electronics

Article

Deep Learning-Based Time-Series Analysis for Detecting Anomalies in Internet of Things

Saroj Gopali and Akbar Siami Namin *

Computer Science Department, Texas Tech University, Lubbock, TX 79409, USA
* Correspondence: akbar.namin@ttu.edu

Abstract: Anomaly detection in time-series data is an integral part in the context of the Internet of Things (IoT). In particular, with the advent of sophisticated deep and machine learning-based techniques, this line of research has attracted many researchers to develop more accurate anomaly detection algorithms. The problem itself has been a long-lasting challenging problem in security and especially in malware detection and data tampering. The advancement of the IoT paradigm as well as the increasing number of cyber attacks on the networks of the Internet of Things worldwide raises the concern of whether flexible and simple yet accurate anomaly detection techniques exist. In this paper, we investigate the performance of deep learning-based models including recurrent neural network-based Bidirectional LSTM (BI-LSTM), Long Short-Term Memory (LSTM), CNN-based Temporal Convolutional (TCN), and CuDNN-LSTM, which is a fast LSTM implementation supported by CuDNN. In particular, we assess the performance of these models with respect to accuracy and the training time needed to build such models. According to our experiment, using different timestamps (i.e., 15, 20, and 30 min), we observe that in terms of performance, the CuDNN-LSTM model outperforms other models, whereas in terms of training time, the TCN-based model is trained faster. We report the results of experiments in comparing these four models with various look-back values.

Keywords: Bidirectional LSTM (BI-LSTM); temporal convolutional network (TCN); Long Short-Term Memory (LSTM); CuDNN-LSTM; anomaly detection

check for updates

Citation: Gopali, S.; Siami Namain, A. Deep Learning-Based Time-Series Analysis for Detecting Anomalies in Internet of Things. *Electronics* **2022**, *11*, 3205. https://doi.org/10.3390/electronics11193205

Academic Editor: Krzysztof Szczypiorski

Received: 1 September 2022
Accepted: 30 September 2022
Published: 6 October 2022

1. Introduction

The Internet of Things (IoT) is an emerging paradigm that focuses on connecting a variety of intelligent physical devices in order to modernize various domains integrating them and thus improving the quality of life. The phrase "Internet of Things" was originally coined in 1999 by Kevin Ashton [1]. In 2020, it was predicted that global spending on the Internet of Things (IoT) technology would reach 749 billion dollars [2]. It has been predicted that 1.1 trillion US dollars would be spent on IoT globally by 2023. The Asia Pacific region accounted for the largest share of the global IoT market. Europe, the Middle East, and Africa came in second, third, and fourth, respectively.

While the use of IoT-based technologies is on the rise, their security is also of utmost importance for ensuring the successful operations of these connected devices. According to Statista [3], the number of malicious attacks worldwide from 2020 to 2021 on the devices connected through the Internet of Things (IoT) shows that 10.83 million attacks occurred in October 2020.

One of the challenging problems in the IoT sector is anomaly detection in streaming data captured through the connected sensors. To overcome this problem, researchers have designed and tested a number of cutting-edge models. Examples include deep learning-based models such as convolutional neural networks (CNN) [4], graph-based modeling [5], the Graph Attention Network [6], and the Temporal Convolutional Network (TCN) [7]. In these approaches, the primary approach is the modeling of the problem through time-series analysis and then detecting what is called anomaly in the time-series streaming data. An open research question is then how the performances of these models vary with respect to accuracy and training time.

In this paper, we investigate the performance of deep learning models, including recurrent neural networks Bidirectional LSTM (BI-LSTM), and Long Short-Term Memory (LSTM), the CNN-based Temporal Convolutional (TCN), and CuDNN-LSTM, which is a fast LSTM implementation supported by CuDNN. What makes our approach different is that in our modelings, we treat the problem as an estimation problem and not a "classification" one, even though we are also reporting the classification results for the sake of comparison. As a result, the performance metric utilized for comparison is based on Root Mean Square Error (RMSE) and deviation of the actual values.

This work is an extension to our previous work [7], where we compared CNN and RNN-based models through classifications. Here, we extend our approach by considering different types of RNN models and also treating the problem as an estimation problem. To have a fair and unbiased comparison, we build various types of RNN and CNN-based models with similar configurations

Our experimental results show that the TCN model and CuDNN-LSTM model have the lowest Root Mean Squared Error (RMSE) and training time when compared to the BI-LSTM and LSTM models. Furthermore, in terms of performance (i.e., RMSE), CuDNN-LSTM offered the lowest RMSE, whereas the TCN-based model slightly outperforms CuDNN-LSTM in training time. This paper makes the following key contributions:

1. We provide a formulation where the problem of anomaly detection is treated as an estimation problem rather than classification.
2. In the context of anomaly detection in time series, we compare the performance of multivariate RNN-based BI-LSTM, LSTM, CNN-based TCN, and CuDNN-LSTM models.
3. In terms of performance, we find that the CuDNN-LSTM model and TCN-based model outperform the BI-LSTM and LSTM models.
4. In terms of training time, the TCN model outperforms the BI-LSTM, LSTM, and CuDNN-LSTM models.

The remaining parts of this paper are organized as follow as: Section 2 briefly reviews the related work. A short background on the conceptual deep learning techniques studied is presented in Section 3. The experimental setup and procedure are presented in Section 4. The anomaly detection technique and the outcomes of various other attacks are reported in Section 5. Section 6 presents the results of the study along with some discussions. Section 7 concludes the paper and highlights the future research directions.

2. Related Work

Chalapathy and Chawla [8] explored a number of approaches to study deep learning-based anomaly detection. In their work, the authors have highlighted some of the problems with deep anomaly detection and the solutions that have been developed so far. However, there are still problems with the Deep Anomaly Detection (DAD) supervised technique: it has a high computational complexity when applied to a real domain. The unsupervised DAD-based technique requires label acquisition, which is an expensive and time-consuming process and thus makes models less robust when dealing with noisy data.

Wu et al. [9] developed an LSTM-Gauss-NBayes technique for outlier identification in Industrial Internet of Things (IIoT), which included LSTM-NN with the Naive Bayes model. The authors employed a stacked LSTM model, taking advantage of its potent learning potential to handle time-series data with long-term, short-term, and weak temporal dependency.

Du et al. [10] introduced a detection technique called DeepLog, which is an LSTM-based approach. DeepLog is basically a semi-supervised log anomaly identification technique. DeepLog uses log templates and parameter vectors to identify abnormal logs (i.e., some quantitative features of the logs). On the other hand, DeepLog tends to ignore the log's semantic information and the temporal dimension's properties in favor of the template's category data. There is some considerable work that can be performed on improving

DeepLog, such as testing the efficiency of additional types of RNNs (recurrent neural networks) and integrating log data from various applications and systems.

Gopali et al. [7] compared the performance on RNN-based LSTM and CNN-based TCN models in the context of anomaly detection in a multivariate time series. The authors showed that the TCN models outperform other models with an F1 score of 0.92. The performance of the TCN-based model then was compared with the LSTM model and Graph Learning with Transformer for Anomaly detection (GTA) [11] models.

Luo et al. [12] proposed a convolutional LSTM-based Auto-Encoder (ConvLSTM-AE) framework for encoding appearance (i.e., motion) and change of appearance for detection of anomaly. In order to remember the visual shift that corresponds to motion, they utilized ConvNet for encoding each frame and a Convolutional LSTM (ConvLSTM), which is a type of LSTM that keeps the spatial information to record the change of appearance.

Ryota et al. [13] described a system that, by combining general and environment-specific information into a single framework, simultaneously identifies and recounts abnormal events. They used a large labeled dataset to train a Fast R-CNN-based model in order to gain general knowledge.

Radavliev and de Roure [14] present a set of optimized algorithms for the purpose of edge computing in which devices are often constrained with low memory. The optimized algorithms are then useful for autonomous environments such as robots or drones where self-adapting and self-evolving mechanisms are critical for continuous operations. In the context of time-series analysis and anomaly detection, the problem is to detect possible anomalies with minimal observations and thus prevent consuming memory capacity.

Gopali et al. [15] conducted a study in which TCN and LSTM models were analyzed and compared with an Average Stochastic Gradient Descent Weighted Dropped Long Short-Term Memory (AWD-LSTM) model in vulnerability detection in smart contracts, where the TCN model outperformed the other models in precision, recall, and F1 score by 95.95, 93.73, and 94.83, respectively.

3. Background

3.1. Long Short-Term Memory (LSTM)

Long-Term Short Memory is a popular innovative recurrent network architecture (RNN) in combination with an appropriate gradient-based learning algorithm [16]. LSTM can recognize long-term dependencies and forecast a sequence of events utilizing feedback links and loops in the network. The "long short-term memory" that the LSTM architecture attempts to give for RNNs can endure thousands of timestamps. The cells in the LSTM are repeated in a chain typically main layers or moduals.

Here, each cell has three interconnected layers, comprising a forget gate layer, an input gate layer, and an output gate layer that make up each cell in an LSTM model. Additionally, the gates are constructed using a sigmoid neural network layer and a point-wise multiplication operation. The only connections that the output layer has are from memory cells. Memory cells and gate units contain bias weights and accept input from input units, memory cells, and gate units. The forget gate is the initial stage of the procedure. Depending on the new input data and the prior concealed state, the forget gate determines what parts of long-term memory need to be forgotten at this time.

3.2. Bidirectional Long Short-Term Memory (BI-LSTM)

The original Real-Time Recurrent Learning (RTRL) and Back Propagation Through Time (BPTT) error gradient were employed in the LSTM training method of Gers et al. [17]. When two LSTMs are applied to the input data, the results are called deep-Bidirectional LSTMs [18]. An expanded BI-LSTM consists of two LSTM cells: a forward LSTM cell and a backward LSTM cell [19]. A Long Short-Term Memory (LSTM) is initially trained on the input sequence known as a forward layer. Second, the LSTM model is given a sequence that is the inverse of the original input sequence, which is called a backward layer. Finally, the forward and backward states may be concatenated to summarize BI-LSTM's output.

3.3. Temporal Convectional Networks (TCN)

For decades, convolutional networks have been used with sequences that were widely utilized for voice recognition [20]. Temporal Convolutional Networks (TCNs) were later used as a general convolutional sequence prediction architecture. TCNs are a form of time-series model that breaks through constrains by capturing long-term patterns with a hierarchy of temporal convolutional filters.

Two distinct kinds of TCNs were initially described by the pioneering authors in the work of Colin et al. [21]

(1) Encoder–Decoder TCN (ED-TCN) employs a temporal convolution, pooling, and upsampling architecture to effectively capture temporal patterns across large ranges of time.

(2) A Dilated TCN incorporates skip connections across layers and employs dilated convolutions in place of pooling and upsampling. It is similar to a convolution using a bigger filter built from the original filter by diluting it with zeros but with far better efficiency [22]. According to Bai et al. in their work [23], TCNs characteristics can be differentiated in following key respects: i.e., the architecture convolutions prevent "leakage" of information from the future to the past; and the architecture may translate an input sequence of any length to an output sequence of the same length.

3.4. CuDNN-LSTM

A CuDNNLSTM, according to the Keras documentation [24], is a fast LSTM implementation supported by CuDNN that only works with on a Graphics Processing Unit (GPU). CuDNN-LSTM is faster than LSTM but has fewer options because it lacks a dropout layer, custom activation function, and masking options.

4. Experimental Setup

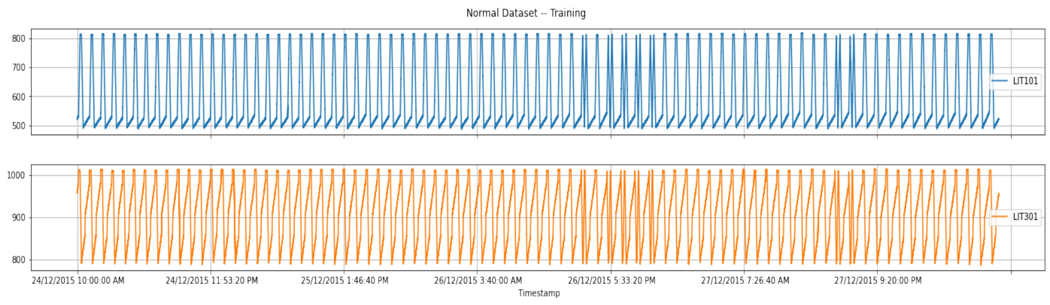
This section describes the data set, data preprocessing, the model architecture and the assessment metrics employed for the experiment conducted.

4.1. Dataset

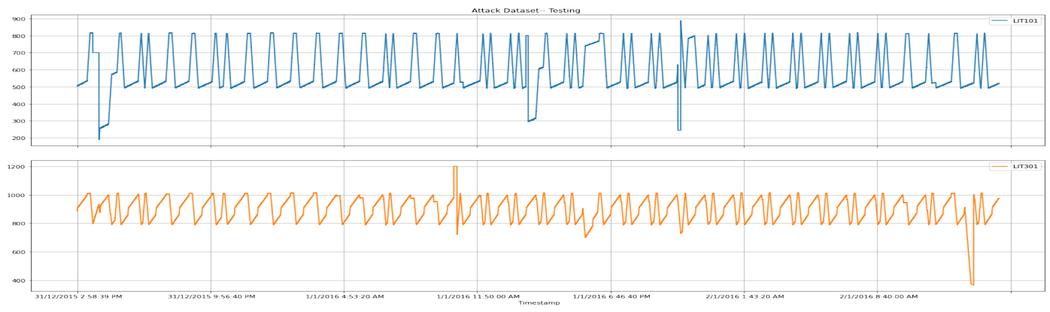
The SWaT dataset [25] (December–January, 2015–2016, dataset with anomaly) from iTrust, Centre for Research in Cyber Security, Singapore University of Technology and Design is used in this experiment. This dataset contains data from Secure Water Treatment (SWaT), which is a water treatment testbed for cybersecurity researchers aiming to collect and analyze data that help in creating secure Cyber Physical Systems. The dataset studied in this work is produced by typical IoT systems where sensors are used for controlling the water level of a water plant.

The dataset includes seven days (24 h) of SWaT running normally and 4 days with various attack scenarios (i.e., Single-Stage Single-Point Attacks, Single-Stage Multi-Point Attacks, Multi-Stage Single-Point Attacks, and Multi-Stage Multi-Point Attacks).

The dataset contains 53 network traffic features as well as values received from sensors and actuators categorized according to their normal or attack behavior. The experiment conducted in the paper focuses primarily on a Single-Stage Single-Point attack, in which only one sensor is targeted as the attack point. More specifically, the sensor data collected by "*LIT-101*" and "*LIT-301*" are studied as features for the experimentation. Figure 1 shows the time series of training and testing datsets for the sensors (i.e., *LIT-101* and *LIT-301*) studied. More over, for the comparison and discussion purpose, we also study other types of attacks such as Single Stage Multi-Point Attacks, Multi Stage Single-Point Attacks, and Multi Stage Multi-Point Attacks.



(**a**) Stream of normal data (Training sets).



(**b**) Stream of data with attacks (Testing sets).

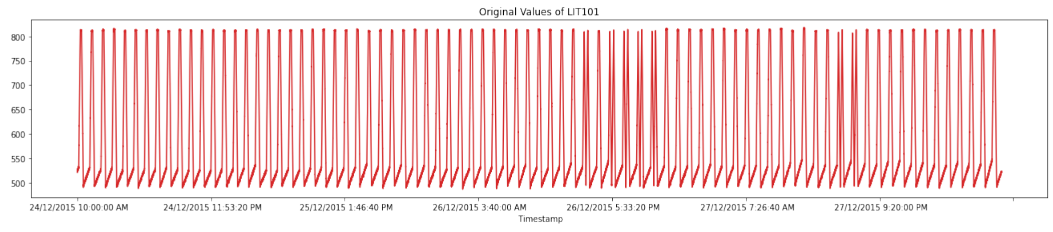
Figure 1. visualization of Training and Testing *LIT-101* and *LIT-301* data.

4.2. Data Preparation

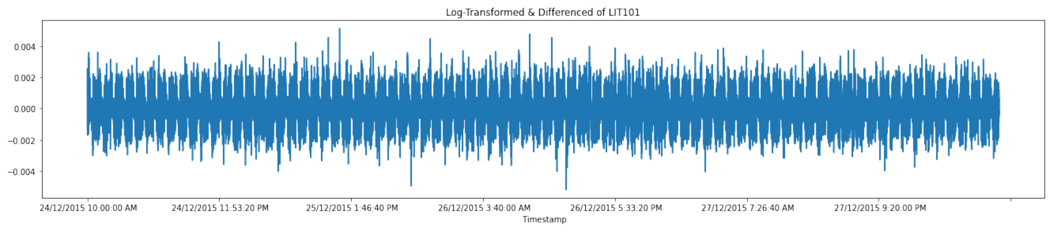
Figure 1a demonstrates the trends of the sensor data studied. The sensor features "*LIT-101*" and "*LIT-301*" of the SWat dataset in Figure 1a that have been selected from a pool of 53 features in the dataset. The sample data were drawn in the range of December 2015 to January 2016.

The training dataset contains 345,600 observations and spans four days, from 24 December 2015 10:00:00 a.m. to 28 December 2015 10:00:00 a.m. During model training and validation, 20% of the observations from the training set was used for validation. The testing set (demonstrated in Figure 1b) consists of 172, 800 observations with *Single-Stage Single-Point Attacks* in the selected features (i.e., "*LIT-101*" and "*LIT-301*") that spans two days (24 h) from 31 December 2015 2:58:39 p.m. to 2 January 2016 2:58:39 p.m.

After removing the trend and seasonality from the training dataset (i.e., the normal dataset in Figure 2a), the training and validation datasets in Figure 2b were prepared. Trends and seasonality in the time-series dataset may need to be removed before modeling. A time series is considered non-stationary if its mean or variance changes over time as a result of trends or seasonality, respectively. It is much simpler to mode in the stationary dataset because it has a stable mean and variance. The seasonality removal is performed by applying log-transformation and then taking differences between data. We have normalized the dataset in the preprocessing stage using the sklearn pythony library [26]. The normalization converts the dataset to a common scale in a range, typically 0 to 1, without distorting the difference in the range value. Figure 2 shows the time-series data of *LIT-101* after the transformation and removal of seasonality.



(**a**) Normal Dataset: "*LIT-101*".



(**b**) Log-Transformed and Differenced in "*LIT-101*".

Figure 2. Normal data after removing trend and seasonality "*LIT-101*".

4.3. Tools and Libraries Used

We used Python's pandas to import the dataset [27]. In the preprocessing stage, we utilized numpy [28] and sklearn [26]. For visualization, we used matplotlib [29]. The keras [24] with tensorflow [30] libraries were utilized in the models' building. The keras library used to build the model architecture of the deep learning models (i.e., RNN-based BI-LSTM, LSTM, CNN-based TCN, and CuDNN-LSTM). We used the skearn library to calculate the performance metrics. Figure 3 depicts the architecture of the models created. As the architecture of these models indicates, the models need to be consistent (i.e., a similar number of internal layers) so the comparison can be meaningful.

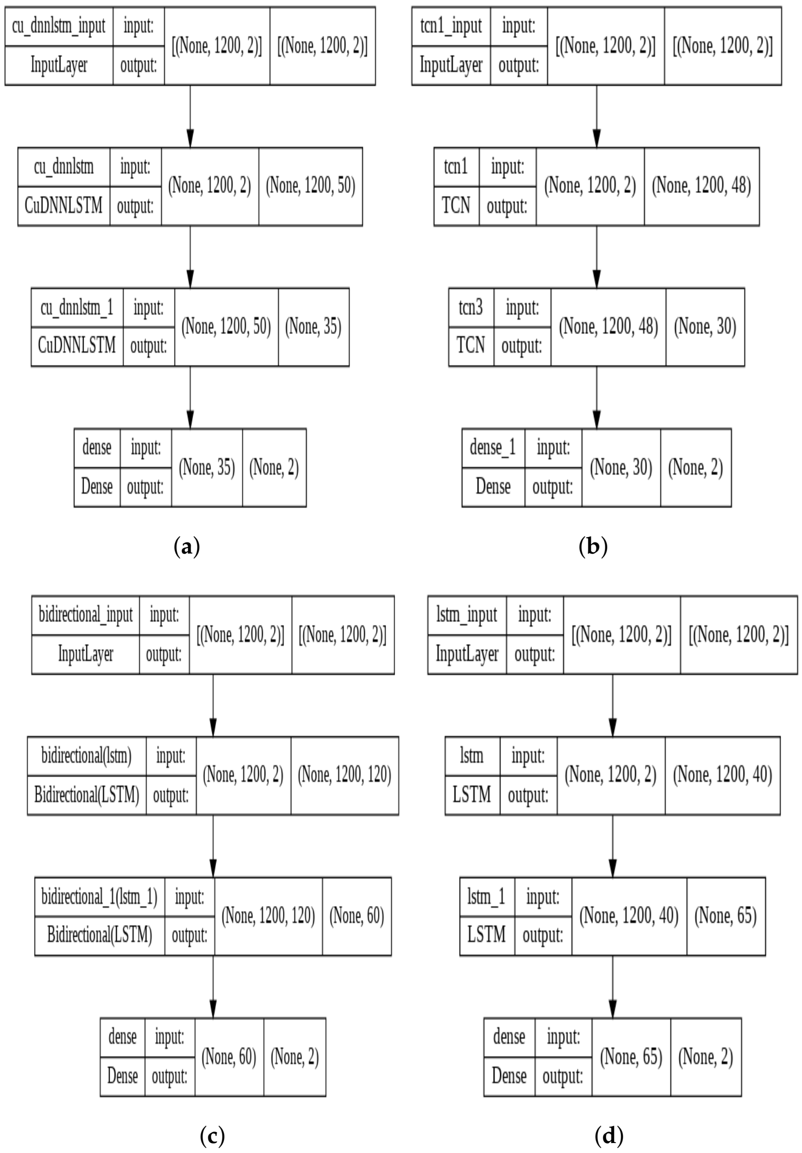

cu_dnnlstm_input | InputLayer | input: | output: | [(None, 1200, 2)] | [(None, 1200, 2)]
cu_dnnlstm | CuDNNLSTM | input: | output: | (None, 1200, 2) | (None, 1200, 50)
cu_dnnlstm_1 | CuDNNLSTM | input: | output: | (None, 1200, 50) | (None, 35)
dense | Dense | input: | output: | (None, 35) | (None, 2)
(a)
tcn1_input | InputLayer | input: | output: | [(None, 1200, 2)] | [(None, 1200, 2)]
tcn1 | TCN | input: | output: | (None, 1200, 2) | (None, 1200, 48)
tcn3 | TCN | input: | output: | (None, 1200, 48) | (None, 30)
dense_1 | Dense | input: | output: | (None, 30) | (None, 2)
(b)
bidirectional_input | InputLayer | input: | output: | [(None, 1200, 2)] | [(None, 1200, 2)]
bidirectional(lstm) | Bidirectional(LSTM) | input: | output: | (None, 1200, 2) | (None, 1200, 120)
bidirectional_1(lstm_1) | Bidirectional(LSTM) | input: | output: | (None, 1200, 120) | (None, 60)
dense | Dense | input: | output: | (None, 60) | (None, 2)
(c)
lstm_input | InputLayer | input: | output: | [(None, 1200, 2)] | [(None, 1200, 2)]
lstm | LSTM | input: | output: | (None, 1200, 2) | (None, 1200, 40)
lstm_1 | LSTM | input: | output: | (None, 1200, 40) | (None, 65)
dense | Dense | input: | output: | (None, 65) | (None, 2)
(d)

Figure 3. The architecture of models. (**a**) CudNN-LSTM Model Architecture. (**b**) TCN Model Architecture. (**c**) BI-LSTM Model Architecture. (**d**) LSTM Model Architecture.

4.4. Model Architecture

We build the deep learning models (i.e., cuDNN-LSTM, TCN, BI-LSTM and LSTM) using the keras python library [24] with TensorFlow [30] in backend to develop and train complex deep learning models.

The models are made up of two hidden layers, each of which has a proper number of neurons for the model based on observations made during the model training phase. The hidden layers process the data that are received at the input layer. The values of previous sensor data, i.e., 15, 20, and 30 min) are used to predict the current data (1 s) depending on the input size designated in the input layer.

The models utilized the activation function 'tanh' with the dropout rate of 0.3. The tanh function has the property that it can only achieve a gradient of 1 when the input value is 0. The activation function's primary benefit is that it generates zero-centered output, which facilitates the back-propagation process [31]. In order to prevent gradients from shifting in a specific direction, the activation function's output should be symmetric at zero, which is known as zero-centered output. The dropout rate prevents against overfitting and provides a way to approximately combine efficiently exponentially many different neural network architectures [32].

For the experiments conducted in this study, we have set the batch $size = 150$ and $epoch = 20$ for all models to maintain consistency. The models also utilized the loss function as the mean squared error and Adam [33] as the optimizer. The Adam optimizer is faster to compute and requires fewer parameters to tune the model.

For a fair comparison, it is essential to build models with similar configurations and structure. We build two models of TCN, LSTM and BI-LSTM, one with a dropout layer and the other one without a dropout layer. As a result, any optimization, in terms of the number of hidden layers, should be consistently applied to all models. In the following sections, we provide detailed information regarding the configuration of each model.

4.4.1. TCN

The TCN models are made up of two TCN hidden layers, one dropout layer and one dense layer. The first hidden layer contains 48 neurons with 3×3 kernel size and dilation (or steps) of 1 and 2, with 'tanh' activation function. The second hidden layer contains 30 neurons with 3×3 kernel size and dilation (or steps) of 1, 2 and 4, with 'tanh' activation function. The dropout layer contain a dropout rate of 0.3. The final (i.e., a dense layer) layer has two neurons as output with a 'tanh' activation function.In the TCN model without the dropout layer, the second TCN hidden layer has a dropout rate of 0.3, but the model does not have a dropout layer.

4.4.2. LSTM

The LSTM model consists of two LSTM hidden layers: one dropout layer and one dense layer. The two LSTM hidden layers contain 40 and 65 neurons, respectively, and follow with a 'tanh' activation function in each layer. Two neurons and the 'tanh' activation function are present in the final layer (also known as the dense layer). The LSTM model without a dropout layer lacks a dropout layer, while the second LSTM hidden layer has a dropout rate of 0.3.

4.4.3. BI-LSTM

The BI-LSTM model is made up with two Bidirectional LSTM hidden layers: one dropout layer and one dense layer. The neurons in the two Bidirectional LSTM layers are 60 and 30, respectively, and each layer has a 'tanh' activation function. The dropout layer has a 0.3 dropout rate. Two neurons with the "tanh" activation function are the output of the final (or dense layer) layer. The dropout layer is not included in the architecture in the BI-LSTM model with no dropout layer, but the other configurations remain the same.

4.4.4. CuDNN-LSTM

The cuDNN-LSTM model is implemented with cuda. The model has two cuDNN-LSTM hidden layers each with 50 and 35 neurons, respectively. The model has a final output (i.e., dense) layer containing two neurons implemented with a 'tanh' activation function.

4.5. Assessment Metrics

The Root Mean Square Error (RMSE) is a commonly used statistic for assessing the accuracy of a model's prediction. It calculates the discrepancies or residuals between actual and predicted values. The metric compares prediction errors of different models for a specific dataset.RMSE can be calculated through the following formula:

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\left(x_i - \hat{x}_i\right)^2}$$

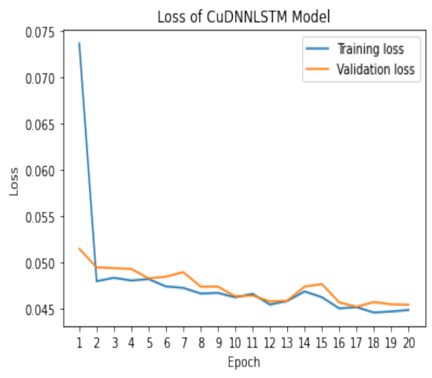
where the total number of observations is n; x_i is the true value, and $\hat{x}_i$ is the predicted one. The key advantage of utilizing RMSE is that it penalizes huge errors. It also scales the obtained results in the same units as the anticipated numbers, which is daily.

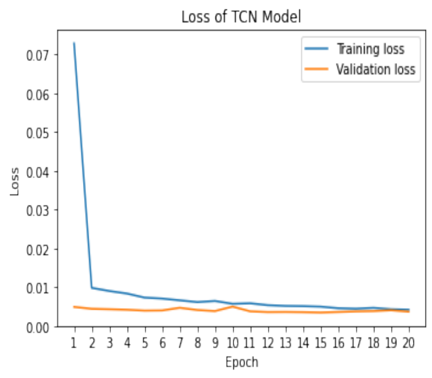
5. Results

The studies are carried out on the Google Colaboratory Pro Environment with Graphics Processing Unit (GPU) enabled. The results and predicted evaluation metrics are generated once the deep learning models (i.e., CuDNN-LSTM, BI-LSTM, LSTM and TCN) are trained.

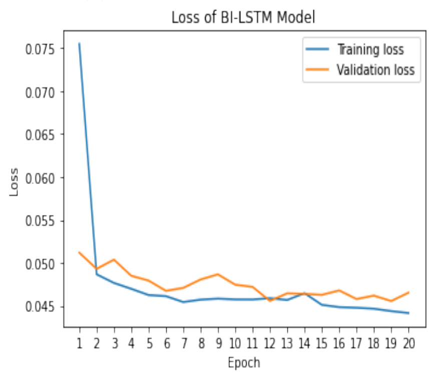
5.1. Loss vs. Epoch Plots

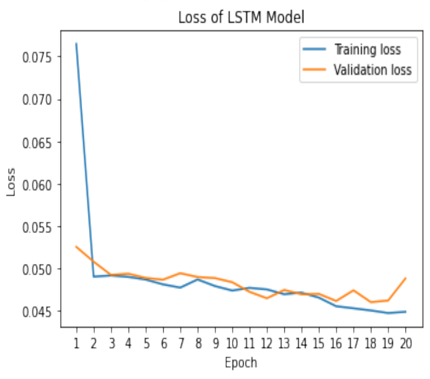
The plots depicted in Figure 4 show the trend of loss values for training and validation stages when building each model, where the x-axis is the epoch number and the y-axis is the loss value. As plots in Figure 4a,d, and Figure 4d demonstrate, these LSTM-based models exhibit similar trends in loss values. The loss values decrease sharply within the first few epochs for the training stage. While the loss value keeps decreasing, the amount of reduction does not seem to be substantial, and the loss values show some stability after a certain number of epochs. Similarly, the LSTM-based models show similar trends for loss values during the validation stage.





(**a**) CuDNN-LSTM Model (**b**) TCN Model





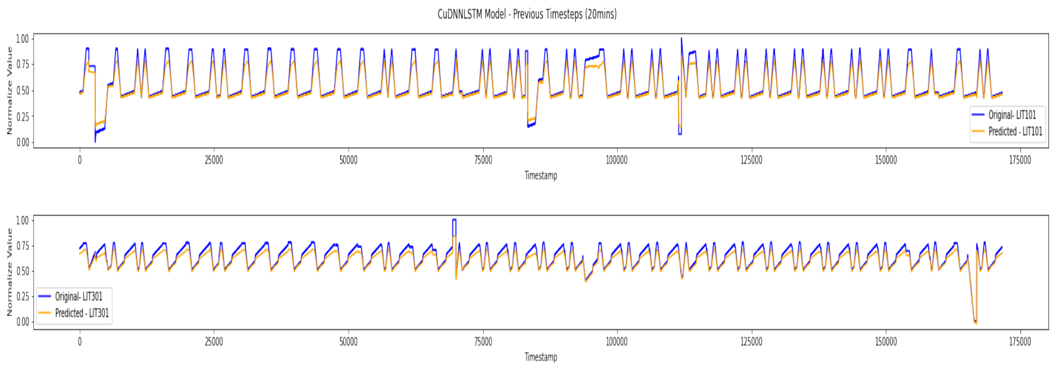
(**c**) BI-LSTM Model (**d**) LSTM Model

Figure 4. Training and Validation Loss of Models with Timestamp of 20 min.

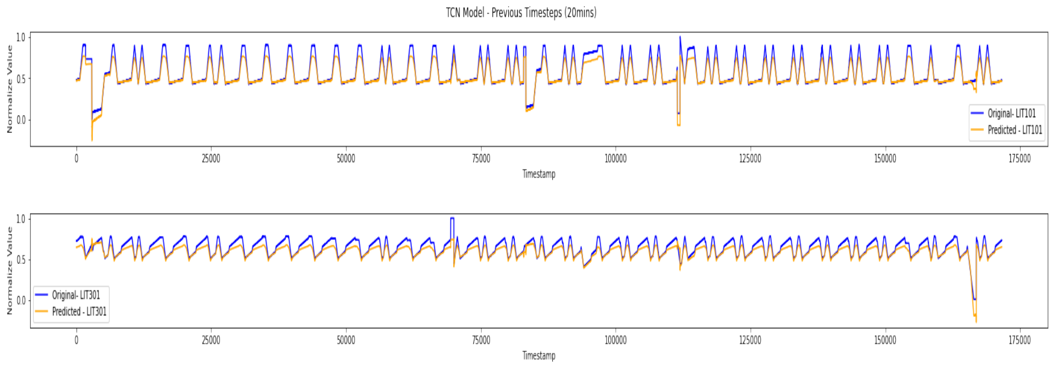
As the plot in Figure 4b depicts, the trend of loss values for the TCN-based model is somehow similar but also different than LSTM-based models. The TCN-based model demonstrates a similar pattern when it comes to a sharp reduction in loss values and the exhibition of stable loss values after a certain number of training and epochs. On the other hand and according to the y-axis of the plot in Figure 4b, the TCN-based model is able to be trained with lower values for loss values. These plots indicate that both LSTM-based and TCN-based models are exhibiting similar patterns in training and validation. However, the TCN-based model is capable of fitting a better model, which is evident by the lower loss values obtained.

5.2. Predicted vs. Actual Values

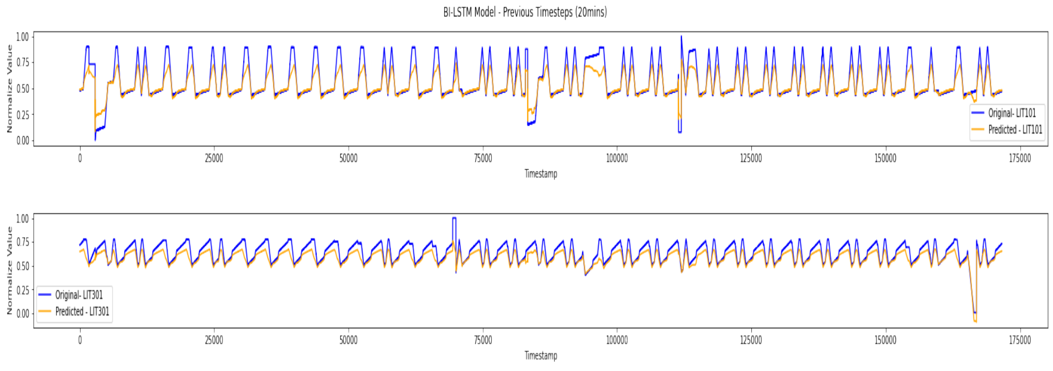
The plots given in Figure 5 demonstrate the predicted (colored in orange) and the actual (colored in blue) values for the sensor data studied and obtained by each model.



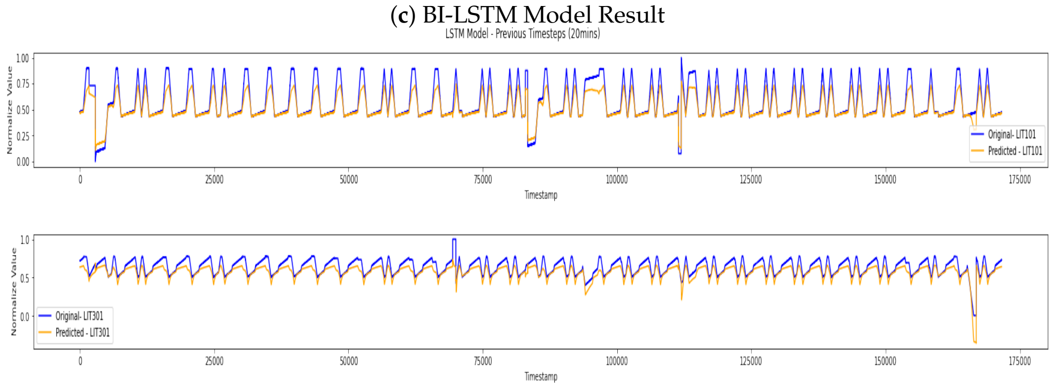
(**a**) CuDNN-LSTM Model Result



(**b**) TCN Model Result



(**c**) BI-LSTM Model Result



(**d**) LSTM Model Result

Figure 5. The Original and Predicted Value of Models.

Table 1 lists the training time and RMSE values for the models. As we observe in the table, the CuDNN-LSTM model builds a more accurate model with RMSE values of 0.035, 0.041, 0.043, 0.041, 0.043 and 0.049 for timestamps of 30, 20, 15, 10, 7 and 5, respectively. On the other hand, the TCN-based model seems to offer building a comparable model with reasonable RMSE but lower training time. More specifically, the TCN-based models are able to train a reasonably accurate model with RMSE values of 0.065, 0.66, 0.061, 0.057, 0.058 and 0.069 with lower time allocated for training for timestamps of 30, 20, 15, 10, 7 and 5 min, respectively.

Table 1. Training time/RMSE for 30, 20, 15, 10, 7 and 5 min timestamps.

Timestamps (s/min)	Model	Training Time H:M:S	RMSE
1800/30	CuDNN-LSTM without Dropout Layer	0:21:24	**0.035**
	TCN without Dropout Layer	**0:19:20**	0.065
	TCN with Dropout Layer	**0:19:30**	0.050
	BI-LSTM without Dropout Layer	0:36:24	0.072
	BI-LSTM with Dropout Layer	0:33:28	0.066
	LSTM without Dropout Layer	0:23:24	0.067
	LSTM with Dropout Layer	0:27:53	0.071
1200/20	CuDNN-LSTM without Dropout Layer	0:14:46	**0.041**
	TCN without Dropout Layer	**0:12:27**	0.066
	TCN with Dropout Layer	**0:13:04**	0.045
	BI-LSTM without Dropout Layer	0:25:17	0.075
	BI-LSTM with Dropout Layer	0:21:46	0.055
	LSTM without Dropout Layer	0:16:24	0.065
	LSTM with Dropout Layer	0:19:25	0.076
900/15	CuDNN-LSTM without Dropout Layer	0:11:09	**0.043**
	TCN without Dropout Layer	**0:10:14**	0.061
	TCN with Dropout Layer	**0:10:12**	0.043
	BI-LSTM without Dropout Layer	0:20:02	0.069
	BI-LSTM with Dropout Layer	0:16:53	0.059
	LSTM without Dropout Layer	0:11:58	0.057
	LSTM with Dropout Layer	0:14:48	0.073
600/10	CuDNN-LSTM without Dropout Layer	0:08:34	**0.041**
	TCN without Dropout Layer	**0:07:37**	0.057
	TCN with Dropout Layer	**0:06:35**	0.056
	BI-LSTM without Dropout Layer	0:11:43	0.054
	BI-LSTM with Dropout Layer	0:09:57	0.068
	LSTM without Dropout Layer	0:08:32	0.066
	LSTM with Dropout Layer	0:08:24	0.060
420/7	CuDNN-LSTM without Dropout Layer	0:06:44	**0.043**
	TCN without Dropout Layer	**0:06:00**	0.058
	TCN with Dropout Layer	**0:05:06**	0.051
	BI-LSTM without Dropout Layer	0:08:45	0.063
	BI-LSTM with Dropout Layer	0:08:55	0.052
	LSTM without Dropout Layer	0:06:24	0.063
	LSTM with Dropout Layer	0:06:23	0.067
300/5	CuDNN-LSTM without Dropout Layer	0:05:06	**0.049**
	TCN without Dropout Layer	**0:04:26**	0.069
	TCN with Dropout Layer	**0:04:44**	0.076
	BI-LSTM without Dropout Layer	0:06:38	0.062
	BI-LSTM with Dropout Layer	0:06:46	0.052
	LSTM without Dropout Layer	0:04:53	0.070
	LSTM with Dropout Layer	0:05:01	0.071

6. Discussion

6.1. Anomaly Detection Methodology

A Python tool called Anomaly Detection Toolkit (ADTK) [34] is used for unsupervised/rule-based time-series anomaly detection. The SWaT dataset [25] has an outlier anomaly known as a spike, in which the value quickly increases or decreases to a level that is off the range of the recent past. To identify the spikes anomaly in the experiments, we employed the `PersistAd` model from the ADTK. The `PersistAD` model's parameter `c` takes an optional float value to determine the range using a historical inter-quartile range, and it is often set to 1.5. Likewise, the `side` parameter is set to `both` and `window` set as the timestamps with the goal of observing the longer lookback.

6.2. Analysis of Different Types of Attacks

6.2.1. Single Stage Single-Point Attacks

In its simple case, the "Single-Stage Single-Point Attack" can demonstrate the performance of prediction without interfering with the complexity of attack types. The plots in Figure 6a illustrate the values of F1 scores captured for different values of timestamps for each model created and studied. As the figure shows, the models exhibit similar patterns. The F1 scores vary between 0.80 and 0.90. While the best performance is exhibited when the timestamp is 400, the F1 scores show steady and constant improvement for timestamp values greater than 1200.

6.2.2. Single Stage Multi-Point Attacks

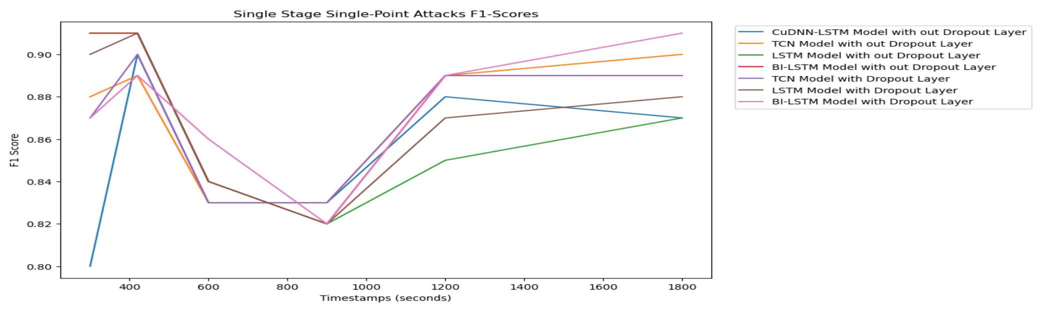
In the Single-Stage Multi-Point Attack, the sensor features "*P-101*" and "*P-102*" have been taken for the experiments. Based on the experiments' outcomes, the plot shown in Figure 7a depicts that the CuDNN-LSTM model without the dropout layer has the lowest RMSE across all timestamps (i.e., 30, 20, 15, 10, 7 and 5 min). In contrast, the TCN-based model with the dropout layer has significant RMSE throughout the timestamp range, with timestamps of 10 min having the highest RMSE of 0.39. The plotted F1 score in Figure 6b shows that the BI-LSTM model with the dropout layer has an F1 score of 0.99 at timestamps 20 and 15 min, and it has the higher F1 scores throughout all timestamps. The plots clearly demonstrate that all models have the lowest F1 scores at timestamps of 10 min.

6.2.3. Multi-Stage Single-Point Attacks

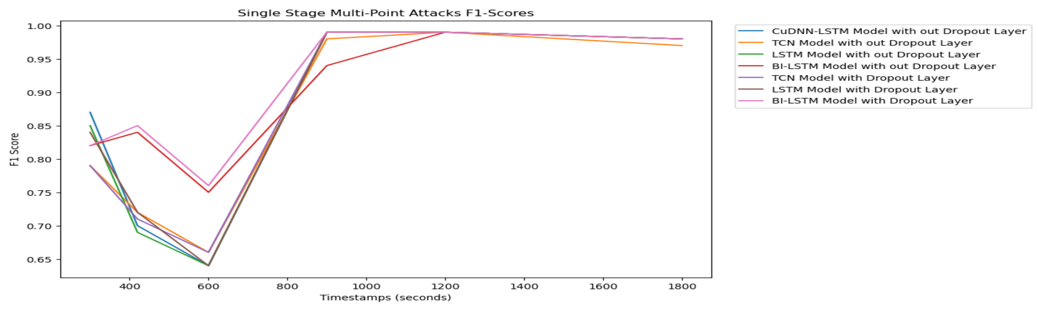
In the Multi-Stage Single-Point Attacks, the sensor features "*AIT-402*" and "*AIT-502*" are utilized for the experiment. The TCN model has the highest RMSE across all of the timestamps, while the BI-LSTM model with no dropout layer has the lowest RMSE, which is demonstrated in the plot shown in Figure 7b. The TCN model has the highest RMSE, 0.62, at a timestamp of 6 min, whereas the BI-LSTM model without a dropout layer has the lowest RMSE, 0.24, at 7 min timestamps. According to the plot shown in Figure 6c, the performance of the TCN model with a dropout layer has the lowest F1 score across the timestamps, whereas the lowest F1 score of 0.87 is recorded at the 30 min timestamp. In contrast, the BI-LSTM model without a dropout layer has the higher F1 scores throughout the timestamps.

6.2.4. Multi-Stage Multi-Point Attacks

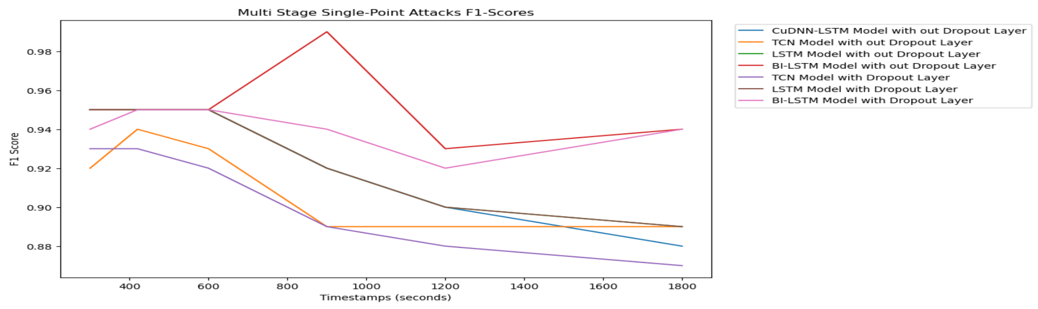
In the Multi-Stage Multi-Point Attacks, the sensor features "*L-101*", "*P-101*" and "*MV-201*" have been utilized for the experiments. The plot in Figure 7c shows that the CuDNN-LSTM model consistently has the lowest RMSE of 0.57 at throughout the timestamps, while the TCN-based model with the dropout layer has the larger RMSE throughout the timestamps (i.e., 30, 20, 15, 10, 7 and 5 min). The TCN model with the dropout layer recorded the highest RMSE of 0.82 at 30 min timestamps. The BI-LSTM Model with the dropout layer has higher F1 scores across all timestamps, as shown by the plot shown in Figure 6d. The plot demonstrates that for timestamps within 10 min, all the model F1 scores are lowest, with the highest F1 scores being 0.99 and the lowest being 0.65.



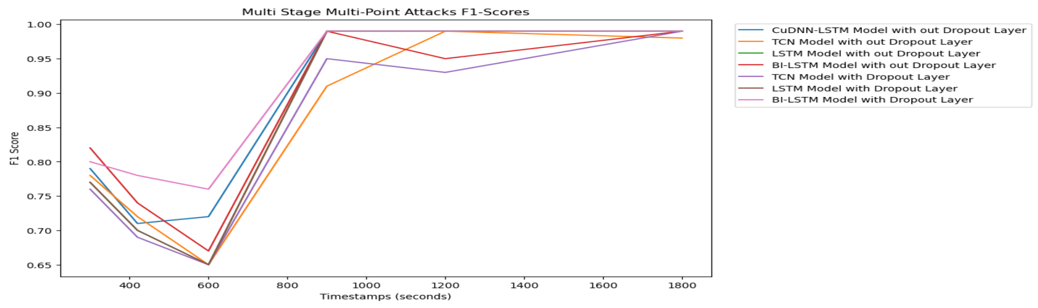
(**a**) Single-Stage Multi-Point Attacks F1 Scores



(**b**) Single-Stage Multi-Point Attacks F1 Scores

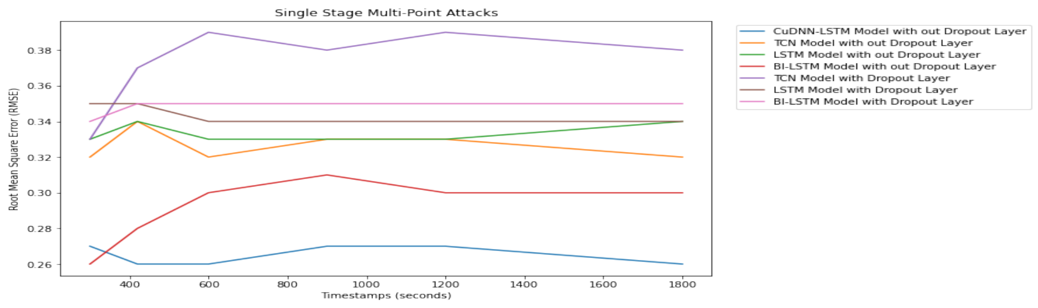


(**c**) Multi-Stage Single-Point Attacks F1 Scores

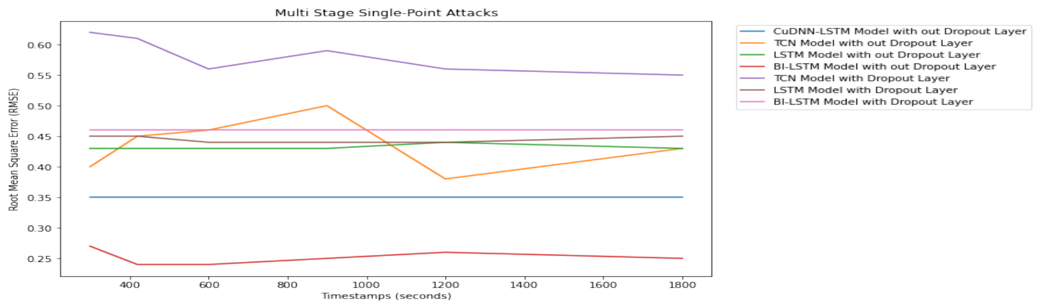


(**d**) Multi-Stage Multi-Point Attacks F1 Scores

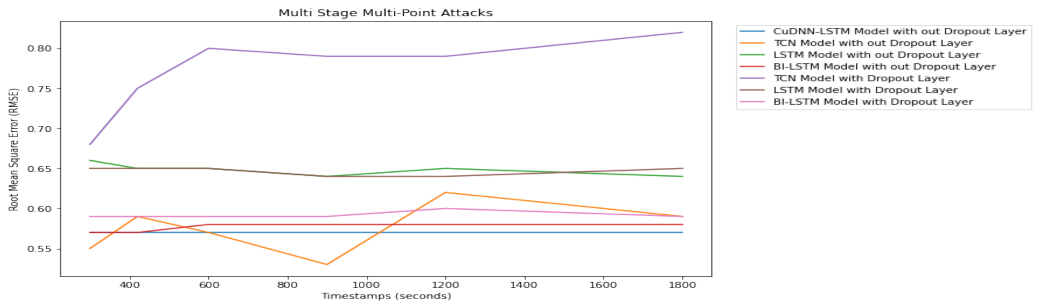
Figure 6. Single-Stage Single-Point Attacks, Single-Stage Multi-Point Attacks, Multi-Stage Single-Point Attacks and Multi-Stage Multi-Point Attacks F1 Scores.



(**a**) Single-Stage Multi-Point Attacks Result



(**b**) Multi-Stage Single-Point Attacks Result



(**c**) Multi-Stage Multi-Point Attacks Result

Figure 7. Single-Stage Multi-Point Attacks, Multi-Stage Single-Point Attacks and Multi-Stage Multi-Point Attacks Results.

6.3. The Effect of Timestamp on Training Time and Performance

The timestamp parameter enables the models to look back and learn from data up to some certain points. It is important to build more accurate models. As a result, it makes sense to identify the intervals of timestamps where the models performed the best.

Table 1 provides a numerical comparison of RMSE values for the models built for various values of timestamps. A glance at the RMSE values indicates that the greater the timestamp values, the smaller the RMSE values will be. For instance, the values of CuDNN-LSTM without a dropout layer show that the RMSE values are 0.035 and 0.049 when timestamps are 1800 and 200, respectively.

On the other side, we also naturally expect that the training time increases when the timestamp value (i.e., look-back) increases. For instance, the values of CuDNN-LSTM without a dropout layer show that the training times are 21 minutes, and 5 minutes when timestamps are 1800 and 200, respectively.

Figure 6 also depicts the F1 scores for different types of attacks for the models built when various values of timestamps are studied. In most cases, we observe that by increasing the timestamp values, the models are improved and F1 scores are increased. The only attack type that behaves differently than the others is the case of "Single-Stage Multi-Point Attacks" where we observe a slight reduction in F1 scores. Given that the reduction in F1 scores is not substantial, we conclude that overall, the greater timestamps lead to a better performance in terms of model accuracy but longer training time.

7. Conclusions and Future Work

The Internet of Things (IoT) enables connecting hundreds of devices together. These types of interconnected devices often capture data from their local environment and exchange them with some other devices to accomplish certain tasks cooperatively. Due to the rise in these types of systems, more and more industries such as manufacturing and energy are adapting them to serve their customers. On the other side, these IoT-based systems are also becoming the target points for cyber attackers with the goal of tampering data and thus malfunctioning the operations. As a result, it is salient to detect any anomalies as early as possible so the damage is prevented or minimized.

There is a good number of anomaly detection techniques that are applied to addressing this problem. With the advent of machine and deep learning and with respect to the amount of data captured through the communications of the interconnected devices, researchers have explored these data-oriented approaches to model and detect anomalies.

This paper compares some of the machine/deep learning techniques and reports their performance. More specifically, the paper studies the performance of a number of variations of LSTM-based models, such as RNN-based models, in comparison with a CNN-based model, which is called a TCN. The deriving motivation is the basic differences in how these two types of models (CNN and RNN) perform.

According to our results, we observe that TCN performs relatively well in detecting anomalies with relatively low RMSE. The key aspect of TCN is that it needs lower training time in comparison with the RNN counterparts. On the other hand, we observe that the fast version of LSTM performs the best in terms of accuracy, but it needs a greater amount of time for training.

The analysis performed and results reported here in this paper are not explicitly recommended, in which cases these two types of neuron networks should be employed. However, it suggests that when accuracy can be sacrificed by some small amount, TCN can be utilized, since it requires lower training time. As a future direction, it is necessary to compare the results with different datasets with various volumes and compare the trade-off between these two models.

Author Contributions: S.G. implemented the algorithms and collected the data. A.S.N. contributed to manuscript development and analysis of the data collected. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Science Foundation (NSF) under Grant No: 1821560.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The SWaT Dataset used and studied in this work are publicly available.

Acknowledgments: This work was supported by the National Science Foundation (NSF) under Grant No: 1821560.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ashton, K. That 'internet of things' thing. *RFID J.* **2009**, *22*, 97–114.
2. Vailshery, L.S. Internet of Things Spending Worldwide 2023 | Statista. Available online: https://www.statista.com/statistics/668996/worldwide-expenditures-for-the-internet-of-things/ (accessed on 28 August 2022).
3. Sava, J.A. Global Number of IOT Attacks 2021. Available online: https://www.statista.com/statistics/1322216/worldwide-internet-of-things-attacks/(accessed on 28 August 2022).
4. Luo, Y.; Xiao, Y.; Cheng, L.; Peng, G.; Yao, D. Deep learning-based anomaly detection in cyber-physical systems: Progress and opportunities. *ACM Comput. Surv. (CSUR)* **2021**, *54*, 1–36. [CrossRef]

5. Sensarma, D.; Sarma, S.S. A survey on different graph based anomaly detection techniques. *Indian J. Sci. Technol.* **2015**, *8*, 1–7. [CrossRef]
6. Zhao, H.; Wang, Y.; Duan, J.; Huang, C.; Cao, D.; Tong, Y.; Xu, B.; Bai, J.; Tong, J.; Zhang, Q. Multivariate Time-Series Anomaly Detection via Graph Attention Network. In Proceedings of the 2020 IEEE International Conference on Data Mining (ICDM), Sorrento, Italy, 17–20 Novermber 2020; pp. 841–850. doi: 10.1109/ICDM50108.2020.00093. [CrossRef]
7. Gopali, S.; Abri, F.; Siami-Namini, S.; Namin, A.S. A Comparison of TCN and LSTM Models in Detecting Anomalies in Time Series Data. In Proceedings of the 2021 IEEE International Conference on Big Data (Big Data), IEEE, Orlando, FL, USA, 15–18 December 2021; pp. 2415–2420.
8. Chalapathy, R.; Chawla, S. Deep learning for anomaly detection: A survey. *arXiv* **2019**, arXiv:1901.03407.
9. Wu, D.; Jiang, Z.; Xie, X.; Wei, X.; Yu, W.; Li, R. LSTM learning with Bayesian and Gaussian processing for anomaly detection in industrial IoT. *IEEE Trans. Ind. Inform.* **2019**, *16*, 5244–5253. [CrossRef]
10. Du, M.; Li, F.; Zheng, G.; Srikumar, V. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1285–1298.
11. Ahmed, C.M.; Palleti, V.R.; Mathur, A.P. WADI: A water distribution testbed for research in the design of secure cyber physical systems. In Proceedings of the International Workshop on Cyber- Physical Systems for Smart Water Networks, Pittsburgh, PA, USA, 21 April 2017; pp. 25–28.
12. Luo, W.; Liu, W.; Gao, S. Remembering history with convolutional lstm for anomaly detection. In Proceedings of the 2017 IEEE International Conference on Multimedia and Expo (ICME), Hong Kong, China, 10–14 July 2017; pp. 439–444.
13. Hinami, R.; Mei, T.; Satoh, S. Joint detection and recounting of abnormal events by learning deep generic knowledge. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 3619–3627.
14. Radanliev, P.; De Roure, D. Review of algorithms for artificial intelligence on low memory devices. *IEEE Access* **2021**, *9*, 109986–109993. [CrossRef]
15. Gopali, S.; Khan, Z.A.; Chhetri, B.; Karki, B.; Namin, A.S. Vulnerability Detection in Smart Contracts Using Deep Learning. In Proceedings of the 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), Los Alamitos, CA, USA, 27 June–1 July 2022; pp. 1249–1255. doi: 10.1109/COMPSAC54236.2022.00197. [CrossRef]
16. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
17. Gers, F.A.; Schraudolph, N.N.; Schmidhuber, J. Learning precise timing with LSTM recurrent networks. *J. Mach. Learn. Res.* **2002**, *3*, 115–143.
18. Schuster, M.; Paliwal, K.K. Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.* **1997**, *45*, 2673–2681. [CrossRef]
19. Xu, G.; Meng, Y.; Qiu, X.; Yu, Z.; Wu, X. Sentiment analysis of comment texts based on BiLSTM. *IEEE Access* **2019**, *7*, 51522–51532. [CrossRef]
20. Waibel, A.; Hanazawa, T.; Hinton, G.; Shikano, K.; Lang, K.J. Phoneme recognition using time-delay neural networks. *IEEE Trans. Acoust. Speech Signal Process.* **1989**, *37*, 328–339. [CrossRef]
21. Lea, C.; Flynn, M.D.; Vidal, R.; Reiter, A.; Hager, G.D. Temporal convolutional networks for action segmentation and detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 156–165.
22. Oord, A.v.d.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; Kavukcuoglu, K. Wavenet: A generative model for raw audio. *arXiv* **2016**, arXiv:1609.03499.
23. Bai, S.; Kolter, J.Z.; Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv* **2018**, arXiv:1803.01271.
24. Chollet, F.K. Keras. Available online: https://keras.io (accessed on 1 August 2022).
25. iTrust. Datasets. Available online: https://itrust.sutd.edu.sg/itrust-labs_.datasets/ (accessed on 1 August 2022).
26. Buitinck, L.; Louppe, G.; Blondel, M.; Pedregosa, F.; Mueller, A.; Grisel, O.; Niculae, V.; Prettenhofer, P.; Gramfort, A.; Grobler, J.; et al. API design for machine learning software: Experiences from the scikit-learn project. In Proceedings of the ECML PKDD Workshop: Languages for Data Mining and Machine Learning, Prague, Czech Republic, 23–27 September, 2013; pp. 108–122.
27. McKinney, W. Data Structures for Statistical Computing in Python. In Proceedings of the 9th Python in Science Conference, Austin, TX, USA, 28 June–3 July 2010; van der Walt, S., Millman, J., Eds.; Creative Commons Attribution: Mountain View, CA, USA, 2010; pp. 56–61; [CrossRef]
28. Harris, C.R.; Millman, K.J.; van der Walt, S.J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N.J.; et al. Array programming with NumPy. *Nature* **2020**, *585*, 357–362. doi: 10.1038/s41586-020-2649-2. [CrossRef] [PubMed]
29. Hunter, J.D. Matplotlib: A 2D graphics environment. *Comput. Sci. Eng.* **2007**, *9*, 90–95. doi: 10.1109/MCSE.2007.55. [CrossRef]
30. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Available online: tensorflow.org (accessed on 1 August 2022).
31. Nwankpa, C.; Ijomah, W.; Gachagan, A.; Marshall, S. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv* **2018**, arXiv:1811.03378.
32. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
33. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2014**, arXiv:1412.6980.
34. Anomaly Detection Toolkit. Available online: https://github.com/arundo/adtk (accessed on 2 September 2022).