Fingerprinting based Localization with LoRa using Deep Learning Techniques

Dr. Xuyu Wang, Jait Purohit

***Abstract* — Fingerprinting Localization, a common technique used in indoor positioning uses short-range radio frequency and addresses problems with multi-path. Although, there are challenges in fingerprinting approaches like spatial ambiguity, long distances, low-bandwidth, scalability, cost and size constraints. Additionally, using BLE and Wi-Fi in indoor and outdoor environments has proven less efficient in comparison to Lora, considering location accuracy, RSSI stability, and packet-drop in line-of-sight and non-line-of-sight scenarios in GPS based applications. This paper implements fingerprinting localization using Lora (Long Range) technology for indoor environments using Deep Learning models. In the offline phase of fingerprinting, 2D data at different locations have been collected, gathering RSSI values from the gateways at a fixed location. The online phase estimates the mean location error using Deep Learning models. The results from multi-layer Deep Learning Neural Network models like Artificial Neural Network (ANN), Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) are compared here. Additionally, we have implemented hyperparameter tuning to improve our results by changing optimizer parameters like learning-rate, batch-size, and tuning model parameters like number of hidden units, number of layers, activation functions and optimizers. Our indoor environment is a university building and data has been collected on the third floor. Indoor experiments using DL techniques achieve 1.2-2.0 [m] of mean distance error. On the contrary, we have compared our deep-learning techniques for publicly available outdoor data-source collected from several Lora wan base-stations and Lora nodes from Antwerp city, Belgium. Interpolation techniques using denoising auto-encoders have helped to interpolate outliers for this data. Our results have demonstrated that a mean distance error of 191.52 [m] from LSTM has out-performed results from the KNN algorithm. The whole approach has been implemented in a python-based framework called TensorFlow. Google Colab has been used to train models as we compare our results from different hardware accelerators like GPU and TPU. Libraries like Scikit-learn and Keras for implementing different classifiers, pandas and numpy for data-preprocessing and seaborn and matplotlib for data visualization have been used in our implementation.**

***Index Terms*—Internet of Things, Received Signal Strength (RSSI), Deep Learning Algorithms, fingerprinting localization, Hyperparameter Tuning.**

I. INTRODUCTION

The Internet of Things (IoT) is a system of interconnected things, computing devices like sensors, objects with an ability to interact over internet over shorter and longer ranges of communication. Industry 4.0 depends heavily on principles of IoT with billions and trillions of interconnected devices over network. These devices become more efficient and scalable in the need of power-efficiency, low-cost, ability to transmit long-ranges and to have reliable communication. To achieve this, low-power communication protocol like Sigfox, Lora, and narrowband (NB)-IoT are being researched and used. Lora (Long range) communication is a spread spectrum modulation technique, which helps in connecting these millions and billions of devices across countries. Lora is the DNA of IoT, connecting edge devices to cloud and enabling real-time analytics of data for more business and software application use-cases.

Positioning these devices is the most important thing in indoor and outdoor IoT eco-system. The need arises with the increased usage of location-based services. Contrary, it becomes difficult to have low-powered and cost-effective solution as it using GPS receivers to communicate with each node. A promising alternative to resolve these problems is fingerprinting-based localization, leveraging the deep-learning techniques to optimize and locate accurate position.

Moreover, fingerprinting localization techniques has two phases: An offline phase (training) and an online phase. In offline phase, received signal strength indicator (RSSI) from base-stations *aka* gateways are collected and stored into database known as fingerprints. Subsequently, in online phase, locations of sensor nodes are predicted accurately by comparing current RSSI values with the values stored in database using pattern matching techniques. Using interpolation techniques like linear, cubic and denoising auto-encoder, we have recovered outliers and missing values in the training phase. In this paper, we are discussing about improving outdoor localization using various Deep Learning techniques for an open-source Lora-wan dataset from Antwerp city, Belgium. We are

leveraging deep-learning algorithms like Long Short-Term Memory (LSTM), Artificial Neural Network (ANN) and K-Nearest Neighbor (KNN) to predict the location using GPU accelerator. Eventually, we will compare our results running our training models on TPU accelerator to optimize our prediction results on test-data. The paper also discusses indoor localization using Lora nodes and gateways, improving the mean location error using Deep Learning techniques. The results and analysis have been covered in following sections.

II. Background and Problem Statement

In this section we have introduced different types of localization techniques used in many GPS-enabled applications like Range-based techniques, Range-free techniques and Fingerprinting methods for an indoor and outdoor environment. Lora has proven important in our approach in comparison to BLE and Wi-Fi.

A. *Background*

GPS, as we know, have become very important in various localization applications. Indoor environments like positing building infrastructure uses localization techniques. On the other hand, outdoor applications like car-navigation systems, cell-phone tracking systems or edge-to-edge devices are heavily dependent on location. Range-based localization techniques focuses more on the distance between end-nodes to estimate location, while range-free techniques focus on target object's proximity to transmit beacons with know locations. Fingerprinting is a mapping technique used in indoor and outdoor positioning applications, that operates in short-range RF propagation pattern and is used to address problems with multi-path scenarios. Lora-wan is a widely used proprietary for long range communication based on chirp spread spectrum modulation technique called as Lora. Lora operates in different frequency bands in different geolocations viz. Europe region has frequency band of 863 to 870 MHz, US region has 902 to 928 MHz while China operates between 779 to 787 MHz Symbols are being encoded using number of chirps, which causes the signal to be spread over various bandwidth of channels. This technique helps to reduce interference with other signals. Spreading Factor (SF) determines the number of chirps and can range from 7 to 12. The value of SF closer to 12 means longer ranges can be achieved at the expense of low data rate in comparison to low spreading factor.

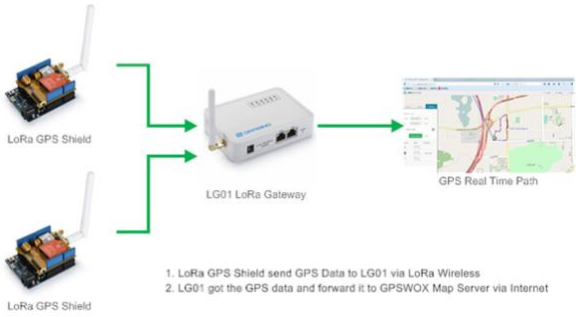


Fig. 1: Example of how Lora sensor nodes interacts with Lora Gateway

Compared to other communication technologies like Sigfox and NB-IoT, Lora works with higher bandwidth which helps in enabling localization with Time Difference of Arrival (TDoA). Therefore, accurate synchronization amongst receiving gateways is very important. Fig. 1 illustrates how Lora end-devices interacts with Lora base-stations and can be used in GPS based data-analytics and real-time processing applications. End-nodes are physical hardware sensor devices that contain sensing capabilities and computing power up to some extent. Gateways, also known as base-stations or an access-points are used to pick up all message payloads from edge devices. These payloads are converted to an array of bits that can be sent to traditional IP networks. Network Server is mainly responsible for routing/forwarding messages to right application. Application Server is the place where actual IoT application is residing and is useful with data collected from edge devices. Application servers can run mostly on cloud to preform advanced analytics and can be helpful in doing machine learning to optimize the code.

B. *Problem Statement*

Localization brings lot of challenges like non-linearity issues between sensor nodes and target base-stations, power-consumption and cost complications, typical infrastructure layouts, spatial ambiguity and RSSi signal instability in an indoor and outdoor environment. The goal of our project is to solve these problems of fingerprinting localization by using deep-learning techniques and improve location accuracy in comparison to traditional approaches like pattern-matching and common machine-learning algorithms like KNN. Each Lora message transmitted from a sensor edge device has to travel through multiple base-stations to reach to its desired location, which can be 100 miles long. This process of multilateration ranging-

based techniques brings lot of challenges described in Fig. (2) where edge-devices are sparsely located. There are lot of uncertainties like optimized signal strength from base-stations, time of arrival (ToA) as increase in the distance, transmission time per message.

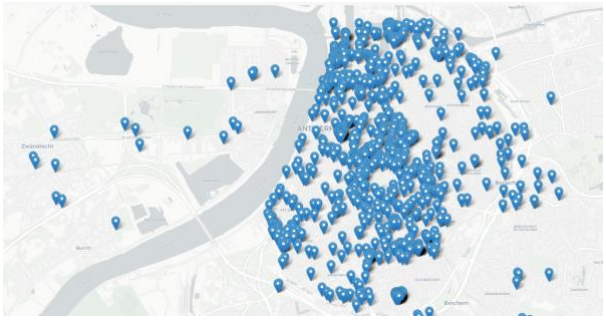


Fig. 2: Multi-lateration process of Lora message transmission

There are also outliers and missing values during the offline-training phase. We have also analyzed with our data exploration that, distance and RSSI values are inversely proportional to each other. These outliers or missing values can cause localization problems as distance increases and aggravate the outdoor fingerprinting localization process. We have discussed strategies to interpolate these values during the offline-training process. These imputed values have been used to predict the locations in online phase. Finally, we have used deep learning algorithms to improve our prediction accuracy.

III. METHODOLOGY

A. *Fingerprinting map generation*

As discussed earlier, fingerprinting localization algorithm consists of two phases viz. offline or training phase and online phase. In offline phase, we collect data about signal strength on real-time basis for various base-stations and are stored in the database for each sensor node in the service area. The uplink messages are transmitted to 68 base-stations from various sensor devices. These base-stations are located at different distances and have different range of RSSI values. According to Fig 3., there are many base-stations with -200 RSSI values and are considered outliers, impacting our output labels during training models. In offline phase, we have corrupted those values as null for interpolation process. Moreover, we select 10 random base stations and generate fingerprinting maps for them using interpolation techniques described in later sections.

'BS 11'	'BS 12'	'BS 13'	'BS 14'	'BS 15'	'BS 16'	'BS 17'	'BS 18'	'BS 19'	'BS 20'	'BS 21'	'BS 22'
-200	-200	-200	-95	-200	-200	-200	-200	-200	-200	-200	-200
-102	-200	-103	-200	-200	-200	-200	-200	-200	-200	-200	-200
-200	-200	-200	-200	-200	-200	-200	-200	-106	-200	-200	-200

Fig. 3: Outlier values of -200 before Interpolation

B. *Common Interpolation Techniques*

Lora signals from edge-devices cannot be measured for all the location. Therefore, the outlier values have to be interpolated using sample data points. Several interpolation techniques like linear, cubic, quadratic and denoising auto-encoders are discussed below.

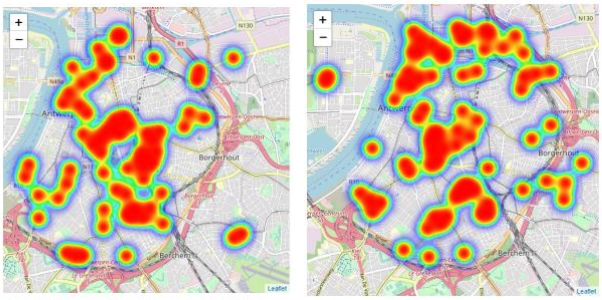
As we have known that fingerprinting technique can be divided into two phases, offline and online. We have also seen that the message payloads have information like RSSI, ID, timestamp, spreading factor (SF), HDOP and geolocation co-ordinates for each edge devices. Each base-station in the multi-lateration process have different signal strength values and are sent to the online phase and are stored with their corresponding coordinates. There are many outliers in the process and can be interpolated using different approaches.

Linear Interpolation used here is a method of fitting the curve using linear polynominals to form new data points within some discrete range of known data points. Linear interpolation for (x_0, y_0), (x_1, y_1) … (x_n, y_n) are defined as the concatenation of interpolants in linear order for each pair of data points. Linear Interpolation in machine learning world is training with missing values by reconstructing the outliers using the curve of the input/output. mathematical concepts of linear interpolation. Similarly, we implemented cubic and quadratic interpolation which have thin line of difference compared to linear interpolation. Quadratic interpolation is made with polynomials of degree two, while cubic uses degree 3 polynomials. Therefore, base-stations with missing values are interpolated using forward or backward pattern of known data. We compared the interpolated results of LoRa signals with linear, cubic and quadratic interpolation functions in python SciPy inbuilt library functions. Fig. 5 shows comparison of interpolation results of f(x,y) using gaussian RBF for linear interpolation of 100 random points for each of the two base-stations.

We have compared our interpolation results with denoising autoencoders in next section. They are a stochastic version of basic autoencoders in deep-learning.

C. *Interpolation using Denoising Autoencoders*

Autoencoders are a part of deep learning neural networks which are mainly used for selection and extraction. Moreover, there are many nodes in the hidden layer than there are inputs, which are called as "Identity Function", also called a "Null Function". Denoising autoencoders solves our problem by corrupting the outlier data on purpose by converting them to null values. Theoretically, the percentage of input nodes which are being set to null values are about 50%, which depends on amount of data and input nodes we have.



'BS 17'	'BS 18'	'BS 20'	'SF'	'HDOP'	Latitude	Longitude
-121.0	-120.0	-79.0	9	0.67	51.223099	4.405893
-121.0	-120.0	-80.0	9	0.75	51.224056	4.405560
-121.0	-120.0	-81.0	8	0.74	51.222733	4.407053
-121.0	-120.0	-82.0	10	0.72	51.223553	4.406034
-121.0	-120.0	-82.0	12	0.82	51.222908	4.405167

Fig. 4: Interpolation results using Denoising Auto-encoders

In our implementation of denoising autoencoders, we have a function to shuffle data around and learn more about the data by attempting to reconstruct it. This process of shuffling has helped to recognize the features within the noise and will allow us to classify the input values. While training the neural network, it generates a model and measures the distance between the benchmark that has been set and the model through a loss function f(n). The loss function f(n) attempts to reduce the loss function by resampling the shuffled inputs and re-constructing the data-value, until it finds those inputs which are true to actual value. Fig. 4 shows the results from two base-stations after interpolation using denoising autoencoders discussed above. Therefore, the whole process attempts to find the risk of identity-function by converting outliers to null values and autoencoders must then denoise or learn to reconstruct it back, minimizing the log-loss function.

Gateway	Interpolation	Average	Standard Deviation
BS-1	Linear	7.85	7.52
	Cubic	10.12	9.23
	Quadratic	12.47	10.63
	Denoising Autoencoder	6.52	3.79
BS-2	Linear	6.97	7.17
	Cubic	9.24	8.47
	Quadratic	10.47	9.37
	Denoising Autoencoder	5.62	3.47

Fig. 5: Comparison of Interpolation Techniques

According to Fig. 5, comparison of various interpolation techniques has been shown based on the approaches discussed in previous sections. Denoising Auto-encoders have given better results in terms of lower average and standard deviation from its interpolated data in comparison to other common interpolation techniques.

IV. Deep Learning Algorithms Used

As growing numbers of IoT sensors and gateways into every sector, businesses are accruing huge amount of data. Understanding the data and extracting meaningful information is a challenge that is starting to be met by using the applications of machine learning (ML) and Deep learning algorithms. Fingerprinting localization is improved to predict the accurate location and reduce generalization error using deep learning techniques discussed in this paper. Using deep learning algorithms like ANN, LSTM, we are able to learn automatically about features with multiple levels of non-linear operations. The hidden layers of neurons, backpropagation of weights, activation functions and neuron counts play an important role in deducing the output in our use-case.

Our implementation has been divided into three approaches for outdoor dataset, while deep learning techniques are used for indoor localization.

- Deep Learning techniques for Fingerprinting Localization using Naïve Approach

- Deep Learning techniques for Fingerprinting Localization using highest Probability amongst 10 randomly selected base-stations
- Deep Learning techniques for Fingerprinting Localization with randomly selected 10 base-stations

A. *Basic ML Algorithms*

a) Simple Linear Regression

Simple Linear Regression is a machine learning algorithm dependent on supervised learning. Regression is used to model a target output value based on independent variables as a part of input. They are used to find relationship between variables between dependent and independent variables. Linear regression performs the task to predict value of output variable y, based on given x. This relationship can be understood by drawing a regression line, which is the best fit for our machine learning model.

In our dataset, input features are RSSI values of base-stations, spreading factor (SF), HDOP, while output variable is multi-variate i.e. latitude and longitude. We have calculating the accuracy, rmse and r2 scores for our prediction using linear regression. Our results vary according to the approach that we select as discussed above.

b) Support Vector Regression (SVR)

Support Vector Regression is a type of regression algorithm but is similar to how SVM is implemented for classification, with a few thin lines of differences. As the name suggests, SVR is used for regression scenarios instead of classification. In simple linear regression, we try to minimize the error rate eventually, but in SVR we try to fit the error within a certain threshold. The hyperplane is the separation line between data classes and boundary lines. Our main goal is to only consider points that are within the boundary line. The hyperplane line which has the maximum number of points is the best fit line in SVR algorithm.

In implementation, we import the model using sklearn.svm and the kernel we have used is linear kernel. By default, rbf kernel is set, if not mentioned. As a part of metrics, r2 score and rmse are used as a part of accuracy score prediction. We also noted that tuning the parameters, we have improved our metrics.

c) K-Nearest Neighbour (KNN)

K-Nearest neighbor is used to store all available use-cases and predict the numerical output based on distance functions. KNN has been used in pattern recognition and statistical estimation. The distance function we have used in our KNN implementation is Euclidean distance as shown in Fig. 6.

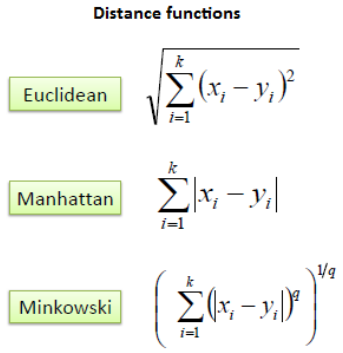


Fig 6. Distance Functions used in KNN implementation

In implementation, we have split the datasets into 65% training, 15% evaluation and 20% test. We then calculate Euclidean distance matrix between training and evaluation. We then use this matrix to find the k nearest neighbors for every evaluation record. We have used centroid of the k-nearest training fingerprints as location estimate. Using this, we now calculate the root-mean squared error for every k-value. Finally, the smallest-k value has the optimal parameter for our location estimate using KNN algorithm. Moreover, we compared our results with three approaches, discussed above.

B. *Deep Learning Algorithms*

Deep Learning algorithms use neural network architecture to find relationships and associations between a set of inputs and outputs. The network is composed of input, output layers and hidden layers as shown in Fig. 7.

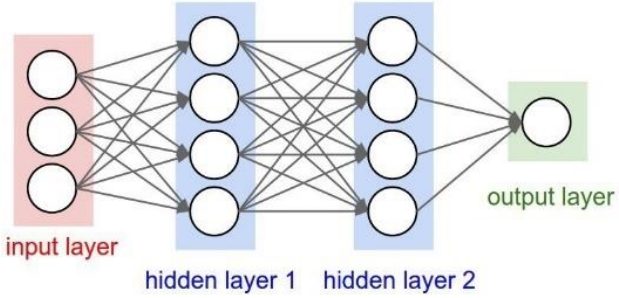


Fig 7. Deep Learning Neural Network

We have implemented few deep-learning algorithms like Artificial Neural Network (ANN), Long short-term memory (LSTM) and Convolutional Neural Network (CNN) for our fingerprinting localization problem.

a) Artificial Neural Network

Artificial Neural network are similar to multi-layer perceptron and work according to weights stored at each neuron and adapts itself accordingly, which is termed as backpropagation. From architecture point of view, it consists of input layers which is the input data we provide to ANN, hidden layers where all the transferring of weights and data takes places in reverse fashion and an output layers where the end-goal computations of the network are executed.

In our implementation of this algorithm, we have more than one hidden layer with different neuron counts in each of the layer. We are running several epochs to train our model using several iterations. Epoch is an iteration of entire training data, at once. We have applied backpropagation once we complete entire epoch, which are broken down into several batches. To sum up about our hidden layers, we have 3 hidden layers, each having 60, 45, 30 neuron counts.

$$Y = \sum (weight * input) + bias$$

Fig 8. Neural Network Neuron Equation

Fig. 8 describes about the how each neuron behaves by calculating the weighted sum of its input, adds a bias and then gets activated. Activation functions, used in our implementation are used to check the Y-output value produced by a neuron is correct or not. There are several activation functions like sigmoid, linear, tanh, ReLu used for different purposes. The usage of each of them depends on how well you understand the function you are trying to approximate has certain characteristics, viz. sigmoid fits well for a classifier. We have used SoftMax as our activation function as we are predicting multi-variate output variables. Log-loss functions like mean-squared-error has been also used which measures the performance of the regression model. Model Checkpointing has been used here as a snapshot of the state of the neural network process in case of system failure. This helps to revert back to latest checkpoint being saved and we can train the model henceforth. The model checkpoint callback class helps to define where to checkpoint the weights of the model and how should the file be named. The model checkpoint can then be passed to the training process while calling fit() function of the model. The trained model can be saved in HDF5 format define as model.load_weights().
We have calculated the accuracy scores and metrics like rmse after the training of the model has been carried out on test data.

b) Long Short Term Memory (LSTM)

Long short-term memory is used to deal with sequence prediction problems. They have an edge in comparison to conventional feed-forward neural networks and RNN in many ways, as they have the ability to remember patterns for longer durations of time.

LSTMs are designed to avoid long-term dependency problems which are seen in RNN. LSTMs have chain like architecture and has a slightly different structure for the repeating module as described in Fig. 9.

The important part of LSTM is the cell state, the horizontal line on the top of each cell. The cell state is similar to conveyor belt and runs straight down the whole chain, with limited linear interactions. There is a gate like structure which helps to add or remove information in the cell state. These gates let the information flow through and are composed of sigmoid neural net layer and have a pointwise multiplication operation.

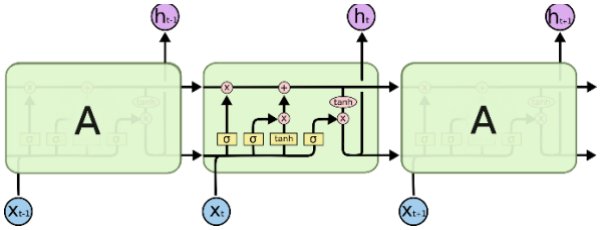


Fig 9. LSTM Architecture with four interacting layers

The sigmoid layer results between zero and one, and gives information about how much each element should be allowed to flow in.

In our implementation of LSTM, to optimize fingerprinting localization, we have created x-array and y-array matrix. These matrices are returned as numpy arrays while calling sequence () function. These sequence functions are set to size equals to 1, which helps in giving results for latitude and

longitude for each edge device. LSTM model is sequential in nature and has linear stack of layers. This model can be passed with input_shape argument to the first layer as well as type of layer i.e. dense.

model.add(LSTM(64,dropout=0.1, recurrent_dropout=0.1, input_shape=(1, 6)))

Dropout is a regularization method where recurrent connections to LSTM and input are removed from activation and weight changes while training the network. Dropout are used here to avoid the reducing overfitting and improving model performance.

In our implementation, LSTM has given us the best optimized results for predicting the geo-locations on test data. It has been observed that, dropout being set to 0.1 has improved our rmse scores. The input shape has been set to (1,6) which points to the input training features. Recurrent dropouts mask the connections between the units, which have been set to 0.1 here. The output dense layer has 2 neurons for multi-variate variables. Mean-squared-error has been used as log-loss function along with Adam as an optimizer.

C. *Outdoor Experimental Results*

Our experimental results from online phase involves three approaches, based on which we have concluded our best approach towards fingerprinting localization. To brief our whole process, we started with 68 base-stations and their RSSI values at respective latitude and longitudes. In offline phase of data gathering, we intentionally made null values of outliers and imputed them using different interpolation techniques. Amongst them, denoising auto-encoders yielded best imputation results for those outliers.

Fig. 10 shows the correlation heatmap of interpolated data of 10 randomly selected base-stations after offline phase. Correlation matrix in pandas is created to visualize the correlations between different variables in a data-frame. Here, the correlation value of a variable with itself is 1. For that reason, the primary diagonal values are 1.00 always.

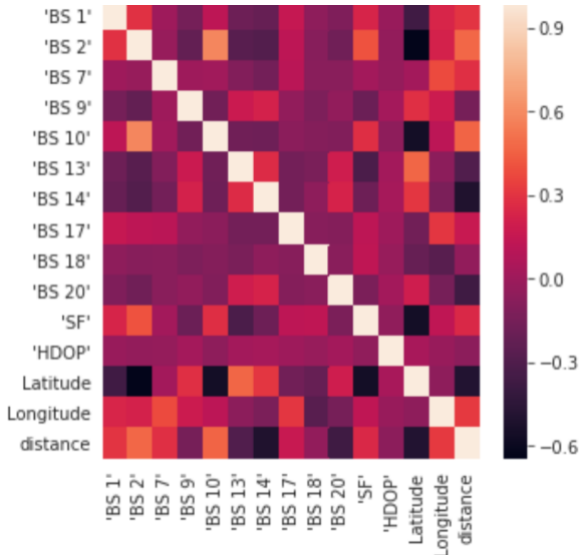


Fig 10. Co-relation Matrix of Data-Frame

a) Naïve Approach Analysis:

Naïve approach discussed here is simple approach of training models using deep learning algorithms, directly on its raw dataset from lorawan base-stations. Basic machine-learning algorithms like knn, svr, linear regression are used here to predict the optimised location. Moreover, we are using deep-learning algorithms like lstm and ann to improvise our accuracy. The accuracy and rmse score are calculated.

b) Deep Learning techniques for Interpolation with probability Approach:

Interpolation with probability approach discussed here is combining interpolation process of denoising autoencoders with probability map created for randomly selected 10 base-stations.

	probability	Latitude	Longitude
BS 1	0.727997	51.214141	4.413844
BS 2	0.651366	51.214141	4.413844
BS 7	0.664138	51.214141	4.413844
BS 9	0.702453	51.214141	4.413844
BS 10	0.651366	51.214141	4.413844
BS 13	0.651366	51.214141	4.413844
BS 14	0.689682	51.214141	4.413844
BS 17	0.676910	51.214141	4.413844
BS 18	0.740769	51.214141	4.413844
BS 20	0.485331	51.214141	4.413844

Fig 11. Probability-scores from Gateways

Using denoising Autoencoders, interpolation of outliers in offline phase was carried out. The RSSI values of end-devices are compared with each fingerprinting map obtained at the end of offline phase. Eventually, candidate regions are extracted for each fingerprinting map. The weighting factor and these regions are combined to create a probability map for each gateway. This process is implemented using Naïve Bayesian theorem. Fig. 11 describes about the probability scores obtained from each base-stations along-with its signal strength for a given latitude and longitude. The base-station 18 had the maximum probability score and can be given highest priority while training deep learning neural network models. Inputs to the training model are BS-18 RSSI values, SF, HDOP and output variables are latitude and longitude. Deep Learning algorithms have been trained here.

For LSTM and CNN, we have seen that trying different permutations and combinations of neuron counts, activation function and introducing dropout layers, the rmse score has improved in comparison to other models that has been discussed. We have also trained our model using basic machine learning algorithms for regression like support vector regression (SVR), linear regression and K-nearest neighbor (KNN).

c) Deep-Learning techniques for Interpolation without probability Approach:

Interpolation without calculating probability is discussed here. Instead of having probability map generation, we have directly used the interpolated results from 10 randonmly base-stations to train deep-learning models, for predicting the optimised fingerprinting location.

Training basic machine learning models for this approach has yielded better rmse scores as compared to previous two approaches upto some extent. The running time of this training model has improved too. While deep-learning neural network models have behaved the best in terms of accuracy and log-loss scores. Trying different permutation and combinations of activation functions, optimizers and tweaking the neuron counts for hidden layers, changing batch-size have produced better accuracy in terms of previous approaches. The running time of these classifiers are more in comparison to basic machine learning algorithms as it takes time to run training data segregated into batches. Fig. 12, describes the overview of our fingerprinting approach using Deep-Learning.

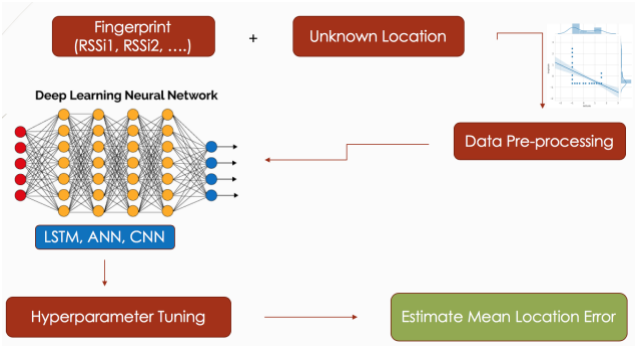


Fig 12: Overview of Fingerprinting Localization using Deep Learning

From the Fig. 13, it shows the deep learning models has out-performed basic machine learning models and LSTM is our best model giving mean location error of 191.52 [m].

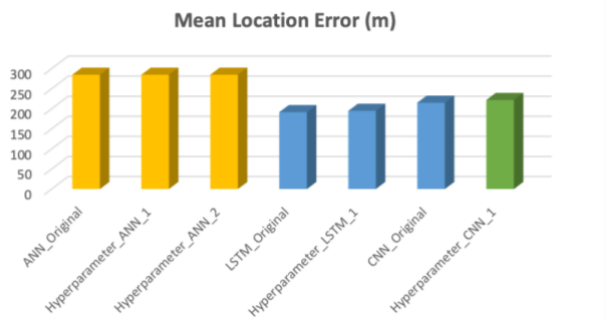


Fig 13: Mean Location Error [m] Outdoor DL Results

Hyperparameter tuning is a part of model optimization, in order to minimize the testing error. They are the settings that can be tweaked to maintain the behavior of an algorithm. Choosing a specific number and diversity of these parameters is very specific to each classifier. We have tweaked number of hidden units, learning-rate, batch-size, optimizers, activation functions and dropout layers. We have improved our results using hyperparameter tuning as described above. Additionally, the cumulative distribution frequency function vs mean distance error has been plotted in Fig. 14. It shows that 60% of LSTM has mean distance error of 180-200 [m] range, while CNN has 200-380 [m] mean error. Thus, LSTM performs better in our implementation.

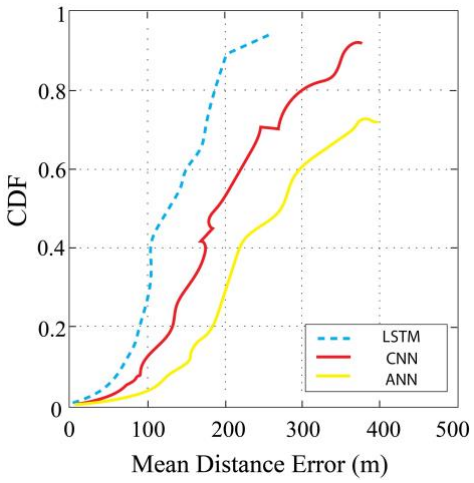


Fig 14: CDF vs Mean Distance Error [m]

d) Impact of different design parameters:

- *Impact of GPU vs TPU on Deep Learning Models*

The results we have gotten so far were ran on GPU hardware accelerator. We have compared it's results with the deep-learning training done on google-colab's TPU accelerator. Our experiments were running on a smaller convolutional neural network and we saw our results were skewed as custom network are not optimized for TPUs. Our dataset had approximately 120K records, divided into 90K training samples and 30K testing samples. We ran our experiments as described here in different scenarios below.

Scenario 1:

Model: LSTM with 3 hidden layers and dropout = 0.1
Total epochs: 100
Activation function: sigmoid

Accelerator	**GPU**	**TPU**
Training Accuracy (%)	94.5	93.2
Validation Accuracy (%)	78.2	84.4
Time/iteration(ms)	102	172
Time/epoch (s)	104	167
Total Time(mins)	22	18

Scenario 2:

Model: CNN with 3 hidden layers and 2 dense layers
Total epochs: 100
Activation function: sigmoid
Iterations per epoch: 100

Accelerator	**GPU**	**TPU**
Training Accuracy (%)	92.5	91.2
Validation Accuracy (%)	76.5	82.2
Time/iteration(s)	110	178
Time/epoch (s)	113	172
Total Time(mins)	25	21

From these experiments, we observed that training time in TPU is considerably more than the training time for GPU when the batch size is small. But, when the batch size increases, the performance of TPU is better than GPU.

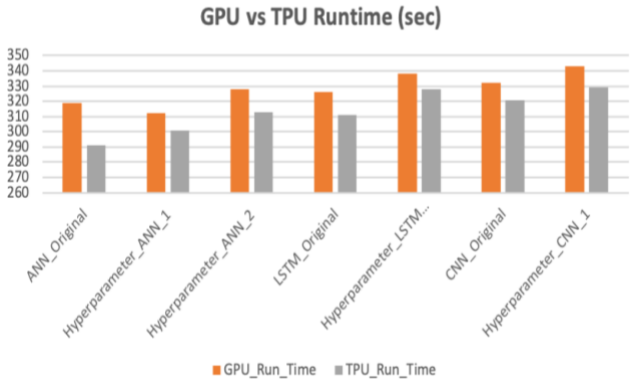


Fig 15: Model Performance on different Hardware Accelerators

- *Impact of Activation Function and Optimizer in Deep Learning Models*

The weight factor and activation functions in deep-learning neural networks have proved very important in model performance improvement while training it during batches. Selection of in-appropriate activation function and optimizer can lead to loss of training data during forward propagation and persistent vanishing gradient-descent problem during back-propagation.

Therefore, the impact of smoothness of these functions on the training data shows that sigmoid performs better than relu as shown in below table.

Classifier	**Activation Function**	**Optimizer**	**Model Performance**
LSTM	ReLu	Adam	193.98
CNN	Tanh	Rmsprop	202.55
ANN	Sigmoid	SGD	212.24
LSTM	ReLu	Rmsprop	195.24
LSTM	Sigmoid	Adam	191.52

CNN	Tanh	Adam	210.23

- *Impact of number of LSTM Layers*

To compare the number of LSTM layers and its effect on the mean distance error, we build 6 deep LSTM models 6 layers respectively. Eventually, we evaluated the performance and it's result against mean distance errors are presented in Fig. 16 below. We deduced that there is a decrease in mean distance error as more layers were used. The smallest possible error was 191.52 [m] for LSTM layer 2 and 3, respectively.

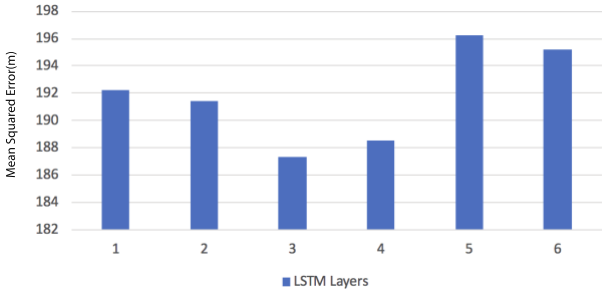


Fig 16: Impact of changing LSTM layers on Mean Error

- *Impact of LSTM model and varied batch size*

The LSTM model, we have used here has 100 epochs. At the end of each training epoch, the weights will be updated. The internal state of LSTM has three hidden layers of 60,30,15 neuron counts. To improve our prediction, we have different solutions implemented.

Solution 1: Changing the batch size = 10, Online Learning

Here, batch size is assigned to 10 and weights of the network are updated after each training example. Hence, by this approach we can have faster learning.

Solution 2: Changing the batch size=N, Batch Forecasting

Here, we are making all predictions of the batch at one single time.

Fig 17: Impact of Batch Size on Mean Distance Error

Solution 3: Copying weights

Different batch sizes for training and predicting are used here. This approach of copying the weights from the network and to create another new network, having pre-trained weights.

Hence, we need to vary the batch size needed for training and prediction with the network. Varying LSTM configuration for batch-based learning and prediction has helped to improve our results.

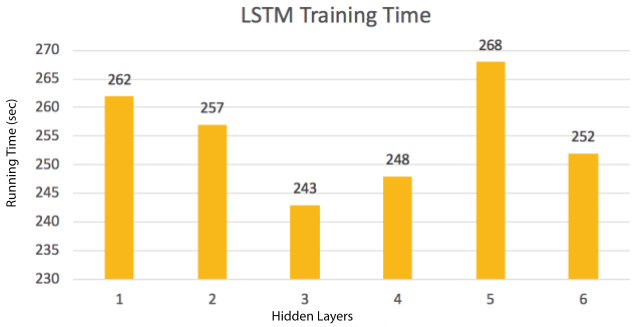


Fig 18: Impact on LSTM training time with varying hidden layers

D. Indoor Experimental Results

The motivation of this indoor experiment is to find a location of Lora node using the RSSI signals of the Lora gateways. The fingerprinting method as discussed in outdoor localization has been used here in offline stage. Due to wide fluctuations of RSSI values form base-stations in an indoor environment, due to non-linear relationship between sensor-node and gateways. There are other problems like instability of fading, device variance and multi-path effects i.e. different sensor nodes have different RSSIs even for same location. Using machine-learning models like KNN, one can determine the node-location by measuring and determining the fingerprint distance at the unknown point in the database. But there are still challenges of non-linearity with these algorithms.

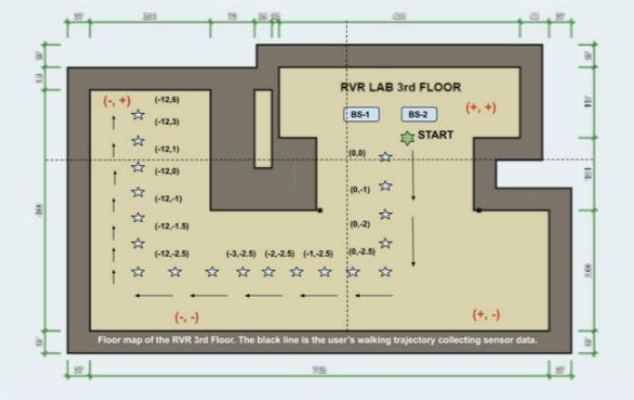


Fig 19: Floor map of indoor test location. RVR 3rd Floor CSUS. The black-arrow is the sensor-node's data trajectory.

Fig. 19 describes the indoor environment virtually divided into (X, Y) co-ordinate system. The base-stations are kept fixed in the lab while Lora nodes sends data to base-stations from different locations. To solve the problems discussed above for indoor environments, we have proposed deep-learning models like Artificial Neural Networks (ANN), Long-Short Term Memory (LSTM) and Convolutional Neural Networks (CNN). Our best performing model was LSTM as it is a special type of RNN. It has special units in addition to standard units, which includes 'memory cell' to maintain information for longer periods. The gates of LSTM are used to control flow of information in the network. Our approach uses sequential LSTM which removes the assumption of speed noise, by collecting the data at different locations. It adjusts the internal weights through training stage. The weighted average filter is applied to overcome the signal instability issues during training and test RSSI calculations. The performance optimization using different hyperparameter tuning has been proposed with varying batch-sizes, activation functions, optimizers, neuron counts and hidden layers. We have also compared our results with other deep-learning models like ANN and CNN, which has performed equivalently well in predicting the location error ranging from 1-2 (m) as shown in Fig. 19.

Fig 19: Mean Location Error [m] from Indoor experiments

V. Future Work

There are lot of future enhancements that can be carried out using methodology or modifying our process. The process of verifying results at same location but using different sensor nodes can generate different results as we gather more indoor data in all four directions. Additionally, classification approach can be implemented using deep-learning techniques based on RSSI values and it ranges. The Deep Learning approach can also be implemented considering BLE, Wi-Fi and new technologies like NB-IoT and compare results with that of Lora. Security implementation of Lora-wan in indoor localization using deep-learning can be very important in research areas.

VI. Conclusion

This paper has described the challenges in fingerprinting localization in an indoor and outdoor environment. Our results described in previous section about outdoor localization on Antwerp dataset improves the mean location error by using deep-learning techniques. These algorithms were implemented in order to improve previously defined results and to analyze short and long-term fluctuations in signal strengths over a large area in the city and understand capabilities of Lora as a technology in comparison to Bluetooth and Wi-Fi. This paper also described a novel approach for fingerprinting localization using Lora based on deep-learning techniques. Our idea was to analyze indoor localization problems like spatial ambiguity, signal-strength fluctuations and non-linear relations in a typical indoor infrastructure layout and how Lora and deep-learning can help us mitigate them. Besides, the analysis and impact of different parameters of vanilla ANN, LSTM and CNN have been discussed in detail.

VII. REFERENCES

[1] Recurrent Neural Networks For Accurate RSSI Indoor Localization - Minh Tu Hoang, Brosnan Yuen, Xiaodai Dong, Tao Lu, Robert Westendorp, and Kishore Reddy

[2] Sigfox and LoRaWAN Datasets for Fingerprint Localization in Large Urban and Rural Areas - Michiel Aernouts, Rafael Berkvens, Koen Van Vlaenderen and Maarten Weyn

[3] DeepML: Deep LSTM for Indoor Localization with Smartphone Magnetic and Light Sensors - Xuyu Wang ; Zhitao Yu ; Shiwen Mao

[4] Deep Convolutional Neural Networks for Indoor Localization with CSI Images - Xuyu Wang ; Xiangyu Wang; Shiwen Mao

[5] Low-Power LoRa Signal-Based Outdoor Positioning Using Fingerprint Algorithm - Wongeun Choi, Yoon-Seop Chang, Yeonuk Jung and Junkeun Song

[6] Localization in Ultra Narrow Band IoT Networks: Design Guidelines and Trade-Offs - Hazem Sallouha , Sreeraj Rajendran