Spring 5-15-2023

Adversarial Patch Attacks on Deep Reinforcement Learning Algorithms

Peizhen Tong

WASHINGTON UNIVERSITY IN ST. LOUIS

McKelvey School of Engineering
Department of Computer Science & Engineering

Thesis Examination Committee:
Yevgeniy Vorobeychik, Chair
Nathan Jacobs
Ning Zhang

Adversarial Patch Attacks on Deep Reinforcement Learning Algorithms

by
Peizhen Tong

A thesis presented to
the Mckelvey School of Engineering
of Washington University in
partial fulfillment of the
requirements for the degree
of Master of Science

May 2023
St. Louis, Missouri

Table of Contents

List of Figures iv
List of Tables v
Acknowledgements vi
Abstract viii
Chapter 1: Introduction 1
Chapter 2: Deep Reinforcement Learning 3
2.1 Deep Reinforcement Learning 3
2.2 Deep Q-Network (DQN) and Duelling Deep Q-Network 3
2.3 Proximal Policy Optimization (PPO) 4
Chapter 3: Literature on Robust Deep Reinforcement Learning and Adversarial Patch 5
3.1 Vulnerability of Deep Neural Networks 5
3.2 Defense for Deep Neural Network 6
3.3 Attacks on Deep Reinforcement Learning 7
3.4 Defense for Deep Reinforcement Learning 10
3.5 Certification for Defense for Deep Reinforcement Learning 14
3.6 Adversarial Patch Attacks 15
3.7 Defense for Adversarial Patch 16
Chapter 4: Adversarial Patch Attack on Deep Q-Networks 17
4.1 Generate all Available States 17
4.2 Attack on the Given Position 17
4.3 Choose Patch Position 18
4.4 Select Best Patch 20
4.5 Experiments of Attack on Deep-Q-Networks 20
4.6 Subsequence Patch Attack on Deep Q-Networks 23
4.7 Experiments of Subsequence Attack on Deep Q-Networks 24

Chapter 5: Defense against Adversarial Patch for Deep Q-Networks 26
5.1 Identify Patch Position with Context Re-Constructor 26
5.2 Mask Defense 28
5.3 Recover Defense 31
5.4 Experiments of Mask Defense and Recover Defense 32
Chapter 6: Adversarial Patch Attack on Proximal Policy Optimization 35
6.1 Attack Method 35
6.2 Experiments of Attack Method 35
Chapter 7: Defense against Adversarial Patch for Proximal Policy Optimization 37
7.1 Mask Defense and Recover Defense for PPO 37
7.2 Experiments of Mask Defense and Recover Defense for PPO 37
Chapter 8: Transferability of Adversarial Patch Attacks 42
Chapter 9: Conclusion and Outlook 45
References 46

List of Figures

Figure 4.1: Efficient patch time attack on Freeway, BankHeist, Pong, RoadRunner and Qbert. X-axis represents the proportions of time steps being attacked on, and Y-axis represents the rewards of the agent. 25

Figure 5.1: Origin state, state with patch on, re-constructed state from left to right, respectively. From up to down, Freeway, BankHeist, and Qbert. 27

Figure 7.1: Origin state, state with patch on, recovered state, synthesis state, for Coinrun..... 38

List of Tables

Table 4.1: The results of patch attack under different ratios r of state, and PGD and FGSM attack under different magnitude of perturbation r. 22

Table 4.2: Clearn Reward for DQN 22

Table 5.1: The results of performance of mask defense and recover defense against adversarial patch. 33

Table 5.2: The results of mask defense when random positions are masked in every time step, and results of recover defense when random positions are filled with white and recovered by context encoder in every time step. 34

Table 6.1: Adversarial patch attack on PPO under different ratios of patch to the size of the image. 36

Table 6.2: PGD attack on PPO under different magnitude of perturbations. 36

Table 7.1: Clean Reward for PPO 39

Table 7.2: Mask defense with unknown patch position 39

Table 7.3: Recover defense with unknown patch position. 39

Table 7.4: Mask defense with known patch position 40

Table 7.5: Recover defense with known patch position. 40

Table 7.6: Mask defense with random masked regions. 41

Table 7.7: Recover defense with random recovered regions. 41

Table 8.1: Attacks on DQN 43

Table 8.2: Attacks on PPO in Training Distribution 43

Table 8.3: Attacks on PPO in Evaluation Distribution 44

Acknowledgments

Two years ago, I moved from Santa Barbara to Saint Louis, where I enjoy the view of Forest Park from my apartment building instead of the view of charming California beach, and where I am fortunate to be advised by Professor Yvegeniy Vorobeychik. It is a wonderful research experience to be guided by Professor Yevgeniy Vorobeychik, who assists my research with great help throughout the year. I also wants to thank Professor Ning Zhang and Professor Nathan Jacobs for reviewing my works in the committee. And I am very grateful that my parents is always being supportive to me. My parents devout their love to me and fulfill the need of me for my studies over years. My grandpa, who had already passed away, had instilled confidence and optimism in me in the journey of growing up. Besides, I am happy to know the friends that I meet here. All these people support me in different perspectives to allow me to complete my thesis.

Peizhen Tong

Washington University in St. Louis

May 2023

Dedicated to my family.

ABSTRACT OF THE THESIS

Adversarial Patch Attacks on Deep Reinforcement Learning Algorithms

by

Peizhen Tong

Master of Science in Computer Science

Washington University in St. Louis, 2023

Professor Yevgeniy Vorobeychik, Chair

Adversarial patch attack has demonstrated that it can cause the misclassification of deep neural networks to the target label when the size of patch is relatively small to the size of input image; however, the effectiveness of adversarial patch attack has never been experimented on deep reinforcement learning algorithms. We design algorithms to generate adversarial patches to attack two types of deep reinforcement learning algorithms, including deep Q-networks (DQN) and proximal policy optimization (PPO). Our algorithms of generating adversarial patch consist of two parts: choosing attack position and training adversarial patch on that position. Under the same bound of total perturbation, adversarial patch attacks achieve comparable results as FGSM and PGD attack, on Atari and Procgen environments, for DQN and PPO respectively. In addition, We also design Context Re-Constructor to reconstruct state when the state is corrupted by the patch. Based on the reconstructed states, we can identify the patch position and then use mask defense and recover defense to defend against adversarial patch. Lastly, we also test the transferability of adversarial patch.

Chapter 1: Introduction

Deep reinforcement learning has proved its ability on many complex applications, including: plan the sequence of future actions for autonomous driving under current environment[1], achieve super-human performance in Atari games [2], and solve continuous control tasks [3]. However, many studies have shown that deep reinforcement learning are vulnerable under adversarial attack [4–6].

When the adversarial noise is imperceptible, FGSM attack on deep Q-network and proximal policy optimization can reduce the rewards of the agent significantly [4]. In addition, FGSM attack is still efficient if we only generate adversarial noise per tenth steps and reuse it in the intermediate steps [7]. It is also plausible to reduce the rewards of the agent dramatically by only attacking partial time steps [8, 9]. However, no physical realizable attack has been used on deep reinforcement learning algorithms. It has been proved that physical realizable attack can be captured by camera and fool the classifier [10, 11]. We propose the first physical realizable attack on deep reinforcement learning algorithms with adversarial patch, which can enable the misclassification of the deep neural network to the target label in the traditional image classification problem [12].

Previous defenses of deep reinforcement learning focus on defending against perturbation of all pixels under certain magnitude. SA-DQN and SA-PPO use regularization terms to reduce the KL distance between the original policy and the perturbed policy [13]. Radial framework utilizes interval bound propagation to enable agent make optimal decision under worst perturbation [14]. Using the loss functions of radial framework, bootstrapped opportunistic curriculum enable the deep reinforcement learning algorithms to resist larger magnitude of perturbation [15]. Nevertheless, the defense against an entirely corrupted region with unbounded perturbation has not been explored, yet.

In this work, we design mask defense and recover defense with Context Re-Constructor together to overcome this obstacle.

The main contributions of our works can be summarized into three parts. Firstly, we propose the adversarial patch attack methods on two deep reinforcement learning algorithms: Deep Q-networks and Proximal Policy Optimization. Our attack method includes how to find the attack position and how to train the adversarial patch on the given region. The experimental results demonstrates that adversarial patch attack can reach comparable performance than PGD attack on Atari and perform even better than PGD attack on Procgen under same total perturbation bound. Second, we design Context Re-Constructor, which can reconstruct the state when the given state is partial corrupted, to identify the position of the patch, and then use mask defense or recover defense to predict the optimal action. Third, we also do the experiments on blackbox attacks of adversarial patch to explore the transferability of adversarial patch attacks.

Chapter 2: Deep Reinforcement Learning

2.1 Deep Reinforcement Learning

Deep reinforcement learning environment can be modeled by the Markov Decision Process (MDP). MDP can be expressed as a tuple $(S, A, P, R, \gamma, s_0)$, such that $S \in \{s_1, s_2, ...\}$ is the set of possible states in the environment, $A \in \{a_1, a_2, ...\}$ is the set of actions that the agent can take, $P(s'|s, a)$ is the transition probability, $R : S \times A \to \mathbb{R}$ is the reward function, γ is the discount factor, s_0 is the initial state of the agent. The goal of the agent is to maximize the total reward, which can be expressed as $\sum_t \gamma^t r^t$, where γ^t is the discounted factor for time step t, r^t is the reward for time step t. We want to find a policy $\pi : S \times A \to \mathbb{R}$ as the possibility to take action a in state s, which can maximize the expected total reward $\mathbb{E}[\sum_t \gamma^t r^t]$.

2.2 Deep Q-Network (DQN) and Duelling Deep Q-Network

Q-value represent the total reward when take action a, starting at state s. Bellman equation states that optimal Q-values satisfy the equation:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a'). \tag{2.1}$$

Using Bellman equation, deep Q-networks consists of the input states and output Q-values, by minimizing the expected temporal lost [2]:

$$\mathcal{L}(\theta) = \mathop{\mathbb{E}}_{(s,a,s',r)}[(r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta))^2]. \tag{2.2}$$

DQN has demonstrated its effectiveness on Atari environments [16], and it is also the first deep learning method of reinforcement learning.

Duelling DQN has the similar neural networks as DQN, but it predicts the advantage the function and value function separately, and then use them to output Q-values [17]. The value function $V(s)$ is the expected reward from the given state s. Advantage function $A(s, a)$ measures the goodness of an action to others. The output Q-values can be calculated as:

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{|A|} \sum_{a'} A(s, a') \tag{2.3}$$

Finally, duelling DQN would use the same loss function as the vanilla DQN to optimize the weights of the deep neural network.

2.3 Proximal Policy Optimization (PPO)

Proximal policy optimization is a policy gradient method for deep reinforcement learning. Proximal policy optimization has advantages over other methods in the family of policy gradient method. Proximal policy optimization have better performance than vanilla policy gradient method on MuJuCo, while it is also easier to implement than trust region policy optmization (TRPO)[18]. The loss function of PPO is expressed as [19]:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s_t, a_t, r_t)}[-\min(\frac{\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta_{old})}A_t, clip(\frac{\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta_{old})}, 1-\epsilon, 1+\epsilon)A_t)], \tag{2.4}$$

where s_t is state in time step t, a_t is the action taken in time step t. θ is the parameter of the PPO model, A_t is the advantage function in time step t, and ϵ is the hyperparameter.

Chapter 3: Literature on Robust Deep Reinforcement Learning and Adversarial Patch

3.1 Vulnerability of Deep Neural Networks

Goodfellow et al. [20] explain the vulnerability of neural network is due to the their linear property. When the dimension of the input is high, small perturbations in each dimension would eventually cause a large perturbation of the output. With this property, they create Fast Gradient Sign Method (FGSM), where each pixel is modified in the direction of the gradient of the pixel by a constant value. The formula for FGSM can be expressed as:

$$X_{adv} = X + \epsilon * \nabla J(X, y_{true}). \tag{3.1}$$

FGSM attack is imperceptible to human eyes, if the perturbation is small, but it is able to produce high error rate to enable the deep neural network to misclassify inputs. To increase the robustness against FGSM attack, we can produce adversarial samples generated by FGSM and apply adversarial training by them.

Basic iterative method [21] uses the same way to calculate the perturbations for the input as FGSM; however, basic iterative method would repeat this process for several iterations and clip the value to an accepted range. They also propose least likely class method, in which we are trying to maximize the probability that the image is classified as the least preferred label.

DeepFool [22] intends to find the perturbation with the minimum distance to the original input to alternate the classification of the deep neural network. DeepFool predicts the position of the closest boundary location by Taylor approximation, and this process repeats until the predicted

label changes. DeepFool can also to record the total distance adjusted to change the label of the input, and total distance can be used to measure the robustness of the classifier.

Kurakin et al. find that there is transferability of adversarial samples for different classifiers [23]. In addition, adversarial samples, which are wrong predicted after the model has been trained with adversarial examples, have weaker transferability. They also suggest that model with high capacity typically has higher robustness against adversarial attack. In addition, they claim that after adversarial training, model predict adversarial data more acccurately than clean data.

3.2 Defense for Deep Neural Network

Goodfellow et al. shows that adversarial training can increase the robustness of deep neural classifier against adversarial attack [20]. The propose an loss function combining the loss for clean input and the adversarial loss for adversarial input with an tradeoff factor $\alpha = 0.5$. The adversarial inputs are generated with fast gradient sign method. They test the performance of adversarial training on MNIST classifier, showing that the error rate decreases from 89.4% to 17.9%.

Curriculum adversarial learning (CAT) has been proved to have better performance than traditional adversarial learning in dataset SVHN and Cifer-10 [24]. Curriculum adversarial training increases the target robustness gradually - It would add adversarial perturbations of a stronger level if it achieve robustness on the current level of strength of adversarial attack. In addition, traditional adversarial training would overfit the strong attack and therefore give accurate prediction on clean test data. Curriculum adversarial learning proposed by Cai et al. solve this problem by batch mixing, in which a batch contains adversarial samples generated by different strength of attack. Cai et al. also propose using quantization to restrict the input, to enable the classifer to endure stronger attack that the classifier has not been trained on. With ResNet-50 and DenseNet-161 as the classifier, CAT significantly outperform traditional adversarial training in Cifer-10 and SVHN.

Internal bound propagation can calculate the upper and lower bound of output under the perturbation. Gowel et al. incorporate the method of interval bound propagation into adversarial training [25]. The loss function for adversarial training includes the cross entropy loss of the normal output and the cross entropy loss of the output calculated by interval bound propagation. This method is tested by Cifer-10, MNIST, and SVHN, and surpasses the robustness of the state of the art result.

Raghunathan et al. provide a certification that given a neural network and the input value, the error can be guaranteed inside a bound [26]. They first calculate the bound of the output based on the perturbed input in one layer classifier and two layer classifier, and then expand the bound of the general classifier. With the output bound of the general case, they provide an loss based on the adversarial input.

3.3 Attacks on Deep Reinforcement Learning

Traditional neural network model is vulnerable to adversarial perturbations. Because deep reinforcement learning also utilizes the prediction of neural network, common attacks on deep neural network might be also effective on deep reinforcement learning. Huang et al. [4]shows Fast Gradient Sign Method (FGSM), which has demonstrates its effectiveness on image classification tasks, can significantly reduce the reward of multiple type of deep reinforcement learning algorithm, including Deep Q-Networks (DQN), Asynchronous Advantage Actor-Critic (A3C), and Trust Region Policy Optimization (PPO). Under l_∞-norm, when $\epsilon = 0.001$, under which the perturbation would be imperceptible to human, the performance of models would be reduced by roughly 50 percent. In addition, the performance of the model would dramatically decreases under l_{norm} with only a few pixels attacked. Huang et al. also tests black-box attack by FGSM, including transferability across policies and algorithms. The performance shows that transferring across policies can still

significantly reduce the reward in most cases, but transferring across algorithms is less effective than transferring across policies.

Kos and Song [7] compare the performance of FGSM attack and random noise on deep reinforcement learning. Their study show that random noise is ineffective when the perturbation is small, and in contrast, FGSM attack are very effective despite the small size of perturbation, which is same as the result of Huang et al.[4]. The paper also presents that FGSM attack is ineffective when perturbations are injected only per tenth frame and no attacks injected in the intermediate frames. But it would be a effective attack if the intermediate frames are attacked by the same perturbation of each tenth frame, and the result can be as good as naive FGSM attack. They also design an attack method that the FGSM perturbation is only injected when value function is above a threshold. Their experiment demonstrates that that their attack inject similar number perturbations as injecting per tenth frame, but more effective than it.

Lin et al. [8] propose two attack method: strategically-timed attack and enchanting attack. Strategically only attack a subset of state, when an action is strongly preferred and therefore important to be taken. The probability of taking an action can be directly predicted by A3C algorithm. For DQN, the probability of taking an action can be generated based on Q-values. Then these states would be attacked by the algorithm proposed by Carlini et al.[27], to be induced to take the least preferred action. Enchanting attack includes two parts: 1. Planning a sequence of states to induce the current state to the target state. 2. Adding the perturbations to states to enable the environment to produce the planned sequence. The sequence of states can be predicted by a generative mode, and the actions for leading the current to the target state can be achieved by sampling-based cross-entropy method. The experimental results show that strategical-timed attack can achieve the same effectiveness of uniform attack by attacking 25 percent of time steps, and the enchanting attack has good successful rate to induce the initial state to the target state if the sequence of planned action is less than 40.

Similarly as the attack algorithm of Lin et al., critical point attack and antagonist attack are proposed by Sun et al. [9] to enable attacker to reduce the award of the agent by attacking only a small subset of time steps. Critical point attack generate all possible sets of actions in N steps, and evaluate the performance of those actions in M steps, such that $M \geq N$. The consequent M states can would be predicated by a state generator, to enable the attacker to predict the sequence of states under different sets of actions. The performance of the action is measured by the danger awareness metric based on divergence function. If the performance of a set of actions has reached a specific threshold, this sequence of action would be selected to mislead the agent. Then the attacker would choose craft the perturbations to mislead the agent to follow the sequence of actions selected. However, the attacker is required to have the knowledge of divergence function. Antagonist attack does not need divergence function to calculate danger awareness metric to evaluate the performance. Antagonist attack uses an adversarial policy: $s \mapsto (p, a)$, such that s is the input state, p is for deciding whether the attacker should attack, and a is the action that the attack want to mislead the agent to take. If p is larger than a specific threshold, then the attacker would add the perturbations to the image based on the method proposed by Carlini et al. [27].

Using the property of State Adversary Markov Decision Process (SA-MDP), [13] suggest that we can use deep reinforcement learning algorithm to find an optimal policy for adversary, named optimal adversary, based on the reward function proposed by Zhang et al. [28]. Optimal adversary is tested on four Mujuco environments, in which the agent and the adversary is trained by Proximal Policy Optimization, and the results show that optimal adversary is more effective than many existing attacks.

Hussenot et al. propose an adversarial attack by luring the agent to follow an adversarial policy [29]. They use FGSM attack to generate perturbations to enable the agent follow the action recommended

by agent policy instead of action recommended by the original policy.

Based on the policy of the agent and reward function, Russo et al. calculate the optimal perturbation to minimize the Q values to attack DQN and DDPG [30]. In the case that the policy of the agent is not known, they provide a loss function to minimize the overall Q values to train a function to generate perturbation. They test their method in four environments and compare the performance with gradient based attack, in which their method generally outperformed gradient based attack for perturbation ranging from 0.01 to 0.07.

Most attacks on deep reinforcement learning focus on changing the observation, instead Gleave et al. tries to find an adversarial policy to minimize the reward of the agent [31]. They train the adversarial policy to minimize the reward of the agent in a zero-sum game.

3.4 Defense for Deep Reinforcement Learning

Zhang et al. propose State Adversary MDP (SA-MDP) [28], in which the set of perturbations of the state is given. They explore many theoretical properties of SA-MDP, including that the optimal policy for the agent may not exist under the optimal adversary. SA-MDP also provide the theoretical foundation to calculate the KL-divergence of the original policy and the policy for the states under perturbations. To improve the robustness of the model, they add policy regularizers, based on KL-divergence, to the loss functions, for the algorithms, including PPO, DDPG, and DQN. Under Robust Sarsa attack and PGD attack in Mujoco and Atari environment, respectively, deep reinforcement learning models show high robustness against them.

Shen et al. combine the method of smoothing and regularization to increase the robustness of DRL [32], named Smooth Regularized Reinforcement Learning (SR^2L). The perturbations is injected into the original states, and the regularizer measures the KL-divergence between the policy

with the original state input and the policy with the state injected with perturbation. They apply Smooth Regularized Reinforcement Learning in 2 reinforcement learning algorithms, including Deep Deterministic Policy Gradient (DDPG) and Trust Region Policy Optimization (TRPO). The regularizer can be expressed as :

$$\mathcal{R}^{\pi}(\theta) = \mathbb{E}[\max_{s'} \mathbb{D}_{kl}.(\pi(s), \pi(s'))] \tag{3.2}$$

They name DDPG-SR for DDPG with smooth regularized reinforcement learning and TRPO-SR for TRPO with smooth regularized reinforcement learning. Using smooth regularized reinforcement learning, DDPG-SR and TRPO-SR perform better than the existed baseline in Mujuco environments, including swimmer and half cheetah. When the pertubation is added and becomes larger, DDPG-SR and TRPO-SR perfrom significantly better than the naive algorithm.

NoiseNets is trained to improve the performance of DQN, Dueling DQN, and A3C with weights of the neural network with noise produced by Gaussian distribution [33]. Noise parameters θ can be expressed as:

$$\theta = \mu + \Sigma \odot \varepsilon, \tag{3.3}$$

where μ and Σ are noisy parameters that the network want to learn, and ε are zero mean noise. The lost function of Network would try to minimize the temporal loss using current neural network and the target neural network, in which both uses noisy parameters. NoistNets are tests in Atari environment, and show significantly improvement than the naive DRL algorithm. Naive DQN, Dueling DQN and A3C is outperformed by human, in BeamRider, Asteroids and Freeway. However, training with noisy nets using these DRL algorithms can have better performance than human players.

Behzadan et al. [5] have demonstrated that DQN can recover from the non-contiguous FGSM attack by itself. In addition, they discover that there is a phase transition point, where the agent is stopped being deteriorated. Apart from it, they find out that through adversarial training, models are more robust against test time attack. They also the robustness of two techniques: NoiseNets and

ϵ-greedy exploration. Training by ϵ-greedy exploration provide more robust model than traning by NoiseNets; however, NoiseNets is able to recover faster under non-contiguous attack.

Pattanik et al. [6] propose adversarial training to imporve the robustness of DQN and Deep Deterministic Policy Gradient (DDPG). In order generate adversarial samples for training, they design two methods: 1. Naive Attack: The perturbation is generated by beta distribution, and if the generated perturbation lead to a bad action, then the perturbation would be added to the original state for adversarial training. 2. Gradient based Attack: They propose an loss function based on the cross entropy loss between adversarial probability and optimal policy. The attacker would implement FGSM attack, using this loss function. These attack method can be applied for adversarial traning method to increase the robustness of the model. They also propose robost control framework that the target of adversarial training is to improve the worst α percentile of expected returns.

Oikarinen et al. provide an adversarial training framework, RADIAL [14]. The loss function of RADIAL include the norm loss and the adversarial loss, which can be expressed as:

$$\mathcal{L}_{RADIAL} = \kappa\mathcal{L}_{norm} + (1-\kappa)\mathcal{L}_{adv}. \tag{3.4}$$

The adversarial loss function focus on finding the upper bound of the output due to the perturbation and trying to minimize the overlap among the bounds of outputs of different actions. Based on this intuition, Oikarinen et al. provide loss functions for a variety of deep learning algorithms, to create RADIAL-DQN, RADIAL-A3C, and RADIAL-PPO. They also invent an evaluation method called Greedy Worst-Case Reward, which can predict the reward of the model under the worst case perturbations. RADIAL-DQN is test on 4 Atari environments. When the perturbation is under 1/255, the reward is barely affected. RADIAL-DQN can still perform well when the perturbation is 5/255. RADIAL-A3C and RADIAL-PPO also demonstrate their robustness on Procgen environments.

Another way to train a robust DQN is to learn a Robust Student DQN based on a trained DQN

[34]. They initially use policy distillation, which can enable the DQN to learn the policy better , to train a normal DQN. For Student DQN, its loss function can be the mean square loss or the cross-entropy-loss related to the output of normal DQN and Student DQN. To train a Robust Student DQN, the input state for Student DQN during training would be added adversarial perturbation generated by PGD attack, and the formula for the loss function for Robust Student DQN is:

$$\mathcal{L} = \mathcal{H}(\sigma(S(s_d, a; \theta_s), \underset{a}{\operatorname{argmax}}\, \mathrm{Q}(s, a; \theta_Q)), \tag{3.5}$$

where s_d is the state after PGD attack, $S(s_d, a; \theta_s)$ is the output of Robust Student DQN and $\mathrm{Q}(s, a; \theta_Q)$ is the output of normal DQN. Fischer et al. also incorporate this method with Deuling DQN and NoiseNets. Robust Student DQN is prove to be robust under ±1 pixle perturbation with 4 steps PGD attack.

Wu and Vorobeychik propose an bootstrapped opportunistic curriculum learning (BCL) [15] to enable the deep reinforcement learning to become robust against relatively large perturbations. They apply adversarial training for curriculum learning, in which the model is trained for the next size of perturbation only when the model in the previous curriculum phase has achieved a robust baseline. The next size of perturbation is chosen in an opportunistic way. And in each curriculum phase, multiple models would be trained on bootstrapped adversarial state based on the robust baseline line model from the previous state, and the model with the best performance of them would be pick for the robust baseline for the next curriculum phase. This process repeats when the model is robust for the target size of perturbation. For adversarial attack, they propose a loss function to minimize the Q-values for the training batch, using softmax function. This method can also combine other method, such as RADIAL, to make the model more robust under some environments. BCL and its variants outperform other robust traning method, such as RADIAL-DQN and SA-DQN, in four Atari environments. BCL and its variants also prevail RADIAL-PPO in two Procgen environments: Fruitbot and Jumper.

Mandlekar et al. add dynamic noise and observation noise to the state, and the experimental results show that DRL algorithms are sensitive to adversarial perturbations. They also apply adversarial training to find an robust policy for DRL. In each training batch, they sample trajectories based on the current trained policy, and add noise to the states in the trajectory based on Bernoulli distribution. They trained 15 agents and select the agent with the best learning curve for the experiment, in each of four Mujuco environments: Inverted Pendulum, Walker2D, half cheetah, and hopper. For inverted Pendulum, using adversarial training proposed, the agent is robust against dynamic noise, and perform well when no noise exist. For walker task, both normal agent and agent with adversarial training are able to resist the dynamic noise. Overall, using adversarial training, the agent becomes robust against noise perturbation in the environment.

3.5 Certification for Defense for Deep Reinforcement Learning

Lütjens et al. provide a lower bound of Q-values under bounded perturbation for Q-learning [35]. In addition, they provide an optimal action selection rule to make sure to maximize the reward under worst-case perturbation.

Wu et al. provide the first robust certification for action and reward for Deep Q-Networks [36]. They propose certification for per-state action via action-value functional smoothing (CROP-LoAct), which shows the lower bound of the radius of perturbation. They also present certification of cumulative reward based on global smoothing (CROP-GRe), which provide the lower bound of reward under ϵ bound of perturbation. However, this absolute lower bound is loose, and therefore they also present a lower bound for the percentile of the expected reward. In addition, they also present the radii r_t^k, which guarantees that the agent would take the top k actions in the time step t. Using the proved radius bound, local smoothing for certified bound (CROP-LoRe) explores different trajectories to find the minimum reward as the lower bound of the reward for the given perturbation.

CROP-LoRe provide an effective way to evaluate the robustness of deep reinforcement learning algorithms.

3.6 Adversarial Patch Attacks

Unlike FGSM, PGD, and other attacks which wants to be imperceptible to human, adversarial patch attack puts a visual patch to the image to let the model incorrectly classify the image to the target label [12]. Adversarial patch is universal, and independent of the input image. The objective function to train the adversarial patch is the expectation of the probability of classifying as the target label under different images, locations and rotations. For example, we train an adversarial patch to enable the classifier believe the input is actually a toaster. If the input image is a banana, after we put the trained patch on the image, the classifier would actually classify the image as a toaster with high confidence.

Instead of creating an universal patch, it is also possible to generate adversarial patch using an generative model for each state [37]. From the generated adversarial state, we can synthesis the new image with the original image. The objective function would intended to maximize the prediction loss of the classifier between the synthesis image and the original image. The generative model for adversarial patch is tested by Image-Net. When the size of patch is 10% of original image, it has roughly 80 percent successful rate to enable the classifier to misclassify the image. The performance of generative model for patch outperforms training patch by gradient descent by Brown et al, using Image-Net.

3.7 Defense for Adversarial Patch

Hayes propose digital watermarking. which uses image gradient to fill the part corrupted by the patch to improve the accuracy of classification against adversarial patch [38]. The region corrupted by patch can be filled as average value of prediction based on surrounding region and their image gradients. If the patch position is unknown, they will utilize saliency map to identify the location of the patch position, and then apply the same procedure to fill the image. The method is tested by ImageNet, and the size of patch ranges from 2 to 25 percent of the original image. If we know the position of the patch, reconstruction image using image gradients achieve very high accuracy on ImageNet. However, if the position of the patch is unknown, the method proposed by Hayes only achieve high successful rate when the size of the patch is very small.

Nasser et al. also an method, named local gradient smoothing, based on image gradient [39]. Using image gradient, they calculate the magnitude of the gradient and normalize them. In addition, to increase possibility to not include benign samples, they also calculate a score for each pixel based on the normalized gradient of the pixels surrounded it within in a rectangle. Then the pixel value would be reduced by the score of the pixel. If a pixel and its surrounding pixel all have high magnitude of image, then the pixel value of it would be reduced to a very small value. Local gradient smoothing outperformed digital watermarking in ImageNet under patch attack, but have lower accuracy for classifying clean image.

Chapter 4: Adversarial Patch Attack on Deep Q-Networks

4.1 Generate all Available States

We create a states sample buffer to store all the possible available states. In this step, we are trying to generate all possible states of the environment. We generate state by two simulations. In first simulation, we take the action with highest Q-value for all time step. In second simulation, we set an random action probability ϵ, which is the probability that we randomly choose an action instead of choosing the action with highest Q values. In both simulations, total numbers of time steps are same. The states sample buffer is used in both training adversarial patch and choosing patch position.

4.2 Attack on the Given Position

We initialize a grey patch, and then we add the patch to the corresponding position on the state. Then we trained the patch to minimize the objective function:

$$\mathcal{L} = \sum_{i=1}^{N} Softmax(Q(m(S_i, P, l)) \odot Q(S_i), \tag{4.1}$$

where P represents the patch that we add on the states, l is the position of the patch, $Q(.)$ returns the an array of Q-values for all the possible actions, $m(S_i, P, l)$ is the modified state that we put the patch P on the position l of state S_i, and N is the number of states in the training batch. We use mini batch stochastic gradient descent to train the patch, and the states in the mini batch is sampled from states sample buffer. We only apply projected gradient descent to the pixels in the patch instead of all pixels of the state, and we clip the values of pixels in the patch to 0 to 1, which is also the range

of pixel.

The intuition behind this algorithm is that we want to increase the value of $Q(m(S_i, P, l), a_i)$, if the original Q-value of action a_i is low, and to decrease the value $Q(m(S_i, P, l), a_i)$, if the original Q-value of action a_j is high. Using the patch trained by this method, DQN would tend to choose the action whose original Q-value is low. As the consequence, the total reward of the simulation would decrease.

4.3 Choose Patch Position

We have designed the method to choose patch position. And this method can divide into 2 parts. The first part is sample states from states sample buffer, and then use the method based on rect angle occlusion attack [40] to find a few selected positions to place the patch. The second part is that we apply method mentioned from section 3.1 to train the adversarial patch in selected positions.

We randomly sample states from states sample buffer. We input these states into model and use formula 3.1 as the loss function, but we do need to put a patch on them in this case. Then we get the gradient of loss function with respect to the pixels of the states. We calculate the sum of gradients for all pixel positions over all input states, and we denote this as ”accumulative gradients”. We use *G* to denote the accumulative gradients of the state, and $G_{i,j}$ is the accumulative gradients for the the pixel who is on the i-th row and j-th column of the state. $G_{i,j}$ can be calculated as:

$$G_{i,j} = \sum_{k=1}^{N} grad_{k,i,j}, \tag{4.2}$$

such that *N* is the number of states in the sample states, $grad_{k,i,j}$ is the gradient of loss function with respect to the pixel in the i-th row and j-th column of the k-th state in the sample states.

Then we try to find the region who has the highest sum of accumulative gradients. We can do this by exhaustive search, and we record the K regions whose sum of accumulative gradients are highest. Then we place a grey patch on each of these K regions (We don't put all stickers on them simultaneously). And we calculate the value of the loss function, and choose the region which produces the lowest value of the objective function. This process is shown in Algorithm 1

Algorithm 1 Patch Position

Input: Sample States S; Image Width W; Image Height H; Regions to put sticker on K;
Output: X coordinates of Patch PosX; Y coordinates of Patch PosY;
$grad$ ←Calculate the gradients for all the sample states
G ←Calculate the accumulate gradients over all states using $grad$
for i in range(W) **do**
    for j in range(H) **do**
        $R_{i,j}$ ←sum of square of accumulate gradients in the region whose upperleft corner's coordinate is in i-th row and j-th column
    end for
end for
$X_{1:K}$ ←X coordinate of upperleft corner of the top K regions with highest accumulate gradients
$Y_{1:k}$ ←←Y coordinate of upperleft corner the top K regions with highest accumulate gradients
MinObjVal ←Variable to store minimum value of objective function, set to a lage value initially
for i in range(K) **do**
    S_i ←Put a grey sticker based on X_i and Y_i on S
    CurObjVal ←The loss after putting grey sticker on
    if CurObjVal < MinObjVal **then**
        MinObjVal = CurObjVal
        PosX = X_i
        PosY = Y_i
    end if
end for
return PosX, PosY

4.4 Select Best Patch

To improve the performance of patch attack, we can repeat the procedures above for a few times and select the best patch generated. To select the best patch, we can simply validate each patch in environments, and select the patch which makes the agent get the lowest reward. In the rest of thesis, we refer the number of patches generated to select the best patch as ”selection count” and all patches can be referred as ”selection patch”. The best patch selected from the set of patches is referred as ”ultimate patch”.

4.5 Experiments of Attack on Deep-Q-Networks

We test the attack method on five Atari environments: Pong, Freeway, BankHeist, RoadRunner, Qbert. In each environment, we generate states sample buffer for it. We simulate 10000 time steps with optimal action. Then we also simulate 10000 time steps with $\epsilon = 0.35$ as the probability of taking random action instead of taking optimal action. Therefore, we have simulated 20000 time steps in total, and we have stored them in the states sample buffer.

For training each selection patch, we randomly draw 128 states form states sample buffer, and calculated the accumulative gradients and then select 5 regions whose sum of accumulative gradients are highest. We put the grey sticker on these regions and choose the region generated the minimal loss. Lastly, we apply project gradient descent on that region to train the patch.

We have test the patch with different ratios of patch to the state, including 0.001, 0.0005, 0.0003, 0.0001 and 0.00005. For each scenario, we train 10 ultimate patches to test the performance of attack, and we the selection count for each ultimate patch is 3. For each ultimate patches, we simulate 10000 time steps, and we calculate the mean and standard deviation over episodes rewards from all ultimate patches in each scenario.

The experimental results are shown in Table 4.1, including the reward under the different ratios of the patch to the state. When the ratio of the patch to the state is 0.0003, it would be sufficient to reduce the reward to minimum or almost minimum in four environments, including Freeway, BankHeist, Pong and RoadRunner, and the reward in Qbert would be reduced by a dramatic value. Even if the ratio of the patch to the state is 0.0001, in most environments, the reward has been reduced at least half of the optimal reward.

Theorem 1 *For state with N pixels, the perturbation bound of patch attack whoso ratio to the state is* r *and the perturbation bound of PGD and FGSM attack whose maximum perturbation is* r *are both bounded by* $r * N$*, under* L_1 *norm.*

In addition, we have compared the performance of optimal adversarial patch to PGD and FGSM [4] attack. When there are N pixels in total and the pixels range from 0 to 1 and the ratio of patch to the state is r, the L_1 norm of the difference between the attacked state and the original state would be bounded by $N * r$. For PGD and FGSM attack, if the perturbation for each pixel can be at most r, then the L_1 nom of the difference between the perturbed image and the original image would also be at most $N * r$, which proves Theorem 1. Therefore, we can compare the performance of patch attack whoso ratio to the state is r to the performance of PGD and FGSM attack whose maximum perturbation is also r.

Form table 4.1 and table 4.2, we can see that when the ratio of the image or the size of perturbation is 0.003 or above, all three attacks have similar and satisfying performance to reduce the rewards in all five environments. Patch attack outperforms PGD and FGSM attack in BankHeist, but have weaker performance in other environments. Despite the less effective performance, patch attack still demonstrates its power to significantly to reduce the reward on deep Q-networks. And most

Environment	Method	0.0005	0.001	0.003	0.005	0.01
Freeway	Patch	14.85 ±1.19	8.28 ±3.40	1.38 ±1.73	0.0 ±0.0	0.0 ±0.0
Freeway	FGSM	22.5 ±1.65	3.75 ±0.43	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0
Freeway	PGD	21.75 ±1.29	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0
BankHeist	Patch	776.0 ±23.32	0.0 ±0.0	0.0±0.0	0.0 ±0.0	0.0 ±0.0
BankHeist	FGSM	1265.0 ±61.84	1147.5 ±38.32	0.0±0.0	0.0 ±0.0	0.0 ±0.0
BankHeist	PGD	1188.0 ±69.68	1165.0 ±32.01	0.0±0.0	0.0 ±0.0	0.0 ±0.0
RoadRunner	Patch	23558.46 ±9878.62	4554.74 ±5480.18	3.38 ±110.21	0.0 ±0.0	0.0 ±0.0
RoadRunner	FGSM	1327.0 ±7049.82	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0
RoadRunner	PGD	11354.54 ±6623.56	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0
Pong	Patch	9.27 ±16.65	-20.90 ±0.30	-21.0 ±0.0	-21.0 ±0.0	-21.0 ±0.0
Pong	FGSM	13.0 ±3.46	-19.70 ±3.92	-21.0 ±0.0	-21.0 ±0.0	-20.83 ±0.37
Pong	PGD	-1.5 ±12.99	-17.57 ±1.91	-20.54 ±0.49	-20.81 ±0.38	-21.00±0.0
Qbert	Patch	447.52 ±207.54	341.15 ±78.20	223.96 ±131.65	210.57 ±86.01	207.55 ±79.05
Qbert	FGSM	103.03 ±74.81	143.96 ±25.97	327.17±67.53	143.96 ±25.97	103.03 ±74.81
Qbert	PGD	486.0 ±286.96	290.74±110.79	258.65 ±24.94	176.66 ±67.37	133.82±44.50

Table 4.1: The results of patch attack under different ratios r of state, and PGD and FGSM attack under different magnitude of perturbation r.

Freeway	BankHeist	RoadRunner	Pong	QBert
33.9 ±0.07	1325.5 ±5.7	43390 ±973	21.0 ±0.0	800.0 ±0.0

Table 4.2: Clearn Reward for DQN

importantly, patch attack does not require backpropgation to access gradients, and does not cost additional construction time for adversarial states during test time.

4.6 Subsequence Patch Attack on Deep Q-Networks

Lin et al. has demonstrated that adversarial attacks can reduce the reward of deep reinforcement learning by a significant amount by using only a subset of frames [8]. They propose strategically time (ST) attack, aiming to attack the DRL when an action is strongly preferred over an other action. The decision to attack is based on c function and a threshold value β. The output of c function has formula:

$$c(s_t) = \max_{a_t} \pi(s_t, a_t) - \min_{a_t} \pi(s_t, a_t), \tag{4.3}$$

in which t is the current time step,π is the policy for probability to choose a action a_t given state s_t. If $c(s_t)$ is higher than the threshold value β, then we would craft the adversarial samples to lead the agent to choose the least preferred action.

For Deep Q-networks, we have Q-values to decide which action to take instead of using probability of taking the action. So we need to convert Q-values to the probability firstly [8]. The c function for deep Q-network can be expressed as:

$$c(s_t) = \max_{a_t} \frac{e^{Q(s_t,a_t)}}{\sum_{a_k} e^{Q(s_t,a_k)}} - \min_{a_t} \frac{e^{Q(s_t,a_t)}}{\sum_{a_k} e^{Q(s_t,a_k)}}. \tag{4.4}$$

An temperature constant can also be used to adjust the Q-values in the formula above. To craft adversarial examples, Lin et al. uses the method proposed by Calini and Wagner, who intend to found an adversarial sample to mislead the agent to target action with minimal perturbation from the original state [27]. Calini and Wagner solves this optimization problem by projected gradient descent or clipped gradient descent.

We propose efficient patch time (EPT) attack on deep reinforcement learning based on strategical time attack. Putting the pre-trained patch on the original state s_t at time step t would produce

adversarial state $s_{patch,t}$. The new c function as formula below:

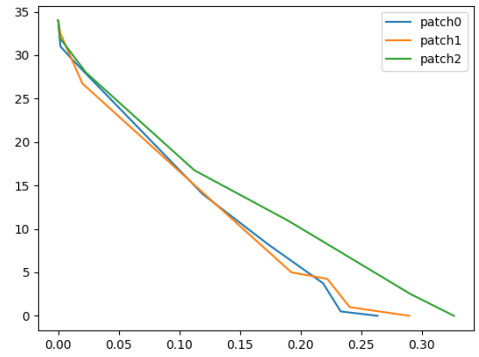
$$c(s_t) = (\max_{a_t} \frac{e^{Q(s_t,a_t)}}{\sum_{a_k} e^{Q(s_t,a_k)}}) - \frac{e^{Q(s_t,\operatorname{argmax}_{a_t} Q(s_{patch,t},a_t))}}{\sum_{a_k} e^{Q(s_t,a_k)}}. \quad (4.5)$$

If output of c function is higher than a threshold value β in time step t, we would attack deep Q-networks with pre-trained patch on the state of time step t.

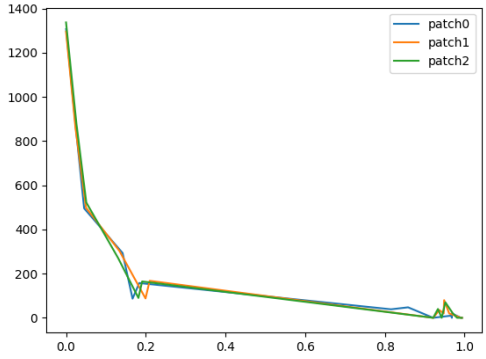
4.7 Experiments of Subsequence Attack on Deep Q-Networks

We have tested efficient patch time attack, in five Atari environments: Pong, Freeway, BankHeist, RoadRunner, and Qbert. For each environment, we use 3 ultimate patches, whose ratio to the state is 0.01.

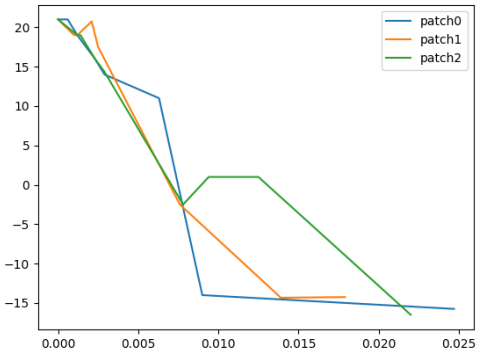
We test the attacks and record the rewards and the percent of time steps being attacked, based on different threshold values. The experimental results are shown in Figure 4.1. We can see that with only partial time steps, adversarial patch can reduce the reward to near minimum. For example, in BankHeist, with around 0.2 of total time steps, all three patches can reduce the reward to 200 from around 1300.



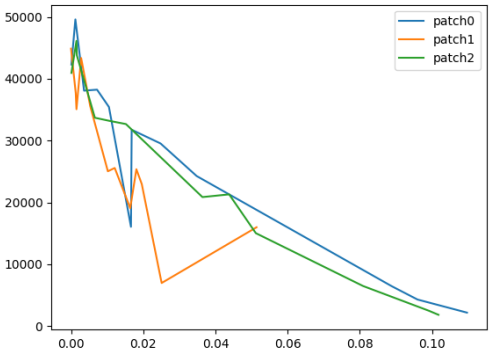
(A) Freeway



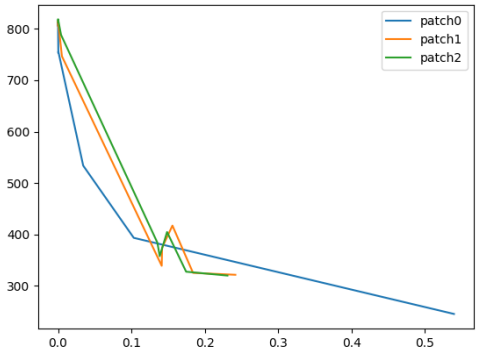
(B) BankHeist



(C) Pong



(D) RoadRunner



(E) Qbert

Figure 4.1: Efficient patch time attack on Freeway, BankHeist, Pong, RoadRunner and Qbert. X-axis represents the proportions of time steps being attacked on, and Y-axis represents the rewards of the agent.

Chapter 5: Defense against Adversarial Patch for Deep Q-Networks

5.1 Identify Patch Position with Context Re-Constructor

Context Encoder utilizes the auto-encoder structure to recover the image with masked region [41]. It uses the image with masked regions as input, and uses encoder to change the original image into an embedding, and decoder interprets the embedding into the output image, whose masked region has been recovered. The loss function of Context Encoder is the distance between the original pixel and the recovered pixel in the masked region, which has the formula:

$$\mathcal{L} = \|\hat{M} \odot (x - F((1 - \hat{M}) \odot x))\|_2^2, \tag{5.1}$$

such that x is the original image, and F is Context Encoder, $\hat{M}$ is the binary mask for the image, in which the missing region is filled with ones.

We use the similar idea to build a image recovery network, named Context Re-Constructor. However, we directly re-construct the whole image instead of the masked region, and an patch is put on the image instead of a black mask. In the thesis, state and image both mean the observation of the environment, and they can be used interchangeably. The loss function of Context Re-Constructor can be expressed as:

$$\mathcal{L} = \|x - F(P(x))\|_2^2, \tag{5.2}$$

in which x is the original image, $P(x)$ is the image with the adversarial patch on, F is the Context Re-Constructor. Context Re-Constructor has the ability to recover the states with adversarial patch to the original state. During training phase for Context Re-Constructor, we randomly select a region

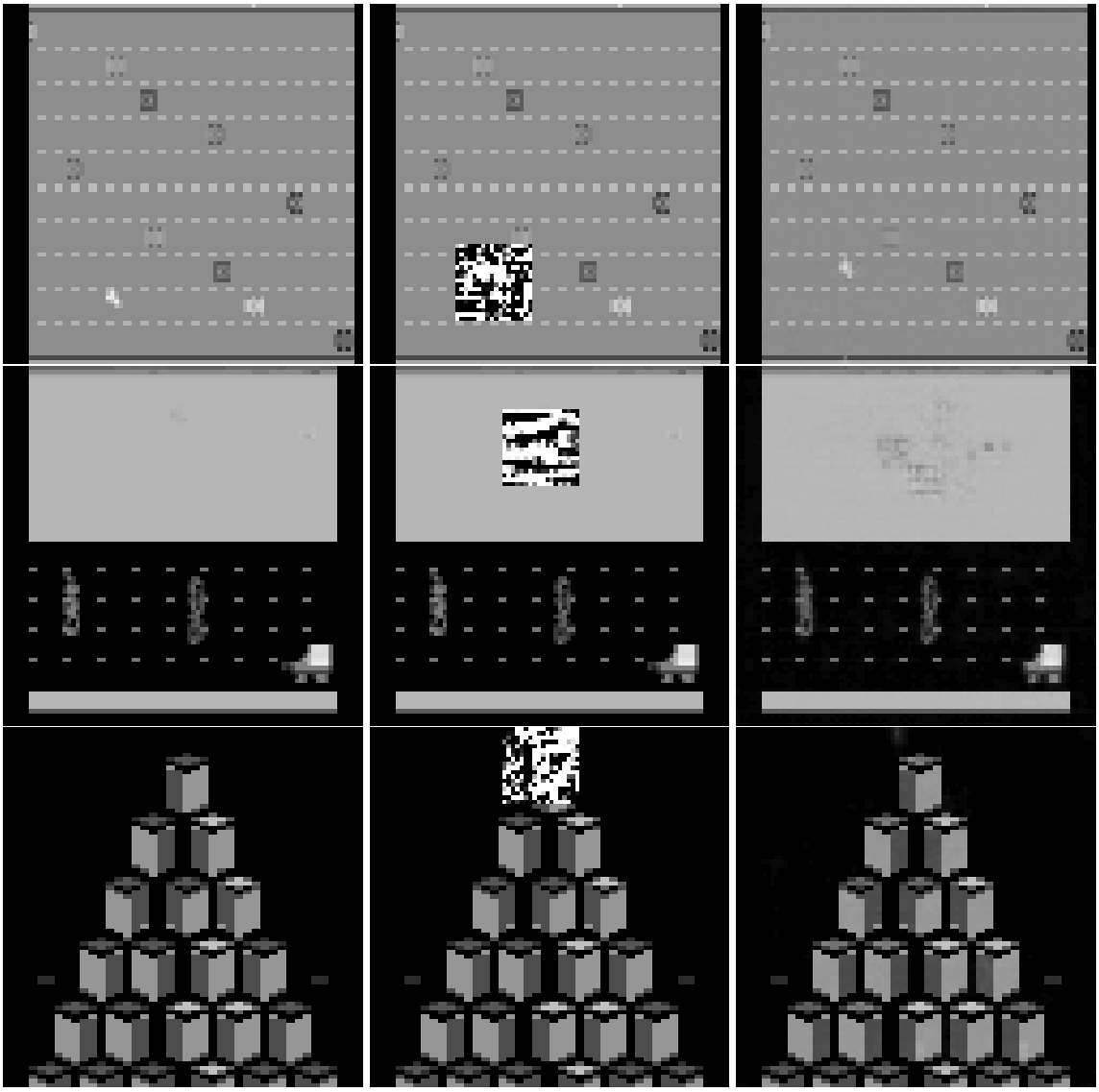


Figure 5.1: Origin state, state with patch on, re-constructed state from left to right, respectively. From up to down, Freeway, BankHeist, and Qbert.

in the sample states, and generate random numbers from 0 to 1 for each pixel in this region. Then we input the corrupted state into the Context Re-Constructor for training.

Samples of state generated by Context Re-Constructor are shown in Figure 6.1. We put a patch whose ratio to the state is 0.05, and let the Context Re-Constructor to recover the image. Because DQN takes gray scaled image as input, the generated patch and re-constructed image are also gray

scaled in Figure 6.1.

After we have the Context Re-Constructor, we can use it to identify the region corrupted by adversarial patch by the following procedures: We input the image with adversarial patch on into the Context Re-Constructor and get the generated image. Then we compare the generated image and the input image with patch on, and then select the regions with highest pixel distances as the adversarial patch position. The pixel distances are calculated using L1 norm. The procedure is shown in Algorithm 2.

Algorithm 2 Identify the Patch Region and Put Mask on the State

Input: State S with adversarial patch at current time step, Context Re-Constructor F
Output Row and Column position of the upperleft corner: maxI, maxJ
S_{gen} ←get the generate image by $F(S)$
Sum ←2D matrix that will store the sum of difference in each regions
for i in range(W - LengthOfPatch) **do**
    for i in range(H - LengthOfPatch) **do**
        $Sum_{i,j}$ ←the sum of differences of the region whose upperleft corner is in the i-th row and j-th column
    end for
end for
maxI, maxJ ←To find the row and column coordinate of the highest value of Sum
return maxI, maxJ

5.2 Mask Defense

Our method to defend against adversarial patch consists of two main steps. The first step is putting a black mask on the position of the patch on the state, and the second step is using a deep Q-network to predict the optimal action of the masked state.

We use adversarial training to train a robust deep Q-network, which can pick the optimal action when states are partially masked with black mask. We put black masks on states to apply adversarial training on DQN. The loss function for adversarial training consists of two parts: standard loss and adversarial loss. Adversarial loss can be expressed as:

$$\mathcal{L}_{adv} = \sum_{i=1}^{N} \sum_{j \neq m}^{K} \max(Q(m(s_i), a_j) - Q(m(s_i), a_h), -c), \tag{5.3}$$

such that K is the number of choices of action, N is the number of states in the batch, $m(s_i)$ is the state with black mask on, c is a small positive constant, and a_h is the optimal action with highest $Q(s_i, a_j)$. The adversarial loss intends to let the optimal action have the highest Q-value by a margin c. Our loss function is based on the regularization term to train SA-DQN[13]. The total loss $\mathcal{L}$ can be expressed as a combination between the standard temporal loss of deep Q-networks and the adversarial loss:

$$\mathcal{L} = \lambda \mathcal{L}_{std} + (1 - \lambda)\mathcal{L}_{adv}, \tag{5.4}$$

where λ is a balance factor between two types of loss.

The other important part of adversarial training is to choose the location to place the black masks. The easiest way to think of this problem is randomly select a position with equal probability for each sample states. However, after we use this method, we discover that after adversarial training, the classifier has high predication accuracy to choose the optimal action when the mask is not put on the region with high gradients of loss function(equation 4.1) but weak performance when the mask is put on the region with high gradients of loss function during test phase.

To solve this problem, we propose a method: half of the states in the training mini-batch would randomly a choose a position to put the mask on, and other half of the states in the mini-batch would choose a position in a set of high gradient regions to put the mask on. The set of High gradient regions are found by the following procedures: We simulate M number of time steps, and M is usually a large number over 10 thousands. Then we randomly select N states from M time steps, in

Algorithm 3 Find the Set of High Gradient Regions

Simulate M time steps and store the states in a replay buffer
Randomly sample N states from the replay buffer
$grad$ ←Calculate the gradients for N sampled states
G ←Calculate the accumulate gradients over all states using $grad$
for i in range(0, W, SkipSize) **do**
    for j in range(0, H, SkipSize) **do**
        $R_{i,j}$ ←Total accumulate gradients in the squares whose upperleft corner's coordinate is in i-th row and j-th column
    end for
end for
$X_{1:K}$ ←X coordinate of the top K regions with highest accumulate gradients
$Y_{1:k}$ ←Y coordinate of the top K regions with highest accumulate gradients
return X, Y

which N is much smaller than M so that we can input N states into DQN in one batch. The reason that we do not simulate N time steps directly is that, sometimes, an episode of Atari game lasts about 2000 time steps and if N is less than this number, the sampled states would not be representative. Then we calculate the accumulative gradients of regions in the state. The formula to calculate accumulative gradients can be referred to equation 4.2. We would select a set of regions whose accumulative gradients are highest and we referred them as "concentration regions". Concentration regions can overlap with each other. The details to find the set of high gradient regions are in Algorithm 3.

After we get the set of concentration regions, we will use them for adversarial training. In each training batch, we divide samples into two parts, in which has equal number of samples . For the first half of samples, we randomly select positions to put the patch on the states. For the second half of samples, we randomly select positions from the set of concentration regions, and put masks on the those positions for the states. The procedure of generating adversarial states is described in Algorithm 4.

Algorithm 4 Generate Adversarial States

Input: Sample States S, List of x position of upperleft corner of high gradient regions $xPos$, List of x position of upperleft corner of high gradient regions $yPos$

Output: Adversarial sample states with masked

M ←the number of sample states in S
selectRandomXPos ←randomly select x positions for the upperleft corner of the masked region
selectRandomYPos ←randomly select y positions for the upperleft corner of the masked region
for i in range(M/2) **do**
    mask S_i based on selectRandomXPos and selectRandomYPos
end for
selectXPos, selectYPos←randomly select pairs of x and y positions for the upperleft corner of the masked region from the $xPos$ and $yPos$
for i in range(M/2, M) **do**
    mask S_i based on selectXPos and selectYPos
end for
return S

5.3 Recover Defense

Recover defense is also a method of adversarial training, but we use context encoder to fill the corrupted region instead of putting a black mask on it. After we identify the position of the patch by context re-constructor, we fill that region to white and let context encoder to recover that region. We put the recover region into the corresponding region of the adversarial state that we observe, and refer the new state as ”synthesis state”. Then we use the model trained by adversarial training to predict the optimal action of synthesis state”.

We use the standard approach to train the context encoder to recover the white regions of the state[41]. To apply adversarial training on the DQN, We use the same method of selecting masked regions to select regions to fill with white, and let context encoder to recover those regions and make synthesis states for adversarial training. The loss function for recover defense is same as the loss function for mask defense, except that we use synthesis state instead of masked state.

5.4 Experiments of Mask Defense and Recover Defense

We test our adversarial training for mask defense and recover defense in five Atari environments, including Freeway, BankHeist, RoadRunner, Qbert, and Pong, under different ratios of patch to state. In each scenario, we test the defense against 10 ultimate patches, and we simulate 10000 time steps for each ultimate patch. Then we calculate the mean and standard deviation over episodes from all ultimate patches in each scenario.

During adversarial training for both defense methods, to acquire concentration regions, we first simulate $M = 10000$ states and randomly sample 512 states, and calculate their accumulative gradients. Then we select 20 concentration regions for all environments and sizes of the patch. The λ is 0.1 for trade-off between standard loss and adversarial loss. When we simulate states for adversarial training, we choose actions with highest Q-values in four environments, including Freeway, BankHeist, RoadRunner, and Qbert. We set $\epsilon = 0.3$ as the probability to randomly choose an action instead of the action of the highest Q-value for Pong.

For referral, Table 4.1 shows that when the ratio of adversarial patch to the size of state is 0.003, patch attack can reduce the rewards to minimum or very low values in all five environments, and Table 4.2 shows the vanilla rewards of the agents in the environments. We can see that both methods show decent robustness and increase the rewards significantly from the results in Table 5.1. For Freeway and BankHeist, both methods can exhibits similar and high robustness against adversarial patch. In Roadrunner, mask defense outperforms recover defense when the ratio of patch to the states equal or exceed 0.01. In Qbert and Pong, both method outperform each other for some size of patches. While adversarial patch can reduce the reward to a very low value for all environments when its ratio to the state is 0.003, mask defense and recover defense can make DQN robust even when the ratio of patch to the states is 0.05 in some environments, which demonstrates the effectiveness of both defense methods.

Environment	Method	0.003	0.01	0.03	0.05
Freeway	Mask	31.35 ±2.06	29.80 ±2.51	29.85 ±2.91	31.68 ±2.73
Freeway	Recover	31.38 ±1.52	29.82 ±2.29	29.80 ±2.51	31.48 ±2.36
BankHeist	Mask	1316.75 ±31.10	1321.25 ±36.89	1097.33 ±248.94	1265.83 ±66.70
BankHeist	Recover	1294.77 ±60.77	1304.25 ±42.01	1308.92 ±37.69	1295.24 ±55.94
RoadRunner	Mask	27642.19 ±6992.65	31505.66 ±7755.69	27986.89 ±10096.72	30046.77 ±11888.73
RoadRunner	Recover	29395.00 ±10457.25	27367.69 ±9428.63	21968.75 ±7004.01	22965.52 ±7570.37
Qbert	Mask	800.47 ±61.20	474.77 ±352.88	694.52 ±149.82	374.20 ±336.11
Qbert	Recover	810.83 ±20.60	556.85 ±285.01	429.05 ±267.49	577.66 ±294.36
Pong	Mask	15.26 ±10.52	10.19 ±13.98	1.93 ±15.37	-8.06 ±13.68
Pong	Recover	15.10 ±9.10	14.54 ±9.49	7.12 ±14.72	-18.73 ±8.50

Table 5.1: The results of performance of mask defense and recover defense against adversarial patch.

We also test the performance of mask defense and recover defense in the case that random positions are masked or recovered by context encoder in each time step, respectively. The experimental outcomes are presented in Table 5.2. For all environments, both defense can maintain half of the reward without attack when the ratio of masked or recovered region to the state is 0.05.

Environment	Method	0.003	0.01	0.03	0.05
Freeway	Mask	32.28 ±2.46	31.57 ±2.19	31.57 ±2.55	29.5 ±2.55
Freeway	Recover	32.22 ±0.78	33.0 ±1.24	32.66 ±1.15	32.11 ±1.91
BankHeist	Mask	1300.0 ±38.35	1264.66 ±107.07	1226.66 ±90.23	1213.33 ±157.12
BankHeist	Recover	1262.0 ±69.68	1256.0 ±68.73	1255.0 ±85.81	1297.77 ±51.37
RoadRunner	Mask	28871.42 ±10794.14	29245.0 ±6263.50	26609.52 ±9176.93	25108.69 ±9648.19
RoadRunner	Recover	3126.66 ±9760.40	23133.33 ±9818.93	19641.17 ±9006.08	22746.15 ±7469.79
Qbert	Mask	798.93.22 ±43.44	788.72 ±115.71	772.54 ±124.58	751.85 ±150.06
Qbert	Recover	807.81 ±18.15	812.5 ±21.65	790.15±88.96	776.56 ±115.40
Pong	Mask	18.14 ±2.03	18.64 ±1.67	13.90 ±3.08	13.54 ±4.39
Pong	Recover	15.66 ±3.01	19.7 ±1.00	16.33 ±2.78	15.5 ±3.08

Table 5.2: The results of mask defense when random positions are masked in every time step, and results of recover defense when random positions are filled with white and recovered by context encoder in every time step.

Chapter 6: Adversarial Patch Attack on Proximal Policy Optimization

6.1 Attack Method

We propose an similar objective function as attack on deep Q-networks, which intends to decrease the chance to choose a_t is $\pi(a_t, S_i)$, and vice versa. The formula of objective function for attack on proximal policy gradient is expressed as:

$$\mathcal{L} = \sum_{i=1}^{N} \pi(a|m(S_i, P, l)) \odot \pi(a|S_i), \tag{6.1}$$

such that N is the number of states in the training batch, $\pi(a, S_i)$ is the vector of policy for taking actions a under state S_i, P is the patch that we want to train, and l is the position of the patch, $m(S_i, P, l)$ is the state with the patch P on the corresponding position l. Except different loss function, the rest of procedures for attack on proximal policy optimization is same as the attack on deep Q-networks.

The attack on proximal policy optimization is test on Procgen environments, where the pixels are represented by three RGB channels instead of grayscaled image. In this case, the accumulative gradients can be calculated by:

$$G_{i,j} = \sum_{i=1}^{C} abs(\sum_{k=1}^{N} grad_{k,i,j}), \tag{6.2}$$

where C is the number of channels for the pixels.

6.2 Experiments of Attack Method

We generate 10 ultimate patches for each size of patch in each environment, and each ultimate patch is chosen from 3 selection patches. For each ultimate patch, we run 3000 time steps. Table

Environ-ment	Distribution	Nominal	0.005	0.01	0.03	0.05
Coinrun	Train	8.31 ±0.32	4.91±5.00	1.18 ±3.22	3.40 ±4.74	0.30 ±1.71
Coinrun	Eval	6.65 ±0.15	2.56 ±4.36	0.62±2.42	2.73 ±4.45	1.32±3.38
Fruitbot	Train	30.20 ±0.23	3.74 ±9.53	-1.47 ±3.12	-1.38 ±2.76	-1.30 ±2.74
Fruitbot	Eval	26.09 ±0.33	3.07 ±9.10	-1.16 ±2.91	-1.33 ±2.84	-1.44 ±2.76
Jumper	Train	8.69 ±0.11	5.83 ±4.93	1.54±3.61	0.57 ±2.32	1.84 ±3.88
Jumper	Eval	4.22 ±0.16	3.67 ±4.82	2.22 ±4.16	0.31 ±1.74	0.56 ±2.29

Table 6.1: Adversarial patch attack on PPO under different ratios of patch to the size of the image.

Environ-ment	Distribution	Nominal	0.005	0.01	0.03	0.05
Coinrun	Train	8.31 ±0.32	5.23±1.08	3.88 ±1.14	2.77 ±1.05	2.94 ±1.10
Coinrun	Eval	6.65 ±0.15	5.0 ±1.11	4.5 ±1.11	3.68 ±1.10	4.21 ±1.13
Fruitbot	Train	30.20 ±0.23	28.62 ±1.74	17.17 ±2.30	8.60 ±1.77	2.94 ±0.97
Fruitbot	Eval	26.09 ±0.33	19.24 ±2.26	12.90 ±2.32	2.32 ±1.31	2.66 ±1.23
Jumper	Train	8.69 ±0.11	5.55 ±0.82	5.20 ±0.99	3.15 ±1.06	3.15 ±1.06
Jumper	Eval	4.22 ±0.16	4.50 ±1.11	4.8 ±0.99	4.50 ±1.11	4.00 ±0.97

Table 6.2: PGD attack on PPO under different magnitude of perturbations.

6.1 shows the mean and standard deviation of episode rewards over all patches in three Procgen environments: Coinrun, Fruitbot, and Jumper. To compare the performance of adversarial patch with PGD attack, we also apply PGD attack on each environment. Using Theorem 1, we can see adversarial patch outperforms PGD attack under same perturbation bound on Procgen.

Chapter 7: Defense against Adversarial Patch for Proximal Policy Optimization

7.1 Mask Defense and Recover Defense for PPO

We apply the same procedures of both defense for Proximal Policy Optimization as the defense methods for deep Q-network. We train a robust model to predict the policy when the partial state is masked. The adversarial loss for mask defense is expressed as:

$$\mathcal{L}_{adv} = \mathbb{E}_{(s_t,a_t,r_t)}[-\min(\frac{\pi(a_t|m(s_t);\theta)}{\pi(a_t|s_t;\theta_{old})}A_t, clip(\frac{\pi(a_t|m(s_t);\theta)}{\pi(a_t|s_t;\theta_{old})}, 1-\epsilon, 1+\epsilon)A_t)], \quad (7.1)$$

such that s_t is the original state, $m(s_t)$ is the masked state, A_t is the advantage function, ϵ is a hyperparameter.

For recover defense, we use the same adversarial loss, but $m(s_t)$ would represents the regions that we recovered by context encoder. In addition, we provide figures of generating synthesis state for recover defense in Figure 7.1. The upper left state from left is the origin state, and the upper right state is the state with adversarial patch on. The bottom left state is the recovered state from context encoder after we fill the patch region with white color, and the bottom right picture is the synthesis state.

7.2 Experiments of Mask Defense and Recover Defense for PPO

We test this method on three cases for mask defense and recover defense: 1) We put adversarial patches on states. Then we use mask context re-constructor to identify the position of the position

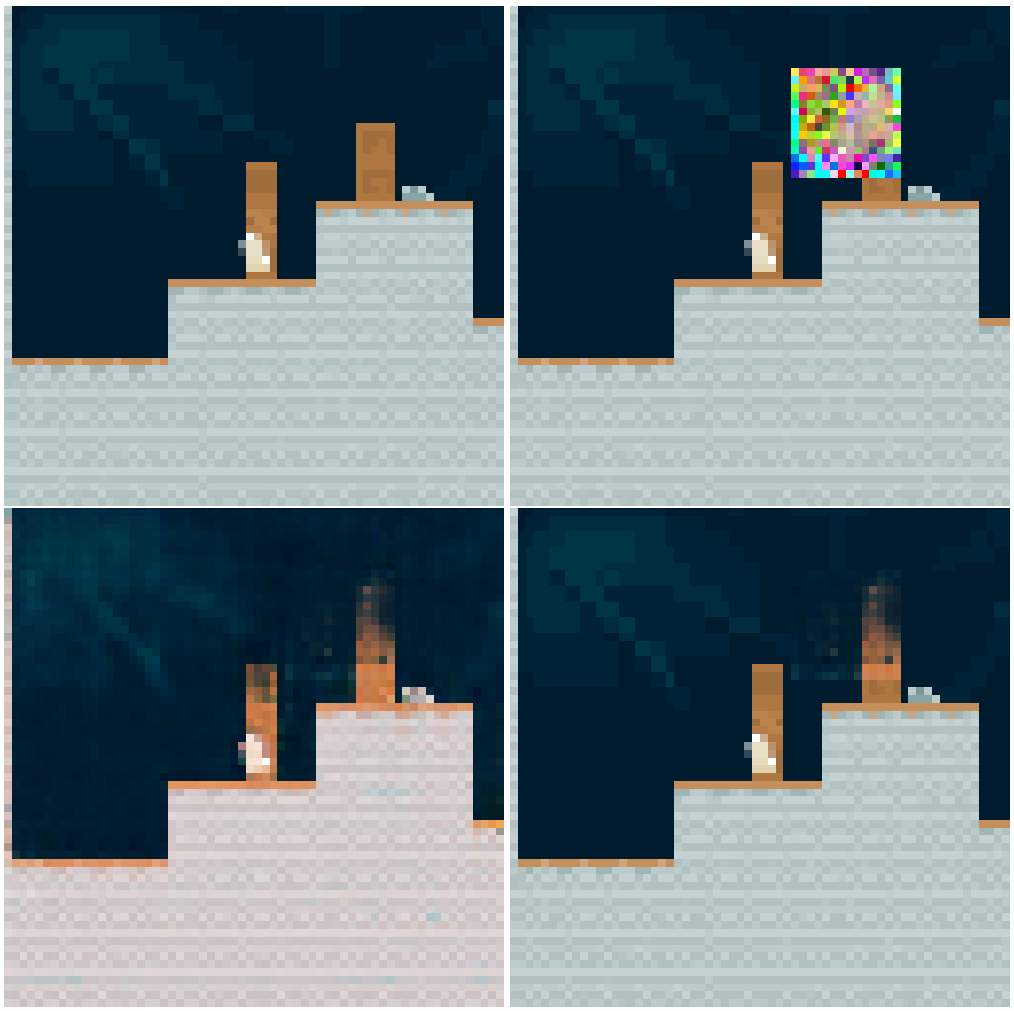


Figure 7.1: Origin state, state with patch on, recovered state, synthesis state, for Coinrun.

and use mask defense and recover defense to defend against them. 2) We assume we know the patch region, and then apply mask defense and recover defense. 3) Different regions are randomly masked or recovered by context encoder, for mask defense and recover defense, respectively. The clean rewards for each environment is shown in Table 7.1.

For the first case, we use 10 ultimate patches to attack PPO model, and each patch attack it for 3000 time steps. At the same time, we apply mask defense or recover defense against the patch attack. To train the robust model, we select 10 concentration regions for selecting attack position during training phase for both defense methods. The results of experiments for defense against adver-

Distribution	Coinrun	Fruitbot	Jumper
train	8.31 ±0.32	30.20 ±0.23	8.69 ±0.11
eval	6.65 ±0.15	26.07 ±0.33	4.22 ±0.16

Table 7.1: Clean Reward for PPO

Environ-ment	Distribution	0.005	0.01	0.03	0.05
Coinrun	Train	8.14 ±3.89	7.84 ±4.11	7.26 ±4.46	8.16 ±3.87
Coinrun	Eval	6.76 ±4.68	6.22 ±4.85	4.85 ±5.00	5.82 ±4.93
Fruitbot	Train	5.71 ±10.52	-0.52 ±5.17	-1.52 ±3.34	-1.01 ±3.64
Fruitbot	Eval	4.17 ±10.47	-0.73 ±4.71	-1.70 ±3.93	-1.33 ±3.86
Jumper	Train	6.35 ±4.81	6.78 ±4.67	4.33 ±4.95	3.45 ±4.76
Jumper	Eval	4.00 ±4.90	4.09 ±4.92	2.94 ±4.56	2.56 ±4.36

Table 7.2: Mask defense with unknown patch position

Environ-ment	Distribution	0.005	0.01	0.03	0.05
Coinrun	Train	7.63 ±4.25	7.10 ±4.54	6.67 ±4.71	7.41 ±4.38
Coinrun	Eval	4.26 ±4.94	5.36 ±4.99	4.50 ±4.97	6.52 ±4.76
Fruitbot	Train	15.31 ±12.38	1.48 ±8.12	-1.28 ±3.63	-0.79 ±3.98
Fruitbot	Eval	11.93 ±13.06	0.96 ±7.92	-1.86 ±4.36	-0.47 ±3.97
Jumper	Train	6.60 ±4.74	6.44 ±4.79	4.17 ±4.93	3.55 ±4.78
Jumper	Eval	3.28 ±4.70	3.06 ±4.61	2.62 ±4.40	3.11 ±4.63

Table 7.3: Recover defense with unknown patch position.

sarial patch are shown in Table 7.2 and Table 7.3 for mask defense and recover defense, respectively.

For the second case, we do not need to use context re-constructor and assume that we know the position of the patch, so that we can make sure that we put the masks in the right regions for the mask defense and recover the right regions for the recover defense. The experimental results are shown in Table 7.4 and Table 7.5. For Coinrun and Jumper, there is no significant change of the rewards for both methods. Bur for Fruitbot, when we know the position of the patch, the reward

Environ-ment	Distribution	0.005	0.01	0.03	0.05
Coinrun	Train	8.24 ±3.81	8.09 ±3.93	8.01 ±3.99	8.05 ±3.97
Coinrun	Eval	6.42 ±4.80	6.51 ±4.77	5.83 ±4.93	5.33 ±4.99
Fruitbot	Train	22.83 ±11.55	15.46 ±12.62	6.45 ±10.31	2.36 ±6.09
Fruitbot	Eval	19.96 ±12.47	17.45 ±12.25	6.40 ±9.14	2.16 ±7.30
Jumper	Train	6.83 ±4.65	7.03 ±4.57	5.77 ±4.94	3.64 ±4.81
Jumper	Eval	4.22 ±4.94	4.38 ±4.96	3.59 ±4.80	3.00 ±4.58

Table 7.4: Mask defense with known patch position

Environ-ment	Distribution	0.005	0.01	0.03	0.05
Coinrun	Train	7.56 ±4.30	7.11 ±4.53	6.93 ±4.61	7.76 ±4.17
Coinrun	Eval	4.00 ±4.90	5.98 ±4.90	4.07 ±4.91	6.62 ±4.73
Fruitbot	Train	24.79 ±10.32	11.95 ±11.63	11.51 ±11.71	4.60 ±8.65
Fruitbot	Eval	23.11±9.60	10.02 ±11.54	8.70 ±10.51	4.20 ±9.04
Jumper	Train	6.99 ±4.59	6.73 ±4.69	5.51 ±4.97	4.15 ±4.93
Jumper	Eval	3.75 ±4.84	3.75 ±4.84	3.53 ±4.78	3.14 ±4.93

Table 7.5: Recover defense with known patch position.

increase dramatically for both methods, which also demonstrates that context re-constructor could not help us to identify the position of the patch accurately for Fuitbot. Recongnizing patch position with context re-constructor works well for defense for most environments that we have test, but the defense results of Fruitbot show that it may not always be the case, which is the limitation of identifying patch position utilizing context re-constructor.

For the third case, we put black masks in random positions on the states to test the performance of mask defense, and fill the random positions with white and recover them with context encoder to test the performance of recover defense. The rewards under random masked regions and random

Environ-ment	Distribution	0.005	0.01	0.03	0.05
Coinrun	Train	8.35 ±0.21	8.33 ±0.21	8.33 ±0.16	8.18 ±0.16
Coinrun	Eval	7.80 ±0.22	7.88 ±0.21	7.76 ±0.18	7.45 ±0.18
Fruitbot	Train	29.57 ±0.74	30.58 ±0.56	28.96 ±0.82	25.30 ±1.05
Fruitbot	Eval	26.38 ±0.97	27.50 ±0.85	26.0 ±0.99	24.75 ±1.05
Jumper	Train	8.24 ±0.21	8.45 ±0.19	8.42 ±0.17	8.52 ±0.16
Jumper	Eval	5.66 ±0.40	5.93 ±0.36	6.08 ±0.35	6.33 ±0.33

Table 7.6: Mask defense with random masked regions.

Environ-ment	Distribution	0.005	0.01	0.03	0.05
Coinrun	Train	8.34 ±0.18	8.09 ±0.15	7.78 ±0.15	5.92 ±0.16
Coinrun	Eval	7.61 ±0.20	7.78 ±0.16	7.19 ±0.16	6.00 ±0.16
Fruitbot	Train	27.14 ±0.95	16.24 ±1.14	12.29 ±1.04	4.96 ±0.61
Fruitbot	Eval	24.92 ±0.99	12.81 ±1.01	9.41 ±0.89	5.46 ±0.69
Jumper	Train	8.16 ±0.17	8.30 ±0.18	6.74 ±0.26	5.26 ±0.43
Jumper	Eval	6.04 ±0.31	6.14 ±0.33	5.77 ±0.38	4.73 ±0.47

Table 7.7: Recover defense with random recovered regions.

recovered regions for mask defense and recover defense are shown in Table 7.6 and Table 7.7, respectively.

Chapter 8: Transferability of Adversarial Patch Attacks

We do experiments on the transferability of adversarial patch attacks on Deep Q-network and Proximal Policy Optimization for blackbox attack. We train a set of new models to explore the transferability of patch attacks from the original set of models. To evaluate the performance of blackbox attack, we also test the performance of whitebox attack and random attack. For random attack, in each time steps, we randomly select a patch region and set values of pixels on the patch uniformly from the range of possible pixel values.

The experimental results of attacks on DQN are shown in Table 8.1, and the experimental results of attacks on PPO are shown in Table 8.3 and 8.4. For whitebox and blackbox attack, we both use 3 ultimate patches for each size of patch and environment, and each ultimate patch run for 10000 steps for DQN and 3000 steps for PPO. From the tables, we can see that transferability of adversarial patch is not guaranteed. The transferability of adversarial patch is strong in some environments, but weak in some other environments.

Environment	Method	0.0005	0.001	0.003	0.005
Freeway	blackbox	22.00 ±1.41	24.83 ±1.62	22.42 ±5.16	22.33 ±1.18
Freeway	whitebox	18.17 ±2.76	16.17 ±1.82	1.58 ±0.76	0.0 ±0.0
Freeway	random	25.25 ±1.78	26.0 ±1.22	24.25 ±1.92	25.0 ±2.23
BankHeist	blackbox	1266.67 ±58.21	1149.33 ±71.41	1094.00 ±66.61	1095.33 ±79.15
BankHeist	whitebox	414.00 ±189.06	230.00 ±36.74	70.53 ±26.05	0.0 ±0.0
BankHeist	random	1335.0 ±8.66	1278.0 ±34.29	1260.0 ±86.89	1050.0 ±70.71
RoadRunner	blackbox	4251.06 ±2490.40	795.74 ±529.13	230.00 ±229.35	103.88 ±81.16
RoadRunner	whitebox	3112.50 ±4887.63	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0
RoadRunner	random	15100.0 ±5068.45	8362.5 ±5049.86	4233.33 ±2783.08	1840.0 ±1158.18
Pong	blackbox	-4.71 ±19.09	-20.72 ±0.56	-20.83 ±0.70	-21.0 ±0.0
Pong	whitebox	6.85 ±17.81	-20.86 ±0.34	-21.00 ±0.00	-21.00 ±0.0
Pong	random	20.6 ±0.79	6.0 ±7.78	-17.16 ±1.21	-18.14 ±0.98

Table 8.1: Attacks on DQN

Environ-ment	Method	0.005	0.01	0.03	0.05
Coinrun	blackbox	4.44 ±4.97	5.29 ±4.99	2.86 ±4.52	2.31 ±4.21
Coinrun	whitebox	4.62 ±4.99	3.85 ±3.87	0.00 ±0.00	0.00 ±0.00
Coinrun	random	8.65 ±3.42	8.56 ±3.51	8.43 ±3.64	8.49 ±3.58
Fruitbot	blackbox	10.74 ±12.20	7.71 ±10.66	0.82 ±7.44	-1.71 ±5.86
Fruitbot	whitebox	4.31 ±7.59	-1.46 ±2.61	-0.94 ±2.32	-1.39 ±2.23
Fruitbot	random	31.12 ±6.73	28.68 ±10.17	17.03 ±13.22	19.43 ±12.34
Jumper	blackbox	6.25 ±4.84	4.29 ±4.95	1.67 ±3.73	2.50 ±4.33
Jumper	whitebox	3.68 ±4.82	0.83 ±2.76	0.0 ±0.0	0.71 ±2.58
Jumper	random	8.43 ±3.64	8.47 ±3.60	7.86 ±4.10	8.11 ±3.92

Table 8.2: Attacks on PPO in Training Distribution

Environ-ment	Method	0.005	0.01	0.03	0.05
Coinrun	blackbox	5.50 ±4.97	4.67 ±4.99	4.00 ±4.90	2.50 ±4.33
Coinrun	whitebox	5.39 ±4.99	1.11 ±3.14	0.00 ±0.00	0.00 ±0.00
Coinrun	random	8.14±3.89	7.82 ±4.13	7.90 ±4.07	6.88 ±4.63
Fruitbot	blackbox	10.38 ±12.98	9.22 ±12.64	0.00 ±5.11	2.39 ±6.92
Fruitbot	whitebox	1.56 ±5.45	-1.05 ±2.51	-1.13 ±2.67	-0.79 ±2.23
Fruitbot	random	25.00 ±10.90	20.36 ±12.78	15.62 ±13.16	18.10 ±12.68
Jumper	blackbox	4.35 ±4.96	2.14 ±4.10	0.83 ±2.76	2.14 ±4.10
Jumper	whitebox	3.53±4.78	2.14 ±4.10	1.43 ±3.50	1.00 ±3.00
Jumper	random	6.33 ±4.82	6.45 ±4.78	6.48±4.78	6.41 ±4.80

Table 8.3: Attacks on PPO in Evaluation Distribution

Chapter 9: Conclusion and Outlook

In this work, we demonstrates that adversarial patch can effectively attack deep q-networks and proximal policy optimization when the size of patch is relative small to the size of the state. In addition, we show that Context Re-Constructor can reconstruct the state and help to identify the patch position. Then we can use mask defense and recover defense to choose actions for the agent. Lastly, we test the transferability of adversarial patch attacks, and the results show that the transferability is varied on different environments.

Despite the fact that using Context Re-Constructor to identify the position of patch for defense works well for the most experiments, Context Re-Constructor is not a panacea, and experiments demonstrate that it fails to identify the position of patch for Fruitbot accurately and there is a significantly difference of rewards when the position of patch is known instead of unknown. In addition, recover defense and mask defense both relies on the knowledge of the position of patch. The future research can focus on developing method that enable to identify the position of patch accurately in more environments, and the future research also can design defense mechanisms that do not require knowing the position of the patch. We leave these limitations to future works.

References

1. Sallab, A. E., Abdou, M., Perot, E. & Yogamani, S. Deep Reinforcement Learning framework for Autonomous Driving. *Electronic Imaging* **29,** 70–76. doi:10.2352/issn.2470-1173.2017.19.avm-023 (2017) (cited on p. 1).
2. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M. *Playing Atari with Deep Reinforcement Learning* 2013. doi:10.48550/ARXIV.1312.5602 (cited on pp. 1, 3).
3. Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. & Wierstra, D. *Continuous control with deep reinforcement learning* 2019. arXiv: 1509.02971 [cs.LG] (cited on p. 1).
4. Huang, S. H., Papernot, N., Goodfellow, I. J., Duan, Y. & Abbeel, P. Adversarial Attacks on Neural Network Policies. *CoRR* **abs/1702.02284.** arXiv: 1702.02284 (2017) (cited on pp. 1, 7, 8, 21).
5. Behzadan, V. & Munir, A. Whatever Does Not Kill Deep Reinforcement Learning, Makes It Stronger. *CoRR* **abs/1712.09344.** arXiv: 1712.09344 (2017) (cited on pp. 1, 11).
6. Pattanaik, A., Tang, Z., Liu, S., Bommannan, G. & Chowdhary, G. Robust Deep Reinforcement Learning with Adversarial Attacks. *CoRR* **abs/1712.03632.** arXiv: 1712.03632 (2017) (cited on pp. 1, 12).
7. Kos, J. & Song, D. *Delving into adversarial attacks on deep policies* 2017. doi:10.48550/ARXIV.1705.06452 (cited on pp. 1, 8).
8. Lin, Y., Hong, Z., Liao, Y., Shih, M., Liu, M. & Sun, M. Tactics of Adversarial Attack on Deep Reinforcement Learning Agents. *CoRR* **abs/1703.06748.** arXiv: 1703.06748 (2017) (cited on pp. 1, 8, 23).
9. Sun, J., Zhang, T., Xie, X., Ma, L., Zheng, Y., Chen, K. & Liu, Y. Stealthy and Efficient Adversarial Attacks against Deep Reinforcement Learning. *CoRR* **abs/2005.07099.** arXiv: 2005.07099 (2020) (cited on pp. 1, 9).
10. Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T. & Song, D. *Robust Physical-World Attacks on Deep Learning Models* 2018. arXiv: 1707.08945 [cs.CR] (cited on p. 1).

11. Boloor, A., He, X., Gill, C., Vorobeychik, Y. & Zhang, X. *Simple Physical Adversarial Examples against End-to-End Autonomous Driving Models* 2019. arXiv: 1903.05157 [cs.RO] (cited on p. 1).

12. Brown, T. B., Mané, D., Roy, A., Abadi, M. & Gilmer, J. Adversarial Patch. *CoRR* **abs/1712.09665.** arXiv: 1712.09665 (2017) (cited on pp. 1, 15).

13. Zhang, H., Chen, H., Boning, D. S. & Hsieh, C. Robust Reinforcement Learning on State Observations with Learned Optimal Adversary. *CoRR* **abs/2101.08452.** arXiv: 2101.08452 (2021) (cited on pp. 1, 9, 29).

14. Oikarinen, T. P., Weng, T. & Daniel, L. Robust Deep Reinforcement Learning through Adversarial Loss. *CoRR* **abs/2008.01976.** arXiv: 2008.01976 (2020) (cited on pp. 1, 12).

15. Wu, J. & Vorobeychik, Y. *Robust Deep Reinforcement Learning through Bootstrapped Opportunistic Curriculum* 2022. doi:10.48550/ARXIV.2206.10057 (cited on pp. 1, 13).

16. Bellemare, M. G., Naddaf, Y., Veness, J. & Bowling, M. The Arcade Learning Environment: An Evaluation Platform for General Agents. *CoRR* **abs/1207.4708.** arXiv: 1207.4708 (2012) (cited on p. 3).

17. Wang, Z., de Freitas, N. & Lanctot, M. Dueling Network Architectures for Deep Reinforcement Learning. *CoRR* **abs/1511.06581.** arXiv: 1511.06581 (2015) (cited on p. 4).

18. Schulman, J., Levine, S., Moritz, P., Jordan, M. I. & Abbeel, P. *Trust Region Policy Optimization* 2017. arXiv: 1502.05477 [cs.LG] (cited on p. 4).

19. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. *Proximal Policy Optimization Algorithms* 2017. arXiv: 1707.06347 [cs.LG] (cited on p. 4).

20. Goodfellow, I. J., Shlens, J. & Szegedy, C. *Explaining and Harnessing Adversarial Examples* 2014. doi:10.48550/ARXIV.1412.6572 (cited on pp. 5, 6).

21. Kurakin, A., Goodfellow, I. J. & Bengio, S. Adversarial examples in the physical world. *CoRR* **abs/1607.02533.** arXiv: 1607.02533 (2016) (cited on p. 5).

22. Moosavi-Dezfooli, S., Fawzi, A. & Frossard, P. DeepFool: a simple and accurate method to fool deep neural networks. *CoRR* **abs/1511.04599.** arXiv: 1511.04599 (2015) (cited on p. 5).

23. Kurakin, A., Goodfellow, I. J. & Bengio, S. Adversarial Machine Learning at Scale. *CoRR* **abs/1611.01236.** arXiv: 1611.01236 (2016) (cited on p. 6).

24. Cai, Q., Du, M., Liu, C. & Song, D. Curriculum Adversarial Training. *CoRR* **abs/1805.04807.** arXiv: 1805.04807 (2018) (cited on p. 6).

25. Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Arandjelovic, R., Mann, T. A. & Kohli, P. On the Effectiveness of Interval Bound Propagation for Training Verifiably Robust Models. *CoRR* **abs/1810.12715.** arXiv: 1810.12715 (2018) (cited on p. 7).

26. Raghunathan, A., Steinhardt, J. & Liang, P. Certified Defenses against Adversarial Examples. *CoRR* **abs/1801.09344.** arXiv: 1801.09344 (2018) (cited on p. 7).

27. Carlini, N. & Wagner, D. A. Towards Evaluating the Robustness of Neural Networks. *CoRR* **abs/1608.04644.** arXiv: 1608.04644 (2016) (cited on pp. 8, 9, 23).

28. Zhang, H., Chen, H., Xiao, C., Li, B., Boning, D. S. & Hsieh, C. Robust Deep Reinforcement Learning against Adversarial Perturbations on Observations. *CoRR* **abs/2003.08938.** arXiv: 2003.08938 (2020) (cited on pp. 9, 10).

29. Hussenot, L., Geist, M. & Pietquin, O. Targeted Attacks on Deep Reinforcement Learning Agents through Adversarial Observations. *CoRR* **abs/1905.12282.** arXiv: 1905.12282 (2019) (cited on p. 9).

30. Russo, A. & Proutière, A. Optimal Attacks on Reinforcement Learning Policies. *CoRR* **abs/1907.13548.** arXiv: 1907.13548 (2019) (cited on p. 10).

31. Gleave, A., Dennis, M., Kant, N., Wild, C., Levine, S. & Russell, S. Adversarial Policies: Attacking Deep Reinforcement Learning. *CoRR* **abs/1905.10615.** arXiv: 1905.10615 (2019) (cited on p. 10).

32. Shen, Q., Li, Y., Jiang, H., Wang, Z. & Zhao, T. Deep Reinforcement Learning with Smooth Policy. *CoRR* **abs/2003.09534.** arXiv: 2003.09534 (2020) (cited on p. 10).

33. Fortunato, M. *et al.* Noisy Networks for Exploration. *CoRR* **abs/1706.10295.** arXiv: 1706.10295 (2017) (cited on p. 11).

34. Fischer, M., Mirman, M., Stalder, S. & Vechev, M. T. Online Robustness Training for Deep Reinforcement Learning. *CoRR* **abs/1911.00887.** arXiv: 1911.00887 (2019) (cited on p. 13).

35. Lütjens, B., Everett, M. & How, J. P. Certified Adversarial Robustness for Deep Reinforcement Learning. *CoRR* **abs/1910.12908.** arXiv: 1910.12908 (2019) (cited on p. 14).

36. Wu, F., Li, L., Huang, Z., Vorobeychik, Y., Zhao, D. & Li, B. CROP: Certifying Robust Policies for Reinforcement Learning through Functional Smoothing. *CoRR* **abs/2106.09292.** arXiv: 2106.09292 (2021) (cited on p. 14).

37. Gittings, T., Schneider, S. A. & Collomosse, J. P. Robust Synthesis of Adversarial Visual Examples Using a Deep Image Prior. *CoRR* **abs/1907.01996.** arXiv: 1907.01996 (2019) (cited on p. 15).

38. Hayes, J. On visible adversarial perturbations digital watermark- ing. *CVPR Workshops* **1597–1604** (2018) (cited on p. 16).

39. Naseer, M., Khan, S. H. & Porikli, F. *Local Gradients Smoothing: Defense against localized adversarial attacks* 2018. doi:10.48550/ARXIV.1807.01216 (cited on p. 16).

40. Wu, T., Tong, L. & Vorobeychik, Y. *Defending Against Physically Realizable Attacks on Image Classification* 2019. doi:10.48550/ARXIV.1909.09552 (cited on p. 18).

41. Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T. & Efros, A. A. Context Encoders: Feature Learning by Inpainting. doi:10.48550/ARXIV.1604.07379 (2016) (cited on pp. 26, 31).