Institute of Computer Science IV
University of Bonn
Bachelor of Science in Computer Science

A Comparison of Adversarial Attacks against Reinforcement Learning based Applications in Tactical Networks

Bachelor's Thesis

Submitted by

Tobias Hürten

Matriculation Number: 3335135

Email: s6tohuer@uni-bonn.de

Examiner: Prof. Dr. Michael Meier[1] and Dr. Paulo H. L. Rettore[2]

Supervisor: Johannes Loevenich[2]

[1]Head of Institute Computer Science IV, University of Bonn, Germany
[1]Head of Institute, Fraunhofer FKIE, Wachtberg, Germany
Email: meier@cs.uni-bonn.de

[2]Research Scientists, Fraunhofer FKIE, Bad Godesberg, Germany
Email: [paulo.lopes.rettore, johannes.loevenich,] @fkie.fraunhofer.de

In collaboration with the Fraunhofer Institute for Communication, Information Processing and Ergonomics (FKIE), Bonn, Germany

February 14, 2023

1 Abstract

The present thesis explores the application of Reinforcement Learning (RL) methods to address stability and reliability challenges for routing problems in tactical networks. However, this approaches are susceptible to security risks in the form of adversarial samples that can manipulate the agent's observations and cause sub-optimal behavior, leading to unstable connections and packet loss. Therefore, this thesis provides evidence of the vulnerability of these applications to attacks and identifies key differences in the suite of attack methods based on metrics such as effectiveness, runtime, and detectability.

2 Acknowledgements

First and foremost I would like to thank my examiners Prof. Dr. Michael Meier and Dr. Paulo H. L. Rettore, without whom this thesis would not have been possible.

I am particularly thankful to my supervisor Johannes Loevenich, whose advice and support have been invaluable and who enriched my scientific career immensely.

Moreover, I am grateful to my fellow student researchers at Fraunhofer FKIE, namely Jonas Bode, Luca Liberto and Florian Spelter. Our cooperation over the past year laid the groundwork for this thesis and affected its quality considerably.

Contents

3 Introduction

Mobile ad hoc networks (MANETs) are becoming increasingly prevalent due to their availability and flexibility in both private and tactical network applications. However, secure and stable connections are critical in tactical networks to establish communication between military assets and personnel. Therefore, in such applications, the quality of each node significantly impacts the overall stability of the network.

Recent research on Mobile Ad Hoc Networks (MANETs) has attempted to incorporate artificial intelligence (AI) concepts to assess network parameters and determine a safe and stable transmission path. Especially Reinforcement Learning (RL) approaches have gained significant attention in network planning, including military applications. The present research aims to leverage RL agents to quantify the link stability and optimize transmission routes. [1][2]

However, when applying such methods in highly security military environments, these advancements need to be monitored for potential vulnerabilities. This thesis aims to expand previous investigations into one such novel attack vector, namely adversarial attacks. These attacks intercept the deployed agent's observation by infiltrating or jamming the network from outside and alter network parameters in a way that manipulates the victim agent to misjudge the stability of certain network paths. [2]

The remainder of this thesis is structured as follows. Section 4 provides the necessary background knowledge to understand how RL algorithms can solve learning tasks through agent-environment interactions and how they can be applied to MANETs. Section 5 formally defines the problem by outlining the environment and its restrictions, followed by the introduction of a set of adversarial attacks in Section 6. Section 7 presents the results of a set of quantitative experiments that investigate the attacks' performance with respect to a set of metrics. Finally, Section 8 summarizes the results and presents prospects for future research on adversarial attacks and defensive strategies to reduce the vulnerability of these attack vectors.

4 Background

Machine Learning (ML), as a subdomain of computer science, covers the idea of digital computers carrying out tasks without being explicitly programmed how to do so. The term was coined by Arthur L. Samuel in 1959. At the time he stated that a lot of tasks that were still carried out by human beings required close to no intellect, but still included some degrees of learning. Therefore it was aimed to replace the need of humans for these mundane tasks by having digital computers undergo a process that - in the context of living organisms - could only be described as learning. Samuel motivated two concepts of achieving this goal: the first he coined "general-purpose learning machines", that learned based on rewards or penalties but was deemed unlikely to be realized at the times. Specifically when keeping in mind the different magnitude of complexity between already existing networks and the actual neurology of living organisms such a machine was considered unrealistic. The second concept on the other hand was that of a network, that was designed to learn only specific things. [3]

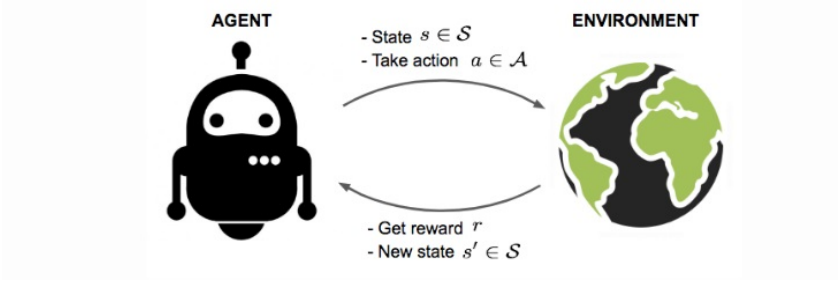


Figure 1. Typical agent-environment interaction: the agent takes action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$, whereupon it is presented a new state $s' \in \mathcal{S}$ and a reward $\in \mathcal{R}$.

As of today the field of ML underwent much more research, as well as computer science in general, which allowed for the development of machines that are capable of much more complex tasks. Hence now Samuel's first approach of a general learning machine is not only realizable, but already being deployed in varying fields.

4.1 Reinforcement Learning

RL is a subdomain of ML, together with supervised and unsupervised learning. While the latter seem to summarize the field sufficiently, RL cannot be considered either of these fields since RL algorithms on one hand do not receive correct data-label pairs, but on the other do not learn a hidden underlying structure either. They rather try to maximize some numerical value called the reward.

The general idea derives from human learning from an early age, as infants have no explicit teachers for most of their learning procedure. More often than not, they seem to execute actions at random and interact with their surroundings by doing so, only receiving some kind of feedback afterwards, which will reassure the action or decrease the probability of a future repetition by some consequence. This makes RL the ML approach that comes closest to actual human learning as already described by Samuel in 1959 [3], even though he deemed the general purpose machine unfeasible at the time.

Hence two key features of RL are *trial-and-error-search* and *delayed rewards* meaning the trial of different actions and observing the feedback or reward, which itself depends on the reward of the next state and therefore all subsequent states.

A commonly described dilemma, that is unique to RL, is that of exploration vs exploitation. Since RL approaches learn by exploring different actions and observing their outcome, exploration is an essential part of RL, however the overarching goal of any learning task is to capitalize on past experience. This concept of exploitation stands in direct contrast to the aforementioned exploration. While it makes sense to bring this dilemma up now, as it perfectly describes the core trade off in RL, possible solutions will be presented in a later section.

Overall, RL is an intuitive concept in which real world applications are deeply integrated into the whole procedure of learning: agents are goal driven entities that interact with their surroundings through actions and are then presented with a new changed state. [4]

4.1.1 Terminology

While a lot of these concepts will be introduced in a more mathematical fashion beginning with Section 4.1.2, it is helpful to firstly have an intuitive idea of how they work.

As already described the RL model is defined by its actor-environment duality. Furthermore, an interaction between the two can be sufficiently described by four features: a state that contains information about the environment, an action that is chosen by the agent, the reward signal that depends on both the selected action and the current state, as well as the following state that describes the environment after the action was taken. Note that constructing and changing states as a problem is part of constructing and defining the environment, not the RL process itself. States are formally defined using Markov Decision Processes (MDPs) in Section 4.1.2. [4]

The reward defines the overarching goal or conversely, the overarching goal defines how to compute the numerical value that is the reward. Hence the reward is chosen in a way, that by maximizing it, the agent can learn how to solve the problem, which is defined by the environment. Sutton et al. (2018) [4] compare the reward to an organism's sense of pleasure and pain since just like these phenomenon, the reward can be of positive or negative nature. Generally, it is desirable that, throughout the learning process, those actions that yielded smaller or even negative rewards should become less likely to be performed again, while the probability of those with large, positive rewards should increase.

The decision making of an RL model is referred to as its policy, which is essentially the behavioral code of the agent and decides which action to perform for a given state. It can therefore be interpreted as a mapping from state to action. Sutton et al. [4] suggest, that the policy may be a simple table lookup, a function that assigns values or a complex computation like a decision tree search. Furthermore literature differentiates between stochastic and determinant policies. While the latter returns a definite action for a given state, most real life applications involve stochastic policies, that yield probabilities with which each of the actions is performed for that state.

To solve the RL problem, an agent is explicitly maximizing the reward over all interactions with the environment. Given a state S_t at time step t the expected reward of an arbitrary action A_t can be written as

$$q^*(a) = \mathbb{E}[R_t | A_t = a]. \tag{1}$$

Since RL does not work with a teacher that could explicitly tell the values of each of the actions, the true values q^* are unknown. Thus the model has to work with estimates, otherwise the identification of the single action that yields the best rewards would be a trivial problem. The so called value function Q can estimate this cumulative sum of rewards over future states, beginning with the current state. These functions therefore indicate long-term benefits of certain actions given a state. To further build upon their previous analogy, the authors of [4] describe pleasure and pain as intermediate rewards, while value functions imitate a more founded and informed judgement. For example the consumption of sweets and candy might lead to immediate pleasure, while an informed, balanced diet will bring long term health benefits.

Whereas rewards are received as feedback from the environment, values have to be estimated and re-estimated from a sequence of observations called trajectory. Ideally these

estimates would be close to $q^*(a)$, the actual value of an arbitrary action. Sutton et al. [4] initially motivate a very basic way of estimating values by averaging over all rewards, that were received for that action so far:

$$Q_t(a) := \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t}. \tag{2}$$

However, the inner workings of Q can vary vastly, it might for example be a simple table, that contains values for reach state-action pair - thus usually referred to as Q-table - or compute the values in a similar way as Equation 2. Therefore, throughout the three main components of an RL model - agent, environment and value estimation function - the latter is by some considered to be the most important one. Either way, a method to efficiently compute these value estimates is probably the most defining component of every RL approach. [4]

So called model-based methods include a fourth component: a model of the environment, that is learned and then used to predict the next states given a state-action pair. The opposite of this type of RL approaches are model-free methods.

The most intuitive method to select actions is by following a greedy policy: select the action whose estimates are greatest. These actions are called greedy actions, of which there has to be at least one. Such greedy action selection means exploiting the current policy and can be expressed as:

$$A_t := \arg\max_a Q_t(a). \tag{3}$$

To further improve the current policy however, it is necessary to explore new actions in addition to exploiting previous experience. This leads to the aforementioned exploitation-exploration dilemma, that is so characteristic for RL methods. The dilemma comprises that exploitation and exploration are mutually exclusive since it is impossible to explore new actions while greedily following the current policy. Hence it is sometimes justified to put up with a worse immediate reward to try out a new action, that might yield a larger value in the long run, which can then be exploited over and over again. One possible solution to the dilemma is using a so called ϵ *greedy policy*: the current policy is exploited greedily only with probability $1 - \epsilon$. With the remaining probability of ϵ a new random action is selected and explored, and therefore considered in the following policy update. A policy that uses a rather small ϵ (e.g. $\epsilon = 0.01$) will improve more slowly, compared to a model using a larger ϵ (such as $\epsilon = 0.1$), however would eventually perform better or at least more consistently because the one with $\epsilon = 0.1$ will still only exploit its policy with a probability of 90%, even if the value estimation was very close to q^*. [4]

This suggests that a larger ϵ is useful at an early stage of the training phase, but might grow to be more of a hindrance once the policy is well developed and optimized. Hence it can be practical to decay ϵ over the course of the learning process. That being said, for a lot of RL problems the true action-values can change over time, making exploration relevant regardless of how well developed the agent is.

Since storing the reward for every time an action was performed can rapidly become memory intensive, the authors of [4] advise to compute the values incrementally:

$$\begin{aligned} Q_{n+1} &:= \frac{1}{n}\sum_{i=1}^{n} R_i \\ &= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\ &= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\ &= \frac{1}{n}\left(R_n + (n-1)Q_n\right) \\ &= \frac{1}{n}\left(R_n + n\cdot Q_n - Q_n\right) \\ &= Q_n + \frac{1}{n}[R_n - Q_n] \end{aligned} \tag{4}$$

This update rule follows the general form:

$$NewEstimate \leftarrow OldEstimate + StepSize\,[Target - OldEstimate], \tag{5}$$

which can be recognized later in Section 4.1.4. Note that the $Target$ is a desirable update direction that aims to reduce the estimate error $[Target - OldEstimate]$. The constant $StepSize$ or learning rate - often denoted as α - discounts rewards that lie back in the past. Equation 6 demonstrates how α is applied recursively:

$$\begin{aligned} Q_{n+1} &= Q_n + \alpha[R_n - Q_n] \\ &= \alpha\cdot R_n + (1-\alpha)\cdot Q_n \\ &= \alpha\cdot R_n + (1-\alpha)[\alpha\cdot R_{n-1} + (1-\alpha)\cdot Q_{n-1}] \\ &= \alpha\cdot R_n + (1-\alpha)\cdot\alpha\cdot R_n + (1-\alpha)^2\cdot Q_{n-1} \\ &\ldots \\ &= (1-\alpha)^n\cdot Q_1 + \sum_{i=1}^{n}\alpha\cdot(1-\alpha)^{n-i}R_i \end{aligned} \tag{6}$$

Since $1-\alpha < 1$ this weight decays exponentially with each recursive step, which is why this method of averaging is often called *exponential recency-weighted average*.

As for value initialization Sutton et al. [4] motivate optimistic initial values since these will result in disappointing actual rewards, whatever action is selected. This in return leads to the trial of new actions until the values are lowered by following the update rule, which encourages exploration even for truly greedy action selection, as can be seen in Figure 2. This however does not account for non stationary problems with changing true values since initialization only happens once.

An alternative way to select actions, other than using action-value estimates, is by exploiting an action preference function. Such a function assigns a numerical value to each action

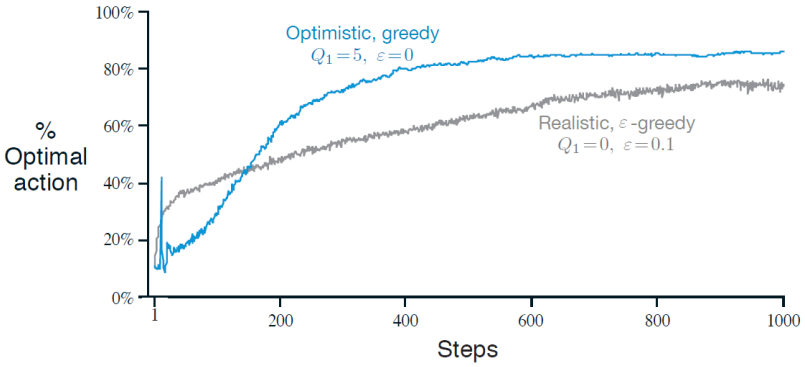


Figure 2. Different value initialization approaches to demonstrate how optimistic initial values encourage exploration. In the beginning each new action will yield a disappointing reward because the values are initialized too optimistically, leading to the trial of new actions. Both approaches use the same constant step-size $\alpha = 0.1$. [5]

that can no longer be interpreted in terms of reward but only relative to other actions. The action probabilities can then be computed according to a *soft-max distribution*:

$$Pr\{A_t = a\} := \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} := \pi_t(a), \tag{7}$$

reflecting the probability of taking action a at time step t. Such an action preference function will become relevant for certain attack strategies in Section 6, when attacks are only performed if the victim agent has a high preference for a specific action over the others. [4]

4.1.2 Finite Markov Decision Processes

MDPs formally define the RL problem. They hence mathematically describe the interaction of the RL model with the environment through rewards, actions and states. As already mentioned in Section 4.1, the two key features of MDPs are therefore the agent and the environment. A typical interaction has the agent observe the environment's state and perform an action which changes said state to in turn provide the agent with a new observation. Additionally the environment presents the agent a reward, some numerical value, that rates the chosen action given the state and which is to be maximized to solve the underlying problem. The aforementioned interaction is depicted in Figure 3, which also emphasizes its cyclic nature.

More formally, at each time step t out of a sequence of time steps the agent is presented a state $S_t \in \mathcal{S}$ (also called observation) and is asked to choose an action $A_t \in \mathcal{A}$, after which it will receive a reward $R_{t+1} \in \mathcal{R}$ and observe the environment in a new state S_{t+1}. This sequential interaction can then be formally expressed as a trajectory:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots \quad . \tag{8}$$

In finite MDPs, which this thesis and most related work is focused on, each of these introduced sets $\mathcal{S}, \mathcal{A}$ and $\mathcal{R}$ has a finite magnitude and therefore a finite number of elements.

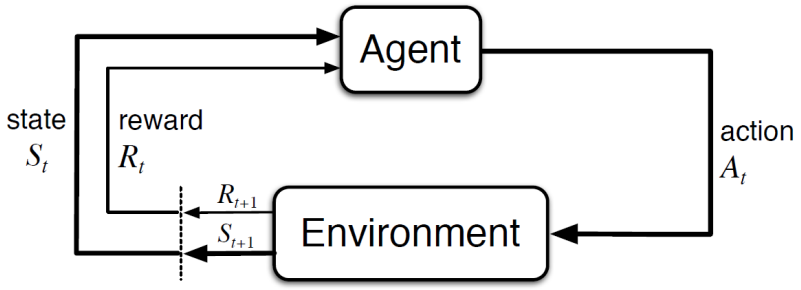


Figure 3. Agent-environment duality in an MDP. [5]
The agent is provided with a state S_t, based on which it selects an action A_t and is in return again presented with a new (subsequent) state S_{t+1}, as well as a reward signal R_{t+1}.

Hence a transition $(s, a) \to (s', r)$ has a specific probability that can be expressed as a function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$:

$$p(s', r|s, a) := Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}, \tag{9}$$

for all $s', s \in \mathcal{S}$, $r \in \mathcal{R}$ and $a \in \mathcal{A}$, indicating the probability of landing in state s' with reward r given action a was chosen in state s. Note that some literature prefers to write $a \in \mathcal{A}(s)$, denoting all actions that can be possibly performed in state s, however considering the probability of performing all other actions that are in $\mathcal{A}$ but not in $\mathcal{A}(s)$ will simply receive a probability of 0, both of these are equivalent.

A very significant part of formalizing the RL problem in this manner is that each state has the *Markov property*, meaning each state s contains all future critic information about past transitions. Therefore the probability of a transition at time step t only depends on the last observed state S_{t-1} and action A_{t-1}, as opposed to the whole trajectory up to t. [4]

4.1.3 Returns and Episodes

As aforementioned, the idea of this interaction-based trial-and-error learning is maximizing some numerical value called the reward. More specifically this reward is to be maximized over a trajectory of transitions - meaning the agent strives to maximize the cumulative reward. This concept can be formalized as a function over the reward called the *return* which in its most simple version can be defined for a trajectory over time steps $t = 0, 1, ..., T$ as:

$$G_t := R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T. \tag{10}$$

Note that this definition makes sense for episodic scenarios, where each trajectory will undeniably end in a terminal state and resetting the environment to run another episode will no longer take into account the terminated trajectory. In other scenarios the trajectories and therefore the return might very well be infinite:

$$G_t := \sum_{k=0}^{\infty} R_{t+k+1}. \tag{11}$$

Using this equation the return can also be expressed recursively:

$$\begin{aligned} G_t &:= \sum_{k=0}^{\infty} R_{t+k+1} \\ &= R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \\ &= R_{t+1} + G_{t+1}, \end{aligned} \tag{12}$$

which allows the return to only depend on the immediate reward and the return of the following time step $t+1$. This has significant implications for RL algorithms, as will be seen in Section 4.1.4.

Another important concept regarding the return is *discounting*. Even though the agent is to maximize the cumulative reward, often times it is more sensible to prefer close rewards over those that lie far in the future. This idea can be formally defined as *discounted return*:

$$G_t := R_{t+1} + \gamma R_{t+2} + ... = R_{t+1} + \gamma G_{t+1} = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{13}$$

where $\gamma \leq 1$ is called the discount factor that decreases the impact of rewards the further in the future they are. $\gamma = 1$ is equivalent to the previous return in Equation 11, while $\gamma = 0$ corresponds to a return function that only values the immediate reward. Note that for $\gamma < 1$ the expression γ^k converges to 0, meaning the return is finite even if $T = \infty$. [4]

4.1.4 Policies and Value Functions

Once again, the concept of policies was already touched upon in Section 4.1.1. Knowing that value functions are estimators that can evaluate either states or state-action pairs and having defined the expected return as a metric to quantize how good a state or state-action pair is in Section 4.1.3, it is no surprise that value functions are only defined with respect to to a specific policy. This is necessary since future rewards depend on future states - and thus future actions which in return are chosen depending on the current policy. Sutton et al. [4] define a policy π as a function that maps states to probabilities of selecting each possible action. Thus $\pi(a|s) \in [0, 1]$ is the conditional probability of selecting action a in state s. A specific RL implementation is mostly defined by its method for value estimation and its policy update rule.

Given a policy π the value function can thus be further specified as:

$$v_\pi(s) := \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1}|S_t = s \right], \text{ for all } s \in \mathcal{S}, \tag{14}$$

meaning the expected discounted reward given the agent follows policy π for all states $s \in \mathcal{S}$. This function v_π is called the *state-value function for policy* π. As aforementioned another way to define a value function is as two parameter function that evaluates state-action pairs. Such a function is called *action-value function for policy* π, usually denoted as $q_\pi(s, a)$:

$$q_\pi(s, a) := \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1}|S_t = s, A_t = a \right]. \tag{15}$$

By keeping an average of the actual returns, the estimated value of $v(s)$ will converge to the actual value $q^*(s)$ if the number of times the state is encountered approaches infinity. Likewise, by keeping separate averages for each action, these will subsequently converge to the actual action-values. These methods of averaging over a large number of encounters are called *Monte Carlo methods* [4].

Another important feature of value functions is the fact that the value of state s only depends on the estimated values of the following states recursively, in a similar fashion as shown for the expected return in Section 4.1.3. This characteristic can hence be expressed by the so called *Bellman equation*:

$$\begin{aligned} v_\pi(s) &:= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma \cdot G_{t+1}|S_t = s] && \text{by Equation 12} \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \cdot \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma \cdot v_\pi(s')\right], && \text{for all } s \in \mathcal{S} \qquad (16) \end{aligned}$$

with $a \in \mathcal{A}(s)$, $s, s' \in \mathcal{S}$ and $r \in \mathcal{R}$.

Using this Bellman equation, the value of each state can be updated using the estimated values of the following states which can then each be updated with respect to their following states. This means that similar to the return (see Equation 12), value functions satisfy recursive relationships [4].

This process of value updates is called *iterative policy evaluation* since all value estimates are iteratively brought closer to the actual values. As aforementioned, this in theory converges to the true values only in the limit, that is for infinite repetition, however in practice it is satisfactory to run the iterative policy evaluation finitely until the updates become sufficiently small.

This value function however still only evaluates states with respect to the current policy π. To improve this action selection strategy the outcome of new actions has to be explored. Given a policy π and the corresponding state-value function $v_\pi(s)$, a new arbitrary action a can be evaluated by assuming this action is selected for state s after which the current policy π will be followed again:

$$\begin{aligned} q_\pi(s, a) &:= \mathbb{E}[R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) \,|\, S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma \cdot v_\pi(s')]. \qquad (17) \end{aligned}$$

This action-value of a given state s therefore depends on the immediate reward of selecting that action plus the discounted state-value of the subsequent state s' with respect to the current policy π. Therefore, if the action-value $q_\pi(s, a)$ is better than the state-value v_π (which averages the value over all possible actions), the probability of selecting action a in state s should be increased for the policy update. Note that "probability" implies a stochastic policy however this analogously applies for deterministic policies as well. [4]

The next logical step is to not only consider a single action for a single state but all actions and all states respectively. The resulting new greedy policy π' is hence given by:

$$\begin{aligned}\pi'(a|s) &:= \arg\max_a q_\pi(s,a) \\ &= \arg\max_a \mathbb{E}[R_{t+1} + \gamma \cdot v_\pi(S_{t+1})|S_t = s, A_t = a] \\ &= \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma \cdot v_\pi(s')], \end{aligned} \tag{18}$$

meaning the policy greedily selects actions according to $v_\pi(s)$ with one step of look-ahead.

This new policy π' will be better than the previous one except for the case that π was already optimal. After updating policy π the state-value function can be computed again which in return can be used to update the policy once more. This process of gradually improving the policy while always updating the value estimates is called *policy iteration* and can be illustrated as a sequence as follows:

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} ... \xrightarrow{I} \pi_* \xrightarrow{E} v_* \tag{19}$$

As Sutton et al. state: since finite MDPs only have a finite number of policies and the policy is guaranteed to be a strict improvement, this policy iteration process must converge to the optimal policy in a finite number of iteration steps [4]. As already stated, *policy evaluation* itself only converges in the limit however it can be truncated. One essential way is to stop policy evaluation after a single step. This is called *value iteration* and essentially turns the Bellman equation into an update rule:

$$\begin{aligned} v_{k+1}(s) &:= \max_a \mathbb{E}[R_{t+1} + \gamma \cdot v_k(S_{t+1})|S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma \cdot v_k(s')], \end{aligned} \tag{20}$$

for all $s \in \mathcal{S}$. Value iteration hence combines one step of policy evaluation and one step of policy improvement. To summarize policy iteration is a process to improve a given policy by iteratively aligning the value function to the current policy and then making the policy greedy with respect to the value function as shown by Figure 4. [4]

4.1.5 Monte Carlo methods

Monte Carlo methods were already referred to in Section 4.1.4 when talking about averaging over large samples. The first key feature of Monte Carlo methods is the fact that they do not rely on information about the environment but only on experience that consists of sequences of states, actions and rewards. These experiences can be gathered either through actual interactions with the environment or by simulating them - in the case of model-based methods.

For simplicity, Sutton et al. [4] limit themselves on episodic problems, that is problems whose experience is divided into episodes that will undeniably terminate at some final time

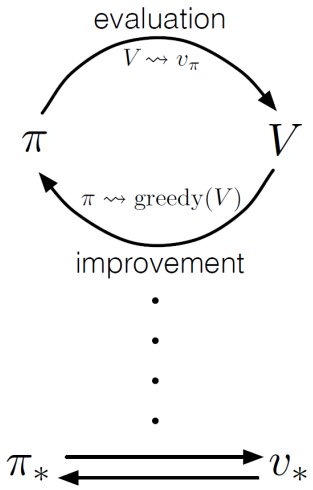


Figure 4. Illustration of the iterative cycle that is typical for value iteration. [4] First the policy is evaluated by computing the value function. Then the policy itself is updated to greedily select the best action with respect to the new value function. This way the algorithm iteratively alternates between policy evaluation and policy improvement.

step T. Policy evaluation and policy improvement take place only after an episode has finished, meaning value estimates and policies are only changed after reaching a terminal state. Monte Carlo methods therefore average over the complete return for each episode iteratively (in the next Section a method will be discussed that learns from partial returns). Consequentially Monte Carlo methods estimate the state-value function v_π using samples that are collected by following policy π a number of episodes, to then average the expected return, that is the expected cumulative sum of future discounted rewards. As the authors of [4] state: As the number of times that a state s is visited increases, this estimate should converge to the expected value by the law of large numbers.

Since the values are estimated using experience of actual episode samples, the estimate of each state-value only depends on that state, as opposed to other states as seen before. This has implications for the computational expense of Monte Carlo methods.

It proves particularly useful to estimate the action-values instead of state-values since these already contain all critic information to follow a greedy policy, simply by selecting the action that - according to the action-value function - yields the best reward and next state pair. The reader should remember that the policy evaluation problem for action-values was defined by choosing a new action a in state s and thereafter following policy π again. The action-value function $q_\pi(s, a)$ can be estimated analogously to the state-values by averaging over each visit of a state-action pair (as opposed to just states), meaning each time a specific action was chosen in the specific state s. Again as the number of times the pair is visited approaches infinity the values converge to q^*.

A simple way of updating the value estimate V of v_π could look like this:

$$V(S_t) \leftarrow V(S_t) + \alpha \cdot [G_t - V(S_t)], \tag{21}$$

where G_t is the actual return following time step t and α the constant step size parameter as introduced in Section 4.1.1. [4]

This only leaves the problem that, by following the policy π, some state-action pairs may actually never be visited, which becomes exceedingly problematic since policy evaluation for action-values works by comparing $q_\pi(s, a)$ with $v_\pi(s)$, the average over all possible actions a in state s. This should remind the reader of the exploration-exploitation dilemma introduced in Section 4.1.1.

Possible solutions are either ensuring that each state-action pair has a nonzero probability of being selected at the start of an episode or to only consider those policies that are both stochastic and have nonzero probabilities for every action in every state.

Finally, Monte Carlo methods can be used to improve the policy in a similar manner as discussed in Section 4.1.4: the algorithm samples an episode to collect experience which is then used to perform the policy evaluation and update the value function after which the policy can be adjusted based on the values of the visited states or state-action pairs. The idea of value iteration (also Section 4.1.4) is one way to eliminate the problem that these value estimates only converge for the limit by truncating policy evaluation to - in the case of value iteration - only one step (of policy evaluation) between each step of policy improvement. [4]

4.1.6 Temporal Difference Learning

Temporal Difference (TD) learning is another significant notion in RL and has meaningful implications for a lot of state-of-the-art algorithms that are being deployed today. In contrast to Monte Carlo methods, which have to wait for an episode to terminate to gather the actual return G_t (see Equation 21), TD methods only need to await the next time step $t+1$. Then TD methods can compute a *target* using the received immediate reward R_{t+1} and the discounted value estimate $V(S_{t+1})$ for the following state S_{t+1}:

$$V(S_t) \leftarrow V(S_t) + \alpha \cdot [R_{t+1} + \gamma \cdot V(S_{t+1}) - V(S_t)]. \tag{22}$$

To be more exact this example is called ***TD(0)*** - a special case of $TD(\lambda)$. Note that Monte Carlo methods also have an (implicit) target: the actual return G_t. The computed difference between the TD target and the current value estimates is called the TD *error*:

$$\delta_t := R_{t+1} + \gamma \cdot V(S_{t+1}) - V(S_t). \tag{23}$$

The first essential advantage of TD learning over Monte Carlo methods is the fact that learning (as in updating the values and policy) does not need to be delayed until after an episode terminates. This saves a considerable amount of time, especially in applications that potentially have very long episodes, as often is the case. Similarly these methods are very advantageous in scenarios with continuous tasks that have no terminating state whatsoever.

4.1.7 Policy Gradient methods

Until this point all methods aimed to estimate the actual values of each action, i.e. the action-value function. This however is exceedingly expensive for higher action space dimensions. Hence so called *gradient policy methods* try to learn a parameterized policy instead. This parameterized policy π_θ can be expressed as:

$$\pi(a \mid s, \theta) = Pr\{A_t = a \mid S_t = s, \theta_t = \theta\}. \tag{24}$$

Then, these parameters θ can then be improved during a policy update that aims to maximize some objective function $J(\theta)$. This can be achieved by estimating the gradient with respect to the policy parameters θ:

$$\theta_{t+1} = \theta_t + \alpha \cdot \widehat{\nabla_\theta J(\theta_t)}, \tag{25}$$

with the policy's parameters θ, learning rate α and the estimated gradient of the objective function $\widehat{\nabla_\theta J(\theta_t)}$. This process of gradually improving the policy's parameters in the direction of the greatest increase of the objective function (its gradient direction) is usually referred to as *Stochastic Gradient Ascent (SGA)*.

Naturally, this method requires the policy π_θ to be differentiable with respect to its parameters. Furthermore, it is generally preferred for policies to never become deterministic, as to ensure exploration. A common way to receive a parameterized policy based on action-values is to use an exponential *soft-max* distribution:

$$\pi(a \mid s, \theta) = \frac{e^{h(s,a,\theta)}}{\sum_{b \in \mathcal{A}} e^{h(s,b,\theta)}}, \tag{26}$$

where $h(s, a, \theta)$ defines a numerical action preference for each state-action pair and could be potentially acquired using a neural network.

One significant advantage of these policy gradient methods is the fact, that the policy defines an actual action probability distribution, meaning actions are not assigned values but a probability by which it should be performed. For probabilities smaller than 1, this means that there will still be a chance of exploration, however if the optimal policy happened to be deterministic, meaning there is one single best action for each state, a policy gradient method will converge to a probability distribution in which the likelihood of optimal actions are infinitely higher than for suboptimal ones. [4]

4.1.8 Actor-Critic methods

The previously discussed RL methods generally fall into two categories: *actor-only methods* and *critic-only methods*. Critic-only methods generally rely on value approximation and try to acquire optimal policies by approximating solutions to the Bellman equation (16), whereas actor-only methods use a parameterized policy which is then updated based on the gradient with respect to θ as introduced in Section 4.1.7. Actor-critic methods try to combine both approaches in an off-policy manner: the critic policy approximates the value function and is then used to update the actor policy's parameters. [6]

This thesis will use two actor-critic algorithms: Proximal Policy Optimization (PPO) and Advantage Actor Critic (A2C).

A2C implements actor-critic behavior in its purest form: the critic policy is updated by approximating a solution to the Bellman equation (16) and computes an advantage function, that assigns each performed action a numerical value which is then used to update the actor policy's parameters θ in the gradient direction. [7]

PPO on the other hand was presented by Schulman et al. in 2017 and aims to extend so called trust-region methods, which impose a constraint on the size of policy updates which reduces the chance of straying too far from the previous policy during a single update which could potentially corrupt the policy. PPO updates the actor policy using a surrogate objective function which clips the product of the advantage function (critic) and the ratio between the new updated policy and the old policy. This way, maximizing the objective function means increasing the probability of actions that have a positive advantage, while the likelihood of those actions with a negative advantage are decreased. Furthermore each update is clipped in size and hence limited to a "trusted" region. [8]

4.2 Adversarial Attacks

After introducing the basic working environment, it is time to introduce the main topic of this thesis: *adversarial attacks*. The vulnerability to so called *adversarial examples* was firstly discovered by Szegedy et al.[9] for computer vision classification in 2014. These models' accuracy was decreased significantly by applying noise to the original images, that was not perceivable for human beings whatsoever (see Figure 5).

4.2.1 Fast Gradient Sign Method

Goodfellow et al. (2014) [5] continued researching this phenomenon and state that these adversarial examples were miss-classified by the computer vision models, even though they were only slightly altered versions of the original images. They suggest that linear behavior in high-dimensional spaces is sufficient to cause the phenomenon of adversarial examples and thus render such image classifiers vulnerable to these samples [5].

Furthermore the authors of [5] established a method to craft these samples efficiently, to then present them to a model during its training state as to make it more robust to such scenarios.

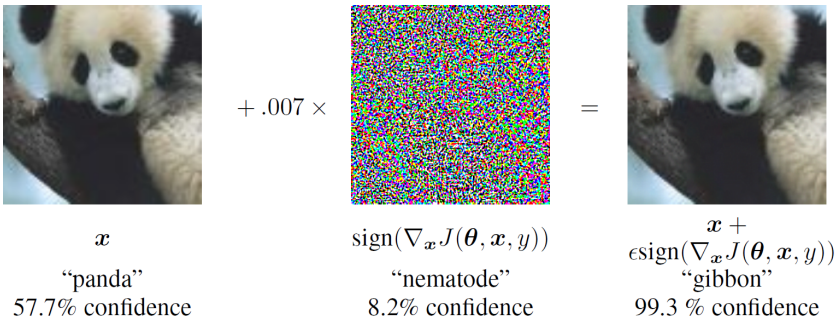


Figure 5. Noise generation using FGSM by [5]. Noise equals the signed gradient of the cost function and is not perceivable to the human eye ($\epsilon = 0.007$). Perturbing the original input results in a miss classification of the image as "gibbon" with an even higher confidence than for the unaltered image.

Firstly an adversarial sample - denoted as $\tilde{x}$ - can be crafted by adding some noise η to the original input x:

$$\tilde{x} := x + \eta. \tag{27}$$

Furthermore an image classifier is expected to respond to an adversarial sample no different than to the original input, if the noise satisfies the L_∞ constraint $||\eta||_\infty < \epsilon$, where ϵ is a constant, that is small enough to be discarded by the sensor or data storage apparatus in use [5].

A typical method for linear image classification is computing the weighted sum, which in the case of adversarial samples, given a weight vector w, is defined as follows:

$$w^T \tilde{x} = w^T x + w^T \eta. \tag{28}$$

This suggests that adversarial perturbation causes the activation to grow by $w^T \eta$ [5]. Therefore it was motivated that $\eta = sign(w)$ maximizes η with respect to the aforementioned constraint. Thereafter Goodfellow et al. (2014) [5] expand this concept, as they suggest that additionally this methodology should also affect neural networks. Given the parameters θ of a neural network model, the input vector x and the correct labels y they introduced an efficient method to craft adversarial samples which they named the *Fast Gradient Sign Method (FGSM)*, that takes an ϵ-scaled version of the name-giving signed gradient of the network's loss function $J(\theta, x, y)$ with respect to the input and computes η as follows:

$$\eta = \epsilon \cdot sign(\Delta_x J(\theta, x, y)) \tag{29}$$

[5]

Figure 5 demonstrates how FGSM is applied to an image, changing the network's output from the correct label to the wrong label with a very high corresponding confidence.

4.2.2 Adversarial Attacks on Neural Network Policies

This concept - while originally developed for supervised image classifiers - was proven applicable for unsupervised learning methods, as well as RL models. Huang et al. (2017) [10] examine the effect of adversarial samples on neural network policies that were trained using deep RL (which applies neural network techniques to RL concepts).

However while supervised image classifiers take images as input x and assign them labels y, an RL model's stochastic policy π takes a state s as input x and outputs a weighting y over all possible actions. To craft an adversarial sample, it is generally assumed that the model solves the underlying problem sufficiently well, meaning the most probable action in π can be considered the best action:

$$a^* = \arg\max_a \pi(s), \tag{30}$$

given state s.

FGSM's objective function can then be expressed as the cross-entropy loss between the policy's probability distribution and the distribution that assigns a^* a probability of 1 and all other actions a probability of 0 since this would be the best probability distribution assuming a^* is indeed the best action. This way FGSM can be applied to RL models without having access to the correct outputs $\hat{y}$, as would be the case for image classifiers.

4.3 Mobile Ad hoc Networks

Lastly after introducing all relevant concepts, it is necessary to address the application case of the RL models that will be attacked in Sections 6 and 7.

The massively increasing number of mobile devices in use by people around the world every day heightens the complexity of setting up conventional network structures exponentially. Traditionally so called infrastructure-based networks provide an efficient and stable manner in which these devices can connect and communicate with each other. Here, each device connects to the closest stationary connection point which then relays the sent information to another base station that distributes it further to the target devices. However the time and monetary cost to set this infrastructure up, combined with an increasing amount of scenarios in which the user requires a specific infrastructure that for whatever reason is not available and also not installable - be it because of geographical reasons or otherwise - caused the need to further explore alternative methods to provide connectivity.

MANETs allow mobile devices to connect with each other ad hoc, that is as the case arises. In a MANET each device can connect to all the other mobile devices within connection range to form an ad hoc network. Within this network each of the connected devices functions as nodes that can relay packets to other nodes, forming an interconnected net of nodes as depicted in Figure 6. [11]

Since each of these devices remains mobile, this means that the topology of the network can change rapidly and unpredictably when devices change their location or disconnect from

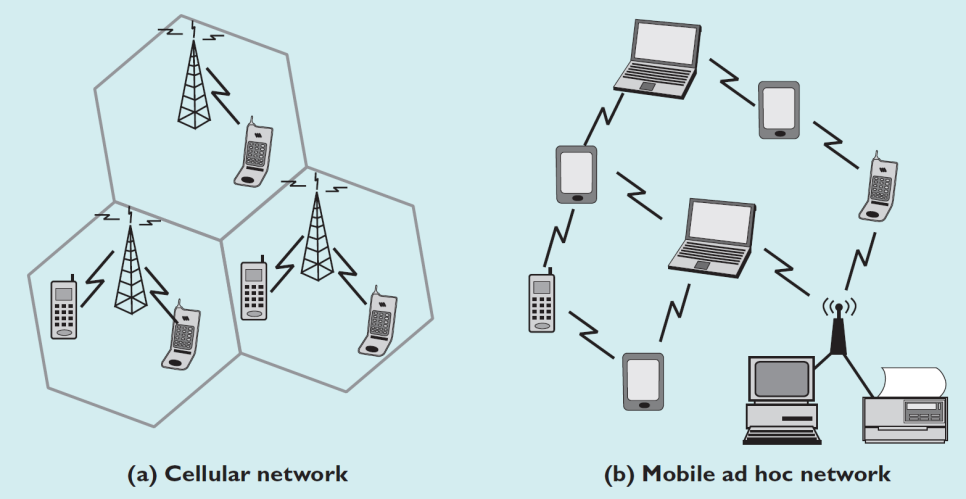


Figure 6. Structure of a conventional cellular network (a) compared to a MANET (b). [12] In (a) each device communicates with the closest station which relays the package to the target device via its closest station. In (b) however each device functions as a node in the network and can relay packages itself.

the network. To accommodate for this, all the nodes within the MANET have to self organize themselves depending on the location and number of connected devices. While a MANET does not have to be connected to the internet to operate, by doing so each node in the network can relay the internet connection to other nodes.

Overall the characteristics of MANETs allow them to be used in scenarios where fixed infrastructure is not available or not trusted. This includes application cases for tactical networks, that is networks used for military communication, which heightens the importance of stable, secure and trusted connections. [12]

Recent research tries to incorporate RL method in MANETs to select the sequence of nodes to transfer information from a source node to a target node within the network. [13] Naturally these routing algorithms inherit all advantages and disadvantages of RL methods, including their vulnerability to adversarial samples, as discussed in Section 4.2. Therefore this thesis aims to investigate the susceptibility of RL routing algorithms for MANETs to a series of adversarial attacks - including different attack methods, strategies and configurations, focusing on the special, highly security critic case of tactical networks.

5 Problem

5.1 The Environment

The selected environment was adopted from Hürten et al. (2022) [2] who previously utilized it to assess adversarial attacks on tactical networks. The environment is designed to emulate a MANET and incentivizes an RL agent to discover the most resilient path by maximizing its reward.

The network is represented by an edge-weighted graph $G := (V, E)$, where the set of tactical nodes V is connected by network links E. Each edge e in the graph is characterized by a three dimensional weight $w : E \to \mathbb{R}^3$. These weights combine three network metrics measured in experiments with real military radios in a laboratory setup [13]. More precise, the weights include the queue length $q_e \in 0.0, 0.2, 0.4, 0.6, 0.8, 1.0$, packet loss rate $l_e \in 0.0, 0.05, 0.15, ..., 1.0$, and varying data rate changes $d_e \in [0.0, 9.6]$.

Next, a path through the network can be described as a set of edges $\phi \subseteq E$, where each edge $e = (v_i, v_j) \in \phi \,|\, w(e) = [q_e, l_e, d_e]$ contains a sender node $v_i \in V$, a receiver node $v_j \in V$ and has an associated queue length q_e, packet loss rate l_e and radio link data rate d_e. Therefore each path $\phi \in \Phi$ between a source node S and a target node T has an associated weight as well, which is defined by the weights of each edge $e \in \phi$: $w(\phi) = [Q, L, \sum_{e\in\phi} \sigma^2[d_e]]$. Therefore the optimal network path ϕ^* can be obtained by solving the following control problem with respect to the weight function w:

$$\begin{aligned} &\min w_1 \sum_{e\in\phi} w_2 \cdot (1 - \prod_{e\in\phi} 1 - l_e) + w_3 \cdot \sigma^2[d_e] \\ &\text{subject to:} \\ &\forall e \in \phi : q_e \leq 1.0, 1 - \prod_{e\in\phi} 1 - l_e \leq 1.0, 0.0 \leq d_e \leq 9.6 \end{aligned} \tag{31}$$

The RL aims to solve the control problem described in Eq. 31 by selecting nodes $v \in V$ to create a path ϕ from a specified source node S to a target node T, with the goal of minimizing the path weight $w(\phi)$. It is important to note that a valid path always exists since the graph is created using a Barabási–Albert (BA) model, which guarantees that it is a *connected* graph.

The observation space (or state space) $\mathcal{S}$ can hence be defined by a set of *next neighbors* $V_n \subseteq V$ and the corresponding edge weights $w(e_{ij})$, where e_{ij} is the edge from the current node $v_i \in V$ to a next neighbor node $v_j \in V_n$.

The action space $\mathcal{A} \subseteq E$ contains all edges e that connect current node v_i to one of its direct neighbors. All other actions are defined to be an *illegal action*, that is selecting an edge to a node which is no direct neighbor of the current node, and are discouraged by a negative reward.

After each action $a \in \mathcal{A}$ the agent is presented with a new state $s \in S$ that contains the updated next neighbors and the corresponding edge weights. Additionally it is provided with a reward that encourages the agent to select a robust path with respect to the control problem defined in 31:

$$r(e) := \begin{cases} 100 & \text{reached target node} \\ \frac{1}{w_1} \cdot q_e + \frac{1}{w_2} \cdot (1 - (1 - l_{e_{v_i}}) \cdot (1 - l_e)) + \frac{1}{w_3} \cdot \mathbb{E}[d_e] & \text{s.t. Equation 31} \\ -5 & \text{selecting an illegal action} \end{cases} \tag{32}$$

This means the agent is rewarded with $+100$ for reaching the destination node, -5 for selecting an illegal action or dependent on the selected link if a legal non target node.

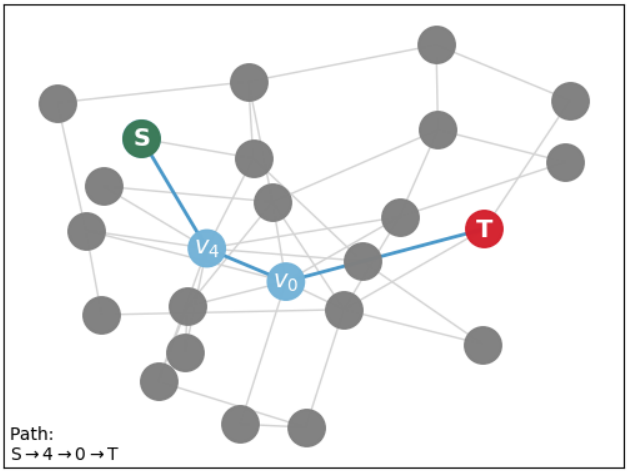


Figure 7. Exemplary graph depicting the abstraction of a MANET defined by the environment in section 5.1. [2]
The applied model finds a robust path from the source node S to target node T using intermediary nodes v_4 and v_0.

5.2 Restrictions

To maintain the integrity of the environment defined in Section 5.1, it is necessary to avoid perturbing certain parameters. Specifically, in the case of the present environment (Section 5.1), it is important not to perturb the current node $v_i \in V$ or the target node T. There are several reasons for this. Firstly, in a real-world scenario, an adversary would likely only have the ability to perturb the edge weights, i.e., the network metrics. Secondly, changing the target node T would significantly impact the experiments described in Section 7. This is because the victim model is designed to not only choose specific edges but also to solve a completely different optimization problem.

6 Methodology

This section introduces the selected set of attacks, each being fundamentally based on the concepts discussed in Section 4.

6.1 Noise constraints

Similar to RL algorithms which face the dilemma of exploration versus exploitation, attacks utilizing adversarial samples are forced to make some kind of trade-off between general performance and detectability. The problem of crafting potent adversarial samples, as described in Section 4.2, becomes increasingly more difficult, the less perturbation is allowed, however this also makes the attack increasingly more difficult to detect.

Huang et al. (2017) [10] provide a selection of noise constraints, that help to monitor said trade-off using the constraining parameter ϵ:

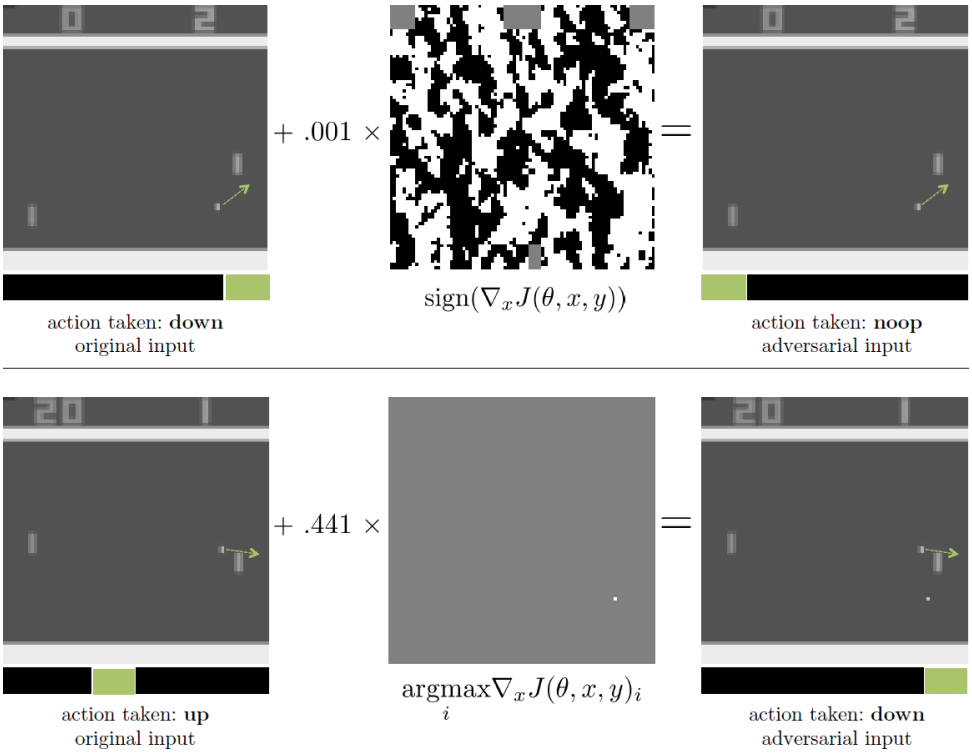


Figure 8. Different noise constraints according to [10]. Small perturbation ($\epsilon = 0.001$) on every feature of the input space (top) versus maximal perturbation ($\epsilon = 0.441$) on a single feature (in this case pixel of the input image) (bottom).

$$\eta = \begin{cases} \epsilon \cdot sign(\Delta_x J(\theta, x, y)) & \text{for constraint } ||\eta||_\infty \leq \epsilon \\ \epsilon \cdot \sqrt{d} \cdot \frac{\Delta_x J(\theta,x,y)}{||\Delta_x J(\theta,x,y)||_2} & \text{for constraint } ||\eta||_2 \leq ||\epsilon \cdot 1_d||_2 \\ \text{maximally perturb highest-impact} & \\ \text{dimensions with budget } \epsilon \cdot d & \text{for constraint } ||\eta||_1 \leq ||\epsilon \cdot 1_d||_1 \end{cases}, \tag{33}$$

with perturbation η, that is the final noise added to the observation to craft an adversarial sample.

Originally FGSM constraints the L_∞-norm, meaning all input features are perturbed by no more than a small amount of noise. Additionally, according to the authors of [10] it might also be desirable to constrain the L_2-norm with respect to the input dimension. Here, each input feature is perturbed by adding a small amount of noise, that scales with the input dimension, in the normalized direction of the gradient. Lastly it may be better to maximally perturb only one input feature, i.e. constraining the L_1-norm. Figure 8 depicts the L_∞ and L_1 noise constraint for the game of Pong. In this example maximally perturbing one feature of the input space corresponds to maximally perturbing a single pixel of the input image by creating a second ball that gives the agent incentive to move the paddle down.

To compare the performance and detectability of an attack depending on how the perturbation η is constrained, this thesis will adapt the L_∞ and L_2 norm constraints as defined in Equation 33.

6.2 Attack Methods

For the following experiments (Section 7) it is important to differentiate between two key aspects: *attack methods* and *attack strategies*. While the latter answers the question *when* to attack, attack methods define the process of adversarial sample crafting and hence *how* to perform an attack.

This includes noise generation, noise application, as well as deciding on a final sample (in the case of iterative attacks). Each of the following methods has access to an RL agent as reference model, which is used to craft effective adversarial samples, however attacks can also be transferred to unfamiliar victim models. It is also absolutely feasible to assume that each attack can train its own reference model as long as it has access to the environment. This essentially eliminates any need to know the inner workings of a potential victim model, as long as it can be assumed to solve the same underlying problem as the attack's reference model. Section 7.2.3 will demonstrate this *transferability* of attacks.

6.2.1 One-step methods

The most naive approach to attack is to simply sample noise once, hold it to the noise constraint and add it to the original observation. This idea will be referred to as *one-step naive approach*.

To generate the required noise, Pattanaik et al. (2017) [14] introduced a *beta distribution* $X \sim B(\alpha, \beta)$, where the probability of pulling $0 \leq x \leq 1$ is given by:

$$\begin{aligned} p(X = x) &= f(x, \alpha, \beta) \\ &= \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}, \end{aligned} \tag{34}$$

with *beta function* $B(\alpha, \beta) := \int_0^1 X^{\alpha-1}(1-X)^{\beta-1}dX$ and its parameters $\alpha, \beta > 0$, the first and second concentration (which themselves can be parameters of the sample crafting method). For the experiments in Section 7, they are set to $\alpha, \beta = 0.5$, leading to non-uniform noise in $[0, 1]$. [15]

To allow the observation parameters to be changed in either direction this noise is then transformed to be in $[-1, 1]$ and held to the aforementioned noise constraints, resulting in the final perturbation η:

$$\begin{gathered} \eta = (2 \cdot B(\alpha, \beta) - 1) \cdot x \\ \text{subject to Equation 33} \end{gathered} \tag{35}$$

Algorithm 1 One-step naive approach attack

Require: state $s \in \mathcal{S}$, reference model π_θ, noise constraint ϵ, parameters of beta distribution α, β

1: $a^* \leftarrow \arg\max \pi_\theta(s)$
2: $a_w \leftarrow \arg\min \pi_\theta(s)$
3: $\eta \sim 2 \cdot B(\alpha, \beta) - 1$
4: $s_{adv} \leftarrow s + s \cdot \epsilon \cdot \eta$ ▷ e.g. L_∞-norm constrain (6.1)
5: **return** s_{adv}

Sampling and adding noise at complete random like this involves the risk of changing the observation in a way, that the best action a^* might become even more probable to be selected than before. While this is unlikely to happen, considering we already assume that the victim model solves the underlying problem sufficiently well, it is undesirable when attacking an RL model nonetheless.

Goodfellow et al. (2014) [5] present the first approach to effectively craft adversarial samples as a direct follow up to the discovery of adversarial vulnerabilities by Szegedy et al. [9] in the same year. As already introduced in Section 4.2.1 and adapted for RL victim models in Section 4.2.2, this method uses the signed gradient of the objective function to generate a noise vector that perturbs the observation such that the best action a^* becomes less probable. FGSM's objective function is hence defined as the cross-entropy loss between the policy's action probability distribution and the distribution that places all weights on a^*:

$$J(s, \pi) := -\sum_{i=1}^{n} p_i \cdot log(\pi(a_i|s)), \tag{36}$$

where $p_i = 1$ if $a_i = a^*$ and 0 otherwise. [10]

The method can be further adjusted to also satisfy the L_2-norm constrained noise condition defined in Equation 33.

Algorithm 2 One-step FGSM attack

Require: state $s \in \mathcal{S}$, reference model π_θ, noise constraint ϵ, parameters of beta distribution α, β

1: $a^* \leftarrow \arg\max \pi_\theta(s)$
2: $a_w \leftarrow \arg\min \pi_\theta(s)$
3: $grad \leftarrow \nabla_s J(s, \pi_\theta)$
4: $grad\_dir \leftarrow \epsilon \cdot sign(grad)$ ▷ e.g. L_∞-norm constrain (6.1)
5: $\eta \sim 2 \cdot B(\alpha, \beta) - 1$
6: $s_{adv} \leftarrow s + s \cdot |\eta| \cdot grad\_dir$
7: **return** s_{adv}

Furthermore Pattanaik et al. (2017) [14] present an alternative to FGSM's objective function, that not only aims to reduce to probability of selecting the best action a^* but also increase

the likelihood of choosing the worst action a_w. Note that both a^* and a_w are defined in the context of greedy action selection and dependent on the reference model's policy π:

$$\begin{aligned} a^* &:= \arg\max \pi(s) \\ a_w &:= \arg\min \pi(s), \end{aligned} \tag{37}$$

again assuming that the policy solves the given RL problem sufficiently well.

This new objective function is defined as the cross-entropy loss between the policy's action probability distribution and an adversarial probability distribution that places all weights on a_w:

$$J(s, \pi) := -\sum_{i=1}^{n} p_i \cdot log(\pi(a_i|s)), \tag{38}$$

where $p_i = 1$ if $a_i = a_w$ and 0 otherwise. Therefore this term equals the negative logarithmic probability of selecting the worst action a_w by following policy π in state s (as can be seen in Equation 39). It can be shown that by minimizing this objective function in fact the probability of selecting a_w is maximized.

$$\begin{aligned} J(s, \pi) &= -\sum_{i=1}^{n} p_i \cdot log(\pi_i) \\ &= -log(\pi(a_w|s)) \\ \Rightarrow \min_s J(s, \pi) &= \min_s - log(\pi(a_w|s)) \\ \Leftrightarrow \min_s J(s, \pi) &= \max_s \pi(a_w|s), \end{aligned} \tag{39}$$

since the logarithmic function is monotonically increasing.

As aforementioned, this new objective function has the benefit of actually maximizing the probability of choosing the worst possible action a_w instead of only minimizing the probability of selecting the best action a^* with respect to the current policy π given state s (as it is achieved by FGSM). The authors of [14] coined the attack method that utilizes this new objective function *gradient-based attack*, however since FGSM also uses the gradient, for better differentiation this thesis will refer to it as *adversarial gradient attack* to emphasize the fact, that this new objective function is the cross-entropy loss between the policy's probability distribution and an adversarial probability distribution.

Algorithm 3 One-step adversarial gradient attack

Require: state $s \in \mathcal{S}$, reference model π_θ, noise constraint ϵ, parameters of beta distribution α, β

$a^* \leftarrow \arg\max \pi_\theta(s)$
$a_w \leftarrow \arg\min \pi_\theta(s)$
$grad \leftarrow \nabla_s J(s, \pi_\theta)$
$grad\_dir \leftarrow \epsilon \cdot \sqrt{d} \cdot \frac{grad}{||grad||_2}$ ▷ e.g. L_2-norm constrain (6.1)
$\eta \sim 2 \cdot B(\alpha, \beta) - 1$
$s_{adv} \leftarrow s + s \cdot |\eta| \cdot grad\_dir$
return s_{adv}

6.2.2 Iterative methods

Up until this point, each of the introduced attack methods crafts the adversarial sample s_{adv} that corresponds to state s_t by sampling noise only once per time step. However, since the noise is randomly sampled from the beta distribution, the probability of crafting a potent adversarial sample with only one attempt is rather slim.

Thus, the performance of the aforementioned attack methods increases significantly when sampling noise multiple times per time step and selecting the best resulting sample, that is the one which reduced the victim model's reward the most. Therefore so called *iterative attacks* sample noise and craft the corresponding potential adversarial samples s_{adv_i} a number of n times for each time step t. Then, out of this set of potential candidates, the one is selected, that proofed most potent with respect to some criterion. This thesis uses three such criteria: the probability of the potential adversarial action in the original probability distribution, the new probability of the best action a^* and the new probability of the worst action a_w. Note that *new probability* refers to the probability of the corresponding action in the probability distribution that is provided by the policy, given the potential adversarial sample: $\pi(s_{adv_i})$.

Firstly, Pattanaik et al. (2017) [14] select adversarial samples based on the Q-values of the corresponding adversarial actions, that is the action that would be chosen based on the reference model's policy given the adversarial sample as observation. Since the authors are working with Q-Learning based algorithms, while this thesis focuses on Actor-Critic algorithms, this criterion would translate as the probability of selecting the adversarial action in the original probability distribution of the unperturbed observation. This means the least probable the action was in the original distribution, the less desirable to perform that action according to the reference model's policy.

Next, the adversarial sample can also be chosen based on the new probabilities of a^* and a_w, meaning the sample that minimizes the probability of choosing the best action (as is achieved by FGSM's objective function) or the one that maximizes the probability of selecting the worst action (as is achieved by the objective function motivated in [14]).

6.2.3 Stochastic Gradient Descent methods

Instead of crafting n complete unrelated adversarial samples for each time step, it appears more rational to start of with a random sample and improve it iteratively. According to [14],

Algorithm 4 Iterative naive attack

Require: state $s \in \mathcal{S}$, reference model π_θ, number of iterations n, noise constraint ϵ, parameters of beta distribution α, β

$a^* \leftarrow \arg\max \pi_\theta(s)$
$a_w \leftarrow \arg\min \pi_\theta(s)$
$s_{adv} \leftarrow s$
for $i = 1$ to n **do**
    $\eta_i \sim 2 \cdot B(\alpha, \beta) - 1$
    $s_{adv_i} \leftarrow s + s \cdot \epsilon \cdot \eta_i$ ▷ e.g. L_∞-norm constrain (6.1)
    if $criterion(s_{adv_i}) > criterion(s_{adv})$ **then** ▷ according to section 6.2.2
        $s_{adv} \leftarrow s_{adv_i}$
    end if
end for
return s_{adv}

this idea can be achieved using Stochastic Gradient Descent (SGD), which is a method to iteratively optimize an objective function and is often used for ML tasks in general. Since SGD requires an objective function to optimize in the first place, this method is only applicable to gradient-based methods, i.e. FGSM and adversarial gradient attacks.

SGD differs from the previously defined iterative gradient approaches only in the inputs that are used to compute the gradient. While iterative gradient methods always compute the gradient based on the original state s, SGD uses the last potential adversarial sample $s_{adv_{i-1}}$. This way the adversarial sample is iteratively improved in the gradient direction, which also eliminates the need of criteria, since each sample is a strict improvement to the previous one.

Algorithm 5 SGD FGSM attack

Require: state $s \in \mathcal{S}$, reference model π_θ, number of iterations n, noise constraint ϵ, parameters of beta distribution α, β

$s_{adv} \leftarrow s$
for $i = 1$ to n **do**
    $grad \leftarrow \nabla_{s_{adv}} J(s_{adv}, \pi_\theta)$
    $grad\_dir \leftarrow \epsilon \cdot sign(grad)$ ▷ e.g. L_∞-norm constrain (6.1)
    $\eta_i \sim 2 \cdot B(\alpha, \beta) - 1$
    $s_{adv} \leftarrow s + s \cdot |\eta| \cdot grad\_dir$
end for
return s_{adv}

6.2.4 Momentum boosted adversarial attacks

Dong et al. (2017) [16] introduce a new concept of boosting adversarial attacks with momentum. To ensure faster and more stable conversion to the optimal adversarial sample, these *momentum boosted adversarial attacks* accumulate a velocity vector in the gradient direction.

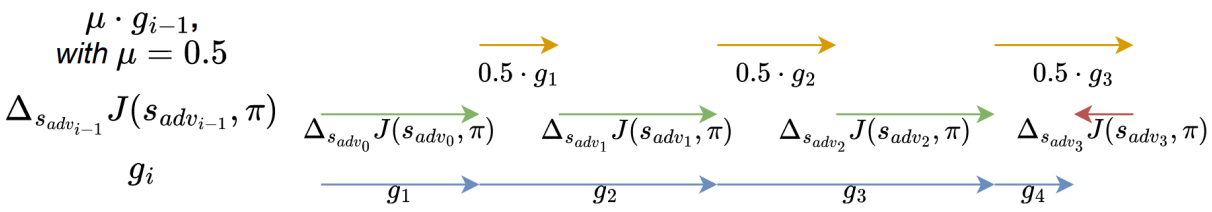


Figure 9. Illustration of momentum boosting.
Each new gradient is scaled with a fraction of the previous gradients. This way the algorithm converges faster while also avoiding to be inhibited by outliers, as these will not be powerful enough to immediately break the momentum in the "correct" direction.
In this case μ is set to 0.5.

This idea can be applied to any iterative gradient descent method - in the case of this thesis, the two SGD variants of FGSM and the adversarial gradient method.

Formally, the gradient g_i for each iteration $i \leq n$ can be defined as:

$$g_i = \mu \cdot g_{i-1} + \Delta_{s_{adv_{i-1}}} J(s_{adv_{i-1}}, \pi), \tag{40}$$

where $\Delta_{s_{adv_{i-1}}} J(s_{adv_{i-1}}, \pi)$ is the gradient of the objective function with respect to the last potential sample $s_{adv_{i-1}}$. Furthermore the momentum coefficient $0 \leq \mu \leq 1$ defines the degree by which each past gradient is considered for the new gradients. Lastly, g_i is used to compute the gradient direction in which the new noise is applied. Figure 9 illustrates how this velocity vector is used to build momentum in the gradient direction.

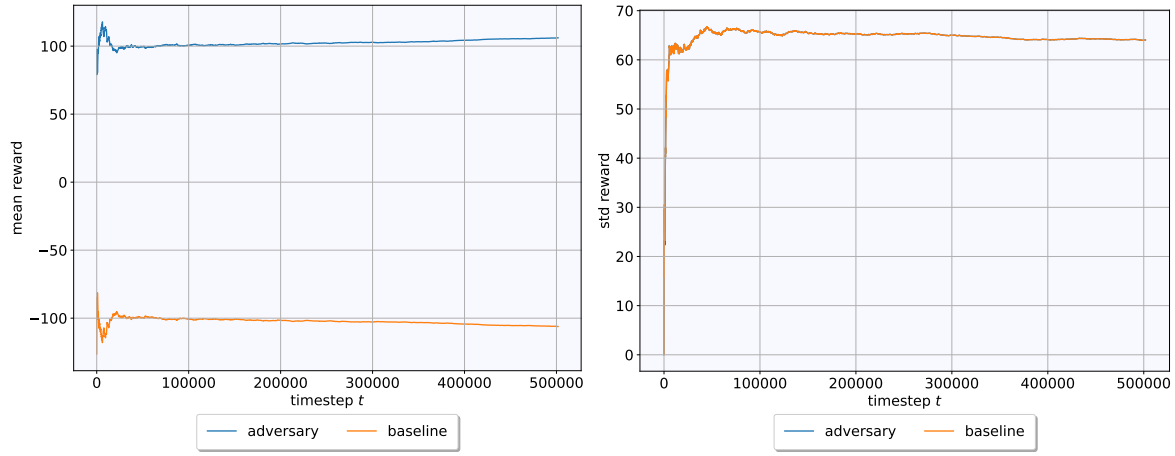
Algorithm 6 Momentum boosted adversarial gradient attack

Require: state $s \in \mathcal{S}$, reference model π_θ, number of iterations n, noise constraint ϵ, parameters of beta distribution α, β

$s_{adv} \leftarrow s$
$grad_0 \leftarrow 0$
for $i = 1$ to n **do**
  $grad_i \leftarrow \mu \cdot grad_{i-1} + \nabla_{s_{adv}} J(s_{adv}, \pi_\theta)$
  $grad\_dir \leftarrow \epsilon \cdot \sqrt{d} \cdot \frac{grad_i}{||grad_i||_2}$ ▷ e.g. L_2-norm constrain (6.1)
  $\eta_i \sim 2 \cdot B(\alpha, \beta) - 1$
  $s_{adv} \leftarrow s + s \cdot |\eta|$
end for
return s_{adv}

6.2.5 Adversarial policy attacks

Gleave et al. (2019) [17] train adversarial agents to play against victim agents in a two-player Markov game. This adversary has the only goal to minimize its opponents reward, which is reflected by the corresponding environments in use.



(a) Mean reward. **(b)** Standard deviation reward.

Figure 10. Illustration of the reward of the adversary and its reference model throughout the training phase.

Hürten et al. (2022) [2] build on this concept and utilize an *adversarial policy* as a result of adversarial training in the context of MANETs, using a modified version of the environment introduced in Section 5.1. This adversarial agent can *learn* the reference model's decision making and compute noise that results in an effective adversarial sample.

While the adversary works with the same observation space $\mathcal{S}$ as the reference model, its action space $\mathcal{A}$ has the same dimension as $\mathcal{S}$ and consists of a tuple containing noise for each parameter of the observation. Furthermore the adversary receives an inverted reward of its reference agent:

$$r_{adv} := (-1) \cdot r. \tag{41}$$

Hence by maximizing this reward, an agent trained on the adversary environment produces noise that results in adversarial samples which cause the victim model to maximize the control problem defined in Equation 31. Figure 10 visualizes the adversarial training experience by illustrating the adversarial reward r_{adv} versus the reference model's reward during training.

While this method requires a training period on the adversary environment, it avoids expensive computation during application, that is each observation s is simply handed to the adversarial policy which in return provides $\eta = \pi_{adv}(s)$ to craft the adversarial sample s_{adv}. Therefore the experiments in Section 7 should prove adversarial policy methods to have much faster runtime than iterative approaches.

Next, this concept of adversarial policies can be extended to include the cumulative perturbation into the reward function. This creates incentive to avoid large amount of noise if possible and aims to decrease susceptibility to detection. Hence this *bound adversarial policy attack* modifies the reward as follows:

Algorithm 7 Adversarial policy attack

Require: state $s \in \mathcal{S}$, adversarial policy π_{adv}, noise constraint ϵ

1: $\eta \leftarrow \pi_{adv}(s)$
2: $s_{adv} \leftarrow s + s \cdot \epsilon \cdot \eta$ ▷ e.g. L_∞-norm constrain (6.1)
3: **return** s_{adv}

$$r_{adv_{bound}} := (-1) \cdot r - \rho \cdot \sum \pi_{adv_{bound}}(s), \tag{42}$$

where $\pi_{adv_{bound}}(s)$ provides the adversary's action as a tuple containing noise for each parameter of the observation like before and $0 \leq \rho \leq 1$ is a scaling coefficient that determines to which degree the cumulative noise is considered for the adversarial reward $r_{adv_{bound}}$.

6.3 Attack Strategies

Given this series of methods to craft adversarial samples, the question of *how* to attack is well answered, leaving only the question of *when* to attack. Assuming that attacks at different time steps have varying effectiveness and that attacking more often leads to higher detectability, it is only natural to consider strategies that reduce the overall number of attacks by limiting them only to those time steps that yield the most destructive potential.

Lin et al. (2017) [18] provide an approach that limits attacks only to those time steps, in which changing the victim agent's action will with a high probability lead to a decreased return. This *strategically-timed attack* uses a preference function c to determine ideal attack timings:

$$\begin{aligned} c(s_t) &= \max_{a_t} \pi(s_t \,|\, a_t) - \min_{a_t} \pi(s_t \,|\, a_t) \\ &= \pi(s_t \,|\, a^*) - \pi(s_t \,|\, a_w) \end{aligned} \tag{43}$$

This preference function will consequently be large if the agent highly prefers a certain action a^* for state s_t in time step t over the least desirable action a_w and small if the agent is rather indifferent regarding the possible actions.

It is now feasible to limit attacks to those time steps (and their corresponding states s_t) for which the preference function $c(s_t)$ reaches some threshold β. Generally a large β should result in fewer attacked time steps while a lower β results in an increased number of attacks, while $\beta = 0$ is referred to as *uniform attack*, where each time step is attacked uniformly. Figure 11 demonstrates this strategy for the game of pong. The state is only perturbed in the case of s_{84} since in that case the agent highly prefers a specific action to defend the ball.

With the correct configuration for the given problem, this will ensure that attacks are performed when necessary while simultaneously reducing the overall amount of attacks to decrease the risk of detection. [18]

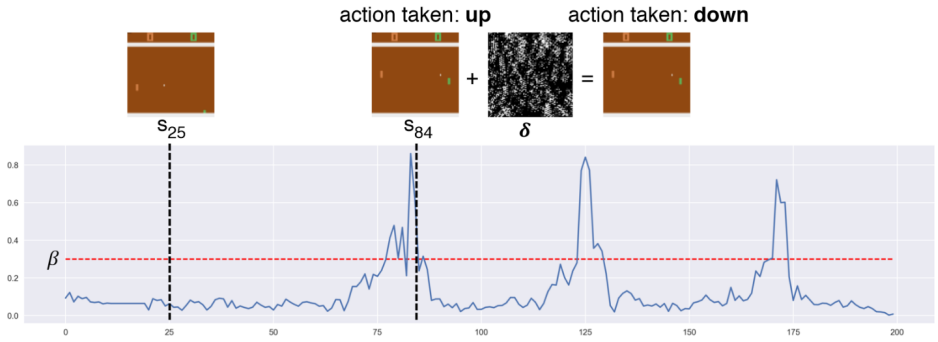


Figure 11. Strategically-timed attack illustrated using the game of pong [18]. At time step $t = 25$ the ball is in the middle of the playing field, both vertically and horizontally, hence the agent has no strong preference for a certain action. However in time step $t = 84$ the ball is close to the border of the playing field resulting in an immediate threat to the agent. Consequently the action *up* is strongly preferred and perturbing the state to manipulate the agent to choose *down* instead leads to a decrease of return. In this case the preference function exceeds the threshold $\beta = 0.35$. Note that the authors of [18] denote the noise as δ.

7 Results

This section discusses quantitative results from a set of experiments. Each attack is evaluated and compared to each other based on a number of metrics, such as *effectiveness*, *transferability*, *runtime* and *detectability*.

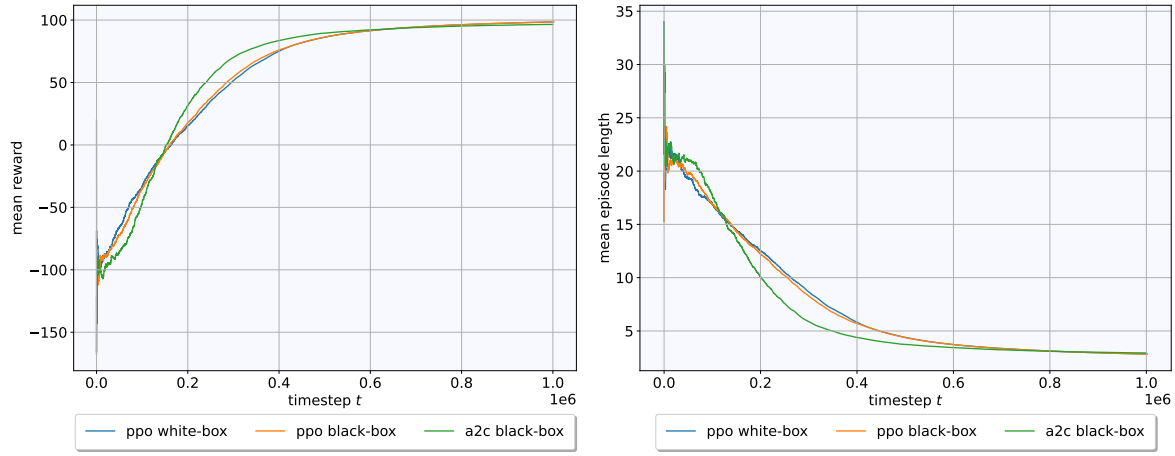
7.1 Configuration

All experiments are run on the same environment (an implementation of the model discussed in Section 5.1). Non adversarial policy attacks sample noise using the beta distribution according to Equation 34.

Furthermore the attacks are evaluated on implementations of three actor-critic models: two PPO agents (one that is also used as reference model and one independent model) and an A2C agent, each trained over one million time steps. Figure 12 graphs the algorithm's performance throughout the training process.

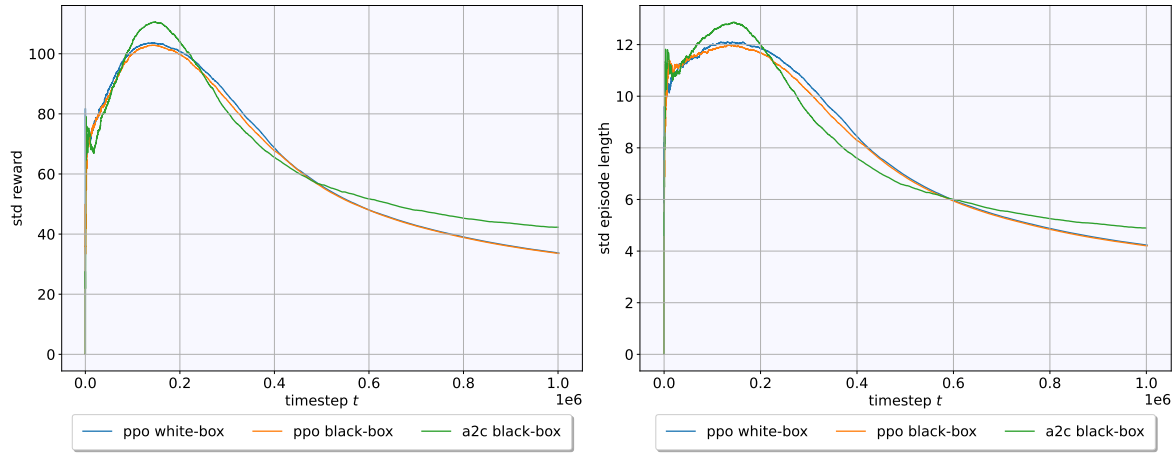
Also, all attacks typically implement the L_∞-norm noise constraint, unless explicitly stated otherwise. Additionally, by default iterative methods use the probability of selecting the new adversarial action in the original probability distribution given by π as criterion to select the most effective adversarial sample.

To receive statistically relevant results, each configuration is tested over a number of episodes to evaluate the average outcome. Figure 13 shows the behavior of the standard deviation of the episode rewards, the episode length and the number of attacks per episode (i.e. the agent chooses a different action on the adversarial sample than on the original observation) as exemplary metrics to get an idea of how many episodes are required for the mean computation. The data suggests that for the chosen environment, RL algorithms and



(a) Mean reward.

(b) Mean episode length.



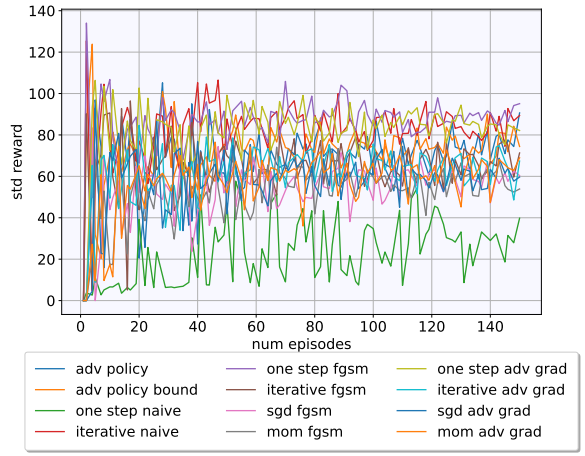
(c) Standard deviation reward.

(d) Standard deviation episode length.

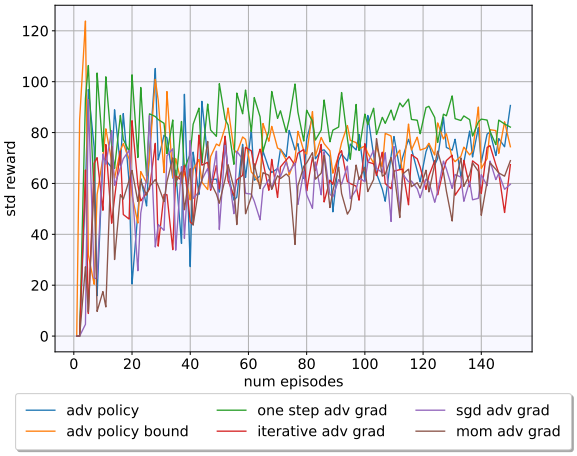
Figure 12. Illustration of the three model's mean reward and episode length throughout the training phase (1 million time steps).

attacks a number of 50 episodes seems to be sufficient, as the deviation of the resulting data does not decrease any further, even for larger episode averages.

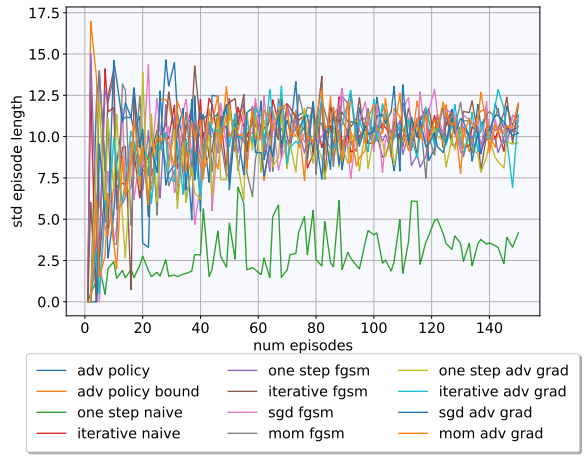
As discussed in Section 6.2.2, iterative methods (as well as their SGD and momentum boosted derivatives) sample noise a number of n times for each state s_t. To be able to compare them to the adversarial policy attacks in a meaningful manner, it is necessary to Figure out the minimum number of sample iterations required to achieve sufficient performance. Figure 14 shows the results of an extensive test with different iteration numbers. The resulting data suggests that generally sampling noise over 15 iterations achieves maximum performance for these specific environment-RL-model combinations - measured as the mean reward reduction and number of changed actions.



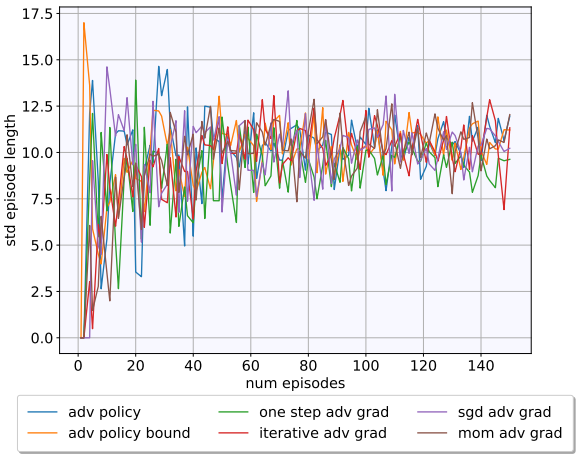
(a) Standard deviation of the reward.



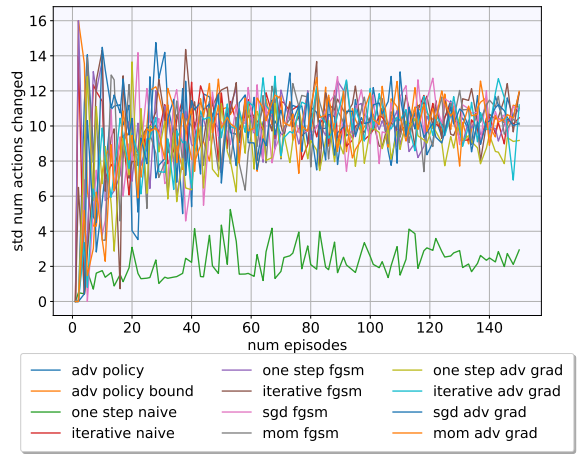
(b) Standard deviation of the reward.



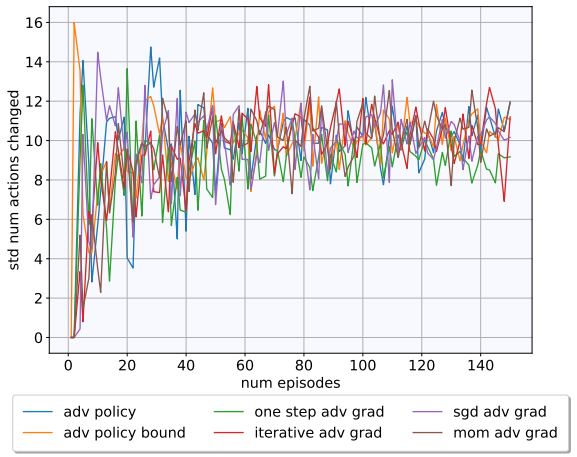
(c) Standard deviation of the episode length.



(d) Standard deviation of the episode length.

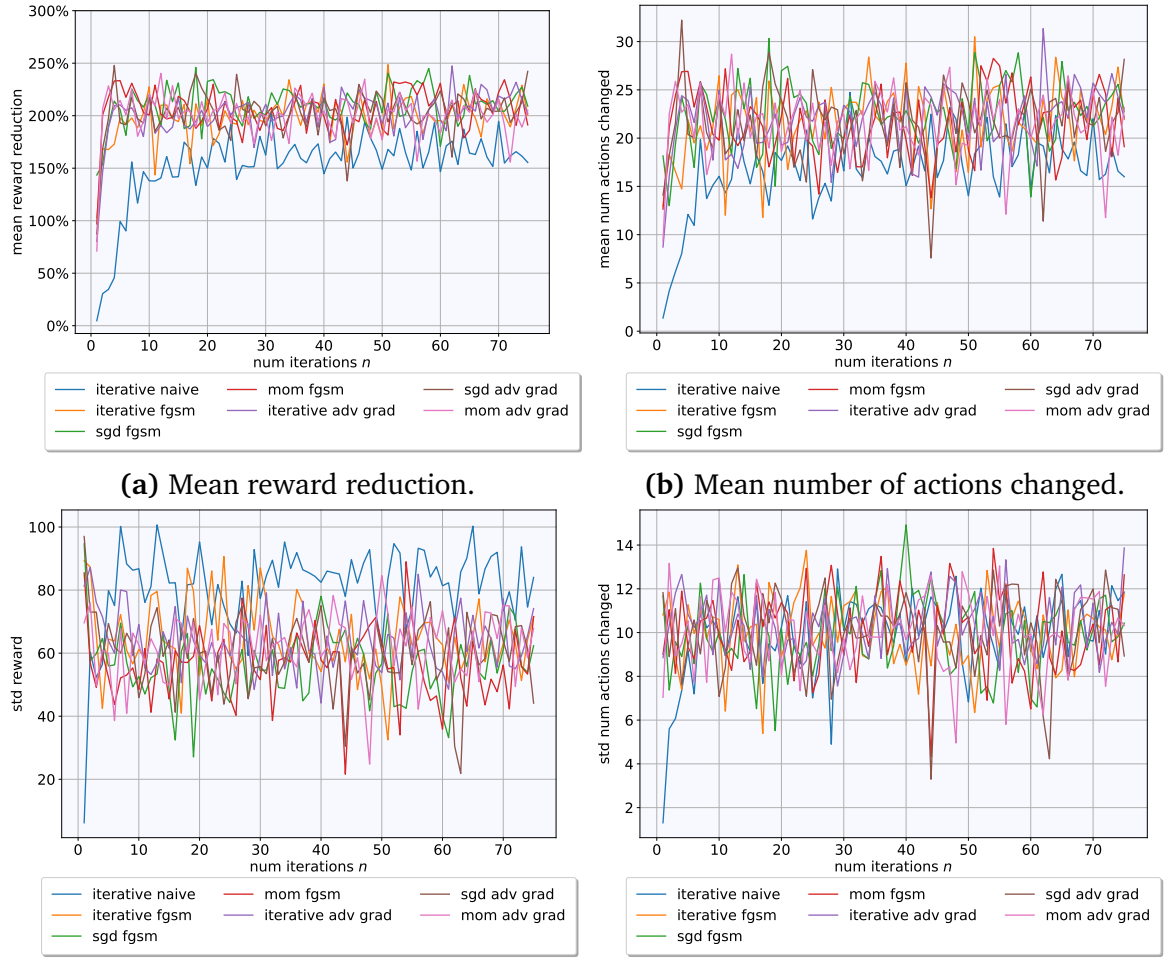


(e) Standard deviation of the number of actions changed.



(f) Standard deviation of the number of actions changed.

Figure 13. Development of the standard deviation with respect to the total number of episodes (right-hand side compares best method of each type).



(a) Mean reward reduction.

(b) Mean number of actions changed.

(c) Std of reward reduction.

(d) Std of number of actions changed.

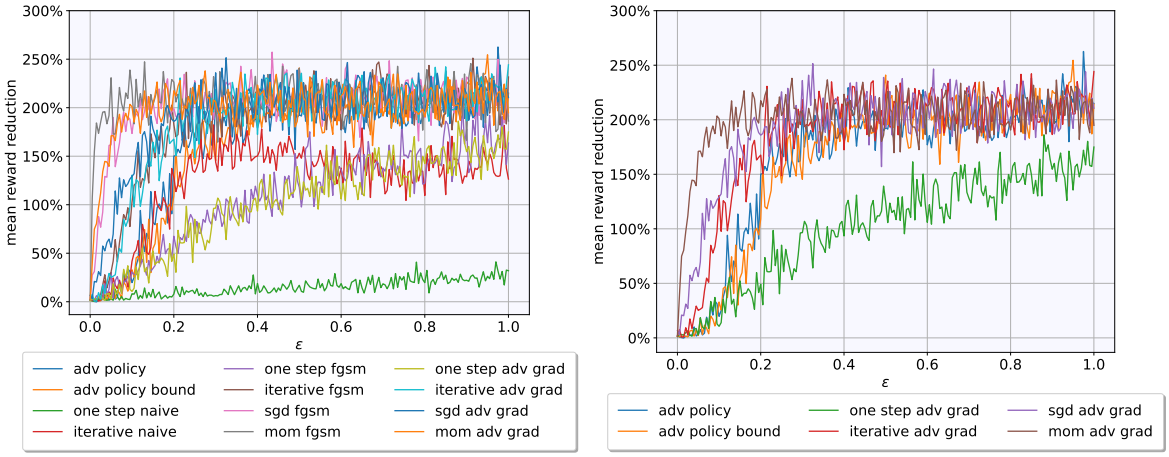
Figure 14. Mean reward reduction (a) and number of actions changed (b) for different numbers of iterations of noise sampling per state s_t.

7.2 Evaluation

7.2.1 Effectiveness

To quantify the effectiveness of each attack, experiments were run on a set of $\epsilon \in [0, 1]$ resulting in the 50 episode averages illustrated by Figure 15.

Figures 16a, 16c, 16b and 16d demonstrate the performance of the naive, FGSM and adversarial gradient approach for each of the types (one-step, iterative, SGD and momentum boosted) with respect to the mean reward reduction. Generally both gradient variants appear to be similarly powerful. While FGSM performs slightly better for smaller values for ϵ, both approaches reach their maximum potential around the same ϵ configuration. Unsurprisingly, the iterative naive attack method performs worse than its gradient-based competitors, although the gap seems to be significantly smaller for iterative naive attacks than for the one-step naive approach.



(a) Mean reward reduction (all attacks). **(b)** Mean reward reduction (comparison).

Figure 15. Illustration of the experiment stated in section 7.2.1. (a) takes all methods into account while (b) compares the best method of each type similar to figure 14. Results are estimated over 50 episodes with varying ϵ.

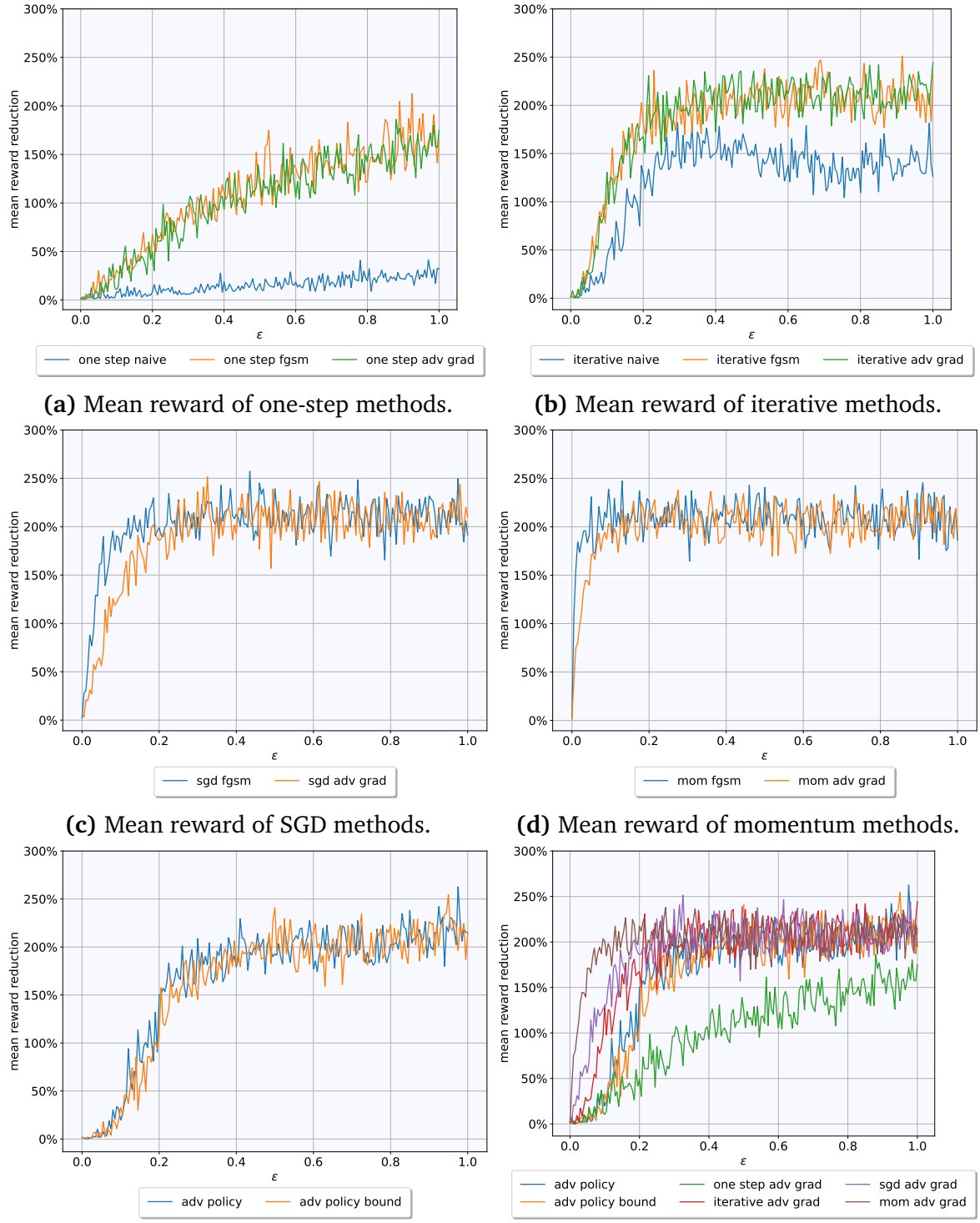
Figure 16e illustrates the mean reward reduction of the two adversarial policy methods. The results suggest that both approaches perform similarly well and achieve maximal reward reduction at around $\epsilon = 0.3$.

To compare the different types of attacks, the most potent approach combinations are graphed in Figure 16f. As expected one-step attacks generally perform worse with mean reward reduction that linearly increases in ϵ and reaches its maximum of about 150% at $\epsilon = 1.0$. Iterative (gradient-based) attacks seem to perform marginally better than adversarial policy attacks for $\epsilon < 0.3$, at which point both attack types reach a maximum reward reduction of 200%. While SGD reaches its maximum potential at $\epsilon = 0.2$, momentum boosted attacks benefit from their velocity vector and reach the reward reduction of 200% even faster, at around $\epsilon = 0.1$.

Generally the experiment suggests that other than one-step attacks every attack method can potentially decrease the victim model's performance dramatically by up to 200% on average, however some attack methods seem to require a more loose noise constraint than others. Figure 17 demonstrates how the victim model is manipulated in selecting illegal actions when planning the network path.

The next experiment tests the performances given different criteria for sample selecting and is depicted in Figure 18. Surprisingly the corresponding data suggests, that the only feasible criterion by which to select a potential adversarial sample for iterative attacks is the probability of the adversarial action a_{adv} in the original distribution. Using the probability of the a^* or a_w as criterion seems to be a poor strategy to select the best adversarial sample, as attacks are unable to reduce the victim model's mean reward regardless of the ϵ configuration.

Momentum boosted attacks have an additional parameter μ, that determines the degree by which the previous gradients influence the computation of the current gradient g_t. An experiment on different values for $\mu \in [0, 1]$ suggests that the specific configuration of μ does increase the mean perturbation, however does not increase the mean reward reduction.



(a) Mean reward of one-step methods.

(b) Mean reward of iterative methods.

(c) Mean reward of SGD methods.

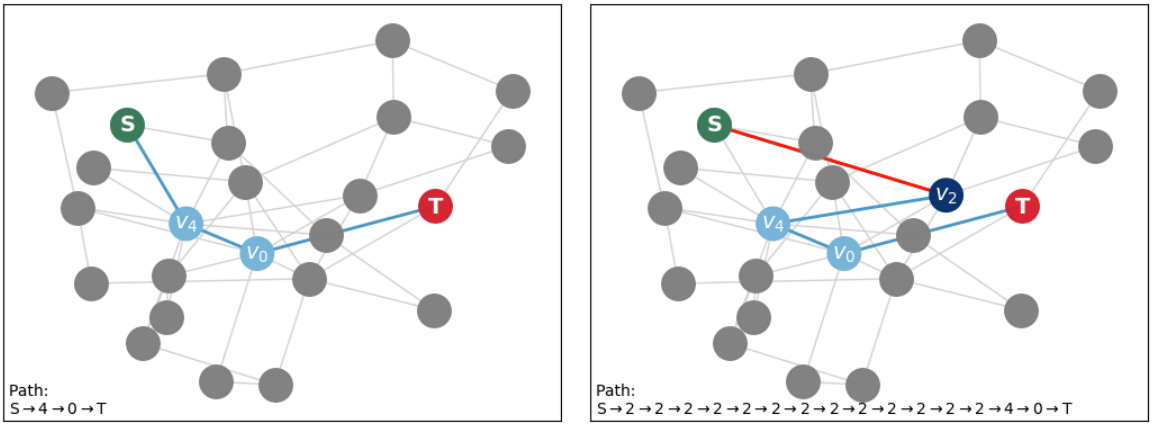
(d) Mean reward of momentum methods.

(e) Mean reward of adv policy methods.

(f) Mean reward (comparison).

Figure 16. Divides the results shown in figure 15 into categories for each type of attack method. Results are estimated over 50 episodes with varying ϵ.

This is due to the fact that the aforementioned test was run on a fixed $\epsilon = 0.35$. However Figure 15b already suggests that the main difference between momentum boosted attacks



(a) Legal path, planned by the baseline without attacks.

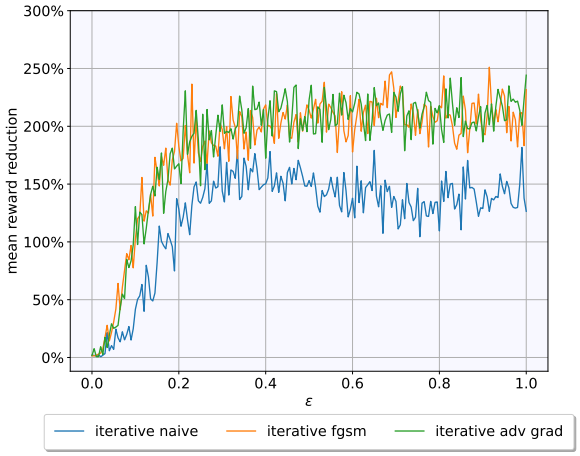
(b) Illegal path, planned by the attacked victim model.

Figure 17. Network path planned by baseline and attacked victim model. [2] Attacks were performed uniformly using the iterative adversarial gradient method. Action legality and frequency is represented by the color choice and intensity.

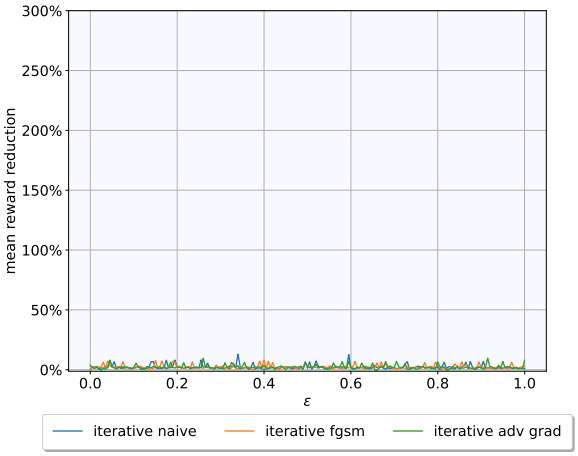
and SGD attacks is highlighted for smaller values for $\epsilon \in [0, 2]$.

Section 6.1 motivates two different noise constraints. Figure 20 depicts the performance of each attack method type for both, the L_∞- and L_2-norm constraint. It is observable, that the L_2-norm constrained attacks seem to reach their maximum performance faster (for smaller ϵ values), however on average also apply more noise per episode. For example the momentum boosted adversarial gradient attack reaches the maximum reward reduction of around 200% for $\epsilon = 0.05$ while using the L_2-norm constrained noise, whereas applying L_∞-norm constrained noise requires a larger $\epsilon = 0.1$. However the mean cumulative perturbation seems to be smaller for $\epsilon < 0.25$ when using the L_∞-norm constraint and larger for $\epsilon > 0.25$. Surprisingly however, the adversarial policy attack seems to perform significantly better with the L_2-norm, even outperforming the previously best method - momentum boosted adversarial gradient attack. Even so, this new noise constraint has the added mean perturbation reach its maximum of around 300 for a comparatively small $\epsilon = 0.2$, while it previously scaled linearly in ϵ, only ever reaching about 200 on average. Hence the L_2-norm constraint seems to be the better option for adversarial policy attacks if the increased perturbation and therefore detectability is tolerable. For the remaining attack method types the choice of either noise constraint seems acceptable and the decision should be based on the minor trade-off between performance and detectability.

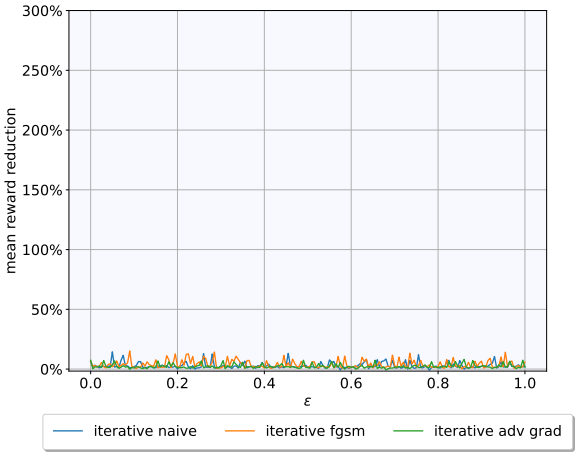
Furthermore attack methods, that utilize the beta distribution to sample noise, can either sample the same noise for all parameter of the observation space uniformly or sample noise for each parameter individually. These two options are compared in Figure 21, which suggests that the performance (i.e. mean reward reduction) experiences a slight increase for larger ϵ-values, especially for one-step methods, when sampling noise non-uniformly. This implies that sampling noise non-uniformly is preferred, when it is only possible to sample noise once per time step, potentially due to limitations on runtime, as will be discussed in Section 7.2.4.



(a) Select s_{adv_i}, that had the lowest probability in the original distribution.



(b) Select s_{adv_i}, that resulted in the lowest new probability for a^*.



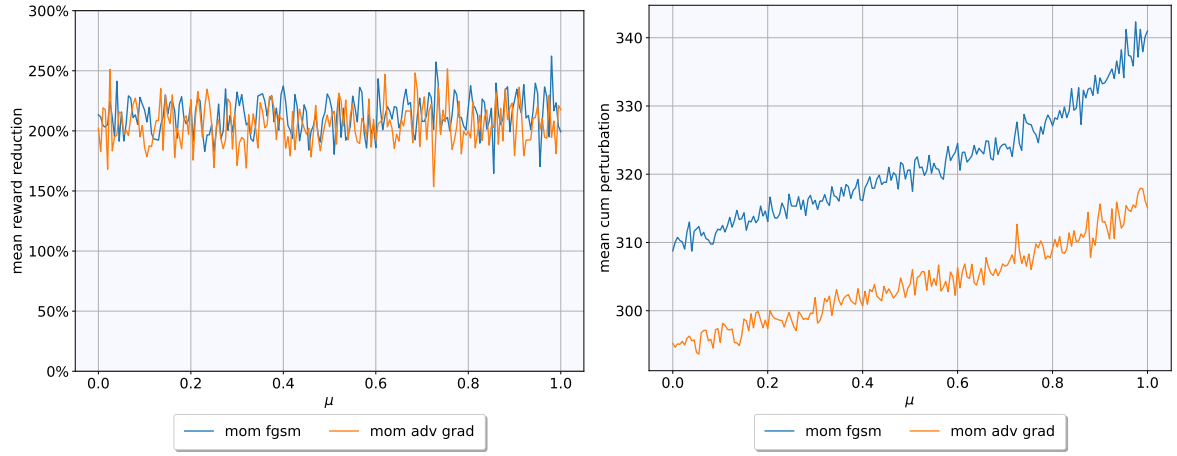
(c) Select s_{adv_i}, that resulted in the highest new probability for a_w.

Figure 18. Comparison of mean reward reduction based on the criterion to choose adversarial samples.

7.2.2 Detectability

Detectability describes the level of ease with which attacks are perceivable to either human observation or more often artificial detectors. Assuming that in general attacks are easier to detect, the more noise is applied to the observations, it is logical to compare the amount of mean cumulative perturbation of the varying attack methods, meaning the total absolute noise added on average in each time step.

It is to be expected, that the noise constrain parameter ϵ has the highest influence on the cumulative perturbation and hence the attack's detectability. For larger ϵ, the added noise is less constrained, leading to larger perturbation and therefore an increased susceptibility to detection, as depicted by Figure 22. Note that only the data for the best method out of each category is shown, in a similar way to Figure 15b. While the mean cumulative perturbation increases linearly in ϵ for one-step, iterative and adversarial policy attacks, SGD



(a) Mean reward reduction.

(b) Mean cumulative perturbation.

Figure 19. Performance of the two momentum boosted attacks with different $\mu \in [0, 1]$.

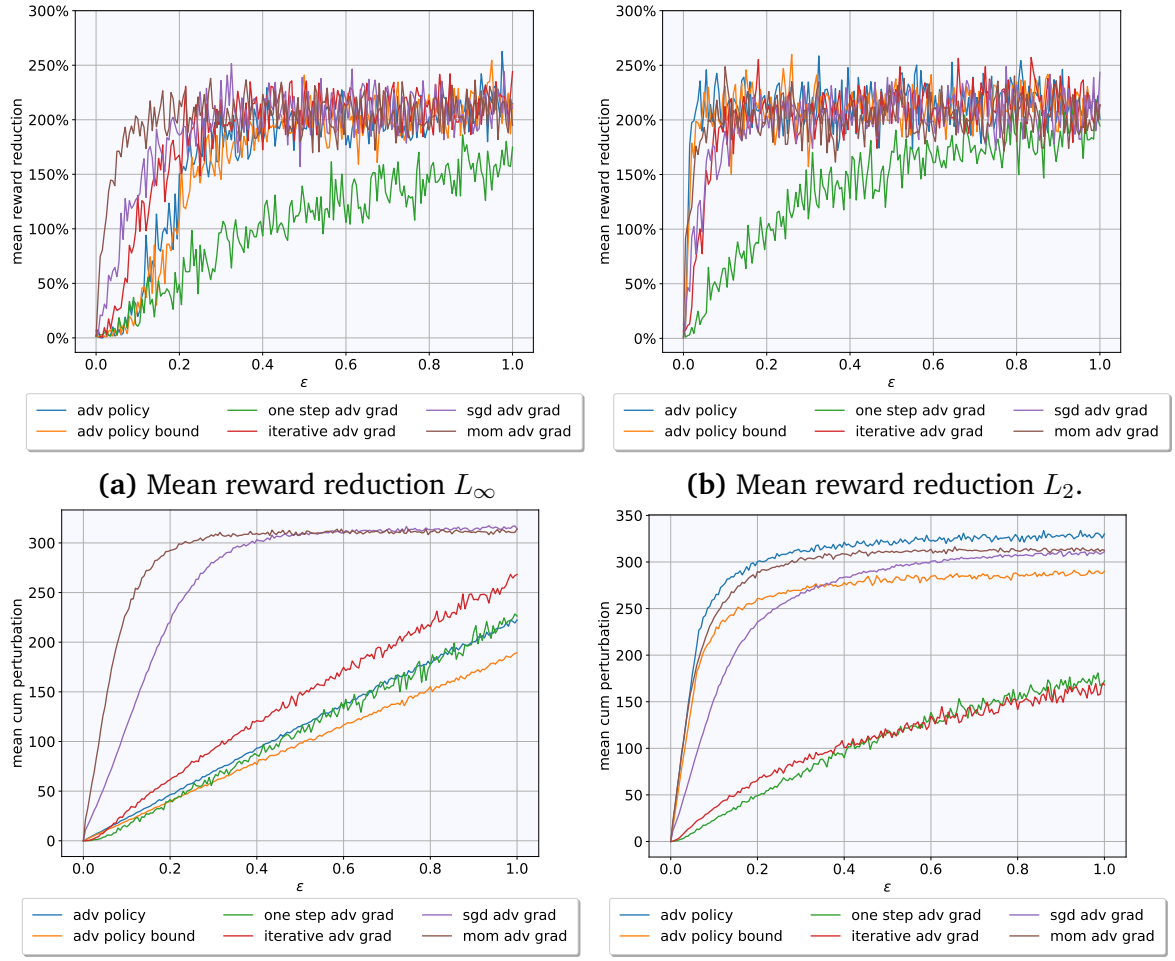
and momentum boosted attacks reach the maximum perturbation at around $\epsilon = 0.2$ and $\epsilon = 0.4$ respectively, at which point the attack is very susceptible to detection.

Figure 23 divides the results into categories for the different attack method types for better clarity. Noticeably, performance seems to not necessarily increase proportionally to the added perturbation, as is the case for one-step methods. For example, adversarial policy attacks reach their maximum mean reward reduction for $\epsilon = 0.3$, while the corresponding mean cumulative perturbation increases further to about three times that amount for $\epsilon = 1$.

For most attacks, a good trade-off between performance and detectability seems to be around $\epsilon = 0.25$ to $\epsilon = 0.35$.

In the previous experiments all attacks were performed as uniform attacks ($\beta = 0$), however by increasing β the victim model is only attacked if the reference model has a significant preference for the best action a^* over selecting the worst action a_w as described in Section 6.3. Consequently less attacks should be performed throughout the episode and only those time steps, that are necessary to decrease the victim model's reward. The data, that resulted from an experiment run on different configurations for $\beta \in [0, 1]$ utilizing the most potent attack method of each type as before, is illustrated in Figure 24. For the chosen environment, as defined in Section 5.1, the victim model is very confident in its choice of actions, hence the preference function is very close to 1 (a^* is preferred to a_w almost by factor 1), which is why for the most part, the attack's performance is unaffected by β. However for a large $\beta > 0.9$, both the mean reward reduction and the number of performed attacks decreases slightly and finally hit 0 for $\beta = 1$, as is to be expected. Consequentially the data suggests, that for the given RL problem β does not seem to be a powerful tool to effectively decrease the detectability of attacks.

Another attempt at limiting noise is to integrate the added perturbation into the reward function of an adversarial policy, as motivated in Section 6.2.5. Here, the reward is computed as the inverted reward of the reference model minus the absolute added perturbation, scaled by some factor ρ. Figure 25 illustrates the bound adversarial attack's performance for



(a) Mean reward reduction L_∞

(b) Mean reward reduction L_2.

(c) Mean cumulative perturbation L_∞.

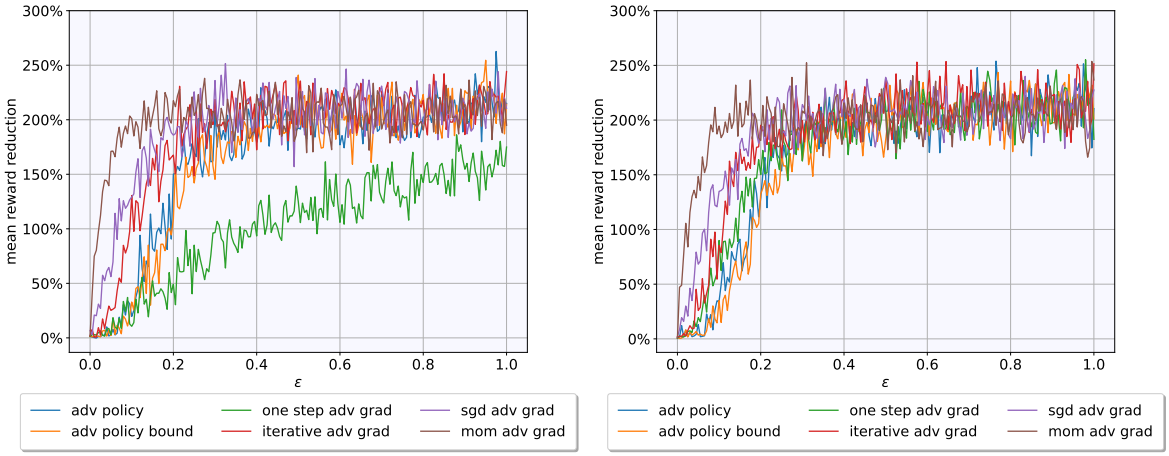
(d) Mean cumulative perturbation L_2.

Figure 20. Each attack type's performance with respect to the L_∞-norm and L_2-norm constrained noise.

different ρ configurations compared to the unbound variant as baseline. The bound adversarial policy was retrained for each value of ρ over 100 000 time steps. It is apparent that this approach manages to reduce the mean amount of noise added during each time step, as the adversarial agent learns to only apply noise if necessary. While the fact that the bound adversarial policy already starts with less mean reward reduction and perturbation compared to its unbound counterpart, even for $\rho = 0$ can generally be explained by the fewer number of training time steps (the two adversarial policies used in the previous experiments were each trained on 500 000 time steps), Figure 10 demonstrated that adversaries can reach a good performance after a relatively short training phase. Hence the experiment nonetheless shows the tendency of a decreasing mean cumulative perturbation for increasing values of ρ, while still retaining a stable performance.

7.2.3 Transferability

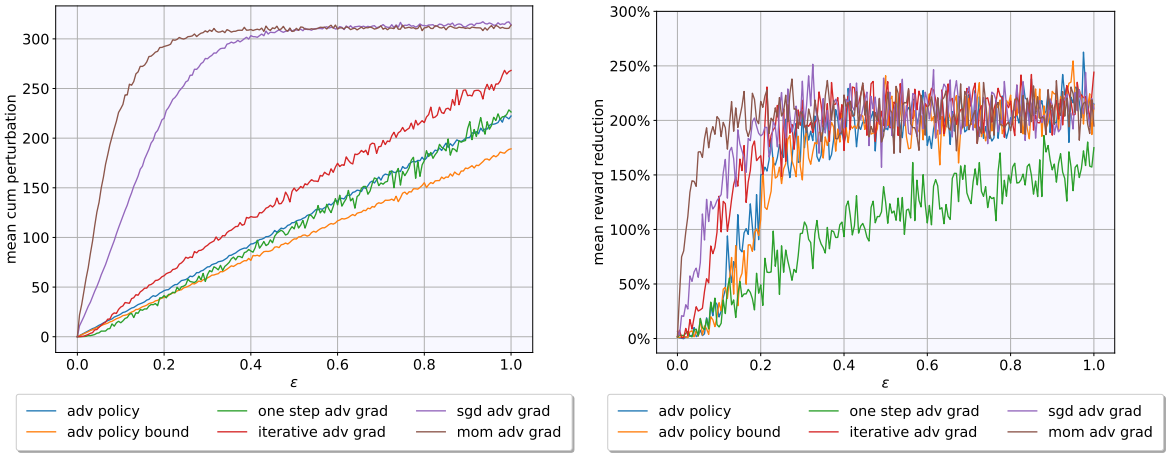
The term transferability describes how well each attack can be extended to RL models, other than its reference model. Note that all iterative attacks at least require a model to evalu-



(a) Mean reward uniform noise.

(b) Mean reward non-uniform noise.

Figure 21. Each attack type's performance with respect to whether noise was sampled uniformly for all parameter of the observation space or specific to each.



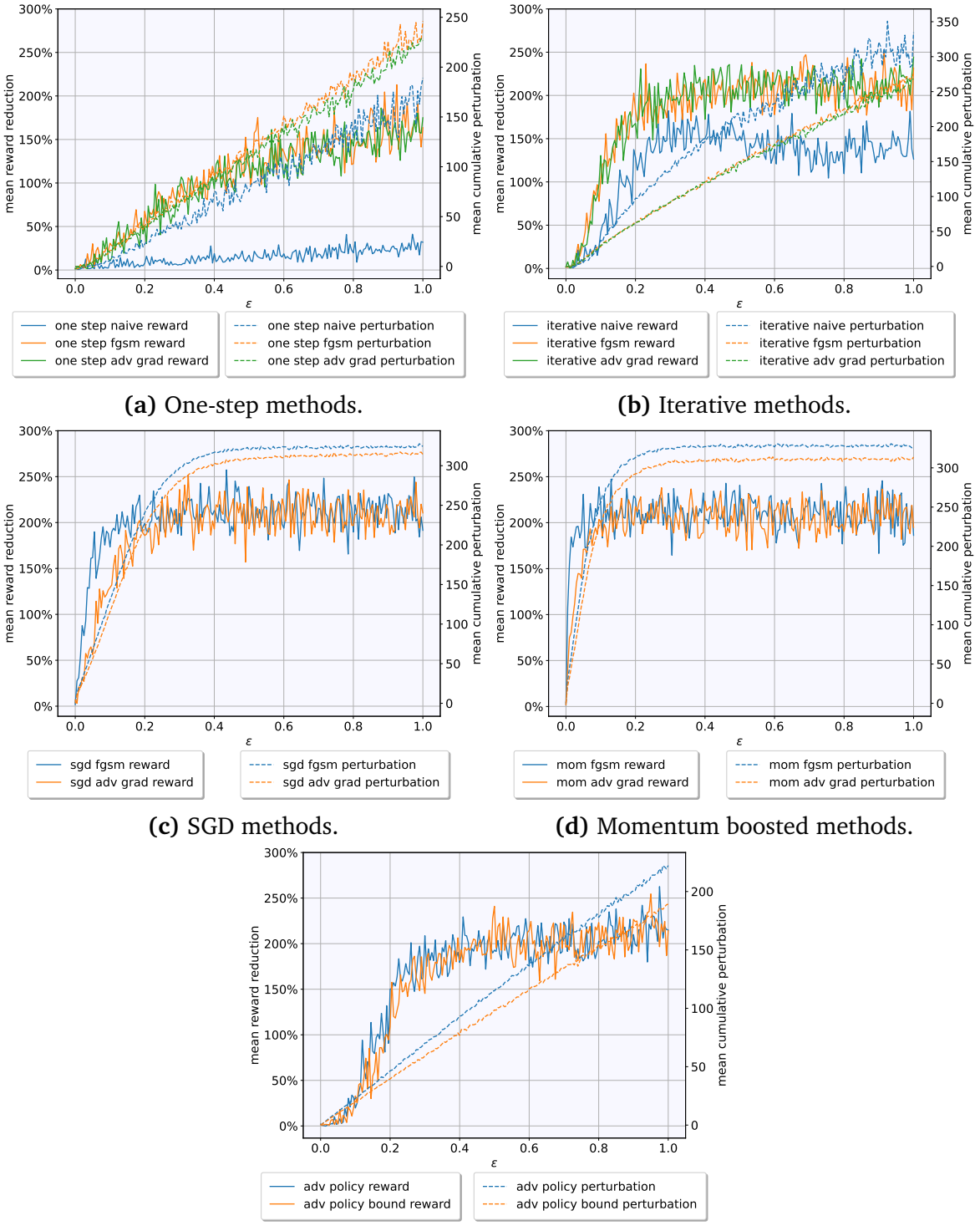
(a) Mean cumulative perturbation of all attacks.

(b) Mean reward reduction of each attack type.

Figure 22. Illustrates the experiment stated in section 7.2.2.

ate each potential adversarial sample s_{adv_i}. Gradient methods additionally need access to a model's policy to compute the gradients. Adversarial policy attacks on the other hand are explicitly trained on a given RL model and learn how to choose their own actions based on the latter's policy.

Attacks targeting these agents are so called *white-box attacks*, meaning the attacker has access to the inner workings of the victim model, whereas real life applications will usually require the attack to be performed in a *black-box* manner. This means the attack has neither access to the model nor information of the victim model's policy, etc. In the case of adversarial policy attacks, an argument can be made that these are actually somewhere between pure white-box and pure black-box attacks, which in literature is often referred to as *semi-black-box attacks*. This is due to the fact that these attack methods do require access to some model, however not necessarily its inner workings such as its policy. It is sufficient to simply invoke



(a) One-step methods.

(b) Iterative methods.

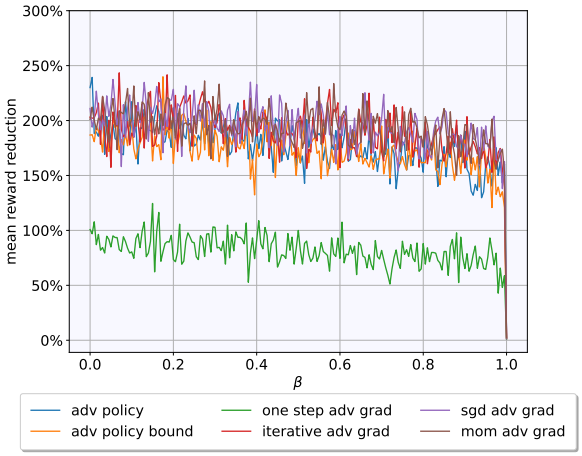
(c) SGD methods.

(d) Momentum boosted methods.

(e) Adversarial policy methods.

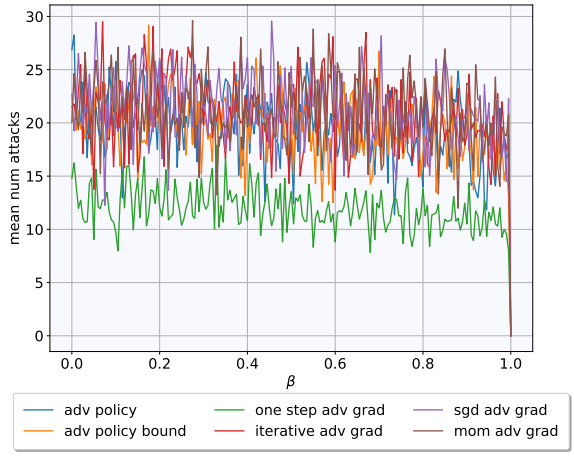
Figure 23. Divides the results shown in figure 22 into categories for each type of attack method and compares mean reward reduction with mean cumulative perturbation.

the reference model's policy implicitly by letting it select actions based on the adversarial policy's samples.

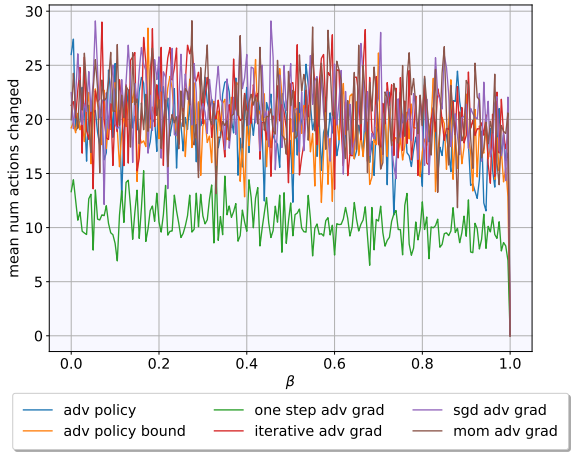


(a) Mean reward reduction.

(b) Mean cumulative perturbation.

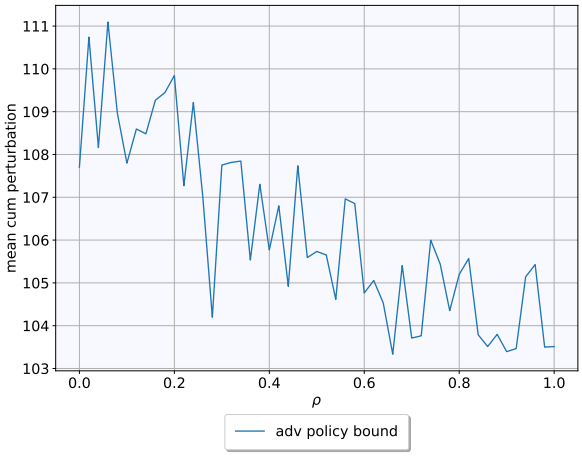


(c) Mean number of attacks per episode.

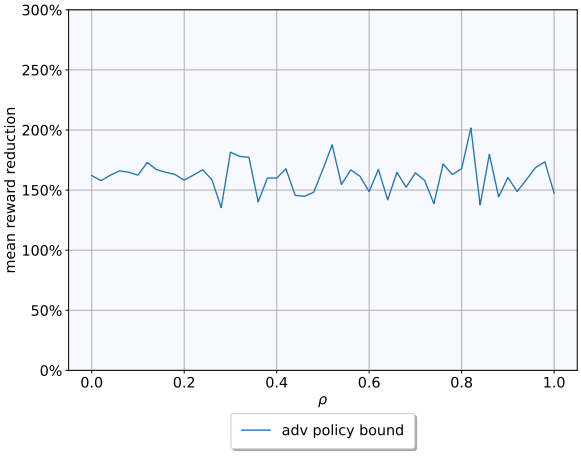


(d) Mean number of actions changed.

Figure 24. Performance of each attack type for different $\beta \in [0, 1]$.



(a) Mean perturbation.



(b) Mean reward reduction.

Figure 25. Trade-off between mean reward reduction and cumulative perturbation for an increasing ρ. ϵ is fixed to 0.35.

However generally it is desirable to perform these attacks as black-box attacks, meaning no information of the actual victim model can be assumed. Indeed this is not an issue because for RL problems it is generally safe to assume that models, which solve the same underlying problem work in a similar enough way so that an attack can achieve similar results on models other than its reference model.

In the following, each attack method was applied as a uniform attack ($\beta = 0$) on its reference model (a PPO agent), as well as two unknown models, i.e. another PPO agent and an A2C agent, the latter two being real black-box attacks. All three models underwent one million time steps of training (Figure 12).

Figure 26 models this experiment on the metrics mean reward (26a), mean number of attacks per episode (26b), mean cumulative perturbation (26c) and mean episode length (26d). While it is reasonable to assume an advantage for white-box attacks compared to black-box attacks, as a matter of fact, this is not the case. All attacks only experience a marginal performance hit, if at all (26a).

Figures 26b, 26c and 26d also suggest that all attacks perform about the same on each of the three models, except for SGD adversarial gradient attack, which on average attacks about 5 additional times on the white-box attack compared to the black-box attacks, leading to a mean episode length, that is also about 5 time steps longer. This however does neither seem to correlate with SGD attacks in general, nor is it recognizable with other adversarial gradient attacks, hence the variation in performance is likely due to the normal deviation in results on the chosen environment as was already observed in Section 7.2.1.

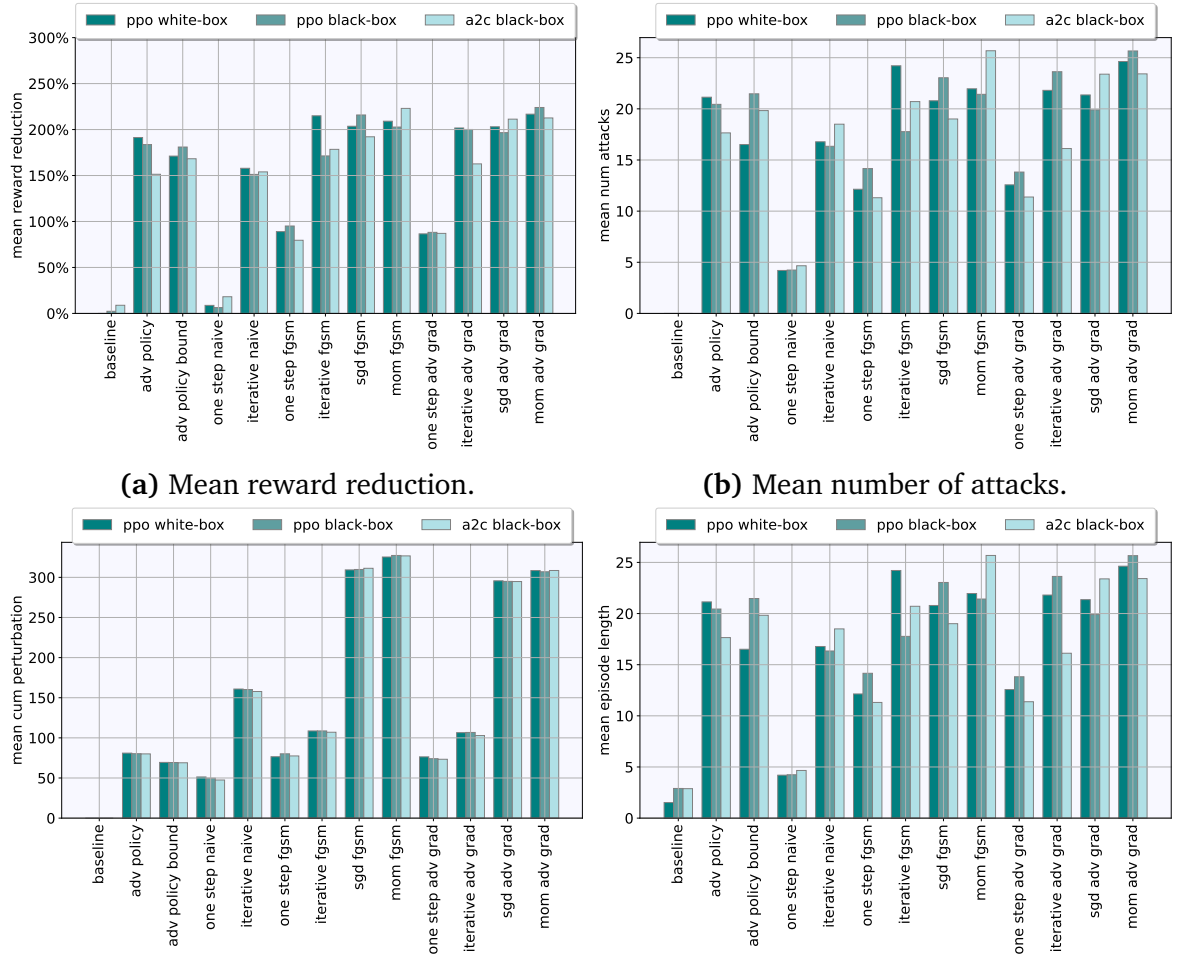
Also note that for some attacks the black-box attack seems to outperform its white-box attack counterpart, however this is very likely again due to the general variance of the results. As a matter of fact, all variations between the three victim models are potentially caused by the simplicity of the environment defined in Section 5.1, as will be discussed in Section 8.

7.2.4 Runtime

As already stated in Section 6.2.5, one definite advantage of adversarial policy attacks over the other presented attack methods is the fact that after completing their training process, they require minimal computing time of $O(1)$. Furthermore due to their iterative nature, iterative attacks, as well as their derivatives SGD and momentum boosted attacks, need to repeat the computation multiple times per time step, leading to a potentially long runtime of $O(n)$.

To make profound conclusions, in the following experiments, each attack's runtime is compared relative to the baseline, therefore taking the model's runtime into account while also decreasing the dependency on hardware specifics.

Figure 27a illustrates the mean runtime (relative to the baseline) of each iterative attack method with respect to the number of iterations n per time step. As expected the mean runtime increases significantly in n, e.g. the chosen configuration for the previous experiments ($n = 15$) extends the baseline's runtime by a factor 1000. Plain iterative attacks require a shorter runtime overall since the gradient is only computed once (or not at all, as in the



(a) Mean reward reduction.

(b) Mean number of attacks.

(c) Mean cumulative perturbation.

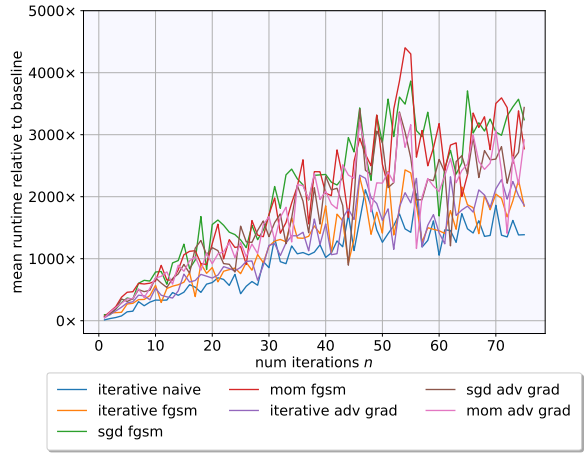
(d) Mean episode length.

Figure 26. Illustration of the experiment stated in section 7.2.3 comparing performance on the three victim agents. In this case results are estimated over 500 episodes with $\epsilon = 0.35$ and $\beta = 0$ (uniform attack).

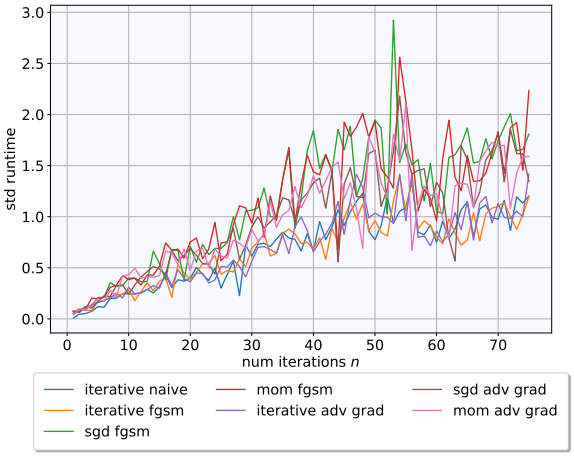
case of the iterative naive attack), whereas SGD and momentum boosted attacks compute the gradient of the last sample $s_{adv_{i-1}}$ in each iteration.

As already stated in Section 7.1, $n = 15$ seems to be a good trade-off between performance and runtime, as exemplarily depicted for the naive iterative approach and the momentum boosted variant of the adversarial gradient method in Figure 27c.

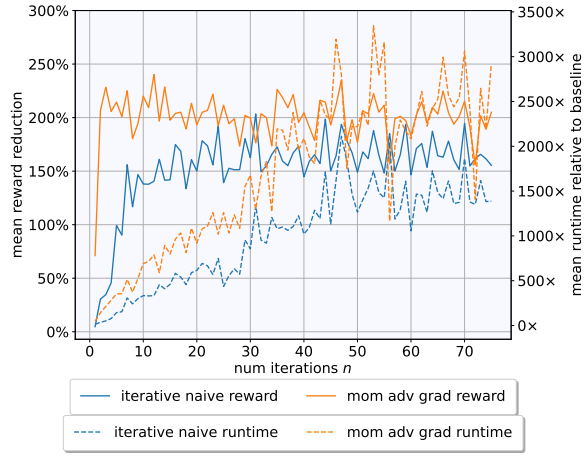
Figure 27d places the previous data in the context of all attacks, hence once again the data was gathered with a fixed $n = 15$ for iterative attack methods. Again, as expected iterative-based methods have the longest runtime relative to the base model. Overall, if runtime is a concern, adversarial policy attacks seem to deliver the best trade-off between performance and runtime.



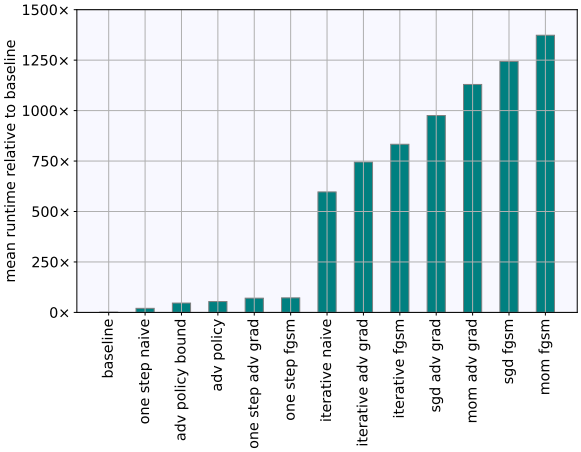
(a) Relative mean runtime of iterative-based methods for different numbers of iterations.



(b) Relative std runtime of iterative-based methods for different numbers of iterations.



(c) Relative mean runtime versus mean reward reduction for different numbers of iterations.



(d) Relative mean runtime of all methods ($n = 15$ for iterative-based methods).

Figure 27. Illustration of the experiments stated in section 7.2.4.

8 Conclusion

This thesis examined the potential of adversarial attacks in the context of RL applications in tactical networks. It extends previous research, while also highlighting some differences between a set of introduced attack methods.

First and foremost, the experiments in Section 7 prove RL agents, deployed to supervise tactical networks, to be indeed susceptible to these sophisticated attacks. A malicious adversary can intercept observations and modify them in a way, that significantly reduces the victim agent's reward and consequentially manipulate the latter to select - for instance - unstable connections for packet transmission. Therefore one has to be aware of these observation based attack vectors when trying to apply RL methods in security relevant fields.

Furthermore Section 7 highlights key differences between attacks with respect to a set of metrics. Most attack methods seem to effectively impact the victim model's performance,

however more complex methods, such as adversarial policy attacks or gradient-based approaches, proved to be most potent.

Additionally the use of these adversarial policies significantly reduce the computation time for sample crafting at runtime at the cost of an extensive training phase before application.

At the same time it is possible to limit the degree of perturbation, that is applied to each observation and thereby the risk of detection, using more restricting attack strategies (e.g. strategically timed attacks) and noise constraints or integrating the noise magnitude into an adversarial policy's objective function.

Although each presented attack uses information about its called reference model to effectively craft adversarial samples, it was proven, that these attacks can indeed be transferred to independent victim models as black-box attacks, as long as it can be reasonably assumed, that the latter solves the same underlying problem in a similar manner as the reference model. Hence, the required reference model can also be acquired through training by the adversary itself, therefore reducing the extent of required knowledge to just the environment.

Further research on a more complex network simulation is necessary to further investigate the advantage of momentum boosted attacks compared to SGD methods, as well as more complex attack approaches, that are currently researched in general, as these were not realistically applicable, due to the chosen environment's limitations.

Finally it will be interesting to see these adversarial attack vectors being applied during the learning phase, to potentially train more robust RL models, as well as investigation done on different learning approaches and their vulnerability against adversarial samples. Also, this thesis focuses on the mean cumulative perturbation as metric for detectability, but it would be intriguing to see tests with actual detectors.

In conclusion, RL methods seem to be an interesting addition to research of tactical networks and MANETs in general and will likely benefit the field, once these potential security risks are sufficiently addressed.

References

[1] S. Kaviani, B. Ryu, E. Ahmed, K. A. Larson, A. Le, A. Yahja, and J. H. Kim, "Robust and scalable routing with multi-agent deep reinforcement learning for manets," 2021.

[2] J. F. Loevenich, J. Bode, T. Hürten, L. Liberto, F. Spelter, P. H. Rettore, and R. R. F. Lopes, "Adversarial attacks against reinforcement learning based tactical networks: A case study," in *MILCOM 2022-2022 IEEE Military Communications Conference (MILCOM)*, pp. 986–992, IEEE, 2022.

[3] A. L. Samuel, "Some studies in machine learning using the game of checkers," 1959.

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning, second edition: An Introduction*. MIT Press, 2018.

[5] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014.

[6] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems* (S. Solla, T. Leen, and K. Müller, eds.), vol. 12, MIT Press, 1999.

[7] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," 2017.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[9] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2014.

[10] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial attacks on neural network policies," 2017.

[11] S. Basagni, M. Conti, S. Giordano, and I. Stojmenović, *Mobile ad hoc networking*, vol. 461. Wiley Online Library, 2004.

[12] J. Hoebeke, I. Moerman, B. Dhoedt, and P. Demeester, "An overview of mobile ad hoc networks: applications and challenges," *Journal-Communications Network*, vol. 3, no. 3, pp. 60–66, 2004.

[13] J. F. Loevenich, A. Sergeev, P. H. L. Rettore, and R. R. F. Lopes, "An intelligent model to quantify the robustness of tactical systems to unplanned link disconnections," in *2022 18th International Conference on the Design of Reliable Communication Networks (DRCN)*, pp. 1–8, 2022.

[14] A. Pattanaik, Z. Tang, S. Liu, G. Bommannan, and G. Chowdhary, "Robust deep reinforcement learning with adversarial attacks," 2017.

[15] M. V. Jambunathan, "Some properties of beta and gamma distributions," *The Annals of Mathematical Statistics*, vol. 25, no. 2, pp. 401–405, 1954.

[16] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting adversarial attacks with momentum," 2017.

[17] A. Gleave, M. Dennis, C. Wild, N. Kant, S. Levine, and S. Russell, "Adversarial policies: Attacking deep reinforcement learning," 2019.

[18] Y.-C. Lin, Z.-W. Hong, Y.-H. Liao, M.-L. Shih, M.-Y. Liu, and M. Sun, "Tactics of adversarial attack on deep reinforcement learning agents," 2017.

9 Appendix A

Algorithm 8 Iterative FGSM attack

Require: state $s \in \mathcal{S}$, reference model π_θ, number of iterations n, noise constraint ϵ, parameters of beta distribution α, β

$a^* \leftarrow \arg\max \pi_\theta(s)$
$a_w \leftarrow \arg\min \pi_\theta(s)$
$grad \leftarrow \nabla_s J(s, \pi_\theta)$
$grad\_dir \leftarrow \epsilon \cdot sign(grad)$ ▷ e.g. L_∞-norm constrain (6.1)
$s_{adv} \leftarrow s$
for $i = 1$ to n **do**
    $\eta_i \sim 2 \cdot B(\alpha, \beta) - 1$
    $s_{adv_i} \leftarrow s + s \cdot |\eta_i| \cdot grad\_dir$
    if $criterion(s_{adv_i}) > criterion(s_{adv})$ **then** ▷ according to section 6.2.2
        $s_{adv} \leftarrow s_{adv_i}$
    end if
end for
return s_{adv}

Algorithm 9 Iterative adversarial gradient attack

Require: state $s \in \mathcal{S}$, reference model π_θ, number of iterations n, noise constraint ϵ, parameters of beta distribution α, β

$a^* \leftarrow \arg\max \pi_\theta(s)$
$a_w \leftarrow \arg\min \pi_\theta(s)$
$grad \leftarrow \nabla_s J(s, \pi_\theta)$
$grad\_dir \leftarrow \epsilon \cdot \sqrt{d} \cdot \frac{grad}{||grad||_2}$ ▷ e.g. L_2-norm constrain (6.1)
$s_{adv} \leftarrow s$
for $i = 1$ to n **do**
    $\eta_i \sim 2 \cdot B(\alpha, \beta) - 1$
    $s_{adv_i} \leftarrow s + s \cdot |\eta_i| \cdot grad\_dir$
    if $criterion(s_{adv_i}) > criterion(s_{adv})$ **then** ▷ according to section 6.2.2
        $s_{adv} \leftarrow s_{adv_i}$
    end if
end for
return s_{adv}

Algorithm 10 SGD adversarial gradient attack

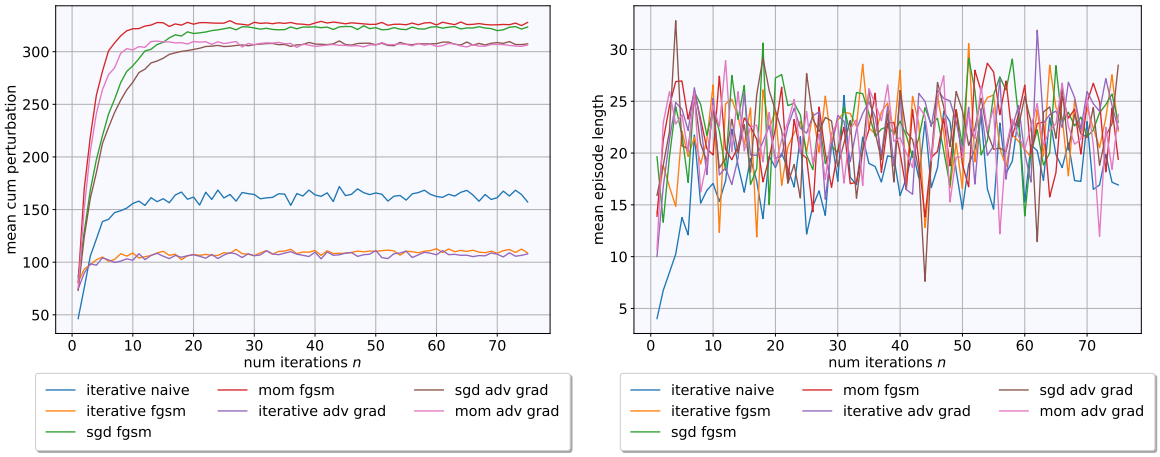
Require: state $s \in \mathcal{S}$, reference model π_θ, number of iterations n, noise constraint ϵ, parameters of beta distribution α, β

$s_{adv} \leftarrow s$
for $i = 1$ to n **do**
  $grad \leftarrow \nabla_{s_{adv}} J(s_{adv}, \pi_\theta)$
  $grad\_dir \leftarrow \epsilon \cdot \sqrt{d} \cdot \frac{grad}{||grad||_2}$ ▷ e.g. L_2-norm constrain (6.1)
  $\eta_i \sim 2 \cdot B(\alpha, \beta) - 1$
  $s_{adv} \leftarrow s + s \cdot |\eta| \cdot grad\_dir$
end for
return s_{adv}

Algorithm 11 Momentum boosted FGSM attack

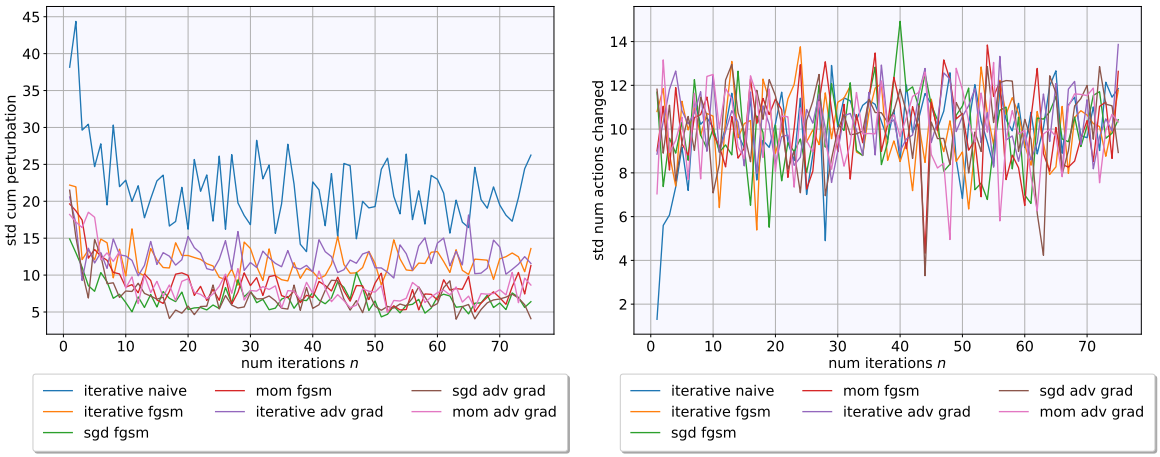
Require: state $s \in \mathcal{S}$, reference model π_θ, number of iterations n, noise constraint ϵ, parameters of beta distribution α, β

$s_{adv} \leftarrow s$
$grad_0 \leftarrow 0$
for $i = 1$ to n **do**
  $grad_i \leftarrow \mu \cdot grad_{i-1} + \nabla_{s_{adv}} J(s_{adv}, \pi_\theta)$
  $grad\_dir \leftarrow \epsilon \cdot sign(grad)$ ▷ e.g. L_∞-norm constrain (6.1)
  $\eta_i \sim 2 \cdot B(\alpha, \beta) - 1$
  $s_{adv} \leftarrow s + s \cdot |\eta|$
end for
return s_{adv}



(a) Mean cumulative perturbation.

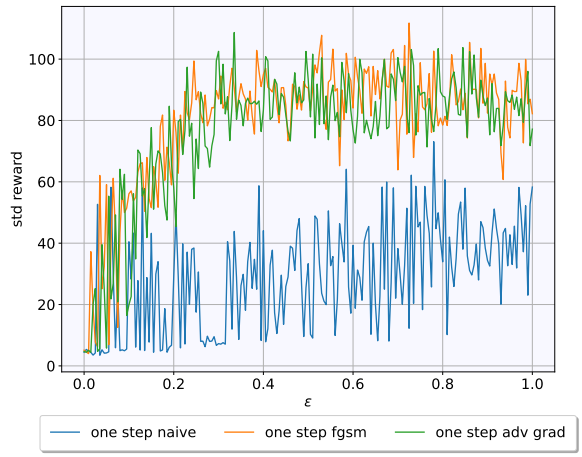
(b) Mean episode length.



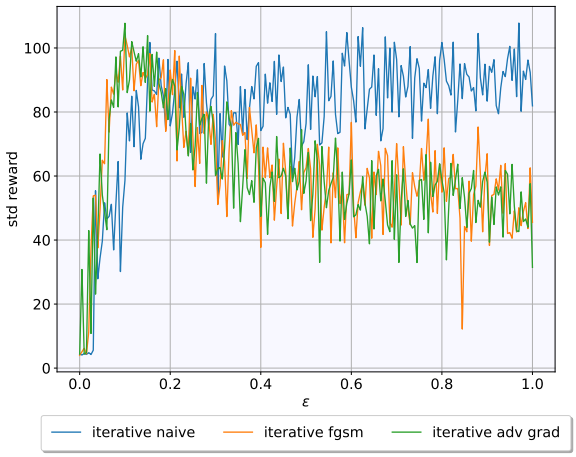
(c) Std cumulative perturbation.

(d) Std episode length.

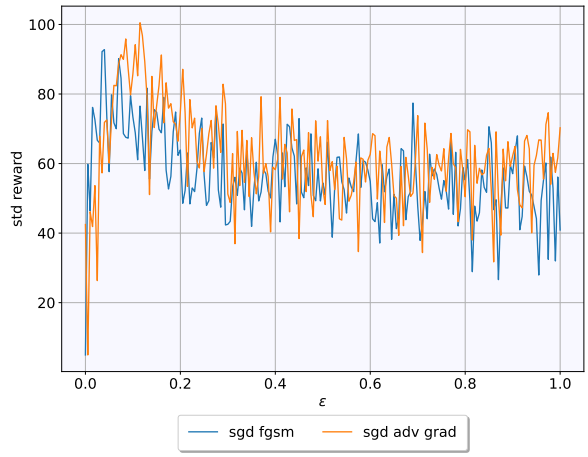
Figure 28. Mean cumulative perturbation (a) and episode length (b) for different numbers of iterations of noise sampling per state s_t.



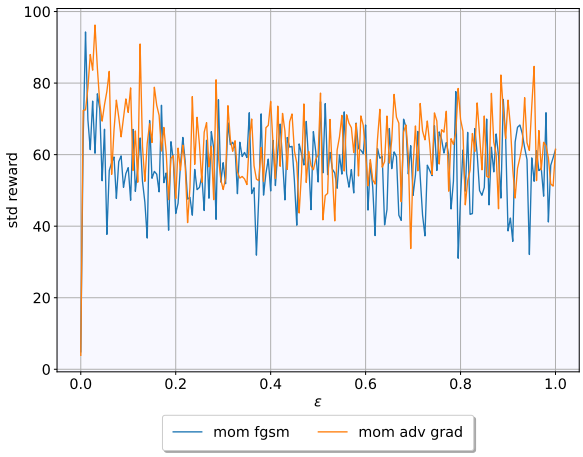
(a) Std reward of one-step methods.



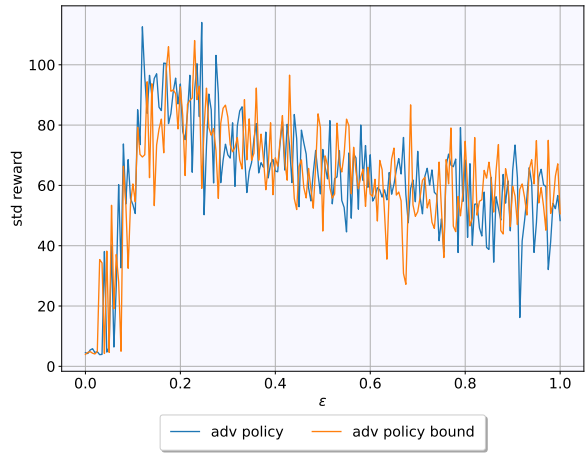
(b) Std reward of iterative methods.



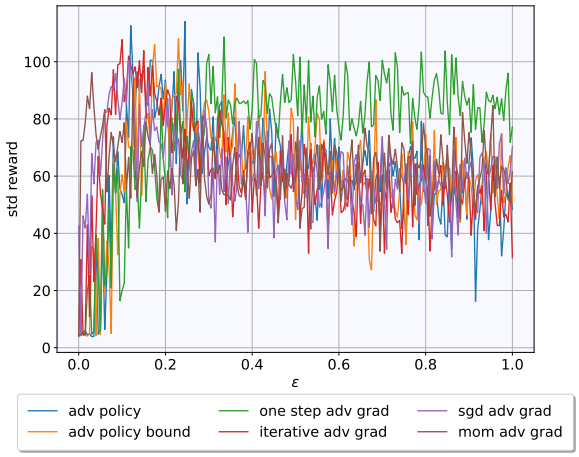
(c) Std reward of SGD methods.



(d) Std reward of momentum methods.

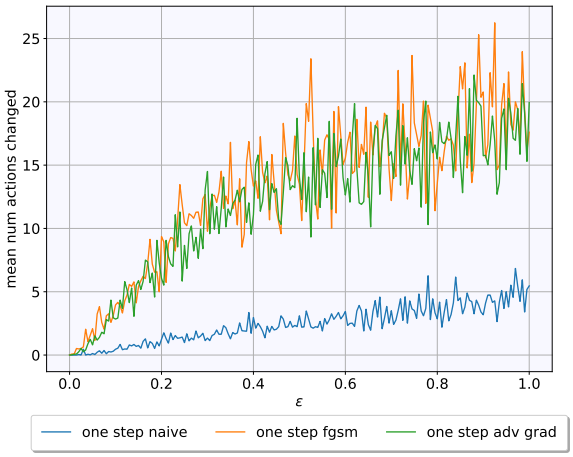


(e) Std reward of adv policy methods.

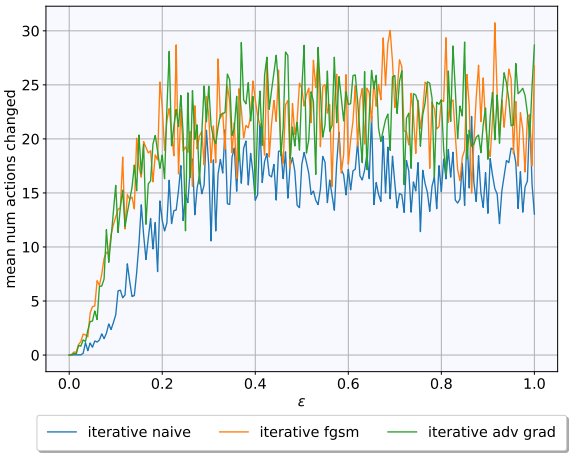


(f) Std reward (comparison).

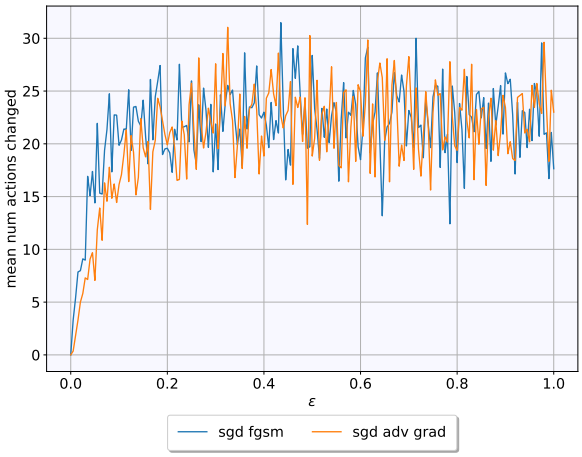
Figure 29. Comparison of the standard deviation of the reward reduction for different attack types. Results are estimated over 50 episodes with varying ϵ.



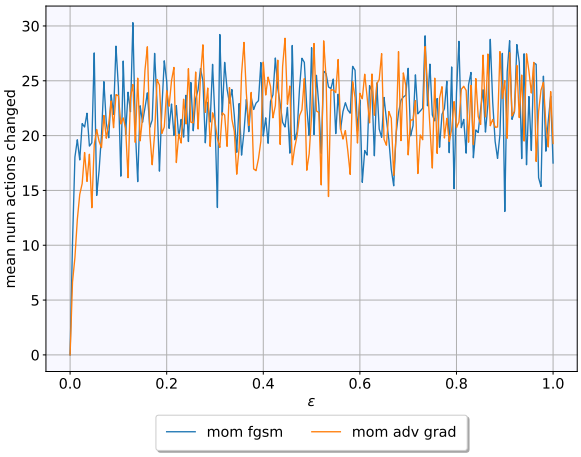
(a) Mean number of actions changed of one-step methods.



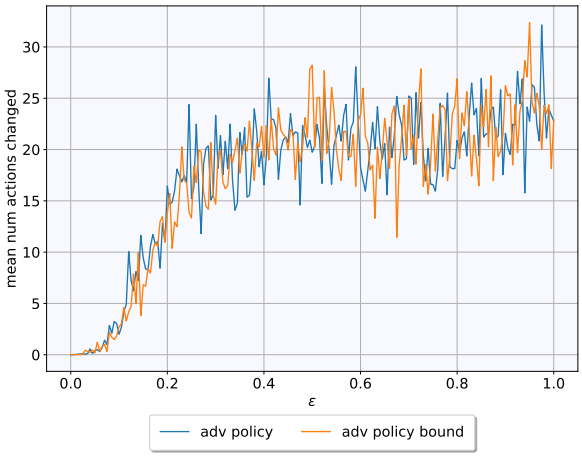
(b) Mean number of actions changed of iterative methods.



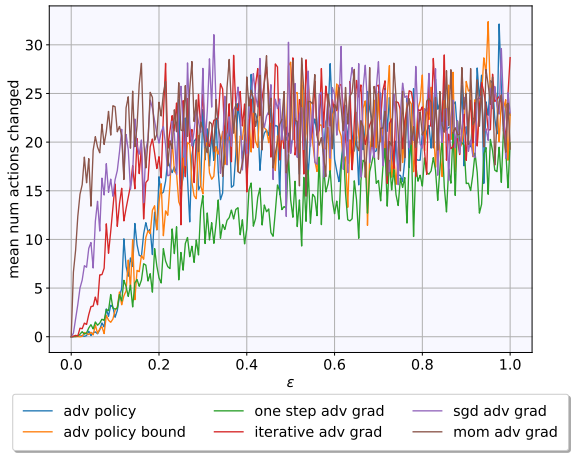
(c) Mean number of actions changed of SGD methods.



(d) Mean number of actions changed of momentum methods.

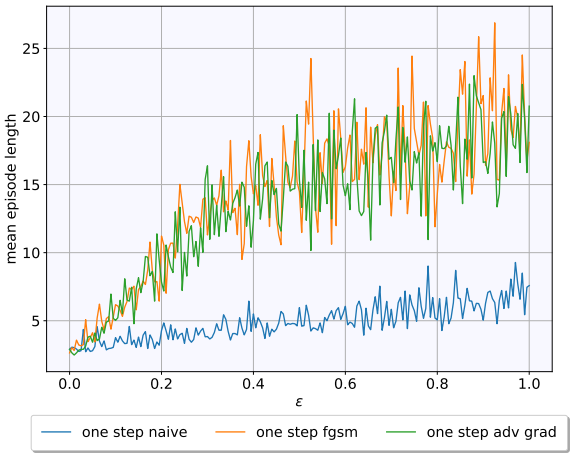


(e) Mean number of actions changed of adv policy methods.

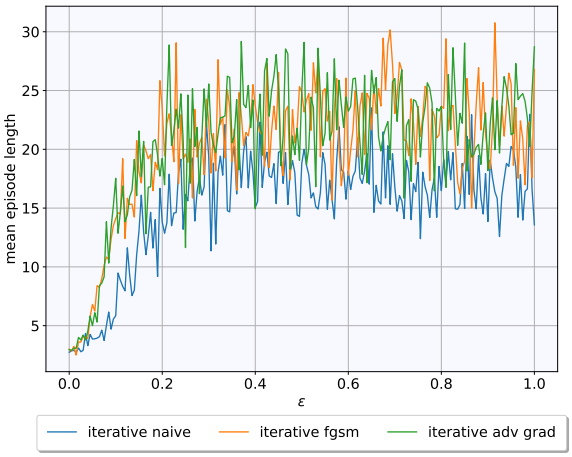


(f) Mean number of actions changed (comparison).

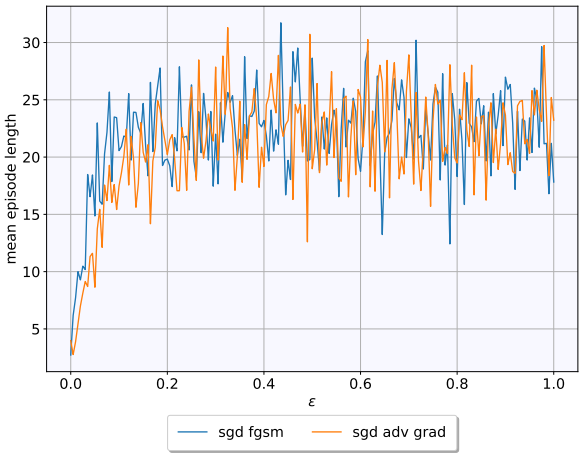
Figure 30. Comparison of mean number of actions changed for different attack types. Results are estimated over 50 episodes with varying ϵ.



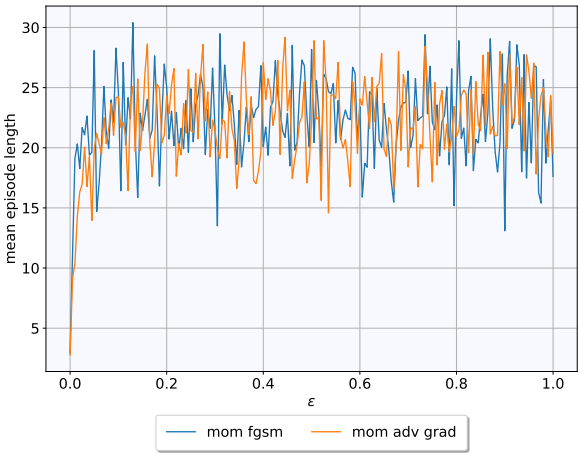
(a) Mean episode length of one-step methods.



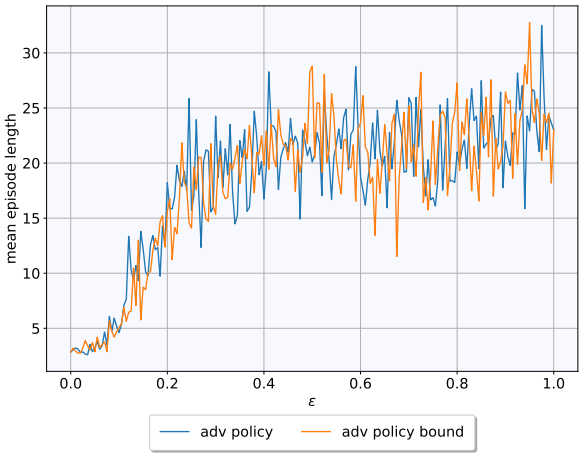
(b) Mean episode length of iterative methods.



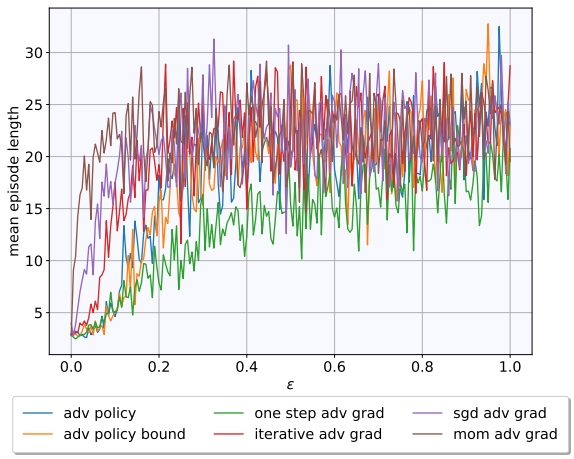
(c) Mean episode length of SGD methods.



(d) Mean episode length of momentum methods.

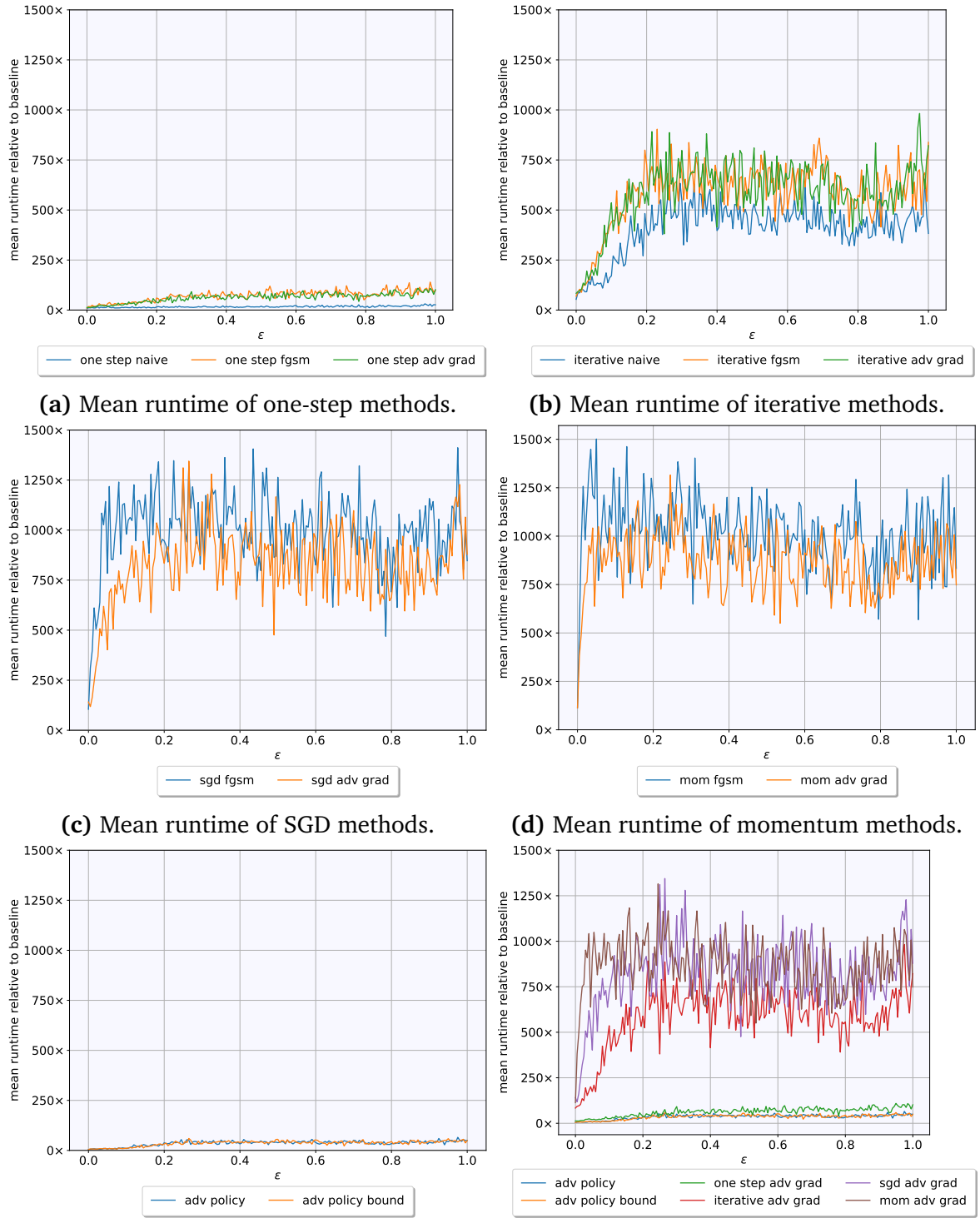


(e) Mean episode length of adv policy methods.



(f) Mean episode length (comparison).

Figure 31. Comparison of mean episode length for different attack types. Results are estimated over 50 episodes with varying ϵ.



(a) Mean runtime of one-step methods.

(b) Mean runtime of iterative methods.

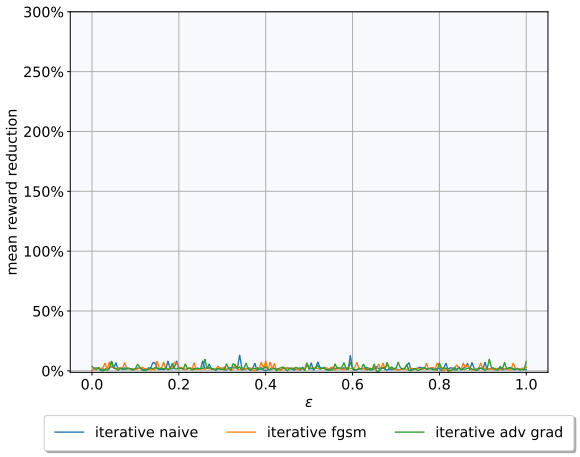
(c) Mean runtime of SGD methods.

(d) Mean runtime of momentum methods.

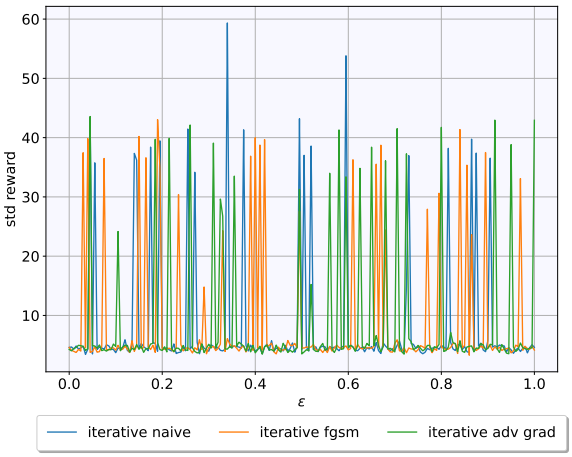
(e) Mean runtime of adv policy methods.

(f) Mean runtime (comparison).

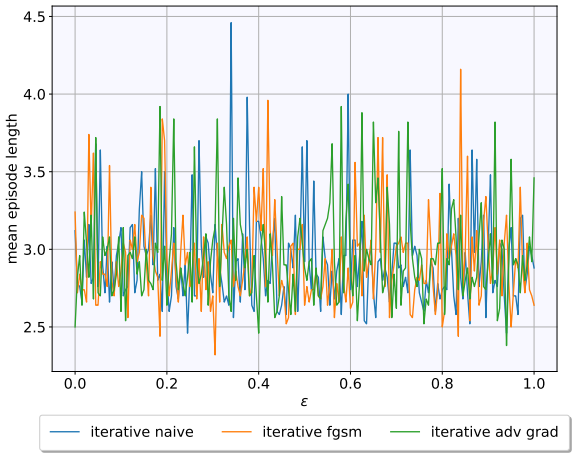
Figure 32. Comparison of mean runtime for different attack types. Results are estimated over 50 episodes with varying ϵ.



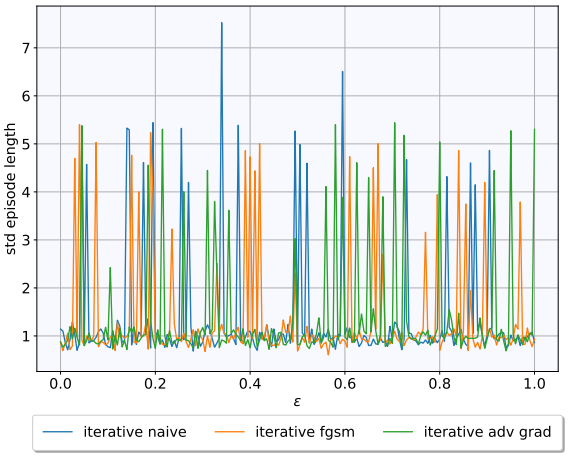
(a) Mean reward of iterative methods.



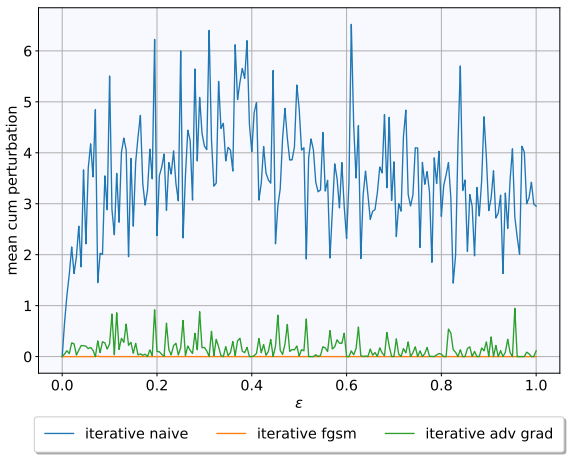
(b) Std reward of iterative methods.



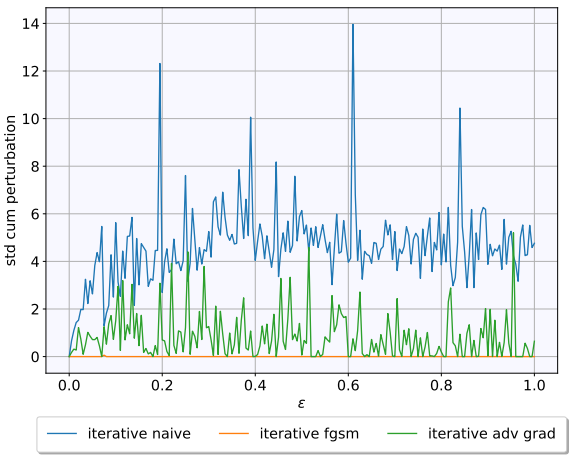
(c) Mean episode length of iterative methods.



(d) Std episode length of iterative methods.

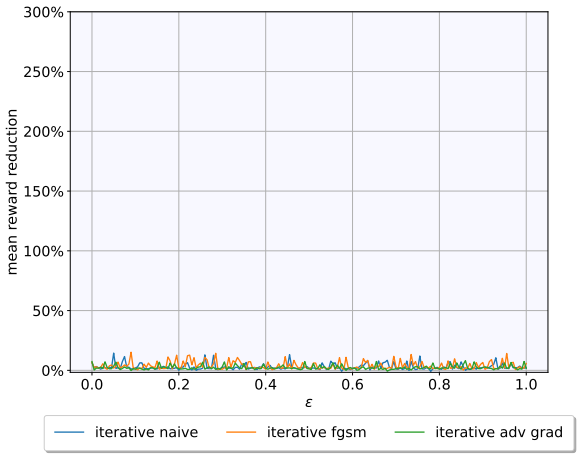


(e) Mean cumulative perturbation of iterative methods.

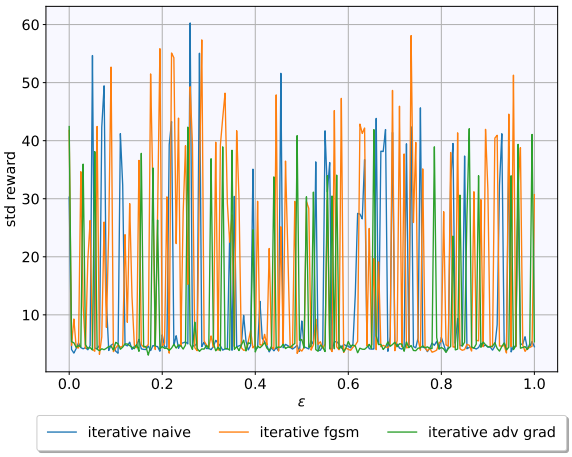


(f) Std cumulative perturbation of iterative methods.

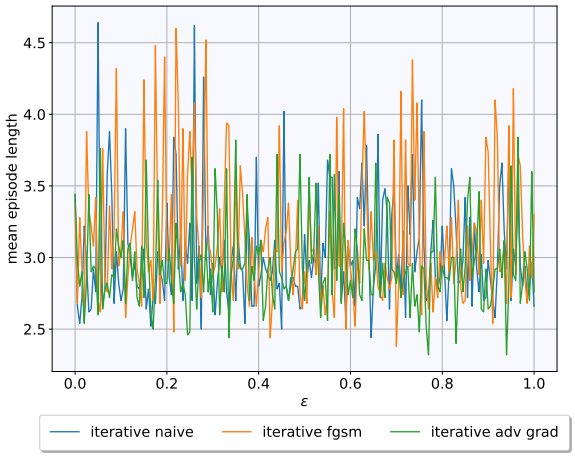
Figure 33. Performance comparison of iterative attacks when selecting the sample based on the probability of the best action a^*. Results are estimated over 50 episodes with varying ϵ.



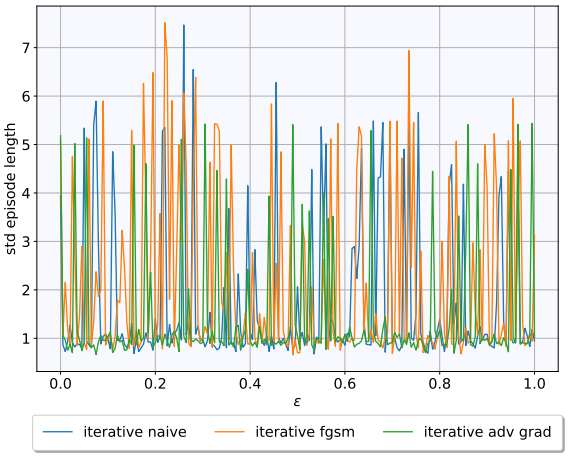
(a) Mean reward of iterative methods.



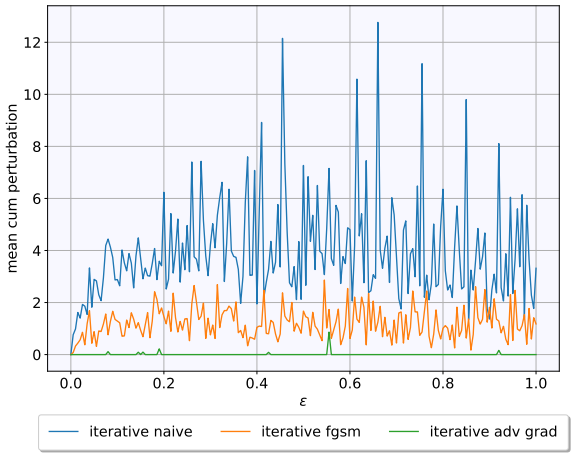
(b) Std reward of iterative methods.



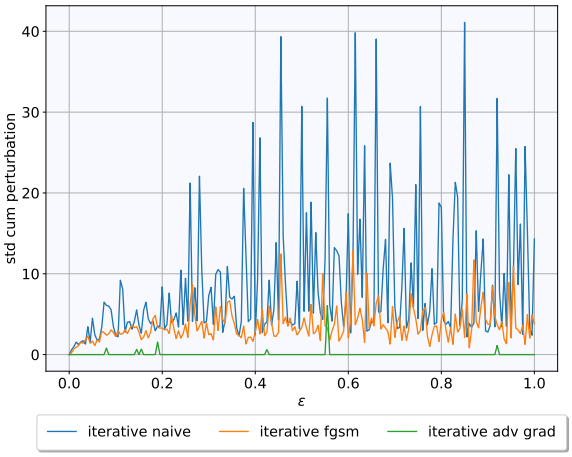
(c) Mean episode length of iterative methods.



(d) Std episode length of iterative methods.

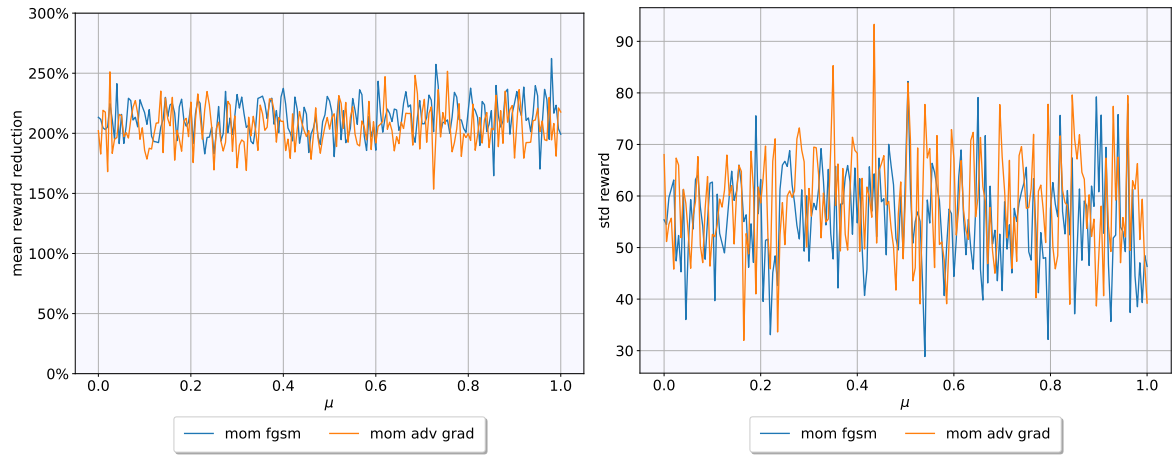


(e) Mean cumulative perturbation of iterative methods.



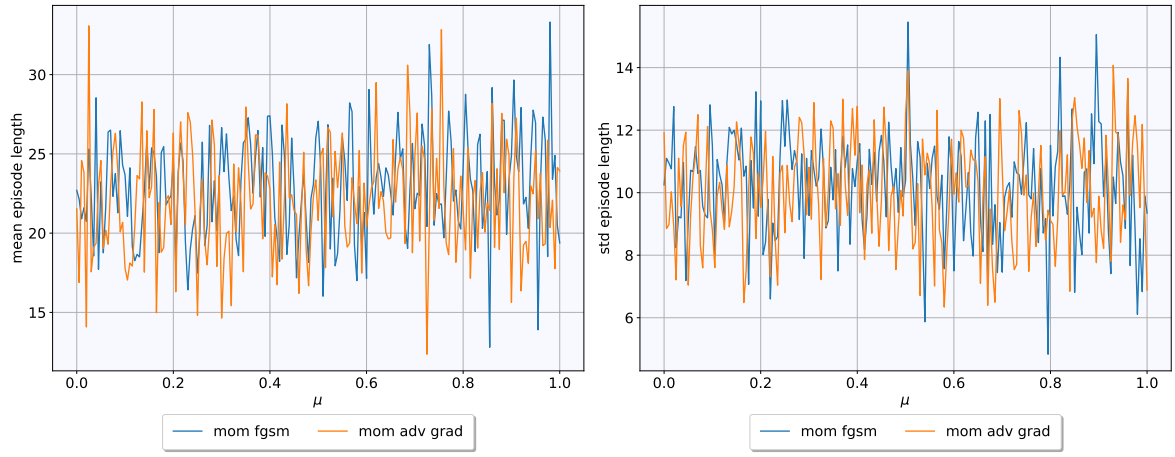
(f) Std cumulative perturbation of iterative methods.

Figure 34. Performance comparison of iterative attacks when selecting the sample based on the probability of the worst action a_w. Results are estimated over 50 episodes with varying ϵ.



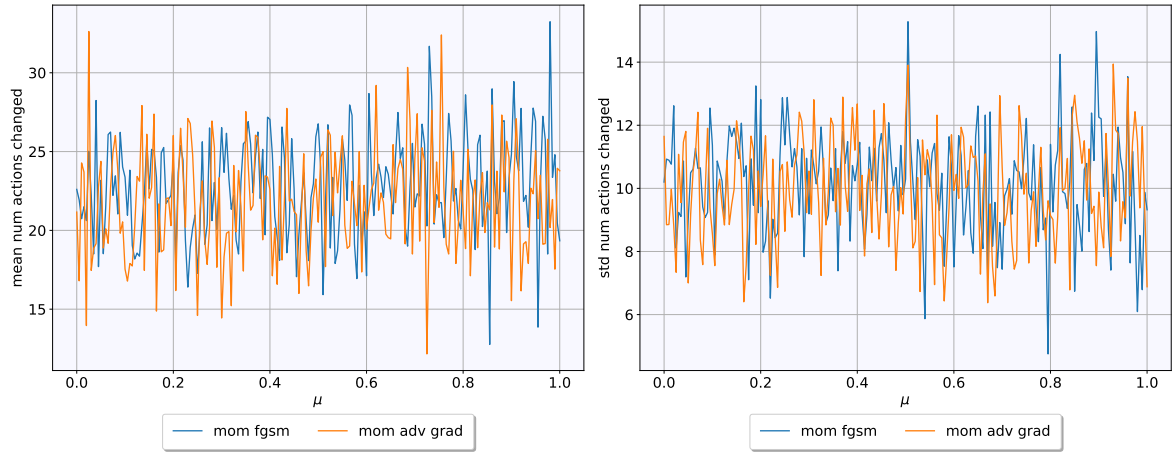
(a) Mean reward reduction.

(b) Standard deviation reward.



(c) Mean episode length.

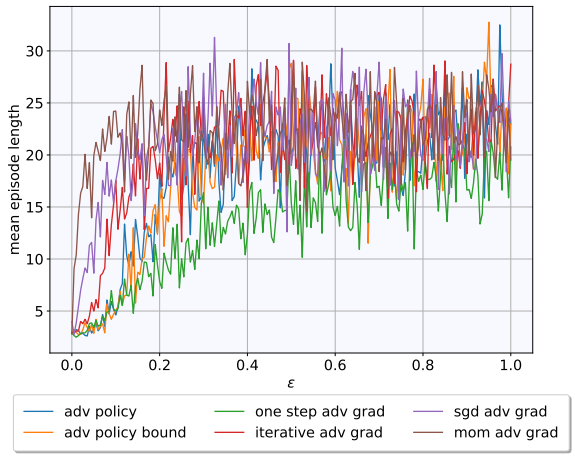
(d) Standard deviation episode length.



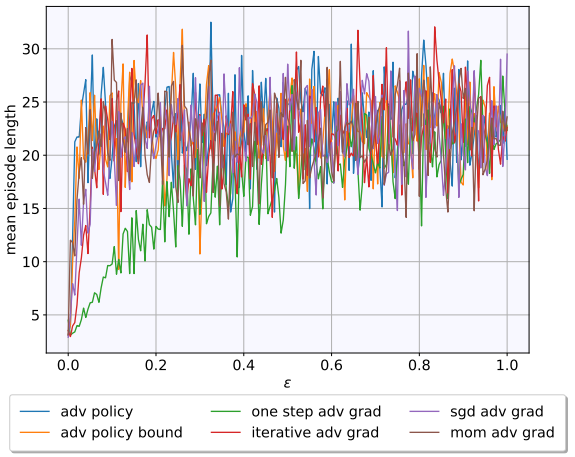
(e) Mean number of actions changed.

(f) Std number of actions changed.

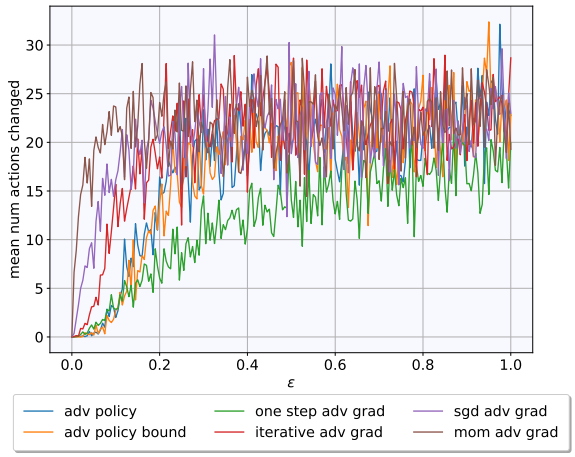
Figure 35. Performance of the two momentum boosted attacks with different $\mu \in [0, 1]$.



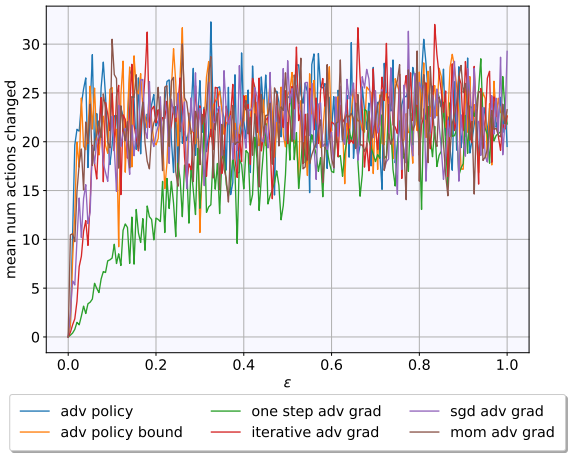
(a) Mean episode length L_∞



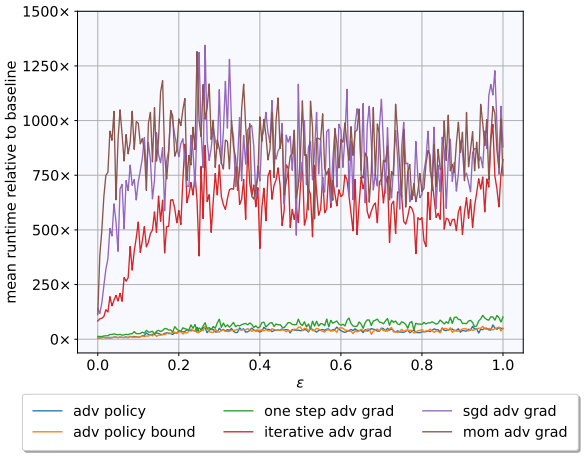
(b) Mean episode length L_2.



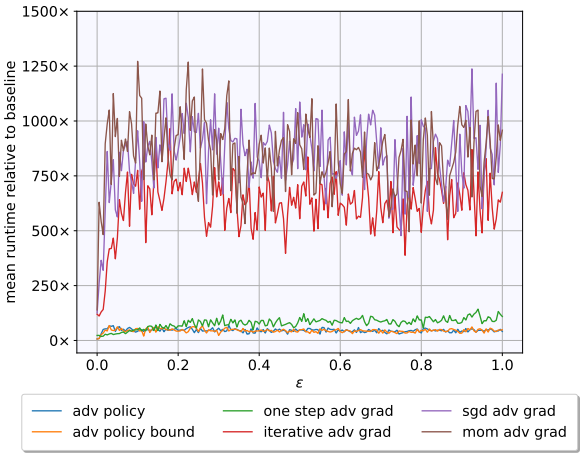
(c) Mean number of actions changed L_∞.



(d) Mean number of actions changed L_2.

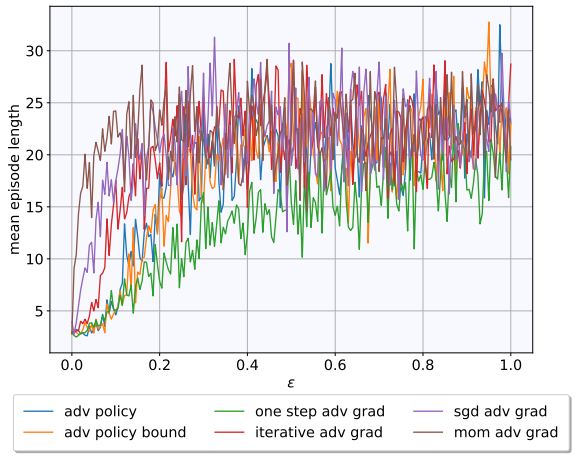


(e) Mean relative runtime L_∞.

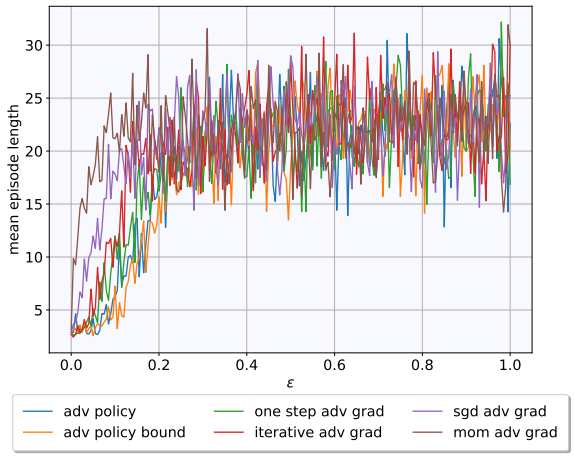


(f) Mean relative runtime L_2.

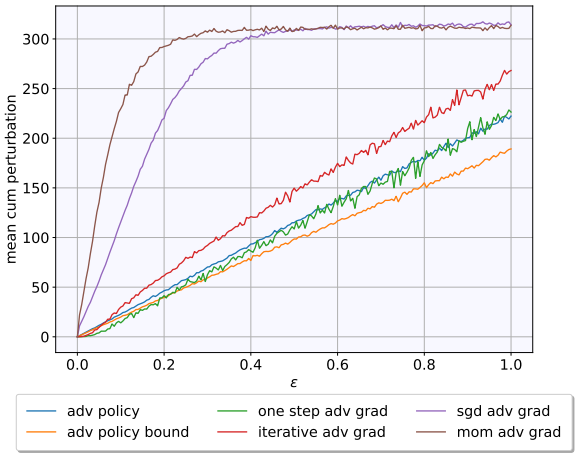
Figure 36. Each attack type's performance with respect to the L_∞-norm and L_2-norm constrained noise.



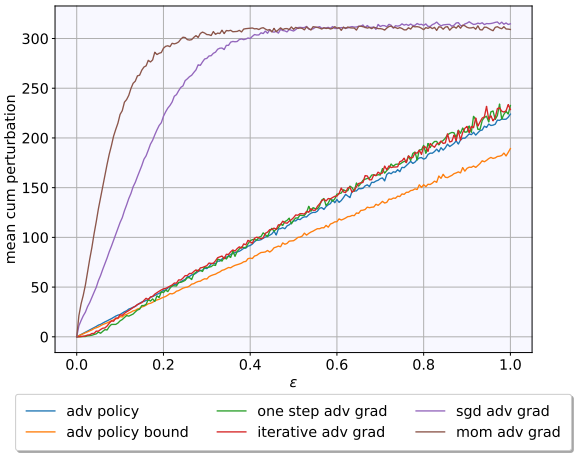
(a) Mean episode length uniform noise.



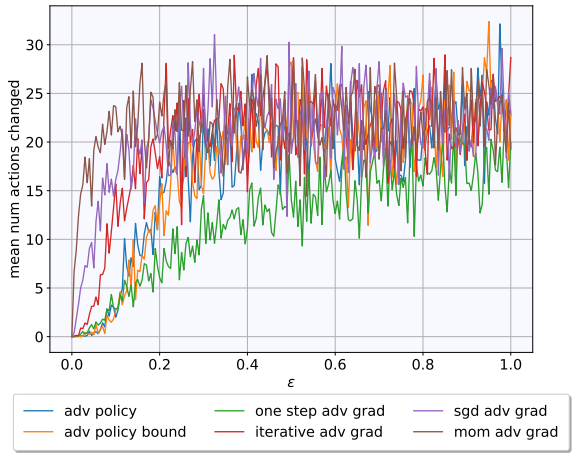
(b) Mean episode length non-uniform noise.



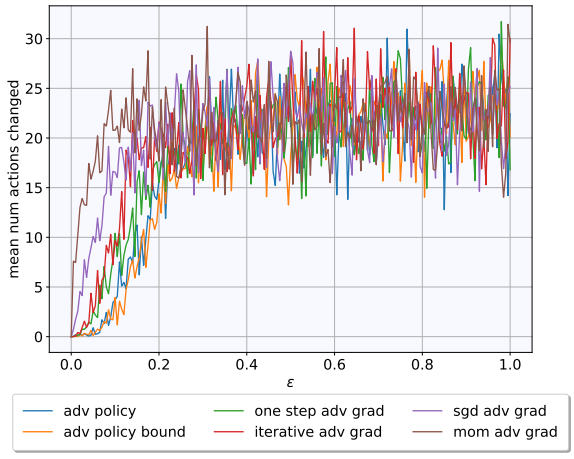
(c) Mean cumulative perturbation uniform noise.



(d) Mean cumulative perturbation non-uniform noise.

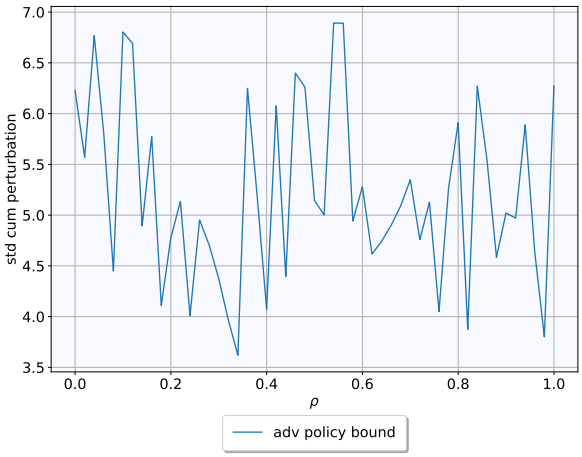


(e) Mean number actions changed uniform noise.

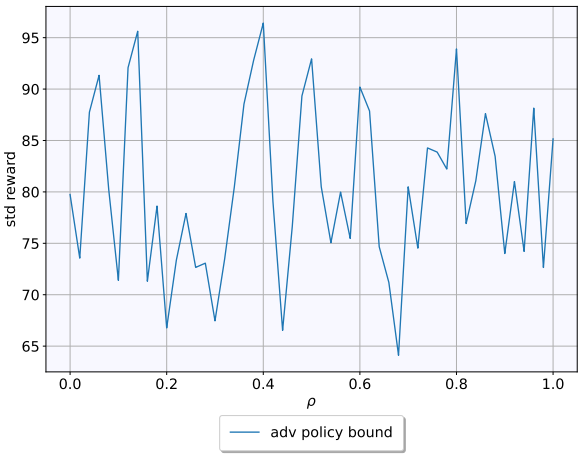


(f) Mean number actions changed non-uniform noise.

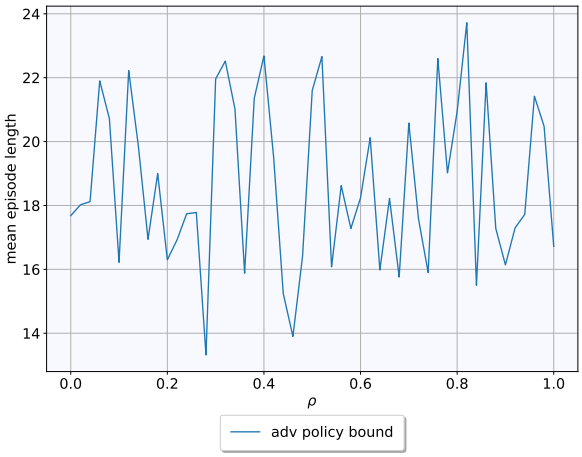
Figure 37. Each attack type's performance with respect to whether noise was sampled uniformly for all parameter of the observation space or specific to each.



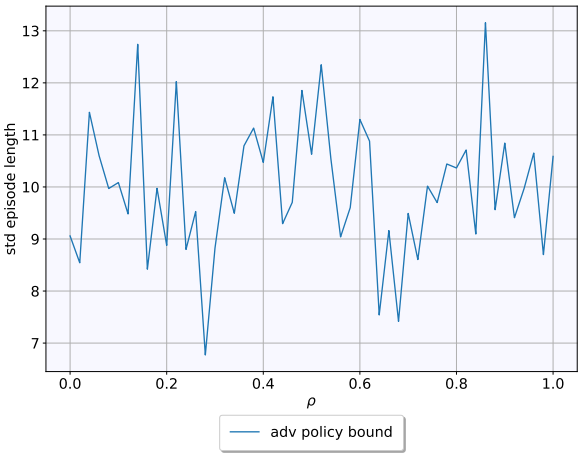
(a) Std number of cumulative perturbation.



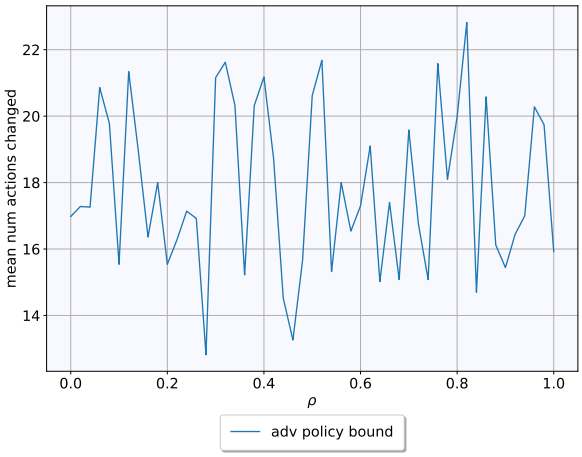
(b) Std number of reward reduction.



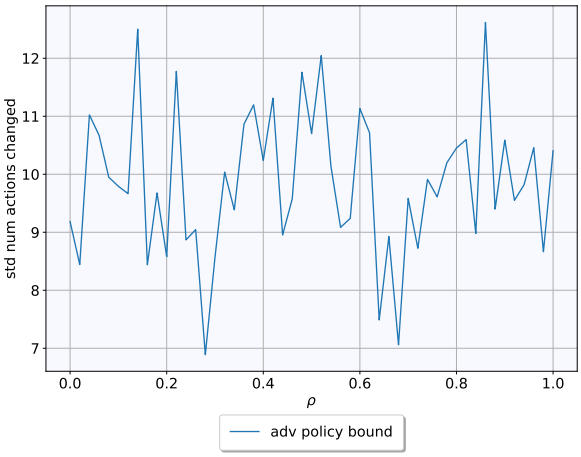
(c) Mean episode length.



(d) Std episode length.

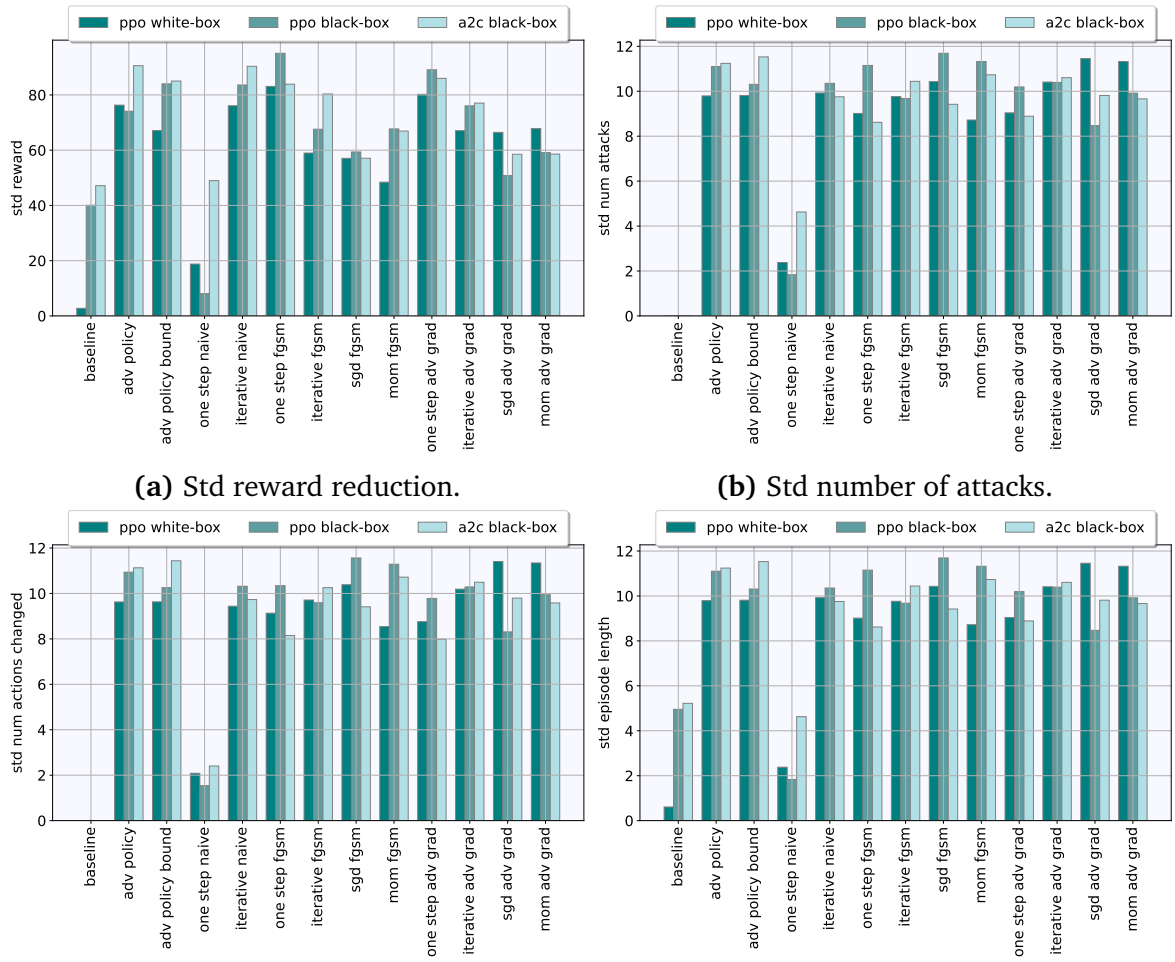


(e) Mean number of actions changed.



(f) Std number of actions changed.

Figure 38. Performance of bound adversarial policy attack for an increasing ρ. ϵ is fixed to 0.35.



(a) Std reward reduction.

(b) Std number of attacks.

(c) Std number of actions changed.

(d) Std episode length.

Figure 39. Standard deviations for transferability experiment. In this case results are estimated over 500 episodes with $\epsilon = 0.35$ and $\beta = 0$ (uniform attack).