Importance Judger: Detecting Observation-Based Attacks in Deep Reinforcement Learning

***Abstract*—Deep reinforcement learning (DRL) has achieved great success in various security-critical applications such as robotics, drones, medical analysis, and autonomous vehicles. However, recent studies show that machine learning models are vulnerable to adversarial examples, which are intentionally crafted instances that aim to cheat the model into making incorrect decisions. DRL can be effectively attacked by adding perturbations to observations. To defend against these attacks, we propose a detector algorithm based on the judgment of the importance of an observation: that is, the extent of the effect of changing the action decision made at this observation to the final reward. We observe that when DRL is under attack, a large proportion of important observations in DRL are judged as unimportant. The changes of the observation importance will lead to fatal action decision changes. Therefore, we design our algorithm Importance Judger to detect attacks based on the changes of importance for these observations. We evaluate our detector under attack FGSM, BIM and CW$_2$ in three OpenAI gym Atari game environments: Pong, Breakout and Seaquest. The detection accuracy and F_1 score on important observations under different attack scenario can be greater than 90%.**

1. Introduction

In recent years, reinforcement algorithms that incorporate deep learning have achieved great success in beating world champions at the game of Go as well as human experts playing numerous Atari video games. With rapid progress being made in this field, deep reinforcement learning algorithms are expected to become highly valuable in more ambiguous, real-life environments. These applications include many safety-critical fields, such as autonomous navigation [1], drones [2], medical analysis [3] and robotics control [4]. However, recent research has shown that deep neural networks are vulnerable to adversarial perturbations [5]–[7]. Deep reinforcement learning systems that incorporate deep neural network models to perform policy making also suffer from similar vulnerabilities. Several works have proposed effective adversarial attacks against reinforcement learning models [8], [9], [21], making the issue a serious safety concern.

We focus on reinforcement learning environment where a state is an observation from the environment. While the agent is interacting with the environment, thousands or even millions of observations will be accumulated. However, only the action decisions made at a small portion of these observations can cause immediate impact on the final reward. Take the Atari game Pong as an example. The agent moves an in-game paddle vertically across the left side of the screen to hit a ball back and forth with another player controlling a paddle on the opposing side. Note that the policy's action decision will only trigger when the ball is approaching the paddle. Here, the agent either returns the ball or loses the game. An unexpected action decision change at such observations could lead to a sharp decline in the final reward. Consequently, a successful attacker will want to generate adversarial examples of those "important" observations to alter the action distribution learned by the agent. Meanwhile, when the ball is far away from the paddle, the well-trained agent should not show strong preference to any action. But an attacker can still generate long-term attacks to slowly lure the agent into a bad state such as leading the paddle far away so there is no way it can reach the incoming ball. Under such circumstances, an action choice is strongly preferred even when the ball is not near the paddle. Based on these characteristics of adversarial attack against DRL, we propose a differential analysis method to detect fatal action decision changes caused by the adversarial examples, and also sense the abnormality of the adversarial environment.

In this paper, we first propose a simple criteria to judge the importance of an observation, or the effectiveness of changing the action decision made at this state in influencing the final reward. The judgment is based on the variance of the action probability vector. We conduct many experiments on well-trained policies to play Atari games to show that the policy can be as negatively affected by adversarial perturbation added only to important observations as a full-time attack. On the other hand, an attack to a majority of the unimportant observations is ineffective.

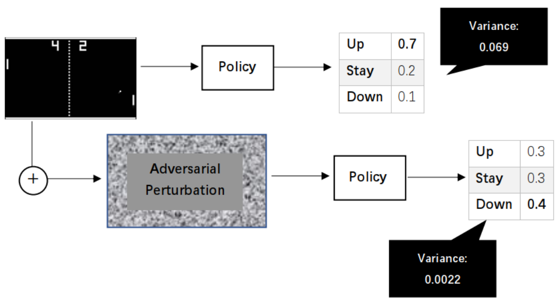

Figure 1. **Variance decrease of important observations** An important observation in Atari-game Pong is given. For a well-trained agent, the probability of choosing the up movement should be high. However, due to the attack, the choice is changed to moving the paddle down, which leads to the failure of hitting the ball back. During this process, the variance of the action probability vector is greatly decreased.

We further find that due to an adversarial attack, the variance of the action probability vector of important observations will largely decrease. As a result, these observations will be judged as unimportant. Figure 1 serves as a vivid illustration of such a situation. Based on this differential analysis method, we can achieve high accuracy in detecting action decision change at important observations.

Our detector mainly consists of a pre-trained CNN classifier to judge whether an observation is important or not. The prediction result given by the CNN model and the calculated result of the policy output will be compared to detect potential attack. The detector focuses on detecting attacks against important observations since attacks on unimportant observations are ineffective. We evaluate our detector under different attacks (FGSM,BIM and CW$_2$) in three OpenAI gym Atari game environments: Pong, Breakout and Seaquest. The detection accuracy and F_1 score on important observations can be as high as 90%.

To summarize, our contributions are the following:

- We propose a simple criteria to judge the importance of a state. Then we propose a detector algorithm that suits a wide range of deep reinforcement learning models without collecting adversarial examples in advance, and is attack model-agnostic.
- We use a detection algorithm to analyze the difference between normal and adversarially perturbed observations based on the change in the importance. Then, we run experiments to determine the effectiveness of the model using three adversarial attack types and three Atari game environments.
- We evaluate the performance of the detection algorithm and discuss the implications of the experimental results.

The rest of this paper is organized as follows. In section 2 we provide the necessary background. Then in section 3, we examine past work on detecting adversarial attack. Section 4 broadly notes our design ideas and related assumptions, while section 5 mathematically defines what we mean by the importance of a state. Then section 6 introduces our intentions for the detector, section 7 reviews the ways we examine its effectiveness, and section 8 features all experimental results. Finally we reflect on our methodology and what can be done as future work.

2. Background

In this section, we will provide a brief introduction to reinforcement learning and adversarial examples.

2.1. Reinforcement Learning

Reinforcement learning is concerned with how agents ought to take actions to interact with an environment in order to maximize the cumulative reward. It differs from supervised learning in that there are no labelled input and output pairs. The environment is formulated as a Markov Decision Process (MDP). An MDP is a 5-tuple (S,A,P,R,γ), where S is a set of states, A is a set of actions, P($s_{t+1}|s_t, a_t$) is the transition probability from state at the t_{th} time-step $s_t \in S$ to $s_{t+1} \in S$ after performing the action $a_t \in A$, R($r_{t+1}|s_t, a_t$) is the probability of receiving reward r_{t+1} after performing action a_t at state s_t, and γ ∈[0,1] is a rate of discounted future reward. The goal of reinforcement learning algorithms is to train a policy π which maps state space S to action space A in order to optimize the cumulative reward received over time.

Deep reinforcement learning(DRL) algorithms are reinforcement learning algorithms that implement deep learning architectures. A deep neural network(DNN) is an artificial neural network with multiple layers between the input and output layer [25].

DRL algorithms can be policy based, such as trust region policy optimization(TRPO) [26] or value based, such as deep Q-networks(DQN) [27] or both, such as asynchronous advantage actor-critic(A3C) [28]. In our work, we implement the double Q-learning(DDQN) [29] to train the policy.

2.2. Generating Adversarial Examples

An adversarial example is an input crafted by an adversary with the goal of producing an incorrect output from a target model. The issue was first identified in the task of classification applying supervised machine learning algorithms [30]. Research in adversarial examples typically defines an adversarial example as a misclassified sample **x'** generated by perturbing a correctly-classified sample **x** by some limited amount.

Adversarial examples can be targeted or untargeted. In a targeted case, the adversary aims at $\mathbf{x}'$ to be classified as a particular class t. Using mathematical expressions, given $\mathbf{x} \in X$ and g(·), a targeted adversary with target t∈Y aims at finding an $\mathbf{x}' \in X$ such that

$$g(\mathbf{x}') = t \wedge \Delta(\mathbf{x}, \mathbf{x}') \leq \epsilon$$

where $\Delta(\mathbf{x}, \mathbf{x'})$ is the difference between input **x** and **x'**, ϵ is the strength of the adversary that models how close an adversarial example needs to be to the original.

In an untargeted case, the adversary aims at $\mathbf{x}'$ to be classified as any class other than its correct class. Using mathematical expressions, given $\mathbf{x} \in X$ and g(·),an untargeted adversary seeks to find an $\mathbf{x}' \in X$ such that

$$g(\mathbf{x}') \neq g(\mathbf{x})\Delta(\mathbf{x}, \mathbf{x}') \leq \epsilon$$

The choice of $\Delta(\cdot)$ is typically an L_p-norm distance metric. We will introduce three norms commonly used for $\Delta(\cdot)$ as follows:

1) L_∞:$||z||_\infty = \max_i |z_i|$.
The L_∞ norm measures the maximum change in any dimension. L_∞ attack is limited by the maximum change it can make to each pixel of the input picture.
2) $L_2 : ||z||_2 = \sqrt{\sum_i z_i}$
The L_2 norm is the Euclidean distance between $\mathbf{x}$ and $\mathbf{x}'$.
3) $L_0 : ||z||_0 = \#\{i|z_i \neq 0\}$.
For images, L_0 calculates the total number of pixels that are altered between $\mathbf{x}$ and $\mathbf{x}'$.

where z=x-x$'$.

We then introduce the three adversarial attack algorithms used in our experiments.

1) *Fast Gradient Sign Method:FGSM (L_{infty}, Untargeted)*
Due to DNNs' linear nature, Goodfellow et al. hypothesized that they are vulnerable to adversarial perturbations [30]. They proposed an efficient algorithm for finding adversarial examples, the *fast gradient sign method*(FGSM). FGSM measures the perturbation $\Delta(\mathbf{x}, \mathbf{x}')$ using the L_{infty}-norm. The strength of perturbation at every dimension is limited by the same constant parameter, ϵ.
As an untargeted attack, the perturbation is calculated using gradient vector of a loss function:

$$\Delta(\mathbf{x}, \mathbf{x}') = \epsilon \cdot \text{sign}(\nabla_x J(g(\mathbf{x}), y))$$

where the loss function, J$(\cdot,\cdot)$ is the loss used to train the specific DNN model, and y is the correct answer for $\mathbf{x}$.

2) *Basic Iterative Method:BIM(L_{infty}, Untargeted)*
Kurakin et al. extended the FGSM method by applying it multiple times with small step size [31]. After each step, the BIM method clips pixel values of intermediate results to ensure that they are limited by the strength of the adversary,ϵ.
For the n^{th} iteration,

$$\mathbf{x}'_{n+1} = \mathbf{x}'_n + \text{Clip}_{\mathbf{x},\epsilon}(\alpha \cdot \text{sign}(\nabla_x J(g(\mathbf{x}_n), y)))$$

The clipping equation, $\text{Clip}_{\mathbf{x},\epsilon}(\mathbf{z})$, performs per-pixel clipping on $\mathbf{z}$ so that the result will not exceed $L_\infty\epsilon-$neighborhood of $\mathbf{x}$ [31].

3) *Carlini/Wagner Attacks(L_2,L_∞ and L_0,Targeted)*
Carlini and Wagner proposed three gradient-based attack algorithms using L_2, L_∞, L_0 that can achieve excellent adversarial success rates with minimal perturbation amounts [32]. We implement CW_2 in our experiments introduced as follows.
The CW$_2$ attack improves the way to solve the optimization problem with two terms: the prediction objective and the distance term through several techniques. Instead of the softmax-cross-entropy loss that is commonly used in other optimization-based attacks, it uses the logits-based objective function, which makes it robust against the defensive distillation method [33]. Besides, it converts the target variable to the *argtanh* space so that it can take advantage of modern optimization solvers, such as Adam. Also, it implements a binary search algorithm to select coefficients that performs a good trade-off between the prediction and the distance terms.

Based on what knowledge the attacker needs to perform the attack, adversarial attacks can be categorized into *white-box* attacks, where the attacker has full knowledge of the target model and *black-box* attacks, where the attacker has less or no knowledge.

3. Related Work

In this section, we discuss relevant works on adversarial attacks and defensive techniques in deep learning and deep reinforcement learning settings.

3.1. Adversarial Attacks against DRL

As deep RL algorithms train policies parametrized by neural networks, recent works have shown that such policies are vulnerable to adversarial attacks [8], [9], [21]. The adversary aims at changing the optimal action the policy chooses to take, resulting in a decrease in the cumulative reward. Huang et al. [8] had shown several properties of such attack, similar to the supervised learning setting. While policies trained with A3C, TRPO, and DQN are all susceptible to adversarial inputs, policies trained with DQN are more susceptible, especially to l_∞-norm perturbation. Besides, transferability across different policies and training algorithms is effective. Policies are also vulnerable to black-box attacks.

Regarding the properties of reinforcement learning, adversarial attacks can also be more efficient and targeted. Lin et al. proposed two tactics of attack [9]. One is the strategically-timed attack that minimizes the number of time steps to attack based on a relative action preference function. Similarly, Kos [10] also shows the effectiveness of attack only on high value states based on the value function. The other is the enchanting attack where the adversary aims at luring the agent to a designated target state.

3.2. General Defensive Techniques

Papernot el al. gave a thorough summary of work on defending against adversarial samples [**?**]. Relevant work mainly focus on adversarial training and gradient masking methods. We discuss them briefly as follows.

Adversarial Training *Adversarial training* injects adversarial examples of known attack methods into the training data set to increase robustness [5], [12]–[14]. The trained model will learn to make correct output when encountering learned adversarial examples. In a deep reinforcement learning setting, Pinto [23] proposes training an RL agent to provide adversarial attack during training so that the agent can be robust against dynamics variations. However, one disadvantage of this adversarial training method is that adversarial examples of all known attack methods must be generated, resulting in an at least double training cost. The generation procedure is also computationally expensive. Besides, for RL tasks, since the method requires accurate modeling of the environment dynamics, it may be hard to be applied in real word tasks such as robotics and self-driving cars [23].

Gradient Masking *Gradient masking* methods forces DNN models to produce near-zero gradients in order to reduce the models' sensitivity to small changes in inputs [15], [33]. However, Papernot et al. concluded that due to the transferability of adversarial examples, such methods will have limited success [11].

Differently, our method focus on detecting adversarial attacks at test time given the policy is well-trained in normal environment. We will further discuss and compare similar works in the following sections.

3.3. Detection Methods for Deep Learning

Several recent works focus on detecting adversarial examples. Xu [22] provided a comprehensive summary of such work, grouping into three broad categories: *sample statistics* [16], [17], *training a detector* [18] and *prediction inconsistency* [19], [22]. Our proposed importance judger belongs to the third group that employs prediction inconsistency. The basic idea of such detection methods is to apply differential analysis of the prediction results given by several models. In the supervised learning setting, one typical successful detection method is feature squeezing [22]. To detect, feature squeezing compares a DNN model's prediction on the original input with that on squeezed inputs.

We implement experiments to see whether the idea of feature squeezing can be used to detect adversarial examples in a reinforcement learning setting as well. We calculate differences in L_1 distance between the action probability vector given by the policy with original and squeezed observation as input. For value-based reinforcement learning algorithms, such as Q learning, the computed Q-values of action is converted into a probability distribution using the softmax function [35]. Three white-box attacks with $\epsilon = 0.004$ are implemented. We try the median filter with two window sizes. The 100-episodes average L_1 distance per observation in Atari game Pong is listed in Table 1 below.

TABLE 1. AVERAGE L_1 DISTANCE BETWEEN ORIGINAL AND SQUEEZED OBSERVATION IN DIFFERENT ENVIRONMENTS

Window Size	Environment	Average L_1 distance
2×2	Normal	0.0110± 0.023
	Attacked(FGSM)	0.0095± 0.011
	Attacked(BIM)	0.0093±0.011
	Attacked(CW)	0.0085± 0.010
3×3	Normal	0.0161± 0.027
	Attacked(FGSM)	0.0107± 0.021
	Attacked(BIM)	0.0110±0.021
	Attacked(CW)	0.0111± 0.022

It can be seen that due to an attack, there is no significant increase in the distance between the original and squeezed sample. The distance in normal environment even surpasses that under attack. The reason may be due to the fact that observations given by the environment are highly-correlated and the policy's task of making action decisions cannot be equivalent to a classifier making classification. Thus, the method of detecting adversarial attack against reinforcement learning should not be designed the same way as supervised learning.

3.4. Detection Method for DRL

Lin proposed a defense algorithm designed for reinforcement learning models [24]. By comparing the action distribution produced by a policy from processing the current observed frame to the action distribution produced by the same policy and from processing the predicted frame from the action-conditioned frame prediction module, the presence of adversarial examples can be detected. However, the training of the frame prediction model is a computationally intensive task. Besides that, we find other potential issues in applying the algorithm.

- **Negative Effect of Sequential Attack on the Prediction Model** As is stated in the paper, the accuracy of frame prediction model greatly influences the performance of the detector. However, since the prediction model is very similar to the policy in purpose, the prediction model can be vulnerable to sequential attack in theory. In fact, as the accuracy of the prediction model can be reflected in the cumulative reward with defense, the phenomenon that the reward gradually decreases with higher attack ratio can be seen from the plots listed in the paper, suggesting that the prediction model is negatively influenced by sequential attack and makes the policy give a different action choice than the optimal one. It is hard to guarantee that under such sequential attack, the distance between the predicted and attacked action distribution will still surpass the distance between the predicted and normal action distribution. Thus, the risk of detection failure of such a dangerous attack is high.
- **Low Efficiency** For a reinforcement learning policy, thousands or even millions of observations will be sent from the environment. It is very time-consuming to do differential analysis based on the result returned by the frame prediction model on each of them. To show the caused decrease in the agent's speed, we compare the 100-episodes average time cost of well-trained agent playing the best-defensed Atari game Freeway with and without the visual foresight framework in normal environment.

TABLE 2. DECREASE IN SPEED DUE TO VISUAL FORESIGHT DETECTOR

Atari Game	Freeway
Average Time Cost Per Episode	9.627± 0.24 s
Average Time Cost Per Episode with Visual Foresight Detector	38.82±0.36 s

The programme is run on Ubuntu18 system with GPU. As is shown in the table, for a simple game environment, the operation of visual foresight will cause the agent to interact more than 4 times slower with the environment, making the detector not very applicable.

Our work differs from detectors for deep learning in that it is specifically designed for reinforcement learning setting. The main difference from the supervised learning model is that reinforcement learning is about sequential decision making and the next input depends on the current decision. The best policy is not labeled, so it is difficult to judge right or wrong on the decision made at each state. As a result, the detector cannot be designed the same way as a supervised learning model.

Our work advantages over the state-of-the-art detector for attacks against reinforcement learning models in two aspects. First of all, we take the sequential characteristic of the attacks into account and focus on sensing the abnormality of the adversarial environment. Our differential analysis is more targeted. We can achieve high accuracy in detecting fatal change in action decision. The detector is also less likely to be negatively influenced by the attack. Secondly, our algorithm is much less computationally intensive, which reduces the negative side effect of the agent's speed decrease.

4. Threat Model and Design Overview

In this section, we will describe the threat model and summarize our design of the detector.

Threat model. We assume a powerful adversary who has full access to a trained target model. The adversary implements white-box attack techniques to generate adversarial examples.

Target Models We apply our approach to a range of Atari 2600 games implemented in OpenAI Gym [37]. Atari 2600 is a challenging RL testbed that presents agents with a high dimensional visual input and a diverse and interesting set of tasks that were designed to be difficult for humans players. We choose three typical games, Pong, Breakout and Seaquest. We implement the DDQN algorithm [29] to train the policy as it is mentioned in [8], policies trained with DQN are more susceptible to adversarial attacks compared with A3C and TRPO. All three policies can play the game well. Their performance is shown in the "Reward in Normal Environment" column in Table 6.

Design Overview We propose a detection algorithm that aims at detecting adversarial examples at test time. The detection idea is based on the differential analysis of the change in importance of an observation due to attack.

We aim at designing a detector that has a simple architecture and focuses on a small portion of observations to enhance analysis efficiency. We use a CNN model as an image classifier. Then we implement a differential analysis between the predicted result from the CNN and the calculated result from the policy. Our attack algorithm will be introduced in section 6.

5. Analysis of Importance Difference Due to Attack

In this section, we first define the importance of a state. Then, the results of a large number of experiments will show the phenomenon that most of the observations judged as important in a normal environment will fail to maintain the variance level after being adversarially perturbed. Such change can be used to detect the abnormality and prevent the policy from taking dangerous actions.

5.1. Judgment of the Importance of a State

We define the importance of a state as the extent of the effect of changing the action decision made at that state to the final reward. At an important state, the action choice made will largely influence the final cumulative reward. Thus, intuitively, for a well-trained agent, the probability of choosing the best action at an important state should be much larger than choosing other actions. On the other hand, the decision of which action to take does not matter much at unimportant states, resulting in the similarity among the learned probability of choosing any action. Therefore, we expect that the abnormality of an attacked environment mainly results from the perturbation applied to important observations. In a paper on strategically-timed attack on deep reinforcement learning policies [9], the phenomenon that the expected accumulated reward can be minimized with much fewer adversarial examples compared with full-time attack has also been shown.

Here is how we compute the importance of an observation. For an action probability vector, the deviation of the probabilities from their mean can be mathematically measured as the variance of the action probability vector.

For policy based reinforcement learning algorithms, such as TRPO, the variance of the action probability vector of s_t can be directly calculated as

$$Var(s_t) = \frac{1}{n}\sum_{i=1}^{n}(\pi(s_t, a_i) - \mu)$$

where n is the number of available actions in set A and μ is the average value of the probabilities.

For value based reinforcement learning algorithms, such as DQN, the algorithm implemented in our work, the computed Q-values of actions can be converted into a probability distribution using the softmax function [35]

$$Prob(s_t, a_i) = \frac{e^{Q(s_t,a_i)}}{\sum_{j=1}^{n} e^{Q(s_t,a_j)}}$$

where $Prob(s_t, a_i)$ indicates the probability of choosing action a_i at state s_t, and Q(s_t, a_i) is the computed Q-value of state-action pair (s_t, a_i).

We choose to compute the importance of an observation using the variance of action probability vector as variance best indicates how far action choice probabilities are spread out from their average value. In [9], Lin proposes a relative action preference function c that uses the difference between the largest and the smallest action probability to solve the when-to-attack problem and enhance the attack efficiency. However, as the range of possible action probabilities that can be reached is unknown and may vary in different environments, the threshold cannot be generally set and must be determined based on observation.

5.2. Change of the State Importance Due to Attack

In this section, we will show the change in importance of states due to attack based on large number of experimental results.

5.2.1. Change in Important States. As adding adversarial perturbation to important observations are effective and fatal, a successful attacker is very likely to perform well at changing the probability distribution of the actions at those states to alter the choice. During this process, we observe that the variance of the action probability vector will greatly decrease and, most of the time, fails to exceed the threshold. Since it is not necessary and also difficult for the attacker to alter the action choice and maintain the variance level at the same time, this change can be used to detect adversarial attacks.

Quantitative Evaluation We conduct experiments with different variance thresholds. The agent plays the game in a normal environment so that various states can be experienced. The observations with variance of the action probability vector exceeding the threshold are considered as important here. Among all important observations, those whose variance fails to exceed the threshold after being adversarially perturbed are counted as changed. The changed percentage is calculated per episode as

$$\text{Changed Percentage} = \frac{\text{num of changed important observations}}{\text{num of important observations}}$$

where changed important observations are those whose variance of the action probability vector fails to exceed the threshold after being adversarially perturbed.

The plots of 100-episodes average changed percentage vs. variance threshold under different attacks in three game environments are listed in Table 3.

5.2.2. Change in Unimportant States. On the other hand, for the unimportant observations, we observe that the variance of the action probability vector of a perceptible percentage among all will exceed the threshold due to attack. This phenomenon may be due to the fact that as the attack on unimportant observations is not very efficient, the attacker will generate long-term attack to lure the policy into bad states [9]. Under this circumstance, a certain action choice is expected in contrast to whichever action taken making not much difference to the final reward in normal environment. The changed percentage per episode is calculated as

$$\text{Changed Percentage} = \frac{\text{num of changed unimportant observations}}{\text{num of unimportant observations}}$$

where changed unimportant observations are those whose variance of the action probability vector exceeds the threshold after being adversarially perturbed.

The plots of 100-episodes average changed percentage vs. variance threshold under different attacks in three game environments are listed in Table 4.

6. Proposed Detection Algorithm

Based on the phenomenon shown in the previous section, we propose a detection algorithm, *Importance Judger*. In this section, we will first introduce design thinking of the algorithm. Then, we will introduce the detector architecture. We will also analyze the robustness of the algorithm.

6.1. Design Thinking

Unlike image classification task in the supervised learning setting, it is not practical to apply comprehensive analysis to each observation image. This is because as the trained policy is interacting with the environment, thousands or even millions of observations will be processed. Therefore, our initial goal is to design a detector with a simple architecture operating together with the policy, and to have a focus instead of treating all incoming states equally.

Based on what we have discussed in the previous sections, important observations should be paid much more attention to than others. Also a convolutional neural network works well for analyzing images. To minimize the negative effect of the detector on the policy speed, the architecture should be as simplified as it can be. Towards this goal, we choose to downgrade the observation image dimension to one and reduce its size before it is put into the CNN model.

Regarding the transferability of adversarial attack on neural networks, the precision goal of the CNN model should be totally different from that of the trained policy. With our focus on important observations, we set the prediction goal of the CNN model to be a binary classification task: whether the incoming observation is important or not.

In section 5.2, we have shown the phenomenon that a large proportion of important observations will fail to maintain high variance in the action probability vector they receive from the policy due to attack. For an observation judged as important by the CNN model, we will thus calculate this variance to detect potential attack.

6.2. Detector Architecture

The detector architecture, reflecting our design thinking, is shown in Figure 2. The main component is a pre-trained CNN classifier to judge the importance of an observation.

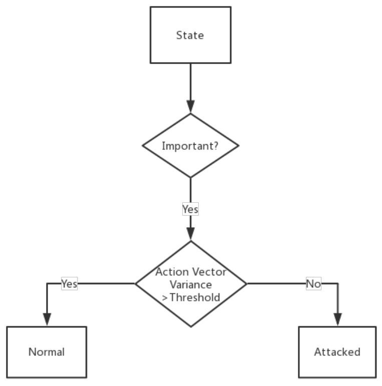

Figure 2. Detection algorithm for attacks on important states

6.3. Robustness

Attacking algorithms require calculating $\Delta_x J(\theta, x, y)$, the gradient of the cost function J(θ,x,y) with respect to the input x. In reinforcement learning settings, J(θ,x,y) is the cost function between y and the distribution that places all weight on the highest-weighted action in y, provided that the policy is well-trained.

If the adversary has no information of our detector, the gradient of the cost function to be calculated is targeting at the policy, which is different from that of the CNN model. The CNN model is thus out of attack range. Therefore, we expect that the generated perturbation should have minimal negative effect on the accuracy of classification. Experimental results on this phenomenon will be shown in section 8.3.

If the adversary has access to the information of the detector, as far as we know, there is no current attacking algorithm that can solve two gradient descent problems at the same time while constraining the attack strength

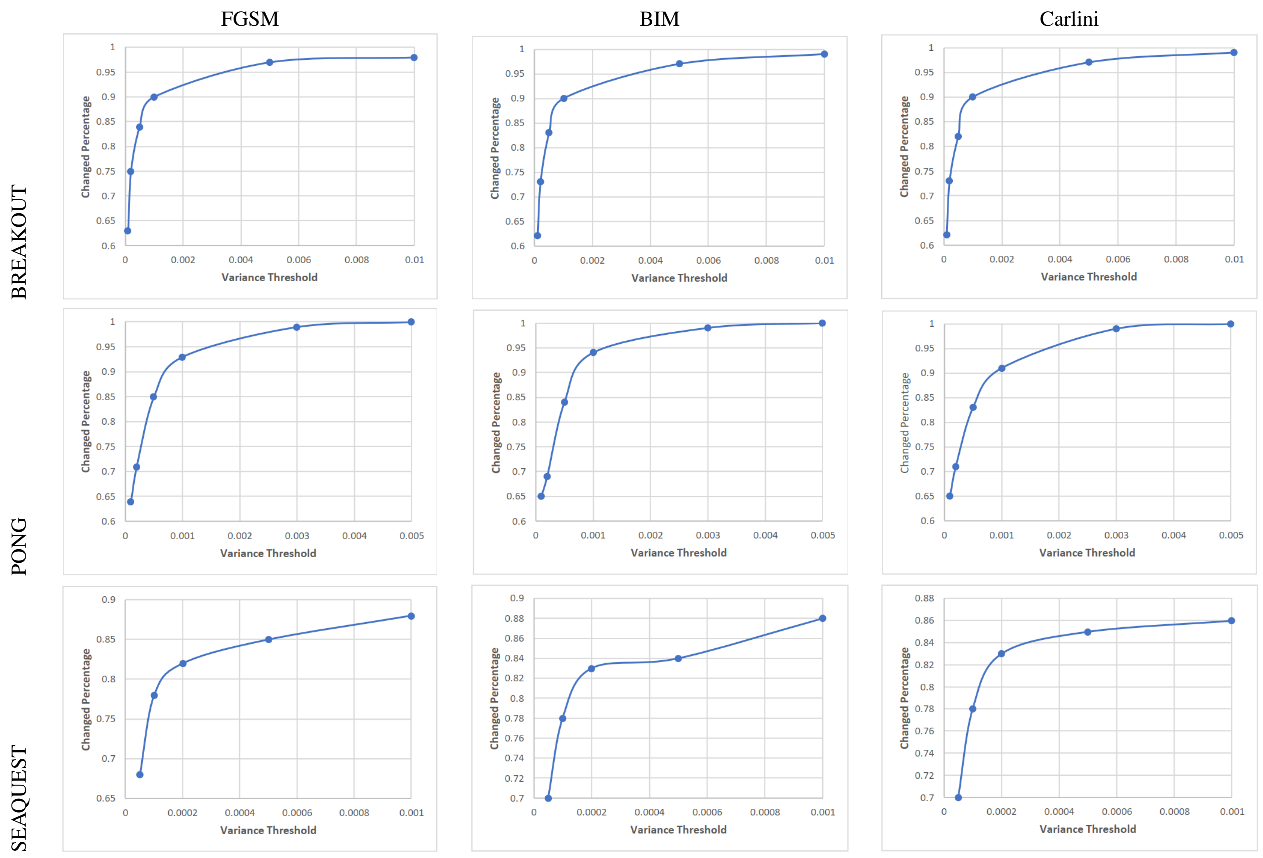

TABLE 3. Changed Percentage Vs. Variance Threshold on Important Obervations First column: Fast Gradient Sign Method(FGSM) [30], Second column: Basic Iterative Method(BIM) [31], and Third column: Carlini et al. [32].The plots show the change of variance among most of important observations due to attack in three Atari game environments. With higher variance threshold, the proportion of important observations among all observations will decrease while the changed percentage among all important observations will increase.

ϵ. Our detection method is thus quite robust against such adaptive attack.

7. Evaluation of the Detector Performance

In this section, we will introduce how we evaluate the detector performance.

7.1. Research Questions for the Evaluation

For the importance judger to be effective in detecting adversarial examples for deep reinforcement learning models, three properties must be satisfied: (1) attack on unimportant observations is ineffective and could not change the reward much; (2) the judger can accurately tell whether an observation is important or not; and (3) after an observation is judged as important, the calculated variance of the action probability vector given by the model fails to exceed the threshold if attacked. The evaluation will answer the questions whether these three properties hold.

7.2. Attack Efficiency

We evaluate three white-box attacks(ϵ=0.004) against well-trained policies playing various Atari-games by 100-episodes average reward and success rate. The success rate is calculated under full-time attack as

$$\text{Success Rate} = \frac{\text{num of obs with changed action decision}}{\text{num of obs}}$$

We further compare random attack performance on all observations, important observations only and unimportant observations only to show that attack on unimportant is rather inefficient.

7.3. Image Classifier Performance

We train an image classifier as the judger of the importance of the observations. It is a supervised learning task to classify observations into two classes: important or unimportant. The training procedure will be described below.

Image Preprocessing As color does not give much information on the importance of the action decision at a certain state, we first convert the observation image to grayscale. To mitigate the possible negative influence of adversarial perturbation on the classifier, we use the median smoothing squeezer with window size 2×2 to further preprocess the image based on its success to defend against adversarial attack in supervised learning setting.

CNN Architecture A 2D convolutional neural network(CNN) is implemented to do the classification task. The architecture we use is shown below. The stride here is set to one.

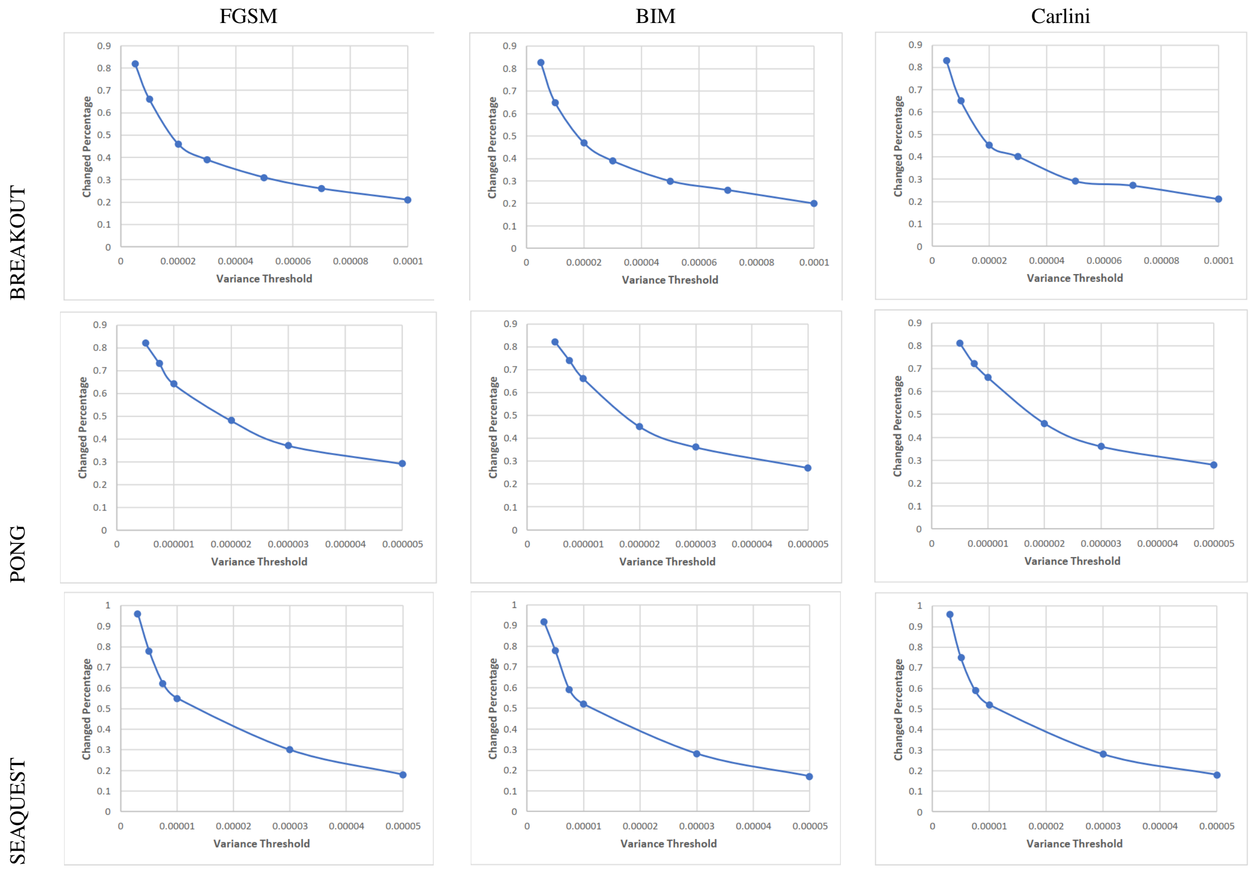

TABLE 4. **CHANGED PERCENTAGE VS. VARIANCE THRESHOLD ON UNIMPORTANT OBSERVATIONS** FIRST COLUMN: FAST GRADIENT SIGN METHOD(FGSM) [30], SECOND COLUMN: BASIC ITERATIVE METHOD(BIM) [31], AND THIRD COLUMN: CARLINI ET AL. [32]. THE PLOTS SHOW THE CHANGE OF VARIANCE AMONG A PERCEPTIBLE PERCENTAGE OF UNIMPORTANT OBSERVATIONS DUE TO ATTACK IN FOUR ATARI GAME ENVIRONMENTS. WITH LOWER VARIANCE THRESHOLD, THE PROPORTION OF UNIMPORTANT OBSERVATIONS AMONG ALL OBSERVATIONS WILL DECREASE WHILE THE CHANGED PERCENTAGE AMONG ALL UNIMPORTANT OBSERVATIONS WILL INCREASE.

TABLE 5. ARCHITECTURE OF IMAGE CLASSIFIER

Layer		Feature Map	Size	Kernel Size	Activation
Input	Image	1	84×84	-	-
1	Convolution	32	82×82	3×3	ReLU
2	Convolution	64	80×80	3×3	ReLU
3	Max Pooling	64	40×40	2×2	-
4	FC	-	128	-	ReLU
Output	FC	-	2	-	Softmax

We then test the classifier accuracy in judging the importance of an observation in normal and attacked environments. The test accuracy is calculated as.

$$\text{Test Accuracy} = \frac{\text{num of correctly classified observations}}{\text{num of observations tested}}$$

7.4. Detection Accuracy per Observation

Since an attack on important observations is very effective, we will evaluate the detector by calculating the accuracy, precision, recall, and F1 score for each important observation. The method to count the number of true positive(TP), true negative(TN), false positive(FP) and false negative(FN) is designed as Algorithm 1.

8. Experimental Results

In this section, we will present experimental results to evaluate our detector performance. In particular, we will present experimental results to answer the three research questions described in Section 7.1. Firstly, we run experiments with different settings of attacks, including attacks to only important observations, only unimportant observations, and a mixture of both. Our experimental results show the fact that even if the adversary is informed of our detection model and avoids all important observations,the attack on pure unimportant observations will be ineffective. Then, we will present the CNN classifier accuracy to show that adversarial perturbation targeting policy will not cause much negative effect on the pre-trained classifier. At last, we will provide detailed tables and plots of observation-based detection accuracy, precision, recall, and F_1 score under different attack scenario.

8.1. Experiments Setup

We conduct experiments in three OpenAI Gym [37] Atari game environments: Pong, Breakout, and Seaquest and implement three white-box attacks: FGSM [30], BIM [31], and CW2 [32]. We implemented three kinds of attacks: attacks on randomly selected observations, on only important observations, and on only unimportant observations. Attacks are generated with different attack probabilities. The attack probability is set in the following way. With each observation, a random number between 0 to 1 will be generated. If it is smaller than the pre-determined probability, the observation will be adversarially perturbed. The observations whose action probability vector has variance exceeding the threshold in normal environment are considered as important and others are considered as unimportant here.

Algorithm 1 Evaluation Method per Observation

```
 1: for each observation do
 2:     actver = variance of the action probability vector
 3:     var_thres = variance threshold set by the classifier
 4:     predict_class= classifier(obs)
 5:     if ((predict_class == important) then
 6:         if (obs is attacked) then
 7:             if (actver < var_thres) then
 8:                 num_TP + = 1
 9:             else
10:                 num_FN + = 1
11:             end if
12:         else
13:             if (actver < var_thres) then
14:                 num_NP + = 1
15:             else
16:                 num_TN + = 1
17:             end if
18:         end if
19:     end if
20: end for
```

8.2. Attack Evaluation

The videos of policies playing games under different attack scenario are listed at https://drive.google.com/drive/folders/1HpZVDOEjQ6ldFkTdp0ZbGSPc8n66PgAB?usp=sharing, serving as a vivid comparison of attack efficiency. In game environment Pong, it can be seen that under attack only on important observations, the policy tries to move the paddle to catch the ball when the ball is still far from the paddle. However, when the ball is near the paddle, the policy stops the paddle or moves it in the direction to keep away from the ball, resulting in losing the game. On the other hand, under attack only on unimportant observations, the policy still tries to reach the ball when the ball is near the paddle, so it can still play the game well.

Table 6 shows that DRL models are very vulnerable to adversarial attacks. Even random attack with probability 0.2 can largely decrease the cumulative reward. Comparing Table 6 with Table 7, it can be seen that with attack purely on important states, the policy plays the game much worse and loses quickly. The average reward drops significantly and nears the average reward resulting from a full-time attack. On the other hand, with attack purely on unimportant observations of high proportion, the policy can still play the game well earning medium or high-level reward. Even the effectiveness of random attack with probability 0.2 can surpass that of attack on the unimportant. This shows that detecting and preventing attacks on unimportant observations does not have a significant impact on the average reward, and that the variance of the action probability vector can be used to judge the importance of a state.

8.3. Image Classifier Performance

We train the CNN model with 50000 observation images sampled from normal environment and test it with 10000 images in both normal and attacked environments. The test accuracy is listed in Table 8. It can be seen that our CNN classifiers reach high accuracy in three game environments. The performance of the image classifier will not deteriorate with adversarial examples as input. The adversarial examples cannot be transferred among neural networks with total different prediction goal.

8.4. Detection Accuracy on Important Observations

We set a high variance threshold to calculate the detection accuracy per observation. We sample 10000 observations from environment under different attack ratio in various game environments. The detection performance is shown in Table 9. We further evaluate the detector under different attack ratio only on important observations. The detection performance is shown in Table 10.

To obtain all detector results, we chose variance threshold 0.001 for Pong and Breakout and variance threshold 0.0005 for Seaquest. Detection metrics were originally calculated using 0.001 for Seaquest as well, but we found them to be relatively low for attacks on important observations. Our justification for this is that too many important observations in Seaquest fell below this variance threshold after attack, likely because the probability distribution of actions is narrower to begin with. Essentially, for this game most variances naturally occur below 0.001 and through experimentation we found 0.0005 to be a more suitable choice for detection.

Recall that with Tables 9 and 10, we compare detection performance on all observations versus those on just important observations. As we know that attacking important observations has a great impact on policy reward, we expect these detection metrics to be similar to each other per game and attack. First, we see that the detector does very well with attack on all observations since accuracy, precision, recall, and F_1 score are almost entirely over 0.9. With attack on important observations, accuracy is mostly contained within the range 0.8-0.9 though Seaquest skews a bit lower (particularly for attack probability 0.2). We suspect this could be due to this game's lower-performing policy. The Seaquest comparison gets more interesting when we look at the other metrics though, since the F_1 scores are not significantly different.

Tables 11 and 12 provide us a more granular look at detection on important observations. Incrementing from attack probabilities 0.1 and 0.2 to 0.9, we see that detector accuracy is almost always increasing. Table 11 trends are very similar among the different attacks if we condition by game. Comparing the games, however, we see that Breakout and Seaquest plateau around 0.8/0.9 while Pong is still sharply increasing. This analysis is exactly the same with Table 12 when we look at F1 score. The most interesting result for both is Attack FGSM on Breakout because the metric value decreases among the highest attack probabilities.

TABLE 6. 100-EPISODES AVERAGE REWARD IN DIFFERENT ENVIRONMENTS AND SUCCESS RATE OF THE ATTACKS

Atari Game	Reward in Normal Environment	Attack	Success Rate	Reward under Full-Time Attack	Reward under Random Attack with Attack Probability 0.6	Reward under Random Attack with Attack Probability 0.2
Breakout	424.09± 124.4	FGSM	88.64± 2.2%	0.34± 0.5	4.82± 3.6	56.61± 54.3
		BIM	89.01±1.7%	0.39± 0.6	5.13± 3.3	58.29± 55.3
		CW	88.66± 1.9 %	0.34± 0.6	5.64 ± 4.0	65.16± 72.9
Pong	18.95± 1.3	FGSM	94.67± 0.8%	-20.5± 0.6	-17.92± 1.8	6.36± 7.0
		BIM	94.53±0.1%	-20.46± 0.6	-18.3± 1.4	5.46± 5.9
		CW	94.47± 0.8 %	-20.44± 0.7	-17.78 ± 1.7	7.58± 4.5
Seaquest	19486.1± 9174.9	FGSM	95.55± 1.5%	176.6± 191.7	1295.8± 1139	8238.1± 5159.8
		BIM	95.6±1.2%	193.2± 168.7	1346.7± 1162.9	8550± 5332.8
		CW	95.28± 1.5 %	160.6± 179.8	1170 ± 846.3	9088.5± 5557.4

TABLE 7. 100-EPISODES AVERAGE REWARD UNDER ATTACKS ONLY ON IMPORTANT OR UNIMPORTANT OBSERVATIONS

Atari Game	Variance Threshold	Attack	Reward under Attack only on Important Observations	Reward under Attack only on Unimportant Observations
Breakout	0.000001	FGSM	0.68 ± 0.75	415.12 ± 88.38
		BIM	0.6 ± 0.663	425.15 ± 93.469
		CW	0.5 ± 0.686	396.49 ± 108.53
	0.0001	FGSM	2.6 ± 1.4	265.35 ± 143.1
		BIM	2.15 ± 1.4	299.4 ± 131.3
		CW	1.8 ± 0.8	286.79 ± 137.7
Pong	0.000001	FGSM	-19.98 ± 0.8	18.57 ± 1.595
		BIM	-20.09 ± 0.83	18.18 ± 1.69
		CW	-20.09 ± 0.74	18 ± 2.05
	0.00001	FGSM	-20.52 ± 0.6	15.66 ± 2.5
		BIM	-20.34 ± 0.7	14.84 ± 3.3
		CW	-20.5 ± 0.8	14.42 ± 2.8
Seaquest	0.000001	FGSM	156.6 ± 138.3	18790.8 ± 9431.87
		BIM	197.8 ± 198.1	17244.1 ± 7657.02
		CW	162 ± 152.6	17177.8 ± 7900.27
	0.00003	FGSM	1053.1 ± 593.76	2789.5 ± 2984.1
		BIM	971.3 ± 557.4	2878.2 ± 3879.2
		CW	975.8 ± 591.3	2285.0 ± 2073.3

TABLE 8. TEST ACCURACY OF THE CLASSIFIER IN DIFFERENT ENVIRONMENTS

Game	Variance Threshold	Normal	Attacked(FGSM)	Attacked(BIM)	Attacked(CW)
Breakout	0.001	0.912	0.857	0.817	0.872
Pong	0.001	0.988	0.986	0.988	0.987
Seaquest	0.0005	0.905	0.918	0.926	0.927

9. Discussions

In this section, we discuss potential ways to break the defense and future directions for improving detection performance.

Image Classifier As is shown in the previous section, the adversarial perturbation aimed at the policy can not ruin worsen the performance of the classifier. As the goal of the policy and the classifier are very different, to the best of our knowledge, no existing attack method can target two networks at the same time.

The classification performance in Pong is the best among the three Atari games. The reason may be that the Pong environment is the simplest with only two paddles and a ball. The relevant information to the importance of a state can be filtered out at each observation to further enhance the performance of the classifier.

Maintaining the Variance Level The simplest way to avoid being detected for the attacker is to maintain the variance level of each adversarially perturbed observation. However, as the loss function of the policy is different from that of maintaining the variance level, there is no attacking algorithm that can solve the two problems simultaneously while constraining the attack strength. Therefore, our detection algorithm is quite robust against such adaptive attack.

Long-term attack on unimportant observations Assume that the adversary has information of the variance threshold that the CNN classifier has been trained, the adversary might choose to generate long-term attack on unimportant observations to lure the policy to bad states. This is a potential concern of our detection algorithm. Based on the variance change phenomenon in unimportant states due to attack presented in section 5.2.2, we have designed a similar detection method focusing on unimportant observations to make our detector more robust. With a CNN classifier pre-trained with a rather low variance threshold, the predicted classification result can be compared with the calculated variance of the action probability vector. The detector architecture is shown in Figure 3.

TABLE 9. Detection Performance under Attack on All Observations

Game	Variance Threshold	Attack	Attack Probability	Accuracy	Precision	Recall	F1 Score
Breakout	0.001	FGSM	1	0.992	1	0.992	0.996
			0.6	0.918	0.989	0.928	0.958
			0.2	0.916	0.965	0.905	0.934
		BIM	1	0.996	1	0.995	0.997
			0.6	0.935	0.993	0.926	0.958
			0.2	0.904	0.966	0.888	0.925
		CW	1	0.992	1	0.992	0.996
			0.6	0.943	0.99	0.937	0.969
			0.2	0.911	0.976	0.889	0.930
Pong	0.001	FGSM	1	0.86	1	0.86	0.925
			0.6	0.922	0.983	0.921	0.951
			0.2	0.93	0.974	0.917	0.945
		BIM	1	0.871	1	0.871	0.931
			0.6	0.917	0.979	0.918	0.948
			0.2	0.94	0.982	0.924	0.952
		CW	1	0.85	1	0.85	0.919
			0.6	0.917	0.979	0.919	0.948
			0.2	0.931	0.974	0.919	0.946
Seaquest	0.0005	FGSM	1	0.929	1	0.929	0.963
			0.6	0.903	0.957	0.881	0.917
			0.2	0.929	0.793	0.880	0.834
		BIM	1	0.939	1	0.939	0.969
			0.6	0.902	0.955	0.880	0.916
			0.2	0.924	0.772	0.863	0.815
		CW	1	0.928	1	0.928	0.963
			0.6	0.903	0.956	0.880	0.916
			0.2	0.926	0.783	0.872	0.825

TABLE 10. Detection Performance under Attack on Important Observations

Game	Variance Threshold	Attack	Attack Probability	Accuracy	Precision	Recall	F1 Score
Breakout	0.001	FGSM	1	0.898	0.987	0.909	0.946
			0.6	0.891	0.953	0.929	0.941
			0.2	0.776	0.819	0.903	0.859
		BIM	1	0.885	0.979	0.902	0.939
			0.6	0.895	0.954	0.933	0.943
			0.2	0.773	0.825	0.891	0.957
		CW	1	0.894	988	0.904	0.944
			0.6	0.888	0.945	0.937	0.939
			0.2	0.786	0.827	0.906	0.864
Pong	0.001	FGSM	1	0.913	0.944	0.966	0.955
			0.6	0.891	0.937	0.945	0.941
			0.2	0.866	0.903	0.935	0.919
		BIM	1	0.902	0.941	0.956	0.948
			0.6	0.893	0.939	0.945	0.942
			0.2	0.852	0.898	0.919	0.908
		CW	1	0.915	0.950	0.961	0.955
			0.6	0.888	0.935	0.943	0.939
			0.2	0.849	0.898	0.919	0.908
Seaquest	0.0005	FGSM	1	0.809	0.941	0.852	0.894
			0.6	0.786	0.906	0.855	0.880
			0.2	0.682	0.757	0.859	0.805
		BIM	1	0.815	0.938	0.862	0.898
			0.6	0.786	0.900	0.860	0.880
			0.2	0.660	0.728	0.858	0.788
		CW	1	0.815	0.942	0.859	0.899
			0.6	0.785	0.898	0.861	0.879
			0.2	0.661	0.727	0.863	0.789

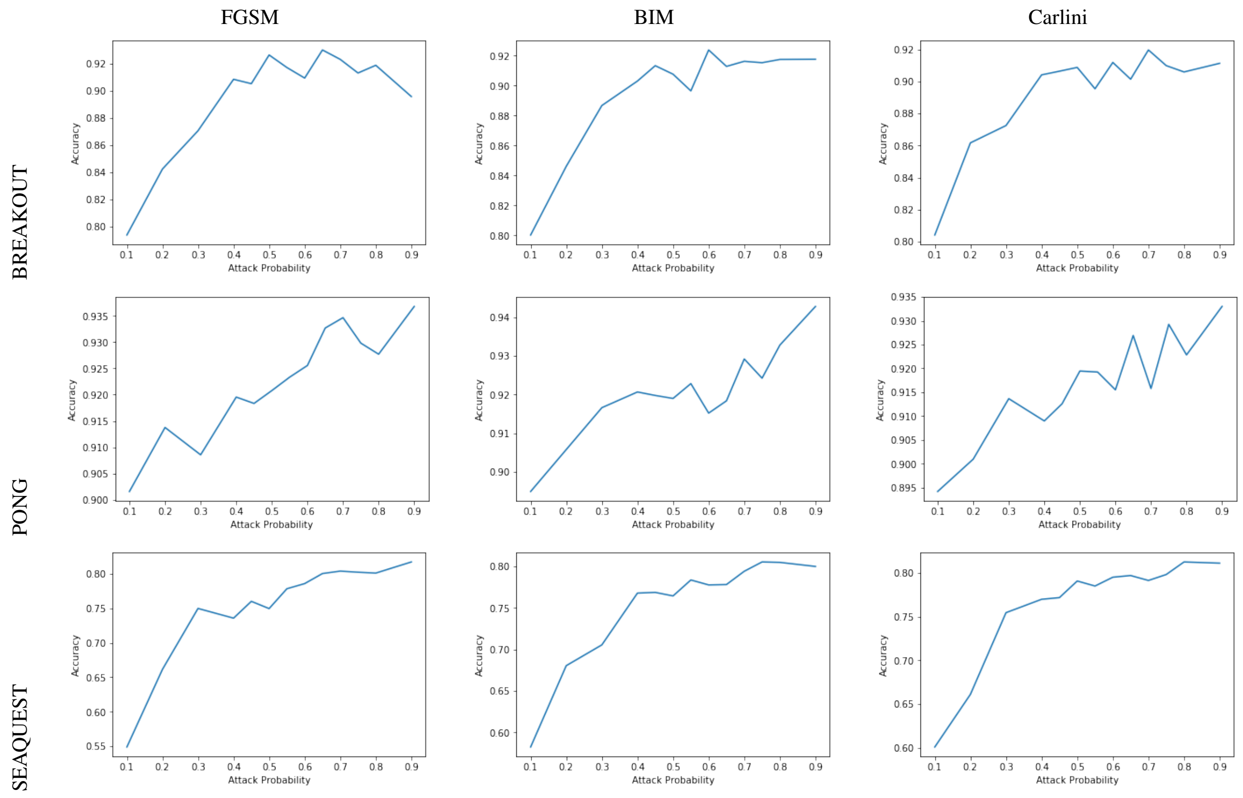

TABLE 11. **Accuracy Vs. Attack Probability on Important Observations** First column: Fast Gradient Sign Method(FGSM) [30], Second column: Basic Iterative Method(BIM) [31], and Third column: Carlini et al. [32]. The plots show the change of attack probability and its impact on the detection accuracy on important observations due to three attack types in three Atari game environments. With higher attack probability, the detection accuracy will increase.

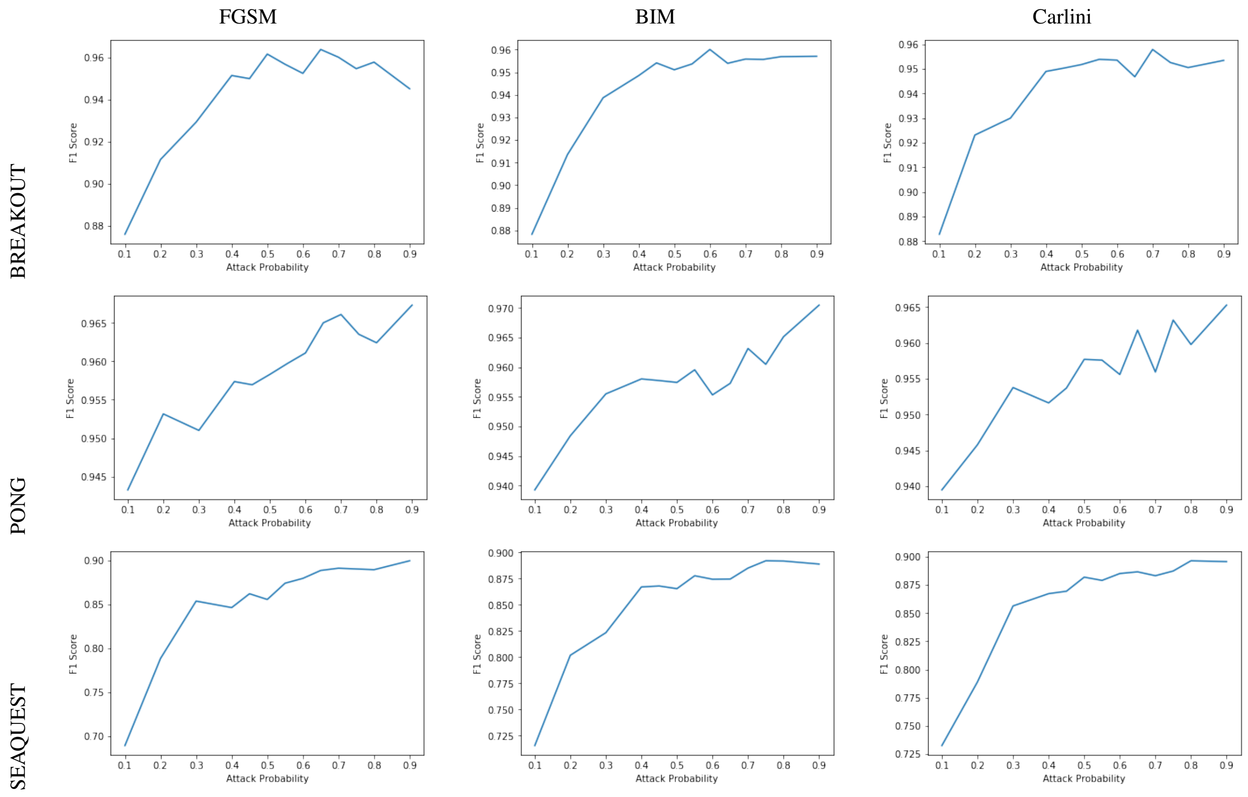

TABLE 12. **F1 Score Vs. Attack Probability on Important Observations** First column: Fast Gradient Sign Method(FGSM) [30], Second column: Basic Iterative Method(BIM) [31], and Third column: Carlini et al. [32]. The plots show the change of attack probability and its impact on the detection F1 score on important observations due to three attack types in three Atari game environments. With higher attack probability, the detection F1 score will increase.

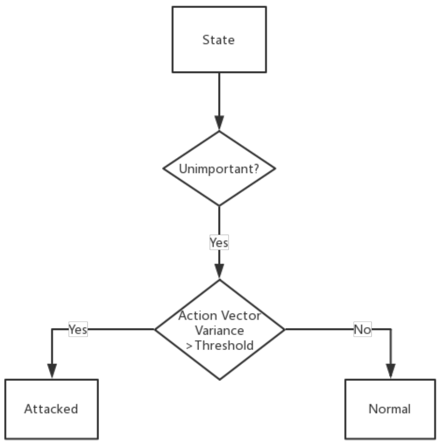

Figure 3. Detection algorithm for attacks on unimportant states

Algorithm 2 Evaluation Method per Episode

```
1:  for $each\ observation$ do
2:      num_obs $+ = 1$
3:      actver $=$ variance of the action probability vector
4:      var_thres $=$ variance threshold set by the classifier

5:      predict_class= classifier(obs)
6:      if (predict_class $==$ high_variance and actvar$<=$
        var_thres) or (predict_class $==$ low_variance and
        actvar$>$var_thres ) then
7:          num_attacked $+ = 1$
8:      end if
9:      if num_obs $==$ n then
10:         if num_attacked $/$ num_obs $>$ $threshold$ then
11:             Attacked! Stop the agent.
12:             break
13:         end if
14:     else
15:         num_attacked $= 0$
16:         num_obs $= 0$
17:     end if
18: end for
```

In Algorithm 2 we design an episode-based evaluation method for detection based on unimportant observations. Regarding the fact that the classifier cannot be one-hundred percent accurate, we set a hyperparameter n to calculate the detected attacked percentage every n observations to tolerate the error caused by misclassification. We run the agent for 100-episodes, half of the time in a normal environment and half of the time under full-time attack. Detection of attack before the episode ends is counted as successful. We then draw the Receiver Operating Characteristic (ROC) curve with different thresholds.

When successfully detected,the detection speed per episode is calculated as

$$\text{Detection Speed} = \frac{\text{num of obs before attack detected}}{\text{num of obs before reward returns}}$$

The evaluation results in game environment Pong can be seen in the appendix.

Limitations and Future Work A current issue of our work lies in the training of Deep Q-Learning models for games of varying complexity. With Pong and Breakout, the only input is moving the paddle side-to-side. As a result, these games have agents that can earn near-optimal scores when trained in a reasonable amount of time. This translates into our detector having notably good performance when detecting attack on important observations.

Seaquest is a more challenging endeavor in that it involves more movement and objectives for the agent to achieve high reward. Training a Deep Q-Learning model in Seaquest, for a similar amount of time, results in scores around $\frac{1}{10}$ of what a human player can achieve. Ultimately, this makes it harder for a detector to find attack on important observations; it is not so evidently clear what is 'important' when the model lacks firm control over its environment.

It should be noted that Seaquest still gains respectable scores in related categories (see Tables 11 and 12). The point is that a more complex game can achieve decent results quickly, but it will take significantly more training time for the detector to have parity with its simpler game equivalents. Going forward, this concern can be alleviated by experimenting with more efficient DRL approaches so that we abstract away how 'well trained' a policy can be in a reasonable amount of time. This evaluation can be carried out on various advanced games to pinpoint what game mechanics contribute to differing performance metrics.

A potential way to improve the detector performance in a complex environment is to design a more specific classifier based on the characteristic of that environment. Take Seaquest as an example, image segmentation techniques can be applied to identify the agent and objects. The importance of an observation can be better judged taking the digit such as the distance between the agent to a nearest object into account. The simple CNN classifier we implemented is a general model. We expect that in real-world application, the importance judger can be further improved with specialization.

10. Conclusion

In this work, we propose a detection method called Importance Judger. We aim at solving the issue of observation-based attacks against deep reinforcement learning models. The detection idea is based on the change in the variance of action probability given by the trained policy due to adversarial attack. The detector algorithm is simple and computationally efficient. We can achieve high detection performance targeting attack against important observations in the experiments done in OpenAI gym Atari game environments.Though mainly targeting white-box attacks where adversary has no information of the detection method, the detector is also quite robust against potential adaptive attacks. The detector algorithm can be further specialized based on the characteristics of different real-world environments. With wider application of deep reinforcement learning algorithms in various fields nowadays, we are eager to make some contribution to solve the potential safety concern issue.

References

[1] Dai, Xiaohui et al. An approach to tune fuzzy controllers based on reinforcement learning for autonomous vehicle control. *IEEE Transactions on Intelligent Transportation Systems* 6(2005): 285-293.

[2] Gandhi, Dhiraj, Lerrel Pinto, and Abhinav Gupta. "Learning to fly by crashing." 2017 *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, 2017.

[3] Nemati, Shamim, Mohammad M. Ghassemi, and Gari D. Clifford. Optimal medication dosing from suboptimal clinical examples: A deep reinforcement learning approach. 2016 *38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC).* IEEE, 2016.

[4] Levine, Sergey, et al. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research* 17.1 (2016): 1334-1373

[5] Goodfellow, Ian J.,Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples In *International Conference on Learning Representations,* 2015

[6] Li, Bo, and Yevgeniy Vorobeychik. Feature cross-substitution in adversarial classification. *Advances in neural information processing systems.* 2014.

[7] Li, Bo, and Yevgeniy Vorobeychik. Scalable optimization of randomized operational decisions in adversarial classification settings. *Artificial Intelligence and Statistics.*2015.

[8] Huang, Sandy, et al. " Adversarial attacks on neural network policies. In *International Conference on Learning Representations.* 2017

[9] Lin, Yen-Chen, et al. Tactics of adversarial attack on deep reinforcement learning agents. arXiv preprint arXiv:1703.06748 (2017).

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

[11] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. SoK: Towards the Science of Security and Privacy in Machine Learning. In *IEEE European Symposium on Security and Privacy*, 2018.

[12] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

[13] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. *arXiv preprint 1706.06083*, 2017.

[14] Tramèr, Florian, et al. "Ensemble adversarial training: Attacks and defenses." *arXiv preprint arXiv:1705.07204* (2017).

[15] Shixiang Gu and Luca Rigazio. Towards Deep Neural Network Architectures Robust to Adversarial Examples. *arXiv preprint 1412.5068*, 2014.

[16] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (Statistical) Detection of Adversarial Examples. *arXiv preprint 1702.06280*, 2017.

[17] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting Adversarial Samples from Artifacts. *arXiv preprint 1703.00410*, 2017.

[18] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On Detecting Adversarial Perturbations. *arXiv preprint 1702.04267*, 2017.

[19] Dongyu Meng and Hao Chen. MagNet: a Two-Pronged Defense against Adversarial Examples. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[20] Tretschk, Edgar, Seong Joon Oh, and Mario Fritz. Sequential Attacks on Agents for Long-Term Adversarial Goals. arXiv preprint arXiv:1805.12487 (2018).

[21] Xiao, Chaowei, et al. Characterizing Attacks on Deep Reinforcement Learning. (2018).

[22] Xu, Weilin, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *Network and Distributed Systems Security Symposium.*2018

[23] Pinto, Lerrel, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2817-2826. JMLR. org, 2017.

[24] Lin, Yen-Chen, et al. Detecting adversarial attacks on neural network policies with visual foresight. arXiv preprint arXiv:1710.00814 (2017).

[25] Bengio, Yoshua. "Learning deep architectures for AI." *Foundations and trends® in Machine Learning.* 2.1 (2009): 1-127.

[26] Schulman, John, et al. Trust region policy optimization. In *International Conference on Machine Learning.* 2015

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Workshop on Deep Learning,* 2013

[28] V. Mnih, A. Puigdomenech Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the Thirty-Third International Conference on Machine Learning,* 2016.

[29] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." In *Thirtieth AAAI Conference on Artificial Intelligence.* 2016.

[30] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *Proceedings of the Third International Conference on Learning Representations,* 2015.

[31] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial machine learning at scale. In ICLR,2017.

[32] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *IEEE SSP,* 2016

[33] Papernot, Nicolas, et al. "Distillation as a defense to adversarial perturbations against deep neural networks." *2016 IEEE Symposium on Security and Privacy (SP).* IEEE, 2016.

[34] Behzadan, Vahid, and Arslan Munir. "Vulnerability of deep reinforcement learning to policy induction attacks." *International Conference on Machine Learning and Data Mining in Pattern Recognition*. Springer, Cham, 2017.

[35] Bishop, Christopher M. Pattern recognition and machine learning. springer, 2006.

[36] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).

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

Appendix

1. Experimental Results for Detection Based on Unimportant Observations

In this paper, we focus on detecting attacks on important observations because the attacks on unimportant observations are not very effective. However, we still try our detection method with attacks on unimportant observations. We introduce the details in the following.

1.1. Experiments Setup. As Atari game Pong is a very typical environment to implement reinforcement learning algorithms with a certain highest reward that can be earned and agents can be trained well using DQN algorithm, we choose to do extensive experiments in Pong to measure the detector's performance in sensing the abnormality of the adversarial environment.Full-time attack is generated to simulate the environment where the agent is under long-term sequential attack. The lower threshold is set as 0.000001.

We run the agent with detector in Atari game environment Pong for 100 episodes,half of the time normal and half of the time under full-time attack. Episodes where attack is detected before the final reward is returned are counted as true positive while episodes in normal environment but still warned as attacked are counted as false positive. We set the hyperparameter n as 10. Various thresholds is implemented. We then draw the receiver operating characteristic curve(ROC curve).

1.2. ROC Curve. The ROC curves of the detector's performance under attack FGSM,BIM,CW_2 are shown in Figure 4,5,6.The area under the curve(AUC) is around 0.86, indicating that the detector can do a good classification job between attacked and normal environment.

1.3. Detection Speed. We choose 3 thresholds with high true positive rate and low false positive rate.The 100-episodes average detection speeds in different environments are listed in Table 13.

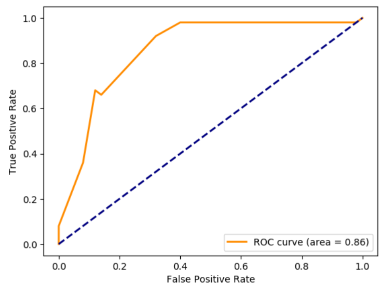

Figure 4. ROC curve of the detector's performance in detecting attack FGSM in Pong

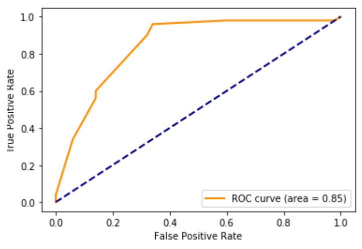

Figure 5. ROC curve of the detector's performance in detecting attack BIM in Pong

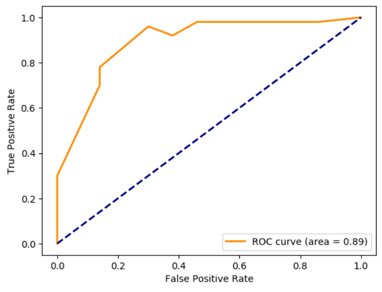

Figure 6. ROC curve of the detector's performance in detecting attack CW in Pong

TABLE 13. 100-EPISODES AVERAGE DETECTION SPEED UNDER DIFFERENT ATTACKS IN PONG

Threshold	Attack	Average Detection Speed
0.55	FGSM	0.092± 0.15
	BIM	0.068± 0.09
	CW	0.083±0.11
0.6	FGSM	0.141± 0.19
	BIM	0.142± 0.18
	CW	0.152±0.20
0.65	FGSM	0.130± 0.22
	BIM	0.145± 0.18
	CW	0.160±0.22

It can be seen that the attack can be detected at an early stage.