2018 IEEE 15th International Conference on Mobile Ad-hoc and Sensor Systems

Top-Down Indoor Localization with Wi-Fi Fingerprints using Deep Q-Network

Fei Dou **Jin Lu** **Zigeng Wang** **Xia Xiao** **Jinbo Bi** **Chun-Hsi Huang**
Department of Computer Science & Engineering
University of Connecticut
Storrs, CT 06269, USA
{fei.dou, jin.lu, zigeng.wang, xia.xiao, jinbo.bi, chunhsi.huang}@uconn.edu

***Abstract*—The location-based services for Internet of Things (IoTs) have attracted extensive research effort during the last decades. Wi-Fi fingerprinting with received signal strength indicator (RSSI) has been widely adopted in vast indoor localization systems due to its relatively low cost and the potency for high accuracy. However, the fluctuation of wireless signal resulting from environment uncertainties leads to considerable variations on RSSIs, which poses grand challenges to the fingerprint-based indoor localization regarding positioning accuracy. In this paper, we propose a top-down searching method using a deep reinforcement learning agent to tackle environment dynamics in indoor positioning with Wi-Fi fingerprints. Our model learns an action policy that is capable to localize 75% of the targets in an area of 25000m^2 within 0.55m.**

***Index Terms*—Indoor Localization, Wi-Fi Fingerprint, RSSI, Deep Reinforcement Learning, Deep Q-Network, Dynamic Environment**

I. Introduction

Applications of indoor location-based service (ILBS) in a wide range of living, commerce, production and public services have attracted much attention recently, which sharpen an urge for accurate and robust indoor positioning schemes. Compared with outdoor localization, it has been challenging as the GPS (Global Positioning System) signal, which serves as a standard solution in outdoor localization, cannot penetrate well in indoor environment. In the past decades, indoor localization solutions have been explored using Wi-Fi, Bluetooth, FM radio, radio-frequency identification (RFID), ultrasound or sound, light, magnetic field, *etc* [1]. Among all the techniques, Wi-Fi fingerprinting with RSSIs from different Wi-Fi Access Points (APs), referred as Reference Points (RPs), has been proven an promising approach due to its high accuracy, simplicity and deployment practicability [2].

Wi-Fi fingerprinting usually involves two phases: an off-line phase where RSSIs are collected from known positions to build a fingerprint database of the environment, as well as an on-line phase where the position is estimated by the current captured RSSIs with those in the database.

Many machine learning algorithms such as k-nearest neighbors (KNN), Naive Bayesian, support vector machine (SVM) and neural network (NN) have been applied to find the most probable location from the fingerprints. Some existing works that modeled the problem as a regression problem by predicting the coordinates of the intended position according to current RSSI values could be very sensitive to the environment dynamics. Others proposed classification-based solution by dividing the floor area into small grids with some certain sizes [3], [4]. However, higher localization accuracy requires smaller size of those grids, thus it must redefine the partition and requires both lots of human efforts and the floor plan as prior knowledge. Hence, it is not scalable since new models must be trained when different location resolutions are required.

Moreover, the fluctuation of wireless signal leads to considerable variations on RSSIs, and factors that have been observed affecting the RSSIs include but not limited to relative humidity level, people presence and movements, and open/closed doors [5] in a dynamic environment. This poses grand challenges to the positioning accuracy of fingerprint-based indoor localization with environment uncertainty.

In this paper, we attempt to shed a light on the questions above and propose a top-down approach to sequentially perform indoor localization in a dynamic environment by deep reinforcement learning. More specifically, the proposed model follows a hierarchical search strategy, which starts from the whole area or a prescribed area and then progressively scales down to the correct location of the target. The contributions of this paper are as follows:

- Our method is proposed to take the environment dynamics into concern. To fit this goal, we model the indoor localization problem as a Markov Decision Process (MDP), where a reward guided deep Q-network (DQN) learning agent interacts with the environment dynamically and selects sequential actions that progressively localize the target by transforming a bounding square window.
- We propose an accurate and efficient top-down searching approach for indoor localization. This approach has two main advantages: First, it doesn't require any prior knowledge of the floor plan in the indoor environment. Second, benefiting from the hierarchical structure, our method is capable to provide on-demand resolution of localization depending on the preference of computational cost.
- We leverage the advantage of DQN in handling online learning tasks since it has the ability that does not need retraining and memorizing all the data samples when

2155-6814/18/$31.00 ©2018 IEEE
DOI 10.1109/MASS.2018.00037

new data is received. Therefore, our localization model permits the entire system to provide sufficient accuracy even real-time positioning is required.

II. Related Work

Reinforcement learning [6] is a machine learning approach for optimal control and decision making processes, where an agent learns an optimal policy of actions over the set of states by interacting with the environment. It has a wide range of applications, such as robotics [7], games [8]–[10], image classification and object detection [11], [12], *etc*. And the best-known successes of reinforcement learning are playing Atari 2600 computer games [8], [9], and AlphaGo solving the challenge of Computer Go [10].

Mnih *et al.* [9] introduced DQN and kick-starts the revolution in deep reinforcement learning. It presented the first deep reinforcement learning model to successfully learn control policies directly at a human level from high dimensional sensory input which are only raw image pixels. [9] stabilized the training of value function approximation using experience replay and target network with convolutional neural networks (CNN), and it also designed a reinforcement learning approach with only the image pixels and the game score as inputs. AlphaGo [10] had made historical events by beating several human world champions in Go and became a milestone in artificial intelligence. This hybrid deep reinforcement system was built with techniques of reinforcement learning, deep convolutional neural network and Monte Carlo tree search (MCTS).

In the field of IoT, the work presented in [4] proposed a semi-supervised deep reinforcement learning model in support of smart IoT services. It leverages more abstract features from both labeled and unlabeled data by adopting variational autoencoders (VAE) [13], and then applies the deep reinforcement model on the extracted features to infer the classification of unlabeled data. The proposed model contains two deep networks that learn the best policies for taking optimal actions.

For indoor localization with Wi-Fi RSSIs in IoT, many machine learning approaches have been proposed. Yang et al. [14] proposed a KNN-based method by investigating the sensors integrated in modern mobile phones and user motions to construct the radio map of a floor plan. [15] adopted the model-based classification approach based on SVM. In [3], a four-layer deep neural network (DNN) generates a coarse positioning estimation by dividing indoor environment into hundreds of square grids.

[16] assessed some literature reviews and compared the performance of the most popular machine learning approaches to Wi-Fi fingerprinting, e.g. weighted k-nearest neighbors, Naive Bayes, neural network. It suggested that with only the Wi-Fi RSSI as the measurement metric, many complex algorithms may not perform as well as simpler ones. Despite the simplicity of weighted k-nearest neighbors method, it excelled in most fingerprinting reviews. So no wonder why KNN is the most widely applied benchmark algorithm in Wi-Fi fingerprinting based indoor localization problems.

III. Indoor Localization as a Dynamic Markov Decision Process

Markov Decision Process (MDP) [6] probabilistically models a goal-oriented agent that keeps interacting with the environment, and thereafter decides the action picked from the prescribed action space in sequences. In this section, we model our problem as a dynamical decision-making process, rather than a regression problem predicting the coordinates of the target, or a classification problem where classes represent the coarse region grids.

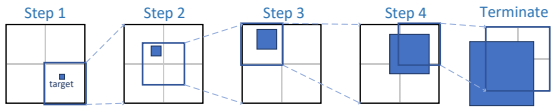


Fig. 1: Illustration of MDP

The process is shown in Fig. 1. In our case, the geometry and the RSSI signals on the single floor are defined as the environment, within which the agent shifts and transforms a bounding square window via a series of actions, and moves to the next state after taking a specific action under the current state. When the targeted object enters the environment and receives any RSSI signal, the agent is expected to localize it progressively by bounding it with a small enough window. In the localization process, the agent should determine at each step how to slide and reshape the window to efficiently localize the target within a number of steps as small as possible.

MDP is parameterized with several components: action space $\mathcal{A}$, the state space $\mathcal{S}$, and the corresponding reward function r. Details will be explained in the following parts.

A. Localization Actions

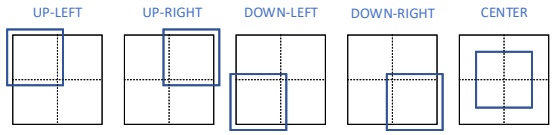


Fig. 2: Five actions in formulated MDP

To serve the purpose of efficient localization, our proposed action space $\mathcal{A}$ composes of finite actions applied to the square window. Fig. 2 presents the exact five actions denoted as "UP-LEFT", "UP-RIGHT", "DOWN-LEFT", "DOWN-RIGHT" and "CENTER". The window, subjected to the action, is uniquely characterized by a vector $\mathbf{o}_t$ at time step t ($\forall t > 0$) with its center coordinates and radius, written as $\mathbf{o}_t = [\mathbf{c}_t, rad_t]$, where $\mathbf{c}_t$ represents the coordinates of the current window center (x_t, y_t) and the radius rad_t denotes the half-length of the window's side.

Specifically, with respect to the action on $\mathbf{c}_t$, namely the determination of shift distance from the current window to the next, we elaborate the predefined rules on $\mathbf{c}_{t+1}$ as below:

- "UP-LEFT":
  $\mathbf{c}_{t+1} = (x_{t+1}, y_{t+1}) = (x_t - rad_t/2, y_t + rad_t/2)$
- "UP-RIGHT":
  $\mathbf{c}_{t+1} = (x_{t+1}, y_{t+1}) = (x_t + rad_t/2, y_t + rad_t/2)$
- "DOWN-LEFT":
  $\mathbf{c}_{t+1} = (x_{t+1}, y_{t+1}) = (x_t - rad_t/2, y_t - rad_t/2)$
- "DOWN-RIGHT":
  $(\mathbf{c}_{t+1} = x_{t+1}, y_{t+1}) = (x_t + rad_t/2, y_t - rad_t/2)$
- "CENTER":
  $\mathbf{c}_{t+1} = (x_{t+1}, y_{t+1}) = (x_t, y_t)$

One can observe the center either keeps unchanged or moves to an arbitrary center of four quarters in the previous window. Concretely, the transformations are obtained by adding or removing some scale of the radius to x or y coordinates depending on the desired effect. Besides, we propose the action on rad at the time $t+1$ following a scaling rate as:

$$rad_{t+1} = \alpha \times rad_t \tag{1}$$

where $\alpha \in (0, 1]$ is the shrinkage ratio on radius between two adjacent time steps.

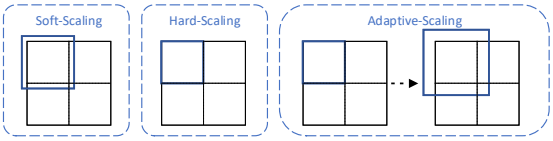


Fig. 3: Three variants of scaling strategy

The value of α needs to be carefully determined since it can considerably influence the complexity of searching space. Intuitively, increasing α is possible to guarantee a sufficient coverage with a compromise on efficiency, however, decreasing α is more efficient but risky to lose the object. We empirically explored it on three variants, shown in Fig. 3:

- **Soft-Scaling**: Fixed Rate and Overlapping.
  Rate α is a fixed number in $(0.5, 1]$, resulting in a overlapping condition of each down scaled window.
- **Hard-Scaling**: Fixed Rate and Non-Overlapping.
  Rate α is a fixed number 0.5, resulting in a non-overlapping condition of each down scaled window.
- **Adaptive-Scaling**: Non-Fixed Rate.
  The starting rate α_0 at the first step is set to be 0.5 in order to make it faster to focus on the expected region of the whole area, as well as not to lose the object. The rate will be increased with each step and infinitely close to the ending rate α_{end}, a number in $(0.5, 1]$ to perform a delicate and precise localization in final steps:

$$\begin{cases} \alpha_0 = 0.5 \\ \alpha_{t+1} = e^{-\lambda} \times \alpha_t + (1 - e^{-\lambda}) \times \alpha_{end} \end{cases} \tag{2}$$

  where λ is a parameter in $(0, 1)$ to control the speed of the rate augmentation.

In all three scaling strategies, α acts as a role to trade off between learning speed and localization accuracy, which needs to be explored further.

B. State

The state $\mathcal{S}$ in our formulated MDP is composed to describe the information of the current step. The representation is a tuple $s = (\mathbf{RSSI}, \mathbf{o}, \mathbf{h})$, defined as follows:

- a vector $\mathbf{RSSI}$ of all RSSI values.
- a vector $\mathbf{o}$ with the center coordinates and radius, written as $\mathbf{o} = [\mathbf{c}, rad]$, where $\mathbf{c}$ represents the coordinates of the current center (x, y) and radius rad denotes half the length of the square window's side.
- a vector $\mathbf{h}$, recording the history of taken actions in each searching round.

The history vector $\mathbf{h}$ captures all the actions that the agent performs during each searching round for detecting a target. We encode $\mathbf{h}$ as a one-hot vector, and each action in $\mathbf{h}$ is represented by a 5-dimensional binary vector where all the values within are zeros, except the one corresponding to the taken action, set to be 1.

The history vector encodes n past actions, leading to $\mathbf{h} \in \mathbb{R}^{5n}$. Here n depends on the largest number of steps to localize the target in the indoor environment. Although the history vector is a relatively low-dimensioned vector compared with the environment information vector $\mathbf{RSSI}$ that contains a large number of RSSI values, it is enough to inform what happened in the past and stabilize searching trajectories.

C. Reward Function

The reward function r reveals the improvement that the agent achieves to localize an object after choosing a specific action. The agent gets a positive reward when the action pushes the region window closer to the target, while a negative reward is gained when the action makes the window further away from the target. The improvement in our model is measured using the Intersection-of-Window (IoW) between the target square window and the predicted window given by a particular action. The reward function can thereafter be attained by the calculation of the improvement from one state to it's next state.

Let w denote the current window, and w^g is the ground truth square window of the target. Then the IoW between w and w^g is a number in $[0, 1]$ and defined as

$$IoW(w, w^g) = area(w \cap w^g)/area(w) \tag{3}$$

where $area$ denotes the area of a window.

In our top-down searching scheme, the region window scales down to the target. At the step t, the agent gains a positive reward if IoW of the next state s_{t+1} is larger than that of the current state s_t, meaning that the agent chooses a "correct" action to get closer to the target. Namely, the correct action keeps the target inside the window as well as having the size of the window smaller. A large positive reward will be assigned to the agent and terminate searching if it successfully localizes the target in a proper way, when IoW of the current state exceeds a threshold δ. Otherwise, when the agent chooses a "fatal" action, leading the window further away from the target, it terminates the searching process and receives a large negative penalty.

When the agent chooses the action a_t, causing the transfer from state s_t to its next state s_{t+1}, the reward function $r_{a_t}(s_t, s_{t+1})$ is defined as:

$$r_{a_t}(s_t, s_{t+1}) = \begin{cases} +\eta & \text{if } IoW(w^{s_{t+1}}, w^g) \in [\delta, 1] \\ +\tau & \text{if } IoW(w^{s_{t+1}}, w^g) \\ & \in (IoW(w^{s_t}, w^g), \delta) \\ -\eta & \text{otherwise} \end{cases} \quad (4)$$

In Equation (4), the stop rewards η take the absolute value of 3.0, so the agent receives a $+3.0$ reward when it successfully localizes the target and gets feedback of a -3.0 penalty when a "fatal" action is made. The intermediate transformation reward τ is set to be 1 as the feedback to a correct action when the window gets closer to the target. The threshold value δ is set to be 0.5, indicating the minimum IoW value allowed to consider a successful detection in the procedure of localization in our proposed model.

IV. Localization with Deep Q-Network

With the components of a MDP formulated above, the goal of the agent is to find a series of windows to zoom into the region of the target by selecting multiple actions. Fig. 4 shows the framework of our proposed top-down model that uses deep Q-network to perform localization for an indoor object using RSSI values.

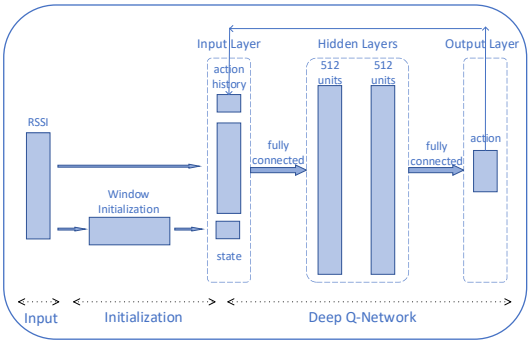


Fig. 4: Deep Q-Network for Indoor Localization

A. Window Initialization

There are two approaches to initialize the square window $\mathbf{o}_0 = [\mathbf{c}_0, rad_0]$. For the first approach, denoted as General Initialization, assume all data samples are horizontally bounded by maximum longitude lon_{max} and minimum longitude lon_{min}, and vertically bounded by maximum latitude lat_{max} and minimum latitude lat_{min}. We set $\mathbf{c}_0$ as the center of the bounded rectangular, and set rad_0 large enough to guarantee the full coverage of our interested area. Specifically, we define the initial square window as follows:

$$\begin{cases} \mathbf{c}_0 = (x_0, y_0) = (\frac{lon_{max}+lon_{min}}{2}, \frac{lat_{max}+lat_{min}}{2}) \\ rad_0 = \frac{max(lon_{max}-lon_{min}, lat_{max}-lat_{min})}{2} + rad_{gt} \end{cases} \quad (5)$$

where rad_{gt} denotes the radius setting of the target window, which defines how small of the target window we'd like to have and also indicates the localization resolution to be achieved.

Another approach to get the initial window is applying some machine learning algorithms to estimate the approximate location of the target according to the corresponding RSSI values, such as KNN or other algorithms. And then select a comparatively small radius to give it a warm start to allow the initial window to fully cover the target window. We will discuss these two initialization approaches in our experiment section.

B. Deep Q-Network for Localization

1) Model Overview: In a deep Q-network approach [9], we consider tasks in which an agent interacts with an environment $\mathcal{E}$ by a sequence of actions, observations and rewards. At each time-step t, the agent observes the current state s_t, selects an action a_t from the action space $\mathcal{A}$, and receives a reward r_t representing the improvement, and then goes to the next state s_{t+1}.

The goal of the agent is to interact with the environment by selecting actions and learn a policy π that maximizes the total future rewards. In [9], the standard assumption is that future rewards are discounted by a factor γ for each step. Define the future discounted *return* at time t as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where T is the step at which the searching round terminates. The optimal action-value function $Q^*(s, a)$ w.r.t s and a is defined as the maximum expected return achieved, after seeing s and then taking action a:

$$Q^*(s, a) = \max_{\pi} \mathbb{E}[R_t | s_t = s, a_t = a, \pi] \quad (6)$$

where π is a policy mapping distributions over actions $\pi = P(a|s)$.

$Q^*(s, a)$ obeys the Bellman Equation [6], which is based on the following intuition: if the optimal value $Q^*(s_{t+1}, a_{t+1})$ of a state s_{t+1} at the next time-step was known for all possible actions a_{t+1}, then the optimal strategy is to select the action a_{t+1} that maximizing the expected value of $r_t + \gamma Q(s_{t+1}, a_{t+1})$:

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim \mathcal{E}}[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) | s_t, a_t] \quad (7)$$

Reinforcement learning algorithm needs to estimate the action-value function by using the Bellman Equation as an iterative update, $Q_{i+1}(s_t, a_t) = \mathbb{E}[r_t + \gamma \max_{a_{t+1}} Q_i(s_{t+1}, a_{t+1}) | s_t, a_t]$, where i denotes the i^{th} iteration. Such value iteration algorithms converge to the optimal action-value function, $Q_i \rightarrow Q^*$ as $i \rightarrow \infty$. In practice, this basic approach is impractical, because the action-value function is estimated separately for each state, without any generalization. Instead, it is common to use a function approximator to estimate the action-value function, $Q(s, a; \theta) \approx Q^*(s, a)$. We use neural network function approximator with weights θ, referred as a Q-network, to estimate the optimal action-value function. This Q-network can be trained by adjusting the parameters θ_i at iteration i to reduce the mean-squared error in the Bellman Equation, where the optimal target values, $r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$,

Algorithm 1: Deep Q-Network for Indoor Localization

Data: A dataset containing RSSI values and labeled coordinates $\mathcal{D} : \{\mathbf{RSSI}_l, (x_l, y_l)\}$
Input: *environment parameters: g,* rad_{gt}*,* α*,* δ*; agent parameters:* γ*,* ϵ*,* $\mathcal{M}$

Randomly initialize DQN parameters θ;
for $iteration \leftarrow 0, \ldots, N$ **do**
  for *each data sample* d_i *in* $\mathcal{D}$ **do**
    Get initial coordinates (x_0^i, y_0^i) and initial radius rad_0 ;
    Initialize $\mathbf{h}^i$;
    Initialize $s_0 = (\mathbf{RSSI}^i, (x_0^i, y_0^i), rad_0, \mathbf{h}^i)$;
    for $t \leftarrow 0, \ldots, T$ **do**
      Select a random action a_t with probability ϵ, otherwise select $a_t = max_a Q^*(s_t, a_t; \theta)$;
      Execute action a_t as to get a reward r_t, new center (x_{t+1}^i, y_{t+1}^i), new radius rad_{t+1}^i and transform from current state s_t to its next state $s_{t+1} : (\mathbf{RSSI}^i, (x_{t+1}^i, y_{t+1}^i), rad_{t+1}^i, \mathbf{h}^i)$;
      Update $\mathbf{h}^i$ with a_t;
      Store transition (s_t, a_t, r_t, s_{t+1}) in replay memory $\mathcal{M}$;
      Sample random mini batch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{M}$;
      Set $y_j = r_j + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta)$;
      Calculate gradient descent according to Equation 10 and update θ by Adam [17] and Dropout [18].

are substituted with approximate target values $y_{i,t} = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_i^-)$, using previous network parameters θ_i^-. The Q-learning update in iteration i follows the below loss function:

$$L_{i,t}(\theta_i) = \mathbb{E}_{s_t, a_t \sim \rho(.)}[(y_{i,t} - Q(s_t, a_t; \theta_i))^2] \tag{8}$$

where

$$y_{i,t} = \mathbb{E}_{s_{t+1}}[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_i^- | s_t, a_t] \tag{9}$$

is the target for iteration i and $\rho(s, a)$ is a probability distribution over states s and actions a which are referred to as the *behavior distribution*. At each stage of optimization, the parameters from the previous iteration θ_i^- are held fixed when optimizing the i^{th} loss function $L_i(\theta_i)$. Differentiating the loss function with respect to the weights we arrive at the following gradient,

$$\begin{aligned}\nabla_{\theta_i} L_{i,t}(\theta_i) =& \mathbb{E}_{s_t, a_t, s_{t+1}}[(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_i^-) \\ & - Q(s_t, a_t; \theta_i)) \nabla_{\theta_i} Q(s_t, a_t; \theta_i)]\end{aligned} \tag{10}$$

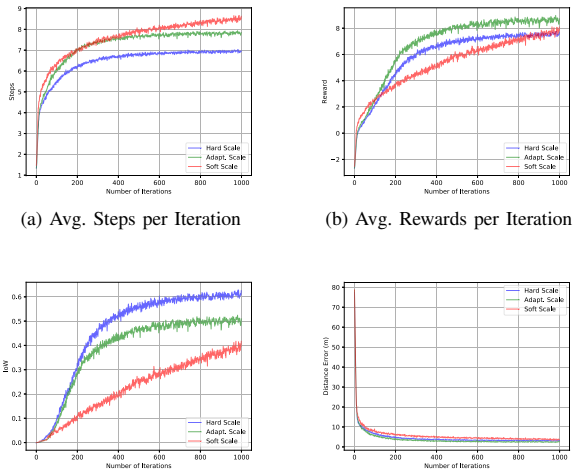
2) Components for Deep Q-Networks: Our algorithm framework for solving the DQN model is presented in Algorithm 1. To be more self-contained, several involved techniques are detailed as below.

- **Discounted Factor** To have a better performance in the long-run, not only the most immediate rewards but the future ones will also be taken into account. We use the discounted reward from Bellman Equation with a value of $\gamma = 0.1$. We set the gamma low since we are more interested in the current rewards, but still give a balance between the immediate rewards and future ones.
- **Explore-Exploitation** The policy used during training is ϵ-greedy [6], which gradually shifts from exploration to exploitation according to the value of ϵ. For exploration, the agent selects random actions and collects multiple experiences, while for exploitation, the agent selects greedy actions according to already learned policy, and then learns from its own successes and mistakes. In our settings, the ϵ-greedy policy starts with $\epsilon = 1$ which means a random choice of action, and decreases to $\epsilon = 0$ with $\epsilon_{i+1} = 0.995 \times \epsilon_i$ at each iteration.
- **Experience Replay** It is a technique [8], [19] where we store the agent's experiences at each time-step, $m_t = (s_t, a_t, r_t, s_{t+1})$ in a Experience Replay Memory $\mathcal{M} = m_1, m_2, \ldots, m_N$. During each training stage, we sample the Q-learning updates using mini bathes from samples of the stored experiences, $m \sim \mathcal{M}$, drawn randomly/weighted from the pool of the memory. In our settings, we use an experience replay of 2000 experiences and a batch size of 100.
- **History Vector** As we discussed in Section III-B, we capture all the actions for each data sample during each iteration in the search for the target. The total number of steps for the agent to find the target in each searching round depends on the initial window's size, the scaling strategy, and the radius of the target window. However, using arbitrary length as inputs to the neural network can be difficult. In our settings, we fix the length of the history actions representation, recording at most recent 10 actions for each target during each iteration, thus, $\mathbf{h} \in \mathbb{R}^{50}$. If the agent stops at a specific step $t < 10$, then the rest of the history vector will be filled with 0s.

V. Experiments and Evaluations

A. Data Description

The dataset used to verify our proposed model is the UJIIndoor Loc dataset [20], which was collected in real-world



(a) Avg. Steps per Iteration (b) Avg. Rewards per Iteration

(c) Avg. IoW per Iteration (d) Avg. Dist. Error per Iteration

Fig. 5: Training performance on different scaling strategies under $rad_{gt} = 0.5m$ (Hard Scaling with $\alpha = 0.5$, Adaptive Scaling with $\lambda = 0.2$, starting rate $\alpha = 0.5$, ending rate $\alpha = 0.6$, and Soft Scaling with $\alpha = 0.6$

including 3 buildings with 4 or 5 floors by more than 20 users using 25 different models of mobile devices within several months. The dataset covers a surface of $108703m^2$ in Universitat Jaume I and consists of 19937 training/reference records and 1111 validation/test records. The number of different APs appearing in the database is 520.

Since our algorithm is proposed to perform indoor localization in a 2D area and given that the UJI dataset describes a multi-building multi-floor environment, we select the data in Building 1 Floor 1 (B1F1) from the dataset, covering an area of approximate $25000m^2$ ($150m \times 160m$) and randomly split it into the training set and the test set with a ratio of $0.8 : 0.2$.

Considering the body of a human being, we choose the target square window with the radius $rad_{gt} = 0.5m$, which should be small enough to indicate the position of a person in an indoor environment.

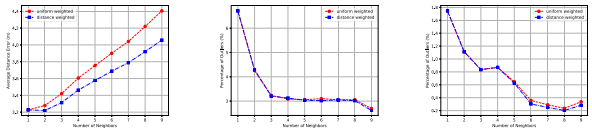
To simulate the environment dynamics, we inject noise into the input of the DQN in every decision-making step. [5] analyzed quantitative effects of those dynamic environmental factors like people, doors, humidity, and the measurement results demonstrate the average vibration on RSSI are approximately 8 dBm, 9 dBm and 0.8 dBm respectively. In our model, we generate the centered Gaussian noise $\mathcal{N}(0, \sigma^2)$ with the standard deviation $\sigma = 10$ to analog the approximate 10 dBm variations of RSSIs caused by environment uncertainty.

B. Training Evaluation

We train our DQN agent in an online fashion by selecting the data samples one by one for $N = 1000$ iterations and evaluate the average total steps, the average total rewards, the average IoW and the average distance error at the end of each iteration. Next we present the configuration for the training procedure.

1) On Different Scaling Strategy: In Fig. 5, we explore the training performance of different scaling strategies on our proposed DQN model. It illustrates that Hard Scaling needs the least steps to find the target on average, compared with Adaptive Scaling and Soft Scaling. Fig. 5b shows rewards are accumulated as the training iterations incenses, suggesting DQN learns the pattern from the localization samples gradually. Fig. 5c shows it takes about 300 iterations for the agent to achieve a value above 0.5 of average IoW under Hard Scaling, while the other two strategies show lower training efficiency, where Adaptive Scaling requires 800 iterations to achieve the same performance, and agent using Soft Scaling strategy shows difficulty to be trained well within 1000 iterations. The performance of an agent using Soft Scaling strategy illustrated in Fig. 5a, Fig. 5b and Fig. 5c all indicate that the agent still needs more time to be successfully trained after 1000 iterations. The training error for distance, measured by the distance between the center of the current window and the target window, shown in Fig. 5d doesn't imply remarkable difference among those three scaling strategies, while the Adaptive Scaling strategy outperforms slightly than the others.

2) On Different Initialization: [16] illustrated that KNN algorithms serve as the benchmark methods on indoor positioning because of its simplicity and accuracy. Thus, we consider the initialization with KNN-based algorithm denoted as KNN Initialization, to compare with our General Initialization approach.

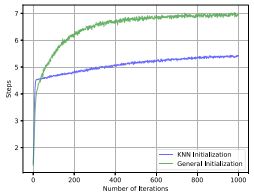


(a) Avg. Dist. Error (b) Error $>$ 20m (c) Error $>$ 30m

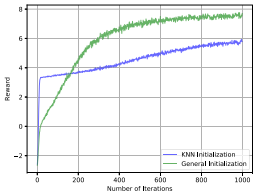
Fig. 6: Performance on different KNN algorithms & percentage of outliers with different distance errors

We evaluate KNN Intialization with two variants of KNN algorithms: the first is vanilla KNN where the k neighbors contribute equally to the estimation, while the second is weighted-KNN, which computes the inverse of RSSI distance as the weight of each neighbour's contribution, so as to emphasize the importance of the closer neighbour during the prediction. Fig. 6a draws the plot of the average distance error vesus the number of neighbors, while Fig. 6b and Fig. 6c illustrates the percentage of predicted examples with high error (outliers), from our observation, taking the balance between estimation accuracy and percentage of outliers into the concern, we choose the weighted-KNN with 5 neighbors as our window initializer and initialize the square window with the radius $rad_0 = 30m$.

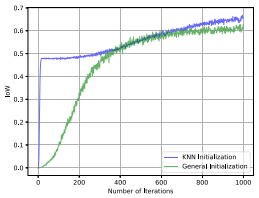
Fig. 7 shows the training performance of our proposed model under different initialization. As expected, the num-



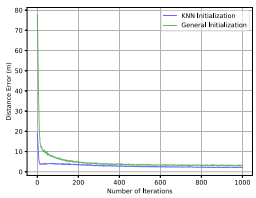
(a) Avg. Steps per Iteration



(b) Avg. Rewards per Iteration



(c) Avg. IoW per Iteration



(d) Avg. Dist. Error per Iteration

Fig. 7: Training performance on different initialization approaches under $rad_{gt} = 0.5m$ and hard scaling strategy)

ber of steps increases when the agent is trained with more iterations and will become stable thereafter. Our model with General Initialization needs 2 more steps to locate the target than KNN initialization, shown in Fig. 7a, resulting in more necessary amount of rewards observed in Fig. 7b.

From Fig. 7c, we can see when the number of iterations reach approximately at 350, the agent is almost well-trained to localize the target successfully since we set the IoW threshold $\delta = 0.5$. Further, one can inform from the same figure that both the two initialization schemes need about 350 iterations to achieve a value above 0.5 for average IoW, despite that KNN Initialization speeds up the training process at the beginning.

In Fig. 7d, the distance error could be dramatically decreased to about 4 meters within 15 iterations training when using an KNN Initialization. However, it is necessary to keep training the agent since we need both a small distance error as well as a small region window bounding the target area by scaling down with more time-steps.

C. Test Evaluation

We carried out our experiments on the test dataset using the trained models and computed Cumulative Distribution Function (CDF) with respect to the distance error under various configurations of the model.

1) Scaling Strategy Selection: We evaluate the performance of our model under different scaling strategies to explore the optimized setting for our model.

We can tell from Fig. 8 that within 90% percentile of CDF, Hard Scaling strategy always performs the best and Adaptive Scaling strategy performs slightly worse. For Soft Scaling, when the scaling rate α is smaller, i.e., $\alpha = 0.6$, our method shows better performance than the performance when $\alpha = 0.7$. The reason could be that the agent needs more steps

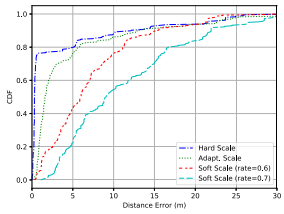


Fig. 8: CDF of distance error on different scaling strategies under $rad_{gt} = 0.5m$ (Hard Scaling with $\alpha = 0.5$, Adaptive Scaling with $\lambda = 0.2$, starting rate $\alpha = 0.5$, ending rate $\alpha = 0.6$, and Soft Scaling with $\alpha = 0.6$ and $\alpha = 0.7$)

to reach the satisfactory region with a larger α, which results from a larger searching space, making it difficult to train an efficient agent. The results of Adaptive Scaling justifies the above reason since the number of steps needed is between the other two strategies. Overall, we select Hard Scaling strategy to achieve a better performance in our model in the following sections.

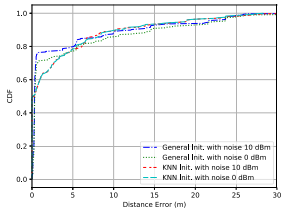


Fig. 9: CDF of distance error on different initialization with different environment settings, under $rad_{gt} = 0.5m$ and Hard Scaling strategy

2) Initialization Selection: We train our model with Hard Scaling strategy and $rad_{gt} = 0.5m$ to compare the performance of the proposed model using different initialization strategies: General Initialization and KNN Initialization. Fig. 9 shows the performance under dynamic environment and static environment. It illustrates that General Initialization performs much better than KNN Initialization when conducting localization task in a dynamic environment, which verifies the sole DQN has enough capacity for our localization tasks with both types of environments. Thus, in the later section, we test our model using a General Initialization.

3) Performance Comparison with Other Learning Algorithms: From the discussion above, we select the trained model with Hard Scaling strategy, using General Initialization, and $rad_{gt} = 0.5m$ and $0.8m$ in a dynamic environment, where the environment noise is Gaussian with $\sigma = 10$ applied in every step of searching.

We compare our proposed model with uniform weighted KNN (U-KNN), distance weighted KNN (D-KNN), Random Forest, Lasso Regression as well as Ridge Regression in Fig. 10, and the details are shown in TABLE I and II. It manifests

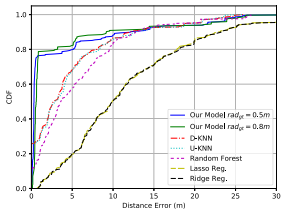


Fig. 10: Performance comparison with different algorithms

that among classic machine learning algorithms, KNN-based method excelled the others, as illustrated in [16]. But our proposed model outperforms all these evaluated algorithms significantly, improving the performance by at least 91% regarding the 75% percentile localization over the benchmark KNN-based method. Specifically, our method is able to achieve the performance of localizing 30%, 50% and 75% of the targets within $0.25m$, $0.31m$ and $0.55m$ respectively, while the two KNN-based models could only achieve a 75% localization percentage within $6.2m$.

Percentile	30%	50%	75%
DQN($0.5m$)	0.25507976	0.31761706	0.5529171
DQN($0.8m$)	0.42925235	0.60638235	0.81489037
D-KNN	1.03801639	2.75196774	6.19801714
U-KNN	1.11440207	2.94827254	6.02964302
RandomF	1.80340348	3.59024213	8.02614405
Lasso	6.50315978	9.90122459	16.10549175
Ridge	6.55371385	9.91698091	16.05958819

TABLE I: Comparison of accuracy under different percentiles

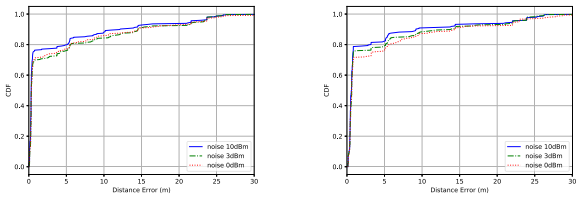
We also conduct an experiment with a larger rad_{gt} set as $0.8m$, indicating a coarser localization accuracy compared with the case of $rad_{gt} = 0.5m$. A very interesting observation can be found that the model trained with $rad_{gt} = 0.5m$ could localize 75% of the targets within $0.55m$, while the one trained with $rad_{gt} = 0.8m$ achieves the same percentage within $0.81m$. It implies that the prediction error of our proposed model can be reduced to less than 1-meter. This could be a result of the selection of the target window size w.r.t. rad_{gt}, since smaller size leads to higher accuracy. During the training phase, our model learns an optimal policy of actions that intends to find a small enough bounding window, satisfying the condition that IoW is greater than the threshold δ.

Moreover, compared with the KNN-based benchmark method, our proposed model is more time-efficient during the localization phase. The KNN model always requires massively calculating the distance between each pairs of samples in the dataset to find the nearest neighbors, while our model only needs much shorter time (apprx. tens of milliseconds) to even localize the target in a few window-scaling steps once the agent is learned successfully, and could be easily applied in real time fashion.

4) Robustness Analysis: Next, we evaluate the performance of our proposed model under different dynamic environments.

Dist. Err(m)	<0.5	<0.6	<0.8	<1	<6
DQN($0.5m$)	71.71%	75.08%	76.43%	76.43%	84.85%
DQN($0.8m$)	45.45%	49.83%	74.41%	78.79%	87.54%
D-KNN	26.93%	27.61%	27.61%	28.95%	73.74%
U-KNN	26.26%	27.27%	27.27%	28.28%	74.75%
RandomF	13.13%	15.15%	16.49%	19.19%	62.96%
Lasso	0%	0%	0.67%	1.35%	23.23%
Ridge	0%	0%	0%	1.35%	23.23%

TABLE II: Comparison of percentiles under different accuracy



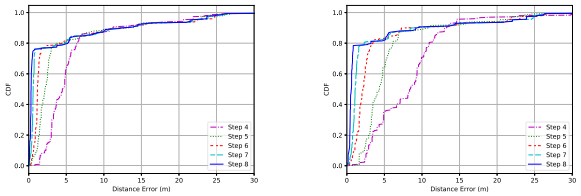
(a) $rad_{gt} = 0.5m$ (b) $rad_{gt} = 0.8m$

Fig. 11: CDF of distance error under different dynamics with Gaussian Noise ($\sigma = 0, 3, 10$) inserted into the input of the deep-Q network for every step during each searching round

We add Gaussian Noise to the RSSI values in every decision-making step of the MDP to simulate the environment dynamics. Fig. 11 shows that our proposed model even performs better in a dynamic environment, especially for the approximately real-world dynamic with a Gaussian noise where $\sigma = 10$, according to the analysis by [5] that the average vibration on RSSI in real world due to the effects of those dynamic environmental factors like people and doors is approximately 10 dBm.

This can be explained by the adaptive ability of DQN methodology to the dynamical environment. A very recent study [21] also confirms that noise drives exploration in many methods for reinforcement learning and enforces the model to be less-sensitive to the small variance on input. Our result also suggests that our model should be capable to track the target when it is a moving object.

5) On-demand Resolution Analysis: Our top-down approach for searching the target in the environment has two main advantages. First, our model can be readily applied to various environment since it doesn't require either prior knowledge of the floor plan, or predefined grids partitioning the whole area. Second, our model is capable of performing localization with on-demand resolution, determined by how many the searching steps are adequate to scale down the window according to one's needs: the proposed method can save time and computational costs by terminating at the earlier searching step; it also can position the target with a smaller window by searching deeper on the preference of localization resolution. Therefore, The hierarchical structure enables to get rid of tedious re-training process, which is compulsory for other existing methods, when calling for different localization resolutions.



(a) $rad_{gt} = 0.5m$ (b) $rad_{gt} = 0.8m$

Fig. 12: CDF of distance error on different steps under General Initialization, $rad_{gt} = 0.5m$ and $0.8m$, Hard Scaling, $\sigma = 10$

Fig. 12 shows the prediction results of model output at different steps. Step 8 is our last step, representing a resolution of $0.5m$ (equivalent to drawing small grids on a floor area with a side of $1m$) in Fig 12a, and step 7, 6, 5, 4 indicates localization resolution of $1m$, $2m$, $4m$, $8m$ respectively. It shows that our model could provide different localization resolution on each steps depending on one's preference. It can be observed that when the number of steps is equal or larger than 6, the performance doesn't dramatically degenerate as the number of steps decreases, implying our method can well approximate the location of the target even in a shallower search. Fig. 12b also illustrates lower accuracy in general when the number of steps is less than 6, which is in our expectation since the radius rad is so large that the window can only provide a relatively coarse region.

VI. Conclusion

In this paper, we develop a deep reinforcement learning scheme for Wi-Fi fingerprinting based indoor localization, which could handle both the variation of RSSIs due to environment dynamics, and online learning that is required for real-time positioning. It takes advantage of a top-down searching strategy to provide on-demand resolution of localization depending on the preference of computational cost; benefiting from the hierarchical structure, our proposed method also doesn't require the prior knowledge of floor plan. Various experimental results demonstrate that our method achieves state-of-the-art performance on localization with Wi-Fi signals. Although the evaluation is based on a specific dataset, it is strong enough to suggest that using a top-down DQN approach will likely lead to a higher accuracy and better scalability in practice. Based on our experiments, we foresee that the decision-making process interacting with the environment has a potential to be transferred to other application scenarios such as target tracking.

References

[1] Suining He and S-H Gary Chan. Wi-fi fingerprint-based indoor positioning: Recent advances and comparisons. *IEEE Communications Surveys & Tutorials*, 18(1):466–490, 2016.

[2] Simon Yiu, Marzieh Dashti, Holger Claussen, and Fernando Perez-Cruz. Wireless rssi fingerprinting localization. *Signal Processing*, 131:235–244, 2017.

[3] Wei Zhang, Kan Liu, Weidong Zhang, Youmei Zhang, and Jason Gu. Deep neural networks for wireless localization in indoor and outdoor environments. *Neurocomputing*, 194:279–287, 2016.

[4] Mehdi Mohammadi, Ala Al-Fuqaha, Mohsen Guizani, and Jun-Seok Oh. Semi-supervised deep reinforcement learning in support of iot and smart city services. *IEEE Internet of Things Journal*, 2017.

[5] Yi-Chao Chen, Ji-Rung Chiang, Hao-hua Chu, Polly Huang, and Arvin Wen Tsui. Sensor-assisted wi-fi indoor location system for adapting to environmental dynamics. In *Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 118–125. ACM, 2005.

[6] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[7] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2786–2793. IEEE, 2017.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *In Deep Learning, Neural Information Processing Systems Workshop*, 2013.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[10] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[11] Miriam Bellver, Xavier Giró-i Nieto, Ferran Marqués, and Jordi Torres. Hierarchical object detection with deep reinforcement learning. *Deep Reinforcement Learning Workshop (NIPS)*, 2016.

[12] Juan C Caicedo and Svetlana Lazebnik. Active object localization with deep reinforcement learning. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, pages 2488–2496. IEEE, 2015.

[13] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

[14] Zheng Yang, Chenshu Wu, and Yunhao Liu. Locating in fingerprint space: wireless indoor localization with little human intervention. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 269–280. ACM, 2012.

[15] Duc A Tran and Cuong Pham. Fast and accurate indoor localization based on spatially hierarchical classification. In *Mobile Ad Hoc and Sensor Systems (MASS), 2014 IEEE 11th International Conference on*, pages 118–126. IEEE, 2014.

[16] Khuong An Nguyen. A performance guaranteed indoor positioning system using conformal prediction and the wifi signal strength. *Journal of Information and Telecommunication*, 1(1):41–65, 2017.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[19] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[20] Joaquín Torres-Sospedra, Raúl Montoliu, Adolfo Martínez-Usó, Joan P Avariento, Tomás J Arnau, Mauri Benedito-Bordonau, and Joaquín Huerta. 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*, pages 261–270. IEEE, 2014.

[21] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *Proceeding of International Conference of Learning Representations (ICLR)*, 2018.