Supplementary for Robust Deep Reinforcement Learning with Adversarial Attacks

Anonymous Author(s)
Affiliation
Address
email

3 Proof of Proposition and comment

Lemma 3.1. *The optimal adversarial policy is a stationary and deterministic policy given by*

$$\pi^*_{adv,k}(s_k) = arg \min_{\tilde{s}_k} \min_{\tilde{a}_k} R(s_k) + \gamma \mathbb{E}[V^*(s_{k+1})|s_k, \tilde{a}_k] \tag{1}$$

where $\tilde{a}_k \in arg \max_{a'} R(\tilde{s}_k) + \gamma \mathbb{E}[V^*(\tilde{s}_{k+1})|\tilde{s}_k, a']$ *and* $||\tilde{s}_k - s_k|| \in \Delta_s$

Proof.

$$V^*_{\pi_{adv,k}}(s_k) = R(s_k) + \gamma \mathbb{E}[\mathbb{E}[\mathbb{E}[V^*(s_{k+1})|s_k, a_k]]] \tag{2}$$

where the first expectation is over the stochastic adversarial policy, that is, $\pi_{adv}(s_{adv,k}|s_k)$, the middle expectation is over the action selected by the RL agent given an adversarially perturbed state, that is, $a_k \sim \pi^*(.|s_k) = P_d(\{a_k \in \text{argmax}_{a'} R(s_{adv,k}) + \gamma \mathbb{E}[V^*(s_{adv,k+1})|s_{adv,k}, a']\})$ where P_d is some arbitrary distribution for agent for breaking tie amongst best actions.
Thus,

$$V^*_{\pi_{adv,k}}(s_k) = R(s_k) + \gamma \mathbb{E}[\mathbb{E}[\mathbb{E}[V^*(s_{k+1})|s_k, \tilde{a}_k]]] \tag{3}$$

$$where$$

$$\tilde{a}_k \in \underset{a'}{\text{argmax}}\, R(s_{adv,k}) + \gamma \mathbb{E}[V^*(s_{adv,k+1})|s_{adv,k}, a'] \tag{4}$$

Let

$$\pi^*_{adv,k}(s_k) = \delta(s_{adv,k} = arg \min_{\tilde{s}_k} \min_{\tilde{a}_k} R(s_k) + \gamma \mathbb{E}[V^*(s_{k+1})|s_k, \tilde{a}_k]) \tag{5}$$

with $\tilde{a}_k$ same as defined in Eq. 4. Thus,

$$\begin{aligned} V^*_{\pi^*_{adv,k}}(s_k) &= R(s_k) + \gamma \min_{\tilde{s}_k} \min_{\tilde{a}_k} \mathbb{E}[V^*(s_{k+1})|s_k, \tilde{a}_k] \\ &\leq R(s_k) + \gamma \mathbb{E}[\mathbb{E}[\mathbb{E}[V^*(s_{k+1})|s_k, \tilde{a}_k]]] \;\; \forall \pi_{adv,k} \\ &= V^*_{\pi_{adv,k}}(s_k) \;\; \forall \pi_{adv,k} \end{aligned} \tag{6}$$

Thus, $\pi^*_{adv,k}$ as proposed in Eq. 5 is the optimal adversarial attack. □

Proposition 3.2. *Given a value function, V, if the fooled agent follows a class of policy (π) given as*

$$\pi(s) \in arg \max_a R(s) + \gamma \mathbb{E}[V(s')|s, a]$$

Then under the state perturbation attack, the worst that the agent can do can be given by

$$V^*_{adv}(s) = \min_{\tilde{s}} \min_{\tilde{a}} R(s) + \gamma \mathbb{E}[V^*_{adv}(s')|s, \tilde{a}] \tag{7}$$

where $\tilde{a} \in arg \max_{a'} R(\tilde{s}) + \gamma \mathbb{E}[V^*_{adv}(\tilde{s}')|\tilde{s}, a']$

Submitted to 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Do not distribute.

Proof. The worst that the agent can do under attack is given by

$$
\begin{aligned}
V^*_{adv}(s) &= \min_{\pi^{\infty}_{adv,0}} \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t)|s_0 = s, a_{adv,0}](a_t = \pi(\tilde{s}_t)) \\
&= \min_{\pi_{adv,0}} R(s_0) + \min_{\pi^{\infty}_{adv,1}} \mathbb{E}[\sum_{t=1}^{\infty} \gamma^t R(s_t)|s_0 = s, a_{adv,0}] \\
&= \min_{\pi_{adv,0}} R(s_0) + \min_{\pi^{\infty}_{adv,1}} \mathbb{E}[\mathbb{E}[\sum_{t=1}^{\infty} \gamma^t R(s_t)|s_1, a_{adv,1}, s_0 = s, a_{adv,0}]|s_0 = s, a_{adv,0}] \\
&= \min_{\pi_{adv,0}} R(s_0) + \gamma \min_{\pi^{\infty}_{adv,1}} \mathbb{E}[\mathbb{E}[\sum_{t=1}^{\infty} \gamma^t R(s_t)|s_1, a_{adv,1}]|s_0 = s, a_{adv,0}] \text{ (Markov property)} \\
&= \min_{\pi_{adv,0}} R(s_0) + \gamma \mathbb{E}[V^*_{adv}(s_1)]|s_0 = s, a_{adv,0}] \\
&= \min_{\tilde{s}}(s) + \gamma \mathbb{E}[V^*_{adv}(s')|s, \tilde{a}]
\end{aligned}
$$

with $\tilde{a} \in arg \max_{a'} R(\tilde{s}) + \gamma \mathbb{E}[V^*_{adv}(\tilde{s}')|\tilde{s}, a']$ □

Proposition 3.3. *Let the Q values of optimal policy be given by $Q^*(s, a)$ and $\pi^*(a|s)$ be conditional probability mass function generated as softmax of $Q^*(s, a)$ (with temperature $T > 0$).*

$$\pi^*(a|s) = \frac{e^{\frac{Q^*(s,a)}{T}}}{\sum_{i=1}^{n} e^{\frac{Q^*(s,a)}{T}}} \tag{8}$$

Let the action which has maximum pmf(π^*) be given as a^* and the worst possible action be given by a_w. Let the adversarial probability distribution be $P_{adv}(a)$ given by

$$P_{adv}(a) = \begin{cases} 1, & \textit{if } a = a_w \\ 0, & \textit{otherwise} \end{cases} \tag{9}$$

Then the objective function whose minimization leads to optimal adversarial attack on RL agent is

$$J(s) = Div(P_{adv}, \pi^*(s))$$

where $Div(P_{adv}, \pi^*(s))$ is any divergence measure between p_{adv} and π

Proof. Let $\pi_{adv}(s)$ be the policy followed by proposed adversarial agent for adversarial attack, that is,

$$\pi_{adv}(s) = arg \min_{s} J(s) = arg \min_{s} Div(P_{adv}, \pi^*(s))$$

For notational brevity, we define $s_{adv} := \pi_{adv}(s)$, then by Gibb's inequality [4], $\pi^*(a|s_{adv}) = P_{adv}$. In other words,

$$\pi^*(a|s_{adv}) = \begin{cases} 1, & \text{if } a = a_w \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

Let $\pi^*(a|s_{adv})$ be the policy followed by a RL agent fooled by adversarial attack and $V^*_{\pi_{adv}}(s)$ be the value function for RL agent corresponding to that policy. Thus,

$$
\begin{aligned}
V^*_{\pi_{adv}}(s) &= Q^*(s, \pi^*(a|s_{adv})) \\
&\leq Q^*(s, a), \quad \forall a \quad \text{(from Eq. 10)} \\
\implies V^*_{\pi_{adv}}(s) &= \min_a Q^*(s, a)
\end{aligned} \tag{11}
$$

Now, using the definition of optimal adversarial attack ($\pi^*_{adv}(s)$), we get

$$
\begin{aligned}
V^*_{\pi^*_{adv}}(s) &= \min_a Q^*(s, a) \\
&= V^*_{\pi_{adv}}(s) \quad \text{(using Eq. 11)} \\
&\leq \max_a Q^*(s, a) = V^*(s) \\
\implies V^*_{\pi_{adv}}(s) &= V^*_{\pi^*_{adv}}(s) \leq V^*(s)
\end{aligned}
$$

Thus, optimization of the proposed objective function leads to optimal adversarial policy.

Comment: The previously proposed adversarial attack [2] is suboptimal with respect to definition of adversarial attack on RL agent. We state their proposition and then prove that it is suboptimal. [2] Let the Q values of optimal policy be given by $Q^*(s,a)$ and $\pi^*(a|s)$ be conditional probability mass function generated as softmax of $Q^*(s,a)$ (with temperature $T > 0$).

$$\pi^*(a|s) = \frac{e^{\frac{Q^*(s,a)}{T}}}{\sum_{i=1}^{n} e^{\frac{Q^*(s,a)}{T}}} \tag{12}$$

Let the action which has maximum pmf(π^*) be given as a^* and the worst possible action be given by a_w. Let the adversarial probability distribution be $P_{adv}(a)$ given by

$$P_{adv}(a) = \begin{cases} 1, & \text{if } a = a^* \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

Then the objective function whose maximization leads to adversarial attack on RL agent is

$$J(s) = -\sum_{i=1}^{n} P_{adv}(a_i) log \pi^*(a_i|s) \tag{14}$$

Proof. We prove that the optimization (maximization) of Eq. 14 is suboptimal with respect to the definition of "optimal" adversarial attack (Definition 2 in main paper). We show it by providing a generic case where the optimization of previously proposed cost function doesn't necessarily lead to adversarial state that cause a reduction in value function for fooled RL agent.

$$\begin{aligned} J(s) &= -\sum_{i=1}^{n} P_{adv}(a_i) log \pi^*(a_i|s) \\ &= -log\pi^*(a^*|s) \quad \text{(using Eq. 13)} \qquad (15) \\ \implies J(s) &\to \infty \quad \text{as} \quad \pi^*(a^*|s) \to 0 \qquad (16) \end{aligned}$$

Thus, $J(s)$ attains maximum value when the adversary fools an agent into state s_{adv} such that $\pi^*(a^*|s) \approx 0$. It completely disregards the pmf corresponding to actions other than optimal action for that state. Let $\pi^*(a|s_{adv})$ be the policy followed by a RL agent fooled by adversarial attack and $V^*_{\pi_{adv}}(s)$ is the value function for RL agent corresponding to that policy. As we have shown, there is complete disregard for pmf corresponding to suboptimal actions. Therefore, we can end up in a situation where

$$\begin{aligned} \min_a Q^*(s,a) &\leq Q^*(s, \pi^*(a|s_{adv})) \\ &= V^*_{\pi_{adv}}(s) \\ &\leq Q^*(s,a^*) \end{aligned}$$

Thus,

$$\min_a Q^*(s,a) = V^*_{\pi^*_{adv}}(s) \leq V^*_{\pi_{adv}}(s)$$

Since, $V^*_{\pi^*_{adv}}(s) \leq V^*_{\pi_{adv}}(s)$, we have proved that this attack is suboptimal.

Proposition 3.4. *The proposed Q learning algorithm with finite state space $\mathcal{S}$ and action space $\mathcal{A}$ given by*

$$Q_{k+1}(s,a) = (1-\alpha_k)Q_k(s,a) + \alpha_k[R(s_k) + \gamma \min_{\tilde{s}'} \min_{\tilde{a}'} Q_k(s', \tilde{a}')] \tag{17}$$

where $\tilde{a}' \in arg\max_{a'} Q_k(\tilde{s}', a')$ converges w.p. 1 to $Q^*(s,a)$ provided $\sum_{k=0}^{\infty} \alpha_k = \infty$, $\sum_{k=0}^{\infty} \alpha_k^2 < \infty$ and all state action pairs (s,a) are explored infinitely often. Here, s' is the next state that agent experiences given that the agent takes action a at state s. The time dependent learning rate is $\alpha_k := \alpha_k(s,a)$ and only the Q value of pair (s,a) that is visited at time instant k is updated.

Proof.

$$Q_{k+1}(s,a) - Q_k^*(s,a) = (1-\alpha_k(s,a))(Q_k(s,a) - Q_k^*(s,a)) + \alpha_k(s,a)(R(s_k) + \gamma \min_{\tilde{s}'} \min_{\tilde{a}'} Q_k(s', \tilde{a}') - Q_k^*(s,a))$$
$$\Delta_{k+1} := (1-\alpha_k(s,a))\Delta_k + \alpha_k(s,a)(R(s_k) + \gamma \min_{\tilde{s}'} \min_{\tilde{a}'} Q_k(s', \tilde{a}') - Q_k^*(s,a))$$

Let $F_k := R(s) + \gamma \min_{\tilde{s}} \min_{\tilde{a}'} Q_k(s', \tilde{a}') - Q_k^*(s,a)$ and $\mathcal{F}_k := \{\Delta_k, \Delta_{k-1}, ..F_{k-1}, ..., \alpha_{k-1}, ..\}$

$$\begin{aligned}
|\mathbb{E}[F_k|\mathcal{F}_k]| &= |\mathbb{E}[R(s_k) + \gamma \min_{\tilde{s}'} \min_{\tilde{a}'} Q_k(s', \tilde{a}') - Q_k^*(s,a)|\mathcal{F}_k]| \\
&= |\mathbb{E}[R(s_k) + \gamma \min_{\tilde{s}'} \min_{\tilde{a}'} Q_k(s', \tilde{a}') - \mathbb{E}[R(s_k) + \gamma \min_{\tilde{s}'_2} \min_{\tilde{a}'_2} Q_k^*(s', \tilde{a}'_2)]|\mathcal{F}_k]| \\
&\leq \gamma \mathbb{E}[|Q_k(s', \tilde{a_1}'^*) - Q_k^*(s', \tilde{a_2}'^*)||\mathcal{F}_k] \text{ (where } \tilde{a}_1^*, \tilde{a}_2^* \text{ are respective optimal sol)} \\
&= \gamma \mathbb{E}[Q_k(s', \tilde{a_1}'^*) - Q_k^*(s', \tilde{a_2}'^*)|\mathcal{F}_k] \text{ or} \\
&\quad \gamma \mathbb{E}[Q_k^*(s', \tilde{a_2}'^*) - Q_k(s', \tilde{a_1}'^*)|\mathcal{F}_k] \\
&\leq \gamma \mathbb{E}[Q_k(s', \tilde{a_2}'^*) - Q_k^*(s', \tilde{a_2}'^*)|\mathcal{F}_k] \quad \text{(as } \tilde{a}_1^* \text{ is the optimal sol) or} \\
&\quad \gamma \mathbb{E}[Q_k^*(s', \tilde{a_1}'^*) - Q_k(s', \tilde{a_1}'^*)|\mathcal{F}_k] \text{ (as } \tilde{a}_2^* \text{ is the optimal sol)} \\
&\leq \gamma ||Q_k - Q^*||_\infty \\
|\mathbb{E}[F_k|\mathcal{F}_k]| &\leq \gamma ||\Delta_k||_\infty
\end{aligned}$$

Now, let $M = \max\{R(s_0), ||Q_0(s,a)||_\infty\}$. Then $|Q_1(s,a)| \leq (1-\epsilon_1(s,a))M + \epsilon_1(s,a)\{M + \gamma M\} = M(1+\gamma)$ or $|Q_1(s,a)| \leq M(1+\gamma)$, repeating it, we'll get $|Q_k(s,a)| \leq M(1+\gamma+\gamma^2 ... + \gamma^k) \leq \frac{M}{1-\gamma}$. Thus, $Q_k(s,a)$ is bounded. Now,

$$\begin{aligned}
&Var[F_k|\mathcal{F}_k] \\
&= \mathbb{E}[\{R(s_k) + \gamma \min_{\tilde{s}} \min_{\tilde{a}'} Q_k(s', \tilde{a}') - Q_k^*(s,a) - \mathbb{E}[R(s) + \gamma \min_{\tilde{s}} \min_{\tilde{a}'} Q_k(s', \tilde{a}') - Q_k^*(s,a)]\}^2|\mathcal{F}_k] \\
&= \mathbb{E}[\{R(s) + \gamma \min_{\tilde{s}} \min_{\tilde{a}'} Q_k(s', \tilde{a}') - \mathbb{E}[R(s) + \gamma \min_{\tilde{s}} \min_{\tilde{a}'} Q_k(s', \tilde{a}')]\}^2|\mathcal{F}_k] \\
&= Var[R(s) + \gamma \min_{\tilde{s}} \min_{\tilde{a}'} Q_k(s', \tilde{a}')] \\
&\leq L \quad \text{(as } Q_k(s,a) \text{ is finite and } R(s,a) \text{ is finite random variable)}
\end{aligned}$$

Now, $||\triangle_k||_\infty$ is finite for all k (as $Q_k(s,a)$ is always finite). So, given L, there exists C such that $L \leq C(1 + ||\triangle_k||_\infty^2)$. This implies $Var[F_k(x)|\mathcal{F}_k] \leq L \leq C(1 + ||\triangle_k||_\infty^2$.
Hence, convergence of stochastic iterative algorithm holds by [3], that is, $\Delta_k := Q_k(s,a) - Q^*(s,a)$ converges w.p. 1 to 0 to complete the proof. □

4 Adversarial attacks

4.1 Gradient based Attack DDQN

We have outlined the gradient based DDQN attack in Algo. 1

4.2 Gradient based attack on DDPG

The gradient based attack for DDPG is similar to DDQN with the objective function that adversary need to minimize being given by the optimal value function of critic ($Q^*(s,a)$). Here the gradient is given by

$$\nabla_s Q^*(s,U) = \frac{\partial Q^*}{\partial s} + \frac{\partial Q^*}{\partial U}\frac{\partial U}{\partial s}$$

where $U(s)$ represents the policy given by actor. The algorithm has been provided in Algo. 2.

4.3 Naive attack on DDQN

We outline the algorithm for Naive adversarial attack on DDQN in Algo. 3

Algorithm 1 Gradient based attack (DDQN)

procedure GRAD($Q^{target}, Q, s, \epsilon, n, \alpha, \beta$) ▷ Gradient based attack function takes Q network (Q), current state(s), adversarial attack magnitude constraint(ϵ), parameters of beta distribution(α, β) and number of times to sample noise(n) as input
    $a^* \leftarrow arg \max_a Q(s, a), Q^* \leftarrow \max_a Q^{target}(s, a)$ ▷ Determine optimal action and value function
    $\pi^{target} \leftarrow softmax(Q^{target})$ ▷ Pass Q through softmax layer to convert it into pmf
    $grad \leftarrow \nabla_s J(s, \pi^{target})$ ▷ Determine the gradient
    $grad\_dir \leftarrow \frac{\nabla_s J(s,\pi^{target})}{||\nabla_s J(s,\pi^{target})||}$ ▷ l_2 constrained norm of gradient
    for $i = 1 : n$ **do** ▷ Sample a few times
        $n_i \sim beta(\alpha, \beta)$ ▷ Sample noise
        $s_i \leftarrow s - n_i * grad\_dir$ ▷ Possible adversarial state determined by sampled noise in the direction of gradient
        $a_{adv} \leftarrow arg \max_a Q(s_i, a)$ ▷ Determine optimal action in potential adversarial state
        $Q^{target}_{adv} \leftarrow Q^{target}(s, a_{adv})$ ▷ Determine the value of potential adversarial action corresponding to potential adversarial state for current state
        if $Q^{target}_{adv} < Q^*$ **then** ▷ if the potential adversarial state leads to bad action
            $Q^* \leftarrow Q^{target}_{adv}$ ▷ Store the value function of that potential bad action
            $s_{adv} \leftarrow s_i$ ▷ Store that state as possible adversarial state
        else
            do nothing
        end if
    end for
    return s_{adv} ▷ Return adversarial state
end procedure

Algorithm 2 Gradient based attack (DDPG)

procedure GRAD($Q^{target}, U, s, \epsilon, n, \alpha, \beta$) ▷ Gradient based attack function takes target Q network (critic) Q^{target} , actor network U, current state(s), adversarial attack magnitude constraint(ϵ), parameters of beta distribution(α, β) and number of times to sample noise(n) as input
    $a^* \leftarrow U(s), Q^* \leftarrow Q^{target}(s, a^*)$ ▷ Determine optimal action and value function
    $grad \leftarrow \nabla_s Q^{target}(s, a)$ ▷ Determine the gradient
    $grad\_dir \leftarrow \frac{\nabla_s Q^{target}(s,a)}{||\nabla_s Q^{target}(s,a)||}$ ▷ l_2 constrained norm of gradient
    for $i = 1 : n$ **do** ▷ Sample a few times
        $n_i \sim beta(\alpha, \beta)$ ▷ Sample noise
        $s_i \leftarrow s - n_i * grad\_dir$ ▷ Possible adversarial state determined by sampled noise in the direction of gradient
        $a_{adv} \leftarrow U(s_i)$ ▷ Determine optimal action in potential adversarial state
        $Q^{target}_{adv} \leftarrow Q^{target}(s, a_{adv})$ ▷ Determine the value of potential adversarial action corresponding to potential adversarial state for current state
        if $Q^{target}_{adv} < Q^*$ **then** ▷ if the potential adversarial state leads to bad action
            $Q^* \leftarrow Q^{target}_{adv}$ ▷ Store the value function of that potential bad action
            $s_{adv} \leftarrow s_i$ ▷ Store that state as possible adversarial state
        else
            do nothing
        end if
    end for
    return s_{adv} ▷ Return adversarial state
end procedure

Algorithm 3 Naive attack (DDQN)

procedure NAIVE($Q^{target}, Q, s, \epsilon, n, \alpha, \beta$) ▷ Naive attack function takes Q network (Q), current state(s), adversarial attack magnitude constraint(ϵ), parameters of beta distribution(α, β) and number of times to sample noise(n) as input
  $a^* \leftarrow arg \max_a Q(s, a), Q^* \leftarrow \max_a Q^{target}(s, a)$ ▷ Determine optimal action value function
  for $i = 1 : n$ **do** ▷ Sample a few times
    $n_i \sim beta(\alpha, \beta) - 0.5$ ▷ Sample noise
    $s_i \leftarrow s + \epsilon * n_i$ ▷ Possible adversarial state determined by sampled noise
    $a_{adv} \leftarrow arg \max_a Q(s_i, a)$ ▷ Determine optimal action in potential adversarial state
    $Q^{target}_{adv} \leftarrow Q^{target}(s, a_{adv})$ ▷ Determine the value of potential adversarial action corresponding to potential adversarial state for current state
    if $Q^{target}_{adv} < Q^*$ **then** ▷ if the potential adversarial state leads to bad action
      $Q^* \leftarrow Q^{target}_{adv}$ ▷ Store the value function of that potential bad action
      $s_{adv} \leftarrow s_i$ ▷ Store possible adversarial state
    else
      do nothing
    end if
  end for
  return s_{adv} ▷ Adversarial state
end procedure

4.4 Naive attack on DDPG

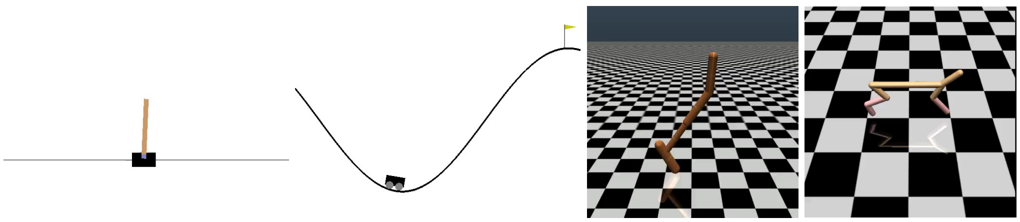
The objective function used by adversary in this case is the $Q^*_{critic}(s, U(s)) := Q^*(s, U)$ (U is the actor network), that is, the value function determined by the trained critic network. Algorithm for naive attack on DDPG has been provided in Algo. 4

Algorithm 4 Naive attack (DDPG)

procedure NAIVE($Q^{target}, U, s, \epsilon, n, \alpha, \beta$) ▷ Naive attack function takes trained target critic network Q^{target}, trained actor network U, current state(s), adversarial attack magnitude constraint(ϵ), parameters of beta distribution(α, β) and number of times to sample noise(n) as input
  $a^* \leftarrow U(s), Q^* \leftarrow Q^{target}(s, a^*)$ ▷ Determine optimal action and action value function
  for $i = 1 : n$ **do** ▷ Sample a few times
    $n_i \sim beta(\alpha, \beta) - 0.5$ ▷ Sample noise
    $s_i \leftarrow s + \epsilon * n_i$ ▷ Possible adversarial state determined by sampled noise
    $a_{adv} \leftarrow U(s_i)$ ▷ Determine optimal action in potential adversarial state
    $Q^{target}_{adv} \leftarrow Q^{target}(s, a_{adv})$ ▷ Determine the value of potential adversarial action corresponding to potential adversarial state for current state
    if $Q^{target}_{adv} < Q^*$ **then** ▷ if the potential adversarial state leads to bad action
      $Q^* \leftarrow Q^{target}_{adv}$ ▷ Store the value function of that potential bad action
      $s_{adv} \leftarrow s_i$ ▷ Store possible adversarial state
    else
      do nothing
    end if
  end for
  return s_{adv} ▷ Adversarial state
end procedure

Algorithm 5 Training with adversarial perturbation (DDQN)

procedure ADV TRAIN (Q^{target}, Q) ▷ Gradient based adversarial training method takes pre-trained network
  for $i = 1 : iterations$ **do** ▷ Train adversarially for number of timesteps
    Reset the environment and receive observation
    while not terminal or not max time steps per episode reached **do**
      $s_{adv} \leftarrow Grad(Q^{target}, Q, s, \epsilon, n, \alpha, \beta)$ ▷ Fool the agent
      $a \leftarrow arg \max_{a} Q(s_{adv,a})$ ▷ Fooled agent takes action according to behavior policy
      $s, r \leftarrow Env(a, s)$ ▷ Environment returns next state and reward corresponding to state s and action a
      Update the weights of network according to DDQN algorithm
    end while
  end for
end procedure



(a) Cart-Pole (b) Mountain Car (c) Hopper (d) Half Cheetah

Figure 1: Different environments in OpenAi gym (MuJoCo for simulation of dynamics)

5 Adversarial Training DDQN

6 Adversarial Training of DDPG

The algorithm for adversarial training of DDPG has been provided in Algo. 6.

Algorithm 6 Training with adversarial perturbation (DDPG)

procedure ADV TRAIN ($Q^{target}, Q, U^{target}, U$) ▷ Gradient based adversarial training method takes pre-trained network
  for $i = 1 : iterations$ **do** ▷ Train adversarially for number of timesteps
    Reset the environment and receive observation
    while not terminal or not max time steps per episode reached **do**
      $s_{adv} \leftarrow Grad(Q^{target}, U, s, \epsilon, n, \alpha, \beta)$ ▷ Fool the agent
      $a \leftarrow U(s_{adv})$ ▷ Fooled agent takes action according to behavior policy
      $s, r \leftarrow Env(a, s)$ ▷ Environment returns next state and reward corresponding to state s and action a
      Update the weights of network according to DDPG algorithm
    end while
  end for
end procedure

7 Experimental Setup

In this section, we give details of the experimental setup that has been used for results presented in the following section. All the experiments have been performed within OpenAi gym environment (Fig. 1)

7.1 DDQN

The Deep Double Q learning for cart pole environment used 3 layers of 16 units each with Rectified Linear Unit (ReLu) activation function whereas the mountain car environment used 2 hidden layers of 100 units each of ReLu activation function. The discount factor for both of them was set at $0.99$and target network update rate were 10^{-2}. The “supervised learning" of networks was done with Adam optimization and learning rate of 10^{-3}. The cartpole environment was trained for 50000 timesteps while Mountain Car was trained for 40000 time steps. The repository that we used was Keras-rl ([5]).

7.2 RBF Q learning

For Cart-pole, each dimension of state input was divided into 3 bins (b). The centroids were uniformly distributed along those bins. The variance of radial activation were $\frac{2}{b^2}$. Discount factor of 0.99 was used. The learning rate was given by 0.001. It was trained for 40000 time steps For Mountain car environment, the learning rate was 0.01 and number of bins were 4. The discount factor was 0.99. Total number of timesteps were 60000.

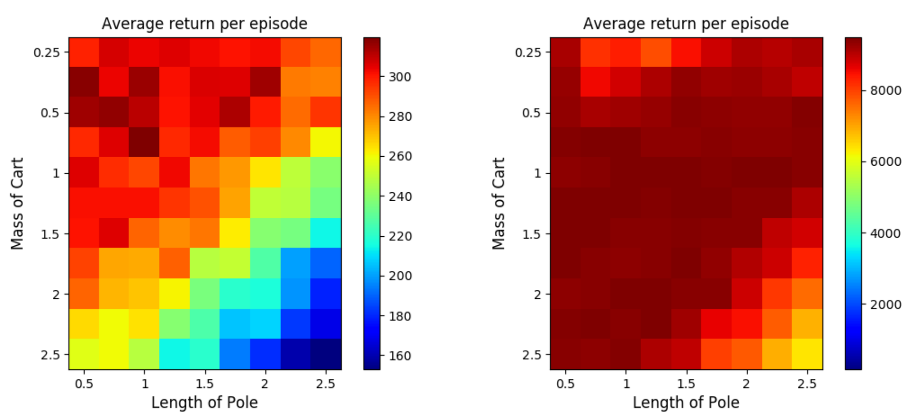
7.3 DDPG

For hopper and half cheetah environment, there were 2 hidden layers of 400 and 300 ReLu units for both actor and critic networks. For Hopper, the number of time steps it was trained were 1 million, discount factor was 0.99. The learning rate of critic network was 10^{-3} while the learning rate of actor was 10^{-4}. Half cheetah also used the same network as hopper. It also had the same learning rate and discount factor. It was trained for 2 million time steps. The repository that we used was rllab [1]

7.4 Adversarial Training

For adversarial training, the sampling frequency was 200 and the vanilla trained network was re-trained adversarially for same amount of time steps. We used adversarial magnitude of 0.05 for half cheetah and 0.03 for hopper. The sampling frequency was 100. We must point out that for the results shown in paper, comparison has been shown between both vanilla and adversarially trained networks that have been trained for exactly same number of timesteps.

8 Robust Training Colormap for Cartpole



(a) DDQN Cartpole

(b) Robust DDQN Cartpole

Figure 2: Subfigure (a) shows the average return per episode for cart-pole environment using DDQN algorithm across variation of mass of cart and length of pole. Subfigure(b) shows the same information for adversarially trained DDQN agent. We can observe significant improvement over the return for agent across different parameters. “Zoomed" colormap for DDQN cartpole comparison has been presented.

References

[1] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

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

[3] Tommi Jaakkola, Michael Jordan, and Satinder Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computations*, 6(6):1185–1201, 1994.

[4] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[5] Matthias Plappert. keras-rl. `https://github.com/matthiasplappert/keras-rl`, 2016.