COPYCAT: TAKING CONTROL OF NEURAL POLICIES WITH CONSTANT ATTACKS

Anonymous authors
Paper under double-blind review

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 the latter, 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 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 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 the agent's inner state. 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 inner 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, 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 (Papernot et al., 2016). 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 inner 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 inner state by stacking the last four observations. Attacking the agent's inner 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. 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.

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 (Mnih et al., 2015; Hessel et al., 2018) 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 parametrize the action-value function Q_ω with a neural network of parameters ω and deep actor-critics (Mnih et al., 2016) 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) (Goodfellow et al., 2015) 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 $-l(\theta, x + \eta, y_{target})$ 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:

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

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 inner 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 $\Delta = \{\delta_i\}_{1 \leq i \leq |\mathcal{A}|}$ than can be used to drive a policy π to follow any policy π^{target}. Each additive mask δ_i is pre-computed to lure π into taking a specific action a_i 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 (Kurakin et al., 2016) to solve the optimization problem.

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

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

We restrict the method to the case where f, the function building the agent's inner state from the observations, is differentiable. Eq. 3 is optimized by alternating between gradient steps with adaptive learning rate (Kingma & Ba, 2014) and projection steps onto the L_∞-ball of radius ϵ.

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} given the sequence of observations and the corresponding mask $\delta_{a_t^{\text{target}}} \in \Delta$ is applied to the last observation o_t before being passed to the agent.

4 RELATED WORK

Vulnerabilities of neural classifiers were highlighted by Szegedy et al. (2013) 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. (2017) uses *fast-gradient-sign method* (Goodfellow et al., 2015), 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 backpropagating through the agent's network, which is not feasible in real-time. Moreover, it modifies

directly the inner 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. (2017) allows the number of attacked states to be divided by four, yet it uses the heavy optimization scheme from Carlini & Wagner (2017) for crafting 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. (2018) 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. (2017) 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. (2018) and Ruderman et al. (2018) 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 (Mnih et al., 2015). The policy π^{target} is trained with Rainbow (Hessel et al., 2018). We select Atari games (Bellemare et al., 2013) 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 do 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., 2000). In the latter, it is 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.

As we set the objective of building a real-time targeted attack, we cannot compare our algorithm with the ones using a complete optimization scheme at each time step. Attacks would not be inferred fast enough to be used in a real-time scenario. The fastest state-of-the-art method can be seen as a variation of Huang et al. (2017). 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. (2017) 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.

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. (2017) showing how behaviors of π under attacks match π^{target} when these attacks are of equal energy.

Experimental setup We always turn the sticky actions on, which make the problem stochastic (Machado et al., 2018). 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 (Kingma & Ba, 2014) 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.

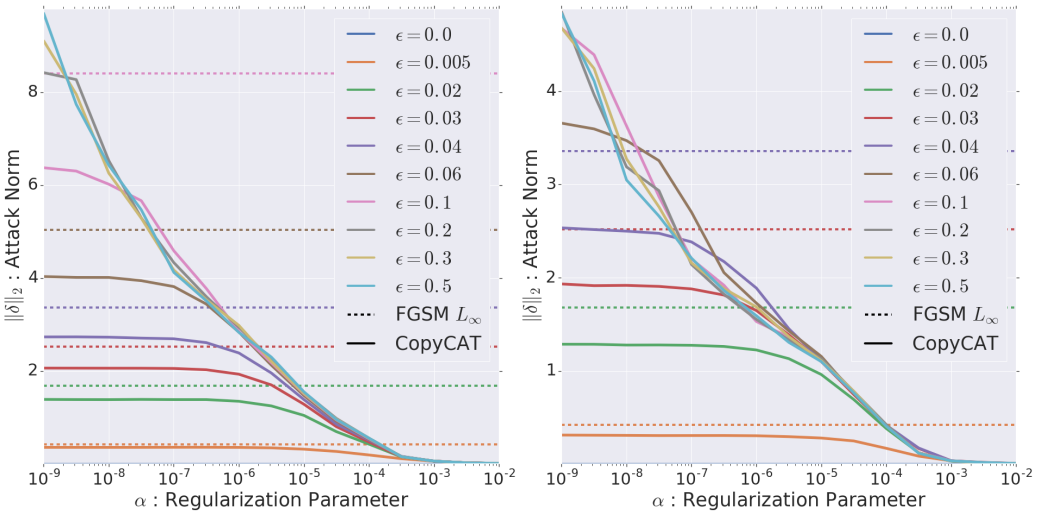

Figure 1: Influence of parameters ϵ and α on the maximal L_2 norm of Δ: $\max_i \|\delta_i\|_2$. On the left, attacks are computed on HERO. On the right, they are computed on Space Invaders.

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.

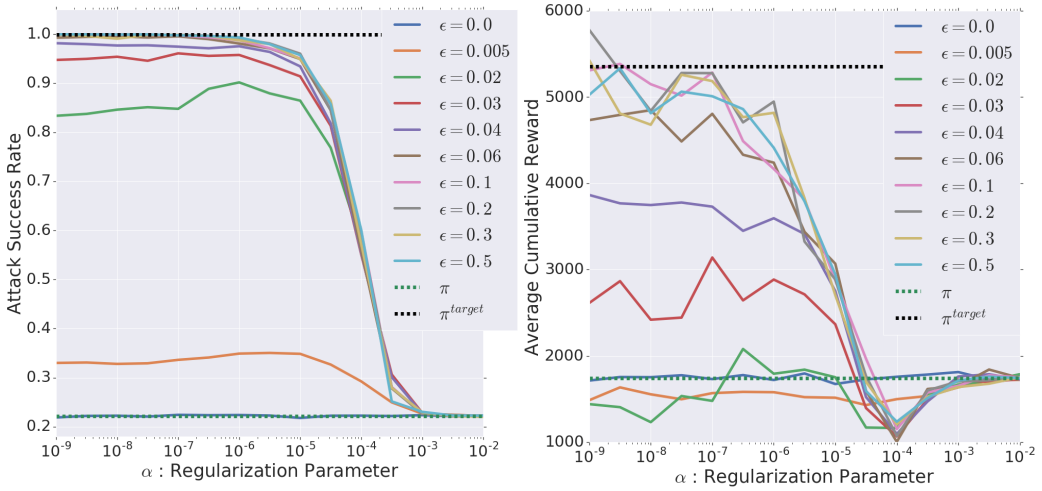

Figure 2: Influence of parameters ϵ and α on the average behavior of the agent playing Space Invaders under CopyCAT attack . On the left, the attack success rate. On the right, the average cumulative reward.

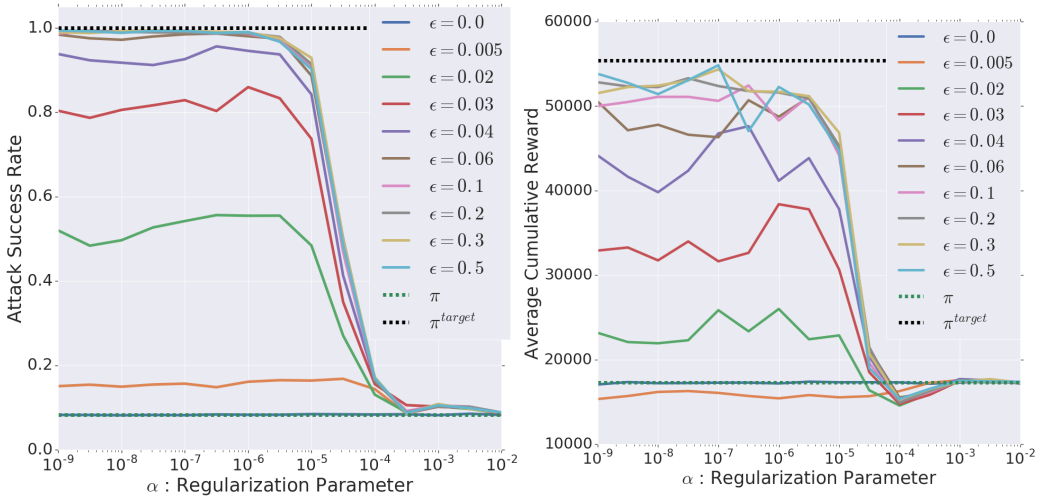

Figure 3: Influence of parameters ϵ and α on the average behavior of the agent playing HERO under CopyCAT attack . On the left, the attack success rate. On the right, the average cumulative reward.

We show on Fig. 2 and 3 (two 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 $\rho(\pi^{\text{target}})$.

Overall, we observe on Fig. 2-right and Fig. 3-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. (2017) 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 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?

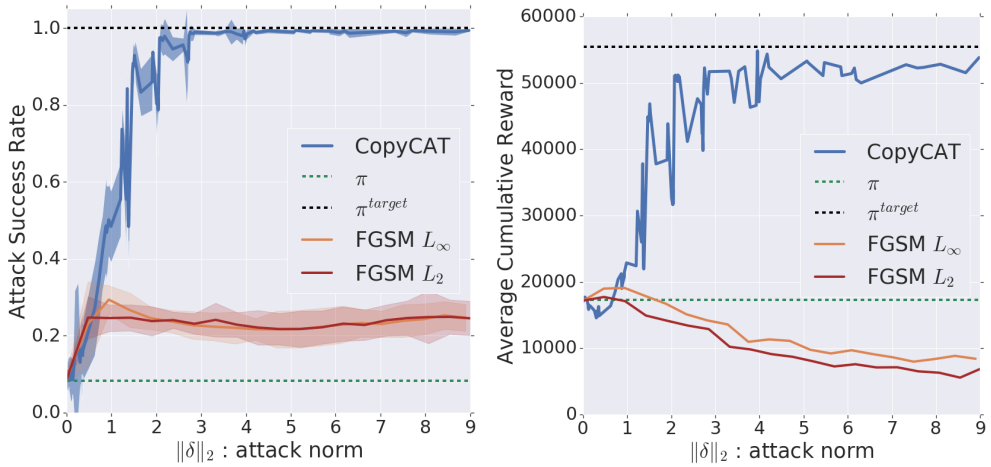

Figure 4: CopyCAT against FGSM for policy targeted attacks in HERO. On the left, the attack success rate. On the right, the average cumulative reward under attack.

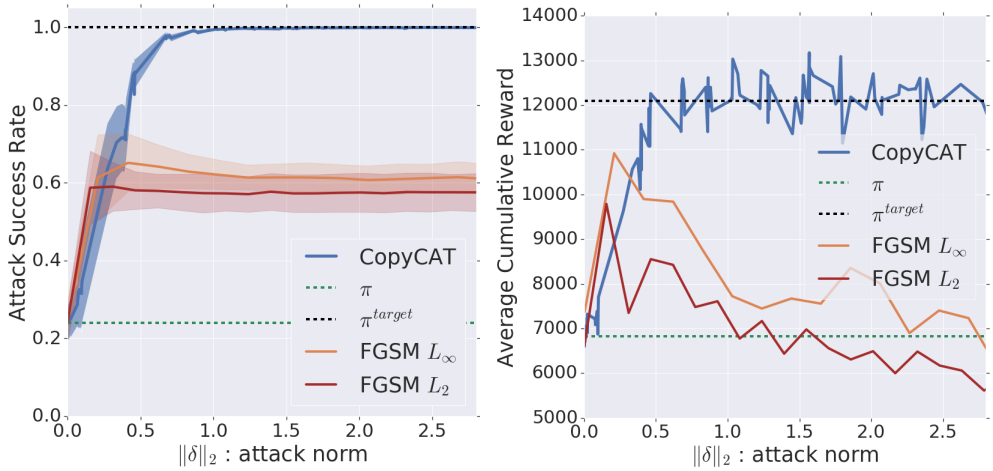

Figure 5: CopyCAT against FGSM for policy targeted attacks in Air Raid. On the left, the attack success rate. On the right, the average cumulative reward under attack.

We show on Fig. 4 and 5 (two 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: $\max_i \|\delta_i\|_2$ 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. 4-left and Fig. 5-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. 4-right and Fig. 5-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. An example of CopyCAT is shown on Fig. 6. The patch δ_i aiming at action "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). On the left, the unattacked observation. In the middle, the attacked observation. Below the images, the action taken by the same policy π when shown the different situations in an online setting.

Figure 6: 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

6 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.

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 test 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 and would probably require to train imitation models to obtain mappings similar to π in order to compute transferable adversarial examples.

REFERENCES

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.

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

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

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. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

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. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pp. 3756–3762. AAAI Press, 2017.

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

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

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.

Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, pp. 2, 2000.

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

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

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *International Conference on Machine Learning*, pp. 2817–2826, 2017.

Avraham Ruderman, Richard Everett, Bristy Sikder, Hubert Soyer, Jonathan Uesato, Ananya Kumar, Charlie Beattie, and Pushmeet Kohli. Uncovering surprising behaviors in reinforcement learning via worst-case analysis. 2018.

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

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

A APPENDIX

In order to keep the core paper not too long, we only showed a subset of the results, they are all provided here. No additional experiments are made. The explanations and interpretations can be found in the paper. **Left:** HERO. **Center:** Space Invaders. **Right:** Air Raid.