Deep Learning Techniques for Fingerprinting Localization using LoRa

\*Note: Sub-titles are not captured in Xplore and should not be used

***Abstract* — Localization in long-range Internet of Things networks is a challenging task, mainly due to the long distances and low bandwidth used. Moreover, the cost, power, and size limitations restrict the integration of a GPS receiver in each device. In this work, we introduce a novel received signal strength indicator (RSSI) based localization solution for ultra narrow band (UNB) long-range IoT networks such as Sigfox. The essence of our approach is to leverage the existence of a few GPS- enabled sensors (*GSN*s) in the network to split the wide coverage into classes, enabling RSSI based fingerprinting of other sensors (*SN*s). By using machine learning algorithms at the network backed-end, the proposed approach does not impose extra power, payload, or hardware requirements. To comprehensively validate the performance of the proposed method, a measurement-based dataset that has been collected in the city of Antwerp is used. We show that a location classification accuracy of 80% is achieved by virtually splitting a city with a radius of 2.5km into seven classes. Moreover, separating classes, by increasing the spacing between them, brings the classification accuracy up-to 92% based on our measurements. Furthermore, when the density of *GSN* nodes is high enough to enable device-to-device communication, using multiliterate, we improve the probability of localizing *SN*s with an error lower than 20m by 40% in our measurement scenario.**

***Index Terms*—Internet of Things, ultra-narrow band, localiza- tion, RSSI, fingerprinting, machine learning**

# 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 Internet of Things, connecting sensors devices to cloud and enabling real-time interaction of data to enhance efficiency and productivity.

Positioning these devices is the most important thing in these 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.

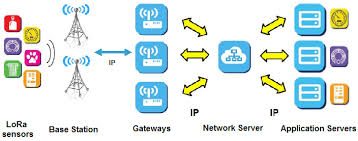
Moreover, fingerprinting localization techniques has two phases: An offline phase (training) and an online phase. In offline phase, received signal strength indicator (RSSI) from base-stations *aka* gateways are collected and stored into database. These data is sometimes referred as fingerprints. Subsequently, in online phase, locations of sensor nodes are predicted accurately by comparing current RSSI values with the values stored in database. Using interpolation techniques like linear, cubic and denoising auto-encoder, we have recovered missing values in the training phase. In this paper, we are comparing our datasets results collected over a period of time from university floor with open-source antewarp lorawan dataset collected from 68 base-stations. 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 cluster. Eventually, we will compare the results in TPU cluster to optimize our prediction results on test-data.

# Background and Problem Statement

In this section we introduce the methodology of interpolation using denoising auto-encoders in offline phase. Subsequently, the localization prediction problem and to improve it’s accuracy using deep learning neural network algorithms is thoroughly detailed.

## Background

LoRaWAN 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.



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. According to Fig. (1), various LoRa components like end-devices, gateways (base-stations), network server and application server have been described. End-nodes are physical hardware sensor devices that contain sensing capabilities and computing power upto 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.

## Problem Statement

Each LoRa message transmitted from a sensor edge device has to travel through multiple base-stations to reach to it’s 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.

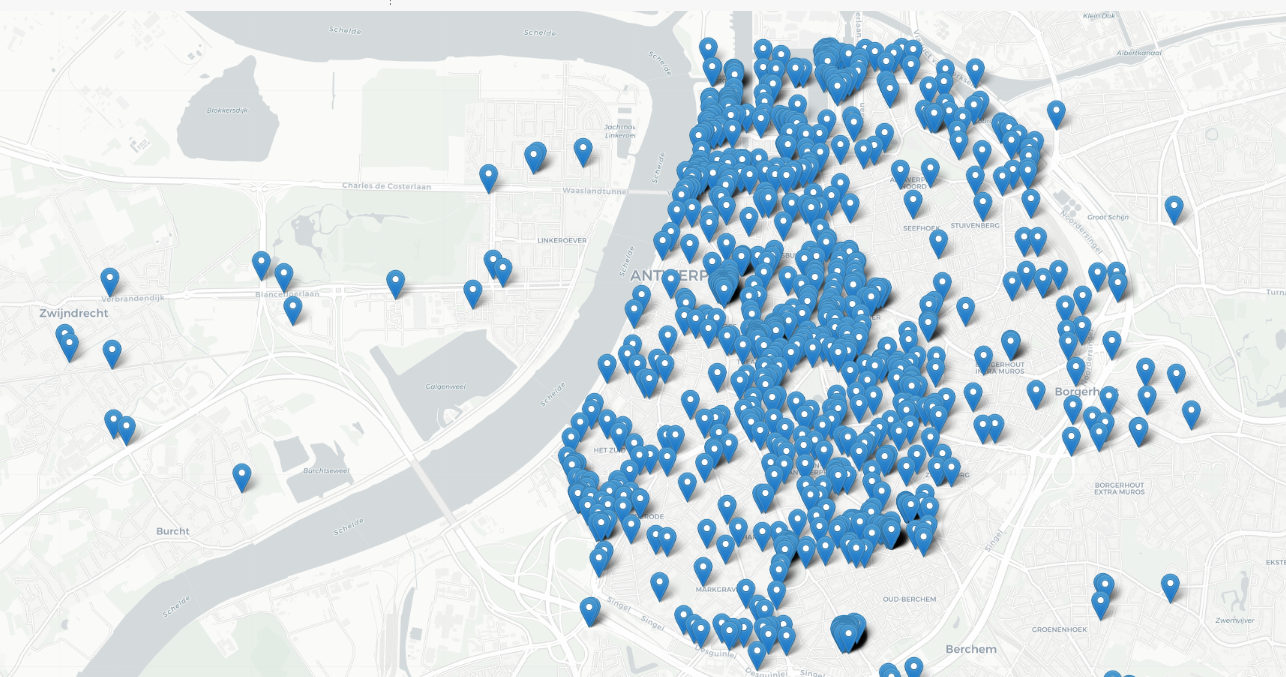


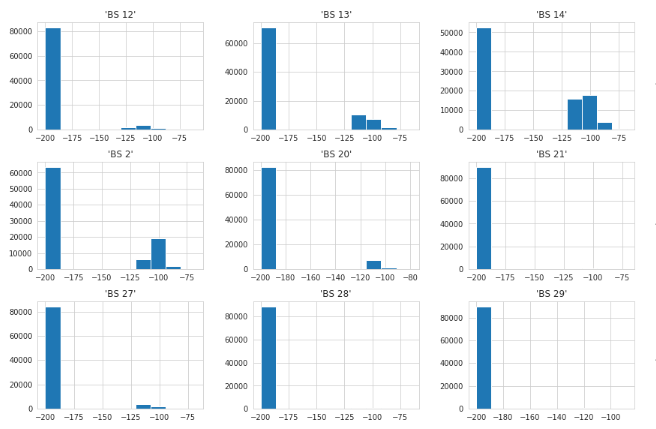
Fig 2. Multilateration on LoRa message transmission

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 for optimized coordinates in the online phase. We have compared publicly available lorawan dataset available from antewarp base-stations to our data collected from university floor over a period of time.

# interpolation using denoising autoencoders

## Fingerprinting map generation

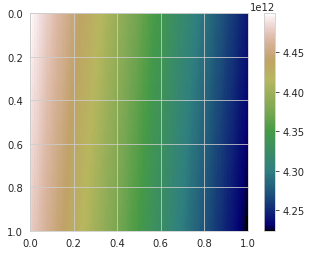
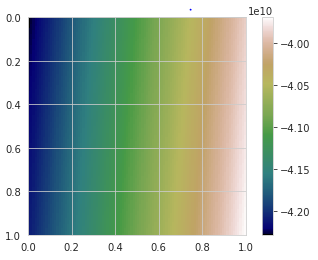
As discussed earlier, fingerprinting localization algorithm consists of two phases viz. offline or training phase and online phase. In offline phase, we collect data about signal strength on rea-time basis for various base-stations and are stored in the database for each sensor node in the service area. The uplink messages are transmitted to 68 base-stations from various sensor devices. These base-stations are located at different distances and can different range of RSSI values. According to Fig (3) , there are many RSSI values which are -200 and are considered outliers. In offline phase, we have made those values as null. Moreover, we select 10 random base stations and generate fingerprinting maps for them using interpolation techniques described in later sections. On the contrary, we have used many DHT11 sensors for this purpose in our university dataset. Eventually, in the online (positioning) phase, the location of the end-devices are estimated using deep-learning algorithms, which are discussed later in this paper.



## 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 implemented 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 multilateration process has 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. By collecting sample points, these interpolated results are represented here in figure (4) as heatmaps for couple of base-stations.

  
Figure (4) Interpolated results from two base-stations

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 are 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. Figure (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.

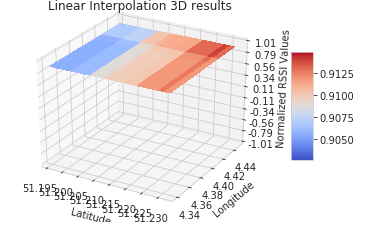
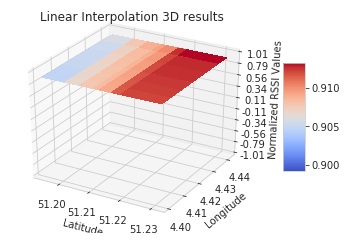
 

Figure (5) Linear Interpolation of two base-stations

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

## 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.

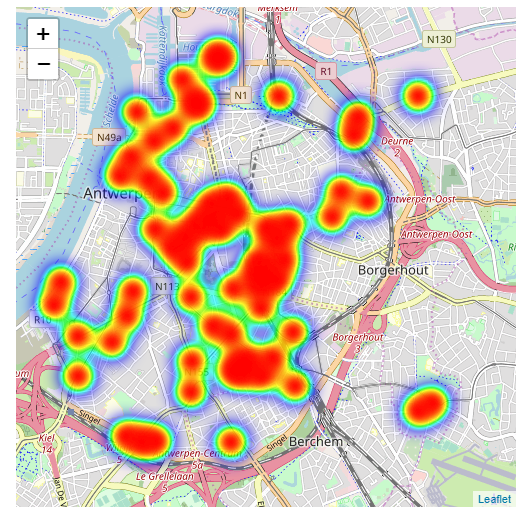
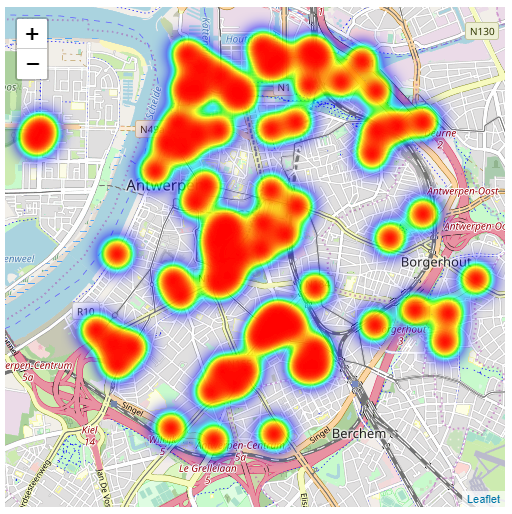
 

Figure (6) Interpolation results from autoencoder denoising

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. Figure (6) 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.

# deep learning algorithms to predict location

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.

* 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

## Basic ML Algorithms

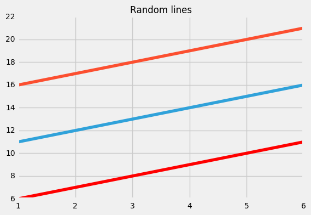
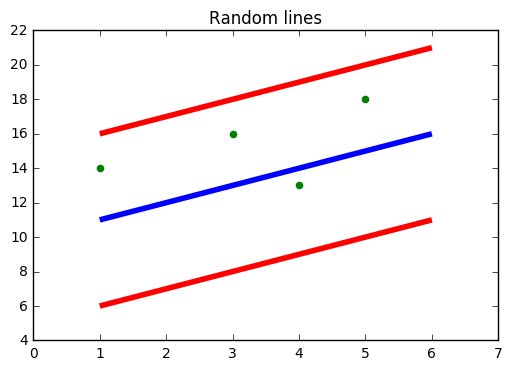
#### Simple Linear Regression

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, rsme and r2 scores for our prediction using linear regression. Our results vary according to the approach that we select as discussed above.

#### 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.

In Figure (7), blue-line indicates hyperplane which is the separation line between data classes, while red-line indicates 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 predition. We also noted that tuning the parameters, we have improved our metrics.

#### K-Nearest Neighbour (KNN)

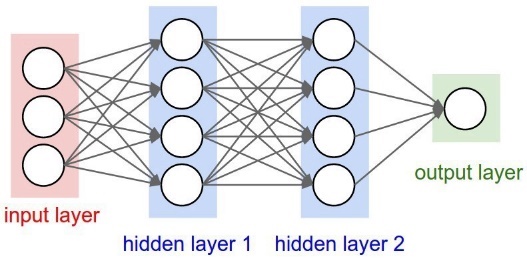
K-Nearest neighbor is used to store all available use-cases and predict the numerical output based on distance functions. KNN has been used in pattern recognition and statistical estimation. The distance function we have used in our KNN implementation is Euclidean distance as shown in figure 8.



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.

## 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 figure (5).



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.

#### 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 layers 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.

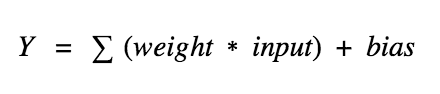


Figure 9. Neural Network Neuron Equation

Figure (9) 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.

#### Long Short Term Memory (LSTM)

Long short term memory are 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 figure 10.

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.

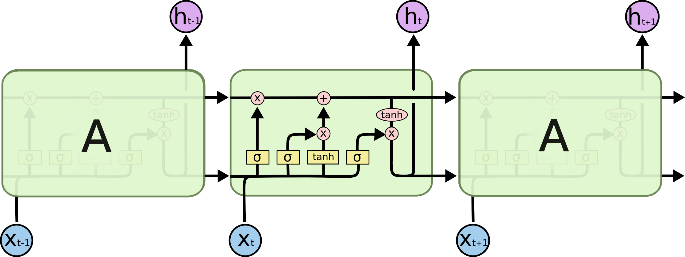


Figure 10. LSTM Architecture with four interacting layers

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 problem, we have created xarray and yarray matrix. These matrix are returned as numpy arrays while calling sequence() function. These sequence function is 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)))*

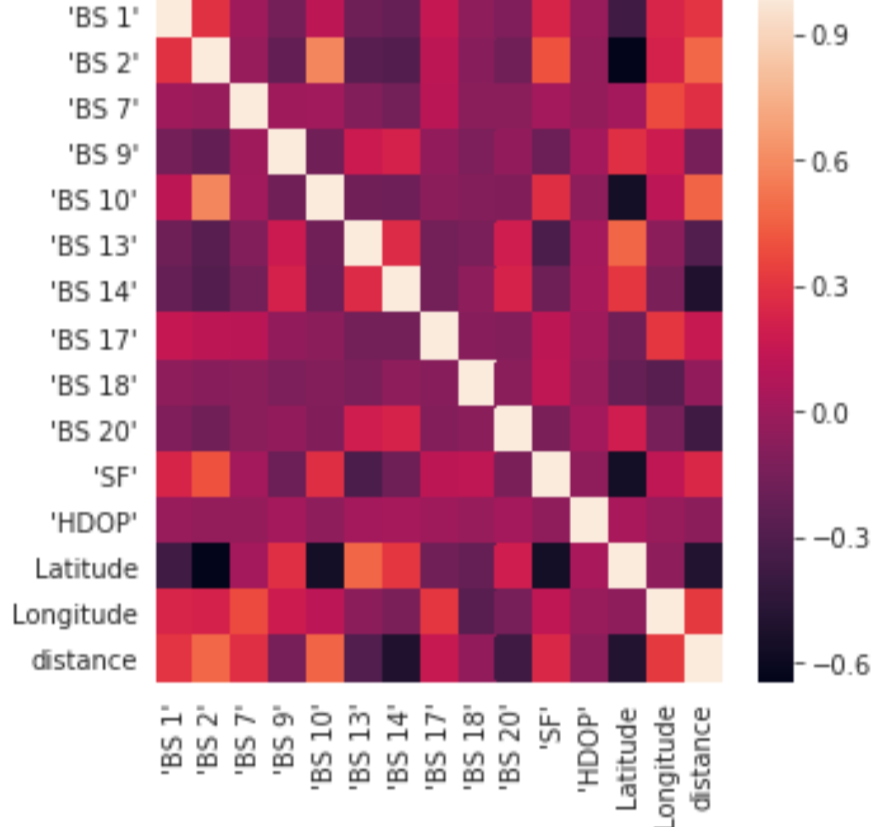
Dropout is a regularization method where recurrent connections to LSTM and input are removed from activation and weight changes while training the network. Dropout are used here to avoid the reducing overfitting and improving model performance.

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 masks 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-squred-error has been used as log-loss function along with adam as an optimizer.

## 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 deliberately made null values of outliers and imputed them using different interpolation techniques. Amongst them, denoising auto-encoders yielded best imputation results of those outliers.

Figure 12 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 dataframe. Here, the correlation value of a variable with itself is 1. For that reason, the primary diagonal values are 1.00 always.



#### 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 discussed below in the table below.

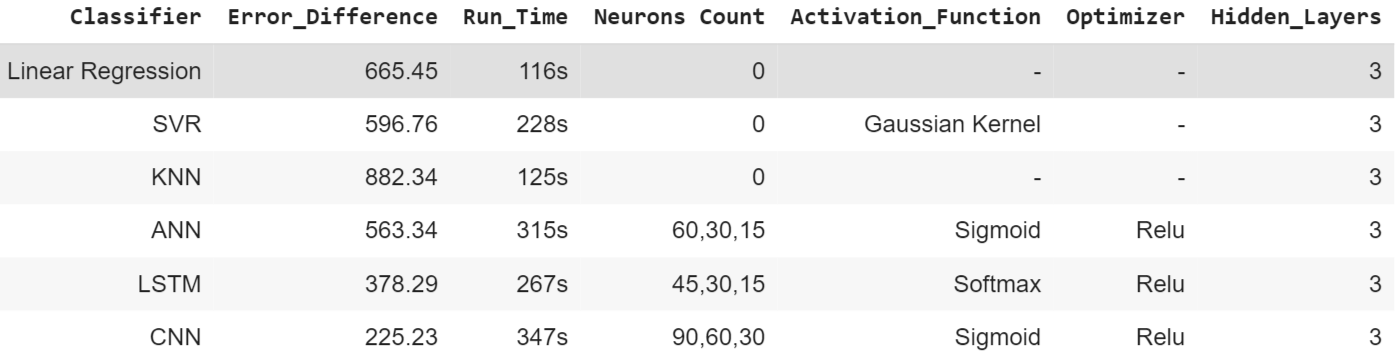


Figure 16. Naïve Approach results

#### 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.

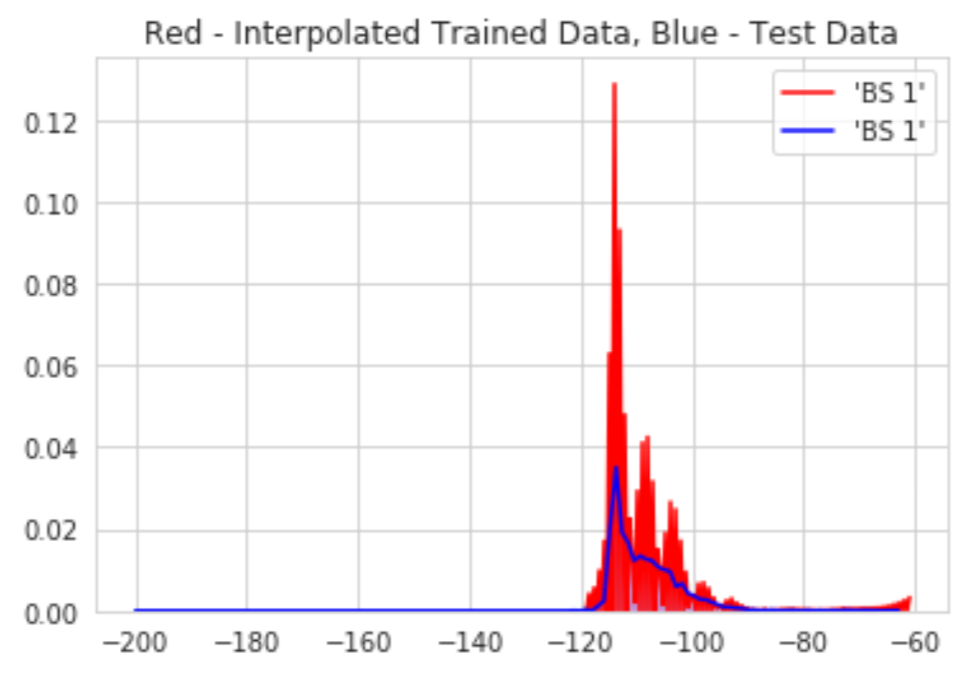


Figure 13. Interpolation results of trained data vs test data

Using denoising Autoencoders, interpolation of outliers in offline phase was carried out and the results of trained data vs test data is shown in figure 13. The RSSI values of end-devices are compared with the 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. Figure 14 describes about the probability scores obtained from each base-stations along-with its signal strength for a given latitude and longitude.

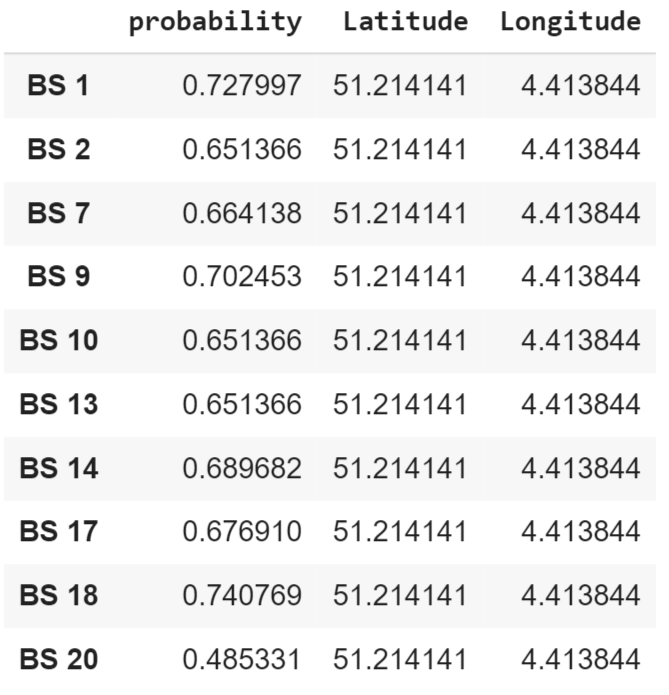


Figure 14. Probability scores from each base-stations

From the probability map table shown in figure 14, we can deduce that base-station 18 has the maximum probability score obtained 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.

For ANN, we have three hidden layers with 60,30,15 neuron counts, activation function used here is softmax and optimizer is adam. Trying different permutations and combinations have yielded us rmse score of 533.65 m. We have drawn epoch v log-loss graph for this model in figure 15. It describes the loss on the training and validation datasets over training epochs.

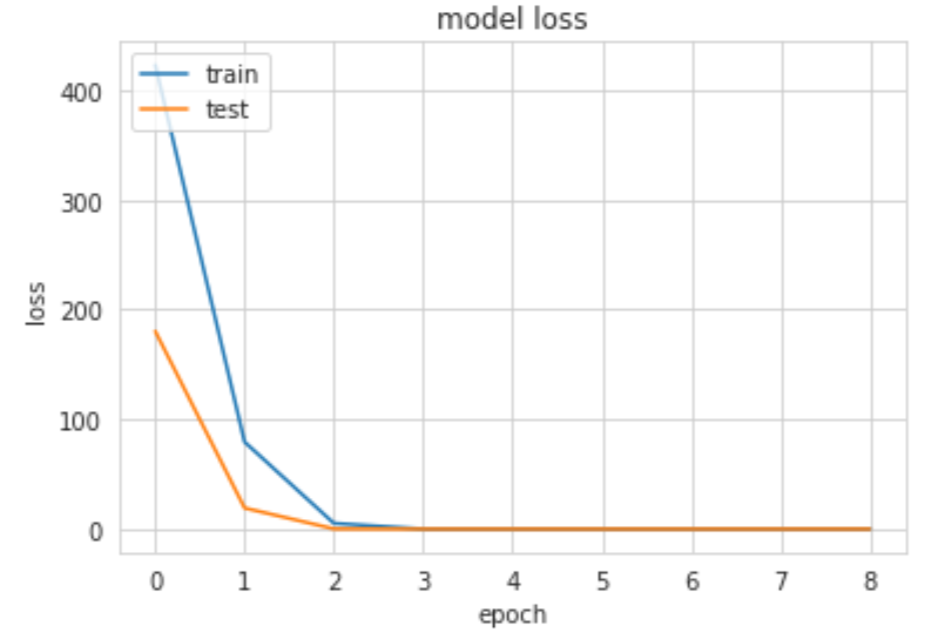


Figure 15. Plot of loss on training and validation for ANN

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). The results are listed below in figure 19.

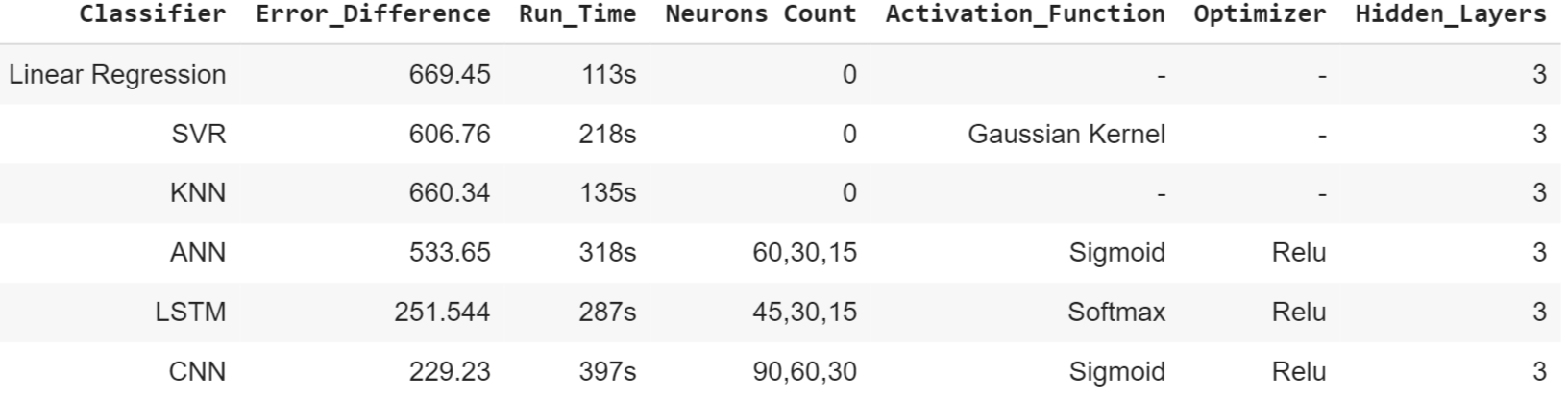
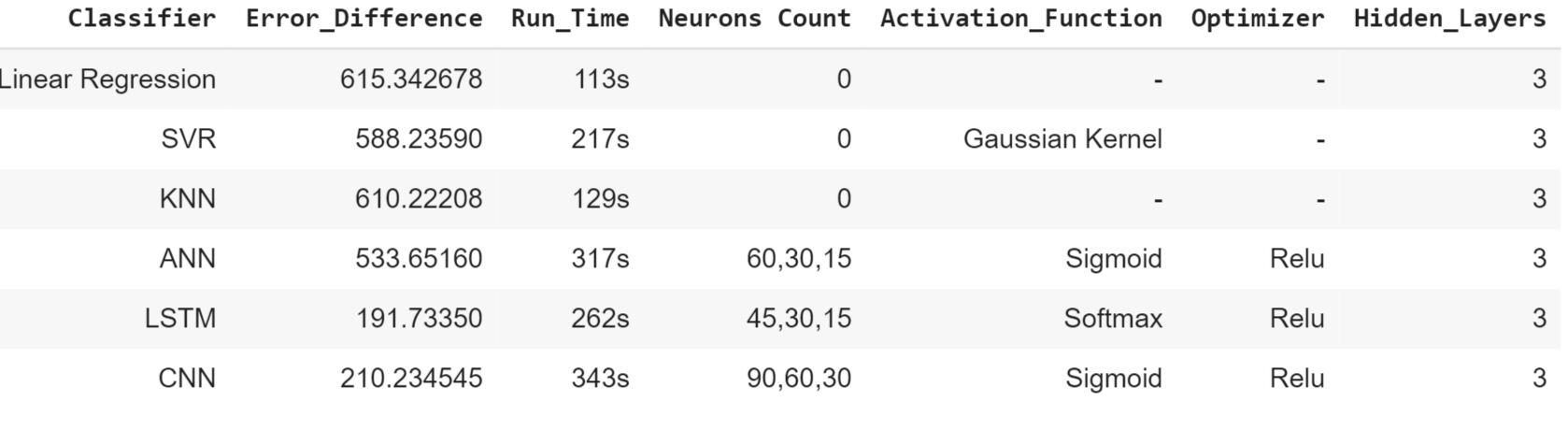


Figure 19. Interpolation with Probability Approach Results

#### 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 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. Figure 20. describes the results from this approach.



From the figure 20., it shows the deep learning models has outperformed basic machine learning models and LSTM is our best model giving mean error difference of 191.73 m.

#### 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. We have compared it’s results with the deep-learning training done on google-colab’s TPU accelerator. Our experiments were ran 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

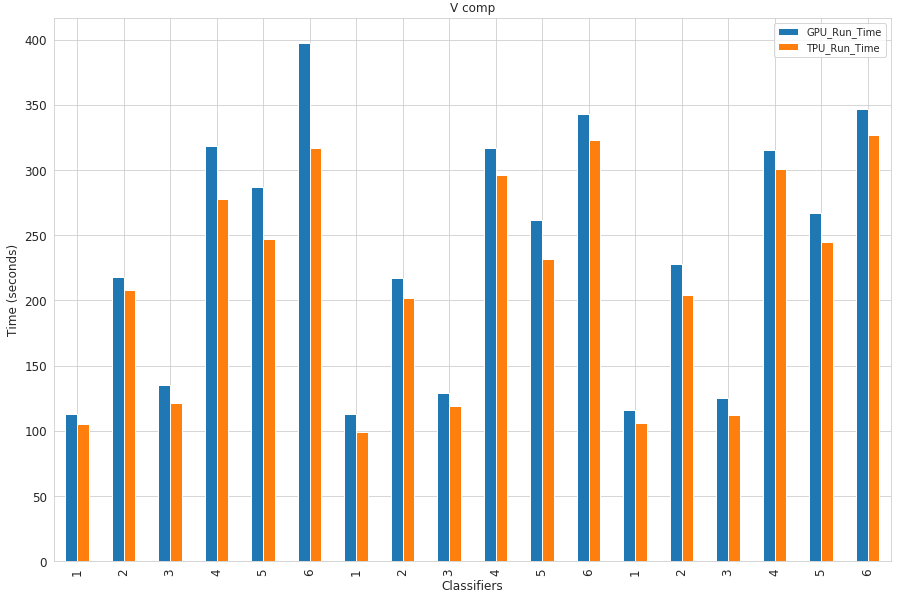
Total epochs: 100

Activation function: sigmoid

Iterations per epoch: 100

|  |  |  |
| --- | --- | --- |
| **Accelerator** | **GPU** | **TPU** |
| Training Accuracy (%) | 92.5 | 91.2 |
| Validation Accuracy (%) | 76.5 | 82.2 |
| Time/iteration(ms) | 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.



* Impact of Activation Function and Optimizer in Deep Learning Models
* Impact of number of hidden layers of ANN and LSTM

# conclusion

##### Acknowledgment *(Heading 5)*

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

##### References

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first ...”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

1. G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*
2. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
3. I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
4. K. Elissa, “Title of paper if known,” unpublished.
5. R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.
6. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
7. M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

**IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove template text from your paper may result in your paper not being published.**