Attacks on RL models

Soumik Sarkar

Short course on Robust ML: Attacks and Defenses
Some Success Stories of Reinforcement Learning

Kohl and Stone, 2004
Ng et al, 2004
Tedrake et al, 2005
Kober and Peters, 2009

Silver et al, 2014 (DPG)
Lillicrap et al, 2015 (DDPG)
Schulman et al, 2016 (TRPO + GAE)
Levine*, Finn*, et al, 2016 (GPS)
Silver*, Huang*, et al, 2016 (AlphaGo)

Mnih et al 2013 (DQN)
Mnih et al, 2015 (A3C)
Basic RL framework
RL Algorithms Landscape

Policy Optimization
- DFO / Evolution
- Policy Gradients

Dynamic Programming
- Policy Iteration
- Value Iteration
- Q-Learning

Actor-Critic Methods
- modified policy iteration
Deep Reinforcement Learning approaches

• Use deep neural networks to represent
  ✓ Value function
  ✓ Policy
  ✓ Model

• Optimize loss function by stochastic gradient descent

Courtesy: David Silver’s tutorial on deep reinforcement learning
Deep Reinforcement Learning approaches – Value function

Q Networks:

Represent value function by *Q-network* with weights $\mathbf{w}$

$$Q(s, a; \mathbf{w}) \approx Q^*(s, a)$$

Courtesy: David Silver’s tutorial on deep reinforcement learning
Deep Q-Learning playing Atari

- End-to-end learning of values $Q(s, a)$ from pixels $s$
- Input state $s$ is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step

Network architecture and hyperparameters fixed across all games
Test time attack on RL agents

(a) without adversary

(b) with adversary

Courtesy: https://arxiv.org/abs/1710.00814
Remember FGSM

$$\eta = \epsilon \text{ sign } (\nabla_{x} J(\theta, x, y))$$

$J$ is the cross-entropy loss between $y$ and the dist. that places all weights on the highest weighted $y$. 

\[ \text{Dist.} \rightarrow \text{model} \rightarrow \text{input} \rightarrow \text{all classes} \]
Different Norm constraints

\( \xi \) (classical) all pixel

\[ \| \eta \|_p \leq 1 \Rightarrow \eta = \epsilon \text{Sign} \left( \sum_k J(\theta, x, y) \right) \]

\( \eta = \max \text{perturb highest impact dim} \leq 1 \Rightarrow \| \eta \|_2 \leq \frac{1}{\sqrt{d}} \]

\[ \eta = \epsilon \sqrt{d} \cdot \frac{\sum_k J(\theta, x, y)}{\| \sum_k J(\theta, x, y) \|_2^{1/2}} \]
Attack on DQN

Courtesy: https://arxiv.org/abs/1702.02284

Demo of the pong game:
https://www.youtube.com/watch?v=bPkwMIcq2tc
Attack on DQN

$l_1$ is most powerful followed by $l_2$ and $l_\infty$.

Courtesy: https://arxiv.org/abs/1702.02284
Policy Optimization

- Consider control policy parameterized by parameter vector $\theta$

$$\max_{\theta} \mathbb{E} \left[ \sum_{t=0}^{H} R(s_t) | \pi_\theta \right]$$

- Stochastic policy class (smooths out the problem):

$$\pi_\theta(u|s) : \text{probability of action } u \text{ in state } s$$

[Figure source: Sutton & Barto, 1998]
Vanilla Policy Gradient Algorithm

**Algorithm 1** “Vanilla” policy gradient algorithm

Initialize policy parameter $\theta$, baseline $b$

for iteration = 1, 2, ... do

Collect a set of trajectories by executing the current policy
At each timestep in each trajectory, compute
the return $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and
the advantage estimate $\hat{A}_t = R_t - b(s_t)$.
Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$
summed over all trajectories and timesteps.
Update the policy, using a policy gradient estimate $\hat{g}$,
which is a sum of terms $\nabla_\theta \log \pi(a_t | s_t, \theta) \hat{A}_t$

end for

* [Williams, 1992]
What Loss to Optimize?

- Policy gradients

\[ \hat{g} = \hat{E}_t \left[ \nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right] \]

- Can differentiate the following loss

\[ L_{PG}(\theta) = \hat{E}_t \left[ \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right] \]

but don’t want to optimize it too far

- Equivalently differentiate

\[ L_{\text{IS}}(\theta) = \hat{E}_t \left[ \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\text{old}}(a_t \mid s_t)} \hat{A}_t \right] \]

at \( \theta = \theta_{\text{old}} \), state-actions are sampled using \( \theta_{\text{old}} \). (IS = importance sampling)

Just the chain rule: \( \nabla_\theta \log f(\theta)|_{\theta_{\text{old}}} = \frac{\nabla_\theta f(\theta)|_{\theta_{\text{old}}}}{f(\theta_{\text{old}})} = \nabla_\theta \left( \frac{f(\theta)}{f(\theta_{\text{old}})} \right)|_{\theta_{\text{old}}} \)

Courtesy: Deep Reinforcement Learning Bootcamp, Berkeley by John Schulman
Trust Region Policy Optimization

- Define the following trust region update:

\[
\max_{\theta} \quad \mathbb{E}_t \left[ \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right]
\]

subject to \( \mathbb{E}_t [\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta \).

- Also worth considering using a penalty instead of a constraint

\[
\max_{\theta} \quad \mathbb{E}_t \left[ \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right] - \beta \mathbb{E}_t [\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]
\]

- Method of Lagrange multipliers: optimality point of \( \delta \)-constrained problem is also an optimality point of \( \beta \)-penalized problem for some \( \beta \).

- In practice, \( \delta \) is easier to tune, and fixed \( \delta \) is better than fixed \( \beta \)
Trust Region Policy Optimization: Pseudocode

- **Pseudocode:**
  
  ```
  for iteration=1, 2, ... do
      Run policy for $T$ timesteps or $N$ trajectories
      Estimate advantage function at all timesteps
      \[
      \max_{\theta} \sum_{n=1}^{N} \frac{\pi_{\theta}(a_n | s_n)}{\pi_{\theta_{old}}(a_n | s_n)} - \hat{A}_n
      \]
      subject to $\text{KL}_{\pi_{\theta_{old}}}(\pi_{\theta}) \leq \delta$
  end for
  ```

- Can solve constrained optimization problem efficiently by using conjugate gradient
- Closely related to natural policy gradients (Kakade, 2002), natural actor critic (Peters and Schaal, 2005), REPS (Peters et al., 2010)
“Proximal” Policy Optimization (PPO): KL Penalty Version

- Use penalty instead of constraint

\[
\text{maximize } \sum_{n=1}^{N} \frac{\pi_{\theta}(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n - C \cdot \text{KL}_{\pi_{\theta_{\text{old}}}}(\pi_{\theta})
\]

- Pseudocode:

  for iteration=1,2,… do
  Run policy for \( T \) timesteps or \( N \) trajectories
  Estimate advantage function at all timesteps
  Do SGD on above objective for some number of epochs
  If KL too high, increase \( \beta \). If KL too low, decrease \( \beta \).
  end for

- \( \approx \) same performance as TRPO, but only first-order optimization

Courtesy: Deep Reinforcement Learning Bootcamp, Berkeley by John Schulman
FGSM attack on TRPO

\[ \ell_\infty \text{-norm} \quad \ell_2 \text{-norm} \quad \ell_1 \text{-norm} \]

\( l_1 \) most effective followed by \( l_2 \) and \( l_\infty \)

TRPO seems to be more robust compared to DQN

Courtesy: https://arxiv.org/abs/1702.02284
Actor-Critic algorithm: Bringing Value and Policy together

- Faster, simpler, more robust, generate better scores
- DRL moves beyond DQNs

- Estimate value function $Q(s, a, w) \approx Q^\pi(s, a)$
- Update policy parameters $u$ by stochastic gradient ascent

\[
\frac{\partial l}{\partial u} = \frac{\partial \log \pi(a|s, u)}{\partial u} \frac{\partial Q(s, a, w)}{\partial u} Q(s, a, w)
\]

or

\[
\frac{\partial l}{\partial u} = \frac{\partial Q(s, a, w)}{\partial a} \frac{\partial a}{\partial u}
\]
Asynchronous Advantage Actor-Critic (A3C) algorithm

- Estimate state-value function
  \[ V(s, v) \approx \mathbb{E}[r_{t+1} + \gamma r_{t+2} + ... | s] \]
- Q-value estimated by an \( n \)-step sample
  \[ q_t = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}, v) \]
- Actor is updated towards target
  \[ \frac{\partial l_u}{\partial u} = \frac{\partial \log \pi(a_t|s_t, u)}{\partial u} (q_t - V(s_t, v)) \]
- Critic is updated to minimise MSE w.r.t. target
  \[ l_v = (q_t - V(s_t, v))^2 \]
FGSM attack on A3C

Graphs showing the average return for different tolerance values ($\epsilon$) for Chopper Command, Pong, Seaquest, and Space Invaders, with $\ell_\infty$-norm, $\ell_2$-norm, and $\ell_1$-norm represented.

Suggested ties:

TRPO > A3C > DQN
Black box attacks

Transferability across policies

Adversary knows
1) true environment
2) true algo & hyperparams
3) network architecture (and the initializer)

Transferability across algorithms

Do not know (2) true algo & hyperparams
Black box attacks on A3C

Figure 3: Transferability of adversarial inputs for policies trained with A3C. Type of transfer: 
- algorithm
- policy
- none
Black box attacks on TRPO

Figure 4: Transferability of adversarial inputs for policies trained with TRPO. Type of transfer:  
algorithm □ policy □ none

IOWA STATE UNIVERSITY
Figure 5: Transferability of adversarial inputs for policies trained with DQN. Type of transfer: algorithm, policy, none