第七章 强化学习IV—无模型策略方法

Chao Yu（余超）

School of Computer Science and Engineering
Sun Yat-Sen University

[based on David Silver and Sergey Levine’s course]

Large-Scale Reinforcement Learning

- Reinforcement learning can be used to solve large problems, e.g.
  - Backgammon: 10^{20} states
  - Computer Go: 10^{170} states
  - Helicopter: continuous state space

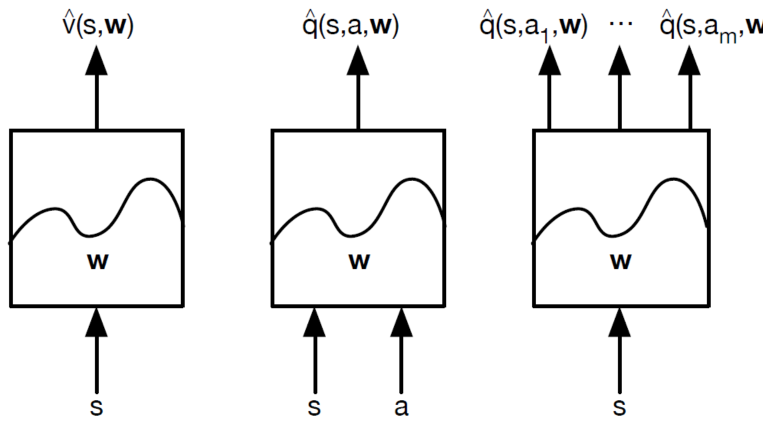
Value Function Approximation

- So far we have represented value function by a *lookup table*
  - Every state s has an entry $V(s)$
  - Or every state-action pair s, a has an entry $Q(s, a)$
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with *function approximation*

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$
$$\text{or } \hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

  - *Generalise* from seen states to unseen states
  - *Update* parameter $\mathbf{w}$ using MC or TD learning

Types of Value Function Approximation



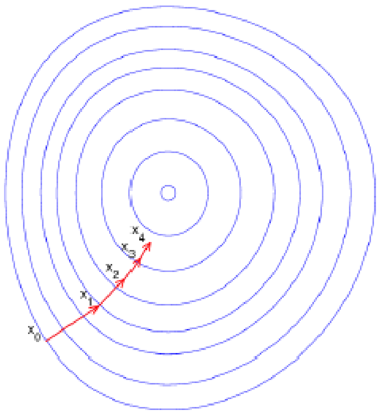
There are many function approximators, e.g.

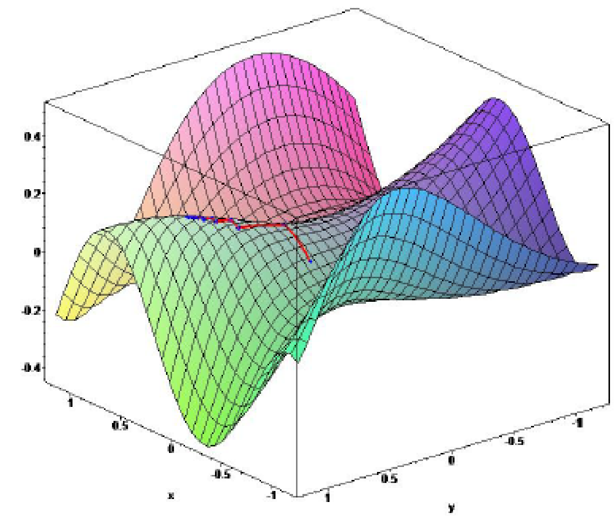
- Linear combinations of features
- Neural network
- Decision tree
- Nearest neighbour
- Fourier / wavelet bases
- ...

Gradient Descent

- Let $J(\mathbf{w})$ be a differentiable function of parameter vector $\mathbf{w}$
- Define the *gradient* of $J(\mathbf{w})$ to be

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$





- To find a local minimum of $J(\mathbf{w})$
- Adjust $\mathbf{w}$ in direction of -ve gradient

$$\Delta \mathbf{w} = -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

where α is a step-size parameter

Value Function Approx. By Stochastic Gradient Descent

- Goal: find parameter vector $\mathbf{w}$ minimising mean-squared error between approximate value fn $\hat{v}(s, \mathbf{w})$ and true value fn $v_\pi(s)$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

- Gradient descent finds a local minimum

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \right] \end{aligned}$$

- Stochastic gradient descent *samples* the gradient

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$$

- Expected update is equal to full gradient update

Feature Vectors

- Represent state by a *feature vector*

$$\mathbf{x}(S) = \begin{pmatrix} \mathbf{x}_1(S) \\ \vdots \\ \mathbf{x}_n(S) \end{pmatrix}$$

- For example:
  - Distance of robot from landmarks
  - Trends in the stock market
  - Piece and pawn configurations in chess

Linear Value Function Approximation

- Represent value function by a linear combination of features

$$\hat{v}(S,\mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S)\mathbf{w}_j$$

- Objective function is quadratic in parameters $\mathbf{w}$

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[(v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w})^2\right]$$

- Stochastic gradient descent converges on *global* optimum
- Update rule is particularly simple

$$\begin{aligned}\nabla_{\mathbf{w}}\hat{v}(S,\mathbf{w}) &= \mathbf{x}(S)\\ \Delta\mathbf{w} &= \alpha(v_\pi(S) - \hat{v}(S,\mathbf{w}))\mathbf{x}(S)\end{aligned}$$

Update = *step-size* × *prediction error* × *feature value*

Incremental Prediction Algorithms

- Have assumed true value function $v_\pi(s)$ given by supervisor
- But in RL there is no supervisor, only rewards
- In practice, we substitute a *target* for $v_\pi(s)$
  - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha({\color{red}G_t} - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$$

  - For TD(0), the target is the TD target $R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha({\color{red}R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w})} - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$$

  - For TD(λ), the target is the λ-return G_t^λ

$$\Delta \mathbf{w} = \alpha({\color{red}G_t^\lambda} - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$$

Monte-Carlo with Value Function Approximation

- Return G_t is an unbiased, noisy sample of true value $v_\pi(S_t)$
- Can therefore apply supervised learning to "training data":

$$\langle S_1, G_1\rangle, \langle S_2, G_2\rangle, ..., \langle S_T, G_T\rangle$$

- For example, using *linear Monte-Carlo policy evaluation*

$$\begin{aligned}\Delta \mathbf{w} &= \alpha({\color{red}G_t} - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w}) \\ &= \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)\end{aligned}$$

- Monte-Carlo evaluation converges to a local optimum
- Even when using non-linear value function approximation

TD Learning with Value Function Approximation

- The TD-target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$ is a *biased* sample of true value $v_\pi(S_t)$
- Can still apply supervised learning to "training data":

$$\langle S_1, R_2 + \gamma \hat{v}(S_2, \mathbf{w})\rangle, \langle S_2, R_3 + \gamma \hat{v}(S_3, \mathbf{w})\rangle, ..., \langle S_{T-1}, R_T\rangle$$

- For example, using *linear TD(0)*

$$\begin{aligned}\Delta \mathbf{w} &= \alpha({\color{red}R + \gamma \hat{v}(S', \mathbf{w})} - \hat{v}(S, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \\ &= \alpha \delta \mathbf{x}(S)\end{aligned}$$

- Linear TD(0) converges (close) to global optimum

Action-Value Function Approximation

- Approximate the action-value function

$$\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A)$$

- Minimise mean-squared error between approximate action-value fn $\hat{q}(S, A, \mathbf{w})$ and true action-value fn $q_\pi(S, A)$

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2\right]$$

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2}\nabla_\mathbf{w} J(\mathbf{w}) = (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_\mathbf{w}\hat{q}(S, A, \mathbf{w})$$

$$\Delta\mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_\mathbf{w}\hat{q}(S, A, \mathbf{w})$$

Linear Action-Value Function Approximation

- Represent state *and* action by a *feature vector*

$$\mathbf{x}(S, A) = \begin{pmatrix} \mathbf{x}_1(S, A) \\ \vdots \\ \mathbf{x}_n(S, A) \end{pmatrix}$$

- Represent action-value fn by linear combination of features

$$\hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S, A)\mathbf{w}_j$$

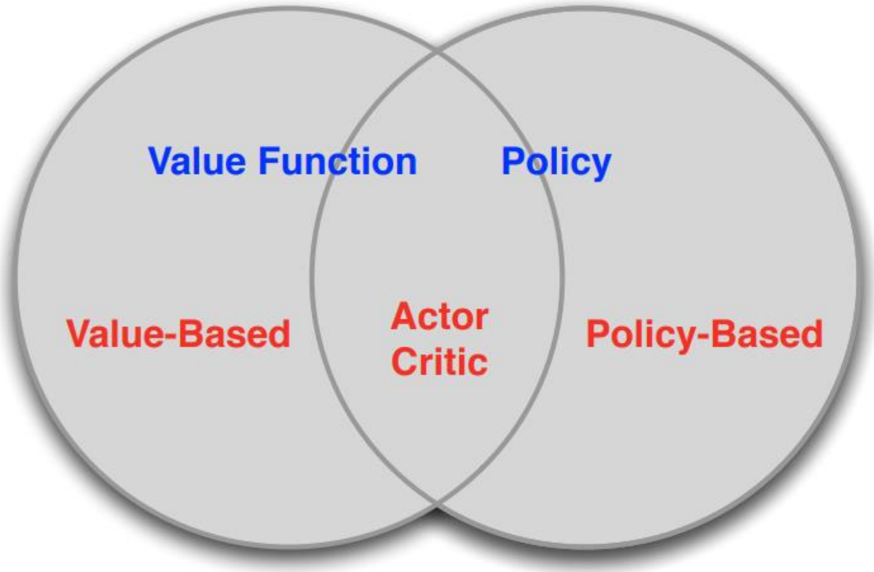
- Stochastic gradient descent update

$$\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)$$

$$\Delta \mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\mathbf{x}(S, A)$$

Value-Based and Policy-Based RL

- Value Based
  - Learnt Value Function
  - Implicit policy (e.g. ϵ-greedy)
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy



Advantages of Policy-Based RL

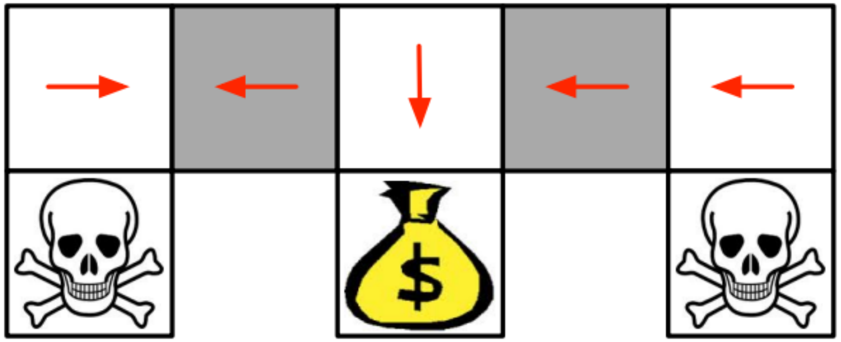
Advantages:

- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Disadvantages:

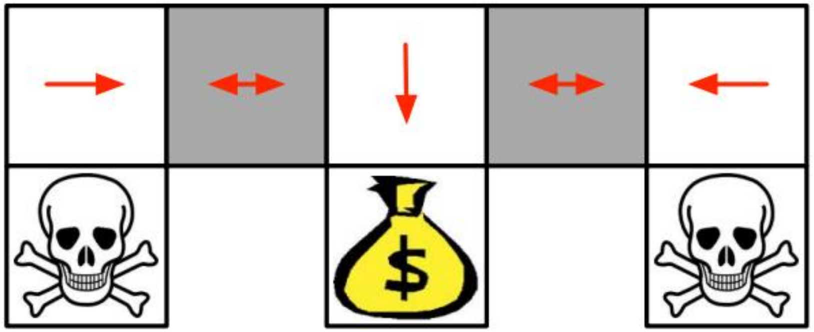
- Typically converge to a local rather than global optimum
- Evaluating a policy is typically inefficient and high variance

Example: Aliased Gridworld



- Under aliasing, an optimal deterministic policy will either
  - move W in both grey states (shown by red arrows)
  - move E in both grey states
- Either way, it can get stuck and *never* reach the money
- Value-based RL learns a near-deterministic policy
  - e.g. greedy or ϵ-greedy
- So it will traverse the corridor for a long time

Example: Aliased Gridworld



- An optimal stochastic policy will randomly move E or W in grey states

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

- It will reach the goal state in a few steps with high probability
- Policy-based RL can learn the optimal stochastic policy

Policy Objective Functions

- Goal: given policy $\pi_\theta(s, a)$ with parameters θ, find best θ
- But how do we measure the quality of a policy π_θ?
- In episodic environments we can use the start value

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

- In continuing environments we can use the average value

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

- Or the average reward per time-step

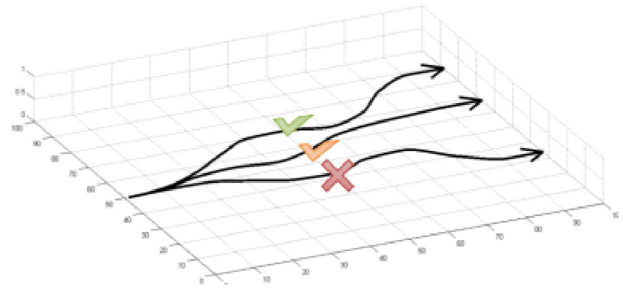
$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) \mathcal{R}_s^a$$

- where $d^{\pi_\theta}(s)$ is stationary distribution of Markov chain for π_θ

Policy Objective Functions

$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$$\theta^\star = \arg\max_\theta \underbrace{E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]}_{J(\theta)}$$



$$J(\theta) = E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

sum over samples from π_θ

Policy Optimisation

- Policy based reinforcement learning is an optimisation problem
- Find θ that maximises $J(\theta)$
- Some approaches do not use gradient
  - Hill climbing
  - Simplex / amoeba / Nelder Mead
  - Genetic algorithms
- Greater efficiency often possible using gradient
  - Gradient descent
  - Conjugate gradient
  - Quasi-newton
- We focus on gradient descent, many extensions possible
- And on methods that exploit sequential structure

Score Function

- We now compute the policy gradient *analytically*
- Assume policy π_θ is differentiable whenever it is non-zero
- and we know the gradient $\nabla_\theta \pi_\theta(s, a)$
- Likelihood ratios exploit the following identity

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

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

One-Step MDPs

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

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

Policy Gradient Theorem

- The policy gradient theorem generalises the likelihood ratio approach to multi-step MDPs
- Replaces instantaneous reward r with long-term value $Q^{\pi}(s, a)$
- Policy gradient theorem applies to start state objective, average reward and average value objective

Theorem

For any differentiable policy $\pi_\theta(s, a)$,
for any of the policy objective functions $J = J_1, J_{avR},$ or $\frac{1}{1-\gamma} J_{avV}$,
the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\ Q^{\pi_\theta}(s, a)\right]$$

Monte-Carlo Policy Gradient (REINFORCE)

- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return v_t as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

function REINFORCE

  Initialise θ arbitrarily

  for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**

    for $t = 1$ to $T - 1$ **do**

      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$

    end for

  end for

  return θ

end function

Direct policy differentiation

$$\theta^\star = \arg\max_\theta \underbrace{E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right]}_{J(\theta)}$$

a convenient identity

$$\underline{p_\theta(\tau)\nabla_\theta \log p_\theta(\tau)} = p_\theta(\tau)\frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} = \underline{\nabla_\theta p_\theta(\tau)}$$

$$J(\theta) = E_{\tau\sim p_\theta(\tau)}[\underbrace{r(\tau)}_{\sum_{t=1}^{T} r(\mathbf{s}_t,\mathbf{a}_t)}] = \int p_\theta(\tau) r(\tau) d\tau$$

$$\nabla_\theta J(\theta) = \int \underline{\nabla_\theta p_\theta(\tau)} r(\tau) d\tau = \int \underline{p_\theta(\tau)\nabla_\theta \log p_\theta(\tau)} r(\tau) d\tau = E_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)]$$

Direct policy differentiation

$$\theta^\star = \arg\max_\theta J(\theta)$$

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}[r(\tau)]$$

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)]$$

$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

log of both sides

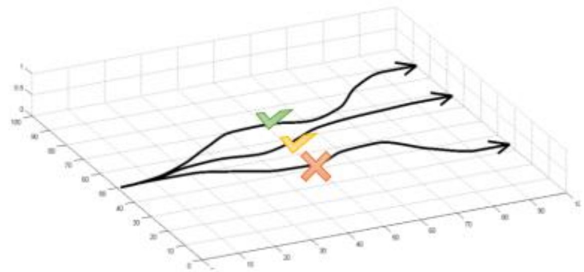
$$\log p_\theta(\tau) = \log p(\mathbf{s}_1) + \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$$\nabla_\theta \left[\cancel{\log p(\mathbf{s}_1)} + \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \cancel{\log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \right]$$

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

Direct policy differentiation

Evaluating the policy gradient

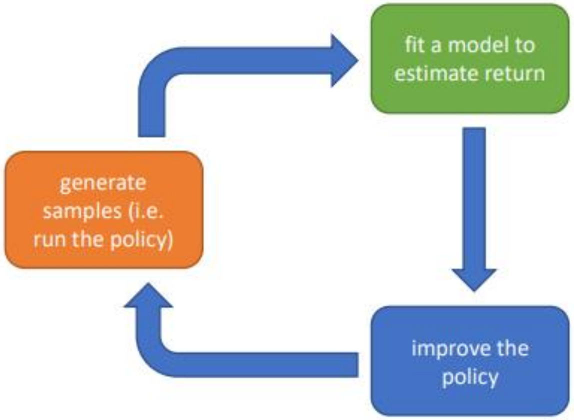


recall: $J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})\right)$$

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



REINFORCE algorithm:

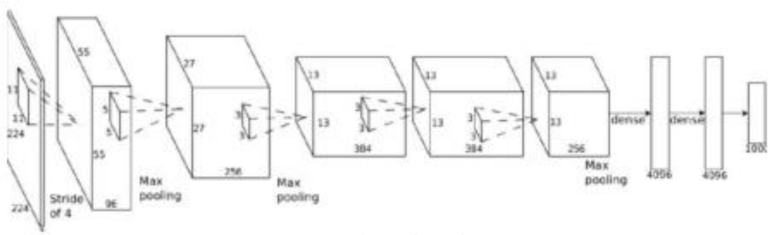
1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Comparison to maximum likelihood

policy gradient: $$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$

maximum likelihood: $$\nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)$$

$\mathbf{s}_t$

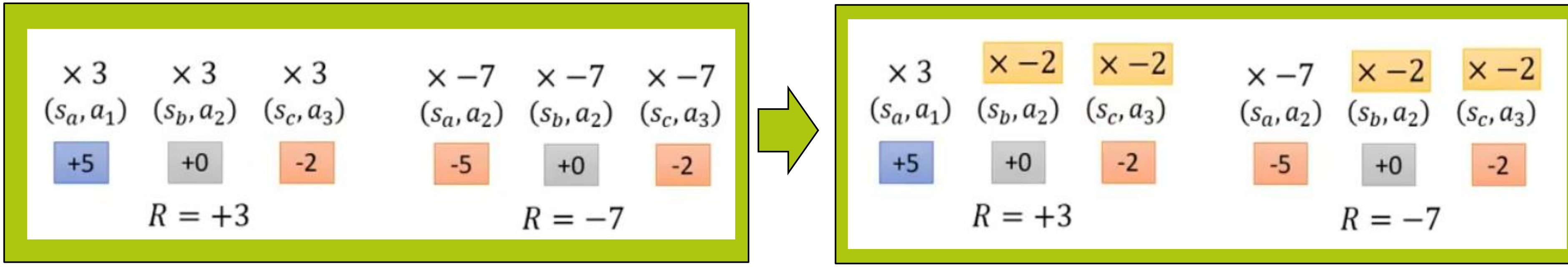


$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$

$\mathbf{a}_t$

$\mathbf{s}_t$ $\mathbf{a}_t$ → training data → supervised learning → $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$

What is wrong with the policy gradient?



$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$

Causality: policy at time t' cannot affect reward at time t when $t < t'$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \underbrace{\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right)}_{\text{“reward to go”}}$$

$$\hat{Q}_{i,t}$$

Reducing Variance Using a Critic

- Monte-Carlo policy gradient still has high variance
- We use a critic to estimate the action-value function,

$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$$

- Actor-critic algorithms maintain *two* sets of parameters
  - Critic: Updates action-value function parameters w
  - Actor: Updates policy parameters θ, in direction suggested by critic
- Actor-critic algorithms follow an *approximate* policy gradient

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\ Q_w(s, a)\right]$$
$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s, a)\ Q_w(s, a)$$

Estimating the Action-Value Function

- The critic is solving a familiar problem: policy evaluation
- How good is policy π_θ for current parameters θ?
- This problem was explored in previous two lectures, e.g.
  - Monte-Carlo policy evaluation
  - Temporal-Difference learning
  - TD(λ)
- Could also use e.g. least-squares policy evaluation

Actor-Critic

- Simple actor-critic algorithm based on action-value critic
- Using linear value fn approx. $Q_w(s,a) = \phi(s,a)^\top w$

  Critic Updates w by linear TD(0)

  Actor Updates θ by policy gradient

function QAC
    Initialise s, θ
    Sample $a \sim \pi_\theta$
    for each step **do**
        Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$
        Sample action $a' \sim \pi_\theta(s', a')$
        $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$
        $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$
        $w \leftarrow w + \beta \delta \phi(s, a)$
        $a \leftarrow a', s \leftarrow s'$
    end for
end function

Reducing Variance Using a Baseline

- We subtract a baseline function $B(s)$ from the policy gradient
- This can reduce variance, without changing expectation

$$
\begin{aligned}
\mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) B(s)\right] &= \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s,a) B(s) \\
&= \sum_{s \in \mathcal{S}} d^{\pi_\theta} B(s) \nabla_\theta \sum_{a \in \mathcal{A}} \pi_\theta(s,a) \\
&= 0
\end{aligned}
$$

- A good baseline is the state value function $B(s) = V^{\pi_\theta}(s)$
- So we can rewrite the policy gradient using the advantage function $A^{\pi_\theta}(s,a)$

$$
\begin{aligned}
A^{\pi_\theta}(s,a) &= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \\
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^{\pi_\theta}(s,a)\right]
\end{aligned}
$$

Reducing Variance Using a Baseline

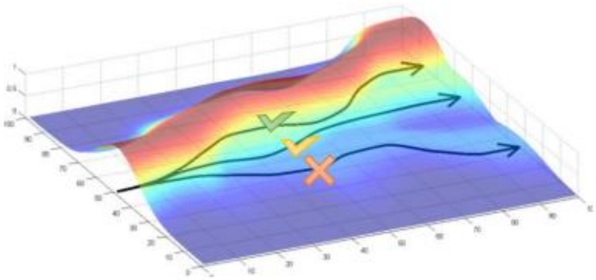
a convenient identity

$$p_\theta(\tau)\nabla_\theta \log p_\theta(\tau) = \nabla_\theta p_\theta(\tau)$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log p_\theta(\tau)[r(\tau) - b]$$

$$b = \frac{1}{N}\sum_{i=1}^{N} r(\tau)$$

but... are we *allowed* to do that??



$$E[\nabla_\theta \log p_\theta(\tau) b] = \int p_\theta(\tau)\nabla_\theta \log p_\theta(\tau) b \, d\tau = \int \nabla_\theta p_\theta(\tau) b \, d\tau = b\nabla_\theta \int p_\theta(\tau) d\tau = b\nabla_\theta 1 = 0$$

subtracting a baseline is *unbiased* in expectation!

average reward is *not* the best baseline, but it's pretty good!

Estimating the Advantage Function (1)

- The advantage function can significantly reduce variance of policy gradient
- So the critic should really estimate the advantage function
- For example, by estimating *both* $V^{\pi_\theta}(s)$ *and* $Q^{\pi_\theta}(s,a)$
- Using two function approximators and two parameter vectors,

$$
\begin{aligned}
V_v(s) &\approx V^{\pi_\theta}(s) \\
Q_w(s,a) &\approx Q^{\pi_\theta}(s,a) \\
A(s,a) &= Q_w(s,a) - V_v(s)
\end{aligned}
$$

- And updating *both* value functions by e.g. TD learning

Estimating the Advantage Function (2)

- For the true value function $V^{\pi_\theta}(s)$, the TD error δ^{π_θ}

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

- is an unbiased estimate of the advantage function

$$\begin{aligned}
\mathbb{E}_{\pi_\theta}\left[\delta^{\pi_\theta} | s, a\right] &= \mathbb{E}_{\pi_\theta}\left[r + \gamma V^{\pi_\theta}(s') | s, a\right] - V^{\pi_\theta}(s) \\
&= Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s) \\
&= A^{\pi_\theta}(s, a)
\end{aligned}$$

- So we can use the TD error to compute the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\ \delta^{\pi_\theta}\right]$$

- In practice we can use an approximate TD error

$$\delta_v = r + \gamma V_v(s') - V_v(s)$$