Lecture 5: Policy Optimization I

Bolei Zhou

https://github.com/zhoubolei/introRL

April 15, 2020

Today's Plan

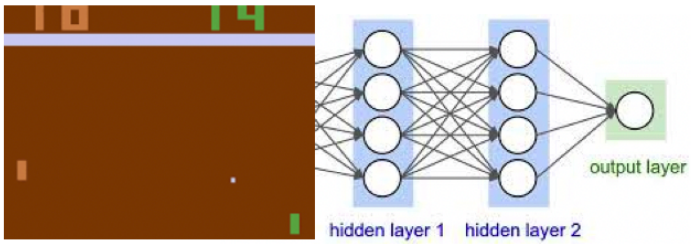
1. Policy-based reinforcement learning
2. Monte-Carlo policy gradient
3. Reduce the variance of policy gradient
4. Actor-critic

Value-based RL versus Policy-based RL

1. Deterministic policy is generated directly from the value function using greedy $a_t = \arg\max_a Q(a, s_t)$
2. Instead we can parameterize the policy function as $\pi_\theta(a|s)$ where θ is the learnable policy parameter and the output is a probability over the action set

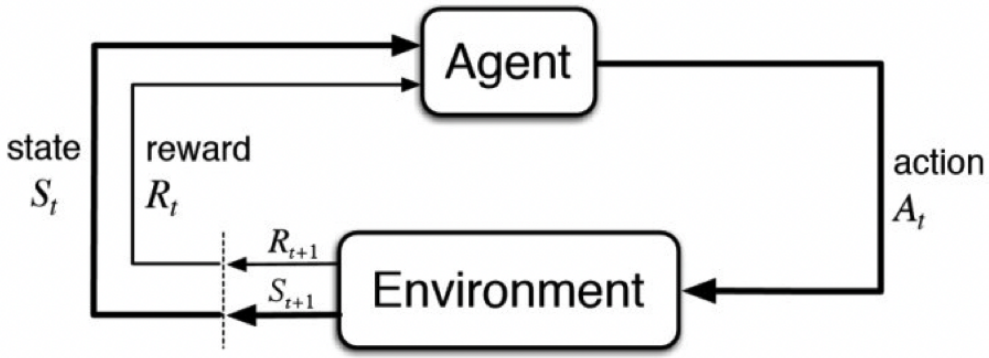
Policy Optimization

1. Action from the policy is all we need then let's optimize the policy directly



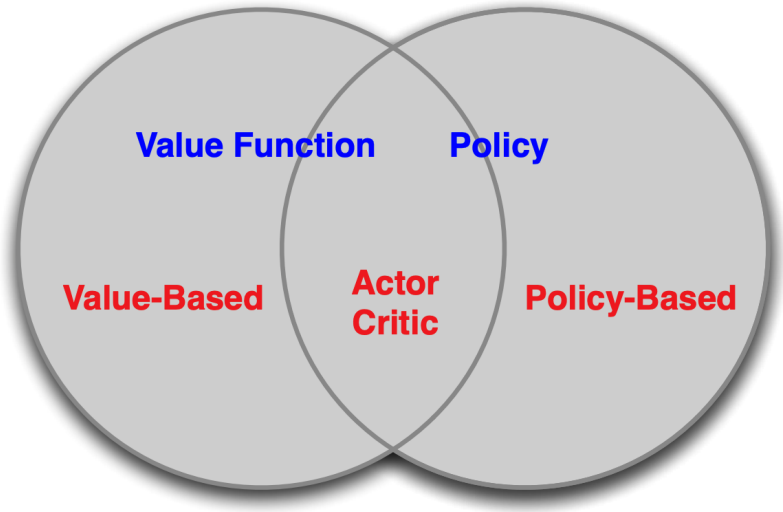
Probability of moving up

$$\pi_\theta(A_t|S_t)$$



Value-based RL versus Policy-based RL

1. Value-based RL
   1. to learn value function
   2. implicit policy based on the value function
2. Policy-based RL
   1. no value function
   2. to learn policy directly
3. Actor-critic
   1. to learn both policy and value function



Advantages of Policy-based RL

1. Advantages:
   1. better convergence properties: we are guaranteed to converge on a local optimum (worst case) or global optimum (best case)
   2. Policy gradient is more effective in high-dimensional action space
   3. Policy gradient can learn stochastic policies, while value function can't
2. Disadvantages:
   1. typically converges to a local optimum
   2. evaluating a policy has high variance

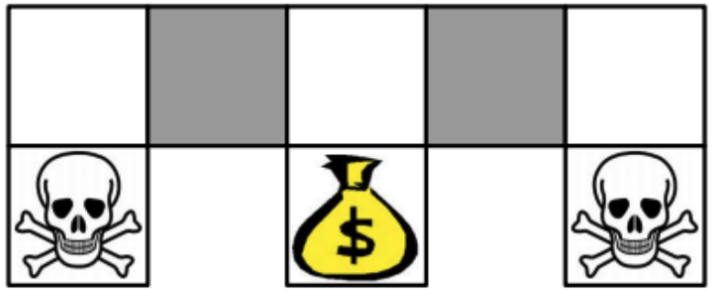
Two Types of Policies

1. Deterministic: given a state, the policy returns a certain action to take
2. Stochastic: given a state, the policy returns a probability distribution of the actions (e.g., 40% chance to turn left, 60% chance to turn right) or a certain Gaussian distribution for continuous action

Example: Rock-Paper-Scissors

1. Two-player game
2. What is the best policy?
   1. A deterministic policy is easily beaten
   2. Uniform random policy is the optimal (Nash equilibrium)

Example: Aliased Gridworld



1. The agent cannot differentiate the two grey states as they look the same to the agent
2. Considers the following features (for all N, E, S, W)

$$\psi(s,a) = [\mathbf{1}(\text{wall to N, a} = \text{move E}), \mathbf{1}(\text{wall to S, a} = \text{move W}), ...]$$

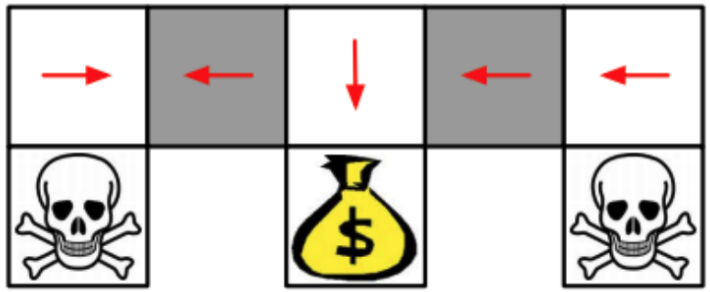
3. If it is value-based RL, value function approximation as:

$$Q_\theta(s,a) = f(\psi(s,a),\theta)$$

4. If it is policy-based RL, policy function approximation as:

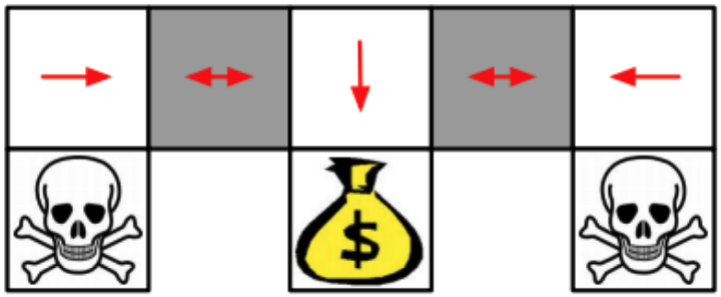
$$\pi_\theta(s,a) = g(\psi(s,a),\theta)$$

Example: Aliased Gridworld



1. Value-based RL learns a deterministic policy, e.g., greedy
2. Because of the aliasing (the agent cannot differentiate two states), an optimal deterministic policy from value-based RL will either
   1. move W in both grey states (shown by red arrows)
   2. move E in both grey states
3. Either way, 50% chance will get stuck

Example: Aliased Gridworld



1. Policy-based RL can learn the optimal stochastic policy
2. An optimal stochastic policy will randomly move E or W in two grey states

$$\pi_\theta(\text{wall to N and W, move E}) = 0.5$$
$$\pi_\theta(\text{wall to N and W, move W}) = 0.5$$

3. For any starting point, it will reach the goal state in a few steps with high probability

Objective of Optimizing Policy

1. Objective: Given a policy approximator $\pi_\theta(s, a)$ with parameter θ, find the best θ
2. How do we measure the quality of a policy π_θ?
3. In episodic environments we can use the start value

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

4. In continuing environments
   1. we can use the average value

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

   2. or the average reward per time-step

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) R(s, a)$$

   where d^{π_θ} is stationary distribution of Markov chain for π_θ

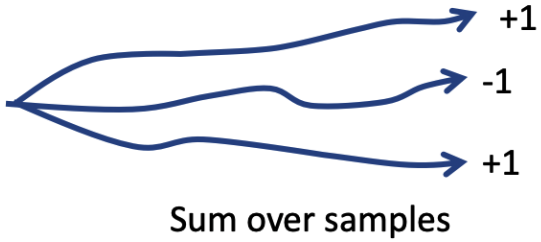
Objective of Optimizing Policy

1. The value of the policy is defined as

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\sum_t r(s_t, a_t^\tau)]$$

$$\approx \frac{1}{m} \sum_m \sum_t r(s_{t,m}, a_{t,m})$$

It is the same as the value function we defined in the value-based RL



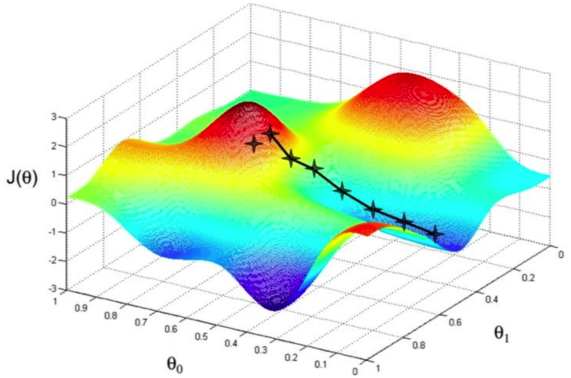
   1. τ is a trajectory sampled from the policy function π_θ
2. The goal of policy-based RL

$$\theta^* = \arg\max_\theta \mathbb{E}_{\tau \sim \pi_\theta}[\sum_t r(s_t, a_t^\tau)]$$

Objective of Optimizing Policy

1. Policy-based RL is an optimization problem that find θ that maximizes $J(\theta)$
2. If $J(\theta)$ is differentiable, we can use gradient-based methods:
   1. gradient ascend
   2. conjugate gradient
   3. quasi-newton
3. If $J(\theta)$ is non-differentiable or hard to compute the derivative, some derivative-free black-box optimization methods:
   1. Cross-entropy method (CEM)
   2. Hill climbing
   3. Evolution algorithm

Policy Optimization using Derivative



1. Consider a function $J(\theta)$ to be any policy objective function
2. Goal is to find parameter θ^* that maximizes $J(\theta)$ by ascending the gradient of the policy,w.r.t parameter θ

$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

3. Adjust θ in the direction of the gradient, where α is step-size
4. Define the gradient of $J(\mathbf{w})$ to be

$$\nabla_\theta J(\theta) = (\frac{\partial J(\theta)}{\partial \theta_1}, \frac{\partial J(\theta)}{\partial \theta_2}, ..., \frac{\partial J(\theta)}{\partial \theta_n})^T$$

Policy Optimization using Derivative-free Methods

1. Sometimes we cannot compute the derivative, i.e., $\nabla_\theta J(\theta)$
2. Derivative-free methods:
    1. Cross Entropy Method (CEM)
    2. Finite Difference

Derivative-free Method: Cross-Entropy Method

1. $\theta^* = \arg\max J(\theta)$
2. Treat $J(\theta)$ as a black box score function (not differentiable)

Algorithm 1 CEM for black-box function optimization

for iter $i = 1$ to N **do**
  $\mathcal{C} = \{\}$
  for parameter set $e = 1$ to N **do**
    sample $\theta^{(e)} \sim P_{\mu^{(i)}}(\theta)$
    execute roll-outs under $\theta^{(e)}$ to evaluate $J(\theta^{(e)})$
    store $(\theta^e, J(\theta^{(e)}))$ in $\mathcal{C}$
  end for
  $\mu^{(i+1)} = \arg\max_\mu \sum_{k \in \hat{\mathcal{C}}} \log P_\mu(\theta^{(k)})$
  where $\hat{\mathcal{C}}$ are the top 10% of $\mathcal{C}$ ranked by $J(\theta^{(e)})$
end for

3. Example of CEM for a simple RL problem:`https://github.com/cuhkrlcourse/RLexample/blob/master/my_learning_agent.py`

Approximate Gradients by Finite Difference

1. To evaluate policy gradient of $\pi_\theta(s, a)$
2. For each dimension $k \in [1, n]$
   1. estimate *kth* partial derivative of objective function by perturbing θ by a small amount ϵ in *kth* dimension

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

   where u_k is unit vector with 1 in *kth* component, 0 else where
3. uses n evaluations to compute policy graident in total n dimensions
4. though noisy and inefficient, but works for arbitrary policies, even if policy is not differentiable.

Computing the Policy Gradient Analytically

1. Assume policy π_θ is differentiable whenever it is no-zero
2. and we can compute the gradient $\nabla_\theta \pi_\theta(s, a)$
3. Likelihood ratios exploit the following tricks

$$\begin{aligned}\nabla_\theta \pi_\theta(s, a) =& \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} \\ =& \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)\end{aligned}$$

4. The score function is $\nabla_\theta \log \pi_\theta(s, a)$

Policy Example: Softmax Policy

1. Simple policy model: weight actions using linear combination of features $\phi(s,a)^T\theta$
2. Probability of action is proportional to the exponetiated weight

$$\pi_\theta(s,a) = \frac{\exp^{\phi(s,a)^T\theta}}{\sum_{a'} \exp^{\phi(s,a')^T\theta}}$$

3. The score function is

$$\nabla_\theta \log \pi_\theta(s,a) = \phi(s,a) - \mathbb{E}_{\pi_\theta}[\phi(s,.)]$$

Policy Example: Gaussian Policy

1. In continuous action spaces, a Gaussian policy can be naturally defined
2. Mean is a linear combination of state features $\mu(s) = \phi(s)^T \theta$
3. Variance may be fixed σ^2 or can also be parameterized
4. Policy is Gaussian, the continuous $a \sim \mathcal{N}(\mu(s), \sigma^2)$
5. The score function is

$$\nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}$$

Policy Gradient for One-Step MDPs

1. Consider a simple class of one-step MDPs
   1. Starting in state $s \sim d(s)$
   2. Terminating after one time-step with reward $r = R(s, a)$
2. Use likelihood ratios to compute the policy gradient

$$\begin{aligned} J(\theta) =& \mathbb{E}_{\pi_\theta}[r] \\ =& \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s, a) r \end{aligned}$$

3. The gradient is as

$$\begin{aligned} \nabla_\theta J(\theta) =& \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a) r \\ =& \mathbb{E}_{\pi_\theta}[r \nabla_\theta \log \pi_\theta(s, a)] \end{aligned}$$

Policy Gradient for Multi-step MDPs

1. Denote a state-action trajectory from one episode as $\tau = (s_0, a_0, r_1, ...s_{T-1}, a_{T-1}, r_T, s_T) \sim (\pi_\theta, P(s_{t+1}|s_t, a_t))$
2. Denote $R(\tau) = \sum_{t=0}^{T} R(s_t, a_t)$ as the sum of rewards over a trajectory τ
3. The policy objective is

$$J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{T} R(s_t, a_t)\Big] = \sum_{\tau} P(\tau;\theta)R(\tau)$$

   where $P(\tau;\theta) = \mu(s_0)\prod_{t=0}^{T-1}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$ denotes the probability over trajectories when executing the policy π_θ
4. Then our goal is to find the policy parameter θ

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \sum_{\tau} P(\tau;\theta)R(\tau)$$

Policy Gradient for Multi-step MDPs

1. Our goal is to find the policy parameter θ

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

2. Take the gradient with respect to θ:

$$\begin{aligned}
\nabla_\theta J(\theta) =& \nabla_\theta \sum_\tau P(\tau;\theta)R(\tau) \\
=& \sum_\tau \nabla_\theta P(\tau;\theta)R(\tau) \\
=& \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta)R(\tau) \\
=& \sum_\tau P(\tau;\theta)R(\tau)\frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} \\
=& \sum_\tau P(\tau;\theta)R(\tau)\nabla_\theta \log P(\tau;\theta)
\end{aligned}$$

Policy Gradient for Multi-step MDPs

1. Our goal is to find the policy parameter θ

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

2. Take the gradient with respect to θ:

$$\nabla_\theta J(\theta) = \sum_\tau P(\tau;\theta)R(\tau)\nabla_\theta \log P(\tau;\theta)$$

3. Approximate with empirical estimate for m sample paths under policy π_θ:

$$\nabla_\theta J(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} R(\tau_i)\nabla_\theta \log P(\tau_i;\theta)$$

Decomposing the Trajectories into States and Actions

1. Approximate with empirical estimate for m sample paths under policy π_θ:

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \nabla_\theta \log P(\tau_i; \theta)$$

2. Decompose $\nabla_\theta \log P(\tau; \theta)$

$$\begin{aligned}
\nabla_\theta \log P(\tau; \theta) =& \nabla_\theta \log \Big[\mu(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \Big] \\
=& \nabla_\theta \Big[\log \mu(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t) \Big] \\
=& \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)
\end{aligned}$$

Likelihood Ratio Policy Gradient

1. Our goal is to find the policy parameter θ

$$\theta^* = \arg\max_\theta V(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

2. Approximate with empirical estimate for m sample paths under policy π_θ:

$$\nabla_\theta V(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} R(\tau_i)\nabla_\theta \log P(\tau_i;\theta)$$

3. Then we have $\nabla_\theta \log P(\tau_i;\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)$

$$\nabla_\theta J(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} R(\tau_i)\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)$$

    1. It do not need to know the dynamics model!

Summary for Part 1

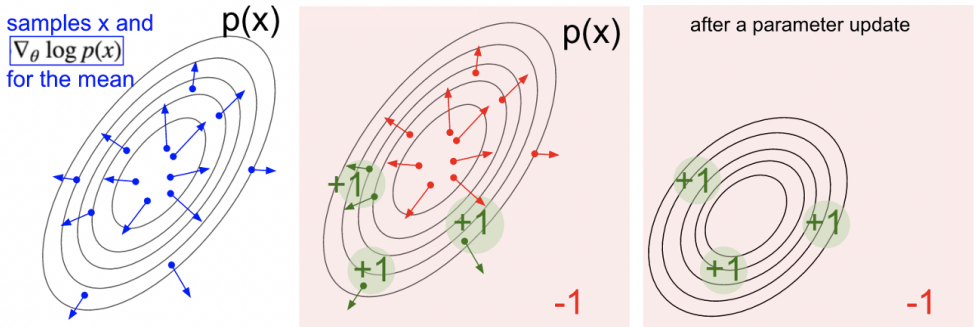
1. Policy-based reinforcement learning
2. Monte-Carlo policy gradient
3. Part 2
    1. Reduce the variance of policy gradient
    2. Actor-critic

Understanding Score Function Gradient Estimator

1. Consider the generic form of $E_{\tau\sim\pi_\theta}[R(\tau)]$ as

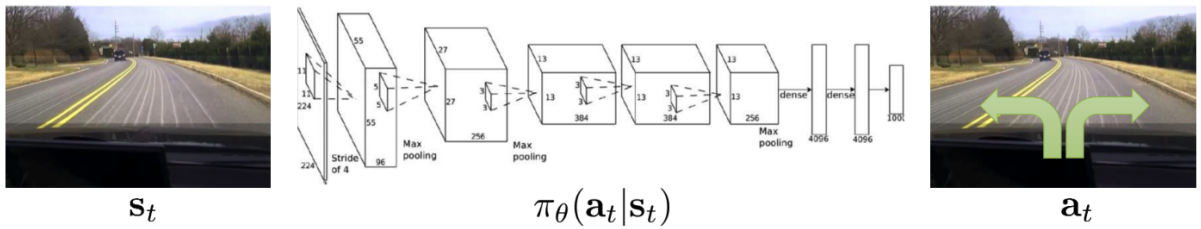
$$\nabla_\theta \mathbb{E}_{p(x;\theta)}[f(x)] = \mathbb{E}_{p(x;\theta)}[f(x)\nabla_\theta \log p(x;\theta)]$$
$$\approx \frac{1}{S}\sum_{s=1}^{S} f(x_s)\nabla_\theta \log p(x_s;\theta), \text{where } x_s \sim p(x;\theta)$$

   1. compute the gradient of an expectation of a function $f(x)$
2. The above gradient can be understood as:
   1. Shift the distribution p through its parameter θ to let its future samples x achieve higher scores as judged by $f(x)$
   2. The direction of $f(x)\nabla_\theta \log p(x;\theta)$ pushes up the log likelihood of the sample, in proportion to how good it is



Comparison to Maximum Likelihood

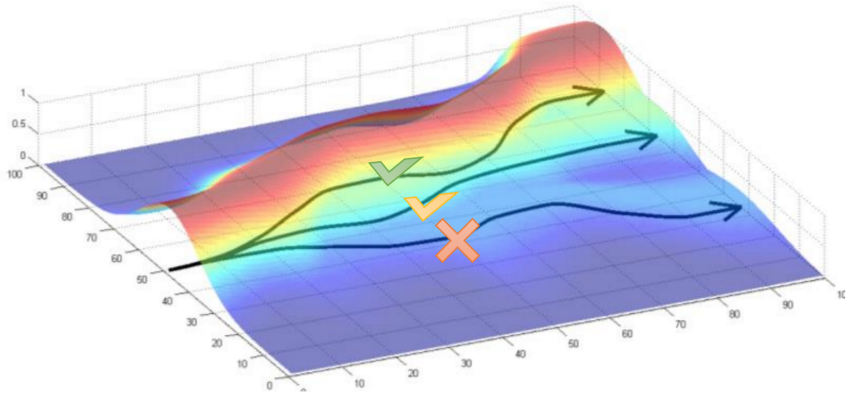
1. Policy gradient estimator: $\nabla_\theta J(\theta) \approx \frac{1}{M}\sum_{m=1}^{M}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{t,m}|s_{t,m})\right)\left(\sum_{t=1}^{T} r(s_{t,m}, a_{t,m})\right)$
2. Maximum likelihood estimator: $\nabla_\theta J_{ML}(\theta) \approx \frac{1}{M}\sum_{m=1}^{M}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{t,m}|s_{t,m})\right)$
3. Interpretation: good action is made more likely, bad action is made less likely



Comparison to Maximum Likelihood

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \nabla_\theta \log P(\tau_i; \theta)$$

1. If going up the hill leads to higher reward, optimize the policy parameters to increase the likelihood of trajectories that move higher



Large Variance of Policy Gradient

1. We have the following approximate update

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i)$$

2. Unbiased but very noisy
3. Two fixes:
    1. Use temporal causality
    2. Include a baseline

Reduce Variance of Policy Gradient using Causality

1. Previously $\nabla_\theta \mathbb{E}_\tau[R] = \mathbb{E}_\tau\left[\left(\sum_{t=0}^{T-1} r_t\right)\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right)\right]$
2. We can derive the gradient estimator for a single reward term $r_{t'}$ as

$$\nabla_\theta \mathbb{E}_\tau[r_{t'}] = \mathbb{E}_\tau\left[r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

3. Summing this formula over t, we obtain

$$\begin{aligned}
\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R] &= \mathbb{E}_\tau\left[\sum_{t'=0}^{T-1} r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \\
&= \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T-1} r_{t'}\right] \\
&= \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right]
\end{aligned}$$

Reduce Variance of Policy Gradient using Causality

1. Therefore we have

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R] = \mathbb{E}_\tau\Big[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\Big]$$

2. $G_t = \sum_{t'=t}^{T-1} r_{t'}$ is the return for a trajectory at step t
3. Causality: policy at time t' cannot affect reward at time t when $t < t'$
4. Then we can have the following estimated update

$$\nabla_\theta \mathbb{E}[R] \approx \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T-1} G_t^{(i)} \cdot \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)$$

REINFORCE: A Monte-Carlo policy gradient algorithm

1. The algorithm simply samples multiple trajectories following the policy π_θ while updating θ using the estimated gradient

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

2. Classic paper: Williams (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning: introduces REINFORCE algorithm

Reducing Variance Using a Baseline

1. The original update

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R] = \mathbb{E}_\tau\Big[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\Big]$$

   1. $G_t = \sum_{t'=t}^{T-1} r_{t'}$ is the return for a trajectory which might have high variance

2. We subtract a baseline $b(s)$ from the policy gradient to reduce variance

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R] = \mathbb{E}_\tau\Big[\sum_{t=0}^{T-1} \big(G_t - b(s_t)\big) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\Big]$$

3. A good baseline is the expected return

$$b(s_t) = \mathbb{E}[r_t + r_{t+1} + ... + r_{T-1}]$$

Reducing Variance Using a Baseline

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R] = \mathbb{E}_\tau\Big[\sum_{t=0}^{T-1}\big(G_t - b(s_t)\big)\cdot\nabla_\theta \log \pi_\theta(a_t|s_t)\Big]$$

1. Interpretation: increase the logprob of action a_t proportional to how much returns G_t are better than the expected return
2. We can **prove** that baseline $b(s)$ can reduce variance, without changing the expectation:

$$\mathbb{E}_\tau\Big[\nabla_\theta \log \pi_\theta(a_t|s_t)b(s_t)\Big] = 0, \tag{1}$$

$$E_\tau\Big[\nabla_\theta \log \pi_\theta(a_t|s_t)(G_t - b(s_t))\Big] = E_\tau\Big[\nabla_\theta \log \pi_\theta(a_t|s_t)G_t\Big] \tag{2}$$

$$Var_\tau\Big[\nabla_\theta \log \pi_\theta(a_t|s_t)(G_t - b(s_t))\Big] < Var_\tau\Big[\nabla_\theta \log \pi_\theta(a_t|s_t)G_t\Big] \tag{3}$$

3. Thus subtracting a baseline is unbiased in expectation but reduces variance

Reducing Variance by a Baseline

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \Big[\sum_{t=0}^{T-1} \big(G_t - b_{\mathbf{w}}(s_t) \big) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \Big]$$

1. Baseline $b(s)$ can reduce variance, without changing the expectation
2. $b_{\mathbf{w}}(s)$ also has a parameter $\mathbf{w}$ to learn so that we have two set of parameters $\mathbf{w}$ and θ

Vanilla Policy Gradient Algorithm with Baseline

procedure POLICY GRADIENT(α)

Initialize policy parameters θ and baseline values $b(s)$ for all s, e.g. to 0

for iteration $= 1, 2, \ldots$ **do**

Collect a set of m trajectories by executing the current policy π_θ

for each time step t of each trajectory $\tau^{(i)}$ **do**

Compute the *return* $G_t^{(i)} = \sum_{t'=t}^{T-1} r_{t'}$

Compute the *advantage estimate* $\hat{A}_t^{(i)} = G_t^{(i)} - b(s_t)$

Re-fit the baseline to the empirical returns by updating $\mathbf{w}$ to minimize

$$\sum_{i=1}^{m} \sum_{t=0}^{T-1} \| b(s_t) - G_t^{(i)} \|^2$$

Update policy parameters θ using the policy gradient estimate $\hat{g}$

$$\hat{g} = \sum_{i=1}^{m} \sum_{t=0}^{T-1} \hat{A}_t^{(i)} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

with an optimizer like SGD ($\theta \leftarrow \theta + \alpha \cdot \hat{g}$) or Adam

return θ and baseline values $b(s)$

1. Sutton, McAllester, Singh, Mansour (1999). Policy gradient methods for reinforcement learning with function approximation

Reducing Variance Using a Critic

1. The update is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\Big]$$

2. In practice, G_t is a sample from Monte Carlo policy gradient, which is the unbiased but noisy estimate of $Q^{\pi_\theta}(s_t, a_t)$
3. Instead we can use a critic to estimate the action-value function,

$$Q_{\mathbf{w}}(s, a) \approx Q^{\pi_\theta}(s, a)$$

4. Then the update becomes

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{T-1} Q_{\mathbf{w}}(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\Big]$$

Actor-Critic Policy Gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} Q_{\mathbf{w}}(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

1. It becomes Actor-Critic Policy Gradient
   1. Actor: the policy function used to generate the action
   2. Critic: the value function used to evaluate the reward of the actions
2. Actor-critic algorithms maintain two sets of parameters
   1. Actor: Updates policy parameters θ, in direction suggested by critic
   2. Critic: Updates action-value function parameters $\mathbf{w}$

Estimating the Action-Value Function

1. The critic is solving a familiar problem: policy evaluation
    1. How good is policy π_θ for current parameter θ
2. Policy evaluation was explored in previous lectures, e.g.
    1. Monte-Carlo policy evaluation
    2. Temporal-Difference learning
    3. Least-squares policy evaluation

Action-Value Actor-Critic Algorithm

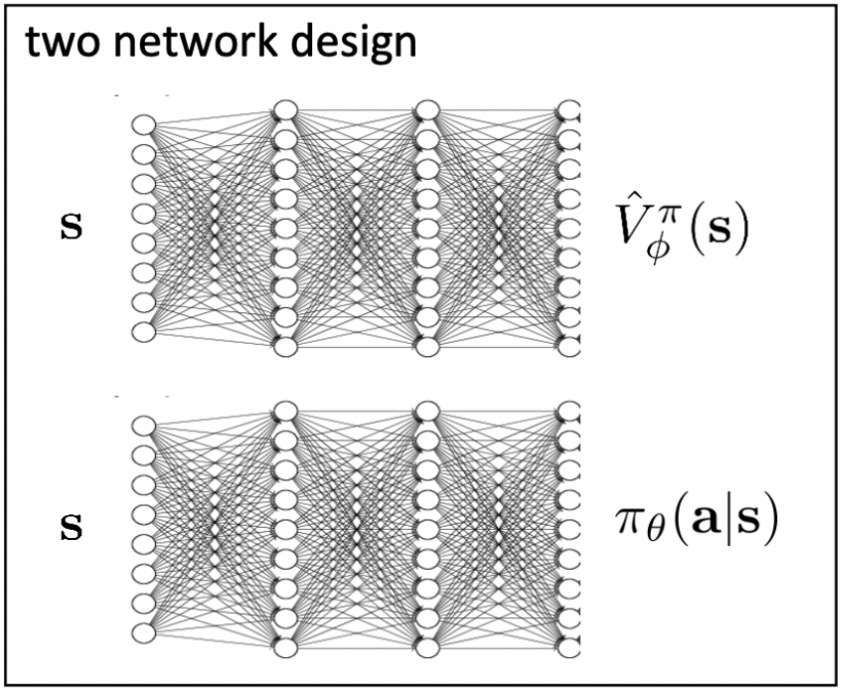
1. Using a linear value function approximation: $Q_{\mathbf{w}}(s,a) = \psi(s,a)^T \mathbf{w}$
   1. Critic: update $\mathbf{w}$ by a linear TD(0)
   2. Actor: update θ by policy gradient

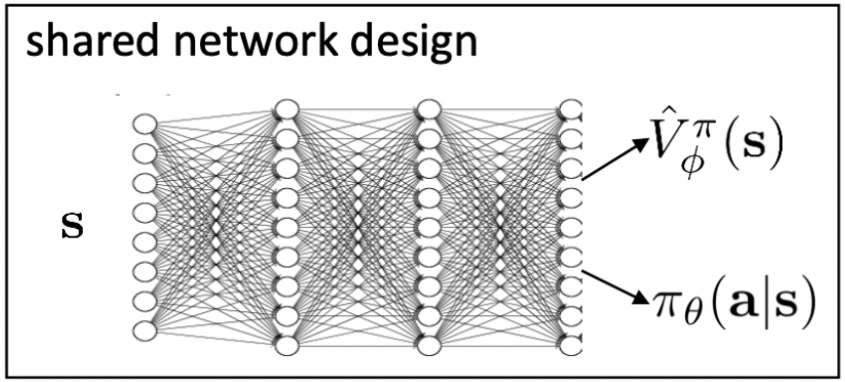
Algorithm 2 Simple QAC

for each step **do**
  generate sample s, a, r, s', a' following π_θ
  $\delta = r + \gamma Q_{\mathbf{w}}(s', a') - Q_{\mathbf{w}}(s, a)$ #TD error
  $\mathbf{w} \leftarrow \mathbf{w} + \beta \delta \psi(s, a)$
  $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_{\mathbf{w}}(s, a)$
end for

Actor-Critic Function Approximators

1. We can have two separate functions to approximate value function and policy function, or use a shared network design (feature extraction is shared but output two heads), as below:





Reducing the Variance of Actor-Critic by a Baseline

1. Recall Q-function / state-action-value function:

$$Q^{\pi,\gamma}(s,a) = \mathbb{E}_{\pi}[r_1 + \gamma r_2 + ...|s_1 = s, a_1 = a]$$

2. State value function can serve as a great baseline

$$\begin{aligned} V^{\pi,\gamma}(s) =& \mathbb{E}_{\pi}[r_1 + \gamma r_2 + ...|s_1 = s] \\ =& \mathbb{E}_{a\sim\pi}[Q^{\pi,\gamma}(s,a)] \end{aligned}$$

3. Advantage function: combining Q with baseline V

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

4. Then the policy gradient becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi,\gamma}(s,a)]$$

N-step estimators

1. We used the Monte-Carlo estimates of the reward
2. We can also us TD methods for the policy gradient update, or any intermediate blend between TD and MC methods:
3. Consider the following n-step returns for $n = 1, 2, \infty$

$$\begin{aligned} n = 1(TD) \quad G_t^{(1)} &= r_{t+1} + \gamma v(s_{t+1}) \\ n = 2 \qquad\quad G_t^{(2)} &= r_{t+1} + \gamma r_{t+2} + \gamma^2 v(s_{t+2}) \\ n = \infty(MC) \quad G_t^{(\infty)} &= r_{t+1} + \gamma r_{t+2} + ... + \gamma^{T-t-1} r_T \end{aligned}$$

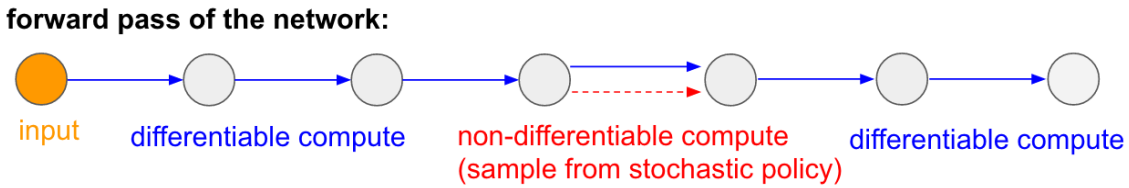
4. Then the advantage estimators become

$$\begin{aligned} \hat{A}_t^{(1)} &= r_{t+1} + \gamma v(s_{t+1}) - v(s_t) \\ \hat{A}_t^{(2)} &= r_{t+1} + \gamma r_{t+2} + \gamma^2 v(s_{t+2}) - v(s_t) \\ \hat{A}_t^{(\infty)} &= r_{t+1} + \gamma r_{t+2} + ... + \gamma^{T-t-1} r_T - v(s_t) \end{aligned}$$

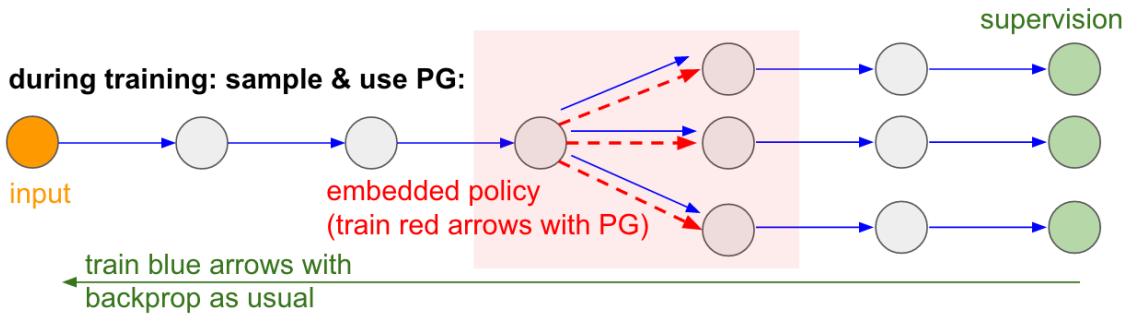
$\hat{A}^{(1)}$ has low variance and high bias. $\hat{A}^{(\infty)}$ has high variance but low bias

Overcoming Non-differentiable Computation

1. Another interesting advantage of Policy Gradient is that it allows us to overcome the non-differentiable computation



2. During training we will produce several samples (indicated by the branches below), and then we'll encourage samples that eventually led to good outcomes (in this case for example measured by the loss at the end)



ref:`http://karpathy.github.io/2016/05/31/rl/`

Extension of Policy Gradient

1. State-of-the-art RL methods are almost all policy-based
    1. **A2C and A3C**: Asynchronous Methods for Deep Reinforcement Learning, ICML'16. Representative high-performance actor-critic algorithm: `https://openai.com/blog/baselines-acktr-a2c/`
    2. **TRPO**: Schulman, L., Moritz, Jordan, Abbeel (2015). Trust region policy optimization: deep RL with natural policy gradient and adaptive step size
    3. **PPO**: Schulman, Wolski, Dhariwal, Radford, Klimov (2017). Proximal policy optimization algorithms: deep RL with importance sampled policy gradient

Different Schools of Reinforcement Learning

1. Value-based RL: solve RL through dynamic programming
   1. Classic RL and control theory
   2. Representative algorithms: Deep Q-learning and its variant
   3. Representative researchers: Richard Sutton (no more than 20 pages on PG out of the 500-page textbook), David Silver, from DeepMind
2. Policy-based RL: solve RL mainly through learning
   1. Machine learning and deep learning
   2. Representative algorithms: PG, and its variants TRPO, PPO, and others
   3. Representative researchers: Pieter Abbeel, Sergey Levine, John Schulman, from OpenAI, Berkelely
3. Some random essay I wrote on Zhihu:`https://www.zhihu.com/question/316626294/answer/627373838`

请问DeepMind和OpenAI身后的两大RL流派有什么具体的区别?

周博磊

机器学习、深度学习（Deep Learning）、人工智能 话题的优秀回答者

1,681 人赞同了该回答

Policy gradient code example

1. A very nice summary of policy graident algorithms: https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html
2. REINFORCE code on CartPole: https://github.com/cuhkrlcourse/RLexample/blob/master/policygradient/reinforce.py
3. Policy Gradient on Pong : https://github.com/cuhkrlcourse/RLexample/blob/master/policygradient/pg-pong-pytorch.py
4. Policy Gradient with Baseline on Pong: https://github.com/cuhkrlcourse/RLexample/blob/master/policygradient/pgb-pong-pytorch.py

Concluding Remarks

1. **Derive the policy gradient by yourself to get a deeper understanding!**
2. Next time: Policy Optimization II SOTA

Next Week on State of the Art for Policy Optimization

Warning: gonna be a brutally challenging lecture! Read the papers beforehand as much as possible

1. Policy Gradient→TRPO→ACKTR→PPO
    1. **TRPO**: Trust region policy optimization. Schulman, L., Moritz, Jordan, Abbeel. 2015
    2. **ACKTR**: Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba. 2017
    3. **PPO**: Proximal policy optimization algorithms. Schulman, Wolski, Dhariwal, Radford, Klimov. 2017
2. Q-learning→DDPG→TD3→SAC
    1. **DDPG**: Deterministic Policy Gradient Algorithms, Silver et al. 2014
    2. **TD3**: Addressing Function Approximation Error in Actor-Critic Methods, Fujimoto et al. 2018
    3. **SAC**: Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, Haarnoja et al. 2018