Transfer Learning via Online System Identification
CPSC 533V Project Report

Daniel Ajisafe (72918105), Curtis Fox (23673149), Tianyue Zhang (24991151)*
University of British Columbia
Vancouver, Canada

Abstract

Sim-to-real gap has been a challenging problem in most Reinforcement Learning (RL) settings. And this is largely due to the fact that the majority of RL algorithms have been trained on selected environment parameters. In this project, we build on an existing method and try to learn a control policy that is prepared to be executed in a dynamic environment, where the parameters of the environment are changing. Such policy with online system identifying capabilities also tends to be more robust to environments outside of the training range, which could help in closing the reality gap. Specifically, we first implement the main components of the baseline paper from scratch and then propose different modifications to the architecture. Quantitatively, we test the robustness of our new modifications and run ablation studies beyond the existing pipeline.

1 Introduction

Reinforcement Learning problems in simulation often train and test agents in a known or pre-determined environment. However, when deploying policies to the real world, more often the environment could be unknown or even dynamic. These kinds of environments can be much harder to work with, as the policy of the agent needs to be far more robust than in classic RL problems with fixed environments, and it is more costly to make mistakes in real environments. Closing this gap between simulation and reality has become a popular research direction and could potentially enable many RL techniques to be applied in real life. Existing works [1][2][3] have tried to approach this problem via transfer learning, learning a robust policy or a more accurate model, etc.

Yu et al. [4] proposed using an online system identification to determine dynamic system parameters on-the-go. The purpose of their approach was meant to handle the more complicated case where the environment is unknown. This entails developing a universal policy that is both applicable and robust to a variety of environments. It also involves the agent being able to determine exactly what environment it is, so that the universal policy can be effectively utilized. This advantage of this approach is practical for many reasons. For one, it simulates the dynamic environment that autonomous cars are deployed in, having being trained first from simulation. Such application requires the agent (car in this case) to have a very robust policy, and be able to accurately assess the current environmental conditions. This is because a poorly predicted estimate of the environment could have fatal consequences. Other applications of RL in unknown environments such emergency response agents are also areas of interest and are being actively researched. These are among many of the possible applications.

To this end, our project follows closely with developing a robust policy for unknown environments. The work of [4] positions itself as our baseline method. Our objective, therefore, is to implement the main components of their approach, attempt to reproduce their results, and moreso, develop some extensions to their current pipeline. At a high level, our project steps went as follows. We picked an environment to work with, and chose the parameters that we would perturb to simulate an unknown environment. In our initial implementation, we developed a code base for the portion of their algorithm that finds a universal policy. Notably, we build on the code for assignment 4 in training the universal policy, which served as a good baseline. We then implemented their OSI algorithm using their text-written pseudocode. After that, we linked these algorithms together to create the final pipeline discussed in our methodology section. However, we highlight some key differences in our project compared to the model paper, and also showcase the propositions that we made as extension to their

[*] Authors contributed equally and listed in alphabetical order as per last names

pipeline. Finally, we ran ablation studies and discuss our results in the results and discussion section.

2 Related Works

2.1 Deep Reinforcement Learning

Neural networks from deep learning are used to find good policies for reinforcement learning problems, giving birth to the name deep RL. A commonly known deep RL method is the double DQN algorithm [5] which is an extension of the initial DQN method. The rationale behind their new algorithm is that previous versions of this algorithm could incorrectly estimate action values, which could in the worst case even lead to sub-optimal policies. By combining very classical RL technique of Q-learning with neural networks, they show that they show that their algorithms are more stable in training and can overall lead to better policies.

Another important aspect of deep reinforcement learning is in entertaining applications such as Atari games. [6] uses a convolutional neural net with image pixels to develop policies for the agent to follow. [7] also applied deep RL to the area of inventory control. The importance of neural networks in RL has grown and has bcecome an active area of research in the space.

2.2 Transfer Learning in Reinforcement Learning

Transfer learning [8] has been an effective training strategy in recent times, where we we use the information found from training one problem to bootstrap the training of a new, but usually similar problem. In the context of RL, this would mean reusing policies for similar problems. One particular application of transfer learning is in the context of multiagent problems. In [9], the authors developed a new Multiagent Policy Transfer Framework (MAPTF), which allows agents to effectively learn from each other. This framework works for agents undergoing various experiences as they interact with their environment while selecting the appropriate policy for each agent to follow (since each agent could be following a different policy).

Another example of transfer learning in RL can be seen in games. [10] used the same idea in transferring data about policies from one game to another. In particular, the authors used a network learned from the game Puckworld as a starting point for training a network for the game Snake. They find that this leads to the agent performing better in the Snake game. Also, transfer learning in RL is also used in the context of robots as well, as seen in [11]. In particular, the robot goes through various experiences in one environment and uses what it has learned to better adapt to a new environment it is put into. This parallels the ideas presented in our model paper [4], where our agent needs a policy that allows it to perform well in a variety of different environments.

2.3 Learning Policy in Unknown Environment

Technically, learning a policy in an unknown and possibly dynamic environment can be challenging. Amongst several works, [1] also explores the situation where the environment is both unknown and dynamic. In particular, their paper uses an actor-critic method and recurrent neural networks to guide robots through an environment that has moving obstacles. They justify their approach by stating that it allows the robot agent to better handle and take more appropriate actions in a noisy environment. A more recent and potentially complicated domain is the work done by [2]. For this paper, the authors explore how to develop a policy for unmanned aerial vehicles (UAV's) to effectively navigate through an unknown environment. They also mentioned that previous approaches to this problem were often insufficient and were broken up into more than one phase in an attempt to explore the environment first, before exploiting the environment in a later phase to learn an optimal policy. However, as they mention, this is not always practical or even useful, since in a dynamic environment this would not work. If the environment is continually changing, the agent needs to be able to learn the optimal policy as the environment changes. They propose an adaptive algorithm that can achieve both exploration and exploitation in one phase.

However, [3] demonstrated the possibility of learning in unknown environments for multi-agent settings. The authors looked at dealing with the situation where more than one agent is present in the environment. In particular, they examined the problem of crowd navigation which is a problem of interest because the agents need to avoid colliding with obstacles in the environment as well as with each other, thereby adding further complexity. Even though our model paper [4] doesn't discuss the multi-agent case, it's very much a relevant future direction.

3 Methodology

In this section, we discuss briefly the UP-OSI algorithm from our model paper [4], and highlight more on our modifications to this approach. The algorithm consists of two main components, a universal policy (UP) and an online system identification model (OSI).

3.1 Training UP - The Universal Policy

Actor-critic Networks proposed by [12] have shown to be successful in developing good control policies, with the advantage of using a critic network to nudge the policy to good or optimal solutions. The model paper [4] used a 2-layer fully-connected network to learn their policy. This is simple but does not take the advantage of

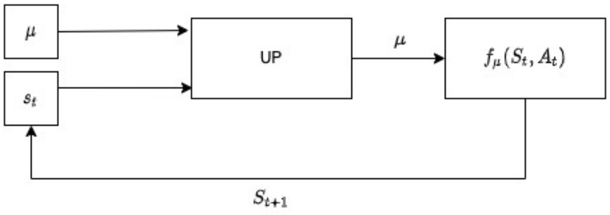

Figure 1: UP Diagram: The universal policy formulated as a reinforcement learning problem where (UP), $\pi : (s, \mu)$ takes as input, the state and environment parameter to give an action u. The action is effected on the environment to produce the next state s_{t+1}.

using a critic network. We leverage that advantage and used a critic network for our policy. Specifically, we start from the codebase in homework 4, where we train an actor-critic network using a proximal policy optimization (PPO) method with a clipped surrogate objective and GAE lambda advantage. However, the policy from this setup is for a fixed environment parameter μ, which is the default parameter given by OpenAI gym [13].

In order to generalize to dynamic settings, we extend the exploration scheme, similar to [4], such that the policy takes the environment parameter μ as an input, which is sampled from a uniform distribution ρ_μ. For each μ_i, we generate a set of rollouts under the policy $\pi(s, \mu_i)$ and store the state, action, and reward collected from forwarding. Then we update the policy π using PPO. See Figure 1.

Algorithm 1 Learning UP with PPO clipped objective

```
Randomly initialize UP network $\pi$
for i=1:K do
    $\mu \sim \rho_\mu$
    $s \sim \rho_0$
    while $R.size \leq MaxStep$ do
        $a = \pi(s, \mu)$
        $s = f_\mu(s, a)$
        $r, terminated = Reward(s, a)$
        Push $(s, a, r)$ into R
    end while
    Update $\pi$ with data in R using PPO
end for
return $\pi$
```

$$L^{CLIP}(\theta) = E_t[\min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)] \tag{1}$$

where ϵ is an hyperparameter, A_t is advantage term, and r_t is the probability ratio defined as,

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{2}$$

3.2 Training OSI - The Online System Identification

With a good universal policy (UP), the model now can perform well if given true but changing environment parameters, but this true information is practically not available in most real-world settings. However, we need a system that can identify the environment online, which means, good enough to work with estimates instead of true parameters. To achieve this, we formulate a supervised problem similar to [4] to train the online system identification model (OSI). In contrast to their approach, we designed our OSI to be independent of the current state s_t, and only need the recent history to sufficiently make parameter estimates. Hence, OSI denoted as ϕ takes as input the state-action pairs to predict the environment parameter μ. The input and illustration of the internal components of OSI is shown in Figure 4.

Our optimization for OSI follows a residual error minimization objective described in Eqn 3.

$$\theta^* = \arg\min_\theta \sum_{(H_i,\mu_i)\subseteq B} |\phi_\theta(H_i) - \mu_i|^2 \tag{3}$$

where θ are the parameters of the neural net ϕ_θ, μ is the true environment parameter, H is a history buffer that stores a fixed size of state and actions taken by the agent, and B is a buffer that stores the information of updated H generated in each environment. We modified their algorithm to better suit our environment and collected data for training OSI. The details are shown in Algorithm 2. We also ablate on different strategies for minimizing our error objective and subsequently training OSI. We discuss these strategies in the experimental results and discussion section.

3.3 Connecting UP and OSI

Once the UP and OSI network are trained, sequentially, we connect them together into a single UP-OSI system. In this system, UP performs online inference in an unknown environment. A history rollout buffer that is generated in real time is fed as input to OSI to make predictions about the environment. The environment parameter estimates are given to the universal policy to take the optimal action. Note that the policy can be further refined in this new environment, but even without online learning, it gets a warm start in an environment that is outside of its training range. The pipeline is shown in Figure 2.

4 Environment

We use Cart-Pole, classic control environment provided by OpenAI gym [13], named *CartPole-v0*. In this environment, a pole is attached to a cart, which moves along a

Algorithm 2 Learning OSI

```
Randomly initialize OSI network $\psi$
for i=1:K do
    $\mu \sim \rho_\mu$
    for j=1:N do
        Initialize history queue H
        Fill H by simulating under $\pi(s, \overline{\mu})$ and $f_{\overline{\mu}}$
        for t=0:T-1 do
            Pop H
            $a_t = \pi(s_t, \overline{\mu})$
            $s_{t+1} = f_{\overline{\mu}}(s_t, a_t)$
            Push $(s_{t+1}, a_t$ in H
            $H, \overline{\mu}$ in B
        end for
    end for
end for
Optimize $\psi$ using data in B
return $\psi$
```

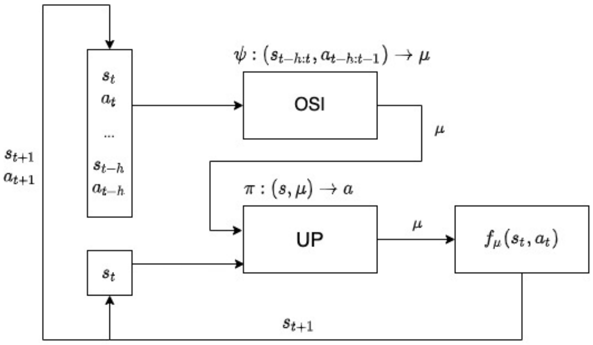

Figure 2: UP-OSI Pipeline: UP takes as inputs the current state and environment parameter μ predicted by OSI, and outputs a policy that steps in the environment to produce the next state and action. This information updates the history rollout buffer H, which is an input of OSI, and the environment parameter prediction μ is updated based on the new state and action history.

frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center. Action space is either moving the cart to the left or to the right. Observation space is 4-dimensional, that includes the cart position, cart velocity, pole angle, and pole angular velocity. See Figure 3

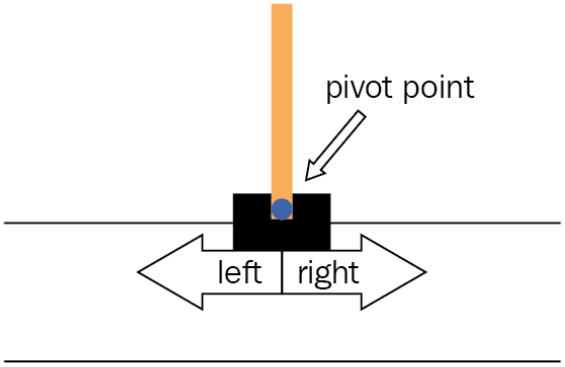

Figure 3: OpenAI Cart-pole action space[14].

Env Prarameters	Training Range	Testing Range
Gravity	[6.8-12.8]	[12.8-15]
Cart Mass	[0.1kg-1kg]	[1.0kg-1.9kg]
Pole Mass	[0.1kg-.5kg]	[0.5kg-0.9kg]
Pole Length	[0.2-0.8]	[0.8-1.4]

Table 1: Environment parameters and the split range used in our experiment

Our environment parameter μ is a 4D vector, with each dimension representing gravity, cart mass, pole mass, and pole length respectively. For the training range, we chose a reasonable set of values below and above the true parameters of the environment. For testing, we define a range outside the training range, to see how policies perform in unseen scenarios, as detailed in 1. And for an extreme case, we set gravity to a very high value, 70 without changing other parameters.

5 Experimental Result and Discussion

In this section, we discuss the result of our experiments as well as the effect of different choices.

5.1 Evaluate UP Policy

First, we compare the performance of direct policy transfer (by directly using policy trained on one environment without parameter input, to perform in another environment) versus the universal policy. For direct transfer, we trained our policy using PPO and GAE lambda advantage on the default cart-pole environment provided by OpenAI gym. We trained on default environment parameters for 250 epochs. For universal policy, we trained with the same setup but with 30 sets of different environment parameters uniformly sampled from the training range, each for 250 epochs.

We ran several tests with the two trained policy weights, using environment parameters that is sampled from the test range. The UP policy did not perform as well as direct policy transfer, we suspect it is because in a simple environment such as cart-pole, our policy is not sensitive to small changes in the environment. Therefore we instead tested the two policies in an environment with

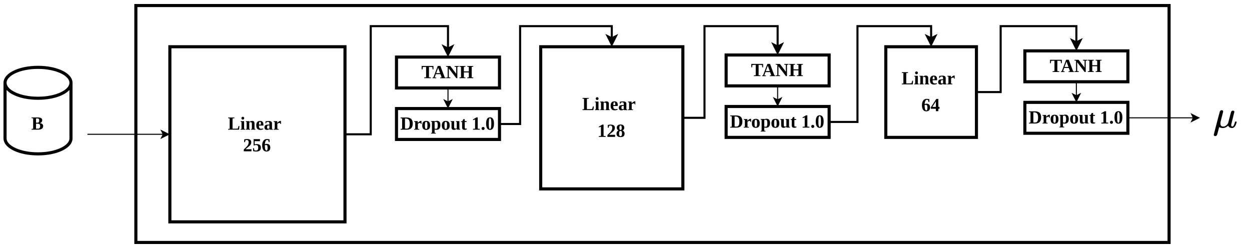

Figure 4: A diagrammatic representation of our OSI model. Similar to [4], the network is composed of 3 Linear layers where the size of each layer is reduced by a factor of 2. Each of them is followed by a tanh activation function and a dropout of 1.0. The input to the network is the history buffer, and the output of the network is the estimated model parameters.

gravity = 70, to see if policies perform differently in an extreme case. The result is shown in Figure 5.

Clearly, universal policy performs better in an environment that is completely different from the training environment. While this trial is only using the trained weights to perform roll-outs, we are also interested in the case where the agent continues to learn the new environment. We allowed the policy to be updated to see if the two methods could achieve the same result. See Figure 6.

From the figure, we can see that although UP policy gets a head start in the new environment, a simple policy without knowing the environment could achieve a higher reward faster. The reason for this could be that there is a trade-off between being robust to environment changes and learning one stationary environment quickly. Direct policy transfer could work better in a stationary environment with enough training. An interesting future direction is to see if universal policy has better online performance, especially in a dynamic environment.

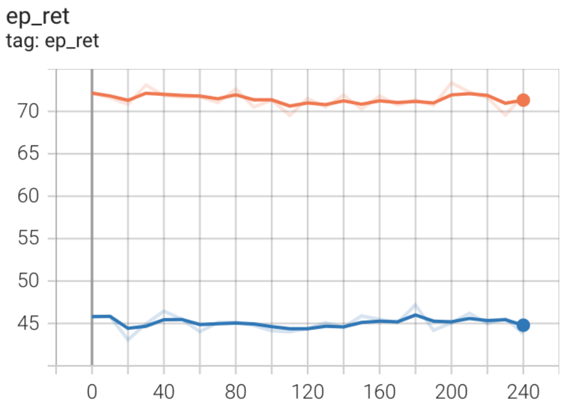

Figure 5: Direct policy transfer(blue) versus universal policy(orange) in an environment that is extremely different from training environment(gravity = 70), without updating policy

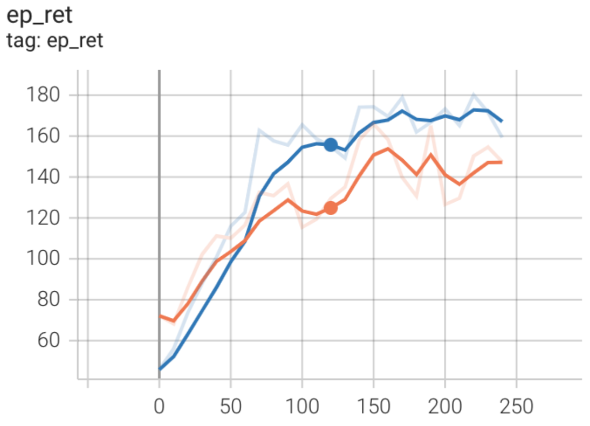

Figure 6: Direct policy transfer(blue) versus universal policy(orange) in an environment that is extremely different from training environment(gravity = 70), while updating policy

5.2 Evaluating OSI

Batch size	Loss Function	Train (error)	Validation (error)
32	L1	0.0207	0.0181
32	L2	0.0387	0.0358
64	L1	0.0129	0.0123
64	L2	0.0221	0.0224
128	L1	0.0067	0.0064
128	L2	0.0112	0.0127
192	L1	0.0045	**0.0040**
192	L2	0.0079	0.0068

Table 2: **Ablation study** on the optimal configuration for training the OSI model. We trained for different combinations and show how each configuration influences the supervised residual objective in Eqn 3.

Training OSI is completely formulated as a supervised learning problem. We start with buffers of information collected during the universal policy rollout. Each buffer contains the recent history of state-action pairs. As our policy relies on a good OSI model, we carefully investigate different training strategies via hyperparameter tuning to identify the most optimal model for the complete pipeline. As shown in Table 2, we test for different batch sizes from 32 to 192, nearly increasing by a factor of 2, except for the last one, 192. As seen in the table, a large mini-batch size produced the least validation error. With a fixed learning rate, this result surprisingly opposes the small mini-batch size proposition made by [15] in his paper. For loss functions, we explored the effect of using an L1 or L2 loss, simply because our data was collected from rollouts which can be noisy. L1 Loss produced the least generalization error of **0.0040** - row 4, column 4, confirming its robustness to noisy samples than L2 loss which is more sensitive to outliers. Therefore, our OSI performed best with a batch size of 192, an L1 loss, and an epoch size of 500.

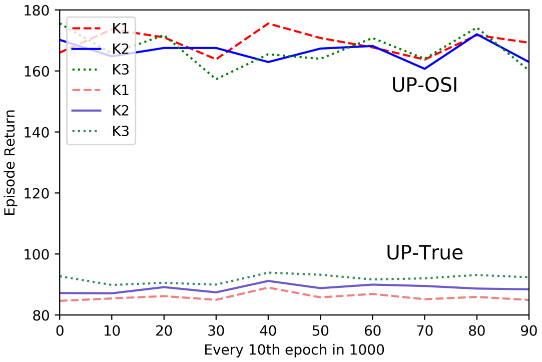

Figure 7: Significant improvements of UP-OSI across new testing environments.

5.3 Evaluating UP-OSI

To evaluate the joint UP-OSI pipeline, we test the algorithm in different environments outside the training range, called the testing range, as shown in Table 1. The ideal parameters for the cartpole environment fall within the training range which means the testing range is completely different from what either UP or OSI is trained on. We sample $K = 3$ environments uniformly within the testing range where each $K \in R^4$ is a set of environment parameters including gravity, cart mass, pole mass, and pole length. In each of these environments, we observe significant episode return for UP-OSI (blue, red, and green) compared to UP-True, seen in Figure 7. Alternatively for UP-True, the return shows a nearly flat curve (after the 50th epoch) indicating that the algorithm is unable to make new progress after a certain point in the new environment. In contrast, UP-OSI produces returns that show some indication of learning/understanding the new environment as the return curve does not lie flat (particularly for K1 and K2 in red and blue). Therefore, using OSI as a self-sustaining system identification model makes a huge difference between UP-True and UP-OSI for better understanding of new environments.

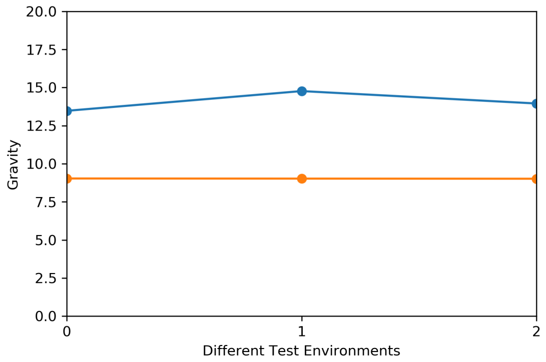

Figure 8: Limited training sub-space using Gravity as an example - Blue is the ground-truth parameter and orange is the prediction at environment K=0, K=1 and K=3.

6 Trial and Limitations

One major limitation we discovered is that the performance of our policy relies on a good OSI model. This means, if the OSI, for reasons unknown, is trained incorrectly or optimized on mis-aligned input data, the result could lead to a poor policy. To overcome this, future work could include training the UP-OSI algorithm all together without any isolated offline pre-training. Additionally, we also discovered that the OSI model can easily overfit a small sample of buffer data, thereby making almost the same predictions at inference time. This is possible if the parameters are sampled from some set of the training range (though this is unexpected). This sub-sampling could lead to making the same predictions, as shown in Figure 8. To overcome this limitation, one needs to sample large-enough training data that mostly covers the entire range, in order to encourage better generalization for the OSI model.

7 Conclusion and Future Work

In this project, we have demonstrated that it is possible to learn a universal policy in a self-sufficient manner for new dynamic environments. We achieve this by developing new modifications to the main idea and experimenting with different selection choices that work best for our environment.

Therefore, there are many directions and experiments that could be explored for future work. One particular extension could be looking at how to handle an unknown environment with multiple agents, each possibly requiring its own universal policies. Another possible extension could be trying a different policy update, such as SAC (Soft Actor-Critic), and see if this leads to potentially more robust universal policies. Finally, looking at how the algorithm may need to be changed to deal with dynamic environments is an interesting problem, since an agent may not always be in a static environment. In terms of possible experiments one could perform, looking at how far outside the training range one could vary the environment so that the universal policy is no longer useful (in other words, how robust can the universal policy be made) could be of interest as well. This will of course depend on the particular environment, but it is something worth exploring and comparing across different environments of varying complexity. As well, using the algorithm written in a more complicated environment other than cart-pole could have led to some interesting findings. In a more complicated environment, even a small change could possibly lead to a wildly different universal policy, and this could potentially be worth studying.

Conclusively, transfer learning from static to a completely dynamic environment is possible. With more ideas, research and a few papers down the line, this area of research could become standard in the RL community.

As a takeaway: This project provided a great opportunity for us to implement an existing idea from zero (which means not using their codebase). Moreso, it created an avenue for us to build on the work of others, and also think critically of how to extend what is possible. Thanks.

8 Appendix

See https://github.com/tianyuehz/533VProject for the link to our github project.

References

[1] Junjie Zeng, Rusheng Ju, Long Qin, Yue Hu, Quanjun Yin, and Cong Hu. Navigation in unknown dynamic environments based on deep reinforcement learning. *Sensors (Basel, Switzerland)*, 19, 2019.

[2] Ashley Peake, Joe McCalmon, Yixin Zhang, Daniel Myers, Sarra Alqahtani, and Paúl Pauca. Deep reinforcement learning for adaptive exploration of unknown environments. *CoRR*, abs/2105.01606, 2021.

[3] Libo Sun, Jinfeng Zhai, and Wenhu Qin. Crowd navigation in an unknown and dynamic environment based on deep reinforcement learning. *IEEE Access*, 7:109544–109554, 2019.

[4] Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification, 2017.

[5] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[7] Robert N. Boute, Joren Gijsbrechts, Willem van Jaarsveld, and Nathalie Vanvuchelen. Deep reinforcement learning for inventory control: A roadmap. *European Journal of Operational Research*, 2021.

[8] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2020.

[9] Tianpei Yang, Weixun Wang, Hongyao Tang, Jianye Hao, Zhaopeng Meng, Hangyu Mao, Dong Li, Wulong Liu, Chengwei Zhang, Yujing Hu, Yingfeng Chen, and Changjie Fan. An efficient transfer learning framework for multiagent reinforcement learning, 2021.

[10] Chaitanya Asawa, Christopher Elamri, and David Pan. Using transfer learning between games to improve deep reinforcement learning performance and stability. 2017.

[11] Evelyn Conceição Santos Batista, Wouter Caarls, Leonardo Forero, and Marco Aurélio Cavalcanti Pacheco. Training an agent to find and reach an object in different environments using visual reinforcement learning and transfer learning. In *ICAART*, 2021.

[12] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[13] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[14] Nazia Habib. Hands-on q-learning with python.

[15] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks, 2018.