INSA Summer Research Fellowship Program
2023
4 Weeks Report

Application number:	ENGS2204
Summer Research Fellow:	Srinithi S L
Personal Email :	slsrinithi1912@gmail.com
College Email:	107121104@nitt.edu
Date Performed:	May 23, 2023 to July 27, 2023
Guide:	Professor R. Ganesh, IPR-Gandhinagar

Acknowledgements

I am deeply grateful to Dr. Pawandeep Kaur, a Postdoctoral fellow at the Weizmann Institute of Science, Israel, for her valuable inputs, expert advice, and constructive feedback. Her mentorship and scholarly contributions have significantly enriched this report.

Turbulence prediction with Recurrent Neural Networks

July 27, 2023

1 Introduction

Turbulence modeling and prediction aims to develop more accurate and efficient techniques for understanding and forecasting turbulence. By gaining a deeper understanding of the underlying dynamics and structures of turbulent flows, researchers can improve the design and optimization of various engineering systems. In the context of the specific research paper titled "Recurrent neural networks and Koopman-based frameworks for temporal predictions in a low-order model of turbulence," **Eivazi et al. 2021**, the authors investigate the application of recurrent neural networks and Koopman-based frameworks for predicting the temporal behavior of turbulence in a low-order model. In the research project, LSTM and RNNs play a significant role in capturing the temporal dynamics of turbulent flows. By leveraging their ability to model sequential data and capture long-term dependencies, LSTM and RNNs enhance the accuracy and efficiency of turbulence predictions, offering valuable insights into turbulence evolution and enabling better control of turbulent flow systems.

2 Predictions with LSTM

The most basic form of neural network is the multilayer perceptron (MLP), which consists of interconnected layers of nodes or neurons. MLPs are commonly used in practice, but their main drawback is that they are designed for point prediction rather than time-series prediction, which requires a more context-aware approach. However, MLPs serve as a solid benchmark in machine learning applications and help highlight the need for more sophisticated network architectures. In a previous study (Srinivasan et al., 2019), the accuracy of MLP predictions was evaluated using the nine-equation model by Moehlis et al. (2004), where various network architectures were used to predict the time evolution of the nine coefficients. RNN(Figure 1) is a type of neural network designed to process sequential data, making it well-suited for time series forecasting. It utilizes feedback connections that allow information to be passed from one time step to the next, enabling the network to capture temporal dependencies and patterns in the data.

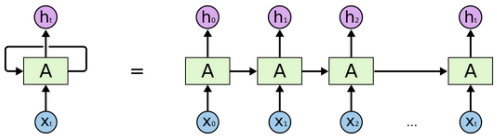

Figure 1: Recurrent Neural Networks. Figure source: **colah2015**.

LSTM (Long Short-Term Memory) (Figure 2) is a type of recurrent neural network (RNN) architecture that is specifically designed to address the vanishing gradient problem and capture long-term dependencies in sequential data. LSTM networks consist of memory cells that store information over time, along with gating mechanisms that control the flow of information into, out of, and within the cells.

The key components of an LSTM cell are:
1. Input Gate (i_t): The input gate controls how much new information is allowed to enter the memory cell. It selectively updates the cell state based on the input and the previous cell state.
2. Forget Gate (f_t): The forget gate controls the amount of old information to forget from the memory cell. It decides which information is no longer relevant and should be discarded.
3. Output Gate (o_t): The output gate regulates the flow of information from the memory cell to the output of the LSTM unit. It determines how much of the current cell state should be exposed as the output.
4. Cell State (c_t): The cell state is responsible for storing and propagating information over time. It acts as the long-term memory of the LSTM unit, allowing the network to capture and remember important patterns or dependencies in the sequential data.
5. Hidden State (h_t): The hidden state is the output of the LSTM cell. It contains information about the current time step and is used for making predictions or passing information to subsequent LSTM cells in a sequence.

Mathematically, the LSTM cell equations can be represented as follows:
1. Input Gate Equation:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{1}$$

2. Forget Gate Equation:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2}$$

3. Output Gate Equation:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3}$$

4. Cell State Update Equation:

$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{4}$$

5. Cell State Equation:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \tag{5}$$

6. Hidden State Equation:

$$h_t = o_t \cdot \tanh(c_t) \tag{6}$$

In these equations, x_t represents the input at time step t, h_{t-1} represents the hidden state at the previous time step, and σ and tanh are the sigmoid and hyperbolic tangent activation functions, respectively. W_i, W_f, W_o, W_c and b_i, b_f, b_o, b_c are learnable parameters (weights and biases) of the LSTM cell.

These equations, with their respective equation numbers, allow the LSTM cell to perform computations on the input sequence, update its internal state (cell state), and pass information to the next time step, enabling the network to capture long-term dependencies in the data.

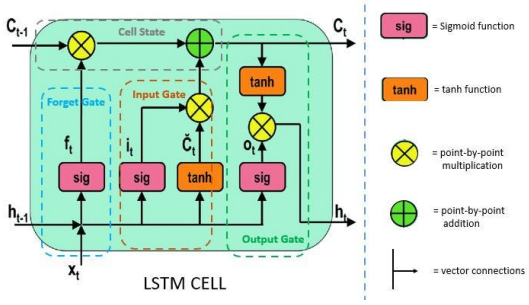

Figure 2: Long Short Term Memory cell. Source: **analyticsvidhya2022**.

3 The Setup

Using python, 9 coupled nonlinear Ordinary Differential Equations (Eq.1 of Srinivasan et al. 2019 paper) are solved to generate a single time series of 4000 time steps and 9 coefficients. To train and evaluate a predictive model, the dataset is split into training and testing sets, with 80% of the data used for training and 20% for testing. For the model to make predictions on this time series data, the TensorFlow library (**TF**) provides a useful function called "TimeseriesGenerator." This function employs the windowing method, which generates input-output pairs from the time series data. The windowing method creates a sliding window of fixed length and slides it along the time series, generating subsequences of data as inputs and corresponding targets. In this project, we employ data windowing techniques to facilitate both single time step prediction and multiple time step predictions. Specifically, we use a window size of 500 previous time steps as input to the model. For single time step prediction, the model predicts the value of the next time step. For multiple time step prediction, the model forecasts the values for the subsequent 32 time steps. These predictions are made by sliding the window along the time series data and generating input-output pairs for training and evaluation.

4 Model Architecture

The model architecture (Figure 3) consists of three layers: one input layer, one LSTM layer, and one output layer. Here is a detailed explanation of each layer used in our project:

a. The *Input Layer* has 9 input nodes, which corresponds to the 9 coefficients of the time series data. The input layer receives a sequence of 500 previous time steps (datapoints) as input, where each time step consists of the 9 coefficients.

b. The *LSTM Layer* consists of 90 LSTM units or cells. Each LSTM unit is responsible for capturing and processing the temporal dependencies within the sequence of 500 previous time steps. The LSTM layer's architecture allows it to remember and utilize information from earlier time steps, capturing the long-term dependencies in the data. The LSTM layer performs computations on the input sequence, updates its internal state, and passes information to the next time step.

c. The *Output Layer* The output layer has 9 nodes, corresponding to the 9 coefficients being predicted. This layer takes the output from the LSTM layer and generates predictions for the next time step values of the 9 coefficients. Each node in the output layer represents a specific coefficient, and the values produced by these nodes are the predicted values for the respective coefficients.

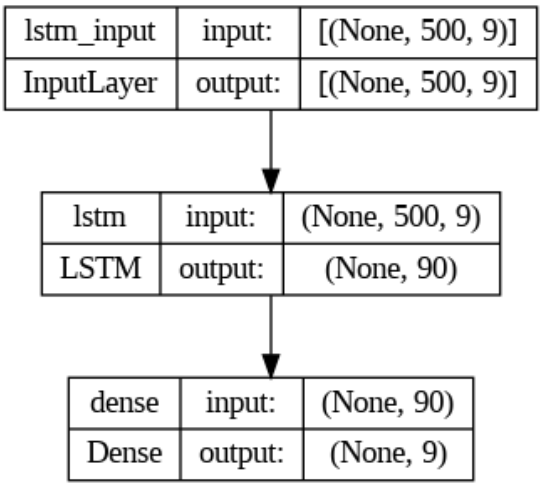

Figure 3: Model Architecture.

The total number of trainable parameters in the model architecture is the sum of the parameters in each layer and equals to 36819, where the number of trainable parameters in LSTM layer and Output Layer are 36000 and 819 respectively.

In the *LSTM layer*: The input-to-hidden connections have 9 input nodes and 90 LSTM units, with each connection having 4 parameters (weight and bias for input, forget, output, and cell state). The hidden-to-hidden connections have 90 LSTM units and each connection has 4 parameters.

$$\text{Total parameters in the LSTM layer} = 9\times90\times4+90\times90\times4 = 36000 \tag{7}$$

In the *Output layer*: The LSTM layer has 90 LSTM units connected to 9 output nodes. Each connection has 1 parameter. Additionally, each output node has its own bias parameter.

$$\text{Total parameters in the Output layer} = 90 \times 9 + 9 = 819 \tag{8}$$

Therefore, the correct total number of trainable parameters in the model is:

$$\text{Total number of trainable parameters} = 36000 + 819 = 36819 \tag{9}$$

The specific model architecture of a single layer LSTM is chosen for turbulence prediction due to its capability to handle sequential time series data, capture long-term dependencies, and facilitate efficient training with a manageable number of parameters. The single layer design strikes a balance between model complexity and computational efficiency, making it suitable for the available dataset size and computational resources. Additionally, the LSTM's memory cells and gating mechanisms enable the model to effectively learn temporal patterns, leading to accurate turbulence predictions.

5 Results and Analysis

The model's training process involves minimizing the mean squared error loss (see Eq.10) during each training iteration. The model's weights are adjusted using the Adam optimizer, which calculates the gradients based on the loss function and updates the model parameters to minimize the loss (figure 4 & figure 5). The training progresses through the specified number of epochs, with the aim of optimizing the model's performance and reducing the mean squared error between the predicted and actual values of the time series. The training process employs the Mean Absolute Error (MAE) see Eq.11) metric to gauge the model's performance. The early stopping callback compares the validation loss between epochs and stops training if the loss does not improve for the specified number of epochs (patience). Early stopping helps prevent over-fitting and improves training efficiency by stopping the process when further training does not lead to significant improvements in the validation loss.

$$\text{Mean Squared Error} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{10}$$

$$\text{Mean Absolute Error} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{11}$$

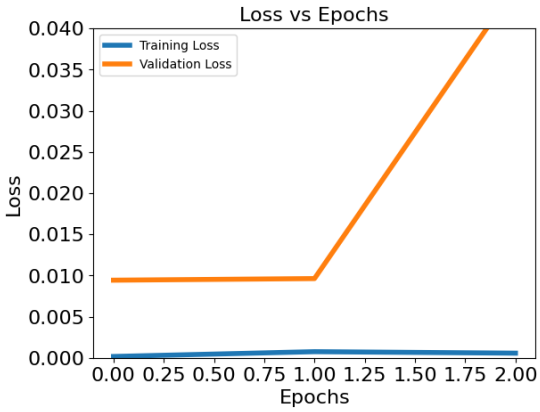

Figure 4: Continuous Error Reduction of the Single Time-Step Model with Epoch.

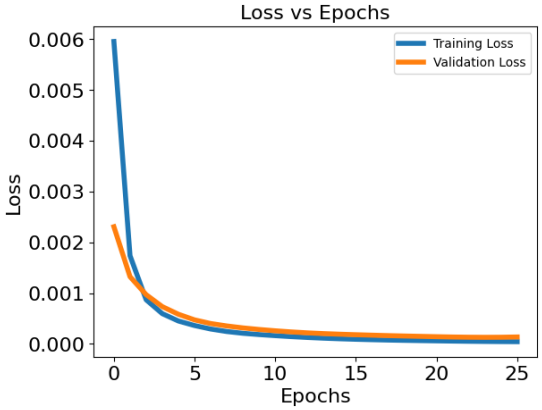

Figure 5: Continuous Error Reduction of the Multistep Time-Step Model with Epoch.

When the validation loss for the single-step model increases while the training loss keeps (Figure 4), it is a clear indication of overfitting. Overfitting occurs when a model becomes too specialized in learning the details of the training data and fails to generalize well to unseen data, such as the validation or test data. This behavior implies that the single-step model is learning the training data very well, but it fails to capture the underlying patterns that apply to the test data or unseen data. One way to address this overfitting issue is by employing ensemble methods or multiple time series data. Ensemble methods involve creating multiple models with different configurations and averaging their predictions to obtain a final result. This helps to reduce the variance and improve the overall generalization of the model.

The trained models for both the single-step and multi-step predictions are evaluated using the test data, and the error metrics are obtained: mean squared error (MSE)

and mean absolute error (MAE) (Table 1).

Prediction Models	MSE(%)	MAE(%)
Multi time-step	0.014	0.833
Single time-step	6.257	19.192

Table 1: Evaluation of Trained Models for Single-Step and Multi-Step Predictions using Test Data

Graphs are generated to display the predicted values for the nine coefficients in the test dataset. The graphs illustrate the forecasted time series data and include predictions for both single-step and multi-step predictions. A total of 300 timesteps are covered in the plots.

Single-step predictions and multi-step predictions are two different approaches used in time series forecasting.

a. *Single-step predictions:* In this approach, the model is trained to predict the value of the next time step given the input sequence of previous time steps. The model takes a fixed window of historical data as input and predicts the value at the very next time step. After making the prediction, the window slides by one time step, and the process is repeated for the next prediction. This method is suitable when we are interested in short-term predictions and immediate future values.

b. *Multi-step predictions:* Multi-step predictions involve forecasting multiple future time steps ahead of the current time step. Instead of predicting just the next value, the model is trained to forecast a sequence of values into the future. The input sequence contains a fixed window of historical data, and the model predicts the values for several future time steps in a single pass. This approach is useful for long-term forecasting, where we want to understand the trends and behavior of the time series over an extended period.

From the plots(Figure 6 & Figure 7), it is observed that multi time step predictions (Figure 7) provide a more comprehensive representation of the future trajectory of the time series compared to single time step predictions (Figure 6). The multi time step predictions exhibit a smoother and more coherent pattern over the forecasted period, capturing the underlying trends and dynamics of the data. In contrast, single time step predictions may exhibit more volatility and fluctuations as they only focus on one-step ahead forecasts. The multi time step predictions showcase a better understanding of the temporal dependencies and correlations within the time series, resulting in more accurate and reliable forecasts.

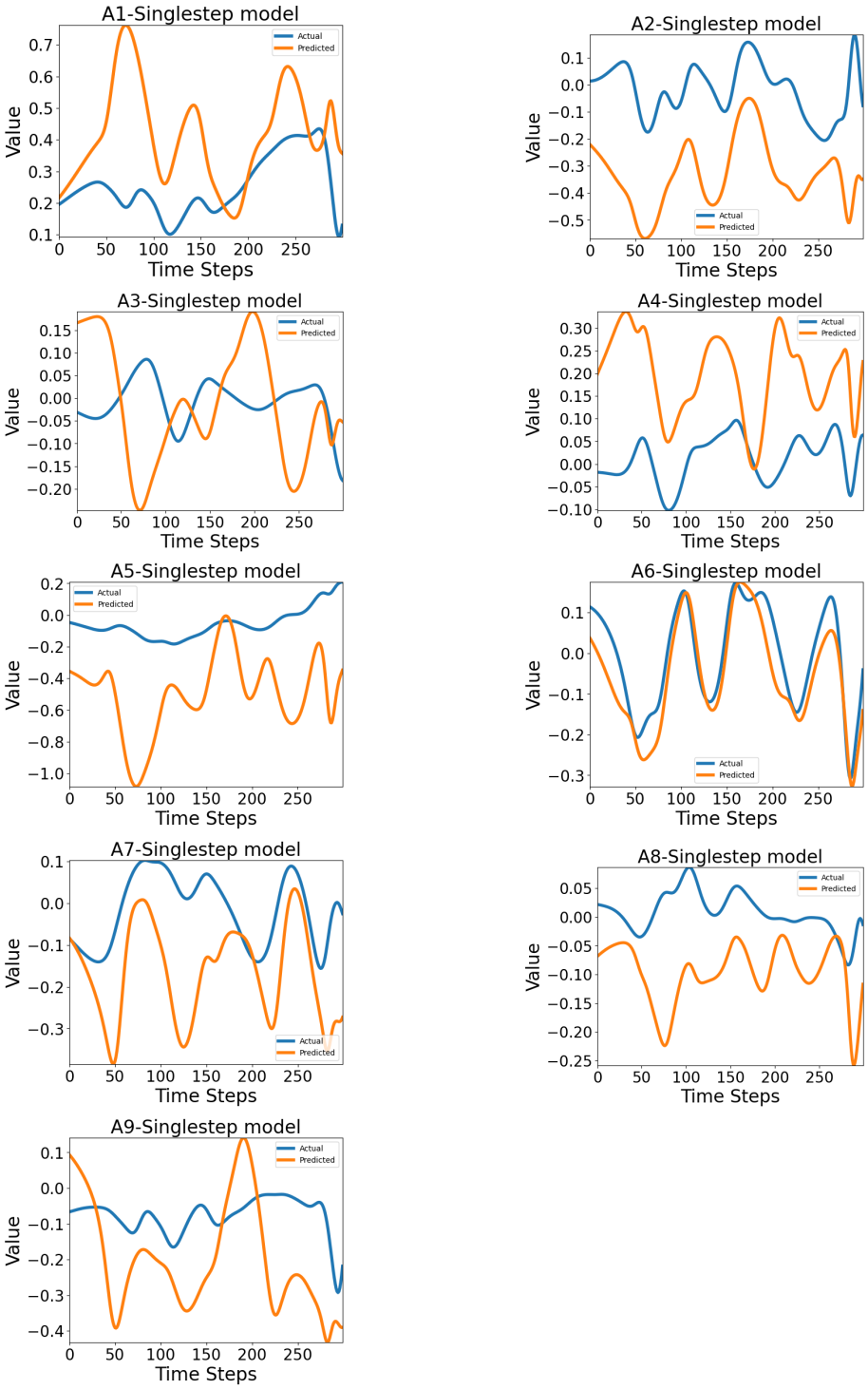

Figure 6: Single time-step predictions for each of 9 coeffients

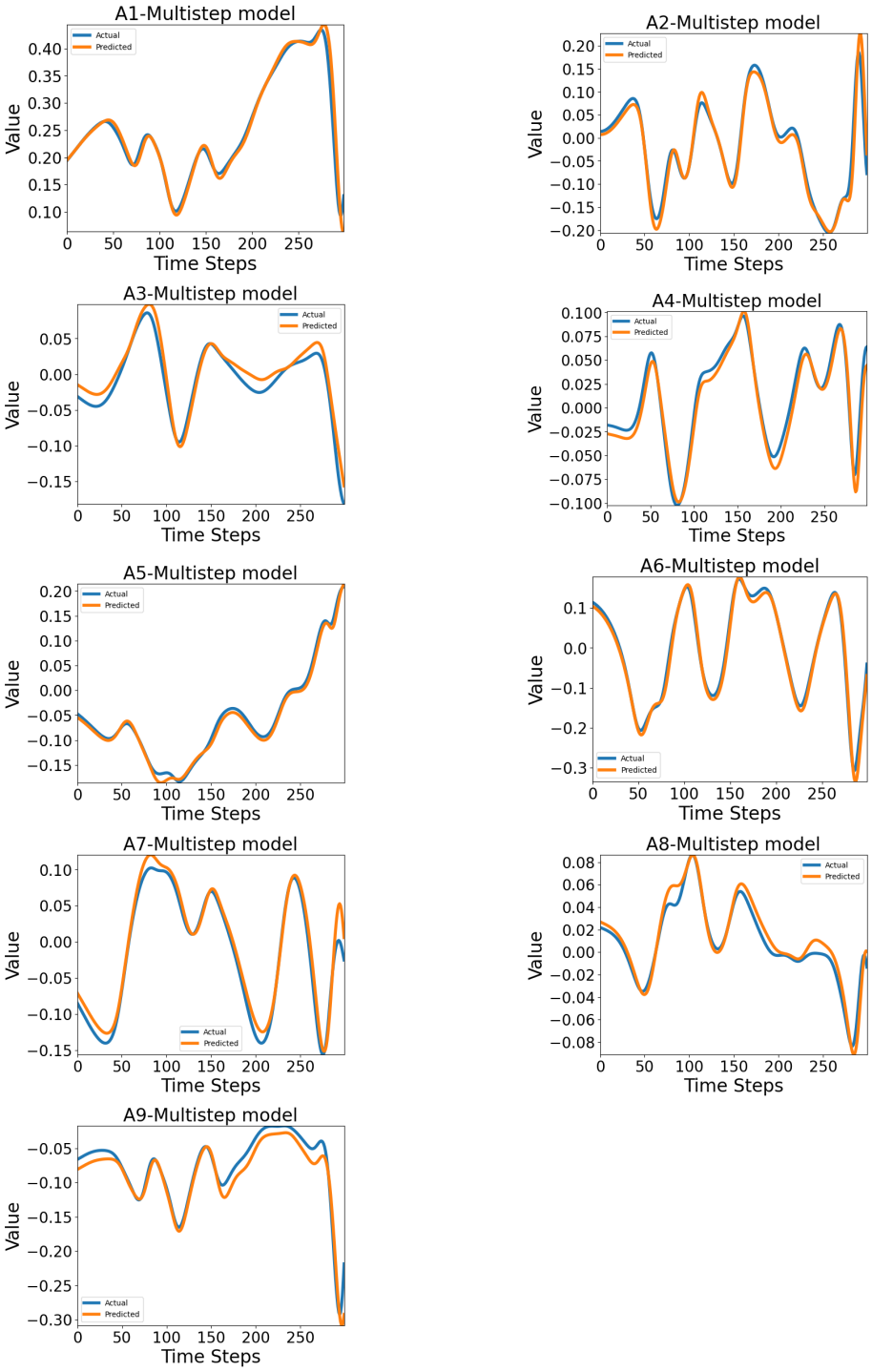

Figure 7: Multi time-step predictions for each of 9 coefficients.

6 Plan for upcoming weeks

- Focus on improving the accuracy and robustness of time series predictions.
- Utilize ensembled averaging techniques to enhance prediction performance.
- Create multiple time series models with different configurations and architectures.
- Train each model on the same dataset with different initializations or hyperparameters.
- Perform ensembled averaging to combine predictions from individual models.
- Analyze experiment results to assess the effectiveness of the ensemble approach.
- Incorporate findings into the report to showcase the improved prediction system.

References

[1] Analytics Vidhya. (2022). Tutorial on RNN, LSTM & GRU with Implementation. *Analytics Vidhya*. Retrieved from https://www.analyticsvidhya.com/blog/2022/01/tutorial-on-rnn-lstm-gru-with-implementation/

[2] Eivazi and Guastoni, Luca and Schlatter, Philipp and Azizpour, Hossein. (2021). Recurrent neural networks and Koopman-based frameworks for temporal predictions in a low-order model of turbulence. *International Journal of Heat and Fluid Flow*, 90, 14. doi: 10.1016/j.ijheatfluidflow.2021.108816

[3] P. A. Srinivasan and L. Guastoni, H. Azizpour and P. Schlatter and R. Vinuesa1. (2019). Predictions of turbulent shear flows using deep neural networks. *PHYSICAL REVIEW FLUIDS*, 4, 15. doi: 2469-990X/2019/4(5)/054603(15)

[4] Colah's Blog. (2015). Understanding LSTM Networks. *colah's blog*. Retrieved from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[5] Towards Data Science. (2021). Tutorial on LSTM: A Computational Perspective. *towards data science*. Retrieved from https://towardsdatascience.com/tutorial-on-lstm-a-computational-perspective-f3417442c2cd

[6] TensorFlow: An open-source deep learning framework developed by Google. It provides a flexible ecosystem of tools, libraries, and resources for building and deploying machine learning models. Available at: https://www.tensorflow.org/

[7] An Introduction to RNN and LSTM [Video file]. (2020, Oct 8). Retrieved from YouTube: https://youtu.be/Mdp5pAKNNW4

End of the Report