DISTILLED AGENT DQN FOR PROVABLE ADVERSARIAL ROBUSTNESS

Anonymous authors
Paper under double-blind review

ABSTRACT

As deep neural networks have become the state of the art for solving complex reinforcement learning tasks, susceptibility to perceptual adversarial examples have become a concern. The transferability of adversarial examples for neural networks is known to enable attacks capable of tricking the agent into bad states or into not being able to learn at all. In this work we demonstrate a simple poisoning attack able to fool DQNs when trained with defense methods commonly used for classification tasks. We also propose an algorithm, called DadQN, which is based on deep Q-networks and enables the use of stronger defenses, including defenses increasing provability.

1 INTRODUCTION

To ensure Reinforcement Learning (RL) behaves reliably in the real world, it is important to consider settings where an adversary aims to actively hinder the learning process of the agent. While prior work has explored robustness in the setting of discrete RL (Morimoto & Doya, 2005; Boyan & Moore, 1995), most recent progress in the field has focused on dealing with continuous states by using neural networks (Sutton et al., 2000; Mnih et al., 2013; Peters & Schaal, 2008).

In this work we present a new approach for training RL systems to be more (provably) robust. The key idea is to decouple the DQN network architecture into a policy (student) network S and a Q-network in a way which allows us to robustly train the policy network and use it for exploration while at the same time preserving the standard way in which the Q network is trained. We then show how to naturally incorporate state of the art defenses, developed in the context of deep supervised, learning to the reinforcement learning setting by training the student network in two ways: (i) via adversarial training with methods such as FGSM where we generate adversarial states that decrease the chance the optimal action is selected, and (ii) via provably robust training with symbolic methods which guarantee the network will select the right action in a given state despite any possible perturbation (within a range) of that state.

Contributions Our main contributions are:

- DadQN: a deep RL algorithm explicitly designed to survive poisoning and testing attacks.
- A poisoning attack, illustrated in Figure 1, that hinders learning *and* test performance of ordinary DQNs.
- A method to incorporate both, state-of-the-art adversarial training as well as provably robust training into our RL algorithm. To our best knowledge, this is the first time, on-line robustness certification has been achieved with a deep RL agent.
- An evaluation demonstrating that DadQN can defend against adversarial attacks when defenses against DQNs catastrophically fail, while being of comparable performance when adversarial attacks are not present.

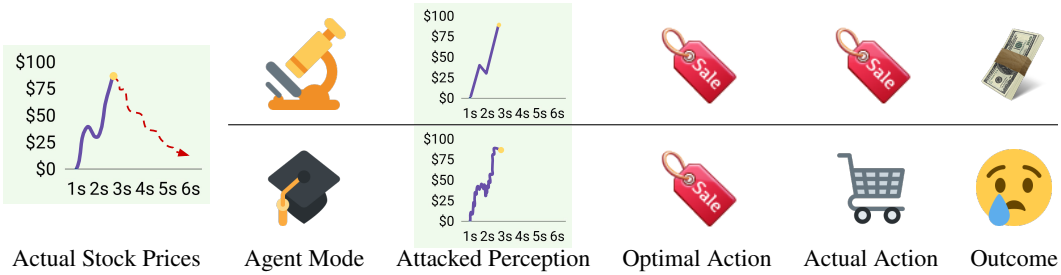

Figure 1: An illustration of the UQP Poisoning Attack on a hypothetical stock trading agent.[2]

2 Background

The goal of reinforcement learning is to determine a behavior policy[1] $\pi(a \mid s)$ for a given game $G\colon \text{State} \times \text{Action} \to \text{State} \times \mathbb{R}$, that maximizes the expected discounted sum of future rewards $Q^\pi(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^t r_t | s_1 = s, a_1 = a, \pi\right]$, where $\gamma \in [0, 1]$ is a discount factor and $s_{i+1}, r_i = G(s_i, a_i)$. The objective of Q-Learning (Watkins & Dayan, 1992) is to learn the function $Q^*(s)_a = \max_\pi Q^\pi(s)_a$ and construct a policy by greedily picking its optimal action.

2.1 Deep Q Learning

In Deep Q-Learning (Mnih et al., 2015; 2013) the goal is to progressively approximate Q^* by weights θ_i for a neural network $Q(s; \theta_i)_a$ at iteration i. Samples are generated and stored in an experience replay buffer $D_i = \{e_1, \ldots, e_i\}$ where $e_i = (s_i, a_i, r_i, s_{i+1})$ is generated using a sufficiently random agent based on the approximation $Q(s; \theta_i)$.

Integral to the DQN algorithm is the use of a network with lagging weights $\theta_i^- = \theta_{k\lfloor \frac{i}{k} \rfloor}$ for some k. The loss at iteration i is described by

$$L_i = \mathbb{E}_{(s,a,r,s') \sim U(D_i)}[(r + \gamma \max_{a'} Q(s'; \theta_i^-)_{a'} - Q(s; \theta_i)_a)^2]$$

A few iterations of stochastic gradient descent are used to minimize L_i and produce θ_{i+1}, although often multiple exploration steps are used first.

2.2 Adversarial Attack

Unlike traditional reinforcement learning, the presence of adversarial attacks introduces the complication that the game has some level of knowledge about the agent playing the game and can act in specific ways knowing its behavior. The environment that our algorithm is designed to address is given by our definition of an *agent aware game*.

Agent Aware Game In agent aware reinforcement learning, the game G not only has access to the agent's choices, but also has access to the learning system and the agent itself. Thus, an adversarial agent's aim is to play the game in such a way as to poison learning, but also potentially play differently when it is aware if the RL system is training or testing.

We consider only *manipulative* agent aware games, defined to be games

$$G\colon \text{State} \times \text{Action} \times \text{Agent} \to \text{State} \times \mathbb{R}$$

which are composed of a core game $H\colon \text{State} \times \text{Action} \to \text{State} \times \mathbb{R}$ and a manipulative adversary $M\colon \text{State} \times \text{Agent} \to \text{State}$ such that $G(s_i, a_i, \pi) = (M(s_{i+1}^-, \pi), r_i)$ where $(s_{i+1}^-, r_i) = H(s_i, a_i)$, and such that for any π where $s_{i+1} = M(s_{i+1}^-, \pi)$ it is true that $\forall a. H(s_{i+1}, a) = H(s_{i+1}^-, a)$. In other words, manipulation does not effect how the core game works or is played.

[1] We write $\pi(s)$ to mean vector of scores for each action in the deterministic greedy policy where $\arg\max_a \pi(s)_a$ is the action taken. When written this way, it is assumed that π is a neural network.

[2] tag sale by PetalArt, money by DesignContest, crying face, microscope, shopping cart and graduation cap by Twitter, Inc., are licensed under CC BY 4.0.

The attacks we consider are L_∞ ϵ-attacks (not to be confused with the ϵ used for greedy exploration). Specifically $M(s, \pi) \in \mathbb{B}_\epsilon(s) = \{s' \mid ||s - s'||_\infty \leq \epsilon\}$, that is, $\mathbb{B}_\epsilon(s)$ is an ϵ sized L_∞ ball.

We consider both, test time attacks, and attacks that change their behavior when the agent is training.

2.3 Testing Attacks

Huang et al. (2017) attack a network policy trained by reinforcement learning, at testing time and find that traditional untargeted attacks such as FGSM are sufficient to significantly reduce the performance of the policy produced by DQN, A3C and TRPO. Lin et al. (2017b) used targeted attacks to "enchant" the agent into performing specific actions. We evaluate only with untargeted FGSM attacks.

2.4 Training Attacks

Attacking the training procedure such that the model never learns is known as poisoning Yang et al. (2017); Steinhardt et al. (2017); Biggio et al. (2012). Behzadan & Munir (2017a) introduce an attack used to prevent the DQN from ever learning the correct Q function. In their attack, they do not assume the attacker has direct access to θ_Q and instead train a version of it, $\theta_{Q'}$, in parallel. They learn an "adversarial policy" which picks an action a' far from the optimal one and then they perturb the observed next-state s' to cause $Q(s'; \theta_{\overline{Q'}})$ to be maximized (they use the non Double DQN variant) for the action a' in hopes the attack transfers to the intended DQN.

We introduce an attack, Untargeted Q-Poisoning (UQP), which does not need to train additional networks, and which has access to the agent's network. We allow the attack the to switch its behavior when the agent is being tested in order to simulate the inevitable asymmetry between production and development environments and common subsequent dysfunction it is known to cause. In UQP during training time, an attack state is chosen in the style of FGSM to reinforce the decision of the agent policy, thus often creating an illusion of highly successful training. During test time, standard FGSM is used where the "classification" is chosen by the policy. We define it as a manipulative adversary:

$$M(s, \pi) = \begin{cases} s - \alpha \text{sign}(\nabla_s \mathcal{H}(\pi(s), \arg\max_a \pi(s)_a)) & \pi(s) \text{ is learning} \\ s + \alpha \text{sign}(\nabla_s \mathcal{H}(\pi(s), \arg\max_a \pi(s)_a)) & \text{otherwise.} \end{cases}$$

where $\mathcal{H}$ is the cross entropy between the probability distribution $\pi(s)$, and the optimal action (encoded as one hot vector) from that probability distribution. This attack is visualized in Figure 1. Here the adversary is aware of whether the agent is training or being actively used, and is able to change the agent's perception of the stock histories to convince it to buy when it should have sold.

3 Our Approach

In the setting of supervised learning, one successful defense is that of Madry et al. (2018). This method uses adversarial examples generated by the PGD attack as training examples (instead of the original examples). While it is experimentally effective, it provides no guarantees about the resulting network's robustness. Katz et al. (2017) provided the first system which could (for very small networks) determine local-robustness, that is, at test time the network is proven to be either ϵ-robust for an individual image. Gehr et al. (2018) used abstract interpretation to scale the local robustness analysis by sacrificing completeness. Raghunathan et al. (2018) introduced a technique to train networks to be certifiably robust (though again limited to small networks). Multiple recent techniques have been developed since to train increasingly larger networks to be certifiably robust including Wong & Kolter (2018); Mirman et al. (2018); Dvijotham et al. (2018); Wong et al. (2018). These techniques represent different points in the spectrum of classification accuracy, certifiable robustness, experimental robustness, and training speed.

In contrast to deep supervised learning, deep reinforcement learning is a harder, more resource intensive task, even for seemingly simple problems. As such, it is important that any proposed defense mechanism is efficient and usable in an online setting and induce as little perturbation as possible to standard network training dynamics.

While Pattanaik et al. (2017) used adversarial examples to train an agent and improve its experimental robustness, Behzadan & Munir (2017b) noticed that often doing this prevents learning. Behzadan & Munir (2017a) further demonstrated that this effect could be intentionally amplified to prevent learning intentionally on the part of an adversary. Gu et al. (2018) demonstrated a technique for training the A3C algorithm to be adversarially robust by training a parallel adversarial actor-critic pair for each protagonist actor-critic pair. The analysis of their technique is limited to noisy environment adversaries, and not intentional agent-aware adversaries such as FGSM. Similarly, Ferdowsi et al. (2018) created an adaptive adversary to improve robustness.

While some progress has been made towards making deep RL experimentally robust, so far there has been less work on certifiable robustness. Towards this, we propose a method to leverage the base Q learning algorithm's ability to learn the correct Q function given a sufficiently random exploration agent (Even-Dar & Mansour, 2002; Bertsekas, 2008; Tsitsiklis, 1994; Watkins & Dayan, 1992).

3.1 Distilled Agent DQN (DadQN)

Rusu et al. (2015) first described the method of improving the learned policy in Deep-Q Learning using *Policy Distillation* (PD). In this technique a Q-approximation is first learned by the standard DQN algorithm. New games are then played using Q as a greedy policy, and the states s are recorded. A student network S is then trained on these states to mimic the behavior of Q. Much like PD, our method involves distilling a policy network from Q. Standard PD only uses the distilled policy during test time, and has so far only been tested in this way. Here we introduce DadQN which trains the student policy S at the same time as it is learning the Q network and uses the student policy instead for exploration. By decoupling the policy network from the Q network, we are able to train the policy with additional defensive constraints and use it for the exploration agent in addition to testing with it. Given a loss L_D and learning rates $\alpha_Q, \alpha_S \in (0, 1)$ our algorithm is as follows:

Algorithm 1 DadQN pseudocode

Initialize a state s, weights θ_Q, θ_S
for $i = 0, \ldots$ **do**
  for $j = 0, \ldots, n$ **do**
    Pick an an action a with a fair strategy (ex. ϵ-greedy) with π_{θ_S} based on $S(s; \theta_S)$
    Play the game $(s', r) := G(s, a, \theta_S)$
    Store (s, a, r, s') in D
    $s := s'$
    if $i \mod k == 0$ **then**
      $\theta_Q^- := \theta_Q$
    end if
    Pick a batch $\mathcal{D} \sim U(D)$
    Train the underlying Q:
      $\theta_Q := \theta_Q - \alpha_Q \nabla_{\theta_Q} \sum_{(s,a,r,s') \in \mathcal{D}} (r + \gamma Y - Q(s; \theta_Q)_a)^2$
    Train the student S from Q:
      $\theta_S := \theta_S - \alpha_S \nabla_{\theta_S} \sum_{(s,a,r,s') \in \mathcal{D}} \left[L_D(s, \theta_Q, \theta_S) \right]$
  end for
end for

Here $Y \equiv Q(s'; \theta_Q^-)_{\arg\max'_a Q(s'; \theta_Q)'_a}$ is the Double DQN (Van Hasselt et al., 2016) next Q estimate.

The intuition behind this algorithm is that the loss for the Q function has not been changed and thus after being trained on sufficiently many random paths, the Q function will approach Q^* regardless of what the student learns. Assuming the student will learn no matter what it is initialized with, the student should be able to handle the concept-shift of Q learning and incorporate the provided constraint. Presumably, the additional constraints allow the student to achieve a better score when playing, so the student should explore higher reward paths.

While L_D could potentially be any optimizable function, we evaluate with either one of the defensive loss described in Section 3.2, or an undefended mean squared error (MSE) loss L_{MSE}. While Rusu et al. (2015) observed that KL divergence works better than MSE and negative-log-likelihood experimentally, we found MSE to be sufficient, simple and common enough for our evaluation.

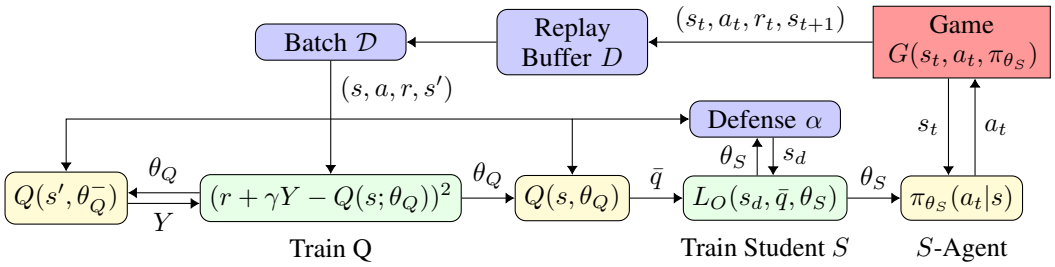

Figure 2: The learning and defense pipeline for DadQN. Yellow boxes have agents, green have losses to be trained, and purple boxes are algorithmic. The game can not be modified and is thus red.

Wang et al. (2015) observed better learning performance by introducing a specific Q network architecture, the Dueling DQN, which is split into two components: an advantage network $A(s; \theta_Q) : \mathbb{R}^m$ computing the relative advantage of the m actions, and a value network $V(s; \theta_Q) : \mathbb{R}$. These are combined as $Q(s; \theta_Q) = V(s; \theta_Q) + (A(s; \theta_Q) - \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} A(s; \theta_Q)_a)$, where $\mathcal{A}$ denotes the action space. The advantage and value are defined to share early network layers. We note that the output of A alone is sufficient to replicate a greedy policy based on Q. However, we observe that the value network may be able to discriminate between states earlier on in the learning than the advantage network. We thus train both the advantage network and value network of the student from the Q independently with the same loss, defining $L_{\text{MSE, Dueling}}$ for use as L_D in Algorithm 1 as follows:

$$L_{\text{MSE, Dueling}}(s, \theta_Q, \theta_S) = ||A_Q(s; \theta_Q) - A_S(s; \theta_S)||_2^2 + ||V_Q(s; \theta_Q) - V_S(s; \theta_S)||_2^2. \quad (1)$$

The Exploration Agent Originally, randomization in exploration was accomplished for the DQN using ϵ-greedy search (Watkins, 1989). Here, the action $\arg\max_a S(s; \theta_S)_a$ is used with probability $1 - p$ and an action is picked uniformly with probability p at every time step (we use p instead of ϵ so to not overload ϵ used in attacks). If $p \geq c$ for a constant $c > 0$, then standard Q learning is known to converge. p is decreased linearly with the number of frames played until it reaches a fixed c.

While this method has good theoretical guarantees, Fortunato et al. (2017) noticed improvements by letting the Q-network learn a mean and variance on noise for the weights in its dense layers. We experiment with the same idea here, however only the student network S learns with noise. We notice that because the student network lags behind the Q network, it might require more diverse samples even when Q has sufficiently learned the correct behavior. We thus introduce an exploration noise constant $\eta \geq 1$, which we multiply with the learned weight variances at exploration time.

3.2 Defending DadQN

The primary motivation for decoupling the DQN network into a policy-student network S and a Q-network is to allow one to leverage additional constraints on the student network without affecting the learning of the correct Q function (provided that exploration is sufficiently noisy). We address the problem of adversarial robustness in reinforcement learning by improving S's robustness. Specifically, we split L_D into a defense α and a defense loss L_O. The defense α uses θ_S and the sample s to produce a *defense goal* s_d (which could be a concrete or symbolic) which is then passed to L_O.

$$L_D(s, \theta_Q, \theta_S) = L_O(s, s_d, Q(s, \theta_Q), \theta_S).$$

The entire DadQN described by Algorithm 1 including the defense is outlined in Figure 2. Here it can be observed that the Q network is used to train the student network, but the defense is only applied to the student network so as to not disturb the Q network's learning dynamics. The defended student network is used to play the game, which may be attacked, and collect learning experiences.

Adversarial Training A variety of techniques have been developed for increasing the robustness of neural networks, typically by training with adversarial examples (Tramèr et al., 2017; Shaham et al., 2015; Madry et al., 2018). Rather than providing the Q network with adversarial examples as in Mandlekar et al. (2017), we provide them (in this case, as concrete defense goals) s_d to the student:

$$\alpha_{\epsilon,\text{FGSM}}(\theta_S, s) = \text{FGSM}_\epsilon(s, \arg\max_a S(s; \theta_S)_a, \theta_S)$$

Here, FGSM_ϵ produces a concrete adversarial example s_d in the ϵ sized L_∞ ball around s that makes the best action in s as least likely to be selected in s_d as possible (that is, it is an untargeted attack).

Finally, we use the traditional MSE loss function:

$$L_{O,\text{MSE}}(s, s_d, \bar{q}, \theta_S) = ||\bar{q} - S(s_d; \theta_S)||_2^2.$$

In practice, we use this loss with probability p, the rest of the time using the non-attacked point s instead of s_d. For Dueling DQNs we calculate the advantage and value separately as in Equation 1.

Provable Robustness Training In addition to adversarial training, we train and certify the robustness of our networks with the DiffAI framework introduced by Mirman et al. (2018) using its Interval domain. This framework has been shown capable of training networks on the scale of DQNs used in Atari with minimal speed and memory overheads over undefended SGD. Additionally, it has been shown to decrease the accuracy of networks very little, which is essential to reinforcement learning where unstable dynamics can be an issue. Formally, we first create the abstract (symbolic) defense goal $s_d = \mathbb{B}_\epsilon(s)$. We then use DiffAI to soundly propagate the defense goal s_d through the network S (via symbolic computation), obtaining a symbolic element g_f as a result. Finally, DiffAI defines a differentiable loss $L_I\colon \text{Interval} \times \mathbb{N} \to \mathbb{R}$ which takes as input the final element g_f and a target (action in our case). The loss has the property that for some target t if $L_I(g_f, t) \leq 1$ then $\forall s' \in \mathbb{B}_\epsilon(s). \arg\max_a S(s'; \theta_S)_a = t$. That is, in this case, we have proved that any element inside the ϵ sized L_∞ ball around s will be classified to the action t by the student network. We define our defensive loss as a combination of this loss and the adversarial loss described earlier for a concrete α:

$$L_{O,\text{Interval}}(s, s_d, \bar{q}, \theta_S) = L_{O,\text{MSE}}(s, \alpha(\theta_S, s), \theta_S) + \lambda L_I(g_f, \arg\max_a \bar{q}_a)$$

where $\lambda \geq 0$ is the constant with which we want to prioritize the DiffAI loss.

4 Experimental Evaluation

We now present our experimental results comparing DadQN to existing methods. We note that while it may be beneficial to *intentionally* train one's own DQN by simulating an adversarially attacking manipulator (in an attempt to defend it), there are learning time attacks which prevent this from being effective. Thus, central to our analysis is the notion that the agent does not know whether it will be attacked during training and with what attack, or whether it will be attacked during testing and with what attack. We show that in each situation, DadQN is capable of providing a stronger or equally strong a defense as existing work, and virtually always avoids the worst case failures. We also show the first improvement in provable robustness in DQNs by using DiffAI as a training defense.

4.1 Experimental Setup

We tested with 3 Atari games (Bellemare et al., 2013) from the OpenAI Gym (Brockman et al., 2016): RoadRunner (RR), Pong and Boxing. Every 10 episodes we play a validation game. In a these games we verify the decision made by the network using the Box domain. We binary-search for the largest ϵ such that the decision is safe. Validation games alternate between never attacking, or attacking with some probability every timestep. Each experiment was run for 4 million frames.

To compare DadQN with DQNs, we implemented a variety of extensions known to increase training performance when enabled in concert as in RainbowDQN (Hessel et al., 2017): Priority Replay (Schaul et al., 2015), DoubleDQN (Van Hasselt et al., 2016), DuelingDQN (Wang et al., 2015), and NoisyNet (Fortunato et al., 2017) using noise constant $\eta = 2$ as described in Section 3.1. During these episodes we disable noise due to NoisyNet and use ϵ-greedy exploration with $\epsilon = 0.005$.

Due to resource-constraints we choose learning parameters for the three games that yielded much faster training than the parameters reported by RainbowDQN [3]. The full set of hyper-parameters can be found in Tables S1 and S2 in the Appendix.

We implemented both DadQN and DQN in PyTorch (Paszke et al., 2017) asynchronously (Mnih et al., 2016). Depending on the parameters and the hardware one run took between 4 to 30 hours,

[3] Hessel et al. (2017) report parameters which are a compromise between many games.

Table 1: Validation game reward comparison for UQP poisoning attack.

			Untargeted Quality Poisoning Training Attack			
Game	Test Attack	DQN	DQN	DadQN	DadQN + FGSM Def	DadQN + DiffAI
RR	none	46821.66	17940.54	25227.04	30513.18	18903.09
	FGSM$^{p=0.4}$	15517.12	15487.56	26516.17	30766.76	20954.75
Pong	none	20.62	12.23	19.19	18.99	11.14
	FGSM$^{p=0.4}$	-14.43	18.61	18.10	19.10	10.55
Boxing	none	87.07	41.07	82.15	93.30	46.82
	FGSM$^{p=0.4}$	67.11	40.45	78.14	93.52	51.52

Table 2: Comparing defended DQN to defended DadQN by validation game score.

		DQN				DadQN			
Game	Test Attack	none	Atk	Def	Atk+Def	none	Atk	Def	Atk+Def
RR	none	20244	15903	242	820	13315	22726	19781	18480
	FGSM$^{p=1}$	780	19478	981	1101	647	15234	18538	17566
Pong	none	19.85	17.83	-21.00	-20.86	20.55	19.73	20.31	18.29
	FGSM$^{p=1}$	-21.00	17.42	-21.00	-20.94	-19.04	13.50	19.12	17.03
Boxing	none	77.71	41.12	-26.67	-23.40	95.49	80.60	79.88	73.11
	FGSM$^{p=1}$	8.60	41.05	-9.20	-52.77	5.07	55.29	56.30	66.75

and the majority took about 20 hours. We evaluated using Nvidia 1080Tis and Nvidia k80s, with otherwise modern hardware. On our fastest machine, using a Nvidia 1080Ti, our implementation of DQN plays 266.9 frames per second and the DadQN algorithm 266.0 frames.

Attacks & Defenses We consider both untargeted FGSM, as well as our UQP attack. For all uses of FGSM we use $\epsilon = 0.004$. During validation game attacks, we use FGSM with probability 0.4 or 1.0, written as FGSM$^{p=0.4}$ or FGSM$^{p=1}$. For standard Q-Networks the semantics of attacking and defending with FGSM, are similar but not quite the same: when attacked, the environment produces adversarial examples which are then stored in the replay buffer and are seen many times in training. When used as a defense, the perturbation is applied to a training example when it is sampled from the replay buffer, thus each perturbation will only be seen once. We apply our attacks and defenses to the 4-stacks of consecutive frames used for training DQNs on Atari.

4.2 Poisoning as an Attack

Table 1 shows the average final episode score (weighted by number of frames) from the 15 consecutive best (by sum score) validation games during training. The leftmost DQN column first shows these scores without any attack on DQNs for reference. When attacking DQN during training we observe significant drop in score for unattacked validation games. We can see from this table that UQP is a strong attack against DQN's as the presence of the attack (i) impacts undefended training performance and (ii) significantly impacts the performance of the agent in unattacked *and* attacked games during testing. To exemplify (i), Figure 3 shows the training reward for Boxing in the presence of an attack.

4.3 Utility of defending with DadQN

Table 2 shows the average final episode score (weighted by number of frames) using the weights from the best validation game during training over 15 validation episodes. We can see that attacking a DQN which has never encountered adversarial examples before (*none* column under DQN) drastically lowers its score. When attacked by the environment (*Atk*), whether intentionally or not, the agent learns to play the game, although not as well as without an attack or DadQN that has been defended, or DadQN being defended *and* attacked. We also see that a DQN using the previously defined FGSM training defense (*Def* - perturbations happen after sampling from the replay buffer) does not learn how to play the game whatsoever. We hypothesize that the attack variant places the adversarial examples in the replay buffer making it easier for the agent to learn since it then encounters the same perturbations more frequently. DadQN trained with the FGSM defense performs nearly as well as baseline DQN without attacks, and much better than baseline with attacks.

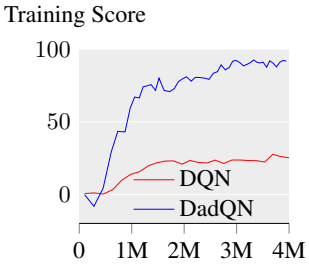

Figure 3: UQP poisoning attacks with number of frames played on x axis.

Table 3: Avg. max provable ϵ with DiffAI Box.[4]

	DQN		DadQN	
Game	none	Atk	Def	DiffAI
RR	2.66e-08	1.76e-06	8.42e-05	7.82e-04
Pong	3.71e-08	1.98e-06	2.19e-05	4.11e-04
Boxing	2.86e-07	3.67e-07	9.20e-07	5.12e-05

In addition to increasing test time attack robustness, we can see in Table 1 that DadQN trained without an explicit defense is robust to poisoning. Rewards for DadQN were nearly always greater than DQN whether playing finally in an attacked or unattacked game. Using the FGSM Defense amplified the effect to recover nearly the full performance of an unattacked DQN.

4.4 Towards Practical Provable Robustness of DQNs with DadQN

For DiffAI we explored a slightly different domain for each game. For Roadrunner we use $\lambda = 0.001$ with a non-defensive $\alpha(s, \theta_S) = s$ and $L_{O,\text{MSE}}$. Pong and Boxing instead use $\lambda = 0.01$ and $\epsilon = 0.0001$ with $\alpha_{0.004}$. For boxing, we use an attack probability of 0.4. Knowing that DiffAI can decrease classification accuracy, some loss in score was expected. However, the last column of Table 1 demonstrates that DiffAI used in this was still a powerful defense against test the test time FGSM attack the UQP attack compared to otherwise undefended DQNs.

Table 3 demonstrates the average maximum ϵ which could be proven robust using DiffAI over the best (by aggregate score) 15 consecutive validation games. From this table, we can see that the best ϵ that DiffAI was able to prove without using any provability training was a tenth the size of the largest which could be found in DiffAI defended networks. Without any defense, the baseline DQN is not robust, and thus has a provability ϵ of less than a ten-thousandth the minimally normally expressible perturbation in 8-bit images (this would be $\epsilon \sim 0.004$). DiffAI on the other hand comes significantly closer than any system we are aware of, at nearly 25% this value.

5 Related Work

Papernot et al. (2015) first proposed that distilling a neural network by training the same architecture with the probability vector outputs of the original network might act as a defense against adversarial examples. While Carlini & Wagner (2016) found that defensive distillation is insufficient to protect against adversarial examples for supervised learning, our algorithm uses distillation in combination with more effective defenses to protect reinforcement learning. Chen et al. (2017) use a pre-defined rule-based teacher policy to assist a deep RL system. They introduce a new DQN algorithm which predicts when to consult the teacher and how to learn from the teacher's experiences. In our setting there is no known external teacher. Lin et al. (2017a) use distillation to improve reinforcement learning itself by introducing the collaborative asynchronous advantage actor-critic algorithm (cA3C). Their method allows for knowledge transfer between agents in cases where multiple agents are playing simultaneously in different environments with potentially different tasks. To do this, they use a "deep alignment network" which learns to transfer the outputs from a teacher network to a student network. Unlike our system, their system only improves agents learning from different tasks, and does not provide a method to defend the learned policy. Shashua & Mannor (2017) train a DQN, that avoids the worst case outcome and performs well in the presence of a non-deterministic environments by using an extended Kalman-Filter. However, their Kalman-update requires multiplication of matrices of size quadratic in the total number of weights and thus does not scale to neural networks that play Atari-Games with the screen input. On the other hand, we optimize a policy to generalize to non-game effecting perturbations in a way which we show is scalable to very large neural networks. We furthermore demonstrate how to do this in a way that increases provability.

[4]Found with binary search between 0 and 1 with a maximum of 20 iterations or maximum accuracy of $1e-6$ using a running average starting point.

REFERENCES

Vahid Behzadan and Arslan Munir. Vulnerability of deep reinforcement learning to policy induction attacks. In *International Conference on Machine Learning and Data Mining in Pattern Recognition*, pp. 262–275. Springer, 2017a.

Vahid Behzadan and Arslan Munir. Whatever does not kill deep reinforcement learning, makes it stronger. *CoRR*, abs/1712.09344, 2017b. URL http://arxiv.org/abs/1712.09344.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

Dimitri P Bertsekas. Neuro-dynamic programming. In *Encyclopedia of optimization*, pp. 2555–2560. Springer, 2008.

Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

Justin A Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in neural information processing systems*, pp. 369–376, 1995.

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

Nicholas Carlini and David Wagner. Defensive distillation is not robust to adversarial examples. *arXiv preprint arXiv:1607.04311*, 2016.

Lu Chen, Xiang Zhou, Cheng Chang, Runzhe Yang, and Kai Yu. Agent-Aware Dropout DQN for Safe and Efficient On-line Dialogue Policy Learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 2454–2464, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. Training verified learners with learned verifiers. *arXiv preprint arXiv:1805.10265*, 2018.

Eyal Even-Dar and Yishay Mansour. Convergence of optimistic and incremental q-learning. In *Advances in neural information processing systems*, pp. 1499–1506, 2002.

Aidin Ferdowsi, Ursula Challita, Walid Saad, and Narayan B. Mandayam. Robust deep reinforcement learning for security and safety in autonomous vehicle systems. *CoRR*, abs/1805.00983, 2018. URL http://arxiv.org/abs/1805.00983.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy Networks for Exploration. June 2017.

Timon Gehr, Matthew Mirman, Petar Tsankov, Dana Drachsler Cohen, Martin Vechev, and Swarat Chaudhuri. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *Symposium on Security and Privacy (SP)*, 2018.

Zhaoyuan Gu, Zhenzhong Jia, and Howie Choset. Adversary a3c for robust reinforcement learning, 2018. URL https://openreview.net/forum?id=SJvrXqvaZ.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

Sandy H. Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *CoRR*, abs/1702.02284, 2017. URL http://arxiv.org/abs/1702.02284.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, 2017.

Kaixiang Lin, Shu Wang, and Jiayu Zhou. Collaborative Deep Reinforcement Learning. *arXiv:1702.05796 [cs]*, February 2017a.

Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. Tactics of adversarial attack on deep reinforcement learning agents. *CoRR*, abs/1703.06748, 2017b. URL http://arxiv.org/abs/1703.06748.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. 2018.

Ajay Mandlekar, Yuke Zhu, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Adversarially robust policy learning through active construction of physically-plausible perturbations. In *IEEE Intl Conf. on Intelligent Robots and Systems (IROS)*, volume 16, 2017.

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning (ICML)*, 2018. URL https://www.icml.cc/Conferences/2018/Schedule?showEvent=2477.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 1476-4687. doi: 10.1038/nature14236.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Jun Morimoto and Kenji Doya. Robust reinforcement learning. *Neural computation*, 17(2):335–359, 2005.

Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *CoRR*, abs/1511.04508, 2015. URL http://arxiv.org/abs/1511.04508.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. Robust deep reinforcement learning with adversarial attacks. *CoRR*, abs/1712.03632, 2017. URL http://arxiv.org/abs/1712.03632.

Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018.

Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy Distillation. *arXiv:1511.06295 [cs]*, November 2015.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432*, 2015.

Shirli Di-Castro Shashua and Shie Mannor. Deep robust kalman filter. *arXiv preprint arXiv:1703.02310*, 2017.

Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified defenses for data poisoning attacks. *CoRR*, abs/1706.03691, 2017. URL `http://arxiv.org/abs/1706.03691`.

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.

Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

John N Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine learning*, 16(3): 185–202, 1994.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 2, pp. 5. Phoenix, AZ, 2016.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. 2018.

Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Scaling provable adversarial defenses. *arXiv preprint arXiv:1805.12514*, 2018.

Chaofei Yang, Qing Wu, Hai Li, and Yiran Chen. Generative poisoning attack method against neural networks. *arXiv preprint arXiv:1703.01340*, 2017.

APPENDIX

Table S1: Hyper-parameters used in the experiments. $\rightarrow$ indicates linear annealing.

	RoadRunner	**Pong**	**Boxing**
Optimizer	Adam		
Adam-ϵ	0.00015		
Learning rate	6.25E-05		
Batch-size	32		
Clip reward to sign	True		
Frame-Stack	4		
γ	0.99		
Priority Replay α	0.5		
Priority Replay β	$0.4 \rightarrow 1.0$ over 100000 frames		
Target net Sync	every 2000 frames		
Q-Net L2-Weight regularization	0.0001	0	0
NoisyNet Explore Constraint η	1	4	4
Frames before learning	10000	80000	80000
Size of replay buffer	100000	120000	200000
ϵ-greedy	0	$1.0 \rightarrow 0.0$ over 20000 frames	$1.0 \rightarrow 0.0$ over 20000 frames

Table S2: Hyper-parameters for the student algorithm used in the experiments.

	RoadRunner	**Pong**	**Boxing**
Student Learning Rate	0.001	2.00E-05	2.00E-05
Discard V	True	False	False
Q-Network is NoisyNet	False		
Student-Net L2-Weight regularization	0.001	0	0