Collision Prediction for a Low Power Wide Area Network using Deep Learning Methods

Shengmin Cui and Inwhee Joe

Abstract: **A low power wide area network (LPWAN) is becoming a popular technology since more and more industrial Internet of things (IoT) applications rely on it. It is able to provide long distance wireless communication with great power saving. Given the fact that an LPWAN covers a wide area where all end nodes communicate directly to a few gateways, a large number of devices have to share the gateway. In this situation, chances are many collisions could occur, leading to waste of limited wireless resources. However, many factors affecting the number of collisions that cannot be solved by traditional time series analysis algorithms. Therefore, deep learning methods can be applied here to predict collisions by analyzing these factors in an LPWAN system. In this paper, we propose long short-term memory extended Kalman filter (LSTM-EKF) model for collision prediction in the LPWAN in terms of the temporal correlation which can improve the LSTM performance. The efficacies of our models are demonstrated on the data set simulated by LoRaSim.**

Index Terms: **Deep Learning, extended Kalman filter, Internet of things, LoRa, LSTM.**

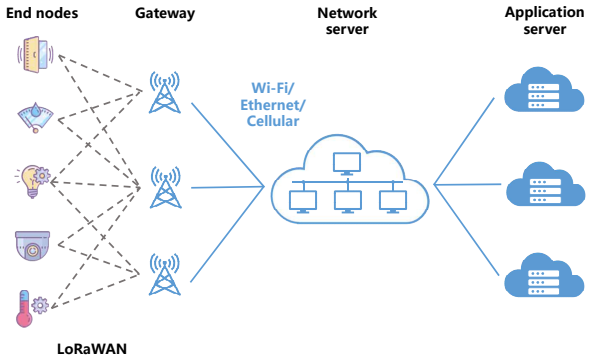

Fig. 1. Structure of LoRa system.

I. INTRODUCTION

THE Internet of things (IoT) technology is changing our lives since it has been incorporated in various fields. Examples of these areas include industrial automation, medical, smart home, health management, transportation, and emergency response to man-made and natural disasters when it is difficult for humans to make decisions [1]–[4]. IoT and machine-to-machine (M2M) industry will increase significantly over the next decade based on multiple independent studies. Recent development of sensors and new communication technologies support the predicted trends. Low power wide area network (LPWAN) represents a novel type of wireless communication that appears to complement cellular networks and short-range wireless networks to meet the diverse needs of IoT applications [5]. LPWAN technology offers wide-area connectivity for low-power and low-data-rate devices that traditional wireless technologies cannot. Traditional non-cellular wireless technologies such as ZigBee, Bluetooth, Z-wave, and Wi-Fi are not suitable for connecting to large geographical areas since they can only cover a few hundred meters. Therefore, these technologies cannot meet the requirements for many applications for personal health, logistics, and smart city due to the short range problem [6]. Although these technologies can be extended by dense deployment of devices and gateways connected using multi-hop mesh networking, large deployments are expensive. Cellular networks such as global system for mobile communications (GSM) and long term evolution (LTE) provide wide-area coverage. However, the complexity and cost of the devices are so high that energy efficiency cannot be achieved. LPWAN technology can cover tens of kilometers [7] and battery life is up to ten years or longer.

There are several competing kinds of LPWAN technologies recently that use a variety of techniques to achieve high scalability, low power, and long range. Some of the most pronounced LPWAN candidates are SixFox, Weightless, Ingenu, and long range (LoRa) [8]. In this paper, we study LoRa which is one of the most widely adopted LPWAN technologies. LoRa is proposed by Semtech and LoRa Alliance [9]. A LoRa system contains end node devices, gateways, network server, and applications servers and these form a star-of-stars topology as shown in Fig 1. End nodes transmit data to one or more gateways, then gateways send data to network servers which forward data to application servers. The reason why LoRa can develop rapidly is that it achieves flexible long-distance communication in a low-power and low cost design. This is achieved by using adaptive data rate chirp modulation technology. LoRa allow multiple users to be accommodated in one channel by using spread spectrum multiple access technique. LoRaWAN is a media access control (MAC) protocol and is implemented on top of LoRa. These features have enabled LoRa to attract a large number of developers to build complete large IoT solutions that quickly capture the market. Scalability of LoRa is currently an important part need to be analyzed and be optimized. Data ex-

Manuscript received October 22, 2019; revised May 29, 2020; approved for publication by Yansha Deng, Guest Editor, May 31, 2020.

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.NRF-2019R1A2C1009894)

S. Cui and I. Joe are with the Department of Computer Science, Hanyang University, email: {shengmincui, iwjoe}@hanyang.ac.kr.

I. Joe is the corresponding author.

Digital Object Identifier: 10.1109/JCN.2020.000017

1229-2370/19/$10.00 © 2020 KICS

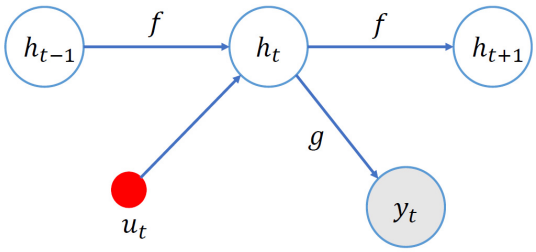

Fig. 2. Structure of SSM.

traction rate (DER) and network energy consumption (NEC) can be used to evaluate the scalability of a LoRa system. Collisions in the LoRa system have a great impact on DER and NEC [10] since LoRaWAN MAC protocol is essentially an ALOHA variant with no collision avoidance provisions. Therefore, collision prediction can help us to improve and analyze the parameter settings of LoRa and lay the foundation for intelligent resource allocation.

Time series prediction has become one of popular research topics in recent years. Traditional time series prediction methods predict future trend based on previously sampled values. Autoregressive moving average (ARMA) [11] algorithm is a combination of the autoregressive (AR) and moving average (MA) that predict the next values by constructing a linear function of values observed and residual errors of prior time steps. Autoregressive integrated moving average (ARIMA) [11] is an extension of ARMA which makes prediction using linear function of differenced observations and residual errors of prior steps. State space model (SSM) is a ubiquitous tool for modeling time series data [12], [13] that only deals with one step dependency. This is because an SSM is based on two assumptions as shown in Fig 2: The output y_t of SSM at time step t depends on hidden state h_t and external input u_t with transfer function g; the hidden state h_t only depends on previous one state h_{t-1} with transfer function f. However, these methods only make predictions based on observations and do not take into account internal influence factors. In this work, we should consider not only the observations of the number of collisions, but also the factors affecting collisions should be considered in order to improve the accuracy of the prediction. Therefore, we consider machine learning (ML) and deep learning (DL) methods which can learn the dynamic relationship between factors and observations effectively.

ML technologies have made breakthroughs in various application fields, such as computer vision [14], [15], speech recognition [16], [17], and medicals [18], [19]. Algorithms and models of ML can learn to make decisions directly from data without having to follow predefined rules. Existing ML algorithms are generally divided into three categories: Supervised learning (SL) that learns classification or regression tasks from labeled data; unsupervised learning (USL) that focuses on classifying sample data into different groups with unlabeled data; and reinforcement learning (RL) [20] that agents interact with the environment and find the best actions to maximize reward. Networking applications have also widely used machine learning algorithms, such as traffic prediction, traffic classification, resource management, and network adaption [21], [22]. Moreover, DL technologies also have made amazing achievements in many fields. For instance, convolutional neural network (CNN) is typically employed for image recognition while recurrent neural network (RNN) is frequently utilized for processing sequential data [23]. An RNN model with its non-linear function can learn more complicated pattern and has flexibility in modeling and can produce an output at each time step or read an entire sequence and then produce a single output. Long short-term memory (LSTM) [24] and gated recurrent unit (GRU) [25] are special RNN that solve the long term dependency by adding several gates in the cell. These methods can model relationship between influence factors and number of collisions with non-linear functions. Among these methods LSTM achieves best performance in several time series data sets [26]–[28]. For the collision prediction issue in this paper, we combine the LSTM and SSM for prediction task that can improve the performance of LSTM. Our model takes previous information of traffics in the LoRa system and make prediction of number of collisions in the future. The prediction results with different parameter settings are examined to find the optimal settings and structure for collisions prediction.

The organization of this paper is as follows. Related work for LoRa scalability and collision problem are described in section II. Section III briefly explains collisions in LoRa system. Then our proposed models based on LSTM for collision prediction is presented in section IV. Next, comparisons of conventional models and proposal models are provided in section V. We then finalize our paper with conclusions in section VI.

II. RELATED WORK

There have been several works that focused on scalability and collision of LoRa system. In [29], the authors studied how the number of end nodes and the throughput requirements affect the scalability of the LoRa system. They built a simulator for evaluating the scalability of a single gateway and their results shown that when the number of end nodes increases to 1000, the packets losses rate will be up to 32%. In [30], the authors proposed a model and present the effect of transmission using the same spreading factor (SF) and different SF on scalability of LoRa system. They derived several signal-to-interference ratio (SIR) distributions in different conditions. The authors of [8] built a model with a stochastic geometry framework for evaluating the LoRa system. Their results shown that the coverage probability decreases exponentially with the increase of number of end nodes when transmit using the same SF. The authors of [31] proposed single gateway and multiple gateway simulators for evaluating the scalability of the LoRa system with different parameter combinations.

On the other hand, there have been several works that committed to improve the performance of the LoRa system. In [32], the authors modelled LoRa system with a new topology that locations of end nodes follow a Poisson cluster process (PCP). Their work shown that the spectral efficiency and energy efficiency can be obtained through adjusting the density of end

nodes around each gateways. In [33], a novel resource allocation method that using multi-layer virtual cell-based spatial time division multiple access (STDMA) was proposed to improve performance of LoRa system. Their method that calculate schedule with information of end nodes and fit the cell radius to fit the communication patterns could simplify the calculation of interference and improve the data rate of LoRa system. The authors of [34] exploit a new MAC layer to improve the scalability and reliability of LoRa system. A gateway dynamically allowed specific transmission power (TP) and SF in each channel then an end node determined its own transmit time, channel, TP, and SF. Their results have shown that these methods could reduce the number of collisions and improve the performance of LoRa system. However, if we consider improving the scalability of the LoRa system through dynamically setting the parameters of transmission, we need to establish a mapping between information of current setting of LoRa system and the number of collisions. Also, we need to predict the number of collisions of future based on current state to better cope with the upcoming communication situation.

III. COLLISIONS IN LORA SYSTEM

There are several conditions that determine whether the receiver can decode or not when more than one transmissions overlap at the receiver [31]. These conditions are carrier frequency (CF), spreading factor (SF), power and timing. When we look at this issue from a holistic perspective, number of nodes, number of gateways, bandwidth (BW), coding rate (CR), data size and period can affect collisions in LoRa transmissions.

A collision behavior is defined as a situation that reception overlap, CF collision, SF collision, and power collision all arise simultaneously. It means that if any of the above events did not happen, packets will not collide at the receiver. For two packets, the situation that reception start time of the signal arriving later is earlier than the reception end time of the signal arriving earlier is defined as reception overlap. However, experiments of [31] indicate that the critical section of a packet starts at the last 5 preamble symbols. Therefore, the reception overlap can be redefined as two packets overlap at one's critical section. CF collision is defined as that the absolute value of difference between CFs of two transmissions is smaller than the minimum tolerable frequency offset. For example, Semtech SX1272's minimum tolerable frequency offset is 60 kHz when the BW is 125 kHz, 120 kHz when the BW is 250 kHz, or 240 kHz when the BW is 500 kHz. In a LoRa system, transmissions using different SF can be decoded successfully. Therefore, SF collision is the situation that two packets have the same SF. The capture effect is defined as the situation that two signals arrive at the receiver the weaker signal would be suppressed by the stronger one. However, when two signals are nearly equal in power, the receiver may switch between two signals and not able to decode either of them. This situation is defined as power collision.

IV. PROPOSED COLLISION PREDICTION MODELS BASED ON LSTM

Our goal is to predict the number of collisions in LoRa system based on previous information of traffics. From the perspective of time series problems, our model receives $\{x_{t-k}, x_{t-k+1}, \cdots, x_{t-1}\}$ and then makes a prediction of y_t where x_t is the information and y_t is the number of collisions of time step t.

Unlike traditional DL models, RNN is able to handle time series data by storing information of past in hidden states. The forward process is defined as:

$$h_t = \tanh(W[x_t, h_{t-1}] + b), \tag{1}$$

where W is weight matrix and b is bias vector. The hidden state h_t can provide output value through linear connection. Nonetheless, it is hard for an RNN to handle long-term dependency due to the gradient vanishing or explosion problems [35]. LSTM is designed to avoid this problem by adding several gates in cells.

There are two ways for training deep learning models that are offline training and online training. Offline training means the weights of the model are updated after a batch of data comes in while online training means the weights are updated after a single data has been presented to the network. Online training may lead to destroying the improvement of preceding learning step. Therefore, the residual error is often bigger than offline training. However, offline trained models no longer update their weights in practical applications even though patterns of data are changed after a while. This situation often occurs in the era of data explosion. In order to solve this problem, we proposed long short-term memory extended Kalman filter (LSTM-EKF) model which use offline trained LSTM as backbone and continue learning when reasoning.

This section first introduces the traditional LSTM and then presents the proposed LSTM-EKF for collision prediction in LoRa system.

A. Long Short-Term Memory

Instead of repeating cells that only have a $tanh$ function, LSTM has more complicated cell structures as shown in Fig 3. Each cell has the same inputs and hidden states as standard RNN, but also has several gates and cell state to control the flow of information. In a LSTM cell, there are one cell state and three gates termed input gate, forget gate and output gate. The cell state runs through the entire chain without any non-linear operations. The input gate decides what will be add to the cell state while the forget gate can decides what will be forgot. Finally, the output gate and the cell state make up the output. The forward process using the following equations:

$$f_t = \sigma(W_f[x_t, h_{t-1}] + b_f) \tag{2}$$
$$i_t = \sigma(W_i[x_t, h_{t-1}] + b_i) \tag{3}$$
$$\tilde{c}_t = \tanh(W_c[x_t, h_{t-1}] + b_c]) \tag{4}$$
$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \tag{5}$$
$$o_t = \sigma(W_o[x_t, h_{t-1}] + b_o) \tag{6}$$
$$h_t = o_t \cdot \tanh(c_t), \tag{7}$$

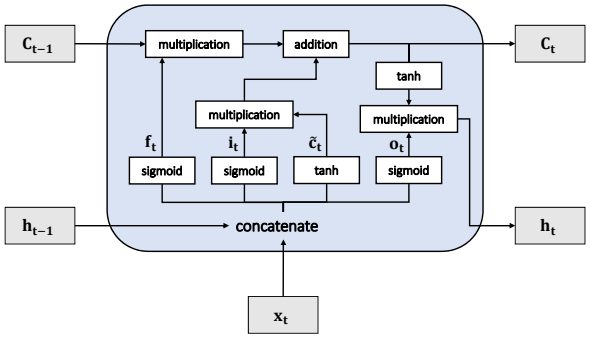

Fig. 3. The LSTM cell consists of three gates (forget gate f_t, input gate i_t, and output gate o_t). Cell state c_t can be maintained by forget gate and input gate. The hidden state h_t is decided by output gate o_t and cell state c_t.

Table 1. The input information of prediction model.

Index	Information	Units
1	Number of end nodes	-
2	Number of gateways	-
3	Spreading factor (SF)	-
4	Bandwidth (BW)	kHz
5	Coding rate (CR)	-
6	Data size	byte
7	Period	ms

where $\sigma()$ is the sigmoid function and W_f, W_i, W_c, W_o are weight matrices and b_f, b_i, b_c, b_o are bias vectors in the cell. Furthermore, x_t is the input vector, c_t is the cell state vector, $\tilde{c}_t$ is the intermediate state vector, h_t is the hidden state vector, and f_t, i_t, o_t are forget, input, output gates. LSTM cleverly provides gate control by using sigmoid function. This method retains information that requires long-term memory and discards unimportant information. This architecture also can be extended to multi-layer by treating the hidden state of last layer as input of the next layer.

B. Long Short-Term Memory Extended Kalman Filter

In this section, we introduce our novel models for collision prediction. As mentioned in section II, different parameter settings will affect the number of collisions in the LoRa system, therefore, SF, BW, and CR are included in the input information. In addition, the system information such as the number of end nodes, the number of gateways, data size, and period has an impact on the amount of network transmission, then these are also added to the input information. Finally, these numerical values at time step t are formed an input vector x_t as shown in Table 1.

B.1 LSTM Regression Architectures

LSTM models could be constructed in a variety of ways depending on the input and output, such as one-to-one, one-to-many, many-to-one, and many-to-many. In this work, our purpose is to make use of information from several past time steps rather than information from one time step to make prediction, thus we consider many-to-one and many-to-many structures to build our prediction models as shown in Fig. 4. First, let consider many-to-one structure that is the prediction y_t of the model is linear combination of the hidden state h_{t-1} with weight vector w_t and bias b_t. Then, we make the final prediction as

$$h_{t-1} = LSTM(x_{t-k}, x_{t-k+1}, \cdots, x_{t-1}) \tag{8}$$

$$\hat{y}_t^{(1)} = w_t^{(1)T} h_{t-1} + b_t^{(1)}, \tag{9}$$

where k is the length of time steps received at one time. In this architecture, the final prediction only depends on h_{t-1}. However, using more implicit information from the past may improve the performance of the forecast. For this purpose, we make prediction depend on all hidden states of LSTM. Hence, the second architecture is defined as

$$[h_{t-k}, h_{t-k+1}, \cdots, h_{t-1}] = LSTM(x_{t-k}, x_{t-k+1}, \cdots, x_{t-1}) \tag{10}$$

$$\hat{y}_t^{(2)} = w_t^{(2)T} [h_{t-k}, h_{t-k+1}, \cdots, h_{t-1}] + b_t^{(2)}. \tag{11}$$

This architecture makes prediction value with more past information than architecture 1. These models are trained by back-propagation through time (BPTT) [36] and stored the whole structure and trained weights with best validation performance. Then the LSTM structure can be used as backbones to extract hidden states for our models.

B.2 LSTM-EKF Architectures

The main idea of LSTM-EKF model is sending the features extracted from pre-trained LSTM to the SSM models and solve the SSM by using EKF algorithms as shown in Fig 5. The EKF is an extension of Kalman filter (KF) which is a well-known linear system. It linearizes the non-linear systems by using Taylor's theorem [37]. There are certain decoupled EKF-based method to train LSTM [38], [39] in an online manner. However, the performance of online training methods are far from the level of offline training. Therefore, in our case, we combine the statistical validity of offline methods and the adaptable characteristics of online methods. When making predictions on the test set, the EKF algorithm estimates weights vector and bias which directly maps the hidden state to the predicted output, while keeping the remaining weights within the LSTM unchanged. This method can make the offline trained model adapt to the test data to further improve the accuracy of the offline model. The EKF assumes that the posterior pdf of the states given the observation is Gaussian [40]. Therefore, our hybrid system based on LSTM architecture 1 in a perspective of SSM can be defined as

$$\begin{bmatrix} w_t^{(1)} \\ b_t^{(1)} \end{bmatrix} = \begin{bmatrix} w_{t-1}^{(1)} \\ b_{t-1}^{(1)} \end{bmatrix} + \begin{bmatrix} \epsilon_t^{(1)} \\ \nu_t^{(1)} \end{bmatrix} \tag{12}$$

$$y_t^{(1)} = w_t^{(1)T} h_{t-1} + b_t^{(1)} + \eta_t^{(1)}, \tag{13}$$

where $\epsilon_t^{(1)}$, $\nu_t^{(1)}$, and $\eta_t^{(1)}$ are Gaussian noises and $[\epsilon_t^{(1)T}, \nu_t^{(1)T}]^T$ and $\eta_t^{(1)}$ are with variances Q_t and R_t. Similarly our hybrid prediction model based on architecture 2 is defined as

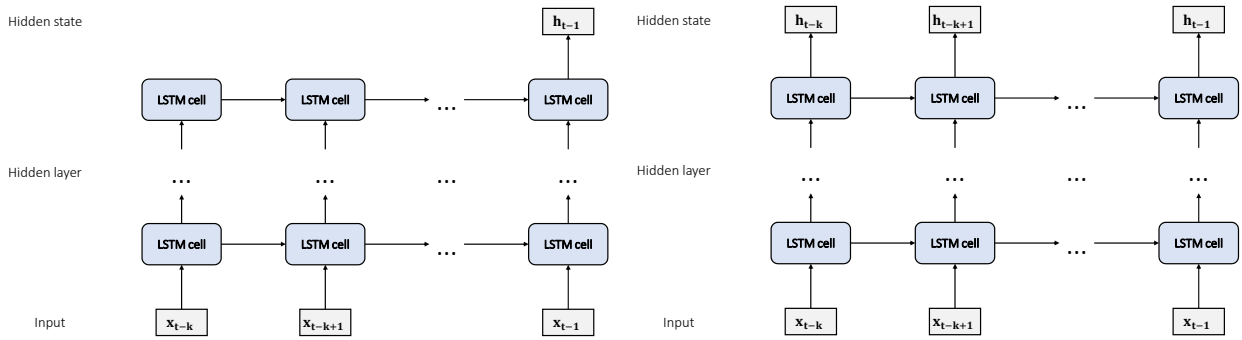

Fig. 4. Many-to-one and many-to-many structures: (a) Many-to-one and (b) many-to-many.

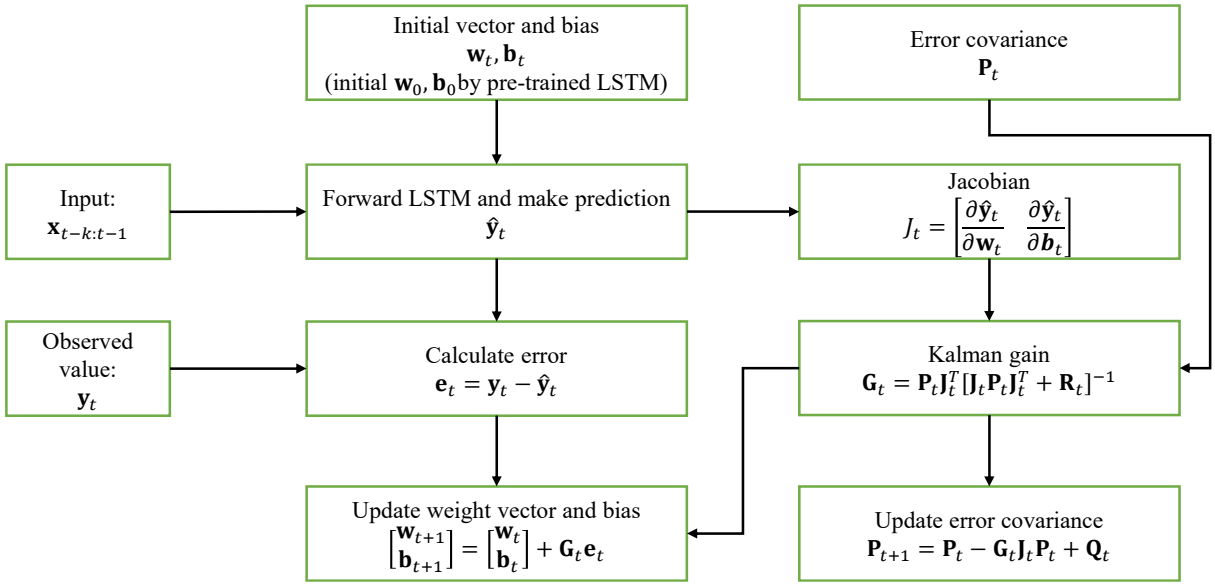

Fig. 5. Process of LSTM-EKF.

$$\begin{bmatrix} w_t^{(2)} \\ b_t^{(2)} \end{bmatrix} = \begin{bmatrix} w_{t-1}^{(2)} \\ b_{t-1}^{(2)} \end{bmatrix} + \begin{bmatrix} \epsilon_t^{(2)} \\ \nu_t^{(2)} \end{bmatrix} \tag{14}$$

$$y_t^{(2)} = w_t^{(2)T}[h_{t-k}, h_{t-k+1}, \cdots, h_{t-1}] + b_t^{(2)} + \eta_t^{(2)}. \tag{15}$$

B.3 LSTM-EKF Process

First, we initialize the hybrid model with pre-trained LSTM and then receive the input and forward through the model to calculate the output $\hat{y}_t$. This step is defined as prediction stage and the prediction result is made based on the previous information and system weighting factor. Then, we can update the w_t and b_t based on the observation. This step is defined as correction stage that calculate the posterior weighting factor conditioned on current observation. Thus, the EKF update w_t and b_t as follows:

$$G_t = P_t J_t^T [J_t P_t J_t^T + R_t]^{-1} \tag{16}$$

$$\begin{bmatrix} w_{t+1} \\ b_{t+1} \end{bmatrix} = \begin{bmatrix} w_t \\ b_t \end{bmatrix} + G_t(y_t - \hat{y}_t) \tag{17}$$

$$P_{t+1} = P_t - G_t J_t P_t + Q_t, \tag{18}$$

where G_t denotes the Kalman gain and P_t denotes the error covariance matrix. The matrix Q_t is the covariance of process noise and the R_t is the measurement noise. Finally, the Jacobian J_t containing the partial derivatives is calculated as

$$J_t = \begin{bmatrix} \frac{\partial \hat{y}_t}{\partial w_t} & \frac{\partial \hat{y}_t}{\partial b_t} \end{bmatrix}. \tag{19}$$

For the Jacobian computation, the two architectures calculate based on (9) and (11). The computational complexity of the EKF updating the weight vector and bias based on the size m and the length k of the time steps of input. Thus, the computational complexity of updating weight vector and bias of the

architecture 1 using EKF is $O(m^3)$ due to the matrix computation while the architecture 2 results in $O(k^3m^3)$.

V. PERFORMANCE EVALUATION

In this section, we evaluate performance of our proposed prediction models for the data sets generated by LoRaSim. Our goal is to predict the future collisions in a LoRa system by examining the past information. We first evaluate the performances of LSTM-EKF model and compare it to the offline trained LSTM model. Then we compare the performance of two different structures of LSTM-EKF. We then examine the parameters such as size of hidden states, length of time steps, and number of layers. To illustrate the EKF updating method is better than fine-tuning, we compare the performance of EKF and stochastic gradient descent (SGD) in online training method. Finally, we compare our model to conventional deep learning methods.

Mean squared error (MSE) and coefficient of determination denoted R^2 are used to assess the quality of these prediction models:

$$MSE = \frac{1}{T}\sum_{t=1}^{T}(y_t - \hat{y}_t)^2 \tag{20}$$

$$R^2 = 1 - \frac{\sum_{t=1}^{T}(y_t - \hat{y}_t)^2}{\sum_{t=1}^{T}(y_t - \bar{y})^2}, \tag{21}$$

where y_t and $\hat{y}_t$ are ground truth value and predict value at time step t respectively. Moreover, $\bar{y}$ denotes mean value of ground truth. MSE measures the average squared error loss between the predicted values and ground truth. It is always positive or 0 and value smaller means better. On the other hand, R^2 is a statistical measure of how well the regression model fit to the data. It is used for evaluating performance of proposed models. R^2 is normally ranges from 0 to 1. $R^2 = 0$ indicates that the regression model cannot explain the data while $R^2 = 1$ means that the regression model explains data completely.

A. DATA SET

A.1 LoRa Simulation

In this work, we use LoRaSim [31] to simulate LoRa link behaviors and generate data sets. LoRaSim is a discrete-event simulator based on SimPy [41] for simulating collisions in LoRa networks and to analyze scalability. It simulates LoRa nodes and base stations in a 2-dimensional space. Parameters such as numbers of nodes, numbers of gateways, BW, SF, CR, data size, period and total simulation time can be set up and the number of collisions can be output as a result. Gateway in LoRaSim simulates the Semtech SX1301 chip which can receive 8 contemporaneous signals since these signals are orthogonal.

A.2 Data Generation

LoRaSim mentioned above is used to generate data set and split to training data and test data. We set parameters manually and collect generated number of collisions. The time step in our work is set as 20 minutes.

We split our data set into a training set and test set by a ratio of 70% and 30% and scaled the data to $[0, 1]$ since doing so could speed up the convergence of models. The performances compared in the experiments in this paper, such as MSE and R2, are calculated using scaled data. We choose Keras library with tensorflow as backend written in Python. Models are trained on single Nvidia GTX 1080 Ti GPU.

B. Performance of LSTM and LSTM-EKF

We first consider whether the LSTM-EKF model can improve the performance of LSTM. For this purpose, we choose the same parameters settings and test on both architectures. For the LSTM, we set parameters length of time steps as $k = 5$, number of LSTM layers as $n = 2$, learning rate as $\mu = 0.01$, and the hidden state size varying among $[5, 25, 45, 50]$. All LSTM models are trained by Adam and RMSprop and the MSE is used as loss function. The model with best validation performance is stored and test on test data. For the LSTM-EKF model, we initialize the model with pre-trained LSTM and we choose $P_0 = 10^{-3}I$, $Q_t = 10^{-4}I$, and $R_t = 0.25$. Fig. 6 shows the performance of LSTM and LSTM-EKF prediction on test data. The MSE and R^2 of architecture 1 with different hidden state size are presented in Figs. 6(a) and 6(b) and the results of architecture 2 are presented in Figs. 6(c) and 6(d). As shown in Fig. 6, whether choose architecture 1 or architecture 2, the prediction results of LSTM-EKF are always better than LSTM in the case of different hidden state sizes. In addition, it can be clearly seen from Fig. 6 the improvement of LSTM-EKF with architecture 2 is more significant than the improvement of LSTM-EKF with architecture 1. For architecture 1, LSTM-EKF reduced MSE by an average of 0.95% and R^2 increased by an average of 0.02% compared to LSTM. Take the architecture 1 model with hidden state size $m = 45$ as an example, the LSTM-EKF model improved the LSTM model with 1.50% reduction of MSE and 0.03% increase of R^2 on test data. However, for architecture 2, MSE decreased by an average of 7.57% and R^2 increased by an average of 0.17%. Wherein the architecture 2 with hidden state size $m = 45$, MSE is reduced by 14.79% and R^2 increased by 0.32%. We believe that the reason of the improvement effect of architecture 2 can be better than architecture 1 is that LSTM-EKF with architecture 2 handles more hidden states and the size of weight matrix which mapping hidden states to final prediction needs to be updated during the reasoning process is much larger than the weight matrix of LSTM-EKF with architecture 1. The ground truth of collisions of test data and the prediction result generated by LSTM and LSTM-EKF with architecture 1, $k = 5$, $n = 2$, $m = 45$ are presented in Fig. 7. As can be seen in Fig. 7, the accuracy of LSTM-EKF is better than LSTM especially on the peak of the number of collisions. In summary, by adding an EKF on the top of LSTM, weights and bias for mapping features to the predict values are updated in an online manner that can improve the performance of LSTM and the improvement with architecture 2 is more obvious than that with architecture 1.

C. Different Architectures

Next, the performance of LSTM-EKF with different architectures and different settings are compared. In this experiment, we set length of time steps as $k = 5$, number of layers among $[1, 2, 3]$, and hidden state size among $[5, 25, 45, 50]$. Generally, more layers and larger hidden state size imply more parameters

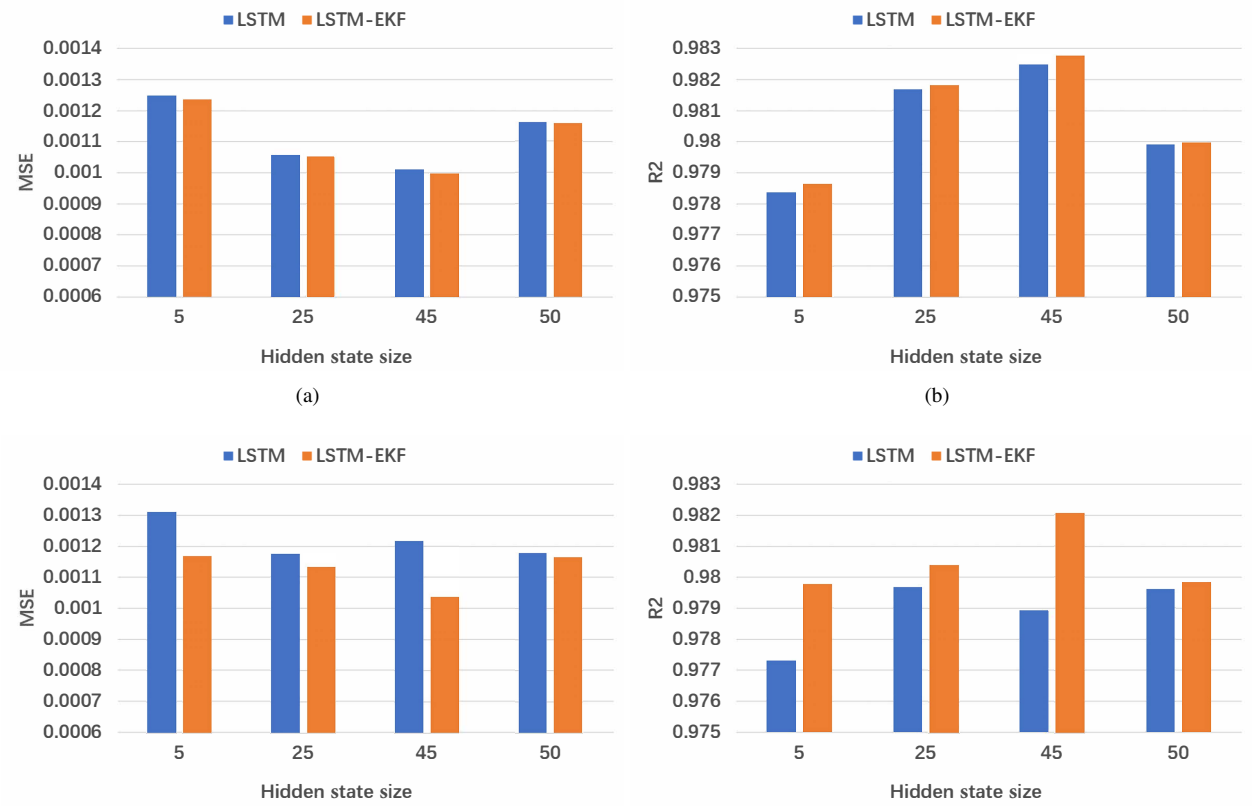

Fig. 6. Performance of LSTM and LSTM-EKF on test data with different architectures: (a) Architecture 1 MSE, (b) architecture 1 R^2, (c) architecture 2 MSE, and (d) architecture 2 R^2.

Table 2. Best performance of LSTM-EKF on test data with different LSTM as backbone.

Parameters(layers n, hidden state size m)	Architecture 1		Architecture 2	
	MSE(10^{-3})	R^2	MSE(10^{-3})	R^2
$n = 1, m = 5$	1.3549	0.97653	1.2299	0.97869
$n = 1, m = 25$	1.1144	0.98018	1.1173	0.98064
$n = 1, m = 45$	1.2042	0.97914	0.9863	0.98291
$n = 1, m = 50$	1.2014	0.97918	1.0430	0.98193
$n = 2, m = 5$	1.2336	0.97863	1.1677	0.97977
$n = 2, m = 25$	1.0503	0.98180	1.1318	0.98039
$n = 2, m = 45$	0.9943	0.98277	1.0367	0.98204
$n = 2, m = 50$	1.1566	0.97996	1.1647	0.97982
$n = 3, m = 5$	1.1359	0.98032	1.1655	0.97981
$n = 3, m = 25$	1.2180	0.97890	1.2536	0.97828
$n = 3, m = 45$	1.3187	0.97715	1.1660	0.97980
$n = 3, m = 50$	1.2031	0.97916	1.3563	0.97650

that can learn data better. However, too many parameters will lead to overfitting problem. On the other hand, too few parameters can not learn data very well. Finally, the architecture 2 handling more hidden states means that it has more parameters than architecture 1 with the same settings.

Test results are tabulated in Table 2. First, we consider the effect of the number of layers on the results. For the architecture 1, the results of two hidden layers models are generally better than one hidden layer models, however, three hidden layers models are even worse. These results indicate that when using architecture 1 to predict number of collisions one hidden layer models do not learn pattern of the data well while three hidden layers models' accuracy decrease due to the overfitting problem caused by too many parameters. For the architecture 2, it obtains the best result with one hidden layer. Such results indicate that using architecture 2 to make predictions in this work, one hidden layer is enough. The reason why the accuracy of two hidden layers and three hidden layers decrease is also due to overfitting problem. Then, we focus on the effect of hidden state size on testing results. According to Table 2, both architectures perform better on test data as the hidden state size increases until they achieve the best results. After reaching the best result, the results of the two architectures are not better with the increase of the number of parameters since overfitting problem. Finally, let consider the performances of different architectures. Architecture 2 is always better than architecture 1 with the same layers

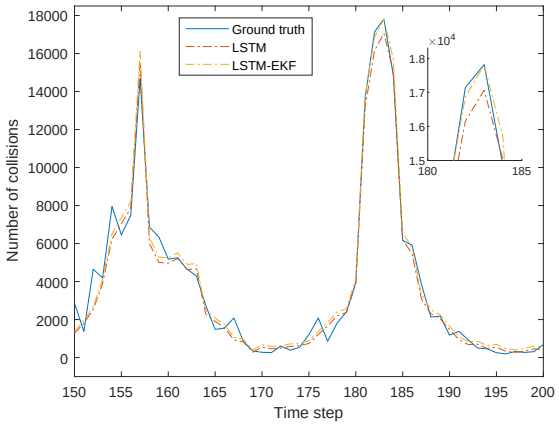

Fig. 7. Comparision of actual data with predicted results of pre-trained LSTM and LSTM-EKF (architecture 1, number of hidden layers=2, length of time step=5, hidden state size=45).

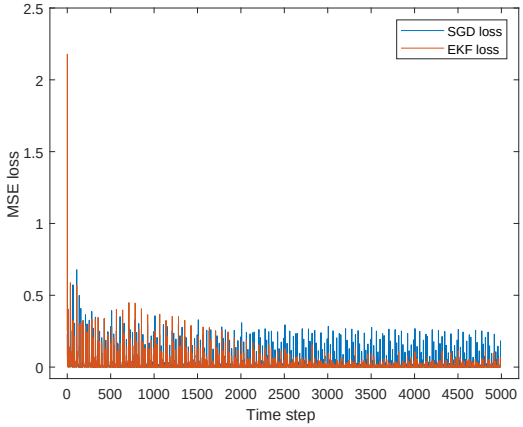

Fig. 8. Comparison of the MSE loss value of online training algorithms: EKF and SGD.

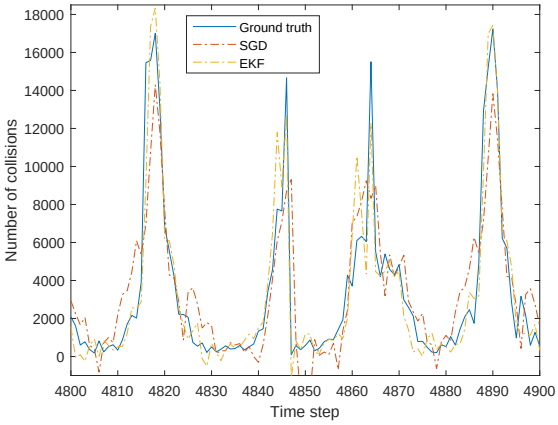

Fig. 9. Comparison of actual data with predicted results of online training algorithms: EKF and SGD.

and hidden state size before reaching the best performance. The best MSE and R^2 result of architecture 2 with one hidden layer is even better than architecture 1 with two hidden layers thanks to it handling more hidden states.

D. Online Training

Then we turn our attention to discuss the performance of on-line training. We choose the parameters settings such that both SGD and EKF reach their best performance for fair comparison. To provide this fair setup, we have the following settings. For the LSTM structure, we choose architecture 2 with length of time steps as $k = 5$, number of LSTM layer $n = 1$, and hidden state size $m = 10$. For the EKF based online training algorithm, we set the initial error covariance as $P = 1.0I$, process noise as $Q_t = 0.001I$, and measurement noise as $R_t = 0.25$. For the SGD based online training algorithm, we set the learning rate as $\mu = 0.01$. Therefore, the online training methods of EKF for LSTM is defined as:

$$[h_{t-k}, h_{t-k+1}, \cdots, h_{t-1}] = LSTM(x_{t-k}, x_{t-k+1}, \cdots, x_{t-1}) \tag{22}$$

$$\Theta_t = \Theta_{t-1} + \epsilon_t \tag{23}$$

$$y_t = w_t^T [h_{t-k}, h_{t-k+1}, \cdots, h_{t-1}] + b_t, \tag{24}$$

where Θ_t denotes whole weights in the system.

As depicted in Fig 8, the EKF converges to smaller MSE. It can be seen that it is more stable than SGD for on-line learning. The ground truth and the predicted values are depicted in Fig 9. Obviously, the predictions made by EKF trained on-line model are much better than SGD-trained model. In summary, EKF is more stable and accurate than SGD with online learning setting.

E. Performance of Conventional Models and LSTM-EKF with different length of time steps

To illustrate the effectiveness of the proposed model, we compare the model to conventional time series models such as RNN, GRU, and LSTM. GRU is recently introduced special kind of RNN and process defined as:

$$u_t = \sigma(W_u[x_t, h_{t-1}] + b_u) \tag{25}$$

$$r_t = \sigma(W_r[x_t, h_{t-1}] + b_r) \tag{26}$$

$$\tilde{h}_t = \tanh(W_h[x_t, r_t \cdot h_{t-1}] + b_h) \tag{27}$$

$$h_t = u_t \cdot h_{t-1} + (1 - u_t) \cdot \tilde{h}_t, \tag{28}$$

where W_r, W_u, and W_t are weight matrices, and b_r, b_u, b_h are bias vectors in the cell. Furthermore, x_t is input vector, u_t is update gate, and r_t is reset gate. All these models with both architectures are trained by Adam and RMSprop with learning rate $\mu = 0.01$ and stored the parameters with best validation performance. For fair comparison, we set parameters length of time steps k varying among $[5, 10, 15, 20]$, number of layers n varying among $[1, 2, 3]$, and hidden state size m varying among $[5, 15, 20, 25, 30, 35, 40, 45, 50]$. The best result of MSE and R^2 for each length of time steps of all these models are tabulated in Table 3. Since our combination of LSTM and EKF approach is generic, we also apply our approach to conventional RNN and GRU models.

As can be seen in Table 3, by adding EKF to these con-

Table 3. Best performance of proposed model and conventional models.

Models	$k=5$		$k=10$		$k=15$		$k=20$	
	MSE(10^{-3})	R^2	MSE(10^{-3})	R^2	MSE(10^{-3})	R^2	MSE(10^{-3})	R^2
RNN	1.0820	0.98125	1.0275	0.98248	0.9924	0.98341	1.1918	0.97956
RNN-EKF	1.0760	0.98136	1.0182	0.98264	0.9890	0.98346	1.1912	0.97957
GRU	1.0122	0.98246	1.0335	0.98238	0.9983	0.98331	1.1346	0.98054
GRU-EKF	0.9957	0.98275	1.0318	0.98240	0.9972	0.98333	1.1269	0.98067
LSTM	1.0014	0.98265	0.9586	0.98365	0.9542	0.98405	0.9596	0.98354
LSTM-EKF	0.9863	0.98291	0.9475	0.98384	0.9363	0.98435	0.9437	0.98381

ventional models the results are improved. When k among $[5, 10, 15]$, the MSE of RNN and LSTM decrease as k increases. However, when $k = 20$, the accuracy of RNN and LSTM both decrease. GRU and GRU-EKF also achieve their best performance at $k = 15$. For each k, the performance of LSTM model is the best in the conventional time series models and the results of RNN and GRU are close. Our proposed LSTM-EKF model achieves the best results in every different k thanks to the performance of LSTM backbones. Finally, since the LSTM-EKF achieves its best performance at $k = 15$, it is a reasonable settings for this prediction task.

VI. CONCLUSION

Deep learning is widely used in various industries with its powerful learning ability. However, in the field of IoT, especially the LPWAN that particularly sensitive to resource allocation, how to apply deep learning is worthy of further study. In this paper, we studied the collision problem in LPWAN system. We then propose LSTM-EKF model for this task. We also test with offline trained model and online trained model. Although offline trained model is much more accurate than online model, it learns the pattern or the distribution of current data. On the other hand, online trained model is more adaptable for changes of patterns but not stable as offline. However, we can see that EKF is much better than first-order gradient-based methods for training an online model. This model is much more stable than online model and can adapt to new subtle changes due to the EKF updating the parameters mapping hidden states to predict results and freeze weights of hidden layers. Based on our proposed prediction models, developers or network administrators can make pre-judgments and deal with allocation management problems. In the future, we would consider to construct a deep learning based resource allocation algorithms for LPWAN system.

REFERENCES

[1] A. Al-Fuqaha *et al.*, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Commun. Surveys Tuts.,* vol. 17, no. 4, pp. 2347–2376, June 2015.

[2] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Trans. Ind. Informat.,* vol. 10, no. 4, pp. 2233–2243, Jan. 2014.

[3] A. Zanella *et al.*, "Internet of things for smart cities," *IEEE Internet Things J.,* vol. 1, no. 1, pp. 22–32, Feb. 2014.

[4] F. Tong *et al.*, "A Tractable Analysis of Positioning Fundamentals in Low-Power Wide Area Internet of Things," *IEEE Trans. Veh. Technol.,* vol. 68, no. 7, pp. 7024–7034, July 2019.

[5] U. Raza, P. Kulkarni, and M. Sooriyabandara, "Low power wide area networks: An overview," *IEEE Commun. Surveys Tuts.,* vol. 19, no. 2, pp. 855–873, Jan. 2017.

[6] X. Xiong *et al.*, "Low power wide area machine-to-machine networks: Key techniques and prototype," *IEEE Commun. Mag.* vol. 53, no. 9, pp. 64–71, Sept. 2015.

[7] J. Petajajarvi *et al.*, "On the coverage of LPWANs: Range evaluation and channel attenuation model for LoRa technology," in *Proc. ITST*, Dec. 2015.

[8] O. Georgiou and U. Raza, "Low power wide area network analysis: Can LoRa scale?," *IEEE Wireless Commun. Lett.,* vol. 6, no. 2, pp. 162–165, Apr. 2017.

[9] *LoRaWAN Specification.*, LoRa Alliance., CA, 2017.

[10] T. Voigt *et al.*, "Mitigating inter-network interference in LoRa networks," *J. arXiv preprint arXiv:1611.00688*, 2016.

[11] G. E. P. Box *et al.*, *Time series analysis: Forecasting and ontrol,* John Wiley & Sons, 2015.

[12] A. Stathopoulos and M. G. Karlaftis, "A multivariate state space approach for urban traffic flow modeling and prediction," *Transportation Research Part C: Emerging Technol.,* vol. 11, no. 1, pp. 121–135, Apr. 2003.

[13] C. Dong *et al.*, "A spatial–temporal-based state space approach for freeway network traffic flow modelling and prediction," *Transportmetrica A: Transport Science,* vol.9935, pp. 547–560, Apr. 2015.

[14] N. Sebe *et al.*, *Machine Learning in Computer Vision,* Springer, 2005.

[15] C. J. Schuler *et al.*, "A machine learning approach for non-blind image deconvolution," in *Proc. IEEE CVPR*, 2013, pp. 1067–1074.

[16] A. Ganapathiraju, J. Hamaker, and J. Picone, "Hybrid SVM/HMM architectures for speech recognition," in *Proc. ICSLP,* 2000, pp. 504–507.

[17] L. Deng and X. Li, "Machine learning paradigms for speech recognition: An overview," *IEEE Trans. Audio, Speech, Language Process.,* vol. 21, no. 5, May 2013.

[18] M. N. Wernick *et al.*, "Machine learning in medical imaging," *IEEE Signal Process. Mag.,* vol. 27, no. 4, pp. 25–38, July 2010.

[19] K. Kourou *et al.*, "Machine learning applications in cancer prognosis and prediction," *Computational Structural Biotechnology J.,* vol. 13, pp. 8–17, 2015.

[20] L. P. Kaelbling, L. M. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artificial Intell. Research,* vol. 4, pp. 237–285, May 1996.

[21] M. Wang *et al.*, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Network,* vol. 32, no. 2, pp. 92–99, Nov. 2017.

[22] B. Mao *et al.*, "An intelligent route computation approach based on real-time deep learning strategy for software defined communication systems," *IEEE Trans. Emerg. Topics Comput.,* p. 1, Feb. 2019.

[23] T. Zhou *et al.*, "A deep-learning-based radio resource assignment technique for 5G ultra dense networks," *IEEE Network,* vol. 32, no. 6, pp. 28–34, Nov. 2018.

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

[25] J. Chung *et al.*, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555,* 2014.

[26] T. Kostas M *et al.*, "A long short-term memory deep learning network for the prediction of epileptic seizures using EEG signals," *J. Comput. Biology Medicine,* vol. 99, pp. 24–37, Aug. 2018.

[27] X. Ma, *et al.*, "Long short-term memory neural network for traffic speed prediction using remote microwave sensor data," *Transportation Research Part C: Emerging Technologies,* vol. 54, pp. 187–197, May. 2015.

[28] E. Chemali *et al.*, "Long short-term memory networks for accurate state-of-charge estimation of li-ion batteries," *IEEE Trans. Ind. Electron.,* vol. 65, no. 8, pp. 6730-6739, Aug. 2018

[29] J. Haxhibeqiri *et al.*, "LoRa Scalability: A Simulation Model Based on Interference Measurements," *J. Sensors,* vol. 17, no. 6, p. 1193, May. 2017.

[30] A. Mahmood *et al.*, "Scalability analysis of a LoRa network under imperfect orthogonality," *IEEE Trans. Ind. Informat.,* vol. 15, no. 3, pp. 1425–1436, Mar. 2019.

[31] M. Bor *et al.*, "Do LoRa low-power wide-area networks scale?," in *Proc. ACM MSWiM*, 2016, pp. 59–67.

[32] Z. Qin *et al.*, "Performance analysis of clustered LoRa networks," *IEEE Trans. Veh. Technol.,* vol. 68, no. 8, pp. 7616–7629, Aug. 2019.
[33] Y. Kawamoto *et al.*, "Multilayer Virtual Cell-Based Resource Allocation in Low-Power Wide-Area Networks," vol. 6, no. 6, pp. 10665–10674, Dec. 2019.
[34] B. Reynders *et al.*, "Improving Reliability and Scalability of LoRaWANs Through Lightweight Scheduling," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 1830–1842, June 2018.
[35] Y. Bengio *et al.*, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.,* vol. 5, no. 2, pp. 157–166, Mar. 1994.
[36] P. J. Werbos, "Backpropagation through time: What it is and how to do it," *Proc. IEEE,* vol. 78, no. 10, pp. 1550–1560, Oct. 1990.
[37] H. Jaeger, *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach,* Bonn: GMD-Forschungszentrum Informationstechnik, 2002.
[38] J. A. Pérez-Ortiz *et al.*, "Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets," *J. Neural Netw.,* vol. 16, no. 2, pp. 241–250, Mar. 2013.
[39] F. A. Gers *et al.*, "DEKF-LSTM," in *Proc. ESANN*, pp. 369–376, 2002.
[40] B. D. Anderson and J. B. Moore, *Optimal filtering,* Courier Corporation, 2012.
[41] A. Meurer *et al.*, "SymPy: Symbolic computing in Python," *J. PeerJ Computer Science 3*, vol. 3, p. e103, 2017.

Shengmin Cui received B.S. degree in Computer Science and Technology from Dalian Nationalities University in 2010, China. He acquired his Master's degree in Computer Software and Theory from Yunnan University in 2013, China. At present, he is pursuing the Ph.D. degree in Computer and Software with Hanyang University, South Korea. His research focus includes Internet of things, wireless network resource management, machine learning, deep learning and reinforcement learning.

Inwhee Joe received the B.S. degrees in Electronics Engineering from Hanyang University, Seoul, South Korea, and the Ph.D. degree in Electrical and Computer Engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 1998. Since 2002, he has been a Falculty Member with the Division of Computer Science and Engineering, Hanyang University, Seoul. His current research interests include Internet of things, cellular systems, wireless power communication networks, embedded systems, network security, machine learning, and performance evaluation.