Using Neural Networks for RSSI Location Estimation in LoRa Networks

Ioannis Daramouskas
Computer Technology Institute and Press "Diophantus"
Patras, Greece
daramousk@cti.gr

Vaggelis Kapoulas
Computer Technology Institute and Press "Diophantus"
Patras, Greece
kapoulas@cti.gr

Michael Paraskevas
Computer Technology Institute and Press "Diophantus,
and
Electrical & Computer Eng. Dept.
University of Peloponnese,
Patras, Greece
mparask@cti.gr

***Abstract*—Localize objects in IoT applications is a very popular trend nowadays. In this work we will present a neural network trained to estimate the location of an object in (Latitude and Longitude) using RSSI measurements through LoRa messages. The results of our work proving that neural networks can be used to in order to predict the location of objects in these kind of networks as our best model achieved average error on the test set 0.34 km, which is better from the average error without using neural networks..**

Keywords—neural networks, LoRa, localization, RSSI

I. INTRODUCTION

A. Intro

Object Localization in LoRa [1] [2] communication technology is a research topic that many researchers are focusing on nowadays. LoRa [1] [2] is a long range communication technology that allows communication between transmitter/receiver in wide ranges using low data rate and consume low energy, characteristics that are very desirable in IoT (Internet of Things) [3] applications. Our aim is to create a neural network to estimate the location of objects (object localization), which are communicating with base stations using LoRa messages. The data we are using in order to perform localization of the object are the Received Signal Strength Indication (RSSI) [1] [4] [5] [6] measurements the base stations receive from the moving object. The RSSI is based on measuring the power present in a signal between receiving station and objects. RSSI readings observed at the receiving stations, which can be used to estimate the corresponding distances from the receiving stations, with the use of mathematical models (known as path loss models) [6], where the signal attenuation is correlated with distance through mathematical formulas.

The rest of the paper is organized as follows: In section I B, we mention related work on the localization Low Power Wide Area Networks (LPWAN) [2] [1] networks. In section II we are describing methods we used to create a model in order to estimate the location of an object. Section III presents the experimental study and the corresponding results and section IV gives a conclusion of our study and presents the future work.

B. Related Work

Most of the works on the topic of localization of objects in LoRa networks focused on techniques like trilateration, multilateration, triangulation, fingerprinting [1] etc. In [6] the authors used trilateration method where they convert the RSSI readings into distances and solve linear systems and some other techniques to reduce the error. In [7] the authors use PSO (Particle Swarm Optimization) algorithm to estimate an objects location. In [8]the authors use the method multilateration where they exploit TDOA (Time Difference of Arrival) as input data to estimate the position. Finally, in [9] authors use the kNN fingerprinting localization method. We used the same dataset and we compare the fingerprinting method and its results with our model's performance, where we try to estimate an object's location using neural networks.

II. METHODS

A. Neural Networks

Artificial neural networks are computer systems inspired by the biological neural network that constitute animal brains. Nowadays they are commonly used in various applications for several reasons such as performance, the amount of available data for a big variety of applications and services and the increased computer power. An artificial neural network enables the computer to learn from examples without being programmed with any task-specific rules. An artificial neural network is based on a collection of connected nodes called "neurons". A neuron can have multiple inputs and one output. The output of a neuron is computed by the sum of its scaled inputs (each input is multiplied with the corresponding weight) and then passing the sum to a non-linear function called activation function to produce a single number. An activation function defines the output of a neuron. This output is given as input to neurons of the next layer. A connection between neurons is called "synapsis" and each synapsis has a weight. The learning process of a neural network includes two phases.

Work funded by the European Regional Development Fund (ERDF) and the Greek State under project "Wearable based Search And Rescue system" (WeSAR) / EPAnEK /NFRF 2014-2020.

The forward propagation and backpropagation. In the forward propagation the train data passed to the network in order to make predictions. When this procedure is finished and we have the predictions for the data we passed through the network, we use a loss function to evaluate the achieved performance. The loss function we used in our experiments is the mean squared error given by formula:

$$MSE = \frac{1}{n}\sum_{1}^{n}(Y_t - Y_p)^2 \quad (1)$$

where Y_t is the the true output, Y_p is the predicted output and n is the number of samples we let to pass to the network in order to calculate the error. When the loss has been estimated, this information is propagated backwards, so the weights are adjusted in such a way that reduces the loss, using an optimizer called stochastic gradient descent. The learning process is finished either if the loss hits a desirable threshold or if a certain number of iterations is reached.

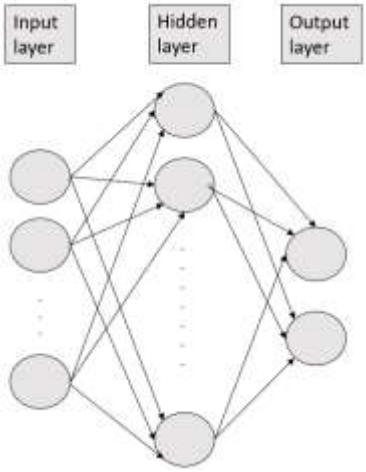

Fig. 1. Neural Network with one hidden layer and 2 output neurons

B. Activation Function

The most commonly used activation function is ReLU which takes as input the sum of the scaled inputs of a neuron and produces as output the positive part of its input. The ReLU activation function is described by the following formula:

$$f(x) = max(0, x) \quad (2)$$

In the equation 2, the parameter x expresses the sum of the scaled inputs of the neuron. In order to perform regression to estimate the position of an object, we also used linear as activation function to the output layer. The linear activation function is described by the following formula:

$$f(x) = x \quad (3)$$

This activation function is also called as identity, because it gives as output the sum of the scaled inputs. In the equation 3, the parameter x expresses the sum of the scaled inputs of the neuron.

C. Stochastic Gradient Descent

Stochastic Gradient Descent [10] optimizes the parameters Θ(weights and biases) of one network with the aim to minimize the loss L(in our case the mean squared error):

$$\Theta = argmin\frac{1}{n}\sum_{1}^{n} L(x_i, \Theta) \quad (4)$$

where $x_{1,\dots,n}$ are the data. A mini batch gradient descent of size m is considered $x_{1,\dots,m}$ and the approximation of the gradient of the loss to the dataset is calculated with the usage of mini batch [10].

III. Exepriments And Results

A. Dataset

The dataset we used to train our models is the lorawan_dataset_antwerp [9]. The dataset contains 123,529 LoRaWAN messages collected from 17 November 2017 until 5 February 2018 in the city of Antwerp, Belgium. Twenty post office cars drove around in the city center carrying a GPS receiver to record its position (latitude and longitude). The information about the location were send in a LoRaWAN message with the Horizontal Dilution of Precision (HDOP) of the GPS signal. There were 68 base stations, available to listen to the messages every car sent. Every row of the dataset represents a LoRaWAN message and indicates the receiving base stations, the timestamp of the message, the LoRa spreading factor, the HDOP, the latitude and the longitude of the transmitter at the current timestamp. We divided our experimental study into three categories. The categories are the following:

- Create models to predict the location of the object using unprocessed data
- Create models to predict the location of the object using processed data
- Create models to predict the location of the object using processed data using mean of measurements

In order to perform the experiments we used the high-level neural networks API Keras [11], which is written in Python and runs on top of TensorFlow library optimized to run on a NVIDIA 1050 Ti GPU with 4GB RAM.

The input of the neural networks was a vector, which contains the RSSI measurements in dB, that each base station received from a car. We conducted several experiments searching for the optimum parameters in order to make our models predict the position of an object with the minimum error according to the distance between the actual and the predicted location.

In each category we tried several topologies and parameters. The main aspect of our study for optimizing the models were:

- The number of hidden layers we used in our models [1, 2, 3, 4]
- Number of neurons in each layer [10, 20, 50, 100, 200, 500]
- Optimizer: Certain optimizers provided by Keras. The parameters we focused on were the learning rate, momentum, decay [sgd, adam, adagrad]

- Initializers: Several initializers provided, we try to find the best one to initialize the weights [normal, uniform, glorot_normal, glorot_uniform]

To evaluate the performance of our models we split the dataset into train and test sets (67%-33%) and we used the predicted output of the network to calculate the distance on earth between the actual and the predicted position. By getting these values we estimate the mean average error. In the models we configured in order to predict object's position, we selected after a series of experiments that proved to work better as initializer of the weights a Keras [11] initializer called "glorot_uniform", also called "Xavier uniform initializer". This initializer draws samples from a uniform distribution within the [-a, a] where:

$$a = sqrt\left(\frac{6}{fan_{in}+fan_{out}}\right) \quad (5)$$

where, fan_{in} is the number of inputs of the neuron and fan_{out} is the number of the outputs. As activation function at the hidden layers we used ReLU and at the output layer we used linear, in order to perform regression with two neurons at the output layer. The number of hidden layers is a crucial parameter for the models we configured. Usually as the number of hidden layers increase, model's accuracy also increases with higher risk of overfit. After a series of experiments, we present in this work the number of hidden layers which proved to achieve better performance (i.e. 1, 2).

B. Unprocessed Data

In this category we trained an artificial neural network using the dataset without making any process of the data it contains, except of the timestamp which we removed because they didn't add any information to the learning process. In this case we had 123.528 instances where every instance contains 68 attributes (i.e. base station measurements). In our experiments we use two hidden layers, as optimizer we choose stochastic gradient descent with momentum 0.9, Nesterov parameter set to true and we trained the model for 500 epochs. We searched for optimum values about the learning rate (Lr), the batch size (Bs) and the number of neurons (N). In the Table I the results of our experiments are presented. The columns E_train shows the average error in the train set, E_test the average error in the test set, the <100 the number of errors that are lower than 100 meters and >1 the number of errors that are bigger than one kilometer.

TABLE I. EXPERIMENTS WITH UNPROCESSED DATA

Exp	Parameters and results						
	Lr	*Bs*	*N*	*E_train*	*E_test*	*<100*	*>1*
1	10^{-3}	250	500	1.614	1.615	434	18243

In figure 2 we can see the losses for the train and the test sets, we observe that the lowest value for mean squared error is about 10^{-3} and the model we configured couldn't fit the data(learn) to make accurate estimation about the position of the object. As a result, we decide to use Batch Normalization [12] in order to help the model fit the data and achieve better performance in the task we assigned it to do. By using Batch normalization, the output of a previous activation layer is normalized by subtracting the batch mean and dividing by the batch standard deviation.

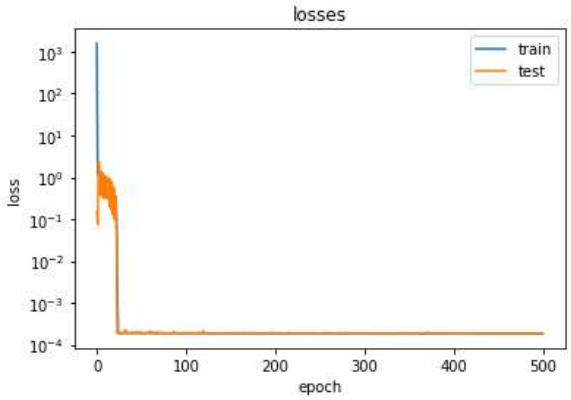

Fig. 2. Loss for train and test sets for the model 1 of t Table I

In the Table II we provide the parameters we used to train the model, and the results of each experiment. In those experiments we change only the number of neurons.

TABLE II. EXPERIMENTS WITH UNPROCESSED DATA

Exp	Parameters and results						
	Lr	*Bs*	*N*	*E_train*	*E_test*	*<100*	*>1*
1	10^{-3}	250	50	1.239	1.222	434	18243
2	10^{-3}	250	25	0.782*	0.782*	1697	10258
3	10^{-3}	250	100	2.101	2.155	256	27631
4	10^{-3}	250	10	0.857	0.858	396	11794
5	10^{-3}	250	15	0.893	0.893	436	12753

There was an improvement, cause the average error on test set reduced to 0.782, but still the performance of the model isn't sufficient enough. In the Table III we present experiments where we keep the best model of the Table II and in those experiments, we changed only the batch size using the same parameters as in Table II. The batch size is a parameter that defines the number of instances need to be passed to the network in order to perform the stochastic gradient descent optimizer.

TABLE III. EXPERIMENTS WITH UNPROCESSED DATA

Exp	Parameters and results						
	Lr	*Bs*	*N*	*E_train*	*E_test*	*<100*	*>1*
1	10^{-3}	150	25	0.750*	0.753*	1428	9010
2	10^{-3}	200	25	0.810	0.806	879	10080
3	10^{-3}	250	25	0.782	0.782	1697	10258

As shown in the Table III, we got a slight improvement in the model, but yet not enough. The next parameter we experiment

with is the learning rate. The learning rate is a parameter that is crucial to the learning process because it controls how much the weights would be adjusted, with respect to the loss gradient. In the Table IV below, we keep the best model of the Table III and we change the learning rate.

TABLE IV. EXPERIMENTS WITH UNPROCESSED DATA

Exp	Parameters and results						
	Lr	*Bs*	*N*	*E_train*	*E_test*	*<100*	*>1*
1	10^{-3}	150	25	0.750	0.753	1428	9010
2	10^{-2}	150	25	0.754	0.752	759	9979

Any change in the parameters of the model we did, didn't produce satisfying results regardless to the average error on the test set, so we decided to perform some preprocessing to examine the performance.

C. Processed Data

In this effort we try to preprocess the data in order to achieve better performance of the models. First, we found out that some of the base stations occurs only 10-20 times in the dataset, therefore they contribute little in the learning process of the model and so we removed the instances in which those base stations had value bigger than -200 and afterwards we remove the corresponding base stations. After this procedure the remaining base stations of the instances are 51 instead of 68. Below in the Table V we present the results using the same network configuration with the preprocessed dataset we mentioned above.

TABLE V. EXPERIMENTS WITH PROCESSED DATA

Exp	Parameters and results						
	Lr	*Bs*	*N*	*E_train*	*E_test*	*<100*	*>1*
1	10^{-3}	150	25	1.006	1.006	421	17083
2	10^{-3}	250	25	0.897	0.898	941	11939
3	10^{-3}	200	25	0.779*	0.772*	1840	11039

In addition, we tried to change the number of neurons of the hidden layers, because the performance didn't improve by removing those base stations as shown in the table 5 but still no improvement. Subsequently we try to use the HDOP attribute of the dataset to remove the instances who had big measurement error. Also, we removed more base station that occurred in a small number of instances (i.e. 40,100, 200). So, the dataset reduced by 15153 instances and each instance had 31 base stations. Also, we used from Keras callbacks, the Learning Rate Scheduler, where we start the training with a high value for the learning rate, and we drop the value using the following equation:

$$lr = in\_lr * drop^{floor\left(\frac{1+epoch}{epoch\_d}\right)} \quad (6)$$

where lr is the learning rate used in each epoch, in_lr is the learning rate at the first epoch, drop is a value between [0,1], epoch is the current epoch and epoch_d is the number of epochs needs to pass in order to drop the learning rate. So, in this case the learning rate will be dropped every epoch_d epochs. We tried several experiments, the parameters we use initial learning rate=0.2, momentum=0.9, clipnorm=1, Nesterov momentum and two hidden layers and again using Batch Normalization [12] between the hidden layers. In order to achieve even better performance and to avoid overfitting we used the Keras callback ModelCheckpoint which is saving the best instance of the model based on the loss during all epochs of the training, in particular we choose to keep the best instance regarding to mean absolute percentage error:

$$M = \frac{100\%}{n}\sum_{1}^{n}\left|\frac{Y_t - Y_p}{Y_t}\right| \quad (7)$$

where n is the batch size, Y_t is the true output and Y_p is the predicted output. The results are presented in the Table VI.

TABLE VI. EXPERIMENTS WITH PROCESSED DATA

Exp	Parameters and results					
	Bs	*N*	*E_train*	*E_test*	*<100*	*>1*
1	250	25	0.525	0.538	4133	4539
2	250	50	0.501	0.523	2509	4338
3	250	100	0.506	0.530	2249	4398
4	250	200	0.485	0.532	3623	4815
5	250	500	0.468	0.513	2740	4130

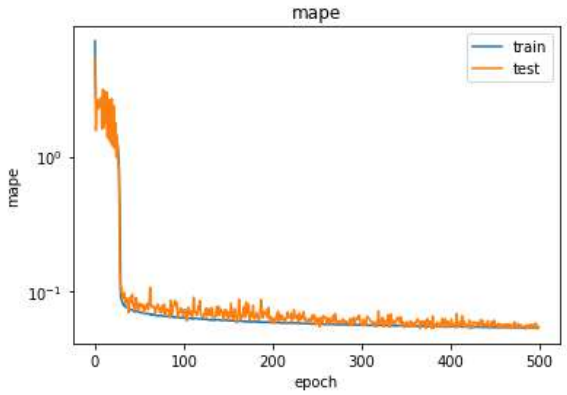

Fig. 3. Mean absolute percentage error for train and test sets for the model 5 of Table VI

As we can see in Table VI and in figures 3 and 4, the model we configured in these experiments improved a lot regarding to the average error on the test set if we compare it with the previous models, having mean absolute percentage error (mape) less than 10% and average error 0.513 which is big improvement compared to the best performance we had so far, but still there is a lot of room for improvement in our model's performance So , we decide to perform some preprocessing techniques to the input data and particularly we perform normalization to the input data, in order to transform them to be in the [0,1], using Keras's MinMaxScaler.

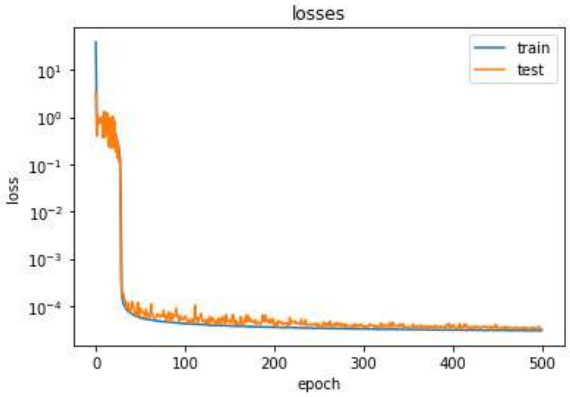

Fig. 4. Loss for train and test sets for the model 5 of TABLE VI

The results of the model we configured, trained with the normalized input data using the MinMaxScaler are presented in the Table VII

TABLE VII. EXPERIMENTS WITH PROCESSED DATA

Exp	Parameters and results					
	Bs	***N***	***E_train***	***E_test***	***<100***	***>1***
1	250	25	0.538	0.544	3765	4657
2	250	50	0.511	0.529	4145	4346
3	250	100	0.491	0.526	5383	4822
4	250	200	0.473	0.498	5443	4233
5	250	500	0.450	0.490	5758	4187

We can observe that the performance of the model is improved when we normalized the input data into [0,1] range, and we had bigger number of errors less than 100 meters, but there is still room for improvement. Finally, we decided to transform the input data to have zero mean and one standard deviation, which is a very commonly used preprocessing technique. In the Table VIII the results are presented from this kind of transformation of the input data.

TABLE VIII. EXPERIMENTS WITH PROCESSED DATA

Exp	Parameters and results					
	Bs	***N***	***E_train***	***E_test***	***<100***	***>1***
1	250	25	0.514	0.515	3.510	2863
2	250	50	0.476	0.475	3.596	2517
3	250	100	0.461	0.464	4119	2442
4	250	200	0.435	0.438	4641	2197
5	250	500	0.408*	0.419*	4808	2049

We can see that this transformation of the input data, transform them to have zero mean and standard deviation equal to one made the model we configured to produce the best results we have obtained so far in all of our experiments we have performed, with the average error on test set to be equal with 0.419.

In the Table IX we show the results from the experiments we perform for 2000 epochs, using the same models and parameters as in the Table VIII.

TABLE IX. EXPERIMENTS WITH PROCESSED DATA

Exp	Parameters and results					
	Bs	***N***	***E_train***	***E_test***	***<100***	***>1***
1	250	25	0.492	0.495	3824	2758
2	250	50	0.466	0.467	4081	2485
3	250	100	0.442	0.446	4037	2229
4	250	200	0.420	0.427	4528	2076
5	250	500	0.381*	0.394*	4969	1996

As we can see in Table IX and figure 5, figure 6 the performance of our model is improved a lot, if we compare the results with the results we get when we used the unprocessed data or the scaled data to [0,1] to train the models because the mean absolute percentage error as we shown in the figure 5 is close to 4% and the average error on the test set is 0.394, which is very satisfying and it is slightly better from the results obtained in [9] where the average error was 0.398. Still an error 0.394 regarding of the application can be not acceptable and so we continued our experiments, with the aim to reduce it furthermore using the mean of measurements.

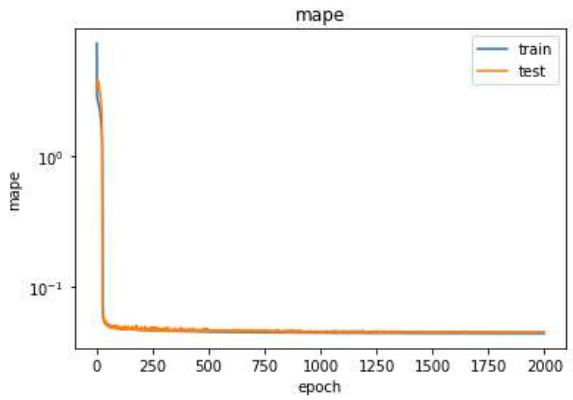

Fig. 5. Mean absolute percentage error for the model 4 of Table I

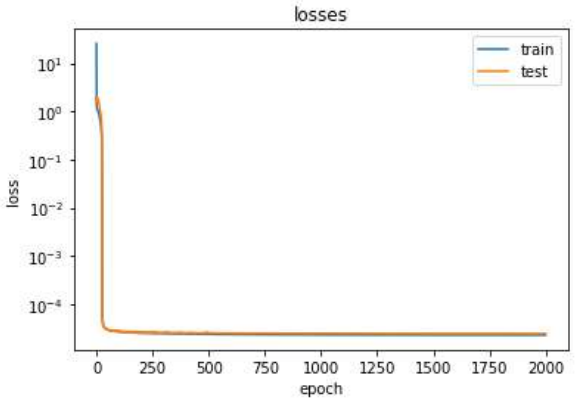

Fig. 6. Loss for train and test sets for the model 4 of Table IX

D. Processed data using mean of measurements

In this category of experiments firstly we transform the data into having zero mean and standard deviation equal to one and afterwards we try to take advantage of the information timestamp of the measurements provides us. There are several measurements within the same minute, so we decide to create a new dataset where an instance was produced with the mean of the measurements for every minute. The final dataset we use to train a model to estimate the location has size 49409 instances with 68 base stations and we split the dataset into 80% and 20% train set and test set. In Table X we present the results from the experiments where we change the number of the neurons in the hidden layers. The other parameters we use in those experiments are, initial learning rate=0.2, momentum=0.9, clipnorm=1, Nesterov momentum, two hidden layers and epochs=500.

TABLE X. EXPERIMENTS WITH PROCESSED DATA

Exp	Parameters and results					
	Bs	***N***	***E_train***	***E_test***	***<100***	***>1***
1	250	25	0.412	0.426	902	595
2	250	50	0.382	0.401	1733	599
3	250	100	0.374	0.356	1649	548
4	250	200	0.357	0.382	1745	528
5	250	500	0.335	0.366*	1855	483

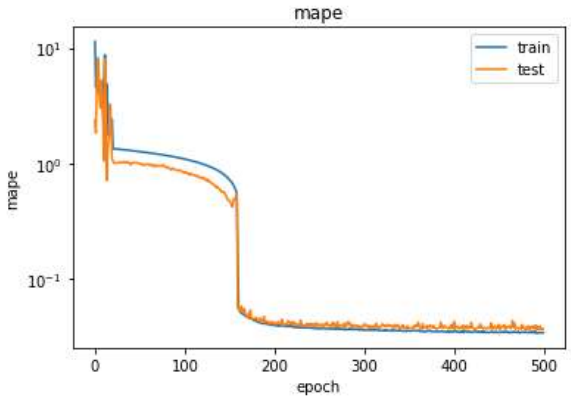

Fig. 7. Mean absolute percentage error for train and test sets for the model 5 of Table X

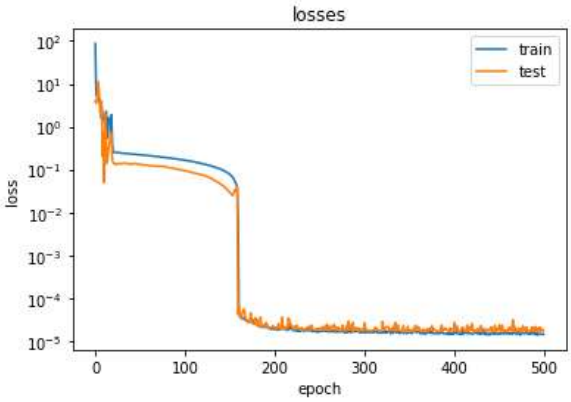

Fig. 8. Loss for train and test sets for the model 5 of Table X

We can observe from the Table X and the figures 7 and 8 that the model performs better than any other model we have tried so far, having mean absolute percentage error close to 3.5% and average error on test set 0.366. In the Table XI we keep the best model and we change the values of the initial learning rate and the batch size. We trained the model for 500 epochs, using batch size=80.

TABLE XI. EXPERIMENTS WITH PROCESSED DATA

Exp	Parameters and results					
	Lr	***N***	***E_train***	***E_test***	***<100***	***>1***
1	0.2	525	0.324	0.354	2047	428
2	0.1	525	0.311	0.345*	2161	444

Again, we can observe that in that case, changing the batch size gives us better performance on the model with reducing the average error on test set to 0.345.

In the Table XII we present the better models from each table we have presented.

TABLE XII. EXPERIMENTS WITH PROCESSED DATA

Exp	Results			
	E_train	***E_test***	***<100***	***>1***
1	0.782	0.782	1697	10258
2	0.750	0.753	1428	9010
3	0.779	0.772	1840	11039
4	0.468	0.513	2740	4130
5	0.450	0.490	5758	4187
6	0.408	0.419	4808	2049
7	0.381	0.394	4969	1996
8	0.335	0.366	1855	483
9	0.311	0.345	2161	444

We can see that the transformation of the data to have zero mean and standard deviation one is very helpful in that kind of data, cause it make the model perform much better reducing the average error on test set by 0.119 km, from 0.513 to 0.394 which is proves that we can use neural networks to estimate the location using RSSI measurements as input data. It is also obvious that the usage of mean of measurements was also really helpful to because the average error in test set dropped even more to 0.345.

IV. CONCLUSIONS

From the experimental study we performed, we obtain that we can use neural networks to predict the position of an object, using data from LoRa messages and, in particular, RSSI measurements from the base stations. We can see that compared to kNN fingerprinting neural network models can perform equally or even better. In our project, we aim to localize people who having high possibility of getting lost using LoRa network. An error of 300 m can is not necessarily unaccepted cause it

gives us a clue about the location of the missing people, so we can inform relatives or the authorities to search for him in the area we locate him. Nevertheless we will continue our work with the aim to reduce the error in Object localization in IoT, and will focus our future work performing unsupervised techniques for data preprocessing and analysis of the data, such us clustering (for grouping data based on the location on the map) and autoencoders (for dimensionality reduction and denoising), and then use neural networks to perform object localization, using data from LoRa messages.

ACKNOWLEDGEMENTS

Work funded by the European Regional Development Fund (ERDF) and the Greek State under project "Wearable based Search And Rescue system" (WeSAR) / EPAnEK (Operational Programme, Competitiveness, Entrepreneurship and Innovation) /NFRF (National Strategic Reference Framework) 2014-2020

REFERENCES

[1] C. Lagoudias, A. Moreira, S. Kim, S. Lee, L. Wirola and C. Fischione, "A Survey of Enabling Technologies for Network Localization, Tracking, and Navigation," in *IEEE Communications Surveys & Tutorials*, 2018.

[2] M. Saari, A. M. b. Baharudin, P. Sillberg, S. Hyrynsalmi and W. Yan, "LoRa — A survey of recent research trends," in *41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2018.

[3] U. Raza, P. Kulkarni and M. Sooriyabandara, "Low Power Wide Area Networks: An Overview," *IEEE Communications Surveys & Tutorials,* 2017.

[4] E. Goldoni, L. Prando, A. Vizziello, P. Savazzi and P. Gamba, "Experimental dataset analysis of RSSI-based indoor and outdoor localization in LoRa networks," *Internet Technology Letters,* 2018.

[5] B. C. Fargas and M. N. Petersen, "GPS-free Geolocation using LoRa in Low-Power WANs," in *Global Internet of Things Summit (GIoTS)*, 2017.

[6] K. H. Lam, C. C. Cheung and W. C. Lee, "LoRa-based localization systems for noisy outdoor environment," in *IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Rome, 2017.

[7] A. Rauniyar, P. Engelstad and J. Moen, "A New Distributed Localization Algorithm Using Social Learning based Particle Swarm Optimization for Internet of Things," in *IEEE 87th Vehicular Technology Conference (VTC Spring)*, Porto, 2018.

[8] N. Podevijn, D. Plets, J. Trogh, L. Martens and P. Suanet, "TDoA-Based Outdoor Positioning with Tracking Algorithm in a Public LoRa Network," *WirelessCommunicationsandMobileComputing,* 2018.

[9] M. Aernouts, R. Berkvens, K. V. Vlaenderen and M. Weyn, "Sigfox and LoRaWAN Datasets for Fingerprint," *Data ,* 2018.

[10] Wikipedia contributors, "Stochastic gradient descent," 29 3 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Stochastic_gradient_descent&oldid=890028980. [Accessed 17 5 2019].

[11] F. Kollet, "keras," 2015. [Online]. Available: https://keras.io.

[12] Wikipedia contributors, "Batch normalization," 11 5 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Batch_normalization&oldid=896612549. [Accessed 17 5 2019].