimization), but the epoch was set to 75. Model 7 was also of the same structure as Model 1, but the units in its two LSTM layers were set to (1024, 512) compared to (1024, 1024). To mention, LSTM has a default embedded activation function architecture within itself (input gate activation: sigmoid; forget gate activation: sigmoid; output gate activation: sigmoid; candidate cell state activation: tanh), so in this study, the ativation function of LSTM was not tuned.

In the GRU architecture, 2 models in all were performed. Model 1 contained 2 layers of GRU, with regularization layers dropout and batch normalization respectively added after each GRU layer. The optimization method was Adam optimization, and the epoch iteration number was set to 50. Model 2 had the same architecture except for the epoch number was set to 25. To mention, GRU has a default embedded activation function architecture within itself as well (reset gate activation: sigmoid; update gate

activation: sigmoid; new memory content activation: tanh), so in this study, the ativation function of GRU was not tuned as well.

Table 1. Model Achitectures with hyper-parameter tuning

	Model1	Model2	Model3	Model4	Model5	Model6	Model7	Model8	Model9	Model10	Model11
Achitecture	Simple RNN	Simple RNN	LSTM	LSTM	LSTM	LSTM	LSTM	LSTM	LSTM	GRU	GRU
Layers	2	2	2	1	1	1	1	2	2	2	2
Units	1024, 1024	1024, 1024	1024, 1024	1024	1024	1024	1024	1024, 1024	1024, 512	1024, 1024	1024, 1024
Regularization	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	✓
Optimizer	Adam	Adam	Adam	Adam	Adam	SGD	Adam	Adam	Adam	Adam	Adam
Epoch	50	25	50	50	50	50	25	75	50	50	25

4 Results

4.1 Simple RNN

The Simple RNN models showed a relatively low accuracy (about 0.61) and a relatively long training time and also a sign of overfitting even with the regularization techniques applied and the training object of complex models (two layers), which indicates Simple RNN may not be a good fit in the task of this study.

Table 2. Model Comparison Among Simple RNN models

	Model 1	Model 2
Total Parms	3534683	3534683
Loss	1.2684	1.2357
Accuracy	0.6156	0.6260

Overfitting	✓	✓
Training Time (s)	5119.560	2491.659

4.2 LSTM

LSTM models showed higher accuracies than the Simple RNN models overall. With two layers of LSTM and the layer units of (1024, 1024), the accuracies were all above 0.82, which was considered as decent. The accuracies of models with one layer of LSTM and the Adam optimization were all above 0.75, which was considered as acceptable. Overall, the LSTM models showed a great performance.

When we look into details of the LSTM models, we can see that for the models of two layers, the accuracies were higher than the models with only one layer, but the training time was longer (almost two times).

We can also infer some information about other hyper-parameter tuning skills such as optimization methods, regularizaiton techniques and epochs selection from this table. When comparing the one layer models, we can see that the one with the SGD optimization (Model 6) method performed worst among all and was far worse than the others with the accuracy lower than 0.4, so in this case, SGD was not considered as a good optimization choice as Adam. Also, the model without the regularizaiton techniques (i.e., dropout and batch normalization) (Model 4) showed a sign of overfit which the ones implementated with regularization techniques did not show, and a 0.005 higher accuracy than its comparable model (Model 5), which indicates the regularization technique dropout can help with preventing overfitting and the technique batch normalization can help the model hold a more robust architecture. The one layer model model with a small epoch number of 25 (Model 7) did not show a significant accuracy decrease than its comparable model (Model 5). The accuracy of Model 7 is 0.01 lower than Model 5, which is acceptable with the huge training time decreasing to 430 seconds (half time of Model 5). However, the result that the decrease in epoch number did not significantly impact the accuracy could be due to the simplicity of the one-layer model so it is hard to come to a conclusion of the epoch number at this point.

When comparing the two layers models, we can see that an increase in epoch number (50 to 75) (Model 3 and Model 8) did increase the accuracy regarding the two LSTM layers models by 0.02, but the training time also increased by ⅓ time. The training process of the two layer LSTM model with 75 epochs was really time-consuming and computationally demanding. Additionall, the study added another parameter tuning in the unit number of each layer. Model 9 is a model with 1024 neuron units in its first LSTM layer and 512 units in its second layer. The accuracy decreased by 0.04 compared to the one with the unit structure of (1024, 1024) (Model 3), but the training time also decreased by 305 seconds.

In conclusion ,the Adam optimization outperformed the SGD, and the regularizaiton techniques dropout and batch normalizaiton effectively prevented overfitting and made the model more robust. As was discussed before, the increase in layer number could lead to a higher accuracy but it was more time-consuming and computationally-consuming, so there was a tradeoff regarding how many layers to be implemented. The increase in epochs within complex model structures (in this case two LSTM layers) also led to a higher accuracy but faced the same issue of more time consumed. The decrease of neuron units in the layer could lead to a bit decrease in accuracy but some of time and computational power was saved.

Table 3. Model Comparison Among LSTM models

	Model 3	Model 4	Model 5	Model 6	Model 7	Model 8	Model 9
Total Parms	13764443	5363547	5367643	5367643	5367643	13764443	8470875
Loss	0.5610	0.7460	0.7516	2.0853	0.7945	0.5013	0.6864
Accuracy	0.8289	0.7713	0.7677	0.3959	0.7546	0.8485	0.7854
Overfitting	×	✓	×	×	×	×	×
Training Time (s)	1863.383	811.365	863.473	857.460	430.735	2789.040	1360.409

4.3 GRU

The GRU models also showed lower accuracy than the LSTM models in this task. They also showed a sign of overfitting even when the epoch was chosen to be 25 and regularization techniques were applied. Though they may have a shorter time in the training process, the low accuracy made them not as a good fit as LSTM models.

Table 4. Model Comparison Among GRU models

	Model 10	Model 11
Total Parms	10360667	10360667
Loss	1.2748	1.0304
Accuracy	0.6137	0.6801

Overfitting	✓	✓
Training Time (s)	1570.835	799.420

4.4 Model Comparison

We choose three models respectively from the Simple RNN structure, the LSTM structure and the GRU structure with the architecture of two layers with 1024 units in each layer, with regularization techniques implemented, Adam optimization, and 50 epochs.

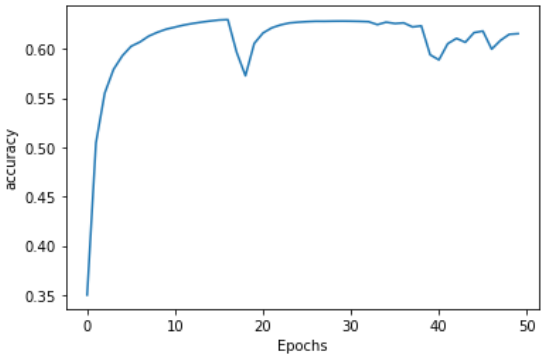
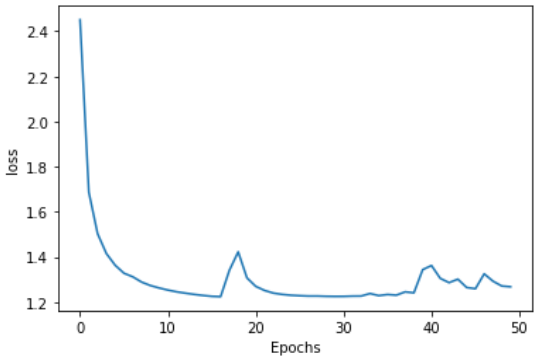
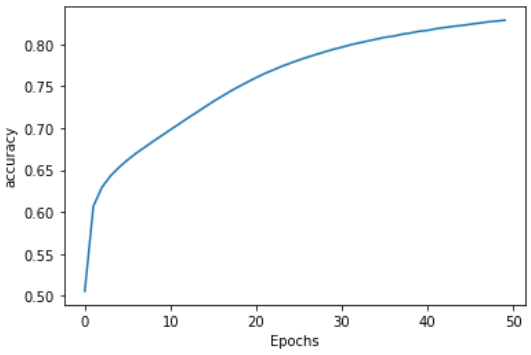
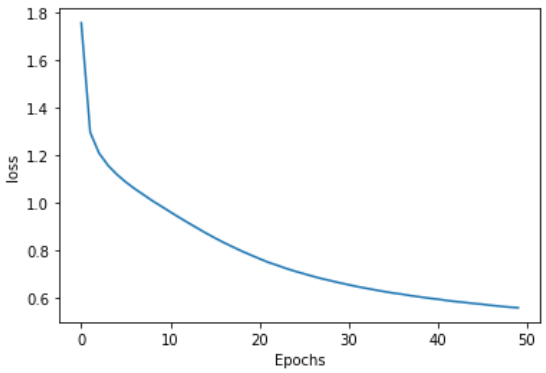
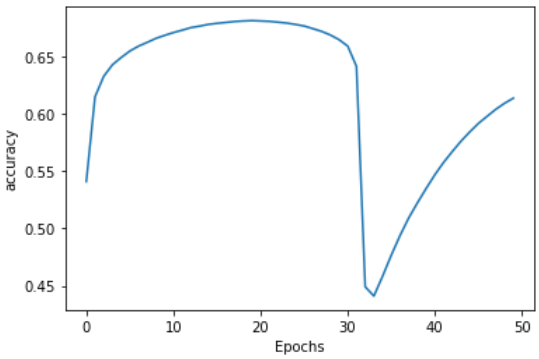
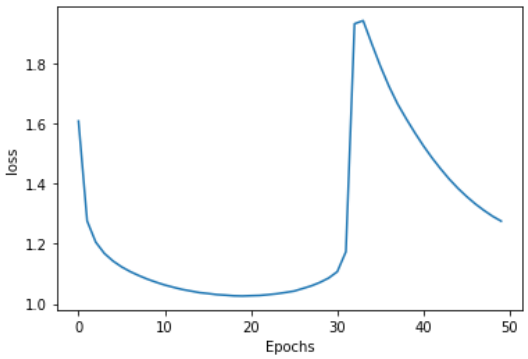
As we can see from the result, the LSTM model performs best in this text generation task with a higher accuracy of 0.82 and no overfitting and an acceptable training time.

Table 5. Model Comparison Among three RNN architectures

	Model 1	Model 3	Model 10
Total Parms	3534683	13764443	10360667
Loss	1.2684	0.5610	1.2748
Accuracy	0.6156	0.8289	0.6137
Overfitting	✓	×	✓
Training Time (s)	5119.560	1863.383	1570.835

4.5 Visualization

Table 6. Accuracy Plot and Loss Plot of Model Comparison

	Accuracy Plot	Loss Plot
Simple RNN (Model 1)		
LSTM (Model 3)		
GRU (Model 10)		

4.6 Text Generalization Review

The text generated by Model 3 (LSTM, 2 layers, Adam Optimization, 50 epochs):

Instruction:

Write the beginning of the text, the program will complete it. Your input is: *There is a line on the letter saying the object he saw was so horrible and he could not even figure out what it was……*

Text Generated (1000 characters):

There is a line on the letter saying the object he saw was so horrible and he could not even figure out what it was for the servant's things to be distunbed.

"What should I know the matter of the Hall," said he, "but at that moment, Watson."

"What does it show us in some ways?"

"There are two faces was moving about the old hall before evil gives us a chance of their matters."

Our visitor staggered.

"Yes," said McMurdo.

"I stood with you as large and as resolute as seaman. I can see him more flew long enough. From acideto to carry out your order, and your attention to assure that she had resented for some time, but from the direction into the lower part of the bare side of the table. passing itself on the steps instantly with the utmost a time when we found ourselves in Paris and bitterly?"

"No, I had a struck as we sat together in the Park. The great Grimpen Mire lay a long, and absolutely dangerous title."

"I know well," said Holmes, glancing with anger, as he was very angry. "I have a narrow, tanishe

(Run time: 16.89352560043335)

5 Conclusion

In conclusion, by comparing the three architectures under the RNN - Simple RNN, LSTM and GRU, LSTM outperforms the other two in this text generation task with a higher accuracy of 0.82, no signs of overfitting and an acceptable training time.

Additionally, with regard to hyper-parameters selection, Adam optimization outperformed the SGD; regularizaiton techniques dropout and batch normalizaiton effectively prevented overfitting and made the model more robust. An increase in layer number could lead to a higher accuracy but it was more time-consuming and computationally-consuming, so there was a tradeoff regarding how many layers to be implemented. The increase in epochs within complex model structures also led to a higher accuracy but faced the same issue of more time consumed. The decrease of neuron units in the layer could lead to a bit decrease in accuracy but some of time and computational power was saved.

References

Celikyilmaz, A., Clark, E., & Gao, J. (2020). Evaluation of text generation: A survey. arXiv preprint arXiv:2006.14799. (n.d.).

Lin, T., Horne, B. G., & Giles, C. L. (1998). How embedded memory in recurrent neural network architectures helps learning long-term temporal dependencies. Neural Networks, 11(5), 861-868. (n.d.).

Lu S, Zhu Y, Zhang W, Wang J, Yu Y. Neural text generation: Past, present and beyond. arXiv preprint arXiv:1803.07133. 2018 Mar 15. (n.d.).

Salehinejad, H., Sankar, S., Barfett, J., Colak, E., & Valaee, S. (2017). Recent advances in recurrent neural networks. arXiv preprint arXiv:1801.01078. (n.d.).

Siami-Namini, S., Tavakoli, N., & Namin, A. S. (2018, December). A comparison of ARIMA and LSTM in forecasting time series. In 2018 17th IEEE international conference on machine learning and applications (ICMLA) (pp. 1394-1401). IEEE. (n.d.).

Tarwani, K. M., & Edem, S. (2017). Survey on recurrent neural network in natural language processing. Int. J. Eng. Trends Technol, 48(6), 301-304. (n.d.).