Enhancing Industrial Predictive Maintenance with LSTM Networks

Andrew Nguyen
Computer Science
University of Texas at Dallas
Richardson, US
AXN210071@utdallas.edu

Ousama Batais
Computer Science
University of Texas at Dallas
Richardson, US
oxb220001@utdallas.edu

John Kasperbauer
Computer Science
University of Texas at Dallas
Richardson, US
jmk190007@utdallas.edu

Shuxin Xue
Computer Science
University of Texas at Dallas
Richardson, US
sxx210006@utdallas.edu

***Abstract*—This article explores how to use Long Short-Term Memory (LSTM) networks to predict maintenance needs based on time series data. LSTMs are a special type of recurrent neural network that is often used to model complex temporal dependencies in equipment sensor data, with the goal of improving the reliability and efficiency of maintenance interventions, such as intervening when maintenance costs are lower. We experiment with the approach using the AI4I 2020 Predictive Maintenance dataset and show model training, and validation are discussed, along with an analysis of the model's performance using classification metrics and ROC curves.**

***Index Terms*—LSTM, Predictive Maintenance, Time Series Prediction, Recurrent Neural Networks, Machine Learning**

I. Introduction

Predictive maintenance is an important process in industrial equipment management, which can save costs and reduce time through timely intervention. Traditional methods are usually applied to scheduled maintenance or reactive methods, which may lead to unnecessary service or risk failure. Machine learning, especially recurrent neural networks (RNNs) like LSTM, can predict maintenance needs by collecting real-time data streams, which will provide a promising and scalable alternative. This article will discuss the implementation of LSTM models, their integration with maintenance workflows, and their advantages and implementation over traditional methods.

II. Background Work

The concept of using machine learning for predictive maintenance has been explored using a variety of models such as classical statistical methods, machine learning techniques, support vector machines, and neural networks. LSTM networks are a type of RNN known for their effectiveness and their ability to avoid long-term dependency issues, making them ideal for predictive maintenance tasks where patterns change over time. They avoid the vanishing gradient problem, which allows them to effectively learn dependencies from long sequences of data. Studies such as [Ref 1] have highlighted the benefits of LSTM models in settings where patterns across time are important to making accurate predictions.

III. Theoretical and Conceptual Study

LSTM networks are an advanced form of RNN models developed to deal with the vanishing and exploding gradient problems typical in traditional RNNs. The main idea of LSTM is the cell state, effectively carrying relevant information throughout the sequence of data using gates that regulate the flow of information into and out of the cell, due to their ability to process large volumes of data with intricate temporal patterns. These gates (input, forget, and output gates) decide which information is essential to keep or discard as it sequentially processes data, making it highly effective for time-series prediction tasks such as predictive maintenance. Theoretical advances in machine learning, especially in the domain of deep learning, have proven effective in sequential data analysis found in time-series applications [ref 2].

IV. Methodology

A. Data Description

The dataset, ”AI4I 2020 Predictive Maintenance Dataset”, is a synthetic dataset that emulates maintenance data seen in real-life industries. In this dataset, Type, Air Temperature, Process Temperature, Rotational Speed, Torque, and Tool wear are our features while Machine failure, Tool Wear Failure (TWF), Heat Dissipation Failure (HDF), Power Failure (PWF), Overstrain Failure (OSF), and Random Failure (RNF) are our targets. In our model, we focus on making Machine Failure our target, as that is the overall failure of every industry. For preprocessing the dataset, our team use a wide variety of methods to analyze and remove elements. After fetching the data from the dataset, features of this dataset are plotted to understand the dataset. Aspects like UID and Product ID are dropped, as they do not play a role in our ML model. The data is also put in a correlation matrix and are standardized. Finally, the dataset is split into testing and training data such that our ML model use our standardized data.

V. Model Development

The implementation of the model was done in Python using minimal external libraries to focus on the core LSTM architecture.

A. Data Description

An LSTM network with configured parameters was adopted to model the dependencies across time steps in sensor readings. These sensor readings looked at the various variables at play in the dataset that affected tool maintenance; air and process temperatures, rotational speed, torque, and tool wear [ref 3]. The synthetic dataset provided a solid foundation for training and testing the predictive capabilities of the LSTM network.

B. Modeling the Dataset

The model's architecture is designed to accommodate the sequential nature of the dataset, optimizing for both short-term and long-term dependencies. It should also be known that the primary algorithm used by the model consists of no built-in libraries aside from within the realms of pre-processing, loading, and results analysis. After the data in the set had been scrutinized, the model plots the correlation of the various features to better visualize patterns between the features and their affects on toll maintenance as seen below in figure 4.

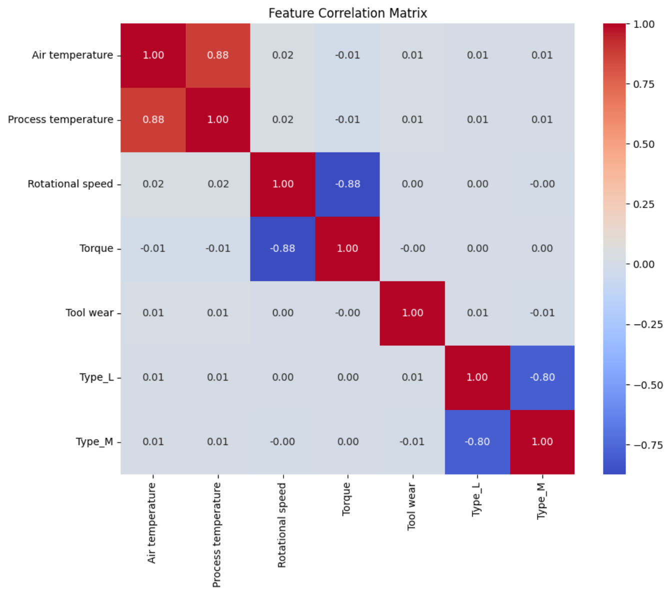


Fig. 1. Correlation Matrix from [ref 4]

After mapping the correlation of variables the dataset features were standardized and split into testing and training sets for the model to evaluate.

C. Model Implementation

With the dataset prepared the LSTM network is implemented.

1) Hyper Parameters: Utilizing the standardized input we declared 61 units in the hidden layer, a 0.4 learning rate, 30 epochs, a batch size of 128 samples and trivially a single input and output layer.

2) Weights: All weights were initialized with random values depending on the size of the layers and current layer. Initializations for the weights followed the formula: $np.random.randn(hidden\_dim, input\_dim)$ for input to hidden layer, $np.random.randn(hidden\_dim, hidden\_dim)$ between hidden layers, and $np.random.randn(output\_dim, hidden\_dim)$ for hidden to output layer. From the input to hidden layers the weights for the gates; 'Wf' ('Forget' gate), 'Wi' ('Input' gate), 'Wo' ('Output' gate), and 'Wc' ('Candidate Cell' gate) were scaled using the formula: $np.sqrt(2/(hidden\_dim + input\_dim)$ (a variant of He Initialization in Python) scaling the initialized weights to the size of the input and hidden layer.

For weights between and connecting the hidden layers, the weights for the gates; 'Uf', 'Ui', 'Uo', and 'Uc' (using the same naming convention and representation as aforementioned) were scaled by using the formula: $np.sqrt(2/(hidden\_dim + hidden\_dim))$ which appropriately scales the weights for the size of the hidden state. A final weight 'V' connects the hidden layer to the single output layer and is similarly scaled using the formula: $np.sqrt(2/(hidden\_dim + output\_dim))$.

3) Biases: Biases for the gates and cell states consisted of; 'bf', 'bi', 'bo', and 'bc' (using the same naming convention and representation as aforementioned with the weights). Unlike the previously defined gates these biases were originally all initialized to zero allowing the model to learn appropriate biases during the training phase.

4) Cell Forward Passes: Both Sigmoid and Tanh activation functions were utilized when calculating the forward passes between the layers of the LSTM network. At each epoch the gates for the LSTM network are calculated in order of 'Forget' gate, 'Input' gate, Candidate Cell State, Next Cell State, 'Output' gate, Next Hidden State, and Predicated values.

The Forget gate, Input gate and Output gate use the same formula: $sigmoid(np.dot(weights[X], xt) + np.dot(weights[Y], h\_prev) + biases[Z])$ where X is the input weight to the hidden layers (*Wf/Wi/Wo*), Y is the weights between hidden layers (*Uf/Ui/Uo*) and Z is the appropriate bias (*bf/bi/bo*). After a summation of the appropriate weights and biases the sum is run through a sigmoid function to get the final values for the gate to be calculated resulting in either a 1 or 0. The meaning behind this binary output depends on the gate; in a Forget gate it determines how much previous memory should be kept, in the Input gate it determines how much new information should be stored in a cell state, and in the Output gate it determines which part of a cell state should be output as a hidden state.

Similarly, the Candidate State is calculated by the formula: $tanh(np.dot(weights['Wc'], xt) + np.dot(weights['Uc'], h\_prev) + biases['bc'])$ to represent new potential memory which can be added to the cell state within the range of [-1,1]. The results of the New Cell State use the results of the Forget gate, Input gate and Cell State in addition to an older cell state using the following formula: $New\ Cell\ State = (Forget\ gate * Previous\ Cell\ State) + (Input\ gate * Candidate\ State)$ which adds part of the New Candidate State to an aforementioned part of an older cell state. After this is calculated the Output gate also mentioned previously is calculated.

The Next Hidden state is calculated with a combination of the Output gate results and the results of the Next Cell State ran through a Tanh function, the results will determine how much of the cell at the current epoch will be passed as a hidden state for the next epoch. The Predicted Output is finally calculated by the formula: $y_pred = sigmoid(np.dot(weights['V'], h\_next))$ where h_next (next hidden state) is passed through a linear transformation and sigmoid is used for binary output.

5) Forward Passes for a Sequence: Forward passes for sequences are calculated by going through each epoch in a given sequence and applying the same steps specified for Cell Forward Passes. After each epoch the resulting hidden states and predicted outputs will be appended to a list of previously known hidden states and predicted outputs.

6) Binary Cross-Entropy Loss: Predicted log loss (binary cross-entropy loss) is computed using the Binary Cross-Entropy Loss function with the predicted loss put in a range between 1e-15 and 1 - (1e-15) before being averaged.

7) Training with Gradient Descent: To train the machine the following text will describe the algorithm that is performed. For every epoch the training data will be split up into different sample batches, and for each sample batch a forward pass as described above will be performed to compute the predicted outputs. A Binary Cross-entropy Loss is then performed as described previously between true labels and the predicted outputs. Afterwards Back Propagation Through Time is performed to compute gradients of loss in respect to the appropriate weights and biases before normalizing the gradients for use with the the learning rate in Gradient Descent to update those same weights and biases. After each epoch has undergone this process the average training loss is then computed and the machine will learn to minimize loss to improve predictions.

VI. Results and Analysis

The LSTM model was developed to predict various types of failures from operational data collected from industrial machines.

A. Training Process

Initially the model was intended to be trained over a maximum of 10 epochs with early stopping implemented to prevent overfitting in the event that validation loss saw no further improvement. Actual training however stopped after just 4 epochs since the validation loss observed indicated no further improvement, stabilizing at a value around 0.0510 as depicted in the figure 2.

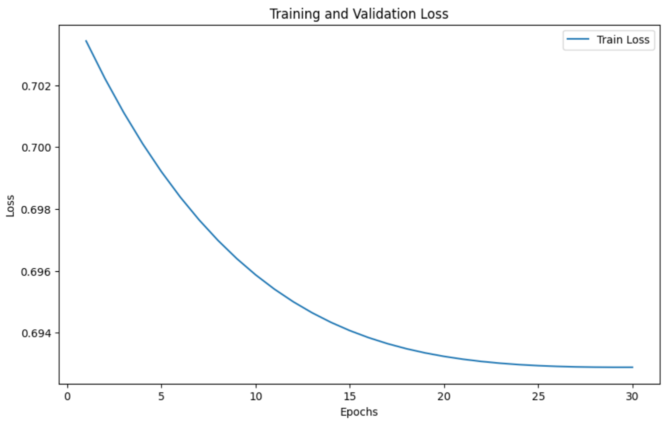


Fig. 2. Training and Validation Loss Over Epochs from [ref 4]

B. Model Evaluation

The LSTM model's performance had been quantitatively assessed using the Mean Squared Error (MSE) loss metric. Within the duration of observation both training and validation losses quickly plateaued indicating that the model had reached learning capacity with the provided features and hyper-parameters.

Actual testing results involved evaluating the model on a separate subset of the dataset hidden from the model during the training phase. This had been done to simulate real-world predictions within the limited data of the dataset. Testing concluded that recorded test loss experienced actual loss to be as low as 0.0517, closely aligning with the validation loss, which as indicated by the figure below sat below 0.694 confirming the model's consistency and predictive reliability.

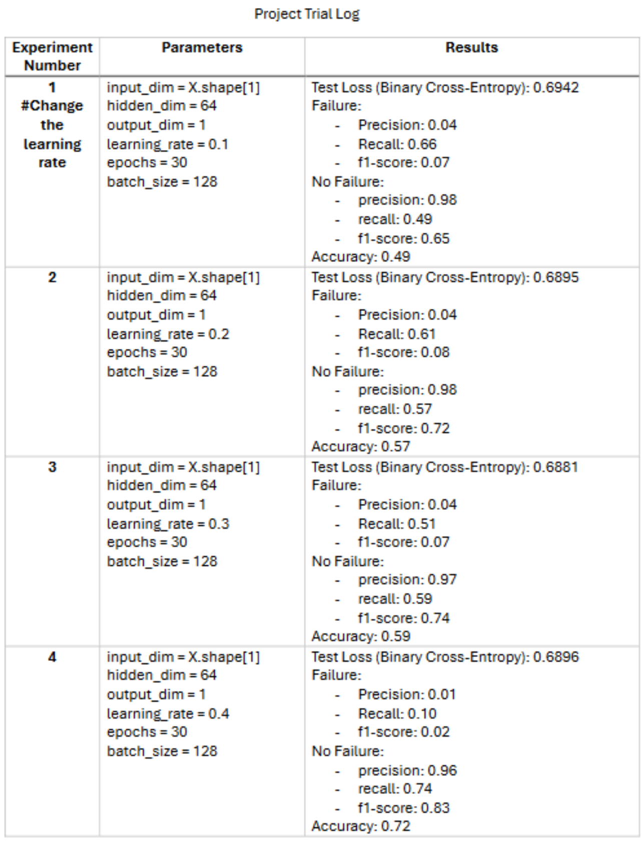
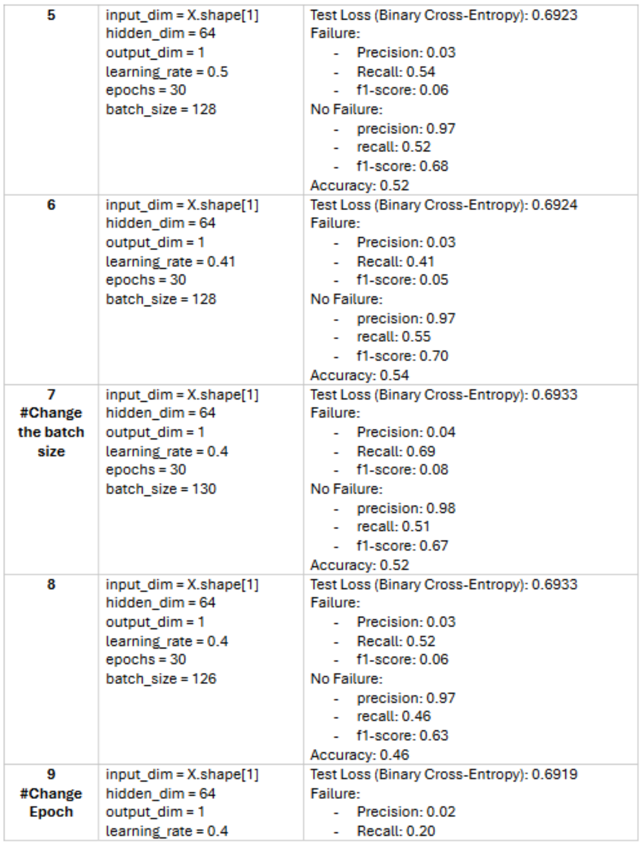
1) Model Trial Log: This result shows that the LSTM model, although simple, is able to effectively capture and predict maintenance needs based on the given features without overfitting, as evidenced by the close performance between the training and testing phases.

VII. Related Work

This section provides the existing literature on the application of LSTM networks in predictive maintenance. Several studies have demonstrated the advantages of LSTM models in capturing long-term dependencies in sequence data, which is crucial for accurate prediction when sensor readings have temporal correlations.

VIII. Conclusion and Future Work

The LSTM model displayed promising results in predicting machine failures, suggesting that such models can significantly improve predictive maintenance strategies. Future research will explore the integration of more diverse data types, such as acoustic and vibration signals, and the use of ensemble techniques to enhance predictive accuracy.

Project Trial Log

Experiment Number	Parameters	Results

1
#Change the learning rate
	input_dim = X.shape[1]
	hidden_dim = 64
	output_dim = 1
	learning_rate = 0.1
	epochs = 30
	batch_size = 128
		Test Loss (Binary Cross-Entropy): 0.6942
		Failure:
		- Precision: 0.04
		- Recall: 0.66
		- f1-score: 0.07
		No Failure:
		- precision: 0.98
		- recall: 0.49
		- f1-score: 0.65
		Accuracy: 0.49

2	input_dim = X.shape[1]
	hidden_dim = 64
	output_dim = 1
	learning_rate = 0.2
	epochs = 30
	batch_size = 128
		Test Loss (Binary Cross-Entropy): 0.6895
		Failure:
		- Precision: 0.04
		- Recall: 0.61
		- f1-score: 0.08
		No Failure:
		- precision: 0.98
		- recall: 0.57
		- f1-score: 0.72
		Accuracy: 0.57

3	input_dim = X.shape[1]
	hidden_dim = 64
	output_dim = 1
	learning_rate = 0.3
	epochs = 30
	batch_size = 128
		Test Loss (Binary Cross-Entropy): 0.6881
		Failure:
		- Precision: 0.04
		- Recall: 0.51
		- f1-score: 0.07
		No Failure:
		- precision: 0.97
		- recall: 0.59
		- f1-score: 0.74
		Accuracy: 0.59

4	input_dim = X.shape[1]
	hidden_dim = 64
	output_dim = 1
	learning_rate = 0.4
	epochs = 30
	batch_size = 128
		Test Loss (Binary Cross-Entropy): 0.6896
		Failure:
		- Precision: 0.01
		- Recall: 0.10
		- f1-score: 0.02
		No Failure:
		- precision: 0.96
		- recall: 0.74
		- f1-score: 0.83
		Accuracy: 0.72

5	input_dim = X.shape[1]
	hidden_dim = 64
	output_dim = 1
	learning_rate = 0.5
	epochs = 30
	batch_size = 128
		Test Loss (Binary Cross-Entropy): 0.6923
		Failure:
		- Precision: 0.03
		- Recall: 0.54
		- f1-score: 0.06
		No Failure:
		- precision: 0.97
		- recall: 0.52
		- f1-score: 0.68
		Accuracy: 0.52

6	input_dim = X.shape[1]
	hidden_dim = 64
	output_dim = 1
	learning_rate = 0.41
	epochs = 30
	batch_size = 128
		Test Loss (Binary Cross-Entropy): 0.6924
		Failure:
		- Precision: 0.03
		- Recall: 0.41
		- f1-score: 0.05
		No Failure:
		- precision: 0.97
		- recall: 0.55
		- f1-score: 0.70
		Accuracy: 0.54

7
#Change the batch size
	input_dim = X.shape[1]
	hidden_dim = 64
	output_dim = 1
	learning_rate = 0.4
	epochs = 30
	batch_size = 130
		Test Loss (Binary Cross-Entropy): 0.6933
		Failure:
		- Precision: 0.04
		- Recall: 0.69
		- f1-score: 0.08
		No Failure:
		- precision: 0.98
		- recall: 0.51
		- f1-score: 0.67
		Accuracy: 0.52

8	input_dim = X.shape[1]
	hidden_dim = 64
	output_dim = 1
	learning_rate = 0.4
	epochs = 30
	batch_size = 126
		Test Loss (Binary Cross-Entropy): 0.6933
		Failure:
		- Precision: 0.03
		- Recall: 0.52
		- f1-score: 0.06
		No Failure:
		- precision: 0.97
		- recall: 0.46
		- f1-score: 0.63
		Accuracy: 0.46

9
#Change Epoch
	input_dim = X.shape[1]
	hidden_dim = 64
	output_dim = 1
	learning_rate = 0.4
		Test Loss (Binary Cross-Entropy): 0.6919
		Failure:
		- Precision: 0.02
		- Recall: 0.20

TABLE I
CONFUSION MATRIX OF THE LSTM MODEL

	Predicted No Failure	**Predicted Failure**
Actual No Failure	955	984
Actual Failure	21	40

ACKNOWLEDGMENT

The authors would like to thank the University of Texas at Dallas and Dr. Anurag for the support and provided the necessary resources for testing during this research project. A further extension of thanks to the folks at UC Irving for providing the synthetic dataset used by this model.

REFERENCES

[1] Keras Team, ”LSTM layer - Keras,” TensorFlow, 2023. [Online]. Available: https://keras.io/api/layers/recurrent$_layers/lstm/.[Accessed: Nov.21,2023]$.

[2] “Machine Learning for Engineers,” LSTM Networks, Available: https://apmonitor.com/pds/index.php/Main/LongShortTermMemory [Accessed: Nov. 21, 2024].

[3] ”AI4I 2020 Predictive Maintenance Dataset,” UCI Machine Learning Repository, 2020. [Online]. Available: https://doi.org/10.24432/C5HS5C. [Accessed: Nov 21, 2024]

[4] Andrew Nguyen, Ousama Batais, Nov. 22, 2024, AI4I Predictive Maintenance Project, ver. 2.0, [Online]. https://colab.research.google.com/drive/1MYmtIrwn17zH2O4CPEvQhbJA nm1?usp=sharing. [Accessed: Nov. 22, 2024]

	epochs = 35
	batch_size = 128
		- f1-score: 0.03
		No Failure:
		- precision: 0.96
		- recall: 0.64
		- f1-score: 0.77
		Accuracy: 0.62

10	# Hyperparameters
	input_dim = X.shape[1]
	hidden_dim = 64
	output_dim = 1
	learning_rate = 0.4
	epochs = 25
	batch_size = 128
		Test Loss (Binary Cross-Entropy): 0.6910
		Failure:
		- Precision: 0.05
		- Recall: 0.75
		- f1-score: 0.09
		No Failure:
		- precision: 0.99
		- recall: 0.53
		- f1-score: 0.69
		Accuracy: 0.54

11
#Change Hidden Dimension
	input_dim = X.shape[1]
	hidden_dim = 67
	output_dim = 1
	learning_rate = 0.4
	epochs = 30
	batch_size = 128
		Test Loss (Binary Cross-Entropy): 0.6903
		Failure:
		- Precision: 0.05
		- Recall: 0.64
		- f1-score: 0.09
		No Failure:
		- precision: 0.98
		- recall: 0.61
		- f1-score: 0.75
		Accuracy: 0.61

12	input_dim = X.shape[1]
	hidden_dim = 61
	output_dim = 1
	learning_rate = 0.4
	epochs = 30
	batch_size = 128
		Test Loss (Binary Cross-Entropy): 0.6893
		Failure:
		- Precision: 0.05
		- Recall: 0.57
		- f1-score: 0.09
		No Failure:
		- precision: 0.98
		- recall: 0.66
		- f1-score: 0.79
		Accuracy: 0.66