CopyCAT: Taking Control of Neural Policies with Constant Attacks

Léonard Hussenot, Matthieu Geist, Olivier Pietquin

CopyCAT: Taking Control of Neural Policies with Constant Attacks

Léonard Hussenot
Google Research
INRIA SequeL, Université de Lille

Matthieu Geist
Google Research

Olivier Pietquin
Google Research

ABSTRACT

We propose a new perspective on adversarial attacks against deep reinforcement learning agents. Our main contribution is CopyCAT, a targeted attack able to consistently lure an agent into following an outsider's policy. It is pre-computed, therefore fast inferred, and could thus be usable in a real-time scenario. We show its effectiveness on Atari 2600 games in the novel *read-only* setting. In this setting, the adversary cannot directly modify the agent's *state* –its representation of the environment– but can only attack the agent's *observation* –its perception of the environment. Directly modifying the agent's state would require a *write-access* to the agent's inner workings and we argue that this assumption is too strong in realistic settings.

1 INTRODUCTION

We are interested in the problem of attacking sequential control systems that use deep neural policies. In the context of supervised learning, previous work developed methods to attack neural classifiers by crafting so-called "adversarial examples". These are malicious inputs particularly successful at fooling deep networks with high-dimensional input-data like images. Within the framework of sequential-decision-making, previous works used these adversarial examples only to break neural policies. Yet the attacks they build are rarely applicable in a real-time setting as they require to craft a new adversarial input at each time step. Besides, these methods use the strong assumption of having a *write-access* to what we call the agent's *state*, the actual input of the neural policy built by the algorithm from the observations. When taking this assumption, the adversary –the algorithm attacking the agent– is not placed at the interface between the agent and the environment where the system is the most vulnerable. We wish to design an attack with a more general purpose than just shattering a neural policy as well as working in a more realistic setting.

Our main contribution is CopyCAT, an algorithm for taking full-control of neural policies. It produces a simple attack that is: (1) targeted towards a policy, *i.e.*, it aims at matching a neural policy's behavior with the one of an arbitrary policy; (2) only altering observation of the environment rather than complete agent's state; (3) composed of a finite set of pre-computed state-independent masks. This way it requires no additional time at inference hence it could be usable in a real-time setting.

We introduce CopyCAT in the white-box scenario (Sec. 3), with *read-only* access to the weights and the architecture of the neural policy. This is a realistic setting as prior work showed that after training substitute models, one could transfer an attack computed on these to the inaccessible attacked model [20]. The context is the following: (1) We are given any agent using a neural-network for decision-making (*e.g.*, the Q-network for value-based agents, the policy network for actor-critic or imitation learning methods) and a **target policy** we want the agent to follow. (2) The only thing one can alter is the observation the agent receives from the environment and **not the full input** of the neural controller (the state). In other words, we are granted a *read-only* access to the agent's inner workings. In the case of Atari 2600 games, the agents builds its state by stacking the last four observations. Attacking the agent's state means writing in the agent's *memory* of the last observations. (3) The computed attack should be inferred fast enough to be used in **real-time**.

We stress the fact that targeting a policy is a more general scheme than untargeted attacks where the goal is to stop the agent from taking its preferred action (hoping for it to take the worst). It is also more general than the targeted scheme of previous works where one wants the agent to take its least preferred action or to reach a specific state. In our setting, one can either hard-code or train a target policy. This policy could be minimizing the agent's true reward but also maximizing the reward for another task. For instance, this could mean taking full control of an autonomous vehicle, possibly bringing it to any place of your choice.

We exemplify this approach on the classical benchmark of Atari 2600 games (Sec.5). We show that taking control of a trained deep RL agent so that its behavior matches a desired policy can be done with this very simple attack. We believe such an attack reveals the vulnerability of autonomous agents. As one could lure them into following catastrophic behaviors, autonomous cars, robots or any agent with high dimensional inputs are exposed to such manipulation. This suggests that it would be worth studying new defense mechanisms that could be specific to RL agents, but this is out of the scope of this paper.

Section 6 presents experiments studying various aspects of the proposed method, as the use of CopyCAT in the black-box setting or in environments with a more complex perception. These results assess the possibility of using the proposed method in various settings even though some of these experiments are still preliminary.

2 BACKGROUND

In **Reinforcement Learning** (RL), an agent interacts sequentially with a dynamic environment so as to learn an optimal control. To do so, the problem is modeled as a Markov Decision Process. It is a tuple $\{\mathcal{S}, \mathcal{A}, P, r, \gamma\}$ with $\mathcal{S}$ the state space, $\mathcal{A}$ the action space we consider as finite in the present work, P the transition kernel defining the dynamics of the environment, r a bounded reward function and $\gamma \in (0, 1)$ a discount factor. The policy π maps states to distributions over actions: $\pi(\cdot|s)$. The (random) discounted return is defined as $G = \sum_{t \geq 0} \gamma^t r_t$. The policy π is trained to maximize the agent expected discounted return. The function $V^\pi(s) = \mathbb{E}_\pi[G|s_0 = s]$

denotes the value function of policy π (where $\mathbb{E}_\pi[\cdot]$ denotes the expectation over all possible trajectories generated by policy π). We also call μ_0 the initial state distribution and $\rho(\pi) = \mathbb{E}_{s\sim\mu_0}[V^\pi(s)]$ the expected cumulative reward starting from μ_0. Value-based algorithms [8, 18] use the value function, or more frequently the action-value function $Q^\pi(s,a) = \mathbb{E}_\pi[G|s_0 = s, a_0 = a]$, to compute π. To handle large state spaces, deep RL uses deep neural networks for function approximation. For instance, value-based deep RL parametrizes the action-value function Q_ω with a neural network of parameters ω and deep actor-critics [17] directly parametrize their policy π_θ with a neural network of parameters θ. In both cases, the taken action is inferred by a forward-pass in a neural network.

Adversarial examples were introduced in the context of supervised classification. Given a classifier C, an input x, a bound ϵ on a norm $\|.\|$, an adversarial example is an input $x' = x + \eta$ such that $C(x) \neq C(x')$ while $\|x - x'\| \leq \epsilon$. *Fast Gradient Sign Method* (FGSM) [7] is a widespread method for generating adversarial examples for the L_∞ norm. From a linear approximation of C, it computes the attack η as:

$$\eta = \epsilon \cdot \text{sign}(\nabla_x l(\theta, x, y)), \tag{1}$$

with $l(\theta, x, y)$ the loss of the classifier and y the true label.

As an adversary, one wishes to **maximize** the loss $l(\theta, x + \eta, y)$ w.r.t. η. Presented this way, it is an *untargeted* attack. It pushes C towards misclassifying x' in *any* other label than y. It can easily be turned into a *targeted* attack by, instead of $l(\theta, x + \eta, y)$, optimizing for $-l(\theta, x + \eta, y_\text{target})$ with y_target the label the adversary wants C to predict for x'. This attack, optimized for the L_∞ norm can also be turned into an L_2 attack by taking:

$$\eta = \epsilon \cdot \frac{\nabla_x l(\theta, x, y)}{\|\nabla_x l(\theta, x, y)\|_2}. \tag{2}$$

As shown by Eqs. (1) and (2), these attacks are computed with one single step of gradient, hence the term "fast". These two attacks can be turned into –more efficient, yet slower– iterative methods [3, 6] by taking several successive steps of gradients. These methods will be referred to as *iterative-FGSM.*

When using **deep networks** to compute its policy, an RL agent can be fooled the same way as a supervised classifier. As a policy can be seen as a mapping $\mathcal{S} \to \mathcal{A}$, untargeted FGSM (1) can be applied to a deep RL agent to stop it from taking its preferred action: $a^* = \arg\max_{a\in\mathcal{A}} \pi(a|s)$. Similarly targeted FGSM can be used to lure the agent into taking a specific action. Yet, this would mean having to compute a new attack at each time step, which is generally not feasible in a real-time setting. Moreover, with this formulation, it needs to directly modify the agent's state, the input of the neural policy, which is a strong assumption.

3 THE COPYCAT ATTACK

In this work, we propose CopyCAT. It is an attack whose goal is to lure an agent into having a given behavior, the latter being specified by another policy. CopyCAT's goal is not only to lure the agent into taking specific actions but to fully control its behavior. Formally, CopyCAT is composed of a set of additive masks $\Delta = \{\delta_a\}_{a\in\mathcal{A}}$ that can be used to drive a policy π to follow any policy π^target. Each additive mask δ_a is pre-computed to lure π into taking a specific action a when added to the current observation regardless of the content of the observation. It is, in this sense, a *universal* attack. CopyCAT is an attack on raw observations and, as Δ is pre-computed, it can be used online in a real-time setting with no additional computation.

Notations. We denote π the attacked policy and π^target the target policy. At time step t, the policy π outputs an action a_t taking the state s_t as input. The agent state is internally computed from the past observations and we denote f the observations-to-state function: $s_t = f(o_t, o_{1:t-1})$ with $o_{1:t-1} = (o_1, o_2 ... o_{t-1})$.

Data Collection. In order to be pre-computed, CopyCAT needs to gather data from the agent. By watching the agent interacting with the environment, CopyCAT gathers a dataset $\mathcal{D}$ of K episodes made of observations: $\mathcal{D} = (o_t^k)_{t\in(1,T_k)}^{k\in(1:K)}$. We recall that the objective in this setting is for CopyCAT to work with a *read-only* access to the inner workings of the agent. We thus stress that $\mathcal{D}$ is made of observations rather than states. If CopyCAT is successful, π is going to behave as π^target and thus may experience observations out of the distribution represented in $\mathcal{D}$. Yet, as will be shown, CopyCAT transfers to unseen observations. We hypothesize that, as we build a *universal* attack, the learned attack is able to move the whole support of observations in a region of $\mathbb{R}^N$ where π chooses a precise action.

Training. A natural strategy for building an adversarial example targeted towards label $\hat{y}$ is the following. Given a classifier $\mathbb{P}(y|x)$ parametrized with a neural network and an input example x, one computes the adversarial example $\hat{x} = x + \delta$ by maximizing $\log \mathbb{P}(\hat{y}|\hat{x})$ subject to the constraint: $\|\delta\|_\infty = \|x - \hat{x}\|_\infty \leq \epsilon$. The adversary then performs either one step of gradient (FGSM) or uses an iterative method [13] to solve the optimization problem.

Instead, CopyCAT is built for its masks to be working whatever the observation it is applied to. For each $a \in \mathcal{A}$ we build δ_a, the additional masks luring π into taking action a, by **maximizing** over δ_a:

$$\mathop{\mathbb{E}}_{o_t^k \in \mathcal{D}} \Big[\log \pi(a|f(o_t^k + \delta_a, o_{1:t-1}^k)) + \alpha \|\delta_a\|_2 \Big] \text{ s.t. } \|\delta_a\|_\infty < \epsilon\,. \tag{3}$$

We restrict the method to the case where f, the function building the agent's inner state from the observations, is differentiable. The Eq. 3 can be optimized by alternating between gradient steps with adaptive learning rate [11] on mini-batches and projection steps onto the L_∞-ball of radius ϵ. Unlike FGSM, CopyCAT is a full optimization method. It does not take only one single step of gradient.

CopyCAT has two main parameters: $\epsilon \in \mathbb{R}^+$, a hard constraint on the L_∞ norm of the attack and $\alpha \in \mathbb{R}^+$, a regularization parameter on the L_2 norm of the attack.

Inference. Once Δ is computed, the attack can be used on π to make it follow any policy π^target. At each time step t and given past observations, π^target infers an action a_t^target. The corresponding mask $\delta_{a_t^\text{target}} \in \Delta$ is applied to the last observation o_t before being passed to the agent. No optimization is made at inference and CopyCAT can thus be used in a setting where the adversary is not let unlimited time to compute an attack between two time-steps.

4 RELATED WORK

Vulnerabilities of neural classifiers were highlighted by Szegedy et al. [26] and several methods were developed to create the so-called *adversarial examples*, maliciously crafted inputs fooling deep networks. In sequential-decision-making, previous works use them to attack deep reinforcement learning agents. However these attacks are not always realistic. The method from Huang et al. [10] uses *fast-gradient-sign method* [7] for the sole purpose of destroying the agent's performance. What's more, it has to craft a new attack at each time step. This implies back-propagating through the agent's network, which is not feasible in real-time. Moreover, it modifies directly the state of the agent by writing in its memory, which is a strong assumption to take on what component of the agent can be altered. The approach of Lin et al. [15] allows the number of attacked states to be divided by four, yet it uses the heavy optimization scheme from Carlini and Wagner [3] to craft their adversarial examples. This is, in general, not doable in a real-time setting. They also take the same strong assumption of having a "read & write-access" to the agent's inner workings. To the best of our knowledge, they are the first to introduce a targeted attack. However, the setting is restricted to targeting one "dangerous state". Pattanaik et al. [21] proposes a method to lure the agent into taking its least preferred action in order to reduce its performance but still uses computationally heavy iterative methods at each time step. Pinto et al. [22] proposed an adversarial method for robust training of agents but only considered attacks on the dynamic of the environment, not on the visual perception of the agent. Zhang et al. [27] and Ruderman et al. [23] developed adversarial environment generation to study agent's generalization and worst-case scenarios. Those are different from this present work where we enlighten how an adversary might take control of a neural policy.

5 ATARI EXPERIMENTS

We wish to build an attack targeted towards the policy π^{target}. At a time step t, the attack is said to be successful if π under attack indeed chooses the targeted action selected by π^{target}. When π is not attacked, the attack success rate corresponds to the *agreement rate* between π and π^{target}, measuring how often the policies agree along an unattacked trajectory of π.

Note that we only deal with trained policies and no learning of neural policies is involved. In other words, π and π^{target} are trained and frozen policies.

What we really want to test is the ability of CopyCAT to lure π into having a specific behavior. For this reason, measuring the attack success rate is not enough. Having a high success rate does not necessarily mean the macroscopic behavior of the attacked agent matches the desired one as will be shown further in this section.

Cumulative reward as a proxy for behavior. We design the following setup. The agent has a policy π trained with DQN [18]. The policy π^{target} is trained with Rainbow [8]. We select Atari games [1] with a clear difference in terms of performance between the two algorithms (where Rainbow obtains higher average cumulative reward than DQN). This way, in addition to measuring the attack success rate, we can compare the cumulative reward obtained by π under attack $\rho(\pi)$ to $\rho(\pi^{\text{target}})$ as a proxy of how well π's behavior is matching the behavior induced by π^{target}. In this setup, if the attacked policy indeed gets cumulative rewards as high as the ones obtained by π^{target}, it will mean that we did not simply turned some actions into other actions we targeted, but that the whole behavior induced by π under attack matches the one induced by π^{target}. This idea that, in reinforcement learning, cumulative reward is the right way to monitor an agent's behavior has been used and developed by the inverse reinforcement learning literature. Ng et al. [19] argued that the value of a policy, *i.e.* its cumulative reward, is the most compact, robust and transferable description of its induced behavior. We argue that measuring cumulative reward is thus a reasonable proxy for monitoring the behavior of π under attack. At this point, we would like to carefully defuse a possible misunderstanding. Our goal is not to show that DQN's performance can be improved by being attacked. We simply want to show that its behavior can be fully manipulated by an opponent and we use the obtained cumulative reward as a proxy for the behavior under attack.

Baseline. We set the objective of building a real-time targeted attack. We thus need to compare our algorithm to baselines applicable to this scenario. The fastest state-of-the-art targeted method can be seen as a variation of Huang et al. [10]. It applies *targeted FGSM* at each time step t to compute a new attack. It first infers the action a^{target} and then back-propagates through the attacked network to compute their attack. CopyCAT only infers a^{target} and then applies the corresponding pre-computed mask. Both methods can thus be considered usable in real-time yet CopyCAT is still faster at inference. We set the objective of attacking only observations rather than complete states so we do not need a *write-access* to the agent's inner workings. DQN stacks four consecutive frames to build its inner state. We thus compare CopyCAT to a version of the method from Huang et al. [10] where the gradient inducing the FGSM attack is only computed w.r.t the last observation, so it produces an attack comparable to CopyCAT, *i.e.*, on a single observation. The gradient from Eq. 1: $\nabla_{s_t} l(\theta, s_t, a^{\text{target}})$ becomes $\nabla_{o_t} l(\theta, f(o_t, o_{1:t-1}), a^{\text{target}})$. To keep the comparison fair, a^{target} is always computed with the exact same policy π^{target} as in CopyCAT. The policy $\pi^{\text{target}} : \mathcal{S} \rightarrow \mathcal{A}$ is fixed.

FGSM-L_∞ has the same parameter ϵ as CopyCAT, bounding the L_∞ norm of the attack. CopyCAT has an additional regularization parameter α allowing the attack to have, for a same ϵ, a lower energy and thus be less detectable. We will compare CopyCAT to the attack from Huang et al. [10] showing how behaviors of π under attacks match π^{target} when these attacks are of equal energy.

Full optimization-based attacks would not be inferred fast enough to be used in a sequential decision making problem at each time step.

Experimental setup. We always turn the sticky actions on, which make the problem stochastic [16]. An attacked observation is always clipped to the valid image range, 0 to 255. For Atari games, DQN uses as its inner state a stack of four observations: $f(o_i, o_{1:i-1}) = [o_i, ..., o_{i-3}]$. For learning the masks of Δ, we gather trajectories generated by π in order to fill $\mathcal{D}$ with 10k observations. We use a batch size of 8 and the Adam optimizer [11] with a learning rate of 0.05. Each point of each plot is the average result over 5 policy π seeds and 80 runs for each seed. Only one seed is used for π^{target} to keep comparison in terms of cumulative reward fair.

***Attack energy*.** As CopyCAT has an extra parameter α, we test its influence on the L_2 norm of the produced attack. For a given ϵ, FGSM-L_∞ computes an attack η of maximal energy. As given by Eq. 1, its L_2 norm is $\|\eta\|_2 = \sqrt{N\epsilon^2}$ with N the input dimension. For a given ϵ, CopyCAT produces $|\mathcal{A}|$ masks. We show in Fig. 1 the largest L_2 norm of the $|\mathcal{A}|$ masks for a varying α (plain curves) and compare it to the norm of the FGSM-L_∞ attack (dashed lines). We want to stress that the attacks are agnostic to the training algorithm so the results are easily transferred to other agents using neural policies trained with another algorithm.

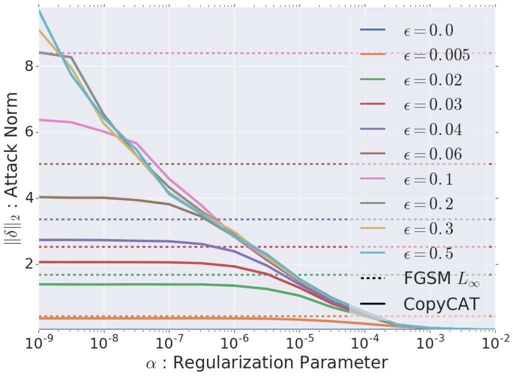

Figure 1: Influence of parameters ϵ and α on the maximal L_2 norm of Δ: $\max_{a\in\mathcal{A}} \|\delta_a\|_2$. CopyCAT is attacking DQN on HERO.

As can be seen on Fig. 1, for a given ϵ and for the range of tested α, the attack produced by CopyCAT has lower energy than FGSM-L_∞. This is especially significant for higher values of ϵ, *e.g* higher than 0.05.

***Influence of parameters over the resulting behavior*.** We wish to show how the agent behaves under attack. As explained before, this analysis is twofold. First, we study results in terms of attack success rate –rate of action chosen by π matching a^{target} when shown attacked observations– as done in supervised learning. Second, we study the behavior matching through the cumulative rewards under attack $\rho(\pi)$.

What we wish to verify in the following experiment is CopyCAT's ability to lure an agent into following a specific behavior. If the attack success rate is high (close to 1), we know that, on a supervised-learning perspective, our attack is successful: it lures the agent into taking specific actions. If, in addition, the average cumulative reward obtained by the agent under attack reaches $\rho(\pi^{\text{target}})$ it means that the attack is really successful in terms of behavior. We recall that we attack a policy with a target policy reaching higher average cumulative reward.

We show on Fig. 2, 3 and 4 (three different games) the attack success rate (left) and the cumulative reward (right) for CopyCAT (plain curves) for different values of the parameters α and ϵ, as well as for unattacked π (green dashed line) and π^{target} (black dashed lines). We observe a gap between having a high success rate and forcing the behavior of π to match the one of π^{target}. There seems to exist a threshold corresponding to the minimal success rate required for the behaviors to match. For example, as seen on the left, CopyCAT with $\epsilon = 5$ and $\alpha < 10^{-5}$ (green curve) is enough to get a 85% success rate on the attack. However, as seen on the right, it is not enough to get the behavior of π under attack to match the one of the target policy as the reward obtained under attack never reaches $\rho(\pi^{\text{target}})$.

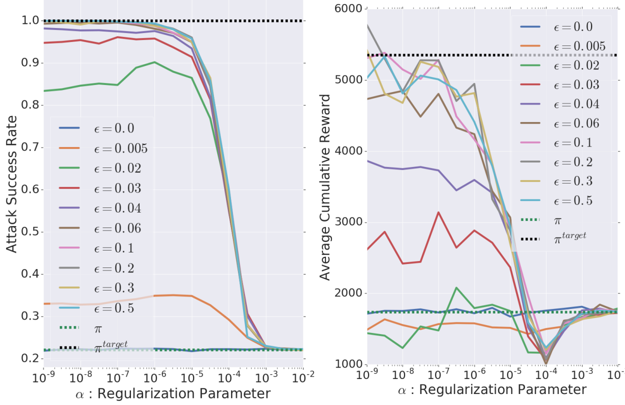

Figure 2: Influence of parameters ϵ and α on the average behavior of the agent playing Space Invaders under CopyCAT attack . On the left, the attack success rate. On the right, the average cumulative reward.

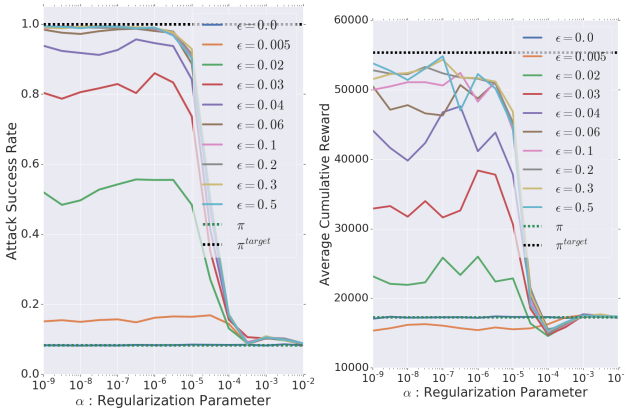

Figure 3: Influence of parameters ϵ and α on the average behavior of the agent playing HERO under CopyCAT attack . On the left, the attack success rate. On the right, the average cumulative reward.

Overall, we observe on Fig. 2-right, Fig. 3-right and Fig. 4-right that with ϵ high enough $\epsilon \geq 0.04$ and $\alpha < 10^{-6}$, CopyCAT is able to consistently lure the agent into following the behaviour induced by π^{target}.

***Comparison to Huang et al. [10]*.** We compare CopyCAT to the targeted version of FGSM on a setup where the gradient is computed only on the last observation. As in the last paragraph, we study both the attack success rate and the average cumulative

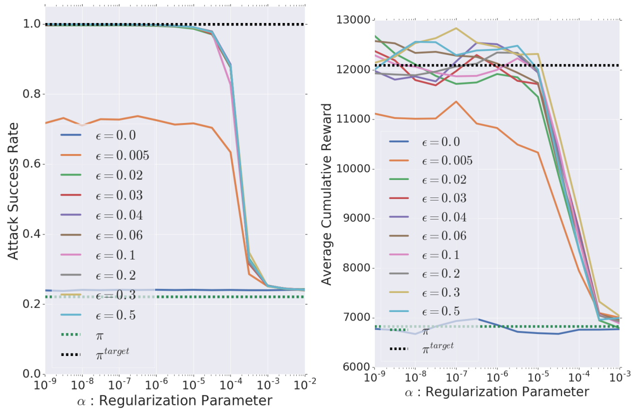

Figure 4: Influence of parameters ϵ and α on the average behavior of the agent playing Air Raid under CopyCAT attack . On the left, the attack success rate. On the right, the average cumulative reward.

reward under attack. We ask the question: is CopyCAT able to lure the agent into following the targeted behavior? Is it better at this task than FGSM in the *real-time* and *read-only* setting?

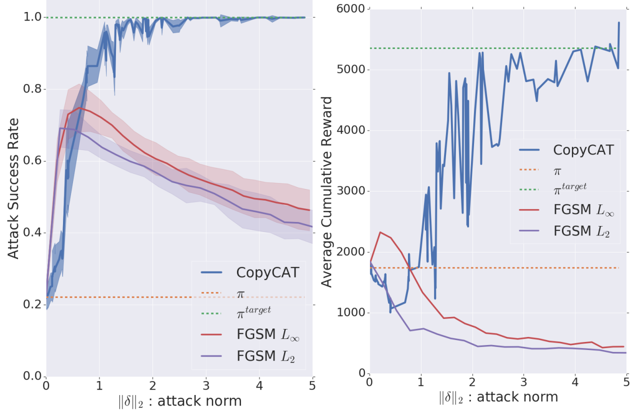

Figure 5: CopyCAT against FGSM for policy targeted attacks in Space Invaders. On the left, the attack success rate. On the right, the average cumulative reward under attack.

We show on Fig. 5, 6 and 7 (three different games) the success rate of CopyCAT and FGSM (y-axis, left) and the average cumulative reward under attack (y-axis, right). These values are plotted (i) against the L_2 norm of the attack for FGSM and (ii) against the largest L_2 norm of the masks: $\max_{a \in \mathcal{A}} \|\delta_a\|_2$ for CopyCAT. We only plot the standard deviation on the attack success rate because it corresponds to the intrinsic noise of CopyCAT. We do not plot it for cumulative reward for the reason that one seed of π^{target} has a great variance (with the sticky actions) and matching π^{target}, even perfectly, implies matching the variance of its cumulative rewards. The same phenomenon can be observed on Fig. 2 and 3: CopyCAT is not itself unstable (left figures, when α decreases or ϵ increases, the rate of successful attacks consistently increases). Yet the cumulative reward is noisier, as the behavior of π is now matching with a high-variance policy. As observed on Fig. 5-right, Fig. 6-left and Fig. 7-left, FGSM is able to turn a potentially significant part of the taken actions into the targeted actions (maximal success rate around 75% on Space Invaders). However, it is never able to make π's behavior match with π^{target}'s behavior as seen on Fig. 5-right, Fig. 6-right and Fig. 7-right. The average cumulative reward obtained by π under FGSM attack never reaches the one of π^{target}. On the contrary, CopyCAT is able to successfully lure π into following the desired macroscopic behavior. First, it turns more than 99% of the taken actions into the targeted actions. Second, it makes $\rho(\pi)$ under attack reach $\rho(\pi^{\text{target}})$. Moreover, it does so using only a finite set of masks while the baselines compute a new attack at each time step.

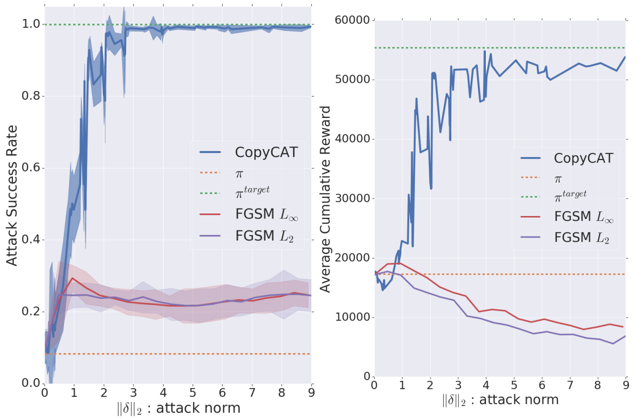

Figure 6: CopyCAT against FGSM for policy targeted attacks in HERO. On the left, the attack success rate. On the right, the average cumulative reward under attack.

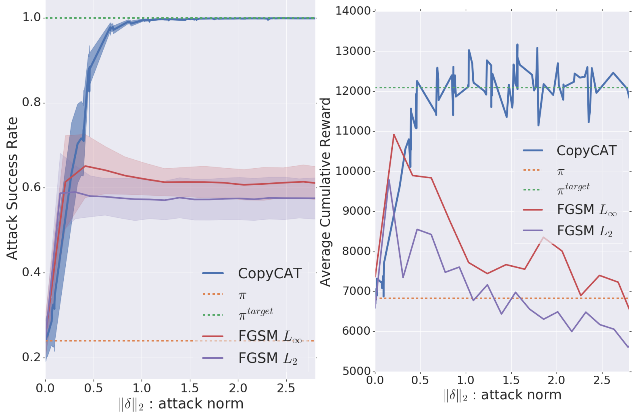

Figure 7: CopyCAT against FGSM for policy targeted attacks in Air Raid. On the left, the attack success rate. On the right, the average cumulative reward under attack.

An example of CopyCAT is shown on Fig. 8. The patch δ_a aiming at action a: "no-op" (*i.e.* do nothing) is applied to an agent playing Space Invaders. The patch itself can be seen on the right (gray represents a zero pixel, black negative and white positive). The

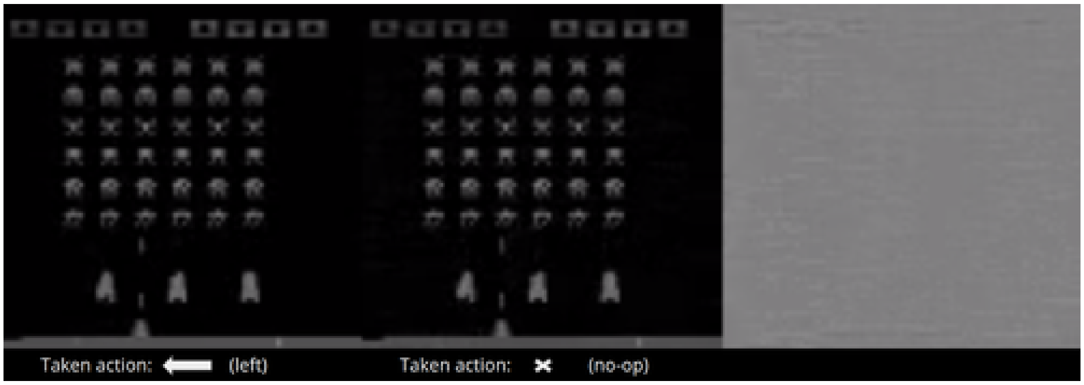

Figure 8: DQN playing Space Invaders. On the left, the unattacked current observation. In the middle, a mask of CopyCAT is applied to lure the agent into taking the "no-op" action. Parameters used are: $\epsilon = 3 \cdot 10^{-2}$, $\alpha = 1.3 \cdot 10^{-5}$. The L_2 norm of the corresponding mask (shown on the right) is 0.78. For this same ϵ, FGSM-L_∞ would produce an attack of L_2 norm: 2.52

unattacked observation is on the left, and the attacked one on the right. Below the images is provided the action taken by the same policy π when shown the different situations in an online setting.

6 ADDITIONAL EXPERIMENTS

In this section, we provide additional experiments to study further various aspects of the proposed approach. We build experiments to assess the possibility that CopyCAT would be successfully applicable in harder contexts, with very dissimilar policies π and π^{target}, in the much more complex black-box setting or in environments with more realistic images than Atari.

6.1 Towards black-box targeted attacks

Papernot et al. [20] observed the transferability of adversarial examples between different models. Thanks to this transferability, one is able to attack a model without having access to its weights. By learning attacks on proxy models, one can build black-box adversarial examples. However Kos and Song [12] enlightened the difficulty for the state-of-the-art methods to build *targeted* adversarial examples in this black-box setting. Starting from the intuition that universal attacks may transfer better between models, we enhanced CopyCAT for it to work in the black-box setting.

We consider a setting where the adversary (1) is given a set of proxy models $\{\pi_1, ..., \pi_n\}$ trained with the same algorithm as π, (2) can also query the attacked model π, but (3) has no access to its weights. In the black-box setting, CopyCAT is divided into two steps: first training multiple additional masks, and second selecting the highest performing ones.

Training. The ensemble-based method from Kos and Song [12] computes its additional mask by attacking the classifier given by the mean predictions of the proxy models. We instead consider that our attack should be efficient against any convex combination of the proxy models' predictions. For each action $a \in \mathcal{A}$, we compute the mask δ_a by maximizing over 100 epochs on the dataset $\mathcal{D}$:

$$\mathbb{E}_{\substack{o_t^k \in \mathcal{D} \\ (\alpha_1 \ldots \alpha_p) \sim \Delta}} \left[\log \sum_{1 \le p \le n} \alpha_p \pi_p(a | f(o_t^k + \delta_a, o_{1:t-1}^k)) + \alpha \, \|\delta_a\|_2 \right] \tag{4}$$
$$\text{s.t. } \|\delta_a\|_\infty < \epsilon \, .$$

with Δ the uniform distribution over the n-simplex. By sampling uniformely over the simplex, we hope to build an attack fooling any classifier with predictions in the convex hull of the proxy models' predictions.

For each action, 100 masks are computed this way. These masks are just computed with different random seeds.

Selection. We then compute a *competition accuracy* for each of these random seeds. This accuracy is computed by querying π on states built as follows. We take four consecutive observations in $\mathcal{D}$, apply 3 masks randomly selected among the previously computed masks on the first 3 observations; the mask δ_a that is actually being tested is applied on the last observation. The attack is considered successful if π outputs the action a corresponding to δ_a. For each action, the mask with the highest *competition accuracy* among the 100 computed masks is selected.

Inference. The selected masks are then used online as in the white-box setting.

Results. We provide preliminary results for the considered black-box setting. Four proxy models of DQN are used to attack π. Again, it is attacked to make it follow the policy π^{target} given by Rainbow. The results can be found in Fig. 9. Each dot is an attack tested over 80 new episodes. Y-axis is the mean success rate (middle) or the cumulative reward (right). X-axis is the maximal norm of the attack. The figure on the left gives the value of α (on the y-axis) corresponding to each color.

We can observe on Fig. 9 that the proposed black-box attack is effective, even if less efficient than its white-box counterpart. The proposed black-box CopyCAT could certainly be improved, and we let this for future work.

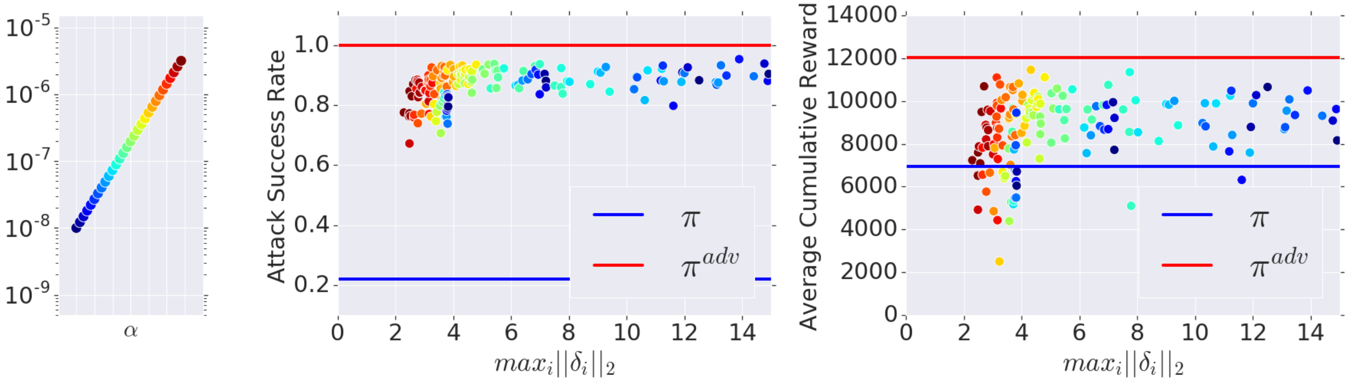

Figure 9: CopyCAT attacking DQN on Air Raid, in the black-box setting to make it match Rainbow's policy.

6.2 Attacking untrained DQN

In Sec. 5, the attacked agent was a trained DQN agent, while the target policy was a trained Rainbow agent. If these agents have clearly different behaviors, one could argue that they were initially trained to solve the same task (getting the highest score as possible), Rainbow achieving better results. To further assess CopyCAT's ability to lure a policy into following another policy, we therefore attack an untrained DQN, with random weights, to follow the policy π^{target} still obtained from a trained Rainbow agent. These two policies are dissimilar as one is random while the other is trained to maximized the game's score.

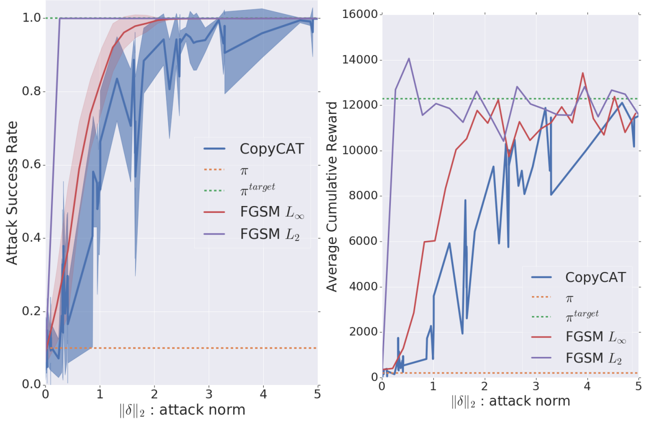

Figure 10: Untrained DQN attacked by CopyCAT to follow Rainbow's policy on Air Raid. Left: the attack success rate. Right: the average cumulative reward.

We see on Fig. 10, 11, 12 that in this case, FGSM is able to lure π into following π^{target} at least as well as CopyCAT. This shows that it is easier to fool an untrained network than a trained one. As expected, trained networks are more robust to adversarial examples. CopyCAT is also able to lure the agent into following π^{target}.

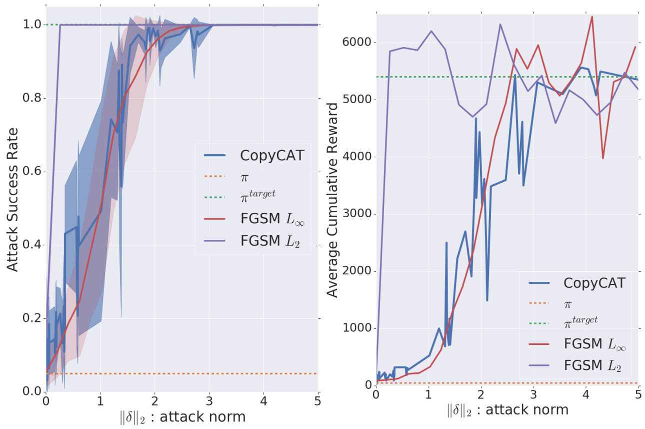

Figure 11: Untrained DQN attacked by CopyCAT to follow Rainbow's policy on Space Invaders. Left: the attack success rate. Right: the average cumulative reward.

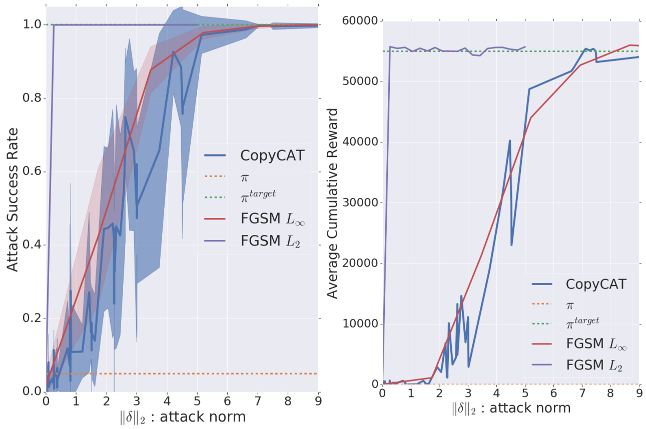

Figure 12: Untrained DQN attacked by CopyCAT to follow Rainbow's policy on HERO. Left: the attack success rate. Right: the average cumulative reward.

6.3 More realistic images

Reinforcement learning led to great improvements for games [24] or robots manipulation [14] but is not able yet to tackle realistic-image environments. While this paper focuses on weaknesses of reinforcement learning agents, the relevance of the proposed method would be diminished if one could not compute universal adversarial examples on realistic datasets. We thus present this proof-of-concept, showing the existence of universal adversarial examples on the ImageNet [5] dataset. Note that Brown et al. [2] already showed the existence of universal attacks but considered a patch completely covering a part of the image rather than an additional mask on the image.

We computed a universal attack on VGG16 [25], targeted towards the label "tiger_shark", the same way CopyCAT does. It is trained on a small training set (10 batches of size 8), and tested on a random subset of ImageNet validation dataset. The network is taken from Keras pretrained models [4] and attacked in a white-box setting. The same procedure as CopyCAT is used. 1000 images are randomly selected in the validation set. Only 80 are used for training and the rest is used for testing. The attack is trained with the same loss, same learning rate and same batch size as CopyCAT, for 200 epochs. The (rescaled) computed attack is shown in Fig. 13. Examples of attacked images from the test set are visible on Fig. 14.

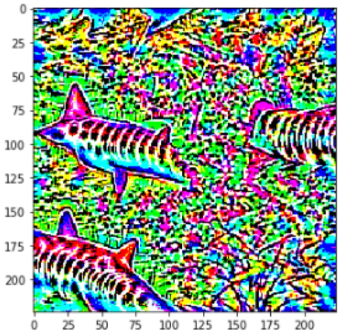

Figure 13: The rescaled attack for the target class "tiger_shark".

After 200 epochs, the train accuracy is 90% and the test accuracy 88.44%. This proof-of-concept experiment validates the existence of universal adversarial examples on realistic images and shows that CopyCAT's scope is not reduced to Atari-like environments. More generally, the existence of adversarial examples have been shown to be a property of high-dimensional manifolds [7]. Going towards more realistic images, hence higher dimensional images, should on the opposite, allow CopyCAT to more easily find universal adversarial examples.

7 CONCLUSION

In this work, we built and showed the effectiveness of CopyCAT, a simple algorithm designed to attack neural policies in order to manipulate them. We showed its ability to lure a policy into having a desired behavior with a finite set of additive masks, usable in a real-time setting while being applied only on observations of the environment. We demonstrated the effectiveness of these universal masks in Atari games, in the white-box setting. We also investigated an extension of CopyCAT in the black-box setting and validated the possibility of using it on potentially more complex environments with realistic observations.

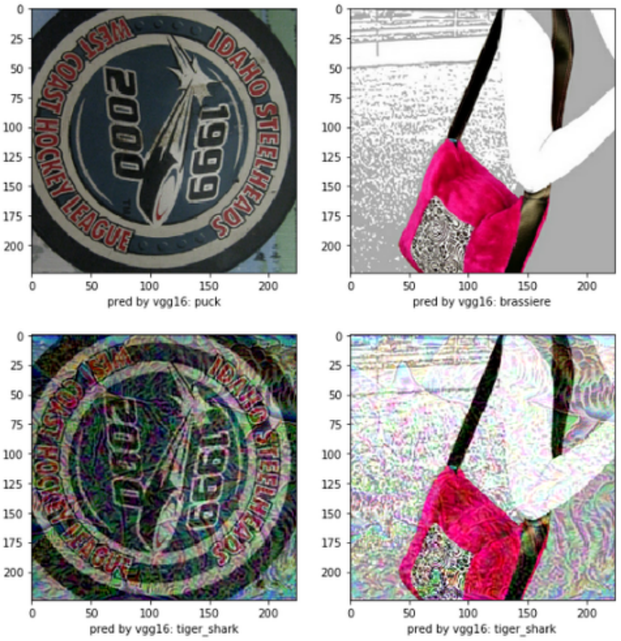

Figure 14: Top: images from the test set, unseen during the training of the attack. Down: attacked images: (image + attack). Below each image, the label predicted by VGG16.

As this work shows that one can easily manipulate a policy's behavior, a natural direction of work is to develop robust algorithms, either able to keep their normal behaviors when attacked or to detect attacks to treat them appropriately. Notice however that in a sequential-decision-making setting, detecting an attack is not enough as the agent cannot necessarily stop the process when detecting an attack and may have to keep outputting actions for incoming observations. It is thus an exciting direction of work to develop algorithm that are able to maintain their behavior under such manipulating attacks. Another interesting direction of work in order to build real-life attacks is to further develop targeted attacks on neural policies in the black-box scenario, with no access to network's weights and architecture. However, targeted adversarial examples are harder to compute than untargeted ones and we may experience more difficulties in reinforcement learning than supervised learning. Indeed, learned representations are known to be less interpretable and the variability between different random seeds to be higher than in supervised learning. Different policies trained with the same algorithm may thus lead to $\mathcal{S} \to \mathcal{A}$ mappings with very different decision boundaries. Transferring targeted examples may not be easy. In order to attack a policy π, training imitation models like the one from Hester et al. [9] to obtain proxy policies that are both (i) solving the same task as π and (ii) similar mappings $\mathcal{S} \to \mathcal{A}$ may be the key to compute transferable adversarial examples.

REFERENCES

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

[2] Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. 2017. Adversarial patch. *arXiv preprint arXiv:1712.09665* (2017).

[3] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 39–57.

[4] François Chollet et al. 2015. Keras. https://keras.io. (2015).

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

[6] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. 2018. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 9185–9193.

[7] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples (2014). *International Conference on Learning Representations* (2015).

[8] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[9] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. 2018. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[10] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. 2017. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284* (2017).

[11] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[12] Jernej Kos and Dawn Song. 2017. Delving into adversarial attacks on deep policies. *arXiv preprint arXiv:1705.06452* (2017).

[13] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533* (2016).

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

[15] Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. 2017. Tactics of adversarial attack on deep reinforcement learning agents. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. AAAI Press, 3756–3762.

[16] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. 2018. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research* 61 (2018), 523–562.

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

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[19] Andrew Y Ng, Stuart J Russell, et al. 2000. Algorithms for inverse reinforcement learning.. In *Icml*, Vol. 1. 2.

[20] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. 2016. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277* (2016).

[21] Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. 2018. Robust deep reinforcement learning with adversarial attacks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2040–2042.

[22] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. 2017. Robust Adversarial Reinforcement Learning. In *International Conference on Machine Learning*. 2817–2826.

[23] Avraham Ruderman, Richard Everett, Bristy Sikder, Hubert Soyer, Jonathan Uesato, Ananya Kumar, Charlie Beattie, and Pushmeet Kohli. 2018. Uncovering Surprising Behaviors in Reinforcement Learning via Worst-case Analysis. (2018).

[24] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815* (2017).

[25] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[26] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[27] H Zhang, J Wang, Z Zhou, W Zhang, Y Wen, Y Yu, and W Li. 2018. Learning to design games: Strategic environments in reinforcement learning. In *IJCAI International Joint Conference on Artificial Intelligence*, Vol. 2018. ArXiv, 3068–3074.