Machine Learning Predictive Failure Times Data

Naveena Yeppala, Eric Schnelker
April 28th, 2021

I Abstract:

Time series data, particularly software failures and other data associated failures, plays a major role in software reliability. Part of its importance lies in the ability to use that past data with machine learning, which is helpful to generate predictions of future data in real time. Being able to predict time series data in real time can be useful, as most problems that require the use of time, in order to solve make it difficult to predict and handle that time. While there are many time series solutions that can be used to solve this issue of needing data in real time, such as logistic regression, multi-layer perceptron classifiers (a type of feedforward artificial neural network, or ANN), support vector machines and many others, they all have a common flaw: they do not store any data, nor do they support scalability. This flaw was solved when neural networks began to be utilized to solve time series problems. In our research, we worked on single-step and multi-step variations of both a recurrent neural network and a Long Short-Term Memory recurrent neural network, in order to utilize given time series data to predict the time between the last software failure and the next one (for single-step) or x (for multi-step, where x is user input) amount of software failures, as well as when they would occur.

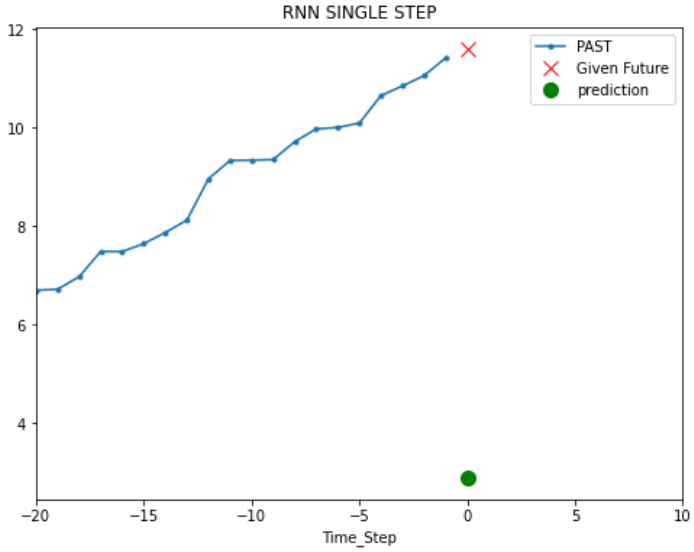
II Introduction:

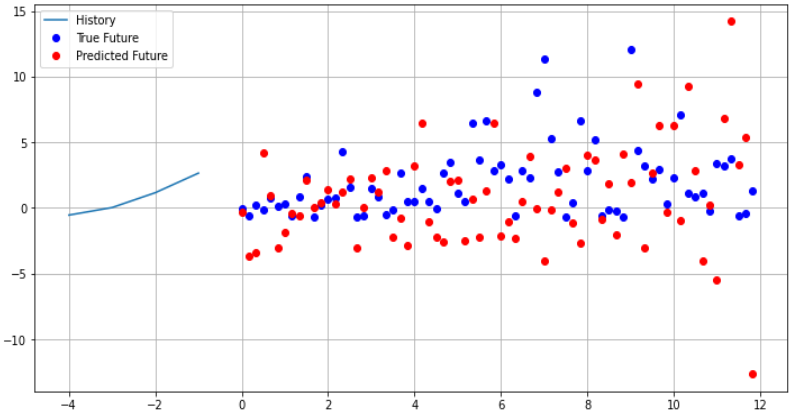
Time series data, which, as the name suggests, is dependent on time, but also is dependent on the data's environment. The prediction of time series data can be estimated with the assistance of methods and tools such as decision trees, multi-layer perceptron classifiers, logistic regression, recurrent neural networks, recurrent neural networks using Long Short-Term Memory, as well as many others. These various solutions use different methods and metrics to get future time series predictions using past data. A significant challenge associated with finding a relatively accurate solution for these predictions is estimating the problem and its parameters of the given data. In the case of our research, we made use of failure times data, which consists of the number of failures, the time between failures (in seconds), as well as the times at which each of these failures occurred (in seconds). The primary problem with this type of software failure data arises when either the given data is small, or it does not have enough features to achieve the required prediction accuracy. In the case of recurrent neural networks and Long Short-Term Memory recurrent neural networks, a large amount of data is required to make accurate predictions. In recurrent neural networks (including the Long Short-Term Memory, as it is a

specific type of neural network), the given data is split into training and testing datasets. The training dataset is used for training the model and utilizes the given failure data to predict a future value. The testing dataset is used to test the trained model. After the creation of these datasets, the training data is fed into the model and the model is trained (and predictions are made in the process as it is trained). Once the model is trained, the testing dataset is fed into the trained model, allowing the model to predict potential future values. Given that failure data is always in demand and is quite useful, it is important to fit the data and predict it by feeding the data into reliable algorithms.

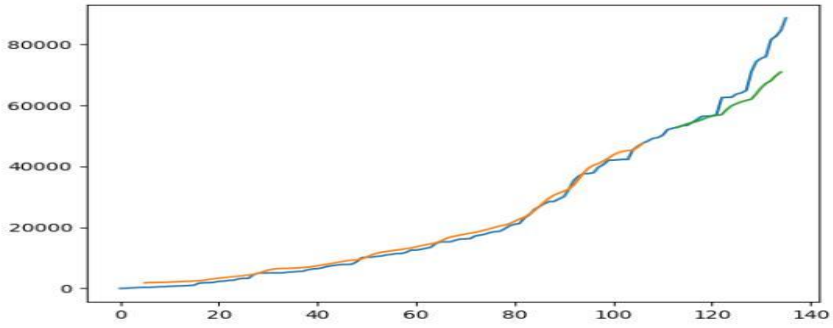
This paper talks about the uses of failure data in recurrent neural networks, as well as more specifically Long Short-Term Memory recurrent neural networks. We utilized the SYS1 dataset to obtain our failure data. The Long Short-Term Memory network overcomes many of the limitations which existed with the other recurrent neural network that we used through the development of a single-step and multi-step Long Short-Term Memory network implementation. To assess the performance of Long Short-Term Memory, the other recurrent neural network's results are compared with those of the Long Short-Term Memory. As mentioned prior, the Long Short-Term memory network is a recurrent neural network, yet, as opposed to our other implementation of a recurrent neural network, the LSTM has many additional standards which makes the LSTM more reliable when evaluating most time series data (in this case, failure data). Part of the reason for this reliability is that the LSTM has memory which is stored for a longer period than the other recurrent neural network and is implemented through the use of a simple structure.

To assess the performance of both the LSTM and the recurrent neural network, the model used for both approaches are sequential models. While we mentioned that the LSTM is reliable, the recurrent neural network's results are not reliable due to its lack of storage. Furthermore, despite the recurrent neural network making some predictions correctly, the accuracy rate of the model is below 5% and the mean squared error during training is very high, showcasing some of the various reasons why this recurrent neural network is not a good option for predicting the time series data. The below graphs are the single-step and multi-step predictions for the Recurrent Neural network. The data here is standardized.





Looking at the data from the LSTM, the data seems to fit well for initial training data, but the test data has some major variances as the last 20% of the initial data has massive variance in the given time between failures, whereas the initial 80% of the data used for the training dataset is more consistent, as shown by the graphs below. The model's calculated accuracy is 90.3%, with an average percent error of 9.7%. The LSTM's implementation involved taking an existing single step LSTM that predicted plane passengers in a given month of a given year, modifying it extensively so that it would work with our dataset, and then modifying it even further to allow it to also function as a multi-step LSTM. The data parameters are failure number, time between failure, and failure time. When reading the data into data frames to prepare and normalize the data for the LSTM, only the failure time is utilized, as calculating the next time between failure or the next failure number would not have been very informative. As such, it becomes clear that this LSTM is a univariate LSTM with single step and multi-step capabilities.



blue is initial data, orange is training, green is test data. Validation data not included

```
Prediction(s): [[530.67676]]
Train Score: 655.60 RMSE
Test Score: 9591.64 RMSE
Prediction(s) Score: 179.68 RMSE
The average percent error of the predictions made is: 51.189962061542204
The accuracy of the model is: 48.810037938457796
```

The first prediction shown is from the single step aspect of the LSTM and, while the training data is accurate, the prediction itself may not be accurate. While the LSTM has a high accuracy for multi-step, it has a higher chance of having a lower accuracy on a single point just due to the potential variance in the low first point and the trained model's prediction. That being said, the root mean squared error on the single-step prediction still shows a fairly accurate prediction. The multi-step aspect of the LSTM can predict any number of future points that the user needs or desires. We tested for up to 50 predictions in one run.

III Data Scaling and Standardization:

To avoid the training data overfitting the initial data, scalar is imported from the sklearn.preprocessing module. Scalar allows for the normalization of data such that the mean of the values would be zero and the standard deviation would be 1. These two factors help ensure that the training data does not overfit by using fit_transform on the X training data and then transforming the X test data as well. Plotting the data before and after scaling shows that this data normalization makes a large difference in the accuracies of the predictions. While data scaling is used heavily in work with neural networks, it is also widely used in various other **algorithms**

III.A Data Standardizing:

Data standardization is also required in some machine learning algorithms when time series data has input values with differing scales. It assumes that observations fit the bell curve with a well-behaved mean and standard deviation.

$$z = \frac{X - \mu}{\sigma}$$

In order to get the standardization, subtract mean from the data you want to convert and divide the result by the standard deviation together gives the standardized data.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

The mean is then subtracted from the data and squared. Then, calculate the sigma notation and divide it by total number of values N, which finally results in the standard deviation of the data

M = sum of terms / number of terms – Mean of the data

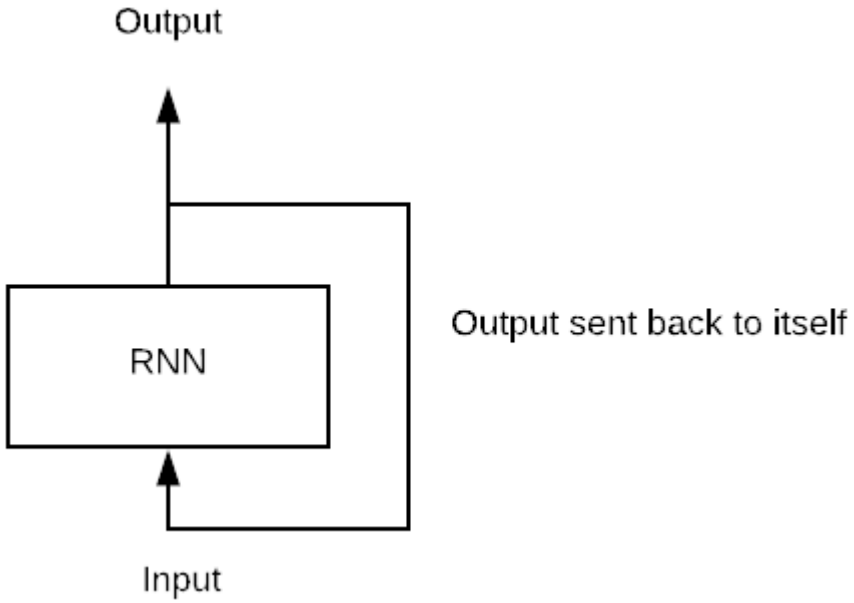
```
print(X_mean)
print(X_std)
print((X))
```

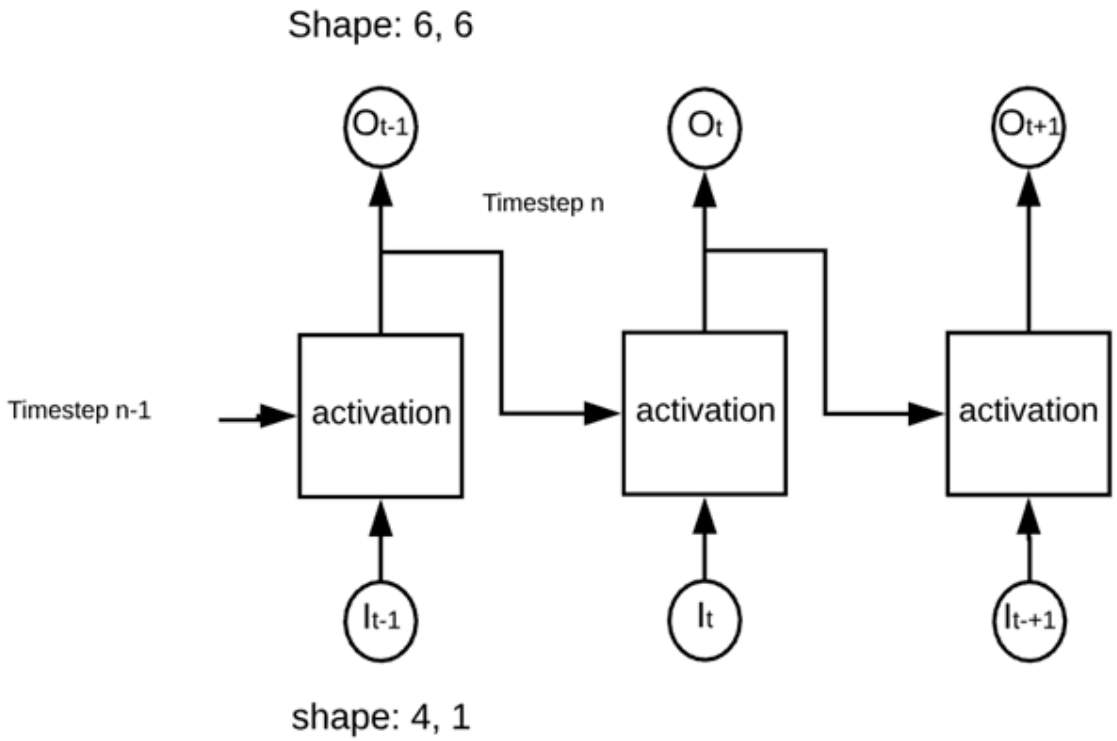
```
2.960594732333751e-17
1.0
[-1.13768175e+00 -1.11707821e+00 -1.03947154e+00 -9.83841980e-01
 -9.04861739e-01 -8.98680677e-01 -8.97307107e-01 -8.34809699e-01
 -7.57889813e-01 -7.47588042e-01 -6.52811753e-01 -6.18472518e-01
 -5.65590096e-01 -5.49107263e-01 -4.74934515e-01 -4.14497461e-01
  4.56482899e-02  1.28062454e-01  1.45918857e-01  2.24212313e-01
  4.47417341e-01  4.85190500e-01  6.51392398e-01  6.98093758e-01
  9.87916902e-01  1.11153815e+00  1.11840600e+00  1.90546127e+00
  2.31753209e+00  2.32783386e+00  2.35255811e+00  2.35530525e+00
```

IV Recurrent Neural Networks:

The recurrent neural network is a network that allows previous outputs to be used as inputs while having hidden states and it is mostly used in natural language processing. Recurrent neural networks learn similarly while training and they generate output from the prior input. The inputs are sequential, and the network also has repeated inputs to generate the outputs.

In the code, The RNN layer is used to iterate over the time stamps of a sequential model, while maintaining an internal state information. The different built-in RNN layers are LSTM, SimpleRNN and GRU. The SimpleRNN used in this model is a fully connected RNN where the output from previous timestamp is to be fed to the next timestep.





The network here shows the four inputs, six neurons and 2-time stamps.

The primary problems with recurrent neural networks are that they tend to have slow computation speeds, cannot access data stored long in the past due to their lack of long-term memory (with LSTMs being an exception to this), and finally poor prediction accuracy. The recurrent neural network presented in this paper can be represented as:

$$\mathbf{h}^{(t)} = f(\mathbf{x}^{(t)}, \mathbf{h}^{(t-1)}).$$

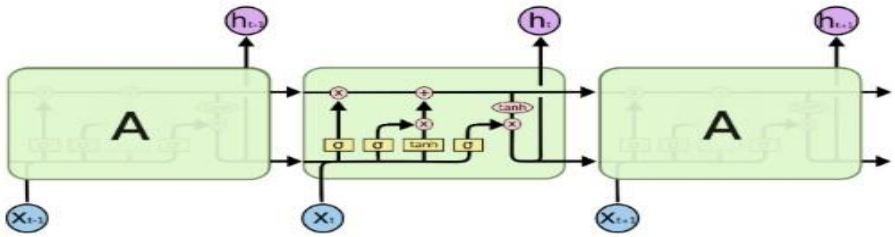
h(t) - new state, f(w) – some function with parameters W
h(t-1) – old state, x(t) – input vector at some time step

By expanding it recursively, we get the following:

$$\mathbf{h}^{(4)} = f(f(f(\mathbf{h}^{(1)}, \mathbf{x}^{(2)}), \mathbf{x}^{(3)}), \mathbf{x}^{(4)}). \qquad (1)$$

V LSTM:

The LSTM is a special kind of recurrent neural network which is used for learning long-term dependencies. The LSTM, as the name suggests, allows for both long-term memory and short-term memory to be utilized. This is achieved due to the recurrent module of the model in which four layers interact with each other. It is not only capable of processing single data points (such as images), but also entire sequences of data, A general LSTM has a cell, an input gate, an output gate, and a forget gate. This data is remembered over a much longer time period than most other recurrent neural networks can do. This long-term memory is well-suited for classifying, processing, and making predictions based on time series data.



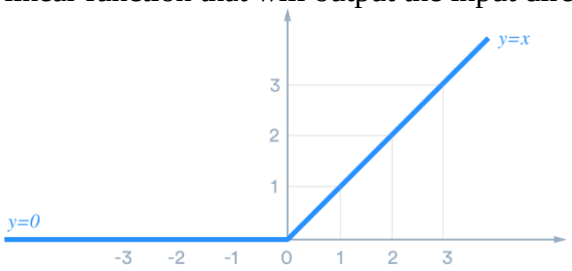
The repeating module in an LSTM contains four interacting layers.

The LSTM created for our research utilized the SYS1 data. After formatting the data into a dataframe and separating usable data from unusable data (as this is a univariate approach), the sklearn module is used to normalize the data. The parameters for the algorithm are failure number, time between failure, and failure time. When reading the data to prepare for the LSTM, only the failure time data is used, as the other data provided little of value in terms of predictions. In order to format the data in the aforementioned way, we made heavy use of the Pandas module to format and handle data in datasets, whereas TensorFlow and Keras were used to build, train, and use the LSTM.

While the input of the LSTM is time between failure data, the output is (are) (a) prediction(s) for mean squared error. To find the next prediction, the root mean squared error (RMSE) was taken. For a single step prediction, this one RMSE was the prediction. For multi-step, this RMSE was added to the data set after calculating failure time and failure number (the time between failures predicted this way were consistently within the confidence interval). After adding this prediction to the data set, the model was retrained to predict the following failure (and it repeats this process until it predicts as many time steps as the user indicates they want at the beginning of the program).

VI Activation function:

The Activation Function plays a key role in Neural Network**.** Without them, our neural network would become a combination of linear functions, so it would be just a linear function itself. The default activation function is used in the RNN that is rectified linear activation or ReLU which is a linear function that will output the input directly, which is mathematically defines as y=max(0,x)



This is the most used function.

$$y_i = \begin{cases} x_i & \text{if } x_i \geq 0 \\ 0 & \text{if } x_i < 0. \end{cases}$$

VII Compile():

The optimizer and loss are the two arguments required for compiling a model, along with metrics. We used loss as the metric for mean squared error and ‘adam’ as our optimizer.

VII.A Loss:

This function shows how far we are from the ideal solution. Here we used a regression loss that is mean squared error, which is sum of squared distances between out target variable and the predicted values. This is a required function in the model compilation

$$MSE = \frac{\sum_{i=1}^{n} (y_i - y_i^p)^2}{n}$$

```
200/200 [==============================] - 2s 6ms/step - loss: 0.4720 - val_loss: 688.1258
Epoch 2/10
200/200 [==============================] - 1s 5ms/step - loss: 0.0107 - val_loss: 680.7972
Epoch 3/10
200/200 [==============================] - 1s 5ms/step - loss: 0.0011 - val_loss: 679.7681
Epoch 4/10
200/200 [==============================] - 1s 5ms/step - loss: 1.1867e-04 - val_loss: 679.4835
Epoch 5/10
200/200 [==============================] - 1s 5ms/step - loss: 6.2540e-06 - val_loss: 682.6265
Epoch 6/10
200/200 [==============================] - 1s 5ms/step - loss: 1.2317e-07 - val_loss: 683.3038
Epoch 7/10
200/200 [==============================] - 1s 6ms/step - loss: 1.3484e-09 - val_loss: 682.4136
Epoch 8/10
200/200 [==============================] - 1s 5ms/step - loss: 9.4184e-12 - val_loss: 678.2592
Epoch 9/10
```

The value of loss is higher there by the ideal solution is far, which indefinitely shows lower accuracy.

VII.B Optimizer:

This function optimizes the input weights by comparing the prediction and loss function. Adam optimizer is the default optimizer which is used in the model compilation.

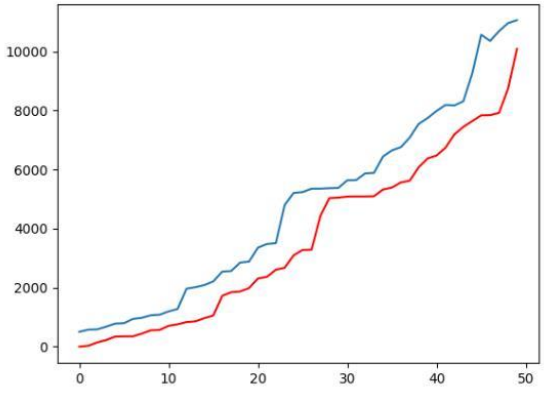
```
keras.optimizers.Adam(learning_rate = 0.001, beta_1 = 0.9, beta_2 = 0.999, amsgrad = False)
```

V11.C Metrics:

This is used to evaluate the performance of the model. Which is similar to loss function, but not used in training process. In the RNN we used simple accuracy metrics.

V11.D Results and Validation:

After training the data, I verified that training worked correctly by using the training data to predict the RMSE of the predictions. The RMSE was low, indicating a good accuracy for the initial data. When repeating on the test data, the RMSE was significantly higher, which indicates some variance and shows that the model tended to overfit the data. This is expected as the data actually has some major variances in failure times in the last 20% of the data (as the train/test split was 80:20). Using 1000 epochs, a look_back of 5, and a batch_size of 1, the validation data (where the number of points used to validate is provided by the user), the RMSE was 279.89, indicating a very low error. Taking the percent error with the actual values fed into the LSTM and the prediction, I calculated the average accuracy of the model to be 87.7%. This accuracy was lower the fewer predictions it made, which makes sense, as the LSTM gets better when it can use old predictions to assist in the creation of new predictions. While the parallelism in the prediction data (in blue) versus the accepted values (in red) is a bit strange, it is important to note the data scale. The RMSE of 279.89 and percent error of 12.3% would indicate this. Therefore, it is clear to me that this model is very accurate for making large multi-step predictions, but not as accurate for single-step predictions.




```
Train Score: 463.02 RMSE
Test Score: 11374.21 RMSE
Prediction(s) Score: 279.89 RMSE
The average percent error of the predictions made is: 12.222135309346722
The accuracy of the model is: 87.77786469065327
```

Multi-Step Prediction

```
Prediction(s): [[530.67676]]
Train Score: 655.60 RMSE
Test Score: 9591.64 RMSE
Prediction(s) Score: 179.68 RMSE
The average percent error of the predictions made is: 51.189962061542204
The accuracy of the model is: 48.810037938457796
```

Single-Step Prediction

VIII Neural Network Model:

This model is used for a plain stack of layers where each layer has input and output and is used in TensorFlow. Multiple layers with different units can be connected. As a result using this most complex networks can also be built. This model is used in LSTM, convolutional neural network and recurrent neural network.

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_22 (SimpleRNN)    (None, 16)                288
_________________________________________________________________
dense_26 (Dense)             (None, 1)                 17
_________________________________________________________________
dense_27 (Dense)             (None, 1)                 2
=================================================================
Total params: 307
Trainable params: 307
Non-trainable params: 0
_________________________________________________________________
```

The Above is the model summary for the single-step Simple RNN.

IX Conclusion:

The failure data in the SYS1 dataset worked well with the LSTM implementation, as the predictions were more reliable, and the model had better accuracy than recurrent neural network. Even though the single-step prediction was less accurate than the multi-step prediction, the prediction was still fairly accurate. The greater number of data points and features fed to the model, the larger the increase and the greater the reliability and accuracy. The single step and multi-step predictions allow for the predictions to be achieved much easier than the recurrent neural network implementations predictions. In order to continue to improve the reliability and accuracy, we also had considered a window method for the predictions, but decided for the lookback regression model for the LSTM instead.

X References:

[1]https://www.facebook.com/MachineLearningMastery, "Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras," *Machine Learning Mastery*, Jul. 20, 2016. https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-pyth on-keras/ (accessed Apr. 28, 2021).

[2]Lianne, "3 Steps to Forecast Time Series: LSTM with TensorFlow Keras | Towards Data Science," *Medium*, Mar. 22, 2020. https://towardsdatascience.com/3-steps-to-forecast-time-series-lstm-with-tensorflow-keras-ba88c6f05237 (accessed Apr. 28, 2021).

[3] J. Chung, C. Gulcehre, and K. Cho, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," 2014. [Online]. Available: https://arxiv.org/pdf/1412.3555.pdf.

[4] M. Phi, "Illustrated Guide to LSTM's and GRU's: A step by step explanation," *Medium*, Sep. 24, 2018. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

[5] https://www.facebook.com/pranjal.srivastava3, "Long Short Term Memory | Architecture Of LSTM," *Analytics Vidhya*, Dec. 10, 2017. https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-ls tm/.

[6]Wikipedia Contributors, "Long short-term memory," *Wikipedia*, Mar. 25, 2021. https://en.wikipedia.org/wiki/Long_short-term_memory#:~:text=Long%20short%2Dterm%20memory%20(LSTM)%20is%20an%20artificial%20recurrent,the%20field%20of%20deep%20learning.&text=LSTM%20networks%20are%20well%2Dsuited,events%20in%20a%20time%20seris.

[7]"CS 230 - Recurrent Neural Networks Cheatsheet," *Stanford.edu*, 2021. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

[8]Mahendran Venkatachalam, "Recurrent Neural Networks - Towards Data Science," *Medium*, Mar. 2019. https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce (accessed Apr. 28, 2021).

[9]"TimeSeries-LSTMModel-Tutorialspoint," *Tutorialspoint.com*,2021. https://www.tutorialspoint.com/time_series/time_series_lstm_model.htm#:~:text=Time%20Series%20%20LSTM%20Model%201%20Neural%20Networks.,capable%20of%20learning%20long%20term%20dependencies%20in%20data.

[10]Wikipedia Contributors, "Recurrent neural network," *Wikipedia*, Apr. 16, 2021. https://en.wikipedia.org/wiki/Recurrent_neural_network.

[11]Enrico Zio, Matteo Broggi, L. R. Golea, and N. Pedroni, “Predicting Reliability by Recurrent Neural Networks,” *ResearchGate*, Jun. 30, 2008. https://www.researchgate.net/publication/266390994_Predicting_Reliability_by_Recurrent_Neur al_Networks.

[12]B. Xu, N. Wang, T. Chen, and M. Li, “Empirical Evaluation of Rectified Activations in Convolution Network,.” Accessed: May 06, 2021. [Online]. Available: https://arxiv.org/pdf/1505.00853.pdf.

[13]Malaiya, Y. and Whitley, D., 1991. Prediction of software reliability using neural networks. [online] Available at:<https://www.researchgate.net/publication/3507480_Prediction_of_software_reliability_using_neural_networks>.

[14]J. Chung, C. Gulcehre, and K. Cho, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” 2014. [Online]. Available: https://arxiv.org/pdf/1412.3555.pdf.

[15]Time series forecasting, “Time series forecasting | TensorFlow Core,” TensorFlow, 2016. https://www.tensorflow.org/tutorials/structured_data/time_series.

[16]N. Karunanithi , D. Whitely, K. Malaya , "Prediction ofSoftware Reliability Using Connectionist Models", IEEETransactions on Software Engineering, vol 18, No 7, July 1992, pp 563-574

[17]M. Xie, “Software reliability models—A selected annotated bibliography,” *Software Testing, Verification and Reliability*, vol. 3, no. 1, pp. 3–28, Mar. 1993, doi: 10.1002/stvr.4370030103.

[18] Keras Team, “Keras documentation: Timeseries forecasting for weather prediction,” *Keras.io*, 2020. https://keras.io/examples/timeseries/timeseries_weather_forecasting/.