Deep Learning Semantic Compression: IoT Support over LORA Use Case

Aicha Dridi
Telecom SudParis
Institut Polytechnique de Paris
Saclay, France
aicha_dridi@telecom-sudparis.eu

Arnaud Debar
Telecom SudParis
Institut Polytechnique de Paris
Saclay, France
arnaud.debar@telecom-sudparis.eu

Vincent Gauthier
Telecom SudParis
Institut Polytechnique de Paris
Saclay, France
vincent.gauthier@it-sudparis.eu

Hatem Ibn Khedher
Telecom SudParis
Centrale-Supelec
Paris Saclay, France
hatem.ibnkhedher@irt-systemx.fr

Hossam Afifi
Telecom SudParis
Institut Polytechnique de Paris
Saclay, France
hossam.afifi@it-sudparis.eu

***Abstract*—Long Range (LORA) networks are serious candidates to support Internet of Things (IoT). Despite the scalability and range of LORA, yet many IoT devices need to send much more data than what is possible in this band. In this paper, a deep learning compression method to squeeze data and transfer its semantic is presented. Data from IoT equipment is considered as a time series and trains a neural network. Resulting neural network weights are periodically sent instead of sending all the IoT raw data. Anomalies are locally detected by a similar neural network and sent separately. The resulting architecture makes it feasible to use LORA for IoT devices that generate very large amounts of data.**

***Index Terms*—Deep Learning (DL), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), Time Series, LORA, Internet of Things (IoT).**

I. Introduction

LORA networks represent a good candidate for IoT data transfer. They scale well, provide new naming techniques and do not consume large amounts of energy. Yet, they do not provide a sufficient bandwidth for a large panel of IoT devices especially regular time-based sensing devices such as power consumption, temperature, pollution sensors etc.

Our experience with a medium scale power sensing deployment in administrative premises has shown that the main issue in this deployment was about the communication part. The link between a multitude of sensing devices and the backend has been a real issue. Although, it was easy to lay down sensors and small embedded PCs, deploying a specific LAN or a WiFi network for such a purpose was difficult (both technically and administratively). Cellular connections can be a good solution but would cost too much. So our left choice was to use a cheap Long Range (LORA) connection.

LORA has a major bottleneck in its limited bandwidth. LORA groups in Internet Engineering Task Force (IETF) work very intensively to provide compression mechanisms on the protocol [1] to make it more efficient. This is great for our problem but still does not solve the dilemma of sending megabytes of data daily.

In this paper, we investigate the usage of neural networks prediction techniques as a substitute for classical compression. The paper idea is a corollary of the successful usage of recurrent neural networks (RNN) in prediction. Since these tools provide an excellent way to make time series prediction, we could use the same configured network as a replacement of data transmission. So instead of sending raw data from the Internet of Things (IoT) sensor, we locally train a neural network in the IoT neighborhood and we only send its weights. On the other-side, when an abnormal situation is detected in the generated data, it is directly sent as an outlier (via similar neural networksà and this will not really constitute an overhead in bandwidth as it is normally a rare event.

The rest of the paper is structured as follows: in Section II, we discuss the state-of-the-art of the IoT monitoring platform through LORA infrastructure. In Section III we detail some related contributions. Section IV is dedicated to the IoT data mining architecture.

Section V describes our prediction algorithms for semantic compression. In Section VI, we evaluate our work in terms of different key performance metrics. Finally, we conclude our work and give future perspectives.

II. Background

In this section we present the context of this work. We first explain the IoT platform, the LORA infrastructure and recurrent neural networks.

A. An energy monitoring IoT platform

Energy consumption monitoring consists in precisely measuring the consumption of different electric devices at different scales. It goes from high power industrial plants to small buildings. The goal being two fold:

978-1-7281-3687-5/19/$31.00 ©2019 IEEE

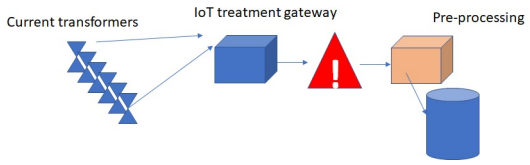

Fig. 1. Energy monitoring in a micro-grid context

- Cyclic monitoring for potential energy consumption analysis, management and reduction.
- Outlier early detection.

The micro-grid concept consists in coupling the consumption with renewable energy production through an energy buffering mechanism typically consisting in a battery (although one can have today other medium and long term storage systems that do not integrate batteries). When the management of the system is designed, power consumption measurement is vital to adapt all the rest of parameters for a better optimization.

Moreover, flexibility in energy management is essential to avoid costly upgrades to the micro-grid system while increasing the penetration of renewable energy sources such as solar, wind and other micro-sources. Therefore, in this field, we analyze a reference architecture that has three components: i) the consumption of the building, ii) the production of renewable energy, iii) and a possible energy storage.

A typical energy monitoring platform is composed of an embedded PC supporting up to 30 sensors. Sensors consist of current loops. We deployed such devices in student dorms and administrative buildings. Several clusters are required to fully cover the electricity consumption in such a case [1].

We start by collecting the data from the residential building. The data collection is essential since the quality of data will impact massively on the output. This part is going to be described more in details in section IV. The second step is the pre-processing of the data. It consists of the manipulation of data for further analysis and processing. The accuracy of the data must be checked. In fact, the raw data cannot be treated. The objective of the pre-processing is to build a dataset that could be utilized for further processing. The new dataset is used as an input for the machine learning algorithm to train and evaluate the model with historical data. The dataset is divided into two different datasets, one for the training and the second one for the testing steps.

B. Compression techniques

Data compression is now a very mature subject. Globally, we find data compression for storage reduction and in transmission in general. The venue of modern video transmission (MPEG) brought the introduction of lossy compression, based on quantization. Here, the idea is to adjust transmitted values to fixed thresholds to reduce compression states. One can refer to [2] for details.

[1]http://dataia.eu/search?search$_apif ulltext = peper$

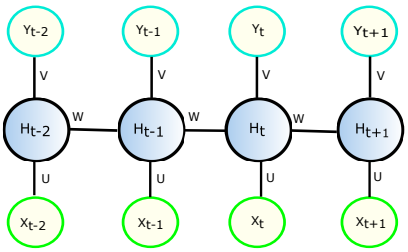

Fig. 2. The RNN blocks.

Prediction can be considered as a compression means. The system considered as a black box, works as follows: When an input is entered, the next k values are output. Several modern regression techniques exist (mainly support vector families and neural networks). We focus here on recurrent neural networks for their high performance in the prediction context.

C. Recurrent Neural Network

Recurrent Neural Networks (RNN) [3] represent a category of neural networks where each neuron is interconnected to its neighbors sequentially. RNNs have performed pretty well in different fields. They have the particularity to remember timely related events. RNNs were applied to Natural Language Processing (NLP), speech recognition, image recognition and time series prediction. RNNs can model different neural network architectures such as One to One (O2O), One to Many, Many to One and Many to Many. It means that in O2O, when one input is given to the neural network, it is capable of output of one result. Note that inputs/outputs are vectors (not necessarily one single value). To Employ RNN to predict a future sequence, we need to train it with historical data. However, using basic RNN with large sequences of data sets occasions the vanishing gradient problem. It means that the RNN is not capable of "remembering" some events.

An RNN is simply represented by a linear equation, that is multiplied by a non linear output filter (called an activation function). Fig. 2 shows a vanilla RNN where U, V, W correspond to the shared weight vectors for layers hidden, output, and different time steps respectively. X and Y represent input and output vectors.

H is the hidden state that can process a sequence of X,Y vectors by applying recurrence formulas (1) and (2) as follows:

$$H_t = \sigma(U * X_t + W * H_{t-1}) \tag{1}$$

$$Y_t = Softmax(V * H_t) \tag{2}$$

Where σ and $Softmax$ is the activation function applied at the final layer.

D. Long Short Term Memory

RNN suffers from the vanishing gradient even if they provide good results while treating time series [3]. Our choice went to Long Short Term Memory (LSTM) which was proposed by Hochreiter and Schmidhuber [4]. LSTM is of the

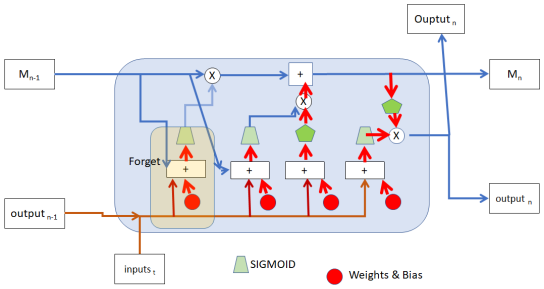

Fig. 3. Nth LSTM cell

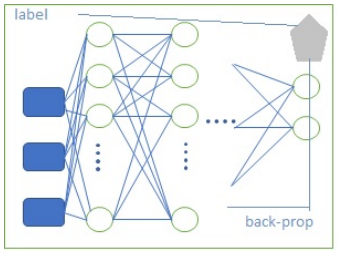

Fig. 4. The general training principle in neural nets

recurrent neural network class. It is composed of three layers: input, output and hidden layer. Each LSTM block is formed of three gates Fig: 3:

- Forget gate that helps to take decisions about what must be eliminated from the previous state and retain only relevant information.
- Input gate that adds new information from the present input to our present cell state.
- Output gate that determines what to output from our cell state after the sigmoid function.

The LSTM is stacked as in the RNN example, we put a sequence of several cells to form a chain. The number of cells in a network depends on the input complexity. Typically, for our time series, 4 to 6 cells are sufficient to give a good performance score.

E. Gated Recurrent Unit

The GRU [5] is a newer generation of RNNs and is pretty similar to an LSTM. GRU got rid of one of the LSTM gates and used the input gate to transfer information. It has two gates, a reset and an update gate.

Both LSTM and GRU gates can learn which data in a sequence is important to keep or to throw away. By doing that, it can pass relevant information down the chain of cells to make predictions.

LSTM is a central algorithm in most social networks applications. Google proposed an evolved LSTM architecture called DLSTM-P [6]. Today, LSTM and GRU are typically concatenated to one or several dense layers to improve performance.

Training all these architectures corresponds to two steps Fig. 4:

- feed-forward requires to input data (inputs correspond to the blue left boxes), force it through the whole architecture and obtain the output (right hand side circles).
- compare the output value to the real output that we want to reach (called the label), and back propagate this error from the last cell to the first one by a set of recursive mathematical equations.

When the error gradient reaches a satisfactory value, we stop. This means that all the weights are tuned to a ”good” training level. The number of weights depends on the size of the neural network (the number of gates, cells and hidden layers).

In the real world, several libraries provide very efficient neural network development. TensorFlow is used in our context. Typically also for time series, few thousands weights are necessary.

F. Outlier Detection

Outlier or anomaly detection is an automatic method that can, based on a data sequence, decide if it is a normal (normal here means regular) output or not. Many methods are proposed in the literature. A K-NN (K-Nearest Neighbor) based algorithm was proposed in [7] to detect sleeping cell in LTE networks. Obviously, K-NN is a robust and fast unsupervised machine learning but it suffers from high-dimensional input data. One-class SVM-based algorithm is proposed in [8] for malware detection. In [9], authors use random forest for network intrusion detection. Random forest is considered to be one of the most accurate methods for anomaly detection. In [10], an Isolation Forest algorithm, which is an enhanced variant of random forest, is proposed for video streaming outlier detection. We used in a separate work [11] a method derived from the LSTM family, to detect anomalies in a cellular environment. It was proved to be more efficient than the above mentioned references. Details about performance can be found in our work. For the sake of paper size and global comprehension, it is to be mentioned that outlier detection is simply a classification of events. When the system (SVM, forest or RNN) is not capable to classify the sequence into a known pattern, it is considered as an outlier.

III. Related Work

In [12], the authors propose a novel architecture for health monitoring tasks at the edge of traditional IoT networks. At edge layer, gateways apply artificial intelligence algorithms and then the results are sent to a LORA based access point to be used for global storage and other cloud services at the cloud layer [13]. This work is very entrusting, however, authors do not specify the impact of neural network meta parameters on the continuous transmission of LoRaWan.

In [14], authors propose adding machine learning techniques at the edge device to perform low power transmission through LoRa. They propose sending only the final output via LORA.

In [15], authors propose a method to determine offloading and transmission strategies that are better to directly send

fragmented packets of raw data or to send the extracted feature vector or the final output of deep learning networks, considering different operational performance metrics.

LORA research groups in IETF [1] have recently proposed a header compression mechanism for LORA. Especially, when IPv6 protocol is used to convey data, the draft gives pretty good compression results. We are involved in this work as well and we consider that header compression for LORA in general will be beneficial to all kinds of transmission, whether using our semantic compression technique or not.

IV. Data collection Cloud Architecture with online training

The Peper project is a research collaboration about deep learning and IoT support for smart grids. The efficient energy management of a microgrid relies on the prediction of the behavior of different actors: producers and consumers. One can thus envisage a system where the equilibrium in real time is no longer ensured only by the modulation of the production but also by the adjustment of the demand, thus making the consumer a central actor. The aim of the project is therefore to collect data from different actors, to use IoT and deep learning techniques to develop production/consumption algorithms and to establish collaboration between the different actors. In other words, the goal is to find a way to make production, consumption and storage work together for a better use of renewable energies.

A. Back-end Component

We briefly described the IoT architecture for energy consumption measurement in the previous section. The decision to recover one sample per second for each register (measurement point) has generated a transmission/storage problem. In fact, one arrives at almost 2 million values a day with about twenty registers (one value is coded as a real). Storage of continuous time series is not straight forward. We have used InfluxDB to sore the multi-variate collected data. This solution is a non-SQL database specialized in the storage of time series. InfluxDB has by default a timestamp accurate to the nano-second and makes it easy to make temporal queries. It is designed to handle a very large number of data without writing time overhead. A recovery interval of 5 minutes was chosen. Thus, every 5 minutes the program performs an XML request, processes the data and stores result in the database.

B. Front-end Component

Front-end is separated from the collection part. It can be remoted and it feeds graphical interfaces (Bokeh[2] in our case) and LSTM tools for prediction and data analysis.

The implemented front-end consists of two main parts:

- A graphical interface of the data stored in InfluxDB. In Fig. 5, we show the implemented graphical interface that displays the data stored in database. We can choose among many gateways. One can possibly check which register one wishes to display. Finally we can select the start date and end of the desired display. Bokeh allows the implementation of other tools on the graph that facilitates visualization and data analysis. This interface makes it possible to visualize the different powers consumed and to detect trends or irregularities. Since Bokeh tool can not handle too many points, a compression system has been put in place. Thus, for a time interval of less than 1 day, 1 point per second is displayed on the graph. Between 1 day and 2 months, the display changes to 1 point per 1 minute.
- A tool to format data into a CSV format, the preferred format for neural networks python tools. The LSTM network will be tailored for a certain number of inputs and another certain number of outputs. Thus this second part is necessary to prepare data to the required training format.

[2]https://bokeh.pydata.org/en/latest/docs/user$_guide/embed.html$

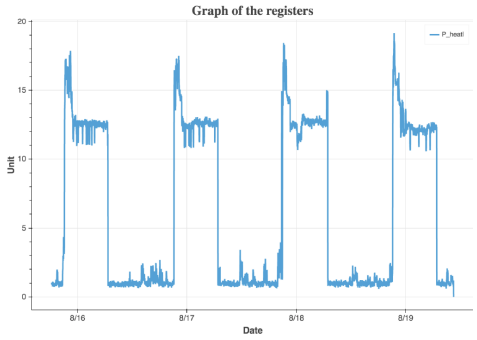

Fig. 5. Time series data generation for a single power sensor (4 days)

Using the above front-end facilities, we prove that we can easily generate time series that reflect an accurate consumption and production. Data inputs are then used for prediction and outlier detection using online deep learning techniques. We describe in the following section the main prediction models that are proposed in our approach.

All the described tools are available for download on: (https://github.com/ComplexNetTSP/Peper).

V. The proposed Semantic Compression

As explained above, IoT data collection requires a large bandwidth for its transmission. We propose hence to divide the data collection process into two parts. In Fig. 6 we show the main stages of the new architecture. First, we go through data collection from the sensor network. Then, the dataset is used locally in the gateway for the training, prediction and the outlier detection. Training output is a set of weights as described above and a periodic sample of the IoT raw data. Combining the raw data as a seed and the neural network will mimic the time series with a very small error (hence the name semantic compression). They can be re-trained frequently if time series change. The figure shows data coming from sensors and inputs. The inputs correspond to context information used for prediction. Here, we have the day of week, minute in day,

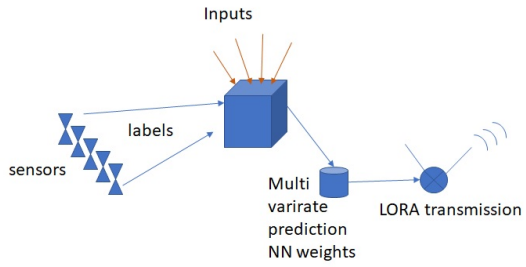

Fig. 6. The different steps of the proposed prediction approach

season, inside temperature and outside temperature. The inputs are as important as data.

A. Outlier Detection and IoT over LORA transmission

The prediction helps in the outlier detection process. Outliers are helpful to detect failures and recommend parameter tuning to avoid the malfunctioning of the IoT. As explained before, we use a previously developed outlier detection algorithm based on LSTM. The idea is that we have a variable threshold making an envelop on the predicted signal. When the real value measured at the sensor overpasses this threshold , it is considered as an outlier and it is sent directly over LORA as an alarm.

Time series outlier detector [16] detects the outliers and raises an alarm.

B. Transmission and data restitution

The IoT over LORA transmission part is straightforward [17]. A burst containing the exact (i.e., optimal) neural network weights is send periodically with a timed sample of the raw data. We can additionnaly use the LORA header compression draft for more performance. In the real deployment of the proposed model, outliers or anomalies are detected separately and sent with a different tag for immediate treatment. Moreover, depending on the capacity of the LORA messages, the maximum amount of information about the time series is sent.

The received weights are reloaded into a python program consisting of the same neural network as the training one. Based on inputs (typically day of the week, temperature, time of the day) and very rare data samples, the output is generated. The output in our case is a predicted consumed power for the input parameters presented to the test neural network. It is considered as a semantic compression since the difference between the generated data at the sensor and the one restituted is equal to the training error. In fact, data is very rarely sent over the network, but mainly its neural network representation.

VI. Performance analysis

In this part we analyse the proposed methodology performance through its steps. As mentioned before, RNN networks are not really used frequently and have been replaced by other methods that give better performance. We use basic RNN, LSTM and GRU as compression candidates. We noticed that GRU has smaller amounts of weights as it presents less gates

TABLE I
Comparison (10 epochs, 3 steps per epoch and 30 neurons) of RNN, LSTM and GRU neural network sizes

Prediction model	*NN weight size*	*MSE*
Simple RNN	19.8 kB	0.0403
LSTM	30.6 kB	0.0406
GRU	27 KB	0.0714

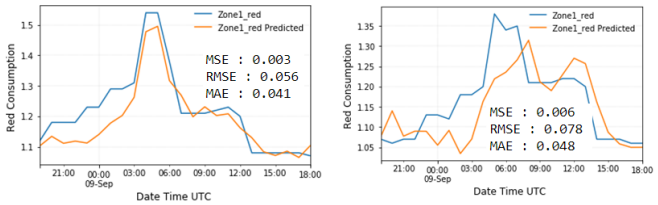

Fig. 7. Predicted consumption with two different time shifts

for the same configuration as an LSTM network. However, LSTM gives much precise prediction and hence better data restitution.

The neural network weight size for each prediction technique is presented in Table I where we observe that the GRU produces notably lower sizes. Recall that the edge gateway sends only the compressed (eg., using zip function) neural network weights. The table shows that in this kind of time series, RNN has a small Mean Square Error compared to LSTM and GRU. GRU is the best in compression size and training time, but with a higher restitution error.

In Fig. 7, we show the LSTM prediction model result applied to energy time series. They predict what is called a critical category of energy consumed in a building (Zone1). Left curve compares real values to predicted ones with one shift. Right graph gives the prediction of 12 values.

As described in the previous section, the LSTM prediction model is also used to feed the time series outlier detector. It detects potential outliers and raises proactive alarms.

Moreover, in Tab. II we show different results of the size of the neural network before and after the online training process. The experiments show that before the training step the most influencing meta parameter on the size of the neural network is the number of neural network layers. After the training step, the neural network size will increase according to the number of iterations per epoch. While testing the various neural network configurations trained with several meta parameters we noticed that the configuration (conf16) gave the best results according to the proposed key performance indicators Mean Squared Error (MSE), Root MSE (RMSE), and Mean Absolute Error (MAE).

a) Practical considerations: In RNN, prediction is usually required to find the last value of a time series. If we use this pattern, we compress data by a factor of 2. One typically asks more values to be predicted in the future and hence achieve much higher compression. This has an impact on the prediction precision. It lowers with the increase of the

TABLE II
NEURAL NETWORK SIZES WITH DIFFERENT CONFIGURATIONS

Id	*Nb of neu-rons*	*Layer*	*iter*	*Size-B-Training (KB)*	*Size-A-Training (KB)*	*MSE*	*RMSE*	*MAE*
1	1	1	10	194	197	0,894	0,946	0,921
2	1	1	100	194	222	0,236	0,486	0,478
3	1	2	100	327	354	0,154	0,392	0,339
4	5	2	100	327	354	0,066	0,256	0,203
5	5	5	100	725	753	0,073	0,27	0,214
6	10	5	100	725	753	0,059	0,244	0,186
7	100	5	100	731	759	0,017	0,13	0,086
8	100	10	100	1401	1429	0,021	0,146	0,101
9	100	10	1000	1401	1677	0,005	0,07	0,053
10	1000	10	500	1419	1557	0,004	0,067	0,052
11	1000	10	4000	1419	2539	0,003	0,055	0,042
12	100	100	100	13579	13606	0,02	0,143	0,099
13	100	100	1000	13579	13855	0,009	0,093	0,068
14	500	100	100	13755	13783	0,01	0,101	0,075
15	100	10	4000	1401	2521	0,004	0,062	0,044
16	100	10	6000	1401	3084	0,003	0,053	0,039

TABLE III
SHIFT IMPACT ON PRECISION

Time W.	*MSE*	*RMSE*	*MAE*
1	0.003	0.056	0.039
6	0.004	0.060	0.046
24	0.005	0.069	0.047

number of predicted values, hence compression rate. Tab. III reveals the impact on the accuracy of predicting one or more values. If we are sending over LORA one value per minute, that means that we possibly need to send the weights of the trained neural network every day. If we are transmitting one sample every hour, we can transmit the weight of the trained neural network once a month. As of today, each category of time series needs its own tuning and its own meta parameters. The neural network can remain the same but always needs an adaptation to the context.

b) LORA bottleneck: Finally, LORA bandwidth remains the most important parameter in the compression process. LORA will limit amount of data that can be sent periodically. Hence, it dictates raw data transmission frequency, kind and size of neural networks. LORA bandwidth varies from an operator to the other. The more frequent we train the network, the more we will require bandwidth to send new weights. As for outliers, it is not really a transmission bottleneck, since an anomaly by definition is a rare event...

VII. CONCLUSION AND FUTURE WORK

In this paper, we have proposed deep learning techniques (specifically recurrent neural networks) to learn data patterns emanating from typical IoT devices. In a LORA context with very low available bandwidth, our method helps in drastically reducing raw IoT data transmission and in replacing it with the learned neural network parameters (weights). Weights and periodic samples are sent to the cloud network through LORA and data is reproduced by using the reverse process. As IoT devices can be used to detect anomalies, we adopt a similar neural network to make the outlier detection. An alarm is directly sent over LORA in that situation.

Most of this contribution is downloadable from Github. In the perspectives of this work, we want to study new RNN categories with less computation overheads and that adapt to different time series. We want also to introduce low cost Tensor Processing Units (TPU) for better scalability. Finally, an initiative for standardizing neural network topologies and weights is necessary.

REFERENCES

[1] G. . Petrov, "Internet-draft schc-over-lorawan," in *Internet Engineering Task Force*, 2019.

[2] B. E. Usevitch, "A tutorial on modern lossy wavelet image compression: foundations of jpeg 2000," *IEEE signal processing magazine*, vol. 18, no. 5, pp. 22–35, 2001.

[3] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*, 2013, pp. 1310–1318.

[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[5] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[6] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *CoRR*, vol. abs/1402.1128, 2014. [Online]. Available: http://arxiv.org/abs/1402.1128

[7] G. R. J. Dromard and P. Owezarski, ""online and scal- able unsupervised network anomaly detection method,"," in *IEEE Transactions on Network and Service Management*, 2017.

[8] E. Burnaev and D. Smolyakov, "One-class svm with privileged information and its application to malware detection," in *arXiv:1609.08039*, 2016.

[9] P. J. G. Prashanth, V. Prashanth and N. Srinivasan, "Using random forests for network-based anomaly detection at active routers,," in *in Signal Processing, Communications and Networking, 2008. ICSCN'08.*, 2008.

[10] Z. Ding and M. Fei, "An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window,," in *in IFAC Proceedings Volumes .*, 2013.

[11] C. Boucetta, B. Nour, S. E. Hammami, H. Moungla, and H. Afifi, "Adaptive range-based anomaly detection in drone-assisted cellular networks," in *15th International Wireless Communications & Mobile Computing Conference, IWCMC 2019, Tangier, Morocco, June 24-28, 2019*, 2019, pp. 1239–1244. [Online]. Available: https://doi.org/10.1109/IWCMC.2019.8766446

[12] J. Peña Queralta, T. Nguyen gia, H. Tenhunen, and T. Westerlund, "Edge-ai in lora-based health monitoring: Fall detection system with fog computing and lstm recurrent neural networks," 07 2019.

[13] H. Ibn-Khedher and E. Abd-Elrahman, "Cdnaas framework: Topsis as multi-criteria decision making for vcdn migration," *Procedia Computer Science*, vol. 110, pp. 274–281, 2017.

[14] V. M. Suresh, R. Sidhu, P. Karkare, A. Patil, Z. Lei, and A. Basu, "Powering the iot through embedded machine learning and lora," in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, Feb 2018, pp. 349–354.

[15] J. Kang and D.-S. Eom, "Offloading and transmission strategies for iot edge devices and networks," *Sensors*, vol. 19, p. 835, 02 2019.

[16] S. E. Hammami, H. Afifi, M. Marot, and V. Gauthier, "Network planning tool based on network classification and load prediction," in *2016 IEEE Wireless Communications and Networking Conference*. IEEE, 2016, pp. 1–6.

[17] B. Mainaud, V. Gauthier, and H. Afifi, "Cooperative communication for wireless sensors network: a mac protocol solution," in *2008 1st IFIP Wireless Days*. IEEE, 2008, pp. 1–5.