This paper has been accepted for publication on SPIE Conference Big Data IV: Learning, Analytics, and Applications, Orlando, Florida, April 2022

On the source-to-target gap of robust double deep Q-learning in digital twin-enabled wireless networks

Maxwell McManus, Zhangyu Guan, Nicholas Mastronarde, and Shaofeng Zou

Department of Electrical Engineering, University at Buffalo, Buffalo, NY 14260, USA

ABSTRACT

Digital twin has been envisioned as a key tool to enable data-driven real-time monitoring and prediction, automated modeling as well as zero-touch control and optimization in next-generation wireless networks. However, because of the mismatch between the dynamics in the source domain (i.e., the digital twin) and the target domain (i.e., the real network), policies generated in source domain by traditional machine learning algorithms may suffer from significant performance degradation when applied in the target domain, i.e., the so-called "*source-to-target (S2T) gap*" problem. In this work we investigate experimentally the S2T gap in digital twin-enabled wireless networks considering a new class of reinforcement learning algorithms referred to as robust deep reinforcement learning. We first design, based on a combination of double deep Q-learning and an R-contamination model, a robust learning framework to control the policy robustness through adversarial dynamics expected in the target domain. Then we test the robustness of the learning framework over UBSim, an event-driven universal simulator for broadband mobile wireless networks. The source domain is first constructed over UBSim by creating a virtual representation of an indoor testing environment at University at Buffalo, and then the target domain is constructed by modifying the source domain in terms of blockage distribution, user locations, among others. We compare the robust learning algorithm with traditional reinforcement learning algorithms in the presence of controlled model mismatch between the source and target domains. Through experiments we demonstrate that, with proper selection of parameter R, robust learning algorithms can reduce significantly the S2T gap, while they can be either too conservative or explorative otherwise. We observe that robust policy transfer is effective especially for target domains with time-varying blockage dynamics.

Keywords: Zero-touch Networks, Digital Twin, Reinforcement Learning, Domain Adaptation, Source-to-Target Gap.

1. INTRODUCTION

Reinforcement learning (RL) has been envisioned as a key technique to enable zero-touch autonomous control in next-generation wireless networks [1–4]. Based on RL, the optimal or at least a desirable network operating point can be achieved by communication agents interacting in a self-organizing manner with the complex network environments. However, there are several challenges to address. First, it typically requires a large amount of data for the policy training to converge, and this process can be very time-consuming if the data is collected online at network run time. The situation will be even worse in large-scale time-varying networks. Second, while the training data can be generated offline using simulators, the trained neural networks may suffer from poor generalizability if the simulated and real transition kernels are very different. Moreover, collecting data directly in real environments may cause safety issues in networks with mobile nodes, since some states may be hazardous to people or the hardware.

To address the above challenges, in this work we explore a new network control approach based on digital twin (DT)-assisted RL. A DT system combines a physical domain system with its virtual replica, through which control, monitoring and prediction objectives can be achieved [5–7]. With DT, the state space can be explored

Further author information: (Send correspondence to M.M.)
M.M.: E-mail: memcmanu@buffalo.edu
Z.G.: E-mail: guan@buffalo.edu
N.M.: E-mail: nmastron@buffalo.edu
S.Z.: E-mail: szou3@buffalo.edu

This paper has been accepted for publication on SPIE Conference Big Data IV: Learning, Analytics, and Applications, Orlando, Florida, April 2022

possibly *faster than real time* by a virtual agent without actually interacting with the physical environment. In recent literature, DT has been shown to have a great potential in accelerating the training phase in data-driven mobile edge computing networks [8], UAV swarm networks [6], reconfigurable intelligent surface assisted wireless communications [9], among others. However, it is both challenging and time consuming to virtualize with high fidelity a physical network environment. As a result, because of the mismatch between dynamics in the source domain (i.e., the digital twin) and the target domain (i.e., the real network), policies learned in the source domain may suffer from the so called *source-to-target (S2T) gap* problem, i.e., their performance may be significantly degraded when applied in the target domain. In this work we take an initial step towards addressing this challenge by studying how to transfer the policy obtained in the source domain to the target domain in digital twin-enabled wireless networks. The primary contributions of this work are summarized as follows:

- *Domain-Adaptation Framework Design.* We first present a novel framework for testing domain adaptation algorithms in digital twin-enabled wireless networks. The framework combines double deep Q-learning for policy optimization and the R-contamination model for robustness to unexpected environmental dynamics. This provides configurable learning parameters for a variety of possible network control problems, as well as a tunable robustness parameter R to account for different levels of discrepancy between domain dynamics.
- *Scenario Development and Experimental Evaluation.* For source domain, we create a virtual replica of an indoor testing environment at University at Buffalo considering different factors that can affect the dynamics, such as blockage distribution and user locations. Then the target domain is constructed by introducing mismatch for those factors. Both source and target domains are deployed over UBSim, a newly developed event-driven broadband network simulator. Then, we create four network scenarios and analyze the domain adaptation capability of the proposed robust learning framework through an extensive experimentation campaign.

2. RELATED WORK

RL, particularly deep RL, has attracted significant research attention in data-driven wireless networks. For example, the authors of [10] use fully-connected deep neural network (DNN) for robot mobility prediction and user association in networks with ultra reliable low latency communications (URLLC). In [11], a double deep Q-learning architecture is used to predict the optimal channel access scheme in multi-user networks. The authors of [12] propose a fully-connected deep Q-network to predict the optimal MIMO beamforming in millimeter-wave networks. Deep Q-learning has also been adopted in [13] to predict the optimal channel access to maximize the long-term probability of successful transmission in heterogeneous networks. Similarly, the authors of [14] leverage deep Q-learning to make binary offloading decisions in mobile edge computing networks. Please refer to [15–17] and references therein for a good survey of the main results in this field. Different from these works, our work focuses on domain adaptation-enabled deep reinforcement learning in digital twin-based wireless systems.

Domain adaptation has also attracted significant attention in existing literature. Representative domain adaptation schemes include importance weighting [18], domain-agnostic features [19, 20], meta-learning [21], system identification [22], robustness mechanisms [23–25] and dynamics classifiers [26]. For example, domain adaptation is demonstrated for robotic control optimization in [19], leveraging domain-adversarial training with pixel-level observation adaptation for effective policy transfer. The authors of [26] propose probabilistic interpretation of RL, which leverages a modified utility function to penalize an agent for interacting with dynamics not present in the target domain. Sim-to-real transfer learning is investigated in [22], where robot navigation tasks are evaluated identically in both simulation and physical scenarios, demonstrating the impact of the reality gap. A model-free approach to robust learning is proposed in [24] to facilitate policy transfer across domains with varying dynamics. In [20], unsupervised adversarial domain adaptation is proven to improve generalization between synthetically-generated and real images through pixel-level domain classification. In [25], a three-phase learning system is devised by combining a source-domain object detection model, an image classification model and a robust object detection model. The authors of [18] introduce importance-weighted adversarial neural networks for partial domain adaptation. Meta-learning (or "learning to learn") is investigated in [21] to improve policy learning based on a generated set of Markov decision processes (MDPs) with different state transition probabilities. Readers are referred to [23] and references therein for a survey of robust reinforcement learning and

This paper has been accepted for publication on SPIE Conference Big Data IV: Learning, Analytics, and Applications, Orlando, Florida, April 2022

[27, 28] for other domain adaptation techniques. Different from these works, we study the domain adaptation capability of robust learning in digital twin-enabled wireless networks.

3. DOMAIN ADAPTION FRAMEWORK

In this section we propose a novel domain adaptation framework based on robust learning with R-contamination model [24]. To this end, we consider double deep Q network (DDQN) as our deep RL algorithm for its stability of convergence, ease of implementation, and the capability of policy generation in continuous state spaces.

3.1 DQN and DDQN: A Primer

Consider a discrete-time MDP problem characterised by $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is the transition kernel, r is a function which maps action a_t in state s_t to the immediate reward provided by the environment in time step t, and $\gamma \in [0, 1)$ is the discount factor. The transition kernel P is defined as $P = \{p_s^a \in \Delta_{|\mathcal{S}|}, a \in \mathcal{A}, s \in \mathcal{S}\}$, where $p_s^a \triangleq [p_{s,s'}^a]_{s' \in \mathcal{S}}$, $p_{s,s'}^a$ denotes the probability of ending up in state s' after taking action a in state s, and $\Delta_{|\mathcal{S}|}$ is the probability simplex over the states. Further define policy π as the mapping from observations in $\mathcal{S}$ to actions selected from $\mathcal{A}$. The action value function associated with policy π is defined as the expected cumulative discounted reward as follows:

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) | s_0 = s, a_0 = a, \pi\right]. \tag{1}$$

The optimal action value for each state-action pair (s_t, a_t), denoted as $Q^*(s_t, a_t)$, can be determined by the Bellman optimality equation [17]:

$$Q^*(s_t, a_t) = \mathbb{E}\left[r_t(s_t, a_t) + \gamma \max_{a \in \mathcal{A}} Q^*(s_{t+1}, a) | s_t, a_t\right]. \tag{2}$$

The goal of double deep Q-learning is to solve the MDP problem by finding the optimal policy π^* that maximizes the action value function $Q^\pi(s, a)$ for any initial state-action pair (s_0, a_0).

In Q-learning, the action value function $Q^\pi(s, a)$ is estimated through iterative updates, and the policy is determined based on a Q-table with dimensions $\mathcal{S} \times \mathcal{A}$ containing each action value. Experience is collected by an agent by making observations from $\mathcal{S}$, selecting actions from $\mathcal{A}$ and discovering rewards $r(s, a)$. These experiences are used to update the corresponding $Q^\pi(s, a)$ in the Q-table according to the following equation:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(r_t(s_t, a_t) + \gamma \max_{a \in \mathcal{A}} Q_t(s_{t+1}, a) - Q_t(s_t, a_t)), \tag{3}$$

where α is the learning rate, Q_t is the Q-table at time t, and the expression $Y_t \triangleq r_t(s_t, a_t) + \gamma \max_{a \in \mathcal{A}} Q_t(s_{t+1}, a)$ is called the *Q-target* [29]. After the Q-table is updated with sufficient experience, the policy π is determined by the maximum-value action in each state, i.e., $\arg\max_{a \in \mathcal{A}} Q^\pi(s_t, a)$.

With deep Q-learning, the policy π can be approximated by training a parameterized neural network [29]. This is especially preferable to the tabular method for MDPs with large or continuous state spaces. In deep Q-learning, experience tuples $(s_t, a_t, r_t(s_t, a_t), s_{t+1})$ are collected from the environment and stored at each time step in an experience replay buffer $\boldsymbol{D}$. The DQN is used to compute a vector of action values $\{Q_{\boldsymbol{\theta}}(s_t, a_{i,t})\}_{i=0}^{|\mathcal{A}|}$ in a given state s_t, where $\boldsymbol{\theta}$ are the parameters or weights of the neural network. Then, the Q-target Y_t can be rewritten as

$$Y_t = r_t(s_t, a_t) + \gamma \max_{a' \in \mathcal{A}} Q_{\boldsymbol{\theta}}(s', a'), \tag{4}$$

where $Q_{\boldsymbol{\theta}}$ is the Q-network parameterized by $\boldsymbol{\theta}$ and $r_t(s_t, a_t)$ is the immediate reward received by taking action a_t in state s_t. To train the DQN, the buffer $\boldsymbol{D}$ is shuffled to prevent time-domain correlation among samples, and a batch of experience tuples are selected to train the network. For each tuple, the maximum action value

This paper has been accepted for publication on SPIE Conference Big Data IV: Learning, Analytics, and Applications, Orlando, Florida, April 2022

$\max_{a\in\mathcal{A}} Q_{\boldsymbol{\theta}}(s_t, a_t)$ is calculated from the Q-network $Q_{\boldsymbol{\theta}}$ and compared to Q-target Y_t calculated in (4). The difference between Y_t and $\max_{a\in\mathcal{A}} Q_{\boldsymbol{\theta}}(s_t, a_t)$ is used to update the parameter set $\boldsymbol{\theta}$ as follows:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha(Y_t - Q_{\boldsymbol{\theta}}(s_t, a_t))\nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}, \tag{5}$$

where α is the learning rate of the network and $\nabla_{\boldsymbol{\theta}}$ is the gradient of the network training loss function.

Because both Y_t and $\max_{a\in\mathcal{A}} Q_{\boldsymbol{\theta}}(s_t, a_t)$ are computed by the same network in DQN, Y_t will keep changing as the action value estimated by $Q_{\boldsymbol{\theta}}$ gets closer to Y_t. This can reduce the training stability and lead to potentially sub-optimal performance. This challenge can be addressed in DDQN by using two DQNs with identical architecture for updating Q-value and Q-target [13, 29]. In DDQN, while the main DQN parameters $\boldsymbol{\theta}_{main}$ are updated according to (5) in the same way as single deep Q-learning, the target DQN parameters $\boldsymbol{\theta}_{target}$ are updated periodically with the same weights as the main network.

3.2 DDQN with Domain Adaptation

The objective of domain adaptation is to allow an agent to learn a policy in the source domain while applying the policy in the target domain. We consider as in [24] the R-contamination model (aka ϵ-contamination model) for predicting the relative difference between source and target domain dynamics. Considering domain transfer with identical transition probabilities, a policy learned in the source domain π_{source} can be applied directly to the target domain. However, as the difference between transition kernels P_{source} and P_{target} increases, the circumstances for which π_{source} is valid in the target domain become increasingly scarce. The goal of domain adaptation through robust learning is to learn a source-domain policy π_{source} using samples generated by P_{source} so that the best possible performance can be achieved with trajectories generated by P_{target}. Denote the transition kernel P at

Algorithm 1 DDQN with Domain Adaptation

Initialize: main network $Q_{\boldsymbol{\theta}_{main}}$, replay memory $\boldsymbol{D}$, target network $Q_{\boldsymbol{\theta}_{target}}$, state space $\mathcal{S}$, and action space $\mathcal{A}$
Design parameters: batch size N, exploration probability $\epsilon = 1$, contamination probability R
for each training episode **do**:
    Initialize $s_0 \in \mathcal{S}$
    for each environment step **do**:
        Observe state s_t
        Select i from the uniform distribution $\mathcal{U}_{[0,1]}$
        if $i \geq \epsilon$ **then**:
            select $a_t \leftarrow \arg\max_a Q_{\boldsymbol{\theta}_{main}}$
        else
            select $a_t \in \mathcal{A}$
            With probability R replace selected a_t with $a'_t \in \mathcal{A}\backslash\{a_t\}$
            Perform a_t and observe r_t and s_{t+1}
            Store tuple $e_t = (s_t, a_t, r_t, s_{t+1})$ in $\boldsymbol{D}$
        end if
    end for
    Shuffle $\boldsymbol{D}$ and sample batch $E = e_0 ... e_N$
    Compute Q-target Y_t using (4)
    Perform gradient descent step on $(Y_t - Q_{\boldsymbol{\theta}_{main}}(s_t, a_t))^2$
    Update $Q_{\boldsymbol{\theta}_{main}}$ parameters
    if training episode % target update period $= 0$ **then**
        Copy $Q_{\boldsymbol{\theta}_{main}}$ parameters to $Q_{\boldsymbol{\theta}_{target}}$
    end if
    Decay exploration probability: $\epsilon \leftarrow \epsilon * e^{-0.001}$
end for

This paper has been accepted for publication on SPIE Conference Big Data IV: Learning, Analytics, and Applications, Orlando, Florida, April 2022

time t as P_t, and the sequence of transition kernels as $\kappa = (P_0, P_1, ...), \forall t \geq 0$, where $P_t \in \mathcal{P}$, with $\mathcal{P}$ being the uncertainty set containing all possible combinations of state and next-state transitions $s, s' \in \mathcal{S}$. Further define the target domain problem as an MDP with unknown $P_{target} \in \mathcal{P}$. The size of $\mathcal{P}$ is an estimate of the amount of differences between P_{source} and P_{target}.

In order to estimate the transitions from P_{target} and account for unexpected transitions, we need to construct an estimate of $\mathcal{P}$ based on both P_{source} and selected transition probabilities from $\mathcal{P}$. Denote $\mathcal{P}_s^a \in \mathcal{P}$ as a subset of $\mathcal{P}$ to represent all possible distributions of P_{source} and P_{target} for state s and action a. We can then apply the R-contamination model to estimate transitions in $\mathcal{P}_s^a$ as follows:

$$\mathcal{P}_s^a = \{(1-R)p_s^a + Rq \mid q \in \Delta_{|\mathcal{S}|}\}, s \in \mathcal{S}, a \in \mathcal{A},\ 0 \leq R \leq 1, \tag{6}$$

where p_s^a is the transition probability introduced in Section 3.1, q is a state transition probability selected from $\mathcal{P}$, $\Delta_{|\mathcal{S}|}$ is the simplex of $\mathcal{S}$, and R is a design parameter representing the probability of transition according to q. The selected value of R describes the size of the uncertainty set $\mathcal{P}$. Based on R-contamination model, we take $\mathcal{P}_s^a$ as a sample-based estimation of the overall uncertainty set $\mathcal{P}$. The estimate $\mathcal{P}_s^a$ can be implemented in the DDQN learner using (6) by modifying possible next-step trajectories with probability R. In this way, the experience collected by an agent by interacting with different trajectories in an environment is perturbed during training to estimate $\mathcal{P}$. We implement the R-contamination in the experience replay of the DQN. Since the transition kernel P_{source} is native to the simulation environment and q is an arbitrary distribution from $\mathcal{P}$, this can be accomplished by randomizing the next state $s' \in \Delta_{|\mathcal{S}|}$ observed by the agent in each step with probability R. This perturbation is designed to train an agent to learn a policy to provide better performance under unexpected dynamics in an environment. The integration of R-contamination to double deep Q-learning is summarized in Algorithm 1.

4. DOMAIN CONSTRUCTION AND PERFORMANCE EVALUATION

In this section we test the domain adaptation capability of DDQN-based robust learning described in Algorithm 1 in a controlled simulation environment. To this end, we first construct different network scenarios in the source

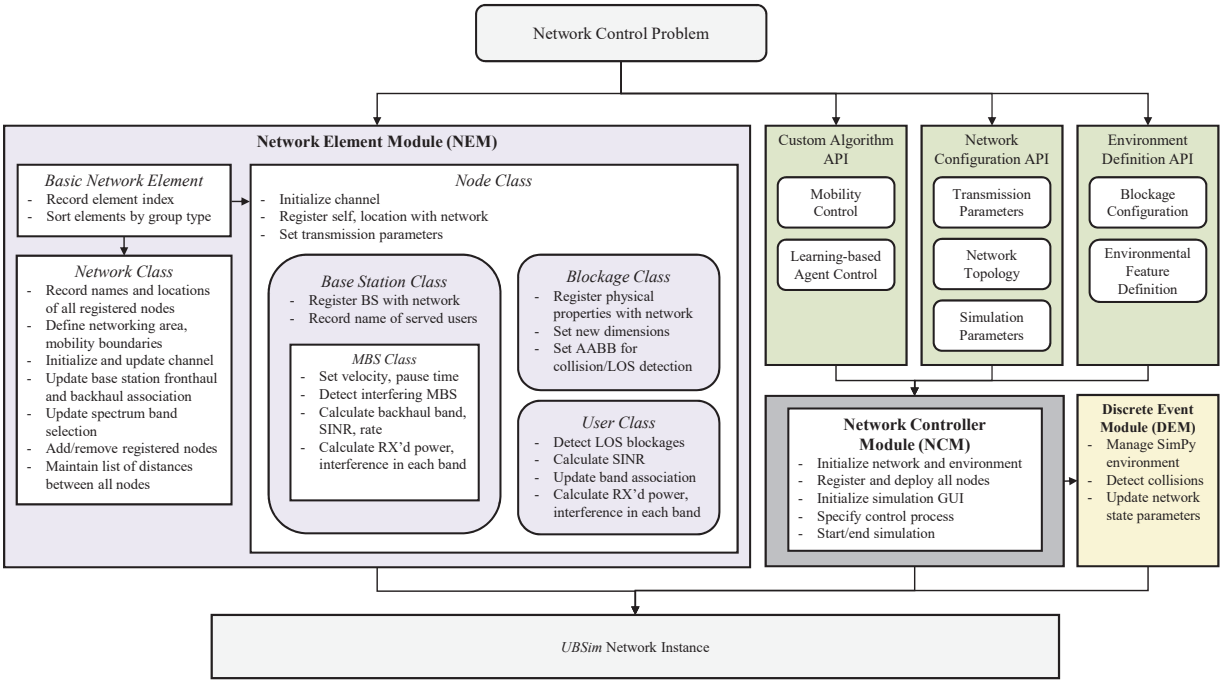

Figure 1: Architecture of the UBSim network simulator.

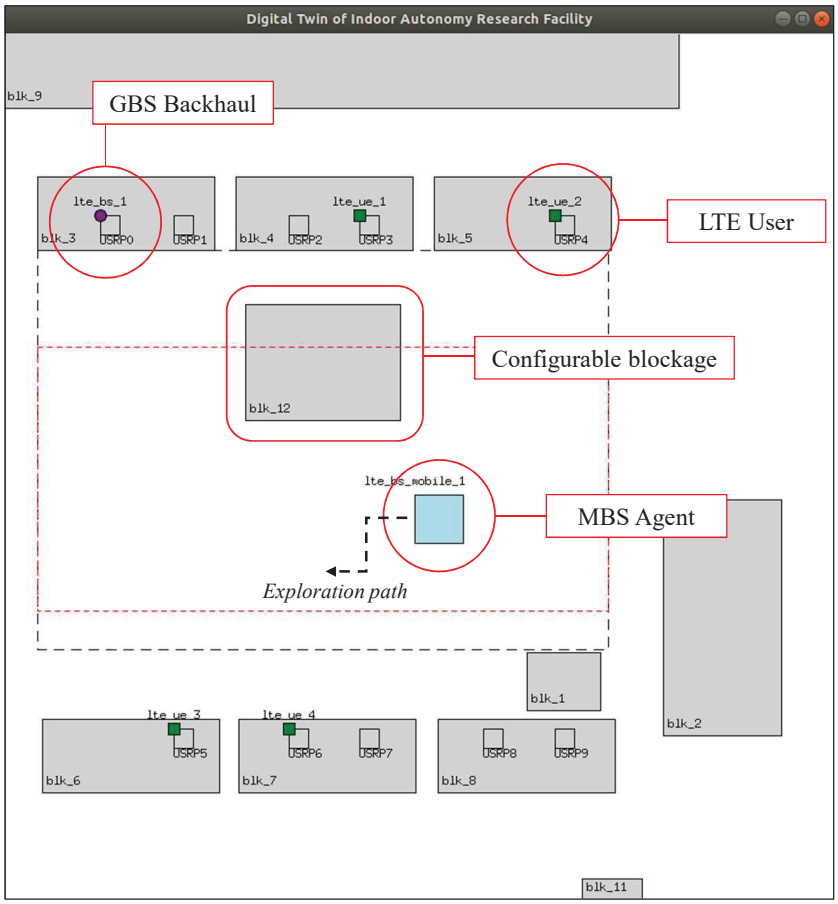

Figure 2: Example scenario in the source domain: the MBS agent determines its optimal location to maximize the sum user throughput.

and target domains over UBSim. UBSim is a custom event-driven wireless network simulator, written in Python 3.9 and supporting heterogeneous aerial-ground network topologies in microwave, millimeter-wave and terahertz bands. The diagram of UBSim is provided in Fig. 1, where there are three primary modules, i.e., network element module (NEM), network controller module (NCM), and discrete event module (DEM). NEM contains property and function definitions of all network element classes, including LTE base station (BS), LTE mobile base station (mBS), LTE UE and blockages. This module defines all functions necessary to model the interactions between network elements, including network association, SINR and interference calculations, and physical properties. The NCM module serves as an interface between NEM and DEM. NCM compiles configuration details provided by the APIs, including transmission parameters, mobility controls, and custom algorithm deployment, and uses these instructions for initialization of the network and virtual networking environment. Finally, DEM implements behavioral modeling of the defined network topology and visualization of the simulations.

4.1 Source and Target Domain Construction

Based on UBSim, the source domain is constructed considering wireless networks with mobile base station deployed over the indoor autonomy research testbed at Unviersity at Buffalo. The indoor autonomy research testbed consists of a netted UAV enclosure, a set of mobile nodes such as custom-built UAVs and ground robots, and a set of Universal Software Radio Peripherals (USRP) software radios (B210 and N210). The UAVs are capable of carrying USRP B210 and the ground robots are able to carry USRP B210 or N210 for experiments in integrated aerial-ground mobile wireless networks. In order to maximize the behavioral similarity between

the virtual and physical testbed deployments, manual measurements of this space were taken and added in UBSim, including networking area, testbed area, locations of user nodes, and other static environmental features such as lab furniture. The access links are configured to operate at 2.4 GHz band with 10 MHz bandwidth and the transmission power is set to 50 mW. A snapshot of the UBSim GUI is given in Fig. 2, where the MBS agent *lte_bs_mobile_1* is exploring the environment according to an ϵ-greedy exploration path, collecting and processing training samples as outlined in Algorithm 1. The agent maintains a communication link with the LTE users *lte_ue_1*, *lte_ue_2* and *lte_ue_3*. In this work, we consider a mobile BS with backhaul link that operates on frequency band orthogonal to the access links (e.g., the millimeter-wave band) and has sufficient capacity. That is, the backhaul link provided by *lte_bs_1* is sufficient to support the maximum possible aggregate throughput for all users. The configurable blockage *blk_12* is designed via the *Environment Definition API* and implemented via the NCM module of UBSim with an absorption coefficient sufficient to block most, but not all, RF communications between *lte_ue_1* and *lte_bs_mobile_1*.

We further design four scenarios in the target domain over UBSim with different levels of fidelity by introducing certain measurable discrepancy between source and target domains. For example, in the first scenario we consider a minimal noise floor (-174 dB) and zero blockages in the source domain. For the target domain, blockages are added to change the transition distribution of the corresponding MDP. Multiple variables are modified between policy training and testing in order to ensure sufficient distribution shift between source and target domains. These variables include the number and locations of static users, transmission bandwidth and frequency, the dimension, location and absorption coefficient of blockages, the number of blockages, and the global RF interference.

The objective of the mobile base station is to maximize the sum rate of the users by moving around the network. We define this network control problem as an MDP as in Section 3.1. Specifically, the current state s is defined as the coordinates of the mobile base station (i.e., the agent) within the netted area; the agent's actions $a \in \mathcal{A}$ are defined as moving up, down, left or right by a predefined distance or staying stationary in each time step; and the reward function $r(s, a)$ is defined as the change in the achievable sum user rate from state s to s'. In this way, the agent will be rewarded for transiting to states that improve the overall sum user rate

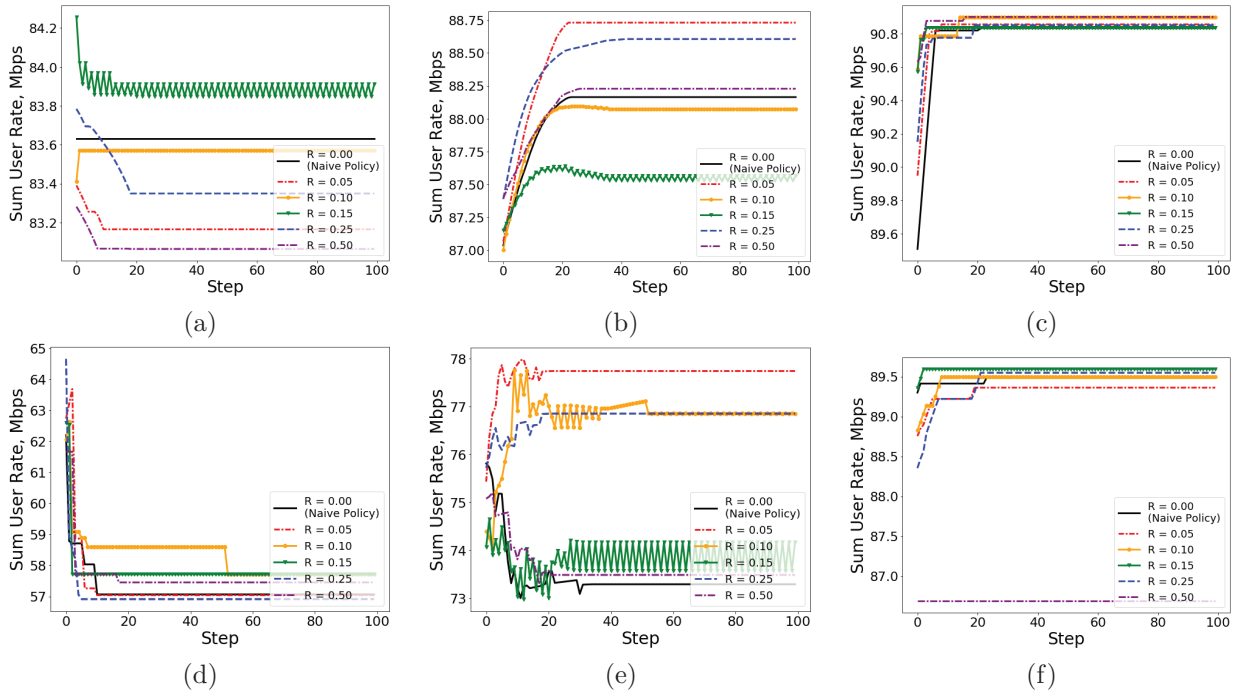

Figure 3: Minimum, average and maximum sum user rates achieved in source (upper) and target (lower) domains with different values of R in Scenario 1.

while will be penalized for transiting to states that reduce this value. The transition probabilities P are defined in Section 3 and the discount factor γ is set to 0.95. All policies are generated through 1000 training episodes in each scenario in the source domain, considering an ϵ-greedy policy with $\epsilon = 0.9$, and tested on 50 episodes each of 100 time steps with one action per step in the corresponding target domain. The DRL agent is designed with a neural network comprised of three dense layers, each with 128 neurons and "relu" activation functions. Tensorflow 2.7 has been used to construct a fully-connected neural network architecture, and we select the Adam optimizer to minimize mean squared error loss during backpropagation after each training episode.

4.2 Experimental Results

In the first scenario, the source domain is constructed with the most basic UBSim network configuration, which virtualizes the indoor autonomy research facility at University at Buffalo. The target domain introduces an arbitrary blockage with high RF absorption placed near the middle of the network area. As a result, the difference in transition kernel is caused by the mobility limitations imposed on the mobile BS node by this obstacle. The results are reported in Fig. 3. It can be seen that training with $R = 0.05$ or $R = 0.25$ provides good transfer learning across domains as shown in Fig. 3(b) for the source domain and Fig. 3(e) for the target domain; while the transfer learning performance of the naive policy ($R = 0$) is severely degraded in the target domain. Specifically, $R = 0.05$ provides the highest average sum user rate among all target domain evaluations at up to 78 Mbps, while $R = 0.10$ provides the highest average minimum value of 58.8 Mbps for most steps and $R = 0.15$ provides the highest average maximum value in general, at 89.6 Mbps. Considering a loss in average throughput of around 17% between source and target domains for the naive case in Scenario 1, optimal R selection reduces this loss to around 12%. In contrast, improper selection of R (i.e. $R = 0.50$ in this case) is shown to potentially limit the maximum throughput in the target domain to 86.5 Mbps, compared to over 89 Mbps for all other values as shown in Fig. 3(f). Additionally, as shown in Figure 3(b) and (e) for $R = 0.50$, improper selection of R may result in no change to average throughput compared to the naive case.

In the second scenario, the difference in transition kernel between source and target domains is attributed to the model mismatch between blockage dimensions and location in the source and target domain. To this

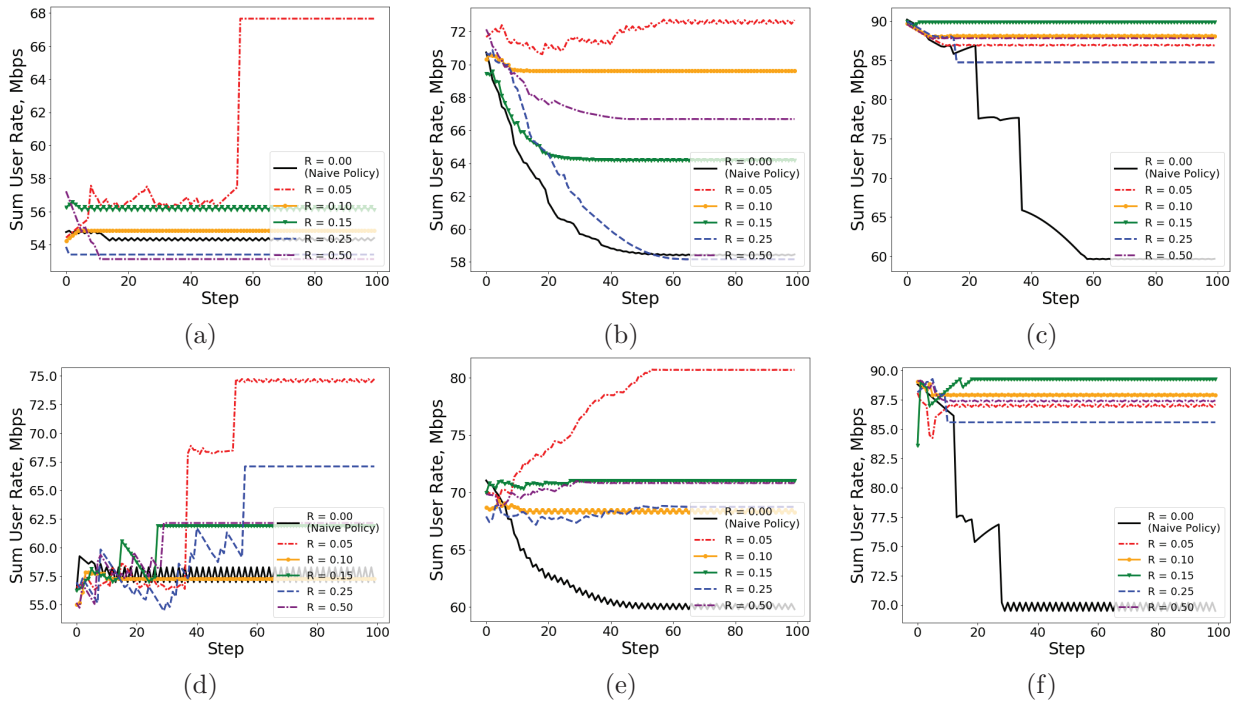

Figure 4: Minimum, average and maximum sum user rates achieved in source (upper) and target (lower) domains with different values of R in Scenario 2.

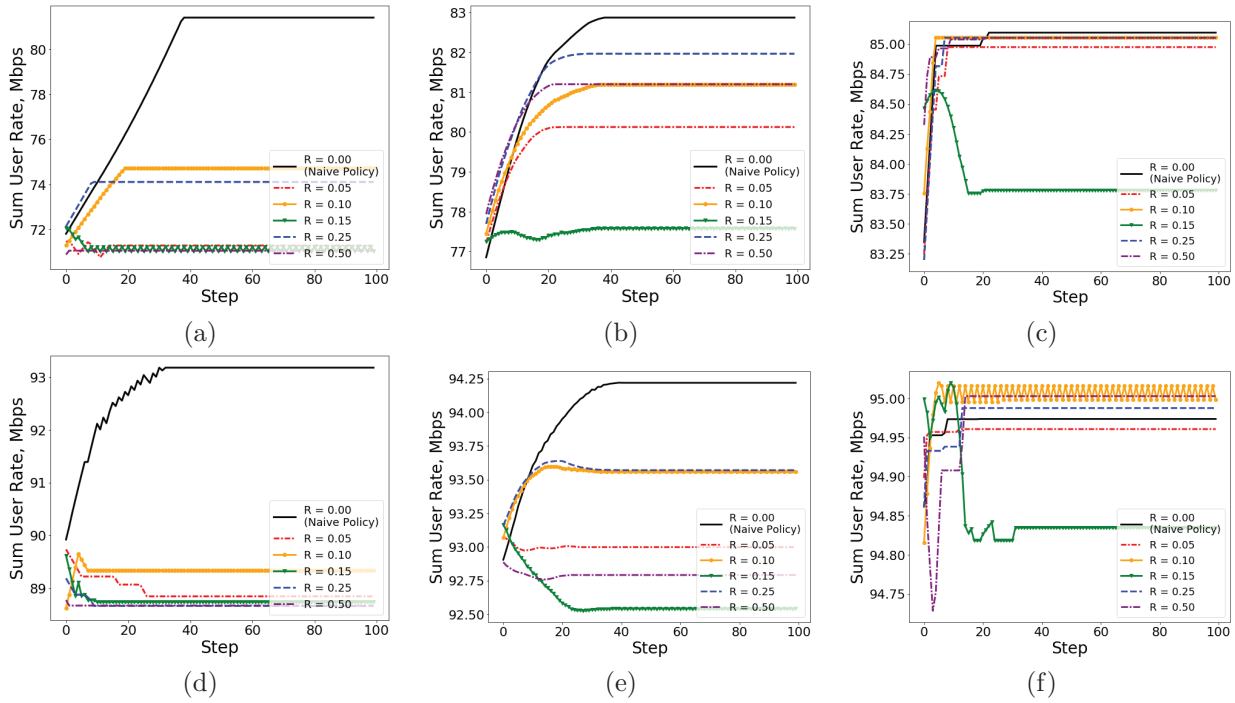

Figure 5: Minimum, average and maximum sum user rates achieved in source (upper) and target (lower) domains with different values of R in Scenario 3.

end, the configurable blockage is dilated by 20 cm in all dimensions and moved 20 cm north and 20 cm west in the target domain. We report the results in Fig. 4, which demonstrates in general much worse naive policy transfer performance than in the first scenario, with the average rate decreasing from 70 Mbps to around 60 Mbps in both source and target domains. Interestingly, the gap between domains is not apparent from the average throughput results, indicating that the naive agent in the source domain was unable to converge to any optimal point. However, we observe improved performance over the naive case for all R values in the source domain, achieving on average 80 Mbps with $R = 0.05$ and 70 Mbps throughput for other R values. Specifically, Figs. 4(a) and (d) demonstrate a significant improvement in the minimum sum user rate across testing episodes provided by robust training at $R = 0.05$, providing an average minimum rate of 75 Mbps by the end of the testing episode, providing 25-30% rate improvement in both domains. However, it is clear from all subfigures in Fig. 4 that the naive policy (i.e. $R = 0$) cannot achieve good performance in general in this scenario, and that the robust policy was able to navigate better in the obstructed environment.

In the third scenario, the transition kernel is altered by the difference in sum user rate distribution hence reward function across the domains. Specifically, we consider the source domain with two LTE users (users a and b) and the target domain with two new users (users c and d). Users a and b start respectively at the positions of USRPs 2 and 3 in Fig. 2 and do not change their positions during the experiment. Users c and d are deployed at the next available USRP indices (i.e., USRPs 4 and 5). It can be seen in Fig. 5 that, which is somewhat surprising, the basic DDQN agent outperforms the robust DDQN implementations in both source and target domains, with consistently higher overall sum user rate. Interestingly, the naive policy was better able to maximize the sum rate with the addition of new users than any $R > 0.00$. The naive policy achieved an average rate of 94.25 Mbps in the target domain, which shows 13.5% improvement over the average source domain rate of 83 Mbps. However, the best selection of $R > 0.00$, which was determined to be $R = 0.25$, achieved 93.5 Mbps in the target domain, compared to the source domain average of 82 Mbps. The worst selection of $R > 0.00$, i.e. $R = 0.15$, demonstrated no throughput gains to maximum, minimum, or average rate with the addition of more users, indicating an inability to account for the difference in optimal BS location. This indicates that the policy generated by the R-contamination model is more conservative than the naive case in this scenario.

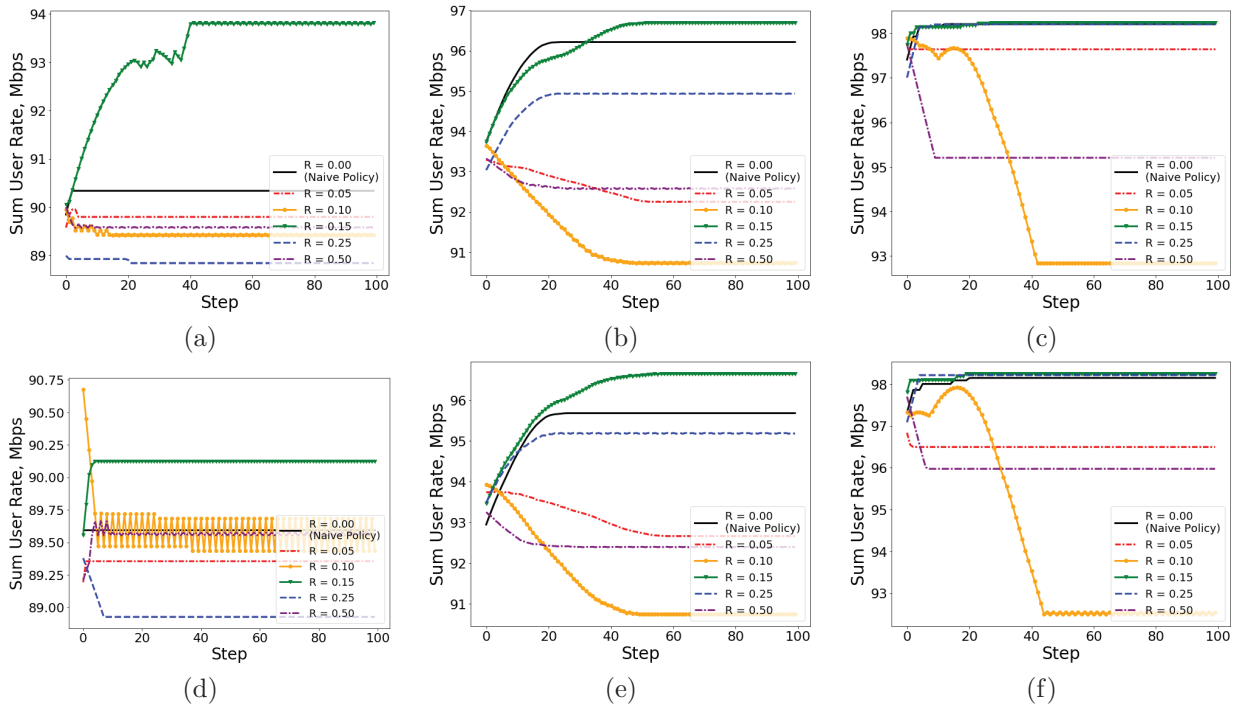

Figure 6: Minimum, average and maximum sum user rates achieved in source (upper) and target (lower) domains with different values of R in Scenario 4.

Finally, in the fourth scenario we consider domain discrepancy caused by users migrating between training and testing time. To this end, we deploy four users starting from the first USRP index (i.e. USRPs 1, 2, 3, 4) in Fig. 1 in the source domain. In the target domain, the four users move to USRPs 3, 4, 5 and 6, respectively. The results are shown in Fig. 6. In Figs. 6(a), (b), (d) and (e), the agent with $R = 0.15$ resembles very closely the naive implementation with $R = 0.00$ in the source domain, achieving 96.7 Mbps compared to 96.2 Mbps in the naive case, while providing an improvement of roughly 1 Mbps to the average rate of 95.5 Mbps achieved by the naive case in the target domain. The minimum rate achieved by $R = 0.15$ was 90.1 Mbps, which shows 0.5 Mbps improvement over the naive minimum rate of 89.6 Mbps. The maximum achieved rate for both policies is the same in source and target domains at roughly 98 Mbps. Although the difference is small, i.e. $< 5\%$ difference in most experiments2, this behavior represents an accurate selection of R in that the robust case provides both similar performance in the source domain and some performance improvement over the naive case in the target domain. We expect the difference to be small since the number of users is not changing and there are no obstructions to LOS conditions of the MBS. In contrast, the worst selection of R, i.e. $R = 0.10$, provided roughly 5 Mbps lower throughput than the naive case in both source and target domains, demonstrating an inability to generalize to the new scenario.

5. CONCLUSIONS AND FUTURE WORK

In this work we have analyzed experimentally the S2T gap of a new class of reinforcement learning algorithms referred to as robust deep reinforcement learning in the context of digital twin-enabled wireless networks. We first designed a robust learning framework to control the policy robustness through adversarial dynamics expected in the target domain based on a combination of double deep Q-learning and R-contamination model. We then tested the robustness of the learning framework over UBSim. The source domain was constructed over UBSim by creating a virtual representation of an indoor testing environment at University at Buffalo, and the target domain was constructed by modifying the source domain, e.g., the blockage distribution and user locations. We compared the robust learning algorithm with traditional reinforcement learning algorithms in the presence of

controlled model mismatch between the source and target domains. Through an extensive simulation campaign we verified that the negative impact of unknown dynamics of a target domain on an agent's performance can be effectively reduced by a robust policy trained on source domain dynamics perturbed by the R-contamination model. From the results we can observe that the robust learning (with $R > 0.00$) can improve the policy transfer performance especially when there are blockage dynamic differences between the source and target domains. However, this requires accurate selection of R, which further requires feedback from the target domain during training. It is also shown in some scenarios that robust learning is demonstrated to be more conservative than naive policy transfer, which may not be favorable without further knowledge of the S2T gap. In future work we will further analyze the domain adaptation capabilities of robust Reinforcement learning by considering testbed experiments for the target domain, and determine the optimal contamination model parameter R based on online interaction between the source and target domains.

References

[1] Huang, H., Yang, Y., Wang, H., Ding, Z., Sari, H., and Adachi, F., "Deep reinforcement learning for UAV navigation through massive MIMO technique," *IEEE Transactions on Vehicular Technology* **68**, 1117–1121 (Nov. 2020).

[2] Moorthy, S. K., Guan, Z., Pudlewski, S., and Bentley, E. S., "FlyBeam: echo state learning for joint flight and beamforming control in wireless UAV networks," in [*Proc. of IEEE International Conference on Communications (ICC)*], (June 2021).

[3] Moorthy, S. K. and Guan, Z., "FlyTera: echo state learning for joint access and flight control in THz-enabled drone networks," in [*Proc. of IEEE International Conference on Sensing, Communication and Networking (SECON)*], (June 2020).

[4] Moorthy, S. K., McManus, M., and Guan, Z., "ESN reinforcement learning for spectrum and flight control in THz-enabled drone networks," *IEEE/ACM Transactions on Networking* (October 2021).

[5] Sleuters, J., Li, Y., Verriet, J., Velikova, M., and Doornbos, R., "A digital twin method for automated behavior analysis of large-scale distributed IoT systems," in [*Proc. of Annual Conference System of Systems Engineering (SoSE)*], (May 2019).

[6] Yang, Y., Meng, W., and Zhu, S., "A digital twin simulation platform for multi-rotor UAV," in [*Proc. of International Conference on Information, Cybernetics, and Computational Social Systems (ICCSS)*], (Nov. 2020).

[7] Zohdi, T. I., "A machine-learning framework for rapid adaptive digital-twin based fire-propagation simulation in complex environments," *Computer Methods in Applied Mechanics and Engineering (Elsevier)* **363**, 1–19 (May 2020).

[8] Lu, Y., Huang, X., Zhang, K., Maharjan, S., and Zhang, Y., "Communication-efficient federated learning and permissioned blockchain for digital twin edge networks," *IEEE Internet of Things Journal* **8**, 2276–2288 (Feb. 2021).

[9] Sheen, B., Yang, J., Feng, X., , and Chowdhury, M. M. U., "A digital twin for reconfigurable intelligent surface assisted wireless communication," *arxiv.org* (Sept. 2020).

[10] She, C., Dong, R., Gu, Z., Hou, Z., Li, Y., Hardjawana, W., Yang, C., Song, L., and Vucetic, B., "Deep learning for ultra-reliable and low-latency communications in 6G networks," *IEEE Network* **34**, 219–225 (September/October 2020).

[11] Janiar, S. B. and Pourahmadi, V., "Deep-reinforcement learning for fair distributed dynamic spectrum access in wireless networks," in [*IEEE Consumer Communications Networking Conference*], (January 2021).

[12] Lizarraga, E. M., Maggio, G. N., and Dowhuszko, A. A., "Deep-reinforcement learning for hybrid beamforming in multi-user millimeter wave wireless systems," in [*IEEE Vehicular Technology Conference*], (April 2021).

[13] Zong, K., "Deep-reinforcement learning-based dynamic multichannel access for heterogeneous wireless networks with DenseNet," in [*IEEE/CIC International Conference on Communications in China (ICC Workshops)*], (July 2021).

[14] Huang, L., Bi, S., and Zhang, Y. A., "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Transactions on Mobile Computing* **19**, 2581–2593 (November 2020).

[15] Lei, L., Tan, Y., Zheng, K., Liu, S., Zhang, K., and Shen, X., "Deep reinforcement learning for autonomous Internet of Things: model, applications and challenges," *IEEE Communications Surveys Tutorials* **22**(3), 1722–1760 (2020).

[16] Feriani, A. and Hossain, E., "Single and multi-agent deep reinforcement learning for AI-enabled wireless networks: a tutorial," *IEEE Communications Surveys Tutorials* **23**(2), 1226–1252 (2021).

[17] Luong, N. C., Hoang, D. T., Gong, S., Niyato, D., Wang, P., Liang, Y.-C., and Kim, D. I., "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surv. Tutor.* **21**, 3133–3174 (Fourth Quarter 2019).

[18] Zhang, J., Ding, Z., Li, W., and Ogunbona, P., "Importance weighted adversarial nets for partial domain adaptation," in [*Conference on Computer Vision and Pattern Recognition*], (June 2018).

[19] Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., Levine, S., and Vanhoucke, V., "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in [*IEEE International Conference on Robotics and Automation*], (May 2018).

[20] Romijnders, R., Meletis, P., and Dubbelman, G., "A domain agnostic normalization layer for unsupervised adversarial domain adaptation," in [*IEEE Winter Conference on Applications of Computer Vision*], (January 2019).

[21] Arndt, K., Hazara, M., Ghadirzadeh, A., and Kyrki, V., "Importance weighted adversarial nets for partial domain adaptation," in [*IEEE International Conference on Robotics and Automation*], (August 2020).

[22] Kadian, A., Truong, J., Gokaslan, A., Clegg, A., Wijmans, E., Lee, S., Savva, M., Chernova, S., and Batra, D., "Sim2Real predictivity: does evaluation in simulation predict real-world performance?," *IEEE Robotics and Automation Letters* **5**, 6670–6677 (October 2020).

[23] Chen, S. and Li, Y., "An overview of robust reinforcement learning," in [*IEEE International Conference on Networking, Sensing, and Control*], (Oct-Nov 2020).

[24] Wang, Y. and Zou, S., "Online robust reinforcement learning with model uncertainty," in [*Advances in Neural Information Processing Systems*], (December 2021).

[25] Khodabandeh, M., Vahdat, A., Ranjbar, M., and Macready, W., "A robust learning approach to domain adaptive object detection," in [*International Conference on Computer Vision*], (October-November 2019).

[26] Eysenbach, B., Asawa, S., Chaudhari, S., Levine, S., and Salakhutdinov, R., "Off-dynamics reinforcement learning: training for transfer with domain classifiers," in [*International Conference on Learning Representations*], (May 2021).

[27] Kouw, W. M. and Loog, M., "A review of domain adaptation without target labels," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **43**, 766–785 (October 2021).

[28] Wilson, G. and Cook, D. J., "A survey of unsupervised deep domain adaptation," *ACM Transactions on Intelligent System Technology* **11**, 51:1–46 (July 2020).

[29] Hasselt, H., Guez, A., and Silber, D., "Deep reinforcement learning with double Q-learning," in [*AAAI Conference on Artificial Intelligence*], (March 2016).