LSTM Deep Recurrent Neural Network Model for Voltage Abnormality Detection at IoT Gateway

Jihad Qaddour
School of IT
Illinois State University

Navid Rajabi
School of IT
Illinois State University

ABSTRACT
In this paper, we have trained and tuned a Deep Neural Network (DNN) model to extract the household's average voltage pattern and utilize it for predicting any potential abnormalities using Artificial Intelligence (AI). To be more specific, this model is a Recurrent Neural Network (RNN) that leverages the Long Short-Term Memory (LSTM) cell as a long-term dependency storage. The logic behind this architecture is to enhance the performance as well as solving the vanishing (and exploding) gradient problem of RNNs. The main objective of this project is to detect abnormalities at the edge (Internet of Thing (IoT) gateway) as a security-oriented proactive approach. Therefore, after training the model with a large-enough train part of the dataset and optimizing it after running the test part of the dataset, the model can be placed at the IoT gateway and recognize the future abnormalities in Time-series voltage dataset of any given household to avoid any potential malicious activity or damage.

General Terms
Artificial Intelligence (AI). Recurrent Neural Network (RNN). Long Short-Term Memory (LSTM). Internet of Thing (IoT). Deep Neural Network (DNN).

Keywords
LSTM; RNN; DNN; IoT; AI; Deep Learning.

1. INTRODUCTION
In recent years, Artificial Intelligence (AI) has been recognized as one of the most influencing trends on a broad spectrum of existing and/or emerging technologies/areas. The most important aspect of AI is the attempt to help solve real-world problems. As an instance, analyzing the patient's medical images and providing a diagnosis with equal to medical doctors, or even better than them. According to the Stanford ML Group's MURA Bone X-Ray Deep Learning Competition, ("MURA (musculoskeletal radiographs) is a large dataset of bone X-rays. Algorithms are tasked with determining whether an X-ray study is normal or abnormal." [2]), all the six most recent developed models (including the most recent one which is uploaded on November 6, 2018) have proved to have a better performance than the Best Radiologist Performance at Stanford University! This is only one of the effective results that have been generated by DNNs in beating human's accuracy in performing high-level tasks. Therefore, there is a significant positive sign which grabs our attention to take DNNs more seriously. Therefore, we've leveraged this advantage to intensify the security of smart buildings.

According to the references like "Smart Cities: Foundations, Principles, and Applications" and "Security and Privacy in Cyber-Physical Systems: Foundations, Principles, and Applications", load altering/manipulation and/or destabilizing voltage attacks that mainly happens with manipulating the voltage magnitude in form of overflow/overload that can cause highly serious problems to the smart cities power system, like increase in flow current (equipment damages), out-of-range deviation of the system frequency, power shortage (blackout/downtimes during the critical moments), disguising the malicious control activities during the blackout, etc. In addition, the sensitivity and fragility of IoT devices to the sudden voltage fluctuations, out-of-range frequency deviations, and consequently the flow current issues, underlines the significance of conducting this research project.

The rest of the paper is organized in seven sections: Section II covers deep recurrent neural networks; Section III presents long short-term memory; Section IV presents proposed model. Section V discusses dataset description. Section VI presents model's performance. Lastly, we have the conclusion and the future work presented in VII.

2. DEEP RECURRENT NEURAL NETWORK
Deep Recurrent Neural Networks (DRNN) are a subset of the typical Deep Neural Networks (DNN) that are being used for analyzing sequential data. In general, Feedforward Neural Networks are the most popular type of DNNs, as shown in Figure 1.

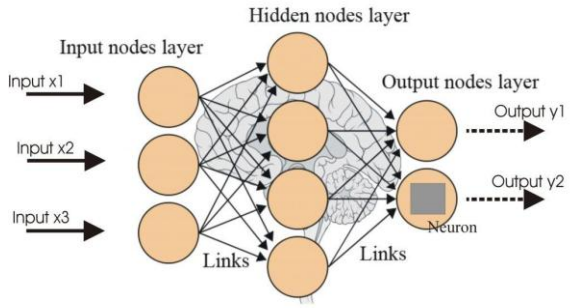

Figure 1. A Generic Neural Network [9]

Feedforward Neural Networks consist of three major stages, called Feedforward, Backpropagation and Iterative Weight Parameter Update:

A. Feedforward: In this stage, the model will be provided with a batch of training data set and calculates the loss function, using the actual values and the predicted values.

B. Backpropagation: In this stage, the model generates the gradient of the loss function by considering the initial weights.

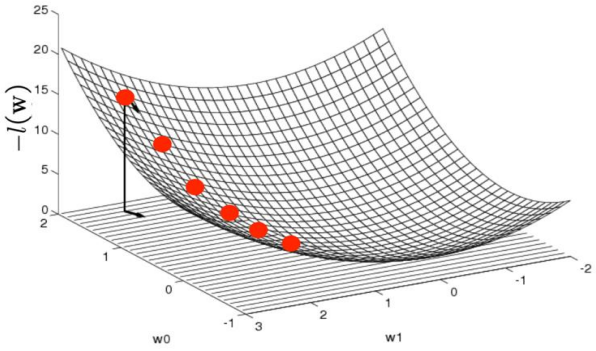

Figure 2. Gradient Descent Convex Optimization [8]

Considering the l(w) as the Mean Square Error (MSE) function, the weights updates will be computed using the following equations to reach the local minimum of the gradient and optimize the convex [8]:

Update rule:

$$\Delta w = \eta \nabla w l(w) \quad (1)$$

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \frac{\partial l(w)}{\partial w_i} \quad (2)$$

Where η is learning rate

RNNs are well-known for sequential processing and time-series analysis. The main concept of RNNs is using a feedback loop at each recurrent layer to store the data as a memory for future predictions. However, the main problem is that this architecture won't have a high level of performance for long-term use cases. So, RNNs suffers from the vanishing (and exploding) gradient issue. As mentioned earlier, this problem comes from the lack of long-term dependency saving module. The main intuition behind the RNNs is that previous hidden and input states, in addition to the current input values, generate the next hidden and output values. This idea was an advancement to the naïve idea of the initial RNNs that was only based on generating the hidden stage using the input, and the output stage based in the hidden layer. As mentioned above, the main architecture of RNNs can be as the following Figure 3.

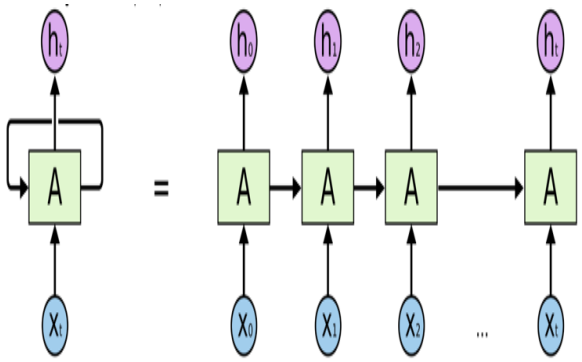

Figure 3. RNN Principle Architecture [7]

The "x" represents the input of each step, the "h" represents the output of each state and the result of each state will be given to the next step. However, the RNN cannot have a satisfactory performance in the situations that the dataset is huge. The reason of not being efficient in those situations is that the model is not capable of remembering the long-term dependencies for a long period of time. As shown in Figure. 4, when to reaches a large enough number (assuming infinity), it cannot remember what was going on in the earliest stages as shown in Figure 4.

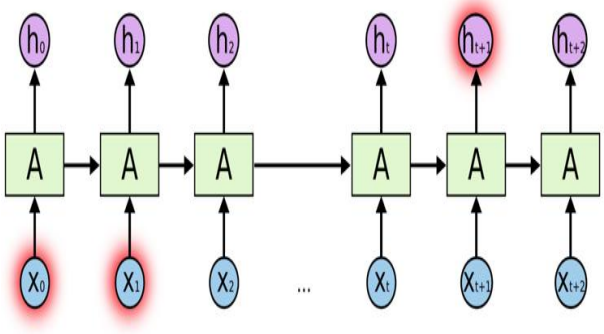

Figure 4. The vanishing problem of RNN [7]

Let's explain the vanishing/exploding gradient problem, which is a prevalent issue in RNNs in-depth. This issue happens when the loss function cannot reach the local minimum, by continuous derivative computations of the gradient descent to update the weight parameters. This implicitly indicates that the model is not representing an acceptable performance, since the actual values and the predicted values are diverging, instead of converging. Consequently, the square of their difference is evading the "0" value (which is the absolute optimum value) after each iteration as shown in Figure 5.

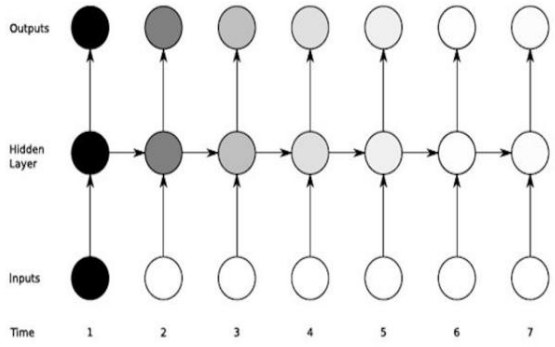

Figure 5. Vanishing gradient problem [7]

3. LONG SHORT-TERM MEMORY

The RNN's lack of performance was the main incentive for the AI scientists to use an LSTM cell unit in combination with the RNN and propose the LSTM RNN model. In despite of the RNN that used to calculate the next hidden state (h) only based on a simple combination of the current input (x) as follows:

$$h_t = \phi\left(W_{x_t} + U h_{t-1}\right) \quad (3)$$

The architecture of the LSTM RNN leverages a more complicated combination of four mini brains (sub-mechanism) to implement the remembering, forgetting and attention capabilities. By the definition, the long-term memory (ltm) is called the "Cell State".

The working memory is called the "Hidden State" (which is like the traditional (vanilla) RNNs). The remember vector is called the "Forget State" (in which '1' means to keep and '0' means to forget).

The save vector is called the "Input gate" and the focus vector

is called the "Output gate". As shown in the following formulas, the remember, save and focus functions are implemented using the "Sigmoid" function but the "ltm" function has been implemented using the "Tanh" function [7].

$$ltm_t = remember_t \circ ltm_{t-1} + save_t \circ ltm'_t \quad (4)$$

$$wm_t = focus_t \circ \tanh(ltm_t) \quad (5)$$

$$remember_t = \sigma(w_r x_t + U_r wmt_{t-1}) \quad (6)$$

$$save_t = \sigma(w_s x_t + U_s wmt_{t-1}) \quad (7)$$

$$focus_t = \sigma(w_f x_t + U_f wmt_{t-1}) \quad (8)$$

$$ltm'_t = tanh(w_l x_t + U_l wmt_{t-1}) \quad (9)$$

According to the above-mentioned formulas of the LSTM model, the combined architecture can be presented as the following:

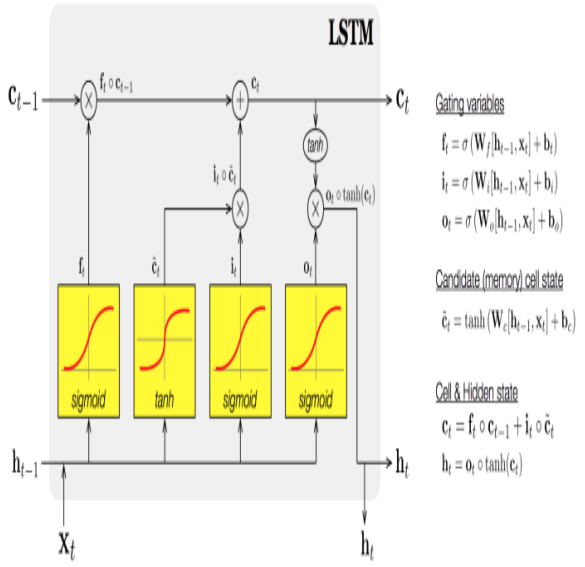

Figure 6. LSTM module's detailed architecture [7]

4. PROPOSE MODEL

We've trained a DNN model which includes two LSTM layers (including 100 neurons each), two Dropout layers, and a Dense layer [10].

In this model, we've used Mean Squared Error (MSE) loss function for computing the accuracy of the model and ADAM for optimization. As the final layer, the Dense layer is using the Linear activation function to generate the final output. The architecture that has been generated by the Keras Deep Learning framework and the Tensor Flow Backend is shown here:

The model uses the voltage data including 5000 and splits it to two different parts to train the model with the first part (which is usually larger) and test the model with the other part (which is usually smaller, and the model has not seen none of it) to evaluate the performance of the model. Moreover, we've set the model to normalize the data to achieve a better performance.

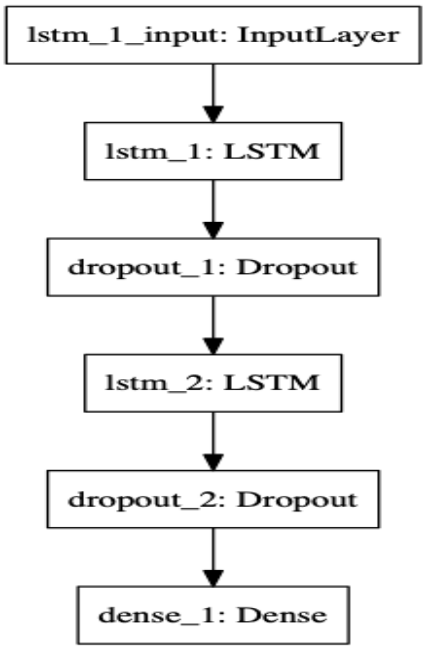

Figure 7. Model's Abstract Architecture (Generated by Keras Framework)

In the training part, we've done the whole process in one epoch with the batch size of 32. For the loss function, we used Mean Square Error (MSE), since it is more compatible for predicting values than other loss functions like Cross-entropy (which is more compatible with the situations that the model is going to be used for binary classification).

For the learning rate (which determines the speed of updating the weights), we utilized the Adaptive Moment (ADAM) optimizer. The logic behind this decision was to make the model more efficient, by letting it choose the learning rate in an adaptive manner, instead of defining a static learning rate for the whole process. This optimizing approach is the most popular one for high-performance neural networks.

After defining the above-mentioned parameters and functions, we defined and LSTM layer as the second layer of the model (since the first layer is the input layer with a single input dimension, which is the voltage value from or dataset). Right after the first LSTM layer, we've put a Dropout layer. The Dropout is a useful to technique to preserve the model from being overfitted.

The overfitting phenomena happens when the model is being trained using too much data during the training. Consequently, the model won't demonstrate a descent performance on the test data, which is completely new to the model.

So, it is highly important in professional Deep Learning models to prevent this phenomenon by using the Dropout layer after each main layer.

The Dropout function spread the weight to different neurons to prevent relying too much on certain specific neurons. This strategy is repeated for the fourth and the fifth layer as well (LSTM and Dropout architecture). Finally, it is designed as a dense layer as the linear output stage using one neuron.

5. DATASET DESCRIPTION

For training and testing the model, we've used a 1-min captured dataset of households power consumption from the "data.world" repository (which is a subset of a larger dataset at UC Irvine Machine Learning Repository) and used the "Voltage" column for prediction [3, 4]. We've defined 0.85 as the training-testing split.

So, since we used 5000 records in total, around 4300 records were used for training and around 700 records for testing the model performance. We also normalized the data by scaling it down. The following plot represents a sample of the raw data for a single day minute-average voltage:

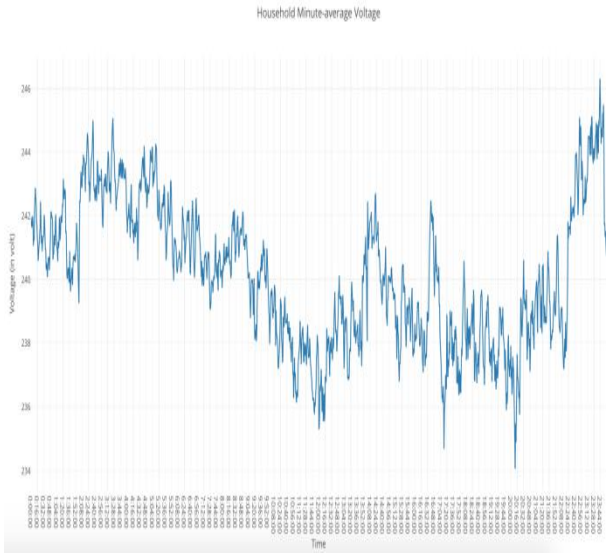

Figure 8. The voltage dataset

6. MODEL'S PERFORMANCE

After running the model, the results were beyond our expectation, since as it can be seen in the Figure 9, the predicted data and the actual data are very close together visually. However, the model can become even much better by using relatively larger datasets and Transfer Learning approaches.

Figure 9. Model's Performance on the Test Dataset

Also, the model's acceptable performance can be proved by looking at the output results from the TensorFlow framework, after the model's execution using the test dataset. As can be seen in the following figure, the output of the Mean Squared Error (MSE) function is 5.3405e-05 in scientific e notation format, which is equal to 0.000053405 in real number format. As we know from the definition that:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{y}_i)^2 \qquad (10)$$

In which the $\widehat{Y_i}$ predicted values and Y_i demonstrates the actual values. Therefore, an ideal model should equal to 0. Consequently, our model is performing pretty well with reference to the calculated MSE value (Figure 10).

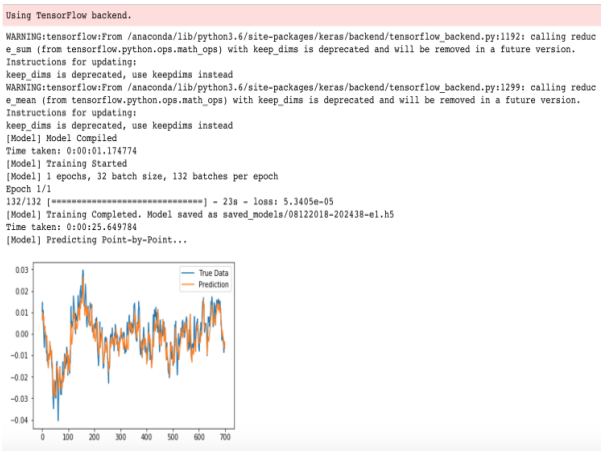

Figure 10. Tensor flow backend log

7. CONCLUSION

To recapitulate, we trained and tuned a Deep Learning model to enhance the security of the Internet of Things (IoT) using a different perspective. The initial incentive for conducting this research project stems from the significance of the IoT security, and the technical hardware/power limitations in proposing robust security solutions for the IoT. So, we came up with the idea to use the Time-Series Analysis (Sequential Data Analysis) to recognize the pattern of the voltage level of the households. According to the reliable references [5,6], the voltage level tampering is a devastating type of attack that can be done for various malicious activities in Smart Cities, since the flow current of all the electronic devices depend on the input voltage level. Therefore, we developed an optimized LSTM RNN to learn the pattern using around 4300 records of 1-minute interval voltage level and test our model with the other 700 records of unseen data, to predict the values using the model and compare them with the actual data. Fortunately, based on the mentioned statistics in the body of the paper, our model was highly successful, because of the proximity of our loss function output to the zero. This model is not only useful for the voltage abnormality detection, but also can be used for pattern recognition in other areas of IoT that are dealing with streams of data. However, any changes to the use case of the model requires descent amount of optimizations, since

several parameters with different correlations are involved in the outcome of the model, which is measured by the loss function. In terms of future works and suggestions to enhance the model, we can mention the following key points:

- Scaling up the model's scope by adding more complexity factors and increase the dimensionality of the dataset to cover more attributes of Smart Cities datasets.
- Increasing the volume of the dataset which will increase the model's performance since we've done the experiment on a subset of the dataset to prove the feasibility.

8. REFERENCES

[1] Understanding LSTM Networks, Recurrent Neural Networks, Posted on August 27, 2015, http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[2] Stanfordgroup,RrajPurkar, Jul2,2018,https://stanfordmlgroup.github.io/competitions/mura/

[3] J.Ortiz,"HouseholdPowerConsumption,"https://data.world/databeats/household-power-consumption.

[4] G. Hebrail, "Individual household electric power consumption Data Set," Aug 30, 2012, https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption.

[5] J. L. Reyes-Ortiz, D. Anguita, A. Ghio, L. Oneto, and X. Parra , "Human Activity Recognition Using Smartphones Data,April2013,Set"https://books.google.com/books?id=MtkoDwAAQBAJ&lpg=PP1&dq=Smart%20Cities%3A%20Foundations%2C%20Principles%2C%20and%20Applications&pg=PP1#v=onepage&q=Smart%20Cities:%20Foundations,%20Principles,%20and%20Applications&f=false.

[6] Googlebook,https://books.google.com/books?id=qEkzDwAAQBAJ&lpg=PP1&pg=PP1#v=onepage&q&f=false.

[7] LSTM_Networks/LSTM Demo.ipynb, "Generating Text using an LSTM Network (No libraries)," Aug 7, 2017, https://github.com/llSourcell/LSTM_Networks/blob/master/LSTM%20Demo.ipynb.

[8] T. Mitchell, A. Singh, "Machine Learning," Carnegie MellonUniversity,2011,http://www.cs.cmu.edu/~aarti/Class/10601/slides/Nnets_11_3_2011.pdf.

[9] S. D. Ali, "The Evolution and Core Concepts of Deep Learning & Neural Networks," August 2016,https://www.analyticsvidhya.com/blog/2016/08/evolution-core-concepts-deep-learning-neural-networks/.

[10] LSTM prediction using Neural Networks, Jan 25, 2018,https://github.com/jaungiers/LSTM-Neural-Network-for-Time-Series-Prediction.