Are AlphaZero-like Agents Robust to Adversarial Perturbations?

Anonymous Author(s)
Affiliation
Address
email

Abstract

AlphaZero (AZ) has demonstrated that neural network-based Go AIs can surpass human game performance by a large margin. Even without any Monte Carlo tree search (MCTS), the Policy-Value neural network (PV-NN), which served as a heuristic in AZ, achieves comparable performance to professional players that most humans cannot reach. However, the robustness of those AZ agents and their PV-NNs has not been studied in the literature. Although it is well known that convolutional networks used in computer vision are not robust, existing adversarial attacks cannot be directly applied due to the difficulty of defining semantically invariant perturbations. Further, the discrete nature of Go prevents the use of efficient gradient-based attacks. In this paper, we develop the first adversarial attack on AZ agents of Go. We show that both PV-NNs and AZ agents with few simulations can be fooled by adding one or two irrelevant stones. For example, on 58% of the AlphaGo Zero self-play games, our method can make the widely used KataGo agent with 50 simulations play a losing action by adding two meaningless stones on the board. Moreover, these mistakes are so obvious that even normal humans can independently interpret them. In the experiments, we use the proposed method to examine the robustness of four publicly available Go AZ agents and one NoGo AZ agent. The results show that those agents with few simulations are vulnerable and will make mistakes way below their level.

1 Introduction

AlphaZero (AZ) ([1]) like algorithms have achieved the state-of-the-art in Go – one of the most challenging games for artificial intelligence. One of the main reasons is that they well train their Policy-Value Neural Network (PV-NN) with reinforcement learning (RL) and then use it to guide the Monte Carlo tree search (MCTS) [2, 3]. According to [4], PV-NN can achieve human professional level (Elo 3055) even without any lookahead. Nowadays, even professional players trust AI agents and endeavor to memorize their strategies. However, are these AIs always better than humans?

Despite its significant impact, the robustness of AZ agents and its PV-NN network is under-explored. Recently, it has been observed that many deep neural network models can be easily fooled by small and human-imperceptible perturbations to the input [5, 6], but it is still unknown whether there exist "adversarial examples" for AZ agents and how easy we can find them. Although adversarial attack algorithms have been developed for computer vision [6], NLP [7, 8], and reinforcement learning [9, 10, 11, 12], none of the existing methods can be directly applied to attack Go AIs due to the difficulties of defining "semantically invariant perturbation" in discrete games like Go.

In this paper, we proposed examining the robustness of agents for games like Go with "action perturbation" that satisfies the following constraints. First, the perturbed state should be a legal state that is "close to" one of the given nature states. If an example is not similar to a natural state, then it

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

will not be surprised that an agent is malfunctioning. However, unlike continuous states, discrete states cannot simply add a small perturbation to a state like changing the x-axis position from 1 to 1.01. Therefore, we use actions as the perturbation and the number of actions we added as distance to define which states are "close". Second, the perturbed state should also be semantically identical to the original state, and the equivalence should be verifiable to humans. This constraint is to mimic the adversarial attacks on image classification, which demands the perturbation should be small enough and human imperceptible. We thus constrain the added actions to be "meaningless" (which will be formally defined in the paper). By adding a limited number of "action perturbations", we aim to find states that the target agent will make mistakes (e.g., predicting a wrong move) way below its normal playing strength. Take Fig. 1 as an example, B19 and P1 (marked as 1 and 2) are the two meaningless actions we added. They are meaningless because both actions do not help its player gain any territory. However, those actions are able to let KataGo [13], a well-known Go AI with 50 simulations, "forget" to kill the white stones marked with squares (action A), and instead want to save the black stones marked with triangles (action B), even if doing so will lose the game. The contributions of this paper can be summarized below:

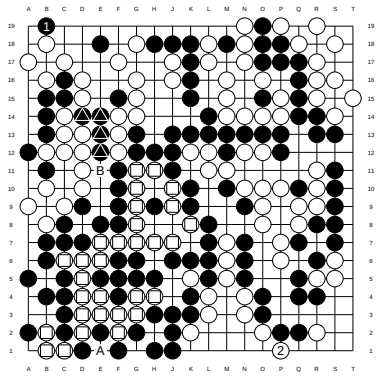


Figure 1: An adversarial example we found that can attack KataGo with 50 MCTS simulations. The example is perturbed from an AlphaGo Zero self-play record. Actions B19 and P1 (marked as 1 and 2) are the meaningless actions we add as the perturbation. In this state, the black player is supposed to play at position A to kill all the white stones marked with squares. However, confused by the actions we added, the agent ends up playing B to save the four black stones marked with triangles and gives up the opportunity to occupy way more territories by playing position A.

- For the first time, we reveal the vulnerability of both PV-NN and AZ agents. With the proposed "action perturbation", the adversarial examples we found (e.g., Fig. 1) are semantically equivalent to a natural state while leading to catastrophic behavior of the target agent. Such examples may be explored during large-scale MCTS and end up misleading the search.
- We design an efficient method to speedup the searching. Our algorithm is usually more than 100 times faster than brute force search in the experiments.
- We conduct a comprehensive study on four state-of-the-art public AZ agents with six different datasets to reveal their robustness in the game on Go. The results demonstrate that all these agents with a small number of simulations are vulnerable. In particular, by adding two meaningless moves, our attack can consistently achieve $\geq 90\%$ success rates on the PV-NNs and achieve $\geq 58\%$ on MCTS Agents with 50 simulations, which is the exact simulation count used by MuZero [14]. Moreover, our method can also find bugs of AZ agents in other games since every game's state can be modified by actions. For example, in our NoGo [15] experiment, adversarial examples are found on 50% of data.

2 Background and Related Works

Terminology Games like Go can be described as a two-player zero-sum deterministic game [16] and can be defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ is the transition function and $\mathcal{R} : \mathcal{S} \mapsto \{1, 0, -1\}$ is the reward. Each game starts from an initial state $s_0 \in \mathcal{S}$ (empty board) at time 0. At the current state s_t, the turn-player, defined as $s_t.c$, can play an action $a_t \in \mathcal{A}(s_t)$. In Go, players take turns placing a stone of his color on the board. Therefore, all actions are composed of a color c and a position p indicating placing a c stone on p. The only exception is the pass move a_{pass}, which means the turn-player does not place any stone (gives up on his turn). After playing action a_t on state s_t, we can get the next state by $s_{t+1} = \mathcal{T}(s_t, a_t)$. The game ends when reaching a terminal state s_T, where we can determine who wins the game and each player receives a reward. Since the games are zero-sum, the reward of opponent player is $-\mathcal{R}(s)$. For the non terminal states, there will be no reward $\mathcal{R}(s) = 0$ for any player.

Policy Value Neural Network (PV-NN) and Policy Value MCTS (Pv-MCTS) PV-NN is proposed in AlphaGo Zero paper [4]. PV-NN takes a state s as input and outputs a value $v(s)$ and a policy $p(s)$. Value $0 \leq v(s) \leq 1$ is a scalar from 0 to 1 that indicates the estimated win-rate of current state. For example, $v(s) < 0.5$ means that PV-NN believes the turn-player of state s is losing. Note that if the value output range is $-1 \leq v(s) \leq 1$, the win-rate can be calculated by $(v(s) + 1)/2$. Policy $p(s)$ is a vector that indicates each action a's probability $0 \leq p(a|s) \leq 1$. A high probability for an action, e.g., $p(a|s) > 0.9$, indicates that the the PV-NN highly recommends action a at state s. Besides PV-NN, AlphaGo Zero [4] also adopted Policy Value MCTS (PV-MCTS) to choose the best action. Different from MCTS, PV-MCTS uses the value output of PV-NN instead of Monte Carlo simulation to evaluate the selected state values. Moreover, PV-MCTS utilized the policy of PV-NN to narrow down the search space with the PUCT algorithm [17]. The search efficiency of PUCT highly depends on the prior probability $P(s, a^*)$ of best action a^*. Therefore, if the root's prior policy is incorrect, the search efficiency will drop dramatically. Take Fig. 1 as an example, even after 50 simulations, KataGo still couldn't explore action A because the prior probability of action A is too small. As more states have been evaluated by PV-NN, PV-MCTS can provide stronger policy $\pi(s)$, value $V(s)$, and action values $Q(s, a)$ of a given root state s.

Adversarial Example It has been observed that many neural networks used in computer vision, NLP and reinforcement learning are vulnerable to adversarial examples [5, 18, 9]. Traditionally, an adversarial example, created by minimally modifying a natural example, is semantically equivalent to the original natural example for humans but can make the target model produce a totally different output. Note that, the perturbed state should also look natural [19]. To define an adversarial example for a particular task, one has to define a reasonable perturbation set around the original example, and design an algorithm to find an example within the set that leads to incorrect behavior of the model (e.g., mis-classification). In computer vision, the perturbation set is usually defined as a small ℓ_p norm ball, as small perturbations to images are usually imperceptible to human. And due to the continuous search space, existing image attacks often rely on gradient-based optimization to find adversarial examples [20, 21, 22, 23]. On the other hand, for text models, there exists multiple definitions for the perturbation set such as synonym substitution [24, 25], edit distance [26], or language model based scores [27]. And due to the discrete natural of text domain, finding an adversarial example leads to a combinatorial search problem, and several attacks have been proposed for this [28, 29, 24]. However, none of these attack methods can be directly applied to attack AZ agents for the following reason: 1) Unlike image or NLP domain, the semantically invariant perturbation in Go is difficult to define (see Sec. 3.1). 2) Unlike other applications, the AI's ability is much stronger than human in Go, so it is non-obvious how to find adversarial examples that can be understood by human. 3) To find an adversarial example in a game, one needs to go through all the states in the trajectory which introduces much higher complexity than attacking a classification model. Although some recent works have studied perturbations to states [12], actions [30], observations [9, 10, 31, 32] or rewards [33] in the reinforcement learning setting, none of them consider discrete input domains with a combinatorial large search space like Go or against planning based agent like AZ agents.

Blind Spots To improve Go AIs, some researchers [13] try to find their "blind spots". For example, from famous Tesuji (brilliant moves) problems, they can find some states such that the prior probability of the best action is almost 0. However, these blind spots are not adversarial examples since the losing policy of the agent is still reasonable for humans even when knowing the best actions, and hence they are hard states for both humans and AI. Further, there is no automatic and systematic way to find those scarce blind spots. On the other hand, we show that $\geq 58\%$ of the games we selected exist adversarial examples that normal humans can be better than state-of-the-art AIs. By finding "bugs" efficiently, researchers may have enough training data to improve their AIs.

3 Method

Given a state s, a target agent (e.g., PV-NN or AZ agents), and a much weaker verifier (e.g., humans), our goal is to find an adversarial example s' that satisfies the following conditions:

- C_1 The perturbed state s' is very close to the original state s in terms of ℓ_0 distance.
- C_2 The perturbed state s' is semantically equivalent to s, and the verifier can verify that.
- C_3 The target agent performs correctly on s but wrongly on s', and the verifier can identify that.

Both C_1 and C_2 define a set of perturbed states, $\mathcal{B}(s)$, for a given natural state s. The problem of finding adversarial examples is then equivalent to finding a $s' \in \mathcal{B}(s)$ that satisfies the success criterion (C_3). In previous image attacks, $\mathcal{B}(s)$ is a small ℓ_p ball [6, 20] and in text attacks $\mathcal{B}(s)$ is usually defined by word substitutions with synonyms [24]. However, in our case the definition of $\mathcal{B}(s)$ is much more sophisticated and is computational expensive to check. The existence of a much weaker verifier ensures that the target network's mistake is so simple that even the verifier can verify it without knowing the examiner's output (but knows about target agent's outputs of s and s'). Hence, the playing strength of the verifier determines the difficulty of finding adversarial examples. The weaker the verifier is, the harder it is to find an example. In addition, the algorithms are supposed to find the s' without using the verifier since verifiers like humans are hard to include in an automatic process. Therefore, we need to carefully design the $\mathcal{B}(s)$ and the success criterion. In the following, we will discuss how to define $\mathcal{B}(s)$ and the success criterion in Subsections 3.1 and 3.2, respectively. A search algorithm will then be introduced in Subsection 3.3 to speedup the search.

3.1 Defining the Perturbation Set

We define the perturbation set $\mathcal{B}(s)$ of a given state s based on conditions C_1 and C_2. For C_1, we use actions as perturbation and the number of actions we added as distance. In our experiments, we set distance ℓ_0 as 2. Since each state in Go can be represented by its trajectory (a list of actions), the perturbed state s' can be presented as

$$a_0, a_1, \ldots, a_{t-1}, b_0, b_1,$$

where $a_0, \ldots, a_{t-1}$ is the trajectory of s and b_0, b_1 are the extra actions. For **1STEP** attack, one of the b_0, b_1 has to be the pass action (denoted as a_{pass}). This means we place an additional stone on the board without changing the turn player. On the other hand, for **2STEP** attack, both b_0, b_1 are not the pass action, which means we add one black and one white stone on the board. We do not include states created by replacement like word substitution in text attacks, since there are no similar actions in Games like Go.

For condition C_2, we aim to maintain the semantic meaning after perturbation. Since there is no ground truth in Go, we resort to an "examiner" agent to verify the equivalence of states. The examiner should be able to provide an accurate value $V(s)$, the policy $\pi(s)$, and the action values $Q(s, a)$ for a given state s. For example, the examiner we used is the strongest PV-MCTS agent that runs with 800 simulations. To distinguish the output of the examiner and the target agent, we use $v(s)$ and $p(a|s)$ as the output of the target agent. With the examiner, we define two states s, s' are semantically equivalent if the two states have the same turn players and similar win rate according to the examiner, which is shown in the following equation.

$$s.c = s'.c \text{ and } |V(s) - V(s')| \leq \eta_{\text{eq}}, \tag{1}$$

where $s.c$ denotes the turn player (c for color) of state s and η_{eq} is the threshold used to define that the win rate is close. However, we might miss some of the equivalent states since the examiner is not perfect and may be incorrect when evaluating $V(s')$ in Eq. 1. After adding perturbed actions, the examiner's prior policy might "forget" to consider the best action a_s^* of s and thought s' is losing. Therefore, we improved the performance of examiner on s' by providing a_s^* as hint. To be precise, we evaluate the value of s' by calculating $\max(V(s'), 1 - V(\mathcal{T}(s', a_s^*))$. This is identical to forcing the examiner to consider a_s^* on s'. This method makes it possible to find adversarial examples even if the target agent is the same as the examiner.

Besides ensuring s, s' to be semantically equivalent, such equivalence should be verifiable to the much weaker verifier like humans. Unlike standard image or text applications where human is treated as an oracle, the AI agents for Go are much more powerful than an human. When the game is too complicated, humans cannot even tell who has the advantage. Therefore, even though both states s and s' are semantically equivalent (Eq. 1) to examiner, the verifier may not be able to tell. Fortunately, we observe that, in the 1STEP attack where one of b_0, b_1 is the pass action, the action we added is easy for humans to understand that it does not affect the win rate. In other words, human players can identify "meaningless" actions which do not benefit the turn-player of the action. On the other hand, in the 2STEP attack, it is harder for humans to verify whether the benefit of b_0 for the turn player is the same as the benefit of b_1 for the next player in high-level games. Therefore, in 2STEP attack, we enforce the action we add to be meaningless: $|V(s) - V(F(s, a))| \leq \eta_{\text{eq}}$, where $F(s, a) : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ return the state after playing action a on state s without changing the turn

player by skipping the opponent's turn (Appendix F). In addition, in the game of Go, we found that most meaningless actions are in one of the player's territories (Appendix E). Also, humans can usually verify such actions faster. Therefore, in the experiments of Go, we further restrict the perturbation action's position $a.p$ within one of the territories $a.p \in \text{Terr}_W \cup \text{Terr}_B$, where Terr_W and Terr_B is the territory of each color. The results show that even with this stronger constraint, we can find adversarial examples. See Appendix G for the comparison without the territory constraint.

3.2 Success Criteria of Adversarial Attack

Next we define the success criteria (C_3) for our attack. We consider two types of attacks: value attack and policy attack. For **value attack**, the goal is to change the prediction of the target agent's value network. Therefore, we define the attack to be successful if

$$|v(s) - V(s)| \leq \eta_{\text{correct}} \text{ and } |v(s') - V(s')| \geq \eta_{\text{adv}}. \tag{2}$$

The first criterion ensures that the target agent produces the correct value on the original state s, and the second criterion ensures that the target value network becomes incorrect after the perturbation. Note that we already enforces $V(s) \approx V(s')$ in Subsection 3.1, so (2) implies that the perturbation will change the target agent's value but not the examiner's value. Constants η_{correct} and η_{adv} are the thresholds to define "correct" and "wrong" in C_3. For example, if we want to find an example that PV-NN misclassified the winner of a state, we can set $\eta_{\text{adv}} = 0.5$. We can further increase η_{adv} if we want the target's output to be wrong by a larger margin. In addition, the verifier can easily tell that target is wrong since s, s' are suppose to have the same winrate (C_2) but $v(s)$ and $v(s')$ are different.

For **policy attack**, we aim to fool the policy output. However, unlike normal classification, there is more than one best action for a state. Therefore, even when the perturbation can significantly change the output policy, it doesn't mean the new policy is incorrect. We thus have to define the successfulness of the policy attack by checking whether the value (computed by the examiner) will be changed after playing the predicted action. With these in mind, we say the policy attack leads to a "wrong" move if

$$|Q(s, a_s^*) - V(s)| \leq \eta_{\text{correct}} \text{ and } |Q(s', a_{s'}^*) - V(s')| \geq \eta_{\text{adv}} \tag{3}$$

where $a_s^* = \arg\max_a(p(a|s))$ and $a_{s'}^* = \arg\max_a(p(a|s'))$ are the target agent's recommended actions on s and s'. The first criterion in (3) ensures that the target agent can predict a proper action for the original state s, and the second criterion checks whether the target agent will output a significantly worse action in the perturbed state s'. With Eq (3), although the examiner can verify that the target agent plays a losing action on the perturbed state; the verifier may be uncertain that the action $a_{s'}^*$ is truly bad. Luckily, in our qualitative study, by providing a_s^* to humans as a hint, humans can immediately tell that a_s^* is much more important to play than $a_{s'}^*$ in the state s' and certify that the target agent is wrong on most pairs (s, s') we found. Hence, we add two more restrictions: $|Q(s', a_s^*) - V(s')| \leq \eta_{\text{eq}}$, and $|Q(s, a_{s'}^*) - V(s))| \geq \eta_{\text{adv}}$. These restrictions ensure that a_s^* is a good action and $a_{s'}^*$ is a bad action for both s and s', so that the verifier can use them as an anchor.

3.3 An Efficient Attack Algorithm

Like other adversarial attacks on reinforcement learning [9, 10], we apply our attack to perturb a state in a given game, since it only takes one mistake for an agent to lose a game. Formally, given a game $G = \{s_0, s_1, \ldots, s_T\}$, if we can find one pair of (s_i, s_i'), where s_i' is an adversarial example of s_i, then we have successfully attacked the agent on that game. We will first apply the 1STEP attack to a game, and if it fails, we then conduct the 2STEP attack.

Since the search space is countable, a naive way to find an adversarial example is to conduct a brute-force search to check all states in the search space. However, this will lead to very high complexity and the search cannot be finished in practice. Taking the 2STEP attack as an example, let $T \approx 300$ denote the game length, $N \approx 150$ denote the average number of actions for a state, and $M \geq 800$ denote the simulation count of the examiner. Then, we need $O(TNM)$ time to obtain the search space for all $\mathcal{B}(s_i), i \in \{0, 1, \ldots, T\}$ by checking which actions are meaningless. Furthermore, assume that there are averagely $\bar{N}$ *meaningless moves* for each state. Then, we need $O(T\bar{N}^2M)$ time to check the success criteria of each perturbation. Note that in both stages the running time of examiner will dominate, as we need to run the examiner with much more MCTS steps. Hence, we propose the following two approaches to reduce the search complexity.

First, we reduce the time complexity of finding meaningless actions by the following observation. If an action a is a meaningful action (not meaningless) of state s_t, then it is likely that a is also a meaningful action of state s_{t-1}. Intuitively, if an action a is a meaningful action at s_t, it means that a can occupy some extra territory that is not occupied at s_t. Hence, if a position is not occupied in s_t, it is likely that the position is not occupied in s_{t-1} and can be occupied by action a too. So action a is also a meaningful action for s_{t-1}. We formally prove this property under some assumptions in Appendix A. Based on this observation, we run the search in the backward manner from the final state s_T to the initial state s_1. Once an action is identified as meaningful at state s_t, we do not need to check it again on any state $\{s_i : i < t\}$. This will significantly save the computational time to enumerate $\mathcal{B}(s)$. The details of getting meaningless actions are shown in Appendix C.

Algorithm 1 Two-Step Value Attack

1: **Input:** a game $s_1, s_2, \ldots, s_T$, target agent t, examiner e
2: **for** $i = T$ **to** 0 **do**
3: **if** $|e.V(s_i) - t.v(s_i)| > \eta_{\text{correct}}$ **then**
4: **continue**
5: cands = `getMeaninglessActions` (e, s_i)
6: **for** b_0 in cands[0] **do**
7: **for** b_1 in cands[1] **do**
8: $s' = \mathcal{T}(\mathcal{T}(s_i, b_0), b_1)$
9: **if** $|t.v(s') - e.V(s_i)| \geq |\eta_{\text{adv}} - \eta_{\text{eq}}|$ **then**
10: **if** $|t.v(s') - e.V(s')| \geq \eta_{\text{adv}}$ **and** $|e.V(s) - e.V(s')| \leq \eta_{\text{eq}}$ **then**
11: **return** s'
12: **return NULL**

Second, for the part of checking the attack success criterion, we derive a bound to quickly filter out unsuccessful perturbations. Taking the value attack as an example, checking (1) and (2) requires running the examiner on the perturbed state s', which is the bottleneck of the algorithm. To reduce this cost, we show that the condition of (1) and (2) implies

$$\begin{aligned} |v(s') - V(s)| &= |(v(s') - V(s')) - (V(s) - V(s'))| \\ &\geq |v(s') - V(s')| - |V(s) - V(s')| \geq |\eta_{\text{adv}} - \eta_{\text{eq}}| \end{aligned} \tag{4}$$

Since (4) only requires running examiner on the original state s instead of s', we can check (4) first before checking (1) and (2). We observe this step can filter out more than 99% of the s'.

The overall attack algorithm is presented in Algorithm 1. The input includes a game $s_1, s_2, \ldots, s_T$, a target agent t, and an examiner agent e. As mentioned before, the search is conducted in a backward manner from s_T back to s_1 (line 2). For each s_i, we first check whether s_i is too hard for the target (line 3). If so, we will skip this state (line 4). We then compute all the meaningless actions of state s_i using the examiner with the efficient implementation mentioned above and store the meaningless actions separately in $cands[0]$ and $cands[1]$ according to their color (line 5) as the candidates of perturbation. We then check whether each 2STEP perturbation will lead to a successful attack (line 9-11). Note that line 9 is based on (4) and examiner does not need to compute $e.V(s_i)$ again since it has evaluated it on line 3. For policy attack, we can skip all the states that are losing $V(s) < \eta_{\text{adv}}$ since for those states there is no correct answer to attack. (More details are in Appendix B).

Table 1: Attack results on different agents with all datasets. EXECUTED NUM shows the average number of target and examiner calls for the attack, and SPEEDUP indicates the speedup of the proposed method over the brute-force search.

		SUCCESS RATE		EXECUTED NUM		
	AGENT	1STEP	2STEP	TARGET	EXAMINER	SPEEDUP
VALUE $\eta_{adv} = 0.5$	KATAGO	0.99	1.00	6152	68	80
	LEELA	0.90	0.98	37533	182	163
	ELF	0.92	1.00	11108	108	90
	CGI	1.00	1.00	40	2	14
VALUE $\eta_{adv} = 0.7$	KATAGO	0.86	0.94	44638	299	125
	LEELA	0.70	0.84	96134	518	150
	ELF	0.65	0.87	42503	398	94
	CGI	1.00	1.00	45	2	15
POLICY $\eta_{adv} = 0.5$	KATAGO	0.70	0.92	128228	666	155
	LEELA	0.92	0.97	17218	118	123
	ELF	0.93	0.97	16387	113	122
	CGI	0.87	0.96	16495	103	133
POLICY $\eta_{adv} = 0.7$	KATAGO	0.55	0.82	231895	707	232
	LEELA	0.75	0.93	36438	150	186
	ELF	0.77	0.88	31258	159	157
	CGI	0.75	0.89	24915	139	146

Table 2: Attack results on different datasets. EXECUTED NUM shows the average number of target and examiner calls for the attack. The first five datasets are described in experiment settings. FOX is the dataset of non-professional players.

		SUCCESS RATE		EXECUTED NUM	
	GAMES	1STEP	2STEP	TARGET	EXAMINER
VALUE $\eta_{adv} = 0.7$	ZZ	0.78	0.89	14691	316
	ZM	0.90	0.95	12977	197
	MH	0.60	0.81	173844	663
	LG	0.85	0.95	12650	161
	ATV	0.88	0.95	14988	185
	FOX	0.83	0.94	53792	213
POLICY $\eta_{adv} = 0.7$	ZZ	0.57	0.82	33314	502
	ZM	0.84	0.94	13374	178
	MH	0.51	0.80	227202	258
	LG	0.84	0.96	52055	274
	ATV	0.75	0.88	79688	232
	FOX	0.75	0.87	141120	225

4 Experimental Results

Experiment Settings We evaluate our method on 19x19 Go with four GPUs (GTX 1080Ti). The four open-source programs we used are KataGo [13], Leela Zero [34], ELF OpenGo [35], and CGI [36]. The strengths of these AI agents are KataGo $\gg$ Leela $>$ ELF $=$ CGI, where KataGo has more than 99% winrate against Leela when both using 800 MCTS simulations. Since KataGo is the strongest agent, we use its 40 blocks PV-NN with 800 simulations as our examiner. For the thresholds, we set $\eta_{\text{eq}} = 0.1$, $\eta_{\text{correct}} = 0.15$, since after testing several different η_{eq} and $\eta_{correct}$ values, this pair lead to more human-understandable results. For the datasets, we selected 99 games from five different sources, which are AlphaGo Zero 40 blocks training self-play record (ZZ), AlphaGo Zero vs AlphaGo Master (ZM), AlphaGo Master vs Human champions (MH), the final games of LG Cup World Go Championship (2001-2020) (LG), and the final games of Asian TV Cup (2001-2020) (ATV). ZZ and ZM present the games played by AIs. MH presents the games of AI vs Humans. LG and ATV present the games of humans. Note that the thinking time for ATV Cup is much shorter than LG Cup, so we expect them to reflect human games with different strengths. All the datasets have 20 games, except ZZ has 19 games since the first game is played by two random agents. After removing states that are hard for examiner's PV-NN to verify ($|e.v(s) - e.V(s)| > \eta_{\text{eq}}$) and the meaningful actions, the average search space of each game contains about $600,000$ states.

Results on Different PV-NNs We first evaluate the robustness of the four agents' PV-NNs since PV-MCTS is much slower. The results are shown in Table 1, where we attack each agent's PV-NN with both 1STEP and 2STEP attacks with various η_{adv}. We also present the average number of evaluations required for the target agent and the examiner agent. The first two groups in Table 1 demonstrate the robustness of these agents against the value attack. Group one ($\eta_{\text{adv}} = 0.5$) shows that even the 1STEP attack can achieve above 90% success rate on all agents and mostly achieve 100% on the 2STEP attack. For those that has the same attack success rate, we can still compare the attack difficulties by the number of target evaluations. For example, although the 2STEP success rates of KataGo, ELF, and CGI are all 100%, the number of states that they have visited is totally different. For CGI, we are able to find an adversarial example after visiting 40 states, while ELF requires visiting $11,108$ states. Hence, we conclude that Leela $>$ ELF $>$ KataGo $>$ CGI in terms of their robustness against value attack when $\eta_{\text{adv}} = 0.5$ and $\eta_{\text{adv}} = 0.7$. Interestingly, this ranking does not match the playing strength of each agent. The third and fourth groups of Table 1 shows the results of policy attack. In general, we observe that it is harder to attack the policy than the value. Interestingly, the ranking of four agents in terms of their policy's robustness is KataGo $\gg$ Leela $>$ ELF $\approx$ CGI, which aligns with the playing strengths of those agents. In addition to Table 1, in order to check whether it is possible to change the PV-NN's output catastrophically, we conduct an extreme experiment on KataGo with $\eta_{\text{adv}} = 0.9$. The result shows that the 2STEP success rates of value and policy attacks still have 62% and 48%, respectively. In Table 1, the column of SPEEDUP demonstrates the speedup of our algorithm compared to the brute force algorithm. The results show that the proposed efficient search method is usually more than a hundred times faster than a brute-force search, especially for harder problems which require more 2STEP attacks.

Results on Different Datasets Besides the robustness of each agent, does the robustness vary between different types of game records? To answer this question, we consider the 5 datasets used in the previous subsection, plus an additional FOX dataset, which consists of games randomly selected from one of the most popular online Go playing platforms named FOX[1]. We add this dataset to represent amateur players and see if the games played by weaker players are harder to attack since those game are easier for human to understand. The results in Table 2 show that PV-NNs are vulnerable to all levels of games, regardless of AI agents, professional players or non-professional players. Fig. 3 shows two of the examples. We observe that the AlphaGo Master vs Human champions (MH) dataset has the lowest success rate on both policy and value attack compared to other datasets. Our hypothesis is that AlphaGo Master is much stronger than human champions (AlphaGo Master

[1] https://www.foxwq.com/

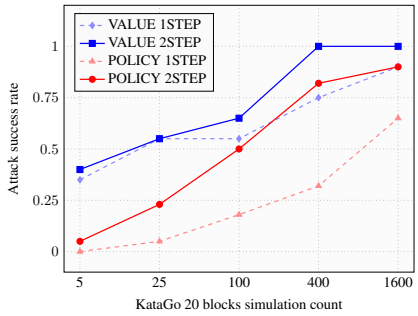


Figure 2: The attack success rate of $\eta_{\text{adv}} = 0.7$ on datasets that are generated by KataGo 40 blocks with 800 simulations vs KataGo 20 blocks with different number of simulations.

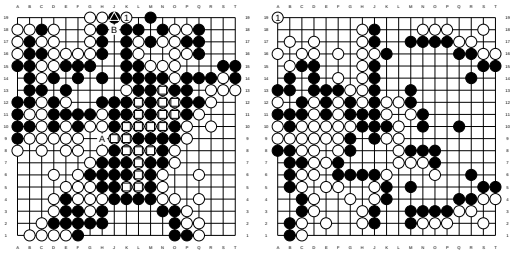
Table 3: KataGo with different simulations on ZZ dataset. EXECUTED NUM shows the average number of target and examiner calls for the attack.

		SUCCESS RATE		EXECUTED NUM	
	SIM	1STEP	2STEP	TARGET	EXAMINER
VALUE $\eta_{adv} = 0.5$	1	1.00	1.00	1016	41
	5	0.89	0.95	4454	2322
	10	0.89	0.95	6501	3147
	25	0.84	0.89	18516	13903
	50	0.53	0.58	76426	42974
POLICY $\eta_{adv} = 0.5$	1	0.84	1.00	18347	869
	5	1.00	1.00	3309	1318
	10	0.95	1.00	6666	2662
	25	0.79	1.00	12466	5487
	50	0.21	0.68	89512	41702

had played 60 games against human champions without losing any of them.) When one player is much stronger than the other, the winner of the game may be too obvious and easy for agents to judge. Also, it is harder for the policy to be weak enough to lose the game. To support this hypothesis, we generate games of KataGo 40 blocks (K40) with 800 simulations vs KataGo 20 blocks (K20) with different simulations (20 games for each simulation) to simulate games played by players with different strengths. Note that K20 is much weaker than K40 even with the same amount of simulations. Therefore, when K20 uses fewer simulations, the gap between two players is even larger. Fig. 2 shows the results of attacking KataGo's PV-NN with those games and $\eta_{\text{adv}} = 0.7$. For games that is generated by K20 only used 5 simulations, both policy and value output of KataGo's PV-NN are hard to attack. The 1STEP policy attack even fails for all of the 20 games. As we increase the simulation count of K20, the success rates of all the attacks increase. This suggests that the playing strength between players of a game will likely affect the hardness of finding adversarial examples.

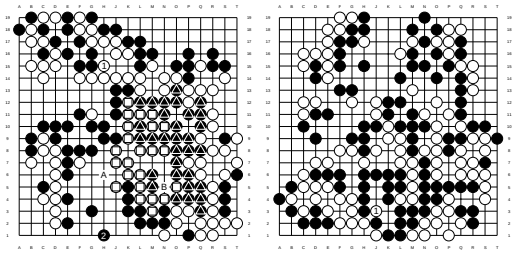
Robustness of PV-MCTS In this paragraph, we investigate the robustness of PV-MCTS with different simulations. Since PV-MCTS is much slower than PV-NN, we only test KataGo on the ZZ dataset with $\eta_{\text{adv}} = 0.5$. The results are shown on Table 3. The first column is the simulation count of KataGo. When simulation equal to one, it is the same as using PV-NN directly. The first group shows the results of value attack. Even with 25 simulations, the 2STEP success rate is still 89%. For group two, we observe that the policy of small simulations are even less robust than PV-NN's policy. For example, when the simulations of KataGo is 10, the 1STEP success rate is 95%, while the policy of PV-NN is 84%. Therefore, we conclude that with a small number of MCTS simulations, the agent will not be able to recover from the bad PV-NN outputs so will still be fooled by adversarial examples. However, when using more MCTS simulations, the attack success rates will still drop since the agent has more chances to discover correct actions.

Quality of adversarial examples To examine the quality of the adversarial examples we found, we randomly selected 100 examples from all the experiments and conduct the following two studies to evaluate the quality of adversarial examples found by our algorithms. Fig. 3 4 show four examples, and more can be found in Appendix H. In the first study, we aim to examine whether the perturbations (one or two additional stones) obtained by our attack method are meaningless to human. For each adversarial example, we present both the original board and the additional stones to 3 humans (with level 2k, 3d, 5d) who served as verifiers, and for 90% of those perturbations they can certify that the perturbations are meaningless with a short thinking time. The failure cases are the moves identified as meaningless by AI agents but it is not obvious to human. Fig. 4 (right) shows one of the states, where the meaningless action J3 seems meaningful to human since it can capture the black stone J2. However, with a longer thinking time and discussion with each other, verifiers can understand that the rest 10% of actions are meaningless. In the second study, we aim to examine whether the "wrong" actions resulted by our policy attack can be verified by verifiers. Similar to the previous experiment, we present the original board, original action, perturbed board and the action after perturbation to the examiners. When the target agent is a PV-NN, human can identify 100% that the recommended actions after perturbation will change the result from winning to losing. However, for the adversarial examples where the target is PV-MCTS agents, only 70% of the recommended actions



(a) (b)

Figure 3: Adversarial example of policy attack (a) and value attack (b) on FOX.



(a) (b)

Figure 4: Adversarial examples generated by policy attack (a) and value attack (b) that cannot be easily verified by human.

after perturbation can be identified as wrong by human. The main reason is that as PV-MCTS agents are able to lookahead, they tend to avoid actions that are clearly wrong, so the errors become more subtle to human. Fig. 4 (left) shows one of the states, where the white should play at A to save the stones marked with square, but with two meaningless moves added the PV-MCTS agent with 100 simulations will play at B. Although after playing B all the stones marked as square will be dead, it is hard for human to verify the outcome of this play.

Agents are sensitive to the ordering of actions in the trajectory Finally, after viewing those adversarial examples, we find that the policy attack succeeds often because the target agents are too reliant on the information of the last action. For example, if KataGo knows the last action of Fig. 1 besides the actions we add is white playing E3, even PV-NN knows that it needs to play E1. Based on this observation, we can simply improve the robustness of PV-NNs by the following method. Given a state s with trajectory $a_0, a_1, \ldots, a_t$, we define an augmented state $\bar{s}$ as $a_0, a_1, a_2, \ldots, a_{t-1}, a_t, a_{t-2}, a_{t-3}$. If $\bar{s}$ is a legal state, we can define a robust policy $p_r(a|s) = (p(a|s) + p(a|\bar{s}))/2$. With this method, PV-NNs can evaluated state s with different actions being the last one. We evaluated this method on the same setting as Table 3 (attacking KataGo's PV-NN with ZZ dataset and $\eta_{\text{adv}} = 0.5$). The success rate of 1STEP and 2STEP policy attacks dropped to 58% and 89%, even better than the original PV-NN and PV-MCTS with 25 simulations.

Experiments on the game of NoGo Our "action perturbation" method can also be used to find severe bugs automatically on other games by removing the territory constrain. We use the following 9x9 NoGo [15] experiments to demonstrate it. We use an AZ agent [37] with 800 simulations as the examiner and use its PV-NN as the target agent. Since NoGo is a game without any human experts, we use a traditional MCTS agent[38] with 1000 simulations as the weaker verifier. Note that the target agent has a 100% winrate against the verifier in 1000 games. Intuitively, the target network will not make a critical mistake that the verifier will not make. However, on 20 self-play games of the target network (the average search space is about 10000), 50% of the games exist adversarial policy examples with $\eta_{adv} = 0.5$ that the verifier can verify. Moreover, our proposed efficient search method based on Eq. 4 is 307x faster than the brute force search.

5 Conclusion

In this paper, we first properly define the perturbation set $\mathcal{B}(s)$ and the success criteria with the help of an examiner, which become even more reliable after giving it some hints. The adversarial examples we find are understandable to a much weaker verifier like normal human player. We also proposed a new efficient search algorithm by reducing the usage of the examiner. Our experiments found that even the strongest AZ agent with small amount of simulations is vulnerable to our adversarial attack. We hope our work can raise the attention that even for AI agents that surpass humans with a large margin, they still can make simple mistakes that humans will not. A limitation of this paper is that we are only able to identify adversarial states but haven't been able to systematically guide the AZ agents to those states [12]. However, we provide an efficient tool for finding those adversarial examples which are "blind spots" of strong AZ agents, and the tool can be used for debugging and potentially improve the performance of those agents, which may be more beneficial in practice.

References

[1] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

[2] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[3] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[4] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[5] 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.

[6] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[7] Takeru Miyato, Andrew M Dai, and Ian Goodfellow. Adversarial training methods for semi-supervised text classification. *arXiv preprint arXiv:1605.07725*, 2016.

[8] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Semantically equivalent adversarial rules for debugging nlp models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 856–865, 2018.

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

[10] 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. *arXiv preprint arXiv:1703.06748*, 2017.

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

[12] Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning. *arXiv preprint arXiv:1905.10615*, 2019.

[13] David J Wu. Accelerating self-play learning in go. *arXiv preprint arXiv:1902.10565*, 2019.

[14] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[15] Martin Müller. Nogo history and competitions. `https://webdocs.cs.ualberta.ca/~mmueller/nogo/history.html` (visited on 2021/03/14), 2015.

[16] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, pages 157–163. Elsevier, 1994.

[17] Christopher D Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.

[18] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*, 2017.

[19] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Xiaodong Song. Natural adversarial examples. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15257–15266, 2021.

[20] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

[21] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[22] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: elastic-net attacks to deep neural networks via adversarial examples. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[23] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *International conference on machine learning*, pages 2206–2216. PMLR, 2020.

[24] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. *arXiv preprint arXiv:1804.07998*, 2018.

[25] Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 1085–1097, 2019.

[26] Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 50–56. IEEE, 2018.

[27] Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. Bert-attack: Adversarial attack against bert using bert. *arXiv preprint arXiv:2004.09984*, 2020.

[28] Qi Lei, Lingfei Wu, Pin-Yu Chen, Alexandros G Dimakis, Inderjit S Dhillon, and Michael Witbrock. Discrete adversarial attacks and submodular optimization with applications to text classification. *arXiv preprint arXiv:1812.00151*, 2018.

[29] Puyudi Yang, Jianbo Chen, Cho-Jui Hsieh, Jane-Ling Wang, and Michael I Jordan. Greedy attack and gumbel attack: Generating adversarial examples for discrete data. *J. Mach. Learn. Res.*, 21(43):1–36, 2020.

[30] Chen Tessler, Yonathan Efroni, and Shie Mannor. Action robust reinforcement learning and applications in continuous control. In *International Conference on Machine Learning*, pages 6215–6224. PMLR, 2019.

[31] Chaowei Xiao, Xinlei Pan, Warren He, Jian Peng, Mingjie Sun, Jinfeng Yi, Mingyan Liu, Bo Li, and Dawn Song. Characterizing attacks on deep reinforcement learning. *arXiv preprint arXiv:1907.09470*, 2019.

[32] Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Mingyan Liu, Duane Boning, and Cho-Jui Hsieh. Robust deep reinforcement learning against adversarial perturbations on state observations. *arXiv preprint arXiv:2003.08938*, 2020.

[33] Amin Rakhsha, Goran Radanovic, Rati Devidze, Xiaojin Zhu, and Adish Singla. Policy teaching in reinforcement learning via environment poisoning attacks. *Journal of Machine Learning Research*, 22(210):1–45, 2021.

[34] G.-C. Pascutto. Leela-zero. `https://github.com/leela-zero/leela-zero`, 2017.

[35] Yuandong Tian, Jerry Ma, Qucheng Gong, Shubho Sengupta, Zhuoyuan Chen, James Pinkerton, and Larry Zitnick. Elf opengo: An analysis and open reimplementation of alphazero. In *International Conference on Machine Learning*, pages 6244–6253. PMLR, 2019.

[36] Ti-Rong Wu, Ting-Han Wei, and I-Chen Wu. Accelerating and improving alphazero using population based training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1046–1053, 2020.

[37] Li-Cheng Lan, Meng-Yu Tsai, Ti-Rong Wu, I Wu, Cho-Jui Hsieh, et al. Learning to stop: Dynamic simulation monte-carlo tree search. 35:259–267, 2021. URL `https://ojs.aaai.org/index.php/AAAI/article/view/16100`.

[38] LC Lan. Hahanogo: An open source nogo program. `https://github.com/lclan1024/HaHaNoGo` (visited on 2021/03/14), 2016.

Reproducibility & Ethics Statements

Reproducibility We have provided all the parameters of our method in the experiment settings. The agents we used are download from GitHub and we just use the default settings of those agents. We also provide our codes in the supplemental material. We will later release it on GitHub. Our code can be used on different games and agents as long as their actions are countable and the agent supported GTP protocol. Our code communicate with each agent with local port, hence each agent can run on different docker. We also provide dockers that can run those agents.

Ethics Our work is currently only use on games like Go and NoGo, therefore, there is no ethical problem at this stage. However, if someone can lead some real world agent into states that found by our method, then it might be dangerous.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] In the Conclusion section
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] In Appendix A
   (b) Did you include complete proofs of all theoretical results? [Yes] In Appendix A
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] As supplemental meterial and will opensource later
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] In Experiment settings
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] No random seeds are used.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In Experiment settings
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] In Experiment settings
   (b) Did you mention the license of the assets? [Yes] In Experiment settings
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] In Experiment settings
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] The agents we used are opensource.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] Yes
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Proof of Meaningful Action

In this section, our goal is to prove that a meaningful action b_0 of the state s_t will also be the meaningful action of the state s_{t-1} under following two assumptions.

- a_1 Changing the order of the trajectory $a_0, a_1, \ldots, a_{t-1}$ of state s_t will not change the state s_t.
- a_3 The action a_{t-1} is the best action of both s_{t-1} and s'_{t-1}
- a_2 Adding an extra action to the board will not reduce the winrate of the action's color.

Both assumptions are true in the most states on Go and NoGo. For convenience, we assume the turn color is of s_t is black and b_0 is one of its meaningful action. We also define $V_B(s)$ as the state value of black for state s. Now our goal is to prove that b_0 is still meaningful action to s_{t-1}.

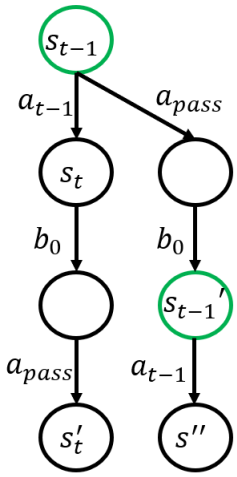


Figure 5: Proof illustration

Since adding an extra action will only benefit to the action's player and the turn player of b_0 is black. According to Fig. 5 We just need to prove that

$$\text{Given } V_B(s_t) < V_B(s'_t), \text{ prove that } V_B(s_{t-1}) < V_B(s'_{t-1}). \tag{5}$$

First since a_{t-1} is the best action of s_{t-1}, we have $V_B(s_t) = V_B(s_{t-1})$. Also, since a_{t-1} is the best action of s'_{t-1}, we have $V_B(s'_{t-1}) = V_B(s'')$.

Next, due to the a_1 assumption, we have $s'_t = s''$.

Finally, we have $V_B(s'_{t-1}) = V_B(s'') = V_B(s'_t) > V_B(s_t) = V_B(s_{t-1})$

B Algorithm of the 2STEP Policy Attack

The algorithm is shown in algorithm 2.

Algorithm 2 Two-Step Policy Attack

Input: a seq states s_i, target agent t, examiner e
for $i = T$ **to** 0 **do**
  $a_s = \arg\max_a(t.p(a|s))$
  if $|e.Q(s_i, a_s) - e.V(s_i)| > \eta_{\text{correct}}$ **or** $e.V(s_i) < \eta_{\text{adv}}$ **then**
    continue
  $actions$ = `getMeaninglessActions` (e, s_i)
  for b_0 in $actions[0]$ **do**
    for b_1 in $actions[1]$ **do**
      $s' = \mathcal{T}(\mathcal{T}(s_i, b_0), b_1)$
      $a_{s'} = \arg\max_a(t.p(a|s'))$
      if $|e.Q(s_i, a_{s'}) - e.V(s_i)| \geq \eta_{\text{adv}}$ **then**
        if $|e.Q(s', a_{s'}) - e.V(s')| \geq \eta_{\text{adv}}$ **and**
          $|e.V(s) - e.V(s')| \leq \eta_{\text{eq}}$ **and**
          $|e.Q(s', a_s) - e.V(s')| \leq \eta_{\text{correct}}$ **then**
          return s'
return NULL

C Algorithm of Getting Meaningless Action

The algorithm is shown in algorithm 3.

Algorithm 3 Get Meaningless Actions

Member variable a set of actions that are meaningful $meaningful_actions$
Input: a state s_i, target agent t, examiner e
ret = [[], []]
$terr$ = e.get_territory(s_i)
a_s = e.get_best_action(s_i)
$V_{s'}$ = e.get_value($\mathcal{T}(s_i, a_s)$)
for a **in** $\mathcal{A}(s_i) \bigcup \mathcal{A}(\mathcal{T}(s_i, a_{pass}))$ **do**
  if $-0.8 \leq terr[a.p] \leq 0.8$ **then**
    continue
  if $a.c == s_i.c$ **then**
    $s'' = \mathcal{T}(\mathcal{T}(\mathcal{T}(s, a), a_{pass}), a_s)$
  else
    $s'' = \mathcal{T}(\mathcal{T}(\mathcal{T}(s, a_{pass}), a), a_s)$
  $v_{s''} = e$.get_quick_value(s'')
  if a **in** $meaningful_actions$ **and** isEq($v_{s''}$, $V_{s'}$) **then**
    if isEq(e.get_value(s''), $V_{s'}$) **then**
      $meaningful_actions$.remove(a)
  else if a **not in** $meaningful_actions$ **and not** isEq($v_{s''}$, $V_{s'}$) **then**
    if not isEq(e.get_value(s''), $V_{s'}$) **then**
      $meaningful_actions$.add(a)
  if a **not in** $meaningful_actions$ **then**
    if $a.c == s_i.c$ **then**
      ret[0].add(a)
    else
      ret[1].add(a)
return ret

D The Input of PV-NN

The inputs of PV-NN are not independent. Take AlphaGo Zero as an example, given a state s_t at step t, it will generate 17 feature planes as the input of PV-NN. Each feature plane is a 19×19 binary 2D array that includes the information the latest eight states. For example, there are eight feature planes $\{X_t, X_{t-1}, \ldots, X_{t-7}\}$ indicates the presence of the current player's stones of $s_t, s_{t-1}, \ldots, s_{t-7}$. If position $p = (i, j)$ has current player's stone at time t, then $X_t[i][j] = 1$ else $X_t[i][j] = 0$. Since each player can only place on stone at a time, X_i will mostly be the same. Hence, the legal inputs feature are not independent. Additionally, most AIs have more complex feature planes as input, including the domain knowledge of Go. For example, a common feature is "liberty," which means how many additional enemy stones are needed to capture the position. This kind of input is also not independent to other feature planes.

E Territory

Starting from an empty board, one player places a stone on a vacant part of the board to surround more territory or defend our territory from being "captured" by the opponent's stones. It is critical to select actions that can gain more territory. Normally, once a position belongs to a color, it is hard to change it. Hence, putting a stone in any color's territory is usually wasting a turn.

Since territory is so important in Go, many agents' PV-NNs ([13]) has an additional output to predict the territory. Given a state s, the territory output $terr(s)$ is a vector of scalar. Each element of the vector $-1 \leq terr(s)[i] \leq 1$ shows how a position i is likely to belong to. For example, $terr(s)[i] <= -0.8$ means that position i is likely belongs to the color white and $terr(s)[i] >= 0.8$ means position i is likely belongs to the color black. In our experiment, we define the positions of meaningless actions should be one of the players' territory. That is, we will not consider an action a as meaningless action if its position p is not one of the player's territory $-0.8 \leq terr(s)[i] \leq 0.8$.

F Formal Definition of the Skip Function

Function $F(s, a) : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ will play the action a on state s without changing the turn player by skipping the opponent's turn. Since action's color $a.c$ might not be the turn color of s, in that case, we need to play action pass a_{pass} first. Finally, F is formulated as follow:

$$F = \begin{cases} \mathcal{T}(\mathcal{T}(s, a), a_{\text{pass}}) & s.c = a.c \\ \mathcal{T}(\mathcal{T}(s, a_{\text{pass}}), a) & s.c \neq a.c \end{cases}$$

where $s.c, a.c$ are the color of the state and the action, a_{pass} is the pass action.

G Go Experiments without Territory

In this section, we attack KataGo's PV-NN without using the territory constraint. The results are shown in Table 4 and can be compared with Table 5 which is the normal setting with territory. Without the territory constraint, it is easier for our method to attack the target model. For example, the 2STEP success rate on policy attack become 100% after removing the territory constraint. Another observation is that without the territory constraint, it is more likely to call the examiner. This might because the meaningless actions under territory constraint are more stable. Hence, if $v(s')$ is different form $V(s)$, it is more likely that it is and adversarial example, instead of s' is semantically different form s.

Table 4: Attack KataGo's PV-NN without territory constraint.

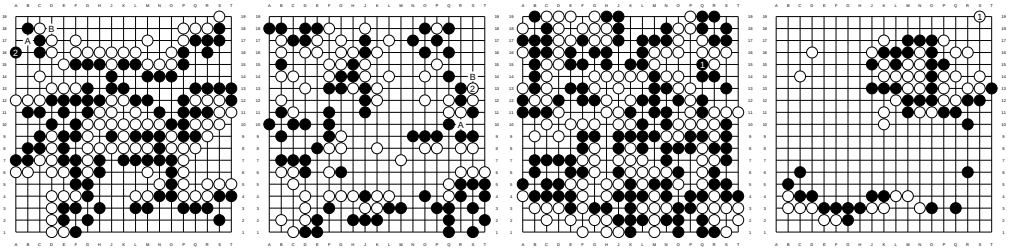
		SUCCESS RATE		EXECUTED NUM	
	GAMES	1STEP	2STEP	TARGET	EXAMINER
VALUE $\eta_{adv} = 0.7$	ZZ	0.89	0.95	61164	797
	ZM	0.90	1.00	57258	2242
	LG	0.95	0.95	60770	835
	ATV	1.00	1.00	5447	83
POLICY $\eta_{adv} = 0.7$	ZZ	0.80	1.00	58895	2716
	ZM	0.95	1.00	16748	560
	LG	0.60	1.00	199201	4832
	ATV	0.75	1.00	258985	5300

Table 5: Attack KataGo's PV-NN with territory constraint.

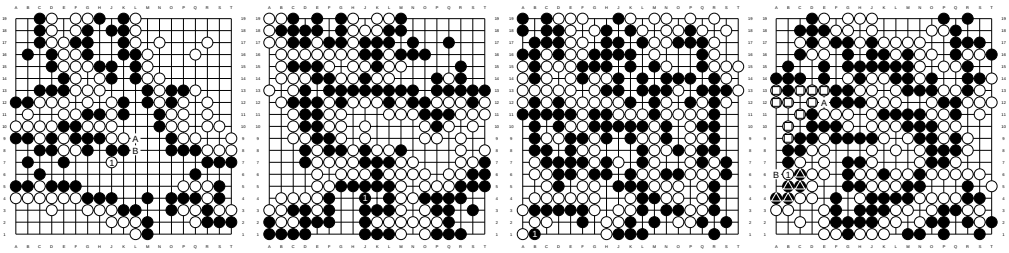
		SUCCESS RATE		EXECUTED NUM	
	GAMES	1STEP	2STEP	TARGET	EXAMINER
VALUE $\eta_{adv} = 0.7$	ZZ	0.84	0.95	24044	476
	ZM	0.85	1.00	24904	347
	LG	0.9	0.95	13405	194
	ATV	0.95	1.00	9215	138
POLICY $\eta_{adv} = 0.7$	ZZ	0.68	0.89	98739	1077
	ZM	0.85	1.00	26028	303
	LG	0.45	0.80	295151	686
	ATV	0.5	0.85	189696	870

H Visualized Examples

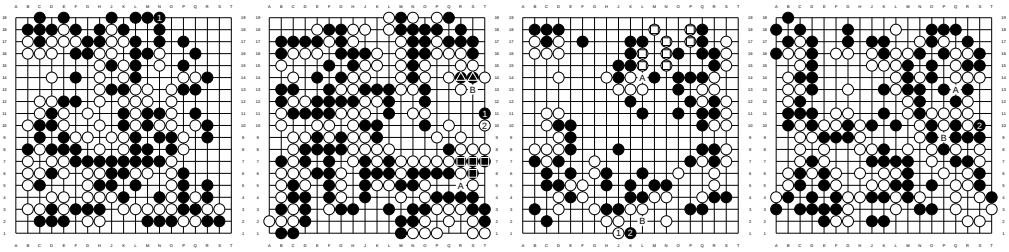
Fig. 6 provides some examples we found on different agents and different types of games. The perturbation actions are marked as 1 and 2. If it is a policy attack, the best action of both s and s' is mark as A and the bad action that the target model want to play is marked as B. The subcaption of each examples first shows the which dataset it is form. For example, "FOX_12k" means that the game from dataset FOX and was generate from level 12k player. For another example, "ZZ_8" means the number 8 game of AlphaGo Zero self-play. Second, the subcaption describe the target model. If it is just PV-NN, we will present it using the first letter of the agent. For example, 'K' for KataGo, 'C' for CGI. On the other hand, if the target model is MCTS, we will add a 'M' and simulation count after the first letter of the model. For example, "KM25" means KataGo MCTS agent with 25 simulation. Finally, the third component of the subcaption show the type of attack and the threshold η_{adv}. For example, "P0.7" means policy attack and $\eta_{adv} = 0.7$.



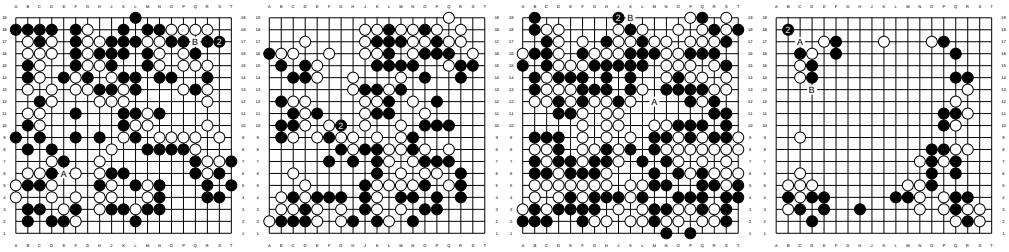
(a) FOX_12k K P0.9 (b) MH_Dang C P0.7 (c) ZM_2 E V0.7 (d) FOX_15k K V0.9



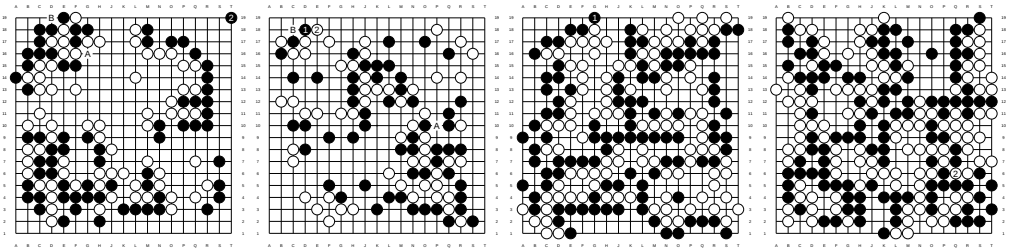
(e) FOX_17K K P0.9 (f) FOX_18k K V0.9 (g) ZM_3 E V0.7 (h) ATV_2 K P0.9



(i) ATV_3 K V0.9 (j) ZZ_8 KM25 P0.7 (k) LG_12 K P0.7 (l) ZZ_10 LM25 P0.7



(m) LG_15 K P0.7 (n) LG_16 K V0.7 (o) ZZ_17 LM50 P0.7 (p) MH_KE E P0.7



(q) MH_LIU C P0.7 (r) LG_3 K P0.9 (s) MH_MEN P V0.7 (t) MH_CHEN E V0.7

Figure 6: In each subfigure, the subcaption provides the name of dataset, the program to be attacked, and the policy attack (P) or value attack (V) concatenating with η_{adv}, separating by space. For example, in (a), "FOX_12k K P0.9" represents a policy attack with $\eta_{\text{adv}} = 0.9$ on KataGo, where the game record is chosen from FOX dataset, and in (j) "ZZ_8 KM25 P0.7" means the attacking policy with $\eta_{\text{adv}} = 0.7$ **K**ataGo **M**CTS with **25** simulation on **ZZ** dataset.