UC San Diego

Introduction to Deep Reinforcement Learning

Model-free Methods

Zhiao Huang

Markov Decision Process

Reinforcement Learning is modeled as a Markov Decision Process S, A, P_a, R_a

- S is the state space and A is the action space
- $p(s'|s, a)$ is the probability of transition (at time t) from state s to state s' under action a
- $R(s, a, s')$ is the immediate reward received after transitioning from state s to s', under action a

RL Problem

Give an MDP, find **policy** π to maximize the expected future reward if we sample trajectory τ based on the policy in the environment:

$$\max_{\pi} E_{\tau \sim p_{\pi}(\tau)}\left[R_{\gamma}(\tau)\right]$$

Value Function

- Value function

$$V^{\pi}(s) = E_{\tau \sim p_{\pi}(\tau|s)}[R(\tau)]$$

- The expected reward if the agent follows the policy π and is currently at the state s

Q function

- Q function

$$Q^{\pi}(s,a) = E_{s\prime \sim p(s\prime|s,a)} R(s,a,s') + \gamma V^{\pi}(s')$$

$$\Rightarrow V^{\pi}(s) = \underset{a \sim \pi(s)}{E} Q^{\pi}(s,a)$$

- Expected reward from state s if we select the action a

Bellman Equations

- Policy π is the optimal policy if and only V^{π} satisfy the bellman equations for all s

$$V^{\pi}(s) = \max_a E_{s' \sim p(s,a)}[R(s,a,s') + \gamma V^{\pi}(s')] = \max_a Q(s,a)$$

  - We use Q^* and V^* to represent the Q and V function of the optimal policy π^*

Value Iteration Algorithm

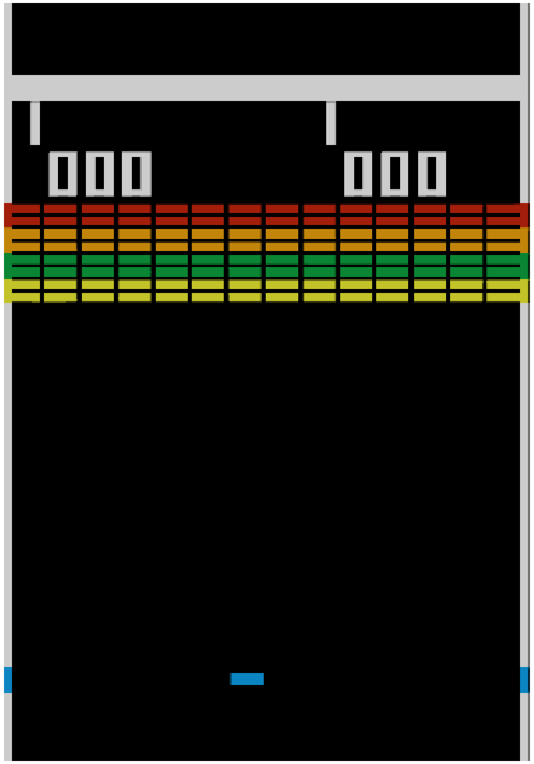
- Initialize V
- Repeat
  - For each state $s \in S$ do
    $V'^{(s)} \leftarrow \max_a E_{s' \sim p(s'|s,a)}[R(s,a,s') + \gamma V(s')]$
  - Stop if $\|V - V'\|_\infty \leq \epsilon$
  - $V \leftarrow V'$

What's the trouble?

- The state space is high-dimensional
- The model $p(s'|s,a)$ and the reward $R(s,a,s')$ are unknown

High-dimensional State Space

- The state space S is high-dimensional
- We can't use an array to store $V(s)$ for all states



The state can be represented by an image

What's the trouble?

- The state space is high-dimensional

- The model $p(s'|s,a)$ and the reward $R(s,a,s')$ are unknown
  - We can only sample trajectories from the environment

What's the trouble?

- The model $p(s'|s,a)$ and the reward $R(s,a,s')$ are unknown

For real robot, we don't know $p(s'|s,a)$ precisely

Deep Q Learning

- Deep neural networks for high-dimensional state space
- Temporal difference learning with the replay buffer

Deep Q Network

- Model $Q^*(s, a)$ with a deep neural network $Q_\theta(s, a)$

Deep Q Network

- Assuming a finite action space, V^* and π^* can be computed from $Q^* = Q_\theta$

$$V^*(s) = \max_a \mathrm{Q}^*(s,a)$$
$$\pi^*(s) = \operatorname*{argmax}_a \mathrm{Q}^*(s,a)$$

Bellman Equations for Q Networks

- The Q network is optimal if and only if it satisfies the Bellman Equations for all (s, a) pairs:

$$Q_\theta(s, a) = E_{s' \sim p(s'|s,a)}[R(s, a, s') + \gamma \max_{a'} \mathrm{Q}_\theta(s', a')]$$

Training Deep Q Network

- Finding θ such that

$$Q_\theta(s,a) = E_{s' \sim p(s'|s,a)}[R(s,a,s') + \gamma \max_{a'} \mathrm{Q}_\theta(s',a')]$$

is equivalent to finding θ to minimize

$$\sum_{s \in S, a \in A} \left\| Q_\theta(s,a) - E_{s' \sim p(s'|s,a)}[R(s,a,s') + \gamma \max_{a'} \mathrm{Q}_\theta(s',a')] \right\|^2$$

Training Deep Q Network

- However, the objective is still intractable

$$\sum_{s \in S, a \in A} \left\| Q_\theta(s,a) - E_{s' \sim p(s\prime|s,a)}\left[R(s,a,s') + \gamma \max_{a'} \mathrm{Q}_\theta(s',a')\right] \right\|^2$$

- S is a high-dimensional space
- $p(s'|s,a)$ is unknown

Temporal Difference Learning

- However, the objective is still intractable

$$\sum_{s\in S, a\in A} \left\| Q_\theta(s,a) - E_{s'\sim p(s'|s,a)}[R(s,a,s') + \gamma \max_{a'} Q_\theta(s',a')] \right\|^2$$

- TD-learning approximate the above objective with sampled transitions from the environment

$$L(\theta) = E_{(s,a,s')\sim \mathrm{Env}}[TD_\theta(s,a,s')]$$

where

$$TD_\theta(s,a,s') = \left\| Q_\theta(s,a) - \left[R(s,a,s') + \gamma \max_{a'} Q_\theta(s',a')\right] \right\|^2$$

Replay Buffer

- Our goal is to train Q_θ with the loss

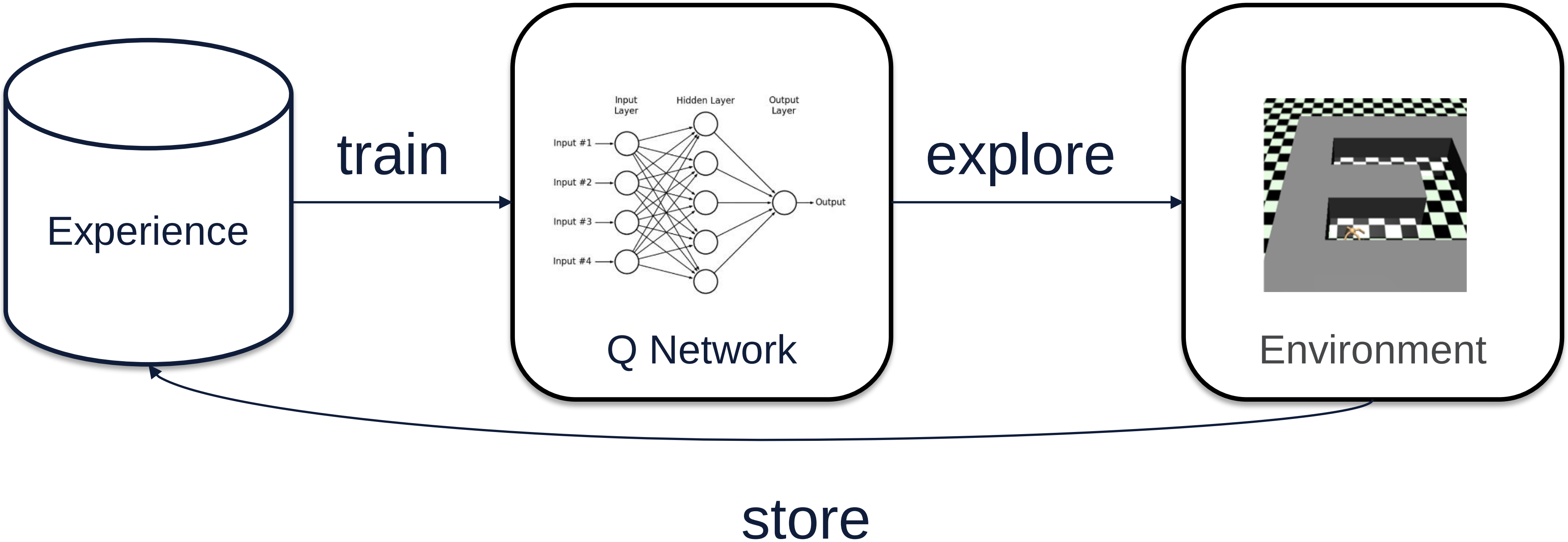
$$L(\theta) = E_{(s,a,s')\sim\text{Env}}[TD_\theta(s,a,s')]$$

- We want to sample most transitions (s,a,s') from the environment
  - At least we want to sample the transitions to cover the optimal solution!

Replay Buffer

- **Exploration** Replay buffer stores all transitions sampled from the environment with the current network

- **Exploitation** Sample transitions from the replay buffer to update the network



Exploration

- In Q learning, we use **ϵ-greedy** to explore the environment. For the current state s
  - With probability ϵ, sample a random action
  - With probability $1 - \epsilon$, choose the best action according to $\max_a Q_\theta(s, a)$

Exploration (cont.)

- **ϵ-greedy** is a trade-off:
  - Random actions to cover most states
  - Sample more on the optimal trajectories for more accurate solution

Exploitation

- To update the network, we simply sample a batch of transitions from the replay buffer to train the network by gradient descent

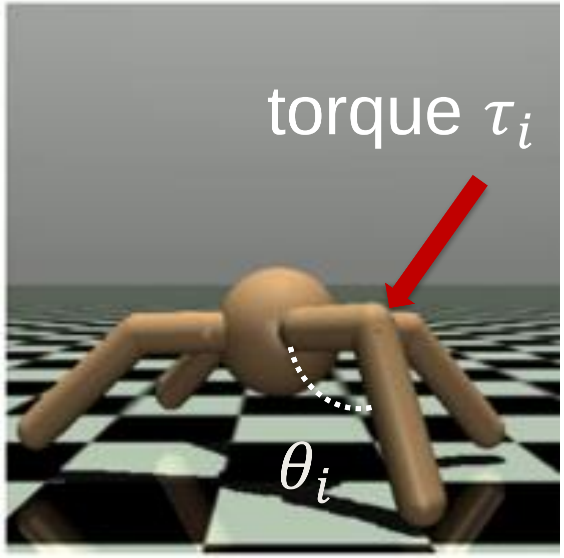
$$\nabla_\theta L(\theta) = E_{(s,a,s')\sim \text{Replay Buffer}}[\nabla_\theta TD_\theta(s,a,s')]$$

Deep Q Learning

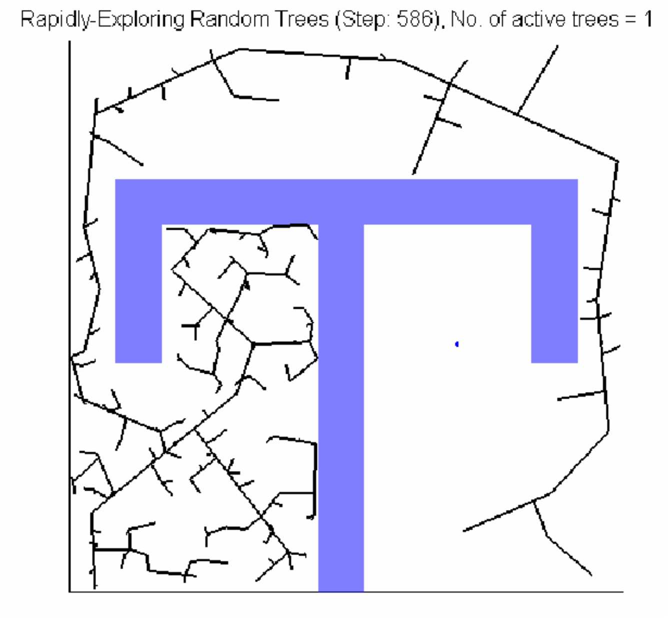
- Initialize replay buffer D and Q network Q_θ
- Sample the initial state $s_0 \sim p(s_0)$
- Repeat
  - Let s be the current state
  - With probability ϵ select a random action a
  - Otherwise select $a = \text{argmax}_a \, Q_\theta(s, a)$
  - Execute a in the environment and observe the reward r and the next state s'
  - Store transitions (s, a, s') in D
  - Sample random minibatch of transitions (s_j, a_j, s_j') from D
  - Set $y_j = r_j + \gamma \max_{a'} \mathrm{Q}_\theta(s_j', a')$
  - Perform a gradient descent step on the TD loss

$$\sum_j \left(y_j - Q_\theta(s_j, a_j)\right)^2$$

Continuous Action Space?



Robot Control



Navigation

Continuous Action Space?

- Finding policy from the Q function becomes non-trivial

$$\pi(s) = \mathrm{argmax}_a Q_\theta(s, a)$$

Deep Deterministic Policy Gradient

- Use a network $a = \pi_\phi(s)$ to model the policy, and we maximize (end to end trainable)

$$\max_\phi E_s\left[Q_\theta\left(s, \pi_\phi(s)\right)\right]$$

- Q Differentiable

Deep Q Learning

- Initialize replay buffer D and Q network Q_θ
- Sample the initial state $s_0 \sim p(s_0)$
- Repeat
    - Let s be the current state
    - With probability ϵ select a random action a
    - Otherwise select $a = \text{argmax}_a\, Q_\theta(s, a)$
    - Execute a in the environment and observe the reward r and the next state s'
    - Store transitions (s, a, s') in D
    - Sample random minibatch of transitions (s_j, a_j, s_j') from D
    - Set $y_j = r_j + \gamma \max_{a'} \mathrm{Q}_\theta\left(s_j', a'\right)$
    - Perform a gradient descent step on the TD loss

$$\sum_j \left(y_j - Q_\theta\left(s_j, a_j\right)\right)^2$$

Deep Deterministic Policy Gradient (simplified)

- Initialize replay buffer D, Q network Q_θ, and policy network π_ϕ
- Sample the initial state $s_0 \sim p(s_0)$
- Repeat
  - Let s be the current state
  - With probability ϵ select a random action a
  - Otherwise select $a = \pi_\phi(s)$
  - Execute a in the environment and observe the reward r and the next state s'
  - Store transitions (s, a, s') in D
  - Sample random minibatch of transitions (s_j, a_j, s_j') from D
  - Set $y_j = r_j + \gamma \mathrm{Q}_\theta\left(s_j', \pi_\phi(s_j')\right)$
  - Perform a gradient descent step on the TD loss

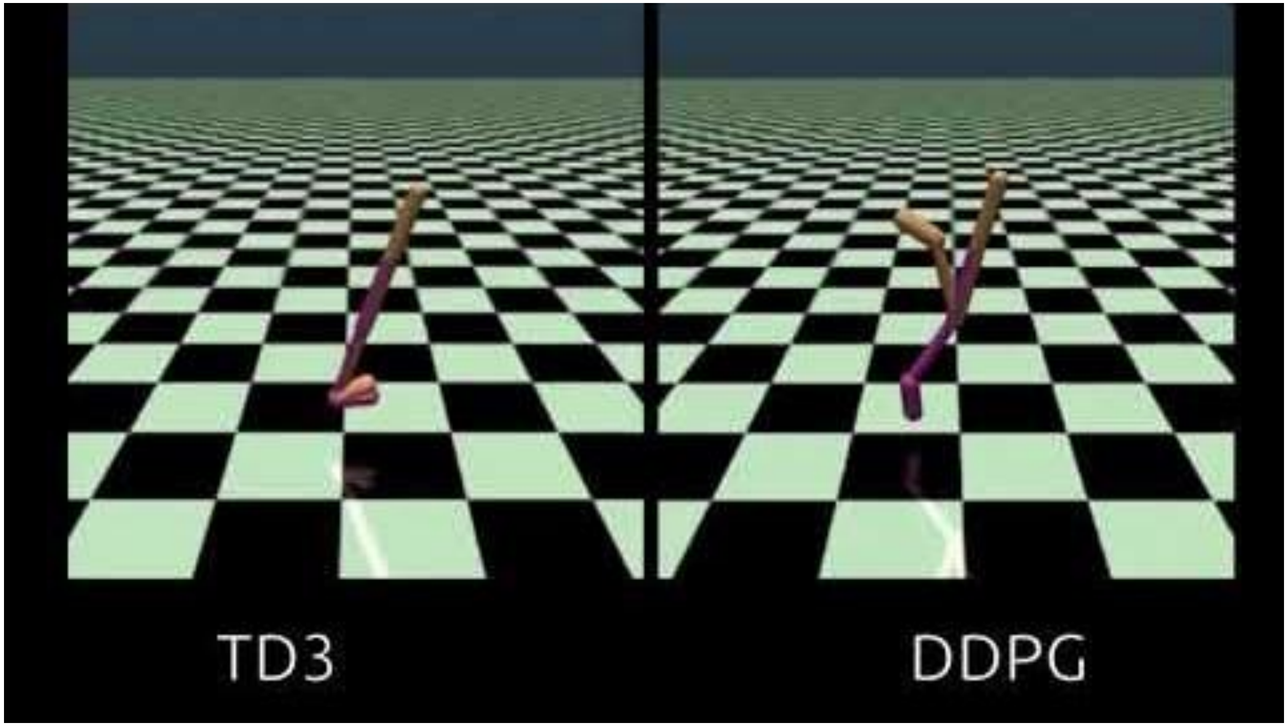
$$\sum_j \left(y_j - Q_\theta\left(s_j, a_j\right)\right)^2$$

  - Perform a gradient ascent with respect to θ to maximize

$$\sum_j Q_\theta(s_j, \pi_\phi(s_j))$$

Performance

- Performance of TD3 (twin delayed DDPG) and DDPG



Off-policy Methods

- In DQN and DDPG, we maintain a replay buffer and optimize $Q_\theta \approx Q^*$ with Bellman equations

- DQN/DDPG are **off-policy** methods

  - We can use any policy that can explore all states to sample transitions from the replay buffer

On-policy Methods

- On-policy methods (e.g. Policy Gradient):
  - Model the policy π_θ
  - Update π_θ by sampling π_θ in the environment

Policy Gradient

- **Policy Gradient** optimizes policy with the sampled trajectories directly

Policy Gradient

- Recall the objective of the policy π_θ is to minimize

$$J(\theta) = E_{\tau\sim p(\pi_\theta)}[R(\tau)] = \int_\tau R(\tau)\, p(\tau|\theta) d\tau$$

- Its gradient is

$$\nabla_\theta J(\theta) = \nabla_\theta \int_\tau R(\tau)\, p(\tau|\theta) d\tau = \int_\tau R(\tau)\, \nabla_\theta p(\tau|\theta) d\tau$$

Policy Gradient

$$\nabla \log \mathrm{f(x)} = \frac{\nabla f(x)}{f(x)} \Rightarrow \nabla f(x) = f(x) \nabla \log \mathrm{f(x)}$$

$$\nabla_\theta J(\theta) = \nabla_\theta \int_\tau R(\tau)\, p(\tau|\theta) d\tau = \int_\tau R(\tau)\, \nabla_\theta p(\tau|\theta) d\tau$$

$$= \int_\tau R(\tau)\, p(\tau|\theta) \nabla_\theta \log p(\tau|\theta)\, d\tau$$

$$= E_\tau [R(\tau) \nabla \log p(\tau|\theta)]$$

Policy Gradient

- The gradient of this objective is

$$\nabla J(\theta) = E_\tau[R(\tau)\nabla \log p(\tau|\theta)]$$

Where

$$\log p(\tau|\theta) = \log p(s_0) + \sum_{i=0\ldots\infty} \log \pi_\theta(a_i|s_i) + \log p(s_{i+1}|s_i, a_i)$$

Policy Gradient

- The gradient of this objective is

$$\nabla_\theta J(\theta) = E_\tau[R(\tau)\nabla \log p(\tau|\theta)]$$

$$\nabla \log p(\tau|\theta) = \nabla \log p(s_0) + \sum_{i=0\ldots\infty} \nabla \log \pi_\theta(a_i|s_i) + \nabla \log p(s_{i+1}|s_i, a_i)$$

$$\boxed{\nabla_\theta J(\theta) = E_\tau[R(\tau) \sum_{i=0\ldots\infty} \nabla \log \pi_\theta(a_i|s_i)]}$$

Intuitive explanation

- High probability for high reward actions.

$$\boxed{\nabla_\theta J(\theta) = E_\tau[R(\tau) \sum_{i=0\ldots\infty} \nabla \log \pi_\theta(a_i|s_i)]}$$

Policy Gradient in Python

- $\pi_\theta(a|s) = N(\mu_\theta(s), \sigma_\theta(s))$ is the gaussian distribution

```python
from torch.distributions import Normal

def PolicyGradient(policy, s, a, R):
    normal_distribution: Normal = policy(s)
    return normal_distribution.log_prob(a) * R.detach()
```

$$\boxed{\nabla_\theta J(\theta) = E_\tau\left[R(\tau) \sum_{i=0\ldots\infty} \nabla \log \pi_\theta(a_i|s_i)\right]}$$

We can compute it directly

Policy Gradient

- Initialize a policy network π_θ
- Repeat
  - Sample trajectories $\{\tau_{1:T}^{i}\}_{i \leq N}$
  - Compute the gradient ∇J by policy gradient

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_i [R(\tau^i) \sum_{j=1\ldots\mathrm{T}} \nabla \log \pi_\theta (a_j^i | s_j^i)]$$

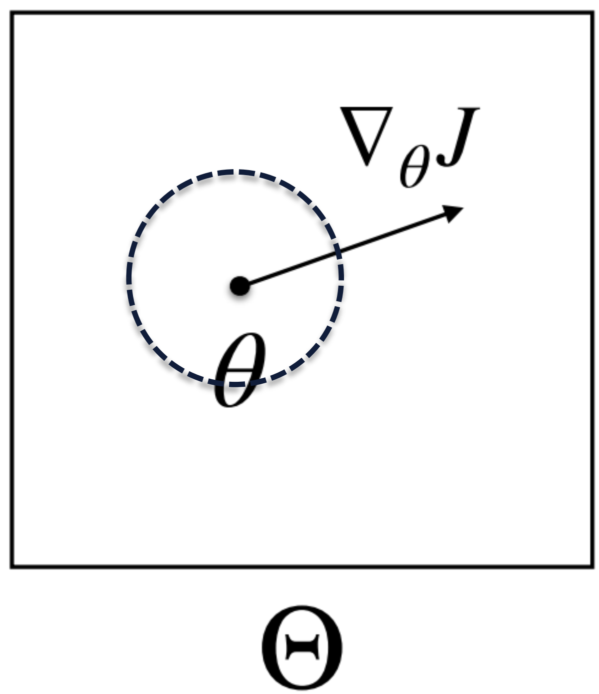
  - Update

$$\theta \leftarrow \theta + \alpha \nabla J(\theta)$$

How to choose the learning rate?

Trust-Region Policy Optimization

- Policy gradient only points at a local direction
- TRPO aims to identify a neighborhood that guarantees improvement (at least in theory)



Gradient Descent Recap

- Gradient update $x = x - \alpha \nabla f(x)$ minimizes the following quadratic approximation regularized by the l_2 norm

$$\mathrm{x} - \alpha \nabla f(x) = \underset{x'}{\operatorname{argmax}} f(x) + \nabla f(x)^T (x' - x) + \frac{1}{2\alpha} \|x' - x\|_2^2$$

Trust-Region Policy Optimization

- In TRPO, we optimize for a better approximation

$$\pi' = \underset{\pi'}{arg\max}\, E_{\tau \sim p_{\pi}(\tau)}[\sum_{i=0}^{\infty} \gamma^{i} A^{\pi}\,(s_i, a_i)]$$

regularized by KL-divergence

$$s.t.\, E_{s\sim\pi}[D_{KL}(\pi(a|s), \pi'(a|s))] \leq \delta$$

  - Here $\pi' = \pi(\theta')$
  - $J^{\pi} = J^{\pi_{\theta}} = J(\theta)$ is the expected reward of policy π_{θ}
  - $\mathrm{A}^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$ is the advantage function

Advantage Function

- The **advantage function**

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$$

- is the advantage of choosing action a compared with the current policy

Policy Evaluation

- We can prove that the following formula estimates the expected reward of policy π' with the advantage function A^{π}:

$$J^{\pi'} = J^{\pi} + E_{\tau' \sim p_{\pi'}(\tau)}[\sum_{i=0}^{\infty} \gamma^{i} A^{\pi}(s_i, a_i)]$$

However we don't want to sample π' every time to find π'

Approximation

- We can estimate how policy values change for $\pi \rightarrow \pi'$

$$J^{\pi'} = J^{\pi} + E_{\tau' \sim p_{\pi'}(\tau)}[\sum_{i=0}^{\infty} \gamma^i A^{\pi} (s_i, a_i)]$$

- Instead, TRPO works with the local approximation

$$J^{\pi'} \approx L_{\pi}(\pi') = J^{\pi} + E_{\tau' \sim p_{\pi}(\tau)}[\sum_{i=0}^{\infty} \gamma^i A^{\pi} (s_i, a_i)]$$

- When π' and π are close, L_{π} approximates $J^{\pi'}$ with clear bounds.

Trust Region

- When π' and π are close, $L_\pi(\pi')$ approximates $J^{\pi'}$
- TRPO constrains the expectation of KL distance between π' and π to be small

$$E_{s\sim\pi}[(\pi(a|s), \pi'(a|s))] \le \delta$$

Trust Region Policy Optimization

After several derivation, we formulates the policy optimization as a constrained optimization problem

$$\max_{\pi'} E_{s,a\sim\pi}\left[\frac{\pi'(a|s)}{\pi\ (a|s)} A^{\pi}(s,a)\right]$$

$$s.t. E_{s\sim\pi}[D_{KL}(\pi(a|s), \pi'(a|s))] \leq \delta$$

Sampling-based Estimation

- For a "rollout" $s_0, a_0, s_1, a_1, \ldots, a_{T-1}, s_T$, we can estimate the Q function

$$Q^\pi(s_t, a_t) \approx \hat{Q}^\pi(s_t, a_t)$$
$$= \sum_{i=t}^{T-1} \gamma^{i-t} R(s_i, a_i, s_{i+1}) + \gamma^{T-t} V^\pi(s_T)$$

- The advantage function

$$\hat{A}^\pi(s_t, a_t) = \hat{Q}^\pi(s_t, a_t) - V^\pi(s_t)$$

Value Network

- $V^{\pi}(s)$ is not necessary

$$
\begin{aligned}
E_{a\sim\pi}\left[\frac{\pi'(a|s)}{\pi\ (a|s)}A_{\pi}(s,a)\right] &= \sum_{a}\pi'(a|s)(Q^{\pi}(s,a)-V^{\pi}(s)) \\
&= \sum_{a}\pi'(a|s)Q^{\pi}(s,a)-V^{\pi}(s)\sum_{a}\pi'(a|s) \\
&= E_{a\sim\pi}\left[\frac{\pi'(a|s)}{\pi\ (a|s)}Q^{\pi}\ (s,a)\right]
\end{aligned}
$$

Value Network

- $V^{\pi}(s)$ is not necessary
- But $A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$ is more stable than $Q^{\pi}(s,a)$ along in practice
- Fit a value network (like Q Network before)

$$\min_{\phi} \sum_{s,a\sim\pi} \left\| V_{\phi} - \hat{Q}^{\pi}(s,a) \right\|^2$$

Value Network

- We can add the omitted remainder back

$$\hat{Q}^{\pi}(s,a) = \sum_{i=t}^{T-1} \gamma^{i-t} R(s_i, a_i, s_{i+1}) + \gamma^{T-t} V_{\phi}(S_T)$$

- The advantage function

$$\hat{A}^{\pi}(s_t, a_t) = \hat{Q}^{\pi}(s_t, a_t) - V_{\phi}\,(s_t)$$

Constrained Optimization

- TRPO solves the following constrained optimization problem

$$\max_{\pi'} E_{s,a\sim\pi}\left[\frac{\pi'(a|s)}{\pi\,(a|s)}\hat{A}^{\pi}(s,a)\right]$$

$$s.t.\, E_{s\sim\pi}[D_{KL}(\pi(a|s),\pi'(a|s))] \leq \delta$$

- Constrained Optimization is easier to tune than the corresponding soft one

Proximal Policy Optimization

- PPO solves the following **unconstrained** optimization problems

$$\max_{\pi'} E_{(s,a)}\left[\min(r\hat{A}^{\pi}, \mathrm{clip}(r, 1-\epsilon, 1+\epsilon)\hat{A}^{\pi})\right]$$

$$\text{where } r(s,a) = \frac{\pi'(a|s)}{\pi\,(a|s)}$$

Proximal Policy Optimization

- Ignoring the objective that push $\pi'(s, a)$ beyond $[(1-\epsilon)\pi(s, a), (1+\epsilon)\pi(s, a)]$

$$\max_{\pi'} E_{(s,a)}\left[\min(r\hat{A}^{\pi}, \text{clip}(r, 1-\epsilon, 1+\epsilon)\hat{A}^{\pi})\right]$$