Summary of Three Recent Papers: Deep Reinforcement Learning and Adversarial Attacks

Presenter: Ji Gao

Department of Computer Science, University of Virginia

https://qdata.github.io/deep2Read/
Paper List

• Adversarial Attacks on Neural Network Policies
• Vulnerability of Deep Reinforcement Learning to Policy Induction Attacks
• Online Robust Policy Learning in the Presence of Unknown Adversaries
• Robust Deep Reinforcement Learning with Adversarial Attacks
Adversarial Attacks on Neural Network Policies

Sandy Huangy, Nicolas Papernot, Ian Goodfellowx, Yan Duanyx, Pieter Abbeely

• 2017 ICLR Workshop

• Idea: Craft adversarial samples in the input feature (State) of RL algorithm, lead to a large degrade of test-time performance

• Test-time
Idea

action taken: down
original input

+ 0.001 \times \text{sign}(\nabla_x J(\theta, x, y))

action taken: noop
adversarial input

action taken: up
original input

+ 0.441 \times \text{argmax}_{\theta_i} \nabla_x J(\theta, x, y)_{\theta_i}

action taken: down
adversarial input
RL algorithm

• DQN: Deep Q Network
• TRPO: Trust Region Policy Optimization. Use a whole trajectory rollout on stochastic policy, penalized by the KL divergence between old and new policy.
• A3C: Asynchronous Advantage Actor-Critic. Asynchronous gradient descent on stochastic policy.
Apply FGSM on Policy

- FGSM directly use $J(\theta, x, y)$ to generate perturbation.
- In this case, just assume the policy $\pi_\theta$ generated is good, and our good is to flip the predicted policy.
- Note: Use an extra softmax on the DQN
Different norms

\[
\eta = \begin{cases} 
\epsilon \text{ sign}(\nabla_x \mathcal{J}(\theta, x, y)) & \text{for constraint } \|\eta\|_\infty \leq \epsilon \\
\epsilon \sqrt{d} \times \frac{\nabla_x \mathcal{J}(\theta, x, y)}{\|\nabla_x \mathcal{J}(\theta, x, y)\|_2} & \text{for constraint } \|\eta\|_2 \leq \|\epsilon 1_d\|_2 \\
\text{maximally perturb highest-impact dimensions with budget } \epsilon d & \text{for constraint } \|\eta\|_1 \leq \|\epsilon 1_d\|_1 
\end{cases}
\]
Figure 2: Comparison of the effectiveness of $\ell_\infty$, $\ell_2$, and $\ell_1$ FGSM adversaries on four Atari games trained with three learning algorithms. The average return is taken across ten trajectories. Constraint on FGSM perturbation: $\ell_\infty$ norm, $\ell_2$ norm, $\ell_1$ norm.
Transferability

Figure 3: Transferability of adversarial inputs for policies trained with A3C. Type of transfer: ▢ algorithm □ policy □ none
Vulnerability of Deep Reinforcement Learning to Policy Induction Attacks

Vahid Behzadan and Arslan Munir
International Conference on Machine Learning and Data Mining in Pattern Recognition 2017

• Attack DQN
Threat model

• Priori info: Structure of DQN, reward function $R$
• Attacker: Work on state, not actions
• Magnitude of perturbation is smaller than $\epsilon$
Method

• Use another DQN to simulate DQN
• Use FGSM and JSMA
Algorithm 1: Exploitation Procedure

input : adversarial policy $\pi_{adv}^*$, initialized replica DQNs $Q'$, $\hat{Q}'$, synchronization frequency $c$, number of iterations $N$

1. for observation = 1, $N$ do
2.     Observe current state $s_t$, action $a_t$, reward $r_t$, and resulting state $s_{t+1}$
3.     if $s_{t+1}$ is not terminal then
4.         set $a'_{adv} = \pi_{adv}^*(s_{t+1})$
5.         Calculate perturbation vector $\delta_{t+1} = \text{Craft}(\hat{Q}', a'_{adv}, s_{t+1})$
6.         Update $s_{t+1} \leftarrow s_{t+1} + \delta_{t+1}$
7.         Set $y_t = (r_t + \max_{a'} Q'(s_{t+1} + \delta_{t+1}, a'; \theta'_-))$
8.         Perform SGD on $(y_t - Q'(s_t, a_t, \theta'))^2$ w.r.t $\theta'$
9.     end
10.    Reveal $s_{t+1}$ to target
11.    if observation $\mod c = 0$ then $\theta'_- \leftarrow \theta'$
12. end
Experiment result

Fig. 4: Success rate of crafting adversarial examples for DQN
Transferability
Online Robust Policy Learning in the Presence of Unknown Adversaries
Aaron J. Havens, Zhanhong Jiang, Soumik Sarkar

• Setting: Finite-horizon discounted MDP
• Base Algorithm: TRPO
• Goal: Online mitigation of the adversarial perturbation
Motivation

• In the learning, if the input changed, the expected reward will be different to the actual reward.
• In the online setting, an advantage (expect result – observed return) can be used as a good indicator that whether the input has been perturbed
Method (General idea)

- Create multiple (2) policies
- A master policy can choose one policy from time to time.
- When it detects a change on the input, it will choose a different policy

(b) MLAH framework
Method

• Start with random policies
• For a step, estimate the advantage of both policies

\[ \mathbf{A}_t = [A_{GAE,t-h|\pi_{nom}}, A_{GAE,t-h|\pi_{adv}}] \in \mathbb{R}^2 \]

\[ a_{master,t} = \pi_{*,t} = \arg\max_a \mathbb{E}_{s_t, \pi, m_i} \left[ \sum_{t=0}^T \gamma^t r(s_t, a|m_i) \right] \in \{\pi_{nom}, \pi_{adv}\} \]

• Optimize master policy according to the loss
• Optimize both policies

(b) MLAH framework
Method

Algorithm 1: MLAH

**Input:** \(\pi_{nom}\) and \(\pi_{adv}\) sub-policies parameterized by \(\theta_{nom}\) and \(\theta_{adv}\); Master policy \(\pi_{master}\) with parameter vector \(\phi\).

1. Initialize \(\theta_{nom}\), \(\theta_{adv}\), \(\phi\).
2. for pre-training iterations [optional] do
   3. Train \(\pi_{nom}\) and \(\theta_{nom}\) on only nominal experiences.
5. end
6. for learning life-time do

   for Time steps \(t\) to \(t + T\) do

   7. Compute \(A_t\) over sub-policies (see eq. 4)
   8. select sub-policy to take action with \(\pi_{master}\) using \(A_t\) as observations

9. end
10. Estimate all \(A_{GAE}\) for \(\pi_{nom}\), \(\pi_{adv}\) over \(T\)
11. Estimate all \(A_{GAE}\) for \(\pi_{master}\) over \(T\) with respect to \(A_t\) observations
12. Optimize \(\theta_{nom}\) based on experiences collected from \(\pi_{nom}\)
13. Optimize \(\theta_{adv}\) based on experiences collected from \(\pi_{adv}\)
14. Optimize \(\phi\) based on all experiences with respect to \(A_t\) observations

15. end
Result
Robust Deep Reinforcement Learning with Adversarial Attacks

Pattanaik, Anay, Zhenyi Tang, Shuijing Liu, Gautham Bommanan, and Girish Chowdhary.
In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems

• Even naïve attack can degrade DRL algorithms
• Adversarial Training leads to significant increase in robustness to parameter variations for RL benchmarks
• Target RL algorithm:
  • DDQN
  • DDPG
Perturbation

• Naïve attack: Random perturbation
• Gradient attack: Gradient based perturbation (FGSM)
Adversarial attack

- Use beta distribution to sample noise from some distribution
- Sample multiple times and pick the largest one

Algorithm 2 Naive attack (DDPG)

```
1: procedure NAIVE(Qtarget, U, s, e, n, α, β) ▷ Naive attack function takes trained target
critic network Qtarget, 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
2:      \[ a^* = U(s), Q^* = Qtarget(s, a^*) \] ▷ Determine optimal action and action value function
3:      for \( i = 1 : n \) do ▷ Sample a few times
4:          \[ n_i \sim \text{beta}(\alpha, \beta) - 0.5 \] ▷ Sample noise
5:          \[ s_i = s + e \cdot n_i \] ▷ Possible adversarial state determined by sampled noise
6:          \[ a_{adv} = U(s_i) \] ▷ Determine optimal action in potential adversarial state
7:          \[ Q_{adv} = Q_{target}(s, a_{adv}) \] ▷ Determine the value of potential adversarial action
8:          corresponding to potential adversarial state for current state
9:          if \[ Q_{adv} < Q^* \] then ▷ if the potential adversarial state leads to bad action
10:             \[ Q^* = Q_{adv} \] ▷ Store the value function of that potential bad action
11:             \[ s_{adv} = s_i \] ▷ Store possible adversarial state
12:      else do nothing
13:      end if
14:      end for
15:      return \( s_{adv} \) ▷ Adversarial state
16: end procedure
```
Adversarial training

Algorithm 5 Training with adversarial perturbation (DDQN)

1: **procedure** \textsc{adv train} ($Q^{target}$, $Q$) \hfill $\triangleright$ Gradient based adversarial training method takes pre-trained network
2: \hspace{1cm} \textbf{for} $i = 1 : \text{iterations}$ \textbf{do} \hfill $\triangleright$ Train adversarially for number of timesteps
3: \hspace{2cm} Reset the environment and receive observation
4: \hspace{2cm} \textbf{while} not terminal or not max time steps per episode reached \textbf{do}
5: \hspace{3cm} $s_{adv} = \text{Grad}(Q^{target}, Q, s, \epsilon, n, \alpha, \beta)$ \hfill $\triangleright$ Fool the agent
6: \hspace{3cm} $a = \arg\max_a Q(s_{adv}, a)$ \hfill $\triangleright$ Fooled agent takes action according to behavior policy
7: \hspace{3cm} $s, r = \text{Env}(a, s)$ \hfill $\triangleright$ Environment returns next state and reward corresponding to state $s$ and action $a$
8: \hspace{3cm} Update the weights of network according to DDQN algorithm
9: \hspace{2cm} \textbf{end while}
10: \hspace{1cm} \textbf{end for}
11: \hspace{1cm} \textbf{end procedure}
Adversarial Attack performance
Robust

• Test the adversarial trained agent on a wide range of parameters and compared it to “vanilla” DRL