A Deep Learning Approach to Fingerprinting Indoor Localization Solutions

Linchen Xiao
RWTH Aachen University
Aachen, Germany
Email: linchen.xiao@rwth-aachen.de

Arash Behboodi
RWTH Aachen University
Aachen, Germany
Email: arash.behboodi@ti.rwth-aachen.de

Rudolf Mathar
RWTH Aachen University
Aachen, Germany
Email:mathar@ti.rwth-aachen.de

***Abstract*—Fingerprinting Localization Solutions (FPSs) enjoy huge popularity due to their good performance and minimal environment information requirement. Considered as a data-driven approach, many modern data analytics can be used to improve its performance. In this paper, we propose tow learning algorithms, namely a deep learning architecture for regression and Support Vector Machine (SVM) for classification, to output the estimated location directly from the measured fingerprints. The design issues of the proposed neural network is discussed including the training algorithm, regularization and hyperparameter selection. It is discussed how data augmentation methods can be utilized to extend the measurements. The deep learning approach can be used to save the data collection time significantly using a pre-trained model. Moreover the run-time complexity is significantly reduced. The numerical analysis show that in some case, only 10 percent of original training database is enough to get acceptable performance on a pre-trained model.**

I. Introduction

Precise location of people and devices in indoor environments is an essential enabler for future wireless networks and location-based services. Significant research has been conducted during recent years on indoor localization. In plethora of indoor localization algorithms, RF based approaches are particularly interesting given their technological accessibility. However the propagation in indoor environments follow complex models that should account for various sources of attenuation and deflection. Given the dynamics of indoor environments, the propagation models should also be constantly updated with changes in the propagation space. Therefore the algorithms that rely on explicit propagation models for localization require significant environmental awareness and continuous manual update. Fingerprinting-based methods, on the other hand, are not model-dependent. They are data-driven approaches working on the assumption that there are certain RF features capable of identifying a location uniquely and stably. The algorithm collect these features at different locations and constructs fingerprint for each point. The fingerprint collection can be done for only finite number of points in the space and it is the most time-consuming part of the algorithm. These pairs of fingerprints and locations are organized into a training database. The main problem of localization is, hence, to find the location corresponding to a new observation. Fingerprinting algorithms can usually be built on top of available infrastructures, such as WiFi networks and their model-independence and minimal infrastructure requirement make them a very attractive choice for fast indoor localization deployment. As their drawbacks, the training database should be updated and sometimes built anew when the environment changes and consequently so are the fingerprints. This is more troubling given the time consuming nature of data collection. In this paper, the goal is to look at the fingerprinting localization algorithms from machine learning point of view and show how these issues can be addressed using the modern learning architectures. Not only utilization of these learning architectures improves upon the classic algorithms but the techniques like model transfer and data augmentation can be borrowed from machine learning to accelerate new training base creation. The rest of the paper is structured as follows. Section II presents the general overview of fingerprinting algorithms particularly from data analytics point of view. Section III describes SVM approach to fingerprinting. In section IV deep learning architecture is proposed for fingerprinting. Different hyperparameter choices are discussed. Section V presents the numerical evaluations for different settings.

II. Architecture of Fingerprinting Algorithms

The localization space is given by the set $\mathcal{D} \subset \mathbb{R}^d$. The goal of localization is to infer the position of a target node placed at $\mathbf{u}$ in the region $\mathcal{D}$ based on the observations $\mathbf{S_u}$, usually a vector in Euclidean space. Therefore the task of localization can be understood as the problem learning the mapping of the observation $\mathbf{S}$ to the location $\mathbf{u}$. Fingerprinting algorithms are data-driven approaches where the localization task utilizes the data gathered from the environment as the basis for location inference. In fingerprinting algorithm, a set of true location-observation pairs $(\mathbf{u}, \mathbf{S_u})$ is known and the localization function is learned using the set mapping new observations to a position. The central task of localization is to learn the localization function denoted by Φ. The building blocks of fingerprinting algorithms are feature collection, fingerprint creation, pattern matching and post-processing [1].

A. Feature Collection

As any data-driven learning algorithm, the fingerprinting algorithm requires the collection of data. In this phase,

the algorithm selects a grid of points, denoted by Λ, in the localization space $\mathcal{D}$. Some choices of the grid include square, hexagonal and random grid. At each point $\mathbf{v}$ in this grid Λ, also called training points, some observations are made that pertain to the location information. For RF-based localization, a signal feature is measured, mostly multiple times to compensate the transient effect of propagation environment such as fading and shadowing. The feature is chosen such that it contains adequate information about the location. The sufficient condition for the signal feature to distinguish training points is discussed in [2, 3]. It states simply that the probabilistic descriptions of the feature at different locations should be enough distinct measured in terms of Kullback-Leibler divergence. In this work, Received Signal Strength Indicator (RSSI) values of multiple anchors are used as the signal feature. The measurements are labeled with the location of the training points and therefore construct a database of labeled data.

B. Fingerprint Creation and Pre-Processing

Fingerprint creation is considered a kind of pre-processing for the data that is essentially heterogeneous. In data analysis applications, it is in general essential to pre-process the data for better presentation, compression and cleaning of the data. There are various guidelines for preparing the data for further analysis. Our data preparation follows the tiny data paradigm developed in [4]. In RF based fingerprinting solutions, given the volatile nature of wireless environments, measurements from different anchors differ in their numbers and scaling. At a location, one might not be able to get the same number of measurements from anchors due to their different signal strengths and incurred packet loss. Therefore the numbers of visible access points and the corresponding measurements differ from location to location. In this work, the training set consists of rows of different size corresponding to each training location. The row contains the position of the training point $\mathbf{v} \in \Lambda$ and the anchor ID and the corresponding measurements. In the light of what we discussed above, each row might have different length. It will be discussed later how to employ different data augmentation techniques to increase the size of training data and construct the rows of same length. At the end, one constructs the training dataset of form $(\mathbf{v}, \mathbf{X_v})$ from the raw observations $(\mathbf{v}, \mathbf{S_v})$ and $\mathbf{X_v}$ is called the fingerprint of the point $\mathbf{v}$.

For RF-based fingerprinting, a simple and popular procedure[5] for building a homogeneous database out of heterogeneous data is to compute the average value of RSSI values obtained from each access point. The averaged values are arranged into a vector and hence the rows of final database consists of the training location and the vector of averaged RSSI value. This vector is the fingerprint of the point $\mathbf{v} \in \Lambda$. There are other ways to create the fingerprint including, to name a few, RSSI quantiles or fitting Multivariate Gaussian distributions to RSSIs. In this work, the averaging method is used as the benchmark. In general, once the data is prepared for further analysis, the fingerprint of the point $\mathbf{v}$ is implicitly understood as the rows corresponding to $\mathbf{v}$ in the database.

C. Pattern Matching and Post-processing

Once the training dataset is built, a function should be learned mapping the new observations to positions in $\mathcal{D}$. In conventional fingerprinting algorithms, the function of pattern matching is to capture the similarity between fingerprints of training points and the fingerprint of test points. Namely, the goal is to find the most similar pairs of test point and training point in the fingerprint space and then use the location information of the training points to estimate the test point in the location space. One of the most well known algorithms for pattern matching employs Euclidean Distance (ED) to measure the similarity between fingerprints given as $d(\mathbf{X_u}, \mathbf{X_v}) = \|\mathbf{X_u} - \mathbf{X_v}\|_2$. With a new test point to be estimated, the Euclidean distance has to be calculated between the fingerprints $\mathbf{X_v}$ of all training points $\mathbf{v}$ in training grid Λ and the fingerprint of test point $\mathbf{X}$. The training point with smallest Euclidean distance from test point would be the best candidate:

$$\hat{\mathbf{u}} = \arg\min_{\mathbf{v}\in\Lambda} \quad d(\mathbf{X_v}, \mathbf{X}). \tag{1}$$

In general, the function d can be any kernel function. This process is completer with post-processing methods such as k-nearest neighbor (kNN) is the last step of traditional fingerprint algorithms. In the previous step, the k closest fingerprints in the training set are chosen and the final location is obtained by the linear combination of the k corresponding locations. The number k and the weights of final linear combination as parameters of learning algorithm are chosen during the matching process.

Indeed, this approach is nothing but an extension of kNN classifier. The problem of learning the localization function Φ is a supervised learning problem. Depending on the particular problem at hand, it can be a classification or regression problem. Regression-based localization function aims at giving the estimated location while classification-based localization function identifies the room or the area in which the target node is placed. In this paper, we consider both classification and regression-based approach and address the design challenges of learning algorithms in this context.

III. Support Vector Machine in Fingerprinting Algorithm

In this section, the localization is considered as a classification problem. The training grid Λ divides the localization space into different regions and the goal of the localization algorithm is to determine the region corresponding to a test point. SVM is used as the classification algorithm.

A. Support Vector Machine Algorithm

SVM is a binary classifier introduced by Vapnik and Chervonenkis in 1963. SVM aims at finding a linear classifier, i.e., a hyperplane which maximizes the margin between two classes. It has extensions to non-linear classifiers

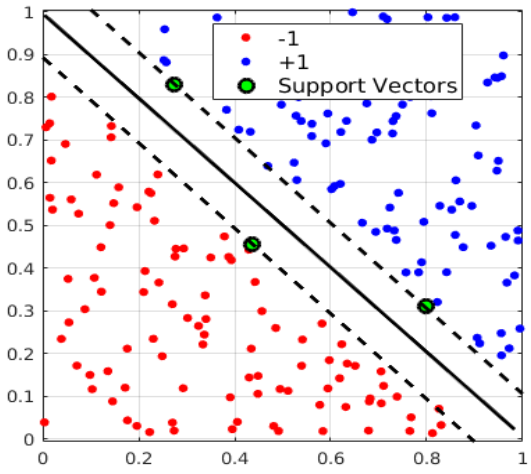

Fig. 1. Linear Support Vector Machine

and non-separable data too. First a training dataset, linearly separable, is given consisting of data points $\mathbf{x}_i \in \mathbb{R}^p$ with labels $y_i \in \{-1, +1\}$. The idea of SVM is to use a hyper-plane $f(\mathbf{x}) = \mathbf{a}^T\mathbf{x} + b$ to separate two classes so that each class lies on one side of the hyper-plane, i.e., $y_i(\mathbf{a}^T\mathbf{x}_i + b) \geq \gamma > 0$ for some $\gamma > 0$. The optimal hyper-plane would be the one which maximum margin between two classes. It can be seen that the following optimization problem provides a solution for $\mathbf{a}$ and b [6]:

$$\arg\min_{\mathbf{a},b} \frac{\|\mathbf{a}\|^2}{2} \quad \text{s.t.} \quad y_i(\mathbf{a}^T\mathbf{x}_i + b) - 1 \geq 0. \tag{2}$$

One can equally solve the dual problem by considering the Lagrangian, yielding the following problem:

$$\begin{aligned} \arg\max_{\boldsymbol{\lambda}} & \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i\lambda_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j \\ & \text{s.t.} \sum_{i=1}^{n} \lambda_i y_i = 0 \quad \text{and} \quad 0 \leq \lambda_i. \end{aligned} \tag{3}$$

Note that the dimension of search space for the dual problem scales with the size of training set but for the primal problem with the dimension of training points. Therefore, although both problems can be considered as quadratic optimization problem and can be easily solved by quadratic programming algorithms, the choice of which problem to solve depends on the number of training points and their dimension. After solving the dual problem, $\mathbf{a}$ can be obtained as $\mathbf{a} = \sum_{i=1}^{n} \lambda_i y_i \mathbf{x}_i$. The support vectors are those with $\lambda_i > 0$ and they solely determine $\mathbf{a}$ and b. For a support vector $\mathbf{x}_k$, b is obtained as $b = y_k - \mathbf{a}^T\mathbf{x}_k$. Th support vectors are shown in Fig.1.

In order to deal with non-separable data, a penalty term is added which tries to minimize the number of points inside the margin. If the margin violation of each point is denoted by ξ_i, the goal is to minimize the ℓ_0-norm of $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_n)^T$ for this purpose. To have a convex formulation ℓ_1-norm is used instead which is well known to provide sparsity. Therefore the following optimization problem is solved:

$$\begin{aligned} \arg\min_{\mathbf{a},b} & \frac{\|\mathbf{a}\|^2}{2} - C\sum_i \xi_i \\ & \text{s.t.} \quad y_i(\mathbf{a}^T\mathbf{x}_i + b)) \geq 1 - \xi_i \text{ and } \xi_i \geq 0. \end{aligned}$$

where C is a parameter which should be correctly chosen. The equivalent dual problem is given by:

$$\begin{aligned} \arg\max_{\boldsymbol{\lambda}} & \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i\lambda_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j \\ & \text{s.t} \sum_{i=1}^{n} \lambda_i y_i = 0 \text{ and } 0 \leq \lambda_i \leq C. \end{aligned} \tag{4}$$

Note that the search space has the dimension $p + n + 1$ for the primal problem while the dimension of the dual problem remains unchanged for both separable and non-separable case, equal to n, hence, making it more efficient to solve for non-separable case.

As another advantage of the dual problem, it can be easily extended to a non-linear SVM classifier which is achieved by using kernel trick. In this case, the inner product $\mathbf{x}_i^T\mathbf{x}_j$ as in (4) is replaced by a kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$. The kernel function represents an inner product of the transformations of these vectors in a feature space, usually higher dimension and possibly infinite dimensional. This transformation is only done implicitly and in many cases the transformation function is not explicitly known. Since the dual problem only depends on the inner product, it is sufficient to know the inner product in the feature space as a function of training points and kernel functions provide this information. By transformation into higher dimensional space, a linearly non-separable data in the ambient space might become linearly separable in the feature space. Some examples of kernel functions are polynomial kernel, Radial Basis Function (RBF) kernel and hyperbolic tangent kernel. In general, SVMs only require to tune few parameters namely the constant C in (4) and parameters of kernel function.

B. SVM in Fingerprinting Algorithm

In the context of fingerprinting algorithm, the fingerprint of each training point is a sequence of equal number of measurements from different anchors. Therefore for A anchors and M measurements per each, the fingerprint at each training point is of dimension $A \times M$. To apply SVM in fingerprinting algorithm, training points are chosen. The Voronoi diagram corresponding to the grid divides the location space into different regions where each region can be seen as a class and all points in one region are seen as one class. The aim is to classify the test point to one of those partitions using SVM. However if the problem contains only a single observation for a given class, SVM would be exactly same as Euclidean distance based localization. Therefore multiple observations are needed for each class.

This can be done using data augmentation which will be discussed later. However casting the localization problem as classification makes possible to utilize the so called imprecise measurements. In that case, it would be enough to know the region in which observations are collected, i.e., the label of measurements. Unlike training points on the grid, no precise location information is needed. Interestingly this improves the localization performance.

One needs particularly to solve a multi-class classification problem. Therefore multiple binary SVM classifiers have to be trained and the final decision is made according to *one vs. one* or *one vs. rest* strategy. For k classes, in one vs. one approach, $\binom{k}{2}$ binary classifiers are constructed and the class with highest number of decisions is chosen. In one vs. rest approach only k classifiers are trained and for each classifier one class is tested against all other classes put into a single one. Note that the kNN can also be used in SVM approach by choosing multiple top classes and finding the linear combination of corresponding training points location.

IV. Deep Learning for Fingerprinting Algorithm

Deep learning architectures emerged as the prime candidate for complex learning problems with excellent performance. In span of few years these architectures outperformed conventional machine learning algorithms in tasks such as pattern recognition. Neural networks consist of multiple units called neurons connected to each other where each neuron computes a function of its input value, mostly a non-linear function. The input to each neuron is the linear combination of the output of some other neurons and the goal is to learn the weights in each linear combination to be able to finally perform certain tasks. This process is called training phase. The training is done usually through a procedure called back propagation where the weights are adjusted iteratively to minimize the output error for a training set. Deep learning refers to an architecture in which the neurons are organized in many consecutive layers. The main challenge in training deep architectures is that the output error as a function of weights contains many local minima and saddle points, making it extremely difficult to find the global optimum. In this section, deep neural networks are used to perform regression task for indoor localization problem. The function Φ is approximated using the neural network mapping fingerprints to the estimated location of this point. The training set is therefore the pairs of training points and their fingerprints. Compared to kNN approach, this approach combines pattern matching and post-processing and provides the location in one shot. On the other hand, the received RSSI values are not averaged and directly used as the input to the algorithm. This might prevent the possible information loss in averaging. There have been some researches which already applied deep learning to indoor localization problems. In [7], the authors propose a fingerprint construction using neural networks based on measured Channel State Informations (CSIs). In [8], RSSI values are used to solve floor classification problem. In this work, we focus on design issues including the influence of different hyperparameters, avoiding overfitting and training algorithms. The hyperparameters in deep learning consists of number of layers, number of neurons in each layer, the choice of non-linearity parameters, learning rate, etc. The choice of hyperparameter is an important problem in deep learning and currently there is almost no unified theory for choosing those parameters. However there are many guidelines derived from vast experimental researches such as in [9]. For simplicity, not all the hyperparameters are discussed in this paper. More focus is put into those hyperparameters with seemingly most important effect.

1) Weight initialization: The boom of deep learning research starts from the idea of layer-wise pre-training the network weights [10]. The idea is to pre-train the weights of neural network in order to put them in a good start point in error space and then fine-tuning the whole network following forward, backward propagation update procedure. However it has been studied in latter research [11] that a straightforward initialization of weights are sufficient for network to converge to a good minimum. In this work, the weights are initialized according to Gaussian distribution, namely the weight matrix of i-th layer $\mathbf{W}^{(i)}$ is a random matrix with i.i.d. entries generated as Gaussian $\mathcal{N}(0, \sqrt{2/n^{(i)}})$ where $n^{(i)}$ is the number of neurons in i-th layer [11].

2) Gradient-based training: Once the weights are initialized, the weight are adjusted to minimize the error for the training set as function of weights $E(\mathbf{W})$. The training set is divided into so called *mini-batches* each one containing B training points. At each step, the error is minimized for a single mini-batch and weights are updated proportional to their contribution to the mini-batch error. The process is called mini-batch stochastic gradient descent, in particular stochastic gradient descent if B is equal to one and batch gradient descent for B equal to the whole training data-set. Pure stochastic gradient descent, $B = 1$, is rarely used in practical application for their slow convergence in small gradient direction and oscillation for large gradient direction. The weights are updated iteratively by repeating the training over each mini-batch for multiple times, called epochs. At iteration n, the weights are updated by $\Delta\mathbf{W}^{(n)}$ which is determined by

$$\Delta\mathbf{W}^{(n)} = \rho\Delta\mathbf{W}^{(n-1)} - \alpha\frac{\partial E(\mathbf{W}^{(n-1)})}{\partial\mathbf{W}^{(n-1)}}. \qquad (5)$$

ρ is the momentum parameter and α is called learning rate. ρ determines how much the last update should be preserved in the next iteration and α determines how fast one can move away from the previous iteration. The update is given by $\mathbf{W}^{(n)} = \mathbf{W}^{(n-1)} + \Delta\mathbf{W}^{(n)}$.

Adaptive learning rates enable different weights to have their own learning rates rather than a unified learning rate α. This can be done by accumulating the sum of the squared gradient for each weight and then divide the

unified learning rate by the sum element-wise. Different algorithms are designed to achieve these two approaches. Adagrad[12] accumulates the squared gradient constantly, but the learning rate will monotonically decrease. RMSprop instead uses a decay parameter to remove the influence of gradients from very early iterations to prevent the problem in Adagrad. Adam[13] combines the ideas of momentum and adaptive learning rate together. In this work Adam is chosen as our default optimization method.

3) Learning rate: This is the single most important hyperparameter in deep learning. It controls the step size of gradient descent. Too large learning rate can cause the update oscillating around minimum or even diverge. On the other hand too small learning rate will make the update very slow and possibly getting stuck in local minima. There are many guidelines for choosing the learning rate. In this work, the parameter is manually fine tuned by looking at the behavior of error after each iteration.

4) Activation function: For a long time, logistic sigmoid function $\frac{1}{1+e^{-x}}$ has been the default choice of activation function but it suffers from vanishing Gradient problem. It is known that Rectified Linear Unit (ReLU), defined as $f(x) = \max(0, x)$ does not suffer from saturation and converges faster than sigmoid to an acceptable minimum.

5) Number of hidden layers and neurons: In [9], it was suggested that higher number of hidden layers with adequate neurons provide significant expressive power. Large number of hidden layers provide more capacity to model more complicated functions with the drawback of difficult training process. For regression and classification task, and not representation learning, it is also recommended to use same size for all layers rather than decreasing or increasing size layers.

6) Regularization: Another central problem in machine learning is overfitting. It occurs when the learning algorithm gives a very good performance on the training data but it performs badly on the test data. This is because the learning algorithm models those features of particular training set which is non-essential to the task and therefore becomes sensitive to variation of those features. Regularization is an important approach to prevent overfitting. The idea is to use techniques during the training process to force partial but effective learning of essential features of the task in hand. One approach is based on using either ℓ_1 or ℓ_2-regularization. In this case, a penalty term $\lambda \sum_{i=1}^{L} \|\mathbf{W}_i\|_p^p$, $p = 1$ or 2, is added to the error function of the neural network to be minimized. The penalty term restricts the norm of weights to be small for $p = 2$ and promotes sparsity of the weights for $p = 1$. Another approach is called Dropout which prevents overfitting by randomly dropping some hidden units during weights update for each iteration [14]. In this way, the intermediate representations of the input is not dependent on only few neurons. Finally, early stopping is a very easy and effective way of regularization. The database is divided into training and validation set and the network is trained only using the training set. However the error is observed for the validation set and the training should stop when the error of validation set stops decreasing.

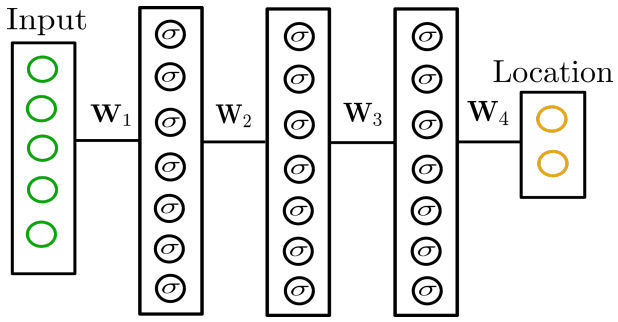

Fig. 2. Neural network configuration

A. Deep Learning in Fingerprinting Algorithm

TABLE I
NEURAL NETWORK CONFIGURATION

Raw RSS values as input
Fully Connected layer with 500 neurons
Dropout layer with 50% rate
Fully Connected layer with 500 neurons
Dropout layer with 50% rate
Fully Connected layer with 500 neurons
Dropout layer with 50% rate
Location coordinates (2 dimensional)

In this work, deep neural networks are trained to approximate the localization function Φ, Fig. 2. We propose to use a three fully connected hidden layers neural network with 500 neurons in each hidden layer. All hidden layers are equipped with the ReLU non-linearity, Table I. The output layer is a linear layer. For each layer we deploy a dropout layer with dropping rate of 50 percent. The weights are initialized by using random procedure suggested above. The neural network is trained using Adam algorithm with learning rate 0.001, momentum parameter 0.9 and mini-batch size 100. Moreover, ℓ_2 penalty is also used with the penalty parameter λ set to 0.03. The input is the raw RSSI values directly obtained from different Access Pointss (APs). The output is the estimated coordinates of the test point which is in our case in two-dimensional space.

V. Numerical Analysis

In this section, the previous learning algorithms are implemented to solve indoor localization problem. The implementation details are discussed and the final algorithms are evaluated. The implementations are done in Python using Tensorflow and Keras.

A. Experimental data

In order to compare different proposed algorithms, the UJIIndoorLoc dataset is used [15]. The dataset contains 19937 training samples and 1111 test samples. Each sample consists of 529 features where the first 520 features are RSSI values from 520 access points ranged from -104 dBm

TABLE II
DISTRIBUTION OF DATA

BuildingFloor ID	Training samples	Test samples
B0F0	1059	78
B0F1	1356	208
B0F2	1443	165
B0F3	1391	85
B1F0	1368	30
B1F1	1484	143
B1F2	1396	87
B1F3	948	47
B2F0	1942	24
B2F1	2162	111
B2F2	1577	54
B2F3	2709	40
B2F4	1102	39
Total	19937	1111

to 0 dBm. The positive value 100 is used to indicate when a signal was not detected. The features 521 to 529 correspond to latitude, longitude, floor, building ID, space ID, relative position, user ID, phone ID and time stamp. The data, Table II, is obtained from 3 buildings with 4, 4 and 5 floors respectively.

B. Experimental Result

In this paper, we only select data from floors of building 0. The undetected signals are denoted by -110 dBm instead of 100 which means very weak signals (-104 dBm is the smallest measured RSSI value in data set). The features are then scaled independently to have zero mean and variance of 1. The absolute positions are converted to relative positions by subtracting the smallest latitude and longitude in the data set. The room size (98.7m $\times$ 110.5m, 104.2m $\times$ 118.4m, 104.2m $\times$ 119.1m, 104.2m $\times$ 119.1m for four floors) can be obtained by looking at the difference between maximum and minimum of latitude and longitude. Moreover there are some access points that are undetected for all points in a certain floor. Those features are removed in order to speed up the training phase.

The traditional Euclidean distance and SVM algorithm are also tested here as comparison to Neural Network solution. k is chosen equal to 3 for kNN. SVM is implemented using RBF kernel $K(\mathbf{x}, \mathbf{y}) = \exp(-\gamma\|\mathbf{x} - \mathbf{y}\|^2)$ with parameter $\gamma = 1$ and C is chosen equal to 7. According to Table III, in all four floors for building one, neural networks gives better average mean error and less error variance.

C. Algorithm Optimization

Despite the good performance of fingerprinting algorithm in indoor localization environment, the data collection has always been a problem due to the long collection time and potential change of indoor environment. Thus we propose two methods that can alleviate this problem, namely data augmentation and model transfer.

1) Data Augmentation: Data augmentation is a common method in deep learning to reduce the effect of overfitting. The idea is to expand the existing dataset using only the available data so that the learning algorithm grasps better those features essential to the task. In case of image classification, common data augmentation approaches include translation, rotation and reflection of images in the dataset since the label of the image is unchanged by those transformations [16]. A similar thing can be done for fingerprints. For an observation of m RSSI values per n APs, a data matrix with size $m \times n$ is obtained for each training point. Since the order of measurements from each AP is irrelevant for localization, assuming constant power and steady environment, one can permute each column and a new data matrix of same size can be generated. This action can be done k times. The labels of these new data are the coordination of the corresponding training point. A sufficient number of permutation can give around 10% improvement in the mean error.

TABLE III
SUMMARY RESULTS FOR THREE ALGORITHMS

	Euclidean Distance	SVM	Neural Network
Floor 0 of Building 0			
Mean error [m]	10.06	8.48	7.62
Error variance	68.30	63.35	43.00
Min. error [m]	0.45	0.25	0.63
Max. error [m]	40.15	53.62	34.02
Floor 1 of Building 0			
	Euclidean Distance	SVM	Neural Network
Mean error [m]	9.93	8.81	8.08
Error variance	195.33	119.73	93.94
Min. error [m]	0.25	0.09	0.40
Max. error [m]	118.64	81.56	90.00
Floor 2 of Building 0			
	Euclidean Distance	SVM	Neural Network
Mean error [m]	9.50	9.42	7.42
Error variance	180.24	175.47	28.43
Min. error [m]	0.07	0.38	0.37
Max. error [m]	105.58	86.69	25.21
Floor 3 of Building 0			
	Euclidean Distance	SVM	Neural Network
Mean error [m]	9.40	7.72	7.27
Error variance	89.87	52.98	29.36
Min. error [m]	0.82	0.22	0.76
Max. error [m]	63.35	39.16	26.41

2) Model Transfer: Inspired by pre-trained models in image classification such as AlexNet[16], it is possible to fine tune pre-trained models by training on small amount of data. Applied to indoor localization application, this idea makes it possible to use an existing deep architecture for a different but similar building or for the same but altered building with far fewer measurements. In order to test the usage of pre-trained models, we take the whole data of Floor 0 and 10 percent of the data from Floor 1 of the building 0 from UJIIndoorLoc dataset. Fig.3 shows the training grids of two floors and it can be seen that two floors have similar structure. First, we train the neural network on the whole data of Floor 0 to get our pre-trained model and then we fine tune the model by only 10 percent of the data from Floor 1. Finally the new model is tested on the test samples from Floor 1.

Table IV shows the test result for three scenarios. first the neural network is trained directly by 10 percent data from Floor 1 and test with test samples from Floor 1; second,

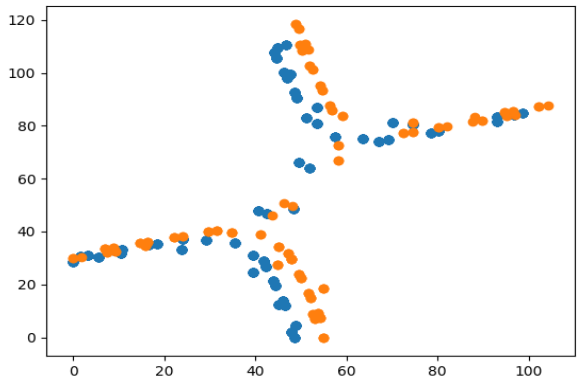

Fig. 3. Training grids of two floors

TABLE IV
SUMMARY RESULTS FOR MODEL TRANSFER

	W/o pre-training	Before fine-tuning	After fine-tuning
Mean error [m]	26.45	12.45	10.55
Error variance	163.86	53.81	112.37
Min. error [m]	1.56	0.46	0.15
Max. error [m]	62.65	47.70	81.20

the neural network is trained by data from Floor 0 and test with the test samples from Floor 1 and third the model is obtained from the second step and fine tuned by 10 percent data from Floor 1 and tested with the test samples from Floor 1. By looking at the first and third scenarios, it can be seen that even when there are not enough data, using pre-trained model can give reasonable result. By comparing second and third scenarios, it can be concluded that by fine-tuning with small amount of latest data, the neural network can be updated in order to fit the new propagation model.

VI. CONCLUSION

In this work, SVM and deep architectures are used to address issues in fingerprinting indoor localization problem. The state of art training methods in deep learning are used to further improve the performance. Moreover we proposed using data augmentation and model transfer to alleviate the problem in data collection which is essential in fingerprinting approaches. It was shown in the end that neural network performs better than the other two algorithms in the sense of mean error and error variance. Future work includes further variations in hyperparameters, e.g. deeper network structure, usage of convolutional neural networks and examining different gradient based methods.

REFERENCES

[1] Filip Lemic et al. “Experimental decomposition of the performance of fingerprinting-based localization algorithms”. In: *Indoor Positioning and Indoor Navigation (IPIN), 2014 International Conference on*. 2014, pp. 355–364.

[2] Arash Behboodi, Filip Lemic, and Adam Wolisz. “Hypothesis Testing Based Model for Fingerprinting Localization Algorithms”. In: *2017 IEEE 85th Vehicular Technology Conference (VTC-Spring’17)*. 2017.

[3] Arash Behboodi et al. “A Mathematical Model for Fingerprinting-based Localization Algorithms”. In: *arxiv:1610.07636* (2016).

[4] Hadley Wickham et al. “Tidy data”. In: *Journal of Statistical Software* 59.10 (2014), pp. 1–23.

[5] Ville Honkavirta et al. “A comparative survey of WLAN location fingerprinting methods”. In: *Positioning, Navigation and Communication, 2009. WPNC 2009. 6th Workshop on*. 2009, pp. 243–251.

[6] Ingo Steinwart and Andreas Christmann. *Support vector machines*. 1st ed. Information science and statistics. New York: Springer, 2008.

[7] Xuyu Wang et al. “CSI-based fingerprinting for indoor localization: A deep learning approach”. In: *IEEE Transactions on Vehicular Technology* 66.1 (2017), pp. 763–776.

[8] Michał Nowicki and Jan Wietrzykowski. “Low-effort place recognition with WiFi fingerprints using deep learning”. In: *arXiv:1611.02049* (2016).

[9] Yoshua Bengio. “Practical recommendations for gradient-based training of deep architectures”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.

[10] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. “A fast learning algorithm for deep belief nets”. In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[11] Kaiming He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[12] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive subgradient methods for online learning and stochastic optimization”. In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[13] Diederik Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv:1412.6980* (2014).

[14] Nitish Srivastava et al. “Dropout: A simple way to prevent neural networks from overfitting”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[15] Joaquín Torres-Sospedra et al. “Ujiindoorloc: A new multi-building and multi-floor database for wlan fingerprint-based indoor localization problems”. In: *Indoor Positioning and Indoor Navigation (IPIN), 2014 International Conference on*. 2014, pp. 261–270.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.