ROBUST DOMAIN RANDOMIZATION FOR REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Producing agents that can generalize to a wide range of environments is a significant challenge in reinforcement learning. One method for overcoming this issue is domain randomization, whereby at the start of each training episode some parameters of the environment are randomized so that the agent is exposed to many possible variations. However, domain randomization is highly inefficient and may lead to policies with high variance across domains. In this work, we formalize the domain randomization problem, and show that minimizing the policy's Lipschitz constant with respect to the randomization parameters leads to low variance in the learned policies. We propose a method where the agent only needs to be trained on one variation of the environment, and its learned state representations are regularized during training to minimize this constant. We conduct experiments that demonstrate that our technique leads to more efficient and robust learning than standard domain randomization, while achieving equal generalization scores.

1 INTRODUCTION

Deep Reinforcement Learning (RL) has proven very successful on complex high-dimensional problems ranging from games like Go (Silver et al., 2017) and Atari games (Mnih et al., 2015) to robot control tasks (Levine et al., 2016). However, one prominent issue is that of overfitting, illustrated in figure 1: agents trained on one domain fail to generalize to other domains that differ only in small ways from the original domain (Sutton, 1996; Cobbe et al., 2018; Zhang et al., 2018b; Packer et al., 2018; Zhang et al., 2018a; Witty et al., 2018; Farebrother et al., 2018). Good generalization is essential for problems such as robotics and autonomous vehicles, where the agent is often trained in a simulator and is then deployed in the real world where novel conditions will certainly be encountered. Transfer from such simulated training environments to the real world is known as crossing the *reality gap* in robotics, and is well known to be difficult, thus providing an important motivation for studying generalization.

To close the reality gap in reinforcement learning, prior work has studied both *domain adaptation* and *domain randomization*. Domain adaptation techniques aim to update the data distribution in simulation to match the real distribution through some form of canonical mapping or using regularization methods (James et al., 2018; Bousmalis et al., 2017; Gamrian & Goldberg, 2018). Alternatively, domain randomization (DR), in which the visual and physical properties of the training domains are randomized at the start of each episode during training, has also been shown to lead to improved generalization and transfer to the real world (Tobin et al., 2017; Sadeghi & Levine, 2016; Antonova et al., 2017; Peng et al., 2017; Mordatch et al., 2015; Rajeswaran et al., 2016; OpenAI, 2018). Domain randomization relies on the expectation that the agent will perceive the difference between the train and test domains as just another variation of the train domain. However, domain randomization has been empirically shown to often lead to suboptimal policies with high variance in performance over different randomizations (Mehta et al., 2019). This issue can cause the learned policy to underperform in any given target domain.

We propose a method for learning policies that are robust to changes in the randomization space, producing agents that ignore irrelevant aspects of the environment. Our work combines aspects from both domain adaptation and domain randomization, in that we maintain the notion of randomized environments but use a regularization method to achieve good generalization over the randomization space. Our contributions are the following:

We formalize the domain randomization problem, and show that the Lipschitz constant of the agent's policy over the randomization parameters provides an upper bound on the agent's robustness to variations in the environment.

We propose an algorithm whereby the agent is only trained on one variation of the environment but its learned representations are regularized so that the Lipschitz constant is minimized.

We experimentally show that our method is more efficient and leads to lower-variance policies than standard domain randomization, while achieving equal or better returns and generalization ability.

This paper is structured as follows. We first review other work related to ours, formalize the domain randomization problem, and present our theory contributions. We then describe our regularization method, and illustrate its application to a toy gridworld problem. Finally, we compare our method with standard domain randomization in complex visual environments.

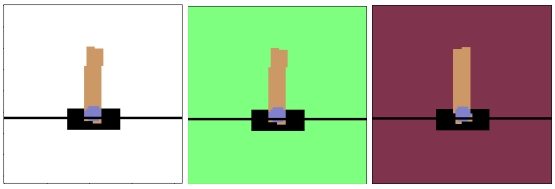


Figure 1: Illustration of the generalization challenge in reinforcement learning. In this visual cartpole domain, the agent must learn to keep the pole upright. However, changes in the background color can completely throw off a trained agent.

2 Related Work

2.1 Generalization in Deep Reinforcement Learning

Generalization to novel samples is well studied in supervised learning, where evaluating generalization through train/test splits is ubiquitous. However, evaluating for generalization to novel conditions through such train/test splits is not common practice in Deep RL. Zhang et al. (2018b) study overfitting of Deep RL in discrete maze tasks. Testing environments are generated with the same maze configuration but different initial positions from training. Deep RL algorithms are shown to suffer from overfitting to training configurations and to memorize training scenarios. Packer et al. (2018) study performance under train-test domain shift by modifying environmental parameters such as robot mass and length to generate new domains. Farebrother et al. (2018) propose using different game modes of Atari 2600 games to measure generalization. They turn to supervised learning for inspiration, finding that both L2 regularization and dropout can help agents learn more generalizable features. These works all show that standard Deep RL algorithms tend to overfit to the environment used during training, hence the urgent need for designing agents that can generalize better. Domain randomized training has been shown to be a promising way of addressing this challenge.

2.2 Domain Randomization

We distinguish between two types of domain randomization: visual randomization, in which the variability between domains should not affect the agent's policy, and dynamics randomization, in which the agent should learn to adjust its behavior to achieve its goal. Visual domain randomization has been successfully used to directly transfer RL agents from simulation to the real world without requiring any real images (Tobin et al., 2017; Sadeghi & Levine, 2016; Kang et al., 2019). These approaches used low fidelity rendering and randomized scene properties such as lighting, textures, camera position, and colors, which led to improved generalization. Dynamics randomization has been successfully used to develop agents that are more robust to uncertainty in the system's dynamics (Antonova et al., 2017; Peng et al., 2017; Mordatch et al., 2015; Rajeswaran et al., 2016; OpenAI, 2018). In this paper, we focus on the visual domain randomization setting.

The work most reminiscent to our proposed method combine domain randomization and domain adaptation techniques (James et al., 2018; Chebotar et al., 2018; Gamrian & Goldberg, 2018). The main idea of these approaches is to both randomize the simulated environment and penalize the gap between the trajectories in the simulations and the real world, either by adding a term to the loss, or learning a mapping between the states of the simulation and the real world. This approach requires a large number of samples of real world trajectories, which can be expensive to collect.

Prior work has, however, noted the inefficiency of Domain Randomization. Mehta et al. (2019) show that domain randomization may lead to suboptimal policies that vary a lot between domains, due to uniform sampling of the environment's parameters. They propose a method to guide domain randomization, by predicting the most informative environment variations within the given randomization ranges. Zakharov et al. (2019) also guide the domain randomization procedure by training a *DeceptionNet*, that learns which randomizations are actually useful to bridge the domain gap for image classification tasks.

2.3 Learning Domain-Invariant Features and Domain Adaptation

Learning domain-invariant features has emerged as a promising approach of taking advantage of the commonalities between domains. This is usually done by minimizing some measure of distance between the source and target domains. In the semi-supervised learning context, there is a large body of work relating to learning domain-invariant features. For instance, Bachman et al. (2014); Sajjadi et al. (2016); Coors et al. (2018); Miyato et al. (2018); Xie et al. (2019) enforce that predictions of their networks be similar for original and augmented data points, with the objective of reducing the required amount of labelled data for training. Our work extends such methods to reinforcement learning.

In the reinforcement learning context, several other papers have also explored this topic. Tzeng et al. (2015) and Gupta et al. (2017) add constraints to encourage networks to learn similar embeddings for samples from both a simulated and a target domain. Daftry et al. (2016) apply a similar approach to transfer policies for controlling aerial vehicles to different environments. Bousmalis et al. (2017) compare different domain adaptation methods in a robot grasping task, and show that they improve generalization. Wulfmeier et al. (2017) use an adversarial loss to train RL agents in such a way that similar policies are learned in both a simulated domain and the target domain. While promising, these methods are designed for cases when simulated and target domains are both known, and cannot straightforwardly be applied when the target domain is only known to be within a distribution of domains.

3 Problem Formulation

We consider Markov decision processes (MDP) defined by $(S; A; R; T; \ )$, where S is the state space, A the action space, $R : S \ A \ ! \ \mathbb{R}$ the reward function, $T : S \ A \ ! \ Pr(S)$ the transition dynamics, and the discount factor. In reinforcement learning, an agent's objective is to find a policy that maps states to distributions over actions such that the cumulative discounted reward yielded by its interactions with the environment is optimized.

3.1 Domain Randomization

Domain randomization requires a set of N simulation parameters to randomize, and a *randomization space* $\mathbb{R}^N$ from which these parameters are sampled. When a configuration 2 is passed to a simulator, it generates a new MDP M with a potentially new set of states and transitions indexed by . At the start of each episode, the parameters are sampled from the chosen randomization space, and the generated MDP is used to train the agent during this episode. Denoting $J(\ ; \ )$ the cumulative returns of a policy , the goal is thus to solve the optimization problem defined by $J(\ ) = \max \ \mathbb{E} \ [J(\ ; \ )]$. When the randomization parameters affect the transitions in the MDP, we refer to dynamics randomization. When the randomization parameters affect only the states, we refer to visual randomization.

Domain randomization empirically produce policies with strongly varying performance over different regions of the randomization space, as demonstrated by Mehta et al. (2019) for the case of

dynamics randomization. Our own experiments, which we discuss later in this paper, also corroborate this observation for visual randomization. This high variance can cause the learned policy to underperform in any given target domain.

To yield insight into the robustness of policies learned by domain randomization, we start by formalizing the notion of a randomized MDP. Although domain randomization may perturb any element of the underlying MDP such as rewards or transitions, in this work we only consider the case where we modify the state space S. This is often used to close the *visual gap* between simulation and reality, for example by randomizing colors or textures during the training in simulation. Contrary to dynamics randomization, such randomizations don't change the transition or reward functions of the underlying MDP.

Definition 1 *Let $M = (S; A; R; T;)$ be an MDP. A randomizer function of M is a mapping $: S \to S'$ where S' is a new set of states. The Randomized MDP $M = (S ; A ; R ; T ;)$ is defined as, for $s; s' \in S, a \in A$:*

$$S = (S); \quad A = A; \quad T((s')j (s); a) = T(s'js; a); \quad R((s); a) = R(s; a); \quad =$$

Given a policy on MDP M and a randomization M , we also define the agent's policy on M as $(js) = (j (s))$.

Despite all randomized MDPs sharing the same underlying rewards and transitions, the agent's policy can vary between domains. For example, in policy-based algorithms (Williams, 1992), if there are several optimal policies then the agent may adopt different policies for different . Furthermore, for value-based algorithms such as DQN (Mnih et al., 2015), two scenarios can lead to there being different policies for different . First, the (unique) optimal Q-function may correspond to several possible policies. Second, imperfect function approximation can lead to different value estimates for different randomizations and thus to different policies. To compare the ways in which policies can differ between randomized domains, we introduce the notion of Lipschitz continuity of a policy over a set of randomizations.

Definition 2 *We assume the state space is equipped with a distance metric. A policy is Lipschitz continuous over a set of randomizations $f g$ if for all randomizations $_1$ and $_2$ in $f g$,*

$$K = \sup_{_1; _2 \in f g} \sup_{s \in S} \frac{D_{TV}((j _1(s)) k (j _2(s)))}{j _1(s) \quad _2(s)j}$$

is finite. Here, $D_{TV}(P kQ)$ is the total variation distance between distributions (given by $\frac{1}{2} \sum_{a \in A} jP(a) \quad Q(a)j$ when the action space is discrete).

The following inequality shows that this Lipschitz constant is crucial in quantifying the robustness of RL agents over a randomization space. The smaller the Lipschitz constant, the less a policy is affected by different randomization parameters. Informally, if a policy is Lipschitz continuous over randomized MDPs, then in the visual domain randomization context this implies for example that small changes in the background color in an environment will have a small impact on the policy.

Proposition 1 *We consider an MDP M and a set of randomizations $f g$ of this MDP. Let be a K-Lipschitz policy over $f g$. Suppose the rewards are bounded by $r_{\max}$ such that $\forall a \in A; s \in S; jr(s; a)j \quad r_{\max}$. Then for all $_1$ and $_2$ in $f g$, the following inequalities hold :*

$$j _1 \quad _2j \quad 2r_{\max} \sum_t {}^t \min(1; (t+1)K \quad k _1 \quad _2k_1) \quad \frac{2r_{\max}K}{(1 \quad)^2} k _1 \quad _2k_1 \qquad (1)$$

Where $_i$ is the expected cumulative return of policy ^{i} on MDP M_i, for $i \in f1; 2g$, and $j _1 \quad _2k_1 = \sup_{s \in S} j _1(s) \quad _2(s)j$.

Proof. See appendix.

These inequalities shows that the smaller the Lipschitz constant, the smaller the maximum variations of the policy over the randomization space can be. In the following, we present a regularization technique that produces low-variance policies over the randomization space by minimizing the Lipschitz constant of the policy.

4 PROPOSED REGULARIZATION

We propose a simple regularization method to produce an agent with policies that vary little over randomized environments, despite being trained on only one environment. We start by choosing one variation of the environment on which to train an agent with a policy parameterized by , and during training we minimize the loss

$$L( ) = L_{RL}( ) + \quad \mathop{\mathsf{E}}_{s} \mathsf{E}\, kf\ (s) \quad f\ (\ (s))k_2^2 \tag{2}$$

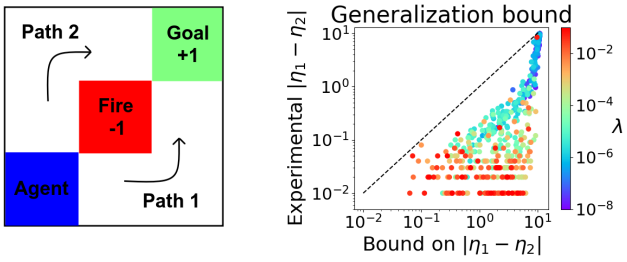
where is a regularization parameter, L_{RL} is the loss corresponding to the chosen reinforcement learning algorithm, the first expectation is taken over the distribution of states visited by the current policy which we assume to be fixed when optimizing this loss, and f is a *feature-extractor* used by the agent's policy. In our experiments, we choose the output of the last hidden layer of the value or policy network as our feature extractor; we note that other choices could also be made. Minimizing the second term in this loss function minimizes the Lipschitz constant as defined above over the states visited by the agent, and causes the agent to learn representations of states that ignore variations caused by the randomization.

Our method can be applied to many RL algorithms, since it involves simply adding an additional term to the learning loss. In the following, we experimentally demonstrate applications to both value-based and policy-based reinforcement learning algorithms. Implementation details can be found in the appendix, and the code will be made available online.

5 EXPERIMENTS

5.1 ILLUSTRATION ON A GRIDWORLD

We first conduct experiments on a simple gridworld to illustrate the theory described above.



Agent	Same path probability
Randomized	86%
Regularized (ours)	**100%**

Figure 2: Left: a simple gridworld, in which the agent must make its way to the goal while avoiding the fire. Center: empirical differences between regularized agents' policies on two randomizations of the gridworld compared to our theoretical bound (the dashed line), shown for 20 training seeds per value of . Right: probability that different agents will choose the same path for different randomizations of this domain. Our regularization method leads to more consistent behavior.

The environment we use is the 3 3 gridworld shown in figure 2, in which two optimal policies exist. The agent starts in the bottom left of the grid and must reach the goal while avoiding the fire. The agent can move either up or right, and in addition to the rewards shown in figure 2 receives -1 reward for invalid actions that would case it to leave the grid. We set a time limit of 10 steps and = 1. We introduce randomization into this environment by describing the state of the agent as a tuple $(x; y; \ )$, where $(x; y)$ is the agent's position and is a randomization parameter with no impact on the underlying MDP. For this toy problem, we consider only two possible values for : +5 and 5. The agents we consider use the REINFORCE algorithm (Sutton et al., 2000) with a baseline (see appendix), and a multi-layer perceptron as the policy network.

First, we observe that even in a simple environment such as this one, a randomized agent regularly learns different paths for different randomizations (figure 2). An agent trained only on = 5 and regularized with our technique, however, consistently learns the same path regardless of . Although both agents easily solve the problem, the variance of the randomized agent's policy can be

problematic in more complex environments in which identifying similarities between domains and ignoring irrelevant differences is important.

Next, we compare the empirical difference between the policies learned by regularized agents on the two domains to the smallest of our theoretical bounds in equation 1, which in this simple environment can be directly calculated. Our results for different values of are shown in gure 2. We observe that increasing does lead to decreases in both the empirical difference in returns and in the theoretical bound.

5.2 VISUAL CARTPOLE WITH DQN

We compare standard domain randomization to our regularization method on a more challenging visual environment, in terms of 1) training stability, 2) returns and variance of the learned policies, and 3) state representations learned by the agents.

5.2.1 EXPERIMENTAL SETTING

To run domain randomization experiments, we use a visual Cartpole environment shown in gure 1, where the states consist of raw pixels of the images. The agent must keep a pole upright as long as possible on a cart that can move left or right. The episode terminates either after 200 time steps, if the cart leaves the track, or if the pole falls over. The randomization consists of changing the color of the background. Each randomized domain 2 corresponds to a color $(r; g; b)$, where $0 \quad r; g; b \quad 1$. Our implementation of this environment is based on the OpenAI Gym (Brockman et al., 2016).

For training, we use the DQN algorithm with a CNN architecture similar to that used by Mnih et al. (2015). In principle, such a value-based algorithm should learn a unique value function independently of the randomization parameters we consider. This is in contrast to the policy-based algorithm used in our previous experiment, in which there is no unique optimal policy. However, as we will show function approximation errors cause different value functions to be learned for different background colors.

We compare the performance of three agents. The Normal agent is trained on only one domain (with a white background). The Randomized agent is trained on a chosen randomization space. The Regularized agent is trained on a white background using our regularization method with respect to randomization space. The training of all three agents is done using the same hyperparameters, and over the same number of steps.

5.2.2 PERFORMANCE DURING TRAINING

Figure 3: Training curves over randomization spaces $_{\text{small}}$ (left) and $_{\text{big}}$ (right). Shaded areas indicate the 95% con dence interval of the mean, obtained over 10 training seeds.

We rst compare the performance of our agents during training. We train all three agents over two randomization spaces (environments with different background colors), having the following sizes :

$_{\text{small}} = f(r; g; b); 0{:}5 \quad r; g; b \quad 1{:}g = [0{:}5; 1]^3$: $\frac{1}{8}$ of the unit cube.

$_{\text{big}} = [0; 1] \quad [0{:}5; 1] \quad [0; 1]$: half the unit cube.

We obtain the training curves shown in gure 3. We nd that the normal and regularized agents have similar training curves and are not affected by the size of the randomization space. However,

the randomized agent learns more slowly on the small randomization space Ω_{small} (left), and also achieves worse performance on the bigger randomization space Ω_{big} (right). In high-dimensional problems, we would like to pick the randomization space to be as large as possible to increase the chances of transferring to the target domain. We find that standard domain randomization scales poorly with the size of the randomization space, whereas our regularization method is more robust to a larger randomization space.

5.2.3 GENERALIZATION AND VARIANCE

Figure 4: Comparison of the average scores of different agents over different domains. The scores are calculated over a plane of the (r,g,b) cube in Ω_{big}, where g = 1 is fixed, averaged over 1000 steps. The training domain for both the regularized and normal agents is located at the top right. The regularized agent learns more stable policies than the randomized agent over these domains.

We compare the returns of the policies learned by the different agents in different domains within the randomization space. We select a plane within Ω_{big} obtained by varying only the R and B channels but keeping G fixed. We plot the scores obtained on this plane in figure 4. We see that despite having only been trained on one domain, the regularized agent achieves consistently high scores on the other domains. On the other hand, the randomized agent's policy exhibits returns with high variance between domains, which indicates that different policies were learned for different domains. We also observe that the regularized agent has much smaller variance and generally higher scores that the randomized agent.

5.2.4 REPRESENTATIONS LEARNED BY THE AGENTS

Agent	Standard Deviations
Normal	10.1
Randomized	6.2
Regularized (ours)	3.7

Figure 5: Left: Visualization of the representations learned by the agents for pink and green background colors and for the same set of states. We observe that the randomized agent learns different representations for the two domains. Right: Standard deviation of estimated value functions over randomized domains, averaged over 10 training seeds.

To understand what causes this difference in behavior between the two agents, we study the representations learned by the agents by analyzing the activations of the final hidden layer. We consider the agents trained on Ω_{big}, and a sample of states obtained by performing a greedy rollout on a white background (which is included in Ω_{big}). For each of these states, we calculate the representation corresponding to that state for another background color in Ω_{big}. We then visualize these representations using t-SNE plots, where each color corresponds to a domain. A representative example of such a plot is shown in figure 5. We see that the regularized agent learns a similar representation for both backgrounds, whereas the randomized agent clearly separates them. This result indicates that the regularized agent learns to ignore the background color, whereas the randomized agent is

likely to learn a different policy for a different background color. Further experiments comparing the representations of both agents can be found in the appendix.

To further study the effect of our regularization method on the representations learned by the agents, we compare the variations in the estimated value function for both agents over $\mathcal{U}_{bg}$. Figure 5 shows the standard deviation of the estimated value function over different background colors, averaged over 10 training seeds and a sample of states obtained by the same procedure as described above. We observe that our regularization technique successfully reduces the variance of the value function over the randomization domain. This can be seen as a consequence of the fact that the representations learned by the agent vary less between domains, and also explains the lower variance of agents trained with our method.

5.3 CAR RACING WITH PPO

Figure 6: Left: frames from the original and randomized CarRacing environment. Right: training curves of our agents, averaged over 5 seeds. Shaded areas indicate the 95% con dence interval of the mean.

To demonstrate the applicability of our regularization method to other domains and algorithms, we also perform experiments with the PPO algorithm (Schulman et al., 2017) on the CarRacing environment (Brockman et al., 2016), in which an agent must drive a car around a racetrack. An example state from this environment and a randomized version in which part of the background changes color are shown in gure 6. We train 4 agents on this domain: a normal agent on the original background, a randomized agent, and two regularized agents for two different values of Randomization in this experiment occurs over the entire RGB cube.

Agent	Return (original)	Return (all colors)
Normal	554 68	60 53
Regularized = 10	622 81	324 51
Regularized = 50	640 40	553 80

Table 1: Average returns on the original environment and its randomizations over all colors, with 95% con dence intervals calculated from 5 training seeds.

Training curves are shown in gure 6. Training curves for both regularized agents are very similar, so only the curve for = 50 is shown here. We see that the randomized agent fails to learn a successful policy, whereas the other agents successfully learn. We also compare the generalization ability of the other agents to other background colors, with our results shown in table 1. These results con rm that our regularization leads to agents that are both successful in training and successfully generalize to a wide range of backgrounds. Moreover, a larger value of yields higher generalization scores.

6 CONCLUSION

In this paper we studied domain randomization in deep reinforcement learning. We formalized the problem, illustrated the inef ciencies of standard domain randomization, and proposed a theoretically grounded method that leads to robust, low-variance policies that are domain invariant. We conducted several experiments in different environments of differing complexities using both on-policy and off-policy algorithms to support our claims.

REFERENCES

Martin Anthony and Peter L Bartlett. Neural network learning: Theoretical foundations. cambridge university press, 2009.

Rika Antonova, Silvia Cruciani, Christian Smith, and Danica Kragic. Reinforcement learning for pivoting task. CoRR, abs/1703.00472, 2017. URL http://arxiv.org/abs/1703.00472.

Philip Bachman, Ouais Alsharif, and Doina Precup. Learning with pseudo-ensembles. In Advances in Neural Information Processing Systems, pp. 3365–3373, 2014.

Peter L Bartlett. For valid generalization the size of the weights is more important than the size of the network. In Advances in neural information processing systems, pp. 134–140, 1997.

Peter L. Bartlett, Dylan J. Foster, and Matus Telgarsky. Spectrally-normalized margin bounds for neural networks. CoRR, abs/1706.08498, 2017. URL http://arxiv.org/abs/1706.08498.

Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, Sergey Levine, and Vincent Vanhoucke. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. CoRR, abs/1709.07857, 2017. URL http://arxiv.org/abs/1709.07857.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan D. Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. CoRR, abs/1810.05687, 2018. URL http://arxiv.org/abs/1810.05687.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. arXiv preprint arXiv:1812.02341, 2018.

Benjamin Coors, Alexandru Condurache, Alfred Mertins, and Andreas Geiger. Learning transformation invariant representations with weak supervision. In VISIGRAPP (5: VISAPP), pp. 64–72, 2018.

Shreyansh Daftry, J. Andrew Bagnell, and Martial Hebert. Learning transferable policies for monocular reactive MAV control. CoRR, abs/1608.00627, 2016. URL http://arxiv.org/abs/1608.00627.

Jesse Farebrother, Marlos C. Machado, and Michael Bowling. Generalization and regularization in DQN. CoRR, abs/1810.00123, 2018. URL http://arxiv.org/abs/1810.00123.

Shani Gamrian and Yoav Goldberg. Transfer learning for related reinforcement learning tasks via image-to-image translation. CoRR, abs/1806.07377, 2018. URL http://arxiv.org/abs/1806.07377.

Abhishek Gupta, Coline Devin, Yuxuan Liu, Pieter Abbeel, and Sergey Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. CoRR, abs/1703.02949, 2017. URL http://arxiv.org/abs/1703.02949.

Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. CoRR, abs/1812.07252, 2018. URL http://arxiv.org/abs/1812.07252.

Katie Kang, Suneel Belkhale, Gregory Kahn, Pieter Abbeel, and Sergey Levine. Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight. CoRR, abs/1902.03701, 2019. URL http://arxiv.org/abs/1902.03701.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. The Journal of Machine Learning Research, 17(1):1334–1373, 2016.

Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J. Pal, and Liam Paull. Active domain randomization. *CoRR*, abs/1904.04762, 2019. URL http://arxiv.org/abs/1904.04762.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2018.

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.

Igor Mordatch, Kendall Lowrey, and Emanuel Todorov. Ensemble-CIO: Full-body dynamic motion planning that transfers to physical humanoids. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5307–5314, 2015.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Conference on Learning Theory*, pp. 1376–1401, 2015.

Adam M. Oberman and Jeff Calder. Lipschitz regularized deep neural networks converge and generalize. *CoRR*, abs/1808.09540, 2018. URL http://arxiv.org/abs/1808.09540.

OpenAI. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018. URL http://arxiv.org/abs/1808.00177.

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *CoRR*, abs/1810.12282, 2018. URL http://arxiv.org/abs/1810.12282.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *CoRR*, abs/1710.06537, 2017. URL http://arxiv.org/abs/1710.06537.

Aravind Rajeswaran, Sarvjeet Ghotra, Sergey Levine, and Balaraman Ravindran. Epopt: Learning robust neural network policies using model ensembles. *CoRR*, abs/1610.01283, 2016. URL http://arxiv.org/abs/1610.01283.

Fereshteh Sadeghi and Sergey Levine. $(\text{CAD})^2\text{RL}$: Real single-image flight without a single real image. *CoRR*, abs/1611.04201, 2016. URL http://arxiv.org/abs/1611.04201.

Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *Advances in Neural Information Processing Systems*, pp. 1163–1171, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pp. 1038–1044, 1996.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.

Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907, 2017. URL http://arxiv.org/abs/1703.06907.

Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Xingchao Peng, Sergey Levine, Kate Saenko, and Trevor Darrell. Towards adapting deep visuomotor representations from simulated to real environments. *CoRR*, abs/1511.07111, 2015. URL http://arxiv.org/abs/1511.07111.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Sam Witty, Jun Ki Lee, Emma Tosch, Akanksha Atrey, Michael L. Littman, and David Jensen. Measuring and characterizing generalization in deep reinforcement learning. *CoRR*, abs/1812.02868, 2018. URL http://arxiv.org/abs/1812.02868.

Markus Wulfmeier, Ingmar Posner, and Pieter Abbeel. Mutual alignment transfer learning. *CoRR*, abs/1707.07907, 2017. URL http://arxiv.org/abs/1707.07907.

Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Unsupervised data augmentation. *arXiv preprint arXiv:1904.12848*, 2019.

Sergey Zakharov, Wadim Kehl, and Slobodan Ilic. Deceptionnet: Network-driven domain randomization. *CoRR*, abs/1904.02750, 2019. URL http://arxiv.org/abs/1904.02750.

Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *CoRR*, abs/1806.07937, 2018a. URL http://arxiv.org/abs/1806.07937.

Chiyuan Zhang, Oriol Vinyals, Rémi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *CoRR*, abs/1804.06893, 2018b. URL http://arxiv.org/abs/1804.06893.

A PROOF OF PROPOSITION 1

The proof presented in the following applies to MDPs with a discrete action space. However, it can straightforwardly be generalized to continuous action spaces by replacing sums over actions with integrals over actions.

The proof uses the following lemma :

Lemma 1 *For two distributions $p(x;y) = p(x)p(y|x)$ and $q(x;y) = q(x)q(y|x)$, we can bound the total variation distance of the joint distribution :*

$$D_{TV}(p(\,;\,)\|q(\,;\,)) \quad D_{TV}(p(\,)\|q(\,)) + \max_x D_{TV}(p(\,|x)\|q(\,|x))$$

Proof of the Lemma.

We have that :

$$\begin{aligned}
D_{TV}(p(\,;\,)\|q(\,;\,)) &= \frac{1}{2}\sum_{x;y} |p(x;y) \quad q(x;y)| \\
&= \frac{1}{2}\sum_{x;y} |p(x)p(y|x) \quad q(x)q(y|x)| \\
&= \frac{1}{2}\sum_{x;y} |p(x)p(y|x) \quad p(x)q(y|x) \\
&\quad + (p(x) \quad q(x))q(y|x)| \\
&\quad \frac{1}{2}\sum_{x;y} p(x)|p(y|x) \quad q(y|x)| \\
&\quad + |p(x) \quad q(x)|q(y|x) \\
&\quad \max_x D_{TV}(p(\,|x)\|q(\,|x)) \\
&\quad + D_{TV}(p(\,)\|q(\,))
\end{aligned}$$

Proof of the proposition.

Let $p^t{}_i(s;a)$ be the probability of being in state $\ _i(s)$ at time t, and executing action a, for $i = 1$ or 2. Since both MDPs have the same reward function, we have by definition that $\ _i = \sum_{s;a\ \ t} \ ^t p^t{}_i(s;a) r_t(s;a)$, so we can write :

$$\begin{aligned}
|\ _1 \quad \ _2| &\sum_{s;a\ \ t} \ ^t |p^t{}_1(s;a) \quad p^t{}_2(s;a)| r_t(s;a) \\
&r_{\max} \sum_{s;a\ \ t} \ ^t |p^t{}_1(s;a) \quad p^t{}_2(s;a)| \\
&= 2r_{\max} \sum_t \ ^t D_{TV}(p^t{}_1(\,;\,)\|p^t{}_2(\,;\,)) \qquad (3)
\end{aligned}$$

But $p^t{}_1(s;a) = p^t{}_1(s) \ \ ^1(a|s)$ and $p^t{}_2(s;a) = p^t{}_2(s) \ \ ^2(a|s)$, Thus (Lemma 1) :

$$\begin{aligned}
D_{TV}(p^t{}_1(\,;\,)\|p^t{}_2(\,;\,)) &\quad D_{TV}(p^t{}_1(\,)\|p^t{}_2(\,)) \\
&\quad + \max_s D_{TV}(\ ^1(\,|s)\| \ ^2(\,|s)) \\
&\quad D_{TV}(p^t{}_1(\,)\|p^t{}_2(\,)) \\
&\quad + K \ k \ _1 \quad \ _2 k_1 \qquad (4)
\end{aligned}$$

We still have to bound $D_{TV}(p^t{}_1(\,)\|p^t{}_2(\,))$. For $s \in S$ we have that :

$$\begin{aligned}
|p^t{}_1(s) - p^t{}_2(s)| &\leq \sum_{s^0} |p_{\pi_1}(s_t = s|s^0)p^t{}_1(s^0) \\
&\quad p_{\pi_2}(s_t = s|s^0)p^t{}_2(s^0)| \\
&= \sum_{s^0} |p_{\pi_1}(s_t = s|s^0)p^t{}_1(s^0) \\
&\quad p_{\pi_2}(s_t = s|s^0)p^t{}_1(s^0) \\
&+ p_{\pi_2}(s_t = s|s^0)p^t{}_1(s^0) \\
&\quad p_{\pi_2}(s_t = s|s^0)p^t{}_2(s^0)| \\
&\leq \sum_{s^0} p^t{}_1(s^0)|p_{\pi_1}(s|s^0) - p_{\pi_2}(s|s^0)| \\
&+ p_{\pi_2}(s|s^0)|p^t{}_1(s^0) - p^t{}_2(s^0)|
\end{aligned}$$

Summing over s we have that

$$\begin{aligned}
D_{TV}(p^t{}_1(\,)\|p^t{}_2(\,)) &\leq \frac{1}{2}\sum_s \mathbb{E}_{s^0 \sim p^t{}_1}[|p_{\pi_1}(s|s^0) \\
&\quad p_{\pi_2}(s|s^0)|] \\
&+ D_{TV}(p^t{}_1{}_1(\,)\|p^t{}_2{}_1(\,))
\end{aligned}$$

But by marginalizing over actions : $p_{\pi_1}(s|s^0) = \sum_a \pi_1(a|s^0)p_{\pi_1}(s|a;s^0)$, and using the fact that $p_{\pi_1}(s|a;s^0) = T_{\pi_1}(s|a;s^0) = T_{\pi_2}(s|a;s^0) = p_{\pi_2}(s|a;s^0) := p(s|a;s^0)$, we have that

$$\begin{aligned}
|p_{\pi_1}(s|s^0) - p_{\pi_2}(s|s^0)| &= |\sum_a p(s|a;s^0)(\pi_1(a|s^0) \\
&\quad \pi_2(a|s^0))| \\
&\leq \sum_a p(s|a;s^0)|\pi_1(a|s^0) \\
&\quad \pi_2(a|s^0)|
\end{aligned}$$

And using $\sum_s p(s|a;s^0) = 1$ we have that :

$$\begin{aligned}
&\frac{1}{2}\sum_s \mathbb{E}_{s^0 \sim p^t{}_1}[|p_{\pi_1}(s|s^0) - p_{\pi_2}(s|s^0)|] \\
&\leq \frac{1}{2}\mathbb{E}_{s^0 \sim p^t{}_1}\sum_a [\sum_s p(s|a;s^0)]|\pi_1(a|s^0) - \pi_2(a|s^0)| \\
&\leq \max_{s^0} D_{TV}(\pi_1(\cdot|s)\|\pi_2(\cdot|s)) \\
&\leq K\|\pi_1 - \pi_2\|_1
\end{aligned}$$

Thus, by induction, and assuming $D_{TV}(p^0{}_1(\,)\|p^0{}_2(\,)) = 0$:

$$D_{TV}(p^t{}_1(\,)\|p^t{}_2(\,)) \leq tK\|\pi_1 - \pi_2\|_1$$

Plugging this into inequality 4, we get

$$D_{TV}(p^t{}_1(\,;\,)\|p^t{}_2(\,;\,)) \leq (t+1)K\|\pi_1 - \pi_2\|_1$$

We also note that the total variation distance takes values between 0 and 1, so we have

$$D_{TV}(p^t{}_1(\,;\,)\|p^t{}_2(\,;\,)) \leq \min(1; (t+1)K\|\pi_1 - \pi_2\|_1)$$

Plugging this into inequality 3 leads to our first bound,

$$|\eta_1 - \eta_2| \leq 2r_{\max}\sum_t \gamma^t \min(1; (t+1)K\|\pi_1 - \pi_2\|_1)$$

Our second, looser bound can now be achieved as follows,

$$j \quad {}_1 \quad {}_2 j \quad 2r_{\max} \sum_t {}^t(t+1)K \quad k \quad {}_1 \quad {}_2 k_1$$

$$j \quad {}_1 \quad {}_2 j \quad \frac{2r_{\max}}{(1 \quad)^2} K \quad k \quad {}_1 \quad {}_2 k_1$$

B EXPERIMENTAL DETAILS

All code used for our experiments will be made available online.

B.1 STATE PREPROCESSING

For our implementation of the visual cartpole environment, each image consists of 84 84 pixels with RGB channels. To include momentum information in our state description, we stack $k = 3$ frames, so the shape of the state that is sent to the agent is 84 84 9.

In CarRacing, each state consists of 96 96 pixels with RGB channels. We introduce frame skipping as is often done for Atari games (Mnih et al. (2015)), with a skip parameter of 5. This restricts the length of an episode to 200 action choices. We then stack 2 frames to include momentum information into the state description. The shape of the state that is sent to the agent is thus 96 96 6.

B.2 VISUAL CARTPOLE

B.2.1 EXTRAPOLATION

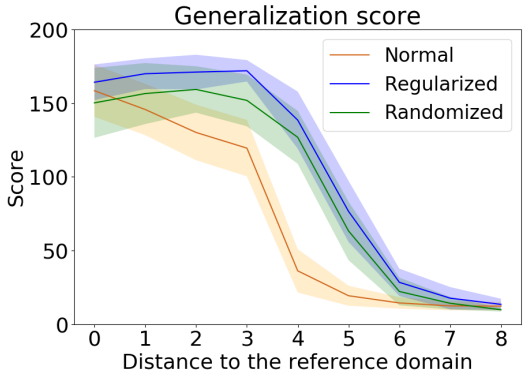


Figure 7: Generalization scores, with 95% confidence intervals obtained over 10 training seeds. The normal agent is trained on white $(1; 1; 1)$, corresponding to a *distance to train* $= 0$. The rest of the domains correspond to $(x; x; x)$, for $x = 0{:}9; 0{:}8; : : : ; 0{:}$

Given that regularized agents are stronger in interpolation over their training domain, it is natural to wonder what the performance of these agents is in extrapolation to colors not within the range of colors sampled within training. For this purpose, we consider randomized and regularized agents trained on $_{big}$, and test them on the set $f(x; x; x); 0 \quad x \quad 1g$. None of these agents was ever exposed to $x \quad 0{:}5$ during training.

Our results are shown in figure 7. We find that although the regularized agent consistently outperforms the randomized agent in interpolation, both agents fail to extrapolate well outside the train domain. Since we only regularize with respect to the training space, there is indeed no guarantee that our regularization method can produce an agent that extrapolates well. Since the objective of domain randomization often is to achieve good transfer to an a priori unknown target domain, this result suggests that it is important that the target domain lie within the randomization space, and that the randomization space be made as large as possible during training.

B.2.2 FURTHER STUDY OF THE REPRESENTATIONS LEARNED BY DIFFERENT AGENTS

We perform further experiments to demonstrate that the randomized agent learns different representations for different domains, whereas the regularized agent learns similar representations. We consider agents trained on $_{split} = [0; 0.2]^3 \; [0.8; 1]^3$, the union of *darker*, and *lighter* backgrounds. We then rollout each agent on a single episode of the domain with a white background and, for each state in this episode, calculate the representations learned by the agent for other background colors. We visualize these representations using the t-SNE plot shown in figure 8. We observe that the randomized agent clearly separates the two training domains, whereas the regularized agent learns similar representations for both domains.

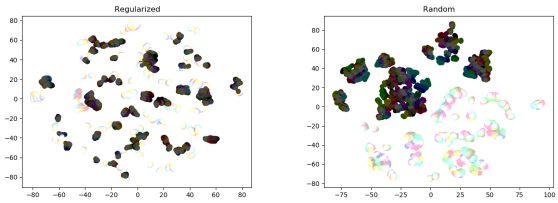


Figure 8: t-SNE of the representations over $_{split}$ of the Regularized (Left) and Randomized (Right) agents. Each color corresponds to a domain. The randomized agent learns very different representations for $[0; 0.2]^3$ and $[0.8; 1]^3$.

We are interested in how robust our agents are to unseen values $\in _{split}$. To visualize this, we rollout both agents in domains having different background colors : $f(x; x; x); 0 \quad x \quad 1g$, i.e ranging from black to white, and collect their features over an episode. We then plot the t-SNEs of these features for both agents in figure 9, where each color corresponds to a domain.

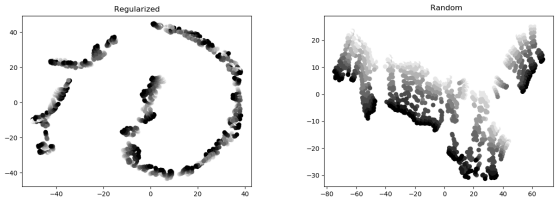


Figure 9: t-SNE of the features of the Regularized (Left) and Randomized (Right) agents. Each color corresponds to a domain.

We observe once again that the regularized agent has much lower variance over unseen domains, whereas the randomized agent learns different features for different domains. This shows that the regularized agent is more robust to domain shifts than the randomized agent.

C FURTHER RELATED WORK: LIPSCHITZ CONTINUITY AND GENERALIZATION IN DEEP LEARNING

Lipschitz-sensitive bounds on the generalization abilities of neural networks have a long history (Bartlett (1997); Anthony & Bartlett (2009); Neyshabur et al. (2015)). Recently, Bartlett et al. (2017) proved a generalization bound in terms of the norms of each layer, which is proportional to the Lipschitz constant of the network. Oberman & Calder (2018) studied generalization through a general empirical risk minimization procedure with Lipschitz regularization, and provides generalization bounds. Similarly, we show that the Lipschitz constant of the network with respect to the randomization parameters plays an important role in achieving zero-shot transfer to a target domain.

D ALGORITHMS

Algorithm 1 Deep Q-learning with our regularization method

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
Initialize the randomization space , and a reference MDP M_{ref} to train on.
Initialize a regularization parameter
Define a feature extractor f
for episode = 1; M **do**
  Sample a randomizer function ^{sampled} uniformly from .
  for $t = 1; T$ **do**
    With probability select a random action a_t
    otherwise select $a_t = \arg\max_a Q(^{ref}(s_t); a;)$
    Execute action a_t in M_{ref} and observe reward r_t and both the reference and randomized states : $^{ref}(s_{t+1}); ^{sampled}(s_{t+1})$
    Store transition ($^{ref}(s_t); ^{sampled}(s_t); a_t; r_t; ^{ref}(s_{t+1}); ^{sampled}(s_{t+1})$) in D
    Sample random minibatch of transitions ($^{ref}_j; ^{sampled}_j; a_j; r_j; ^{ref}_{j+1}; ^{sampled}_{j+1}$) from D
    Set $y_j = \begin{cases} r_j \text{ for terminal } ^{ref}_{j+1}: \\ r_j + \max_{a'} Q(^{ref}_{j+1}; a';) \\ \text{otherwise.} \end{cases}$
    Perform a gradient descent step on $(y_j \quad Q(^{ref}_j; a_j;))^2 + kf(^{ref}_j) \quad f(^{sampled}_j)k_2^2$
  end for
end for

Algorithm 2 Policy Gradient with a baseline using our regularization method

Initialize policy network function with random weights , baseline b
Initialize the randomization space , and a reference MDP M_{ref} to train on.
Initialize a regularization parameter
Define a feature extractor f
for episode = 1; M **do**
  Sample a randomizer function ^{sampled} uniformly from .
  Collect a set of trajectories $(s_0; a_0; r_1; \dots; s_{T-1};$
  $a_{T-1}; r_T)$ by executing on M_{ref}.
  for $t = 1; T$ in each trajectory **do**
    Compute the return $R_t = \sum_{t'=t}^{T} {}^{t'-t} r_{t'}$
    Estimate the advantage $\hat{A}_t = R_t - b(^{ref}(s_t))$
  end for
  Perform a gradient descent step on

$$\sum_{t=0} \hat{A}_t \log(a_t | ^{ref}(s_t)) + k R_t - b(^{ref}(s_t))k_2^2 + k f(^{ref}(s_t)) - f(^{sampled}(s_t))k_2^2$$

end for

Note that algorithm 2 can be straightforwardly adapted to several state of the art policy gradient algorithms such as PPO.

E A DYNAMICS RANDOMIZATION EXPERIMENT

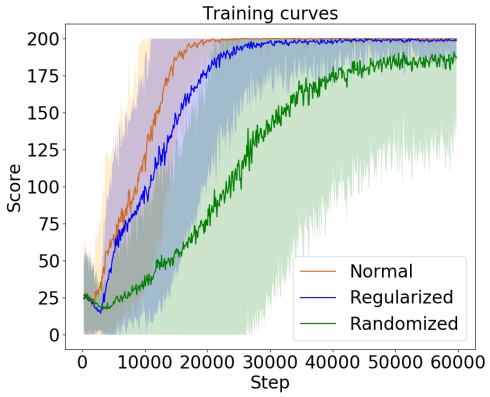


Figure 10: Training curves of the agents on the Cartpole domain, averaged over 100 seeds.

We also perform an experiment to demonstrate that learning domain-invariant features can be helpful not only for visual randomization, but also for some instances of dynamics randomization. We consider once again the Cartpole domain, where this time we randomize some physical dynamics of the environment. Specifically, we choose to randomize the pole length l, and the gravity g. The state for our reinforcement learning agent is the 6-dimensional vector $(x; \dot{x}; ; \dot{}; l; g)$, where x is the position of the cart, $\dot{x}$ its velocity, the angle between the pole and the horizontal plane, and $\dot{}$ the angular velocity. We train all three agents (Normal, Regularized, Randomized) using only two randomized domains : $l = 0.5; g = 9.8$ and $l = 1; g = 50$. We use the DQN algorithm with a Multi-Layer Perceptron architecture for this experiment.