# 策略梯度（Policy Gradient）

吉建民

USTC
jianmin@ustc.edu.cn

2023 年 11 月 6 日

## Used Materials

Disclaimer: 本课件大量采用了 Rich Sutton's RL class, David Silver's Deep RL tutorial 和其他网络课程课件，也采用了 GitHub 中开源代码，以及部分网络博客内容

## Table of Contents

## Model-free Prediction $V_\pi(s)$: MC, TD(0)

- Given episodes generated by policy $\pi$:

$$S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, A_{t+2}, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$$
$$S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, A_{t+2}, \ldots$$

- MC:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T$$
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

- TD(0):

$$\text{TD target: } G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$$
$$\text{TD error: } \delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$
$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$
$$V(S_t) \leftarrow V(S_t) + \alpha\delta_t$$

## Model-free Prediction $V_\pi(s)$: n-step TD, TD($\lambda$)

- n-step TD:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(n)} - V(S_t))$$

- Forward View TD($\lambda$):

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)} + \lambda^{T-t-1} G_t^{(T-t)}$$
$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^\lambda - V(S_t))$$

- Backward View TD($\lambda$):

$$E_0(s) = \mathbf{1}(S_t = s)$$
$$E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}(S_t = s)$$
$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$
$$V(s) \leftarrow V(s) + \alpha\delta_t E_t(s)$$

## Model-free Prediction $Q_\pi(s,a)$: MC, TD(0), n-step TD

- MC:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T$$
$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha(G_t - Q(S_t,A_t))$$

- TD(0):

$$\text{TD target: } q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1},A_{t+1})$$
$$\text{TD error: } \delta_t = R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)$$
$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t))$$
$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\delta_t$$

- n-step TD:

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n},A_{t+n})$$
$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha(q_t^{(n)} - Q(S_t,A_t))$$

## Model-free Prediction $Q_\pi(s, a)$: TD($\lambda$)

- Forward View TD($\lambda$):

$$q_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}q_t^{(n)} + \lambda^{T-t-1}q_t^{(T-t)}$$

$$q_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}q_t^{(n)}$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(q_t^\lambda - Q(S_t, A_t))$$

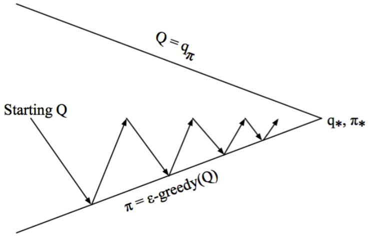
- Backward View TD($\lambda$):

$$\begin{aligned} E_0(s,a) &= \mathbf{1}(S_t = s, A_t = a) \\ E_t(s,a) &= \gamma\lambda E_{t-1}(s,a) + \mathbf{1}(S_t = s, A_t = a) \\ \delta_t &= R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \\ Q(s,a) &\leftarrow Q(s,a) + \alpha\delta_t E_t(s,a) \end{aligned}$$

## Model-free Control: MC, On-policy

In On-policy control methods the policy is generally "soft", meaning that:

$$\pi(a \mid s) > 0 \quad \forall s \in S, a \in A(s)$$



Every episode:

Policy evaluation Monte-Carlo policy evaluation, $Q \approx q_\pi$

Policy improvement $\epsilon$-greedy policy improvement

$$\pi(a \mid s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a* = argmax_{a \in A} Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

## Model-free Control: MC, Off-policy

- target policy $\pi$, behavior policy $\mu$
- Assumption of coverage: for all $s \in S$ and $a \in A(s)$

$$\pi(a \mid s) > 0 \text{ implies } \mu(a \mid s) > 0$$

- Off-policy MC control:
  - Given episodes generated by policy $\mu$
  - $G_t$ estimates $V_\mu(S_t)$ or $Q_\mu(S_t, A_t)$: $E[G_t \mid S_t] = V_\mu(S_t)$, $E[G_t \mid S_t, A_t] = Q_\mu(S_t, A_t)$
  - Importance Sampling ratio

$$\rho^v_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k \mid S_k)}{\mu(A_k \mid S_k)}$$

$$\rho^q_{t:T-1} = \prod_{k=t+1}^{T-1} \frac{\pi(A_k \mid S_k)}{\mu(A_k \mid S_k)}$$

  - Importance Sampling: $E\left[\rho^v_{t:T-1} G_t \mid S_t\right] = V_\pi(S_t)$, $E\left[\rho^q_{t:T-1} G_t \mid S_t, A_t\right] = Q_\pi(S_t, A_t)$

# Model-free Control: MC, Off-policy

- Weighted importance sampling:

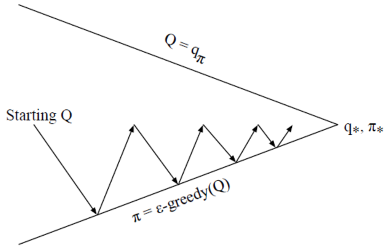
$$Q(s, a) = \frac{\sum_{t \in \mathcal{T}(s)} \rho^{q}_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho^{q}_{t:T(t)-1}}$$

- Incremental method:
  Given episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$
  For $t = T-1, T-2, \dots, 0$:

$$\begin{aligned} G &\leftarrow \gamma G + R_{t+1} \\ C(S_t, A_t) &\leftarrow C(S_t, A_t) + W \\ Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} \left(G - Q(S_t, A_t)\right) \\ W &\leftarrow W \frac{\pi(A_t \mid S_t)}{\mu(A_t \mid S_t)} \end{aligned}$$

## Model-free Control: TD, On-policy $\rightarrow$ Sarsa

- Sarsa: TD(0), $q_\pi^{(1)}$
- n-step Sarsa: n-step TD, $q_\pi^{(n)}$
- Sarsa($\lambda$): forward and backward view TD($\lambda$), $q_\pi^\lambda$



Every time-step:

Policy evaluation Sarsa, $Q \approx q_\pi$

Policy improvement $\epsilon$-greedy policy improvement

## Model-free Control: TD, Off-policy, without importance sampling → Q-learning

- The target policy $\pi$ is greedy w.r.t. $Q(s, a)$

$$\pi(S_{t+1}) = argmax_{a'} Q(S_{t+1}, a')$$

- The behavior policy $\mu$ is e.g. $\epsilon$-greedy w.r.t. $Q(s, a)$
- The Q-learning target then simplifies:

$$\pi(A' \mid S_{t+1}) = \begin{cases} 1 & \text{if } A' = argmax_{a'} Q(S_{t+1}, a') \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} R_{t+1} + \gamma Q(S_{t+1}, A') &= R_{t+1} + \gamma Q\left(S_{t+1}, argmax_{a'} Q(S_{t+1}, a')\right) \\ &= R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') \end{aligned}$$

- Q-Learning

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left( R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t) \right)$$

## Value Function Approximation

- Value function approximation:

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

  For example, linear value function approximation

$$\hat{v}(s, \boldsymbol{w}) = \boldsymbol{x}(s)^\top \boldsymbol{w} = \sum_{j=1}^{n} x_j(s) w_j$$

- Objective function:

$$J(\boldsymbol{w}) = E_\pi \left[ \left( v_\pi(s) - \boldsymbol{x}(s)^\top \boldsymbol{w} \right)^2 \right]$$

- Gradient descent:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{1}{2} \alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w})$$

## Value Function Approximation

- Update rule:

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$$

- Gradient descent finds a local minimum

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

- Stochastic gradient descent samples the gradient finds a global minimum

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

  Update $=$ step-size $\times$ prediction error $\times$ feature value
- Use a *target* to approximate $v_\pi(s)$

## Incremental Prediction Algorithms $v_\pi(s)$

- For MC, the target is the return $G_t$

$$\begin{aligned}\Delta w &= \alpha(G_t - \hat{v}(s_t, \boldsymbol{w}))\nabla_w \hat{v}(s_t, \boldsymbol{w}) \\ &= \alpha(G_t - \hat{v}(s_t, \boldsymbol{w}))\boldsymbol{x}(s_t)\end{aligned}$$

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

$$\begin{aligned}\Delta \boldsymbol{w} &= \alpha(R_{t+1} + \gamma \hat{v}(s_{t+1}, \boldsymbol{w}) - \hat{v}(s_t, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{v}(s_t, \boldsymbol{w}) \\ &= \alpha(R_{t+1} + \gamma \hat{v}(s_{t+1}, \boldsymbol{w}) - \hat{v}(s_t, \boldsymbol{w}))\boldsymbol{x}(s_t)\end{aligned}$$

- For TD($\lambda$), the target is the $\lambda$-return $G_t^\lambda$
  - Forward view linear TD($\lambda$)

$$\begin{aligned}\Delta \boldsymbol{w} &= \alpha(G_t^\lambda - \hat{v}(s_t, \boldsymbol{w}))\nabla_w \hat{v}(s_t, \boldsymbol{w}) \\ &= \alpha(G_t^\lambda - \hat{v}(s_t, \boldsymbol{w}))\boldsymbol{x}(s_t)\end{aligned}$$

  - Backward view linear TD($\lambda$)

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma \hat{v}(s_{t+1}, \boldsymbol{w}) - \hat{v}(s_t, \boldsymbol{w}) \\ E_t &= \gamma \lambda E_{t-1} + \boldsymbol{x}(s_t) \\ \Delta \boldsymbol{w} &= \alpha \delta_t E_t\end{aligned}$$

## Incremental Prediction Algorithms $q_\pi(s, a)$

- For MC, the target is the return $G_t$

$$\begin{aligned}\Delta w &= \alpha(G_t - \hat{q}(s_t, a_t, \boldsymbol{w}))\nabla_w \hat{q}(s_t, a_t, \boldsymbol{w}) \\ &= \alpha(G_t - \hat{q}(s_t, a_t, \boldsymbol{w}))\boldsymbol{x}(s_t, a_t)\end{aligned}$$

- For TD(0), the target is the TD target $R_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, \boldsymbol{w})$

$$\begin{aligned}\Delta \boldsymbol{w} &= \alpha(R_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, \boldsymbol{w}) - \hat{q}(s_t, a_t, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{q}(s_t, a_t, \boldsymbol{w}) \\ &= \alpha(R_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, \boldsymbol{w}) - \hat{q}(s_t, a_t, \boldsymbol{w}))\boldsymbol{x}(s_t, a_t)\end{aligned}$$

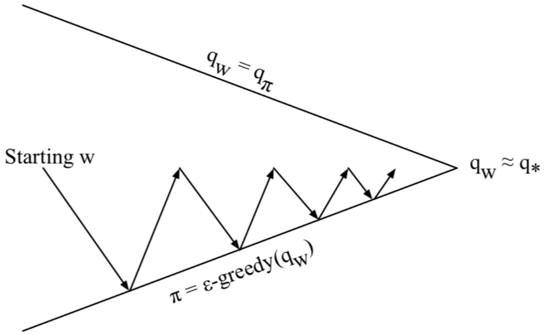
- For TD($\lambda$), the target is the $\lambda$-return $q_t^\lambda$
  - Forward view linear TD($\lambda$)

$$\begin{aligned}\Delta \boldsymbol{w} &= \alpha(q_t^\lambda - \hat{q}(s_t, a_t, \boldsymbol{w}))\nabla_w \hat{q}(s_t, a_t, \boldsymbol{w}) \\ &= \alpha(q_t^\lambda - \hat{q}(s_t, a_t, \boldsymbol{w}))\boldsymbol{x}(s_t, a_t)\end{aligned}$$

  - Backward view linear TD($\lambda$)

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, \boldsymbol{w}) - \hat{q}(s_t, a_t, \boldsymbol{w}) \\ E_t &= \gamma \lambda E_{t-1} + \boldsymbol{x}(s_t, a_t) \\ \Delta \boldsymbol{w} &= \alpha \delta_t E_t\end{aligned}$$

# Control with Value Function Approximation

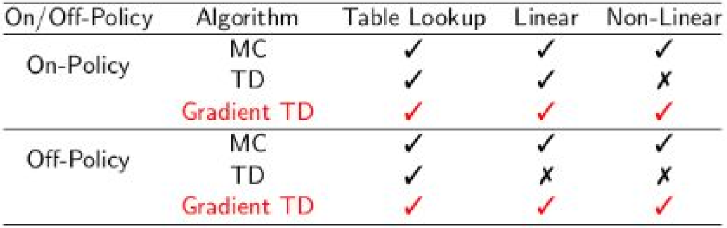


Policy evaluation Approximate policy evaluation, $\hat{q}(\cdot, \cdot, \mathbf{w}) \approx q_\pi$

Policy improvement $\epsilon$-greedy policy improvement

## Convergence of Prediction Algorithms

| On/Off-Policy | Algorithm | Table Lookup | Linear | Non-Linear |
|---|---|---|---|---|
| On-Policy | MC | ✓ | ✓ | ✓ |
| | TD(0) | ✓ | ✓ | ✗ |
| | TD($\lambda$) | ✓ | ✓ | ✗ |
| Off-Policy | MC | ✓ | ✓ | ✓ |
| | TD(0) | ✓ | ✗ | ✗ |
| | TD($\lambda$) | ✓ | ✗ | ✗ |



| On/Off-Policy | Algorithm | Table Lookup | Linear | Non-Linear |
|---|---|---|---|---|
| On-Policy | MC | ✓ | ✓ | ✓ |
| | TD | ✓ | ✓ | ✗ |
| | Gradient TD | ✓ | ✓ | ✓ |
| Off-Policy | MC | ✓ | ✓ | ✓ |
| | TD | ✓ | ✗ | ✗ |
| | Gradient TD | ✓ | ✓ | ✓ |

## Convergence of Control Algorithms

| Algorithm | Table Lookup | Linear | Non-Linear |
|---|---|---|---|
| Monte-Carlo Control | ✓ | (✓) | ✗ |
| Sarsa | ✓ | (✓) | ✗ |
| Q-learning | ✓ | ✗ | ✗ |
| Gradient Q-learning | ✓ | ✓ | ✗ |

(✓) = chatters around near-optimal value function

## Stochastic Gradient Descent with Experience Replay

- Given experience consisting of $\langle \mathit{state}, \mathit{value} \rangle$ pairs

$$D = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \ldots, \langle s_T, v_T^\pi \rangle\}$$

- Least squares algorithms find parameter vector $\boldsymbol{w}$ minimising sum-squared error between $\hat{v}(s_t, \boldsymbol{w})$ and target values $v_t^\pi$

$$LS(\boldsymbol{w}) = \sum_{t=1}^{T} \left(v_t^\pi - \hat{v}(s_t, \boldsymbol{w})\right)^2 = E_D\left[\left(v^\pi - \hat{v}(s, \boldsymbol{w})\right)^2\right]$$

- Repeat:
  1. Sample state, value from experience $\langle s, v^\pi \rangle \sim D$
  2. Apply stochastic gradient descent update

$$\Delta \boldsymbol{w} = \alpha(v^\pi - \hat{v}(s, \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{v}(s, \boldsymbol{w})$$

## Linear Least Squares Prediction

- Using linear value function approximation, $\hat{v}(s, \boldsymbol{w}) = \boldsymbol{x}(s)^\top \boldsymbol{w}$
- From $E_D[\Delta \boldsymbol{w}] = 0$,

$$\boldsymbol{w} = \left( \sum_{t=1}^{T} \boldsymbol{x}(s_t) \boldsymbol{x}(s_t)^\top \right)^{-1} \sum_{t=1}^{T} \boldsymbol{x}(s_t) v_t^\pi$$

- LSMC (Least Squares Monte-Carlo), $v_t^\pi \approx G_t$

$$\boldsymbol{w} = \left( \sum_{t=1}^{T} \boldsymbol{x}(s_t) \boldsymbol{x}(s_t)^\top \right)^{-1} \sum_{t=1}^{T} \boldsymbol{x}(s_t) G_t$$

## Linear Least Squares Prediction

- LSTD (Least Squares Temporal-Difference), $v_t^\pi \approx R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w})$

$$\boldsymbol{w} = \left( \sum_{t=1}^{T} \boldsymbol{x}(s_t)(\boldsymbol{x}(s_t) - \gamma \boldsymbol{x}(s_{t+1}))^\top \right)^{-1} \sum_{t=1}^{T} \boldsymbol{x}(s_t) R_{t+1}$$

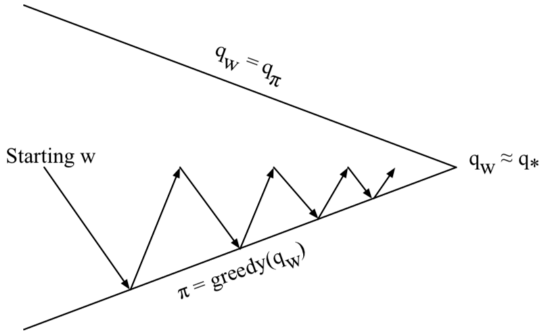
- LSTD($\lambda$) (Least Squares TD($\lambda$)), $v_t^\pi \approx G_t^\lambda$

$$E_t = \gamma \lambda E_{t-1} + \boldsymbol{x}(S_t)$$
$$\boldsymbol{w} = \left( \sum_{t=1}^{T} E_t (\boldsymbol{x}(s_t) - \gamma \boldsymbol{x}(s_{t+1}))^\top \right)^{-1} \sum_{t=1}^{T} E_t R_{t+1}$$

## Convergence of Linear Least Squares Prediction Algorithms

| On/Off-Policy | Algorithm | Table Lookup | Linear | Non-Linear |
|---|---|---|---|---|
| On-Policy | MC | ✓ | ✓ | ✓ |
| | LSMC | ✓ | ✓ | - |
| | TD | ✓ | ✓ | ✗ |
| | LSTD | ✓ | ✓ | - |
| Off-Policy | MC | ✓ | ✓ | ✓ |
| | LSMC | ✓ | ✓ | - |
| | TD | ✓ | ✗ | ✗ |
| | LSTD | ✓ | ✓ | - |

## Least Squares Policy Iteration



| | |
|---|---|
| Policy evaluation | Policy evaluation by least squares Q-learning |
| Policy improvement | Greedy policy improvement |

## Least Squares Q-Learning

- Consider the following linear Q-learning update

$$\begin{aligned} \delta &= R_{t+1} + \gamma \hat{q}(S_{t+1}, \pi(S_{t+1}), \boldsymbol{w}) - \hat{q}(S_t, A_t, \boldsymbol{w}) \\ \Delta \boldsymbol{w} &= \alpha \delta \boldsymbol{x}(S_t, A_t) \end{aligned}$$

- LSTDQ algorithm: solve for total update = zero

$$\begin{aligned} 0 &= \sum_{t=1}^{T} \alpha (R_{t+1} + \gamma \hat{q}(S_{t+1}, \pi(S_{t+1}), \boldsymbol{w}) - \hat{q}(S_t, A_t, \boldsymbol{w})) \boldsymbol{x}(S_t, A_t) \\ \boldsymbol{w} &= \left( \sum_{t=1}^{T} \boldsymbol{x}(S_t, A_t)(\boldsymbol{x}(S_t, A_t) - \gamma \boldsymbol{x}(S_{t+1}, \pi(S_{t+1})))^\top \right)^{-1} \sum_{t=1}^{T} \boldsymbol{x}(S_t, A_t) R_{t+1} \end{aligned}$$

## Least Squares Policy Iteration Algorithm

- LSTDQ for policy evaluation
- repeatedly re-evaluates experience $D$ with different policies

```
function LSPI-TD($\mathcal{D}, \pi_0$)
    $\pi' \leftarrow \pi_0$
    repeat
        $\pi \leftarrow \pi'$
        $Q \leftarrow$ LSTDQ($\pi, \mathcal{D}$)
        for all $s \in \mathcal{S}$ do
            $\pi'(s) \leftarrow \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a)$
        end for
    until ($\pi \approx \pi'$)
    return $\pi$
end function
```

## Convergence of Control Algorithms

| Algorithm | Table Lookup | Linear | Non-Linear |
|---|---|---|---|
| Monte-Carlo Control | ✓ | (✓) | ✗ |
| Sarsa | ✓ | (✓) | ✗ |
| Q-learning | ✓ | ✗ | ✗ |
| LSPI | ✓ | (✓) | - |

(✓ ) = chatters around near-optimal value function

## Table of Contents

## 简介

- 值函数估计介绍了，如何对价值函数进行近似的参数化表达，包括状态价值函数和行为价值函数：

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

- 随后一个策略可以直接从价值函数中产生，比如使用 $\epsilon$-greedy
- 本节将直接参数化策略本身，同时参数化的策略将不再是一个概率集合而是一个函数：

$$\pi_\theta(s, a) = P[a \mid s, \theta]$$

上式将策略函数理解成参数化的策略函数 $\pi_\theta$

## Policy-Based Reinforcement Learning

$$\pi_\theta(s, a) = P[a \mid s, \theta]$$

- 策略函数确定了在给定的状态和一定的参数设置下，采取任何可能行为的概率，因此事实上它是一个概率密度函数。在实际应用策略产生行为时，是按照这个概率分布进行行为采样的。策略函数里的参数决定了概率分布的形态
- 参数化的目的是为了解决大规模问题。在大规模的问题里，把每一个状态严格的独立出来指出某个状态下应该执行某个行为是不太可能的。因此我们需要参数化，用少量的参数来合理近似实际的函数
- 我们要做的是利用参数化的策略函数，通过调整这些参数来得到一个较优策略，遵循这个策略产生的行为将得到较多的奖励。具体的机制是设计一个目标函数，对其使用梯度上升（Gradient Ascent）算法优化参数以最大化奖励

## Parameterized Policy: Softmax Policy

- Softmax Policy 是针对离散的行为常用的一个策略。我们希望有平滑的参数化的策略来决策：针对每一个离散的行为，应该以什么样的概率来执行它。
- 给定 $(s, a)$ 有相应的线性特征 $\phi(s, a)$，$\theta$ 为相应的权重

$$\phi(s,a) = \begin{pmatrix} \phi_1(s,a) \\ \phi_2(s,a) \\ \vdots \\ \phi_n(s,a) \end{pmatrix}$$

$$h(s,a,\theta) = \phi(s,a)^\top \theta$$

- 我们采取某一具体行为的概率与 $e \approx 2.71828$ 的该值次幂成正比：

$$\pi_\theta(s,a) \propto e^{h(s,a,\theta)} = e^{\phi(s,a)^\top \theta}$$

- Softmax policy:

$$\pi_\theta(s,a) = \frac{exp(h(s,a,\theta))}{\sum_b exp(h(s,b,\theta))} = \frac{exp(\phi(s,a)^\top \theta)}{\sum_b exp(\phi(s,b)^\top \theta)}$$

## Parameterized Policy: Gaussian Policy

- 与 Softmax Policy 不同，Gaussian Policy 常应用于连续行为空间。例如，控制机器人行走，要调整流经控制某个电机的电流值，而这是一个连续的取值
- 使用高斯策略时，通常对于均值有一个参数化的表示，同样可以是一些特征的线性代数和：

  $$\mu(s) = \phi(s)^\top \theta$$

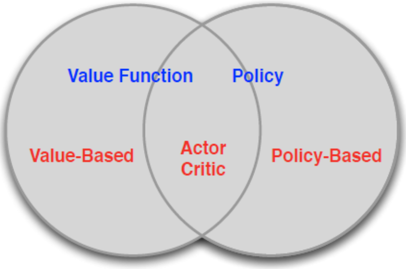
  方差可以是固定值，也可以用参数化表示
- Action $a$ 是连续的，对应于一个具体的数值，该数值从以 $\mu(s)$ 为均值, $\sigma$ 为标准差的高斯分布中随机采样产生：

  $$a \sim \mathcal{N}(\mu(s), \sigma^2)$$

  $$\pi_\theta(a \mid s) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(a - \mu(s))^2}{2\sigma^2}\right)$$

## Value-Based and Policy-Based RL

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



## 基于策略学习的优点

- 优点：
  - 基于策略的学习可能会具有更好的收敛性，这是因为基于策略的学习虽然每次只改善一点点，但总是朝着好的方向在改善；但是上讲提到有些价值函数在后期会一直围绕最优价值函数持续小的震荡而不收敛
  - 在对于那些拥有高维度或连续行动空间来说，使用基于价值函数的学习在得到价值函数后，制定策略时，需要比较各种行为对应的价值大小，这样如果行为空间维度较高或者是连续的，则从中比较得出一个有最大价值函数的行为这个过程就比较难了，这时候使用基于策略的学习就高效的多
  - 能够学到一些随机策略；但是基于价值函数的学习通常是学不到随机策略的
  - 有时候计算价值函数非常复杂。比如当小球从从空中某个位置落下你需要左右移动接住时，计算小球在某一个位置时采取什么行为的价值是很难得；但是基于策略就简单许多，你只需要朝着小球落地的方向移动修改策略就行

## 基于策略学习的缺点

- 缺点：
  - 通常收敛到局部最优而不是全局最优
  - 原始（Naive）的基于策略的学习有时候效率不高，有时候还有较高的变异性（方差，Variance）
    - 基于价值函数的策略决定每次都是推促个体去选择一个最大价值的行为；但是基于策略的，更多的时候策略的选择时仅会在策略某一参数梯度上移动一点点，使得整个的学习比较平滑，因此不够高效
    - 有时候计算朝着梯度方向改变的增量也会有较高的变异性（方差），以至于拖累了整个算法速度，但是通过一些修饰，可以改进

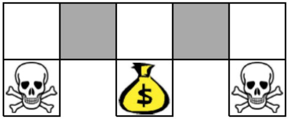
## 随机策略有时是最优策略

- 问题 1： 如果最优策略是随机的，那么用值迭代不能学习到随机策略
- Example： 剪刀—石头—布游戏

- 对于石头剪刀布的游戏，只要一方有一个确定性的策略，就会被对手抓住进而整体上输掉。这个时候最好的策略就是随机选择每次出法，以得到最大可能的总体奖励，即剪刀、石头、布的概率均为 $\frac{1}{3}$
- 如果用 Sarsa, Q-learning 等方法学习 Q 值，并用 $\epsilon$-greedy 选择最优动作，会得到类似这种策略：出剪刀概率为 1-$\epsilon$，其他两个的策略分别为 $\frac{\epsilon}{2}$

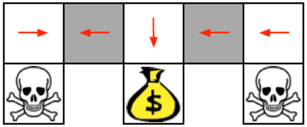
## Example: Aliased Gridworld

问题 2：基于价值函数的策略有时无法得到最优策略



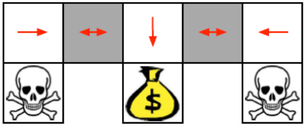
- 在格子世界中，个体需要避免碰到骷髅而尽可能找到钱袋子。在上方的 5 个格子组成的“长廊”中，当以某些对个体来说较容易观测的特征来描述状态空间时，灰色的两个格子将会是无法区分的
- 当我们用某一个格子的某个方向是否有墙挡住这些特征来描述格子状态，也就是作为格子世界状态空间的特征时，就会发生灰色格子状态一样的情况，这就是状态重名（Aliased）
- 当我们可以用“某格子在北面有墙，同时向东移步”来作为状态行为空间的特征时，也会发生上述情况

## Example: Aliased Gridworld



- 当发生格子重名的（Aliased）情况时，如果采用确定性的策略话，在个体处于无论哪个灰色格子时，都只能选取相同的行为。假设个体现在学到了一个价值函数，在这个价值函数里状态就是基于上述特征的参数化表示，此时当个体处在灰色格子中，如果采取的是 greedy 执行的方式，价值函数给出的策略要么都是向东，要么都是向西。如果是向西，那么当个体处在左侧灰色格子时，它将一直（对于 greedy 执行）或很长时间（对于 $\epsilon$-greedy 执行）徘徊在长廊左侧两个格子之间而无法到达有钱袋子的格子，因而很长时间得不到奖励

## Example: Aliased Gridworld



- 当发生状态重名情况时，随机策略将会优于确定性的策略。之前的理论告诉我们对于任何 MDP 总有一个确定性的最优策略。不过那是针对状态可完美观测、或者使用的特征可以完美描述状态的情况下的。当发生状态重名无法区分或者使用的近似函数里描述状态的特征限制了对状态的完美描述时，个体得到的状态信息等效于部分观测的环境信息，问题将不具备马尔科夫性。此时最优策略将不再是确定性的。而直接基于策略的学习将能学习到最优策略，这就是我们为什么要直接基于策略进行强化学习的原因

## Table of Contents

## 策略目标函数（Objective Functions）

- 目标：给定 Policy $\pi_\theta(s, a)$，找到最好的参数 $\theta$
- 如何评价策略 $\pi_\theta(s, a)$？直觉解释
    - $\tau$ 为 episode，其总回报为 $r(\tau)$

$$\tau = (S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, \ldots)$$

$$r(\tau) = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

    - $\pi_\theta(\tau)$ 为给定 Policy $\pi_\theta$ 下，episode $\tau$ 的概率分布
    - 目标函数为：

$$J(\theta) = E_{\tau \sim \pi_\theta(\tau)}\left[r(\tau)\right]$$

## Policy Objective Functions: Start value

针对不同的问题类型，有三种具体的目标函数可以选择：

- **Start value**：在能够产生完整 Episode的环境下，也就是在个体可以到达终止状态时，我们可以用这样一个值来衡量整个策略的优劣：从某状态 $s_1$ 算起知道终止状态个体获得的累计奖励。这个值称为 start value。这个数值的意思是说：如果个体总是从某个状态 $s_1$ 开始，或者以一定的概率分布从 $s_1$ 开始，那么从该状态开始到 Episode 结束个体将会得到怎样的期望最终奖励

$$J_1(\theta) = V^{\pi_\theta}(s_1) = E_{\tau \sim \pi_\theta(\tau)}[r(\tau) \mid S_t(\tau) = s_1]$$

## Policy Objective Functions

- **Average Value**：对于连续环境条件，不存在一个开始状态，这个时候可以使用 average value。考虑我们个体在某时刻处在某状态下的概率，也就是个体在该时刻的状态分布，针对每个可能的状态计算从该时刻开始一直持续与环境交互下去能够得到的奖励，按该时刻各状态的概率分布求和：

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

- **Average reward per time-step**：我们可以使用每一个时间步长在各种情况下所能得到的平均奖励，也就是说在一个确定的时间步长里，查看个体处于所有状态的可能性，然后每一种状态下采取所有行为能够得到的即时奖励，所有奖励按概率求和得到：

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

$$\mathcal{R}_s^a = E[R_{t+1} \mid S_t = s, A_t = a]$$

注：$d^{\pi_\theta}(s)$ 是在当前策略下马尔科夫链的关于状态的一个静态分布。

## 优化目标函数

- Policy based reinforcement learning is an optimisation problem
- Find $\theta$ that maximises $J(\theta)$
- Some approaches do not use gradient
  - Hill climbing（爬山法）
  - Simulated annealing（模拟退火）
  - Genetic algorithms（遗传算法）
- Greater efficiency often possible using gradient
  - Gradient descent（梯度下降）
  - Conjugate gradient（共轭梯度法）
  - Newton / Quasi-Newton（牛顿法 / 拟牛顿法）
- 本讲内容将主要聚焦于使用梯度的策略优化，同时使用基于序列结构片段（Sequential structure）的方法
  - 我们选取个体与环境交互中的一个序列结构片段，通过这种序列结构片段来学习，优化策略进而知道个体后续与环境的交互

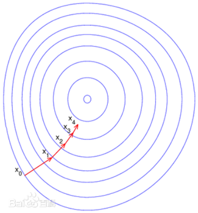
## 策略梯度（Policy Gradient）

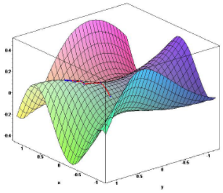
- 令 $J(\theta)$ 为任何类型的策略目标函数，策略梯度算法可以使 $J(\theta)$ 沿着其梯度上升至局部最大值（local maximum）。同时确定获得最大值时的参数 $\theta$（$\alpha$ 为步长参数）：

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

- 其中 $\nabla_\theta J(\theta)$ 是策略梯度 (policy gradient)

$$\nabla J(\theta) = \left( \frac{\partial J(\theta)}{\partial \theta_1}, \frac{\partial J(\theta)}{\partial \theta_2}, \dots, \frac{\partial J(\theta)}{\partial \theta_n} \right)^\top$$





## 有限差分法（Finite Differences）计算策略梯度

- 给定策略 $\pi_\theta(s, a)$，当 $J(\theta)$ 不可微分时，可以采用有限差分法计算 $\nabla_\theta J(\theta)$
- 这是非常常用的数值计算方法，特别是当梯度函数本身很难得到的时候
- 针对参数 $\theta$ 的每一个分量 $\theta_k$，使用如下的公式粗略计算梯度：

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

  $u_k$ 是一个单位向量，仅在第 $k$ 个维度上值为 1，其余维度为 0
- 有限差分法简单，不要求策略函数可微分，适用于任意策略；但有噪声，且大多数时候不高效

## $J(\theta)$ 的梯度

- 给定 Policy $\pi_\theta$，目标函数 $J(\theta)$ 为

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)] = \int \pi_\theta(\tau) r(\tau)\, \mathrm{d}\tau$$

- 一个有用的公式

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

- 策略梯度（policy gradient）为

$$\begin{aligned}\nabla_\theta J(\theta) &= \int \nabla_\theta \pi_\theta(\tau) r(\tau)\, \mathrm{d}\tau = \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) r(\tau)\, \mathrm{d}\tau \\ &= E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]\end{aligned}$$

- 对于期望，我们总可以通过 sampling 来近似；怎么计算 $\nabla_\theta \log \pi_\theta(\tau)$ ?

## $\nabla_\theta \log \pi_\theta(\tau)$

- $\pi_\theta(\tau)$ 的定义

$$\begin{aligned}\pi_\theta(\tau) &= \pi_\theta(S_t, A_t, R_{t+1}, \ldots, S_T, A_T) \\ &= P(S_t)\prod_{k=t}^{T} \pi_\theta(A_k \mid S_k) P(S_{k+1} \mid S_k, A_k)\end{aligned}$$

- Take the log:

$$\log \pi_\theta(\tau) = \log P(S_t) + \sum_{k=t}^{T} \log \pi_\theta(A_k \mid S_k) + \log P(S_{k+1} \mid S_k, A_k)$$

- $\nabla_\theta \log \pi_\theta(\tau)$

$$\begin{aligned}&= \nabla_\theta \left[ \cancel{\log P(S_t)} + \sum_{k=t}^{T} \log \pi_\theta(A_k \mid S_k) + \cancel{\sum_{k=t}^{T} \log P(S_{k+1} \mid S_k, A_k)} \right] \\ &= \nabla_\theta \sum_{k=t}^{T} \log \pi_\theta(A_k \mid S_k) = \sum_{k=t}^{T} \nabla_\theta \log \pi_\theta(A_k \mid S_k)\end{aligned}$$

## $J(\theta)$ 的梯度

- 由于去掉了 $P(S_{k+1} \mid S_k, A_k)$，我们得到 Model-free 的更新规则

$$
\begin{aligned}
\nabla_\theta J(\theta) &= E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)] \\
&= E_{\tau\sim\pi_\theta(\tau)}\left[\left(\sum_{k=t}^{T} \nabla_\theta \log \pi_\theta(A_k|S_k)\right) r(\tau)\right] \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{k=t_i}^{T_i} \nabla_\theta \log \pi_\theta(A_k^i|S_k^i)\right)\left(\sum_{k=t_i}^{T_i} R(S_k^i, A_k^i)\right) \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{k=t_i}^{T_i} \nabla_\theta \log \pi_\theta(A_k^i|S_k^i)\left(\sum_{k'=k}^{T_i} R(S_{k'}^i, A_{k'}^i)\right)\right) \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{k=t_i}^{T_i} \nabla_\theta \log \pi_\theta(A_k^i|S_k^i)\left(\sum_{k'=k}^{T_i} \gamma^{k'-k} R(S_{k'}^i, A_{k'}^i)\right)\right)
\end{aligned}
$$

## One-Step MDPs

- One-Step MDPs: 从一个分布 $d(s)$ 中采样得到一个状态 $s$，从 $s$ 开始，采取一个行为 $a$，得到即时奖励 $r = \mathcal{R}_s^a$ 然后终止
- 由于是单步过程，因此三种目标函数的形式是一样的：

$$J(\theta) = E_{\pi_\theta}[r] = \sum_s d(s) \sum_a \pi_\theta(s,a)\mathcal{R}_s^a$$

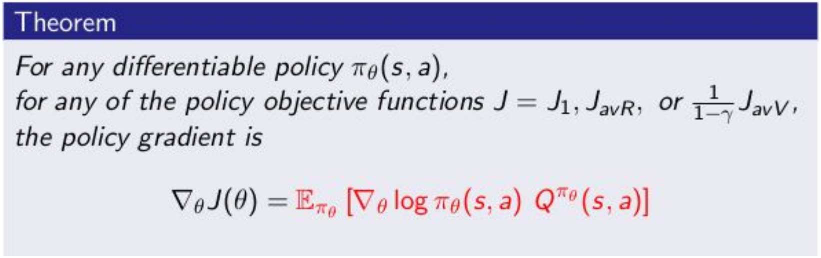
- 相应梯度为：

$$\begin{aligned}
\nabla_\theta J(\theta) &= \sum_s d(s) \sum_a \nabla_\theta \pi_\theta(s,a)\mathcal{R}_s^a \\
&= \sum_s d(s) \sum_a \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)\mathcal{R}_s^a \\
&= E_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\mathcal{R}_s^a\right]
\end{aligned}$$

## 策略梯度定理（Policy Gradient Theorem）

- 对于多步 MDPs，只需要将 $\mathcal{R}_s^a$ 替换为 $Q^{\pi_\theta}(s,a)$
- 策略梯度定理（Policy Gradient Theorem），对于 differentiable policy $\pi_\theta(s,a)$

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



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*

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

## Policy Gradient Theorem

- 如何直观理解 policy gradient theorem ?

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)]$$

  - 上式红色部分：看做采取某个动作的方向
  - 上式绿色部分：看做采取该动作后的 reward
  - 奖励：如果 reward 比较大，那么对这个动作的更新权重就大，下次采取该动作的可能性增加
  - 惩罚：如果 reward 比较小，或者为负数，那么对这个动作更新权重小，下次采取该动作的可能性减少

## Score Function

- Policy gradient theorem:
  $\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)]$
- $\nabla_\theta \log \pi_\theta(s, a)$ 称为 score function
- Softmax Policy

$$\pi_\theta(s, a) = \frac{exp(\phi(s, a)^\top \theta)}{\sum_b exp(\phi(s, b)^\top \theta)}$$

$$\begin{aligned}
\nabla_\theta \log \pi_\theta(s, a) &= \nabla_\theta \left( \phi(s, a)^\top \theta - \log \sum_b exp(\phi(s, b)^\top \theta) \right) \\
&= \phi(s, a) - \frac{\sum_b exp(\phi(s, b)^\top \theta) \phi(s, b)}{\sum_b exp(\phi(s, b)^\top \theta)} \\
&= \phi(s, a) - E_{\pi_\theta}[\phi(s, \cdot)]
\end{aligned}$$

- Gaussian Policy

$$\pi_\theta(s, a) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left( -\frac{(a - \mu(s))^2}{2\sigma^2} \right)$$

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

## Table of Contents

## REINFORCE: Monte Carlo Policy Gradient

- 针对具有完整 Episode 的情况，基于 Policy Gradient Theorem，很快我们就可以构建第一个 PG 算法-REINFORCE(Williams,1992)

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t \mid s_t) Q^{\pi_\theta}(s_t, a_t)]$$

- 如果用 MC 计算回报 $G_t$ 的方法，代替 $Q^{\pi_\theta}(s_t, a_t)$

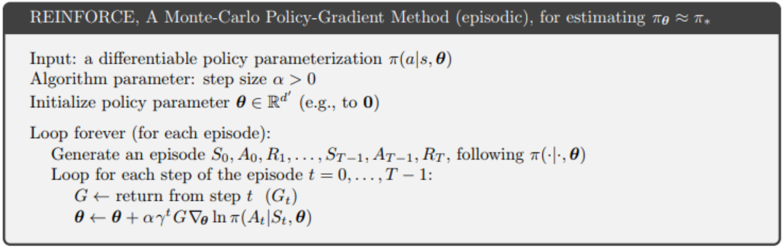
$$\begin{aligned}\nabla_\theta J(\theta) &= E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t \mid s_t) Q^{\pi_\theta}(s_t, a_t)] \\ &\approx E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t \mid s_t) G_t]\end{aligned}$$

- 采用随机梯度下降更新 $\theta$

$$\theta_{t+1} = \theta_t + \alpha G_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

## REINFORCE: Monte Carlo Policy Gradient

Sutton's algorithm: "Reinforcement Learning: An Introduction"

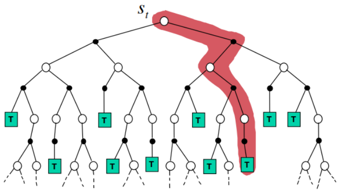


```
REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
    Loop for each step of the episode $t = 0, \ldots, T-1$:
        $G \leftarrow$ return from step $t$   $(G_t)$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t,\boldsymbol{\theta})$
```

MC 无偏，但高方差估计



## REINFORCE: Monte Carlo Policy Gradient

Silver's REINFORCE algorithm

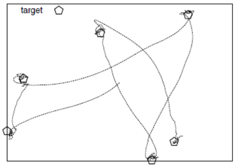
```
function REINFORCE
    Initialise $\theta$ arbitrarily
    for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ do
        for $t = 1$ to $T - 1$ do
            $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
        end for
    end for
    return $\theta$
end function
```

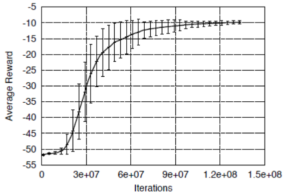
Note that the $G_t$ item in Sutton's REINFORCE algorithm and the $v_t$ item in Silver's REINFORCE algorithm are the same things

$$G_t = R_{t+1} + \gamma \times R_{t+2} + \gamma^2 \times R_{t+3} + \cdots + \gamma^{T-t+1} \times R_T$$

However, Silver's REINFORCE algorithm lacked a $\gamma^t$ item than Sutton's algorithm. It turned out that both of the algorithms are correct. Sutton's algorithm worked for the episodic case maximizing the value of start state, while Silver's algorithm worked for the continuing case maximizing the averaged value.

## Puck World Example





- 在区域里追踪一个目标的例子：有一个目标物体，同时还有一个 Agent
- 状态空间：个体观察自己的位置 $(x, y)$，速度 $(v_x, v_y)$ 以及目标物体（图中的五角形）的位置 $(t_x, t_y)$，共 6 个特征
- 行为空间：个体控制自己在上、下、左、右四个方向上的油门（速率的增量），和不操作 5 个行为
- 环境动力学：将个体的行为转化为其速度和位置的变化。目标物体出现位置随机，且每 30 秒时间更新位置
- 奖励：奖励值的大小基于个体与目标物体之间的距离，距离越小奖励越大
- 使用蒙特卡洛策略梯度算法收敛速度慢，需要的迭代次数长，还存在较高的方差

## Table of Contents

## Reducing Variance Using a Critic

- 使用蒙特卡洛策略梯度方法使用了收获作为状态价值的估计，它虽然是无偏的，但是噪声却比较大，方差较高
  - 采用 TD ? $\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$
- 用 Cirtic 相对准确地估计状态价值，用它来指导策略更新

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

- 基于 Actor-Critic 策略梯度学习分为两部分内容：
  1. Critic：参数化行为价值函数 $Q_w(s, a)$
  2. Actor：按照 Critic 部分得到的价值引导策略函数参数 $\theta$ 的更新
- 这样，Actor-Critic 算法遵循的是一个近似的策略梯度：

$$\nabla_\theta J(\theta) \approx E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)]$$
$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$$

## 估计 Action-Value 函数

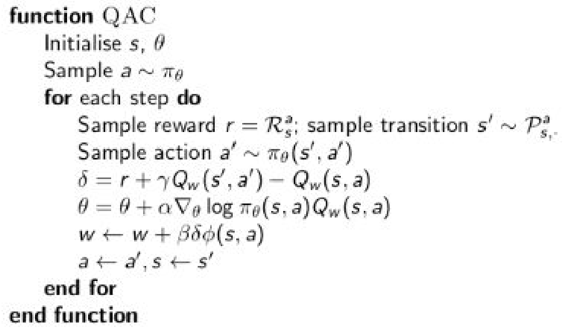
- Critic 做的事情上节课已经介绍了：策略评估，它要告诉个体，在由参数 $\theta$ 确定的策略 $\pi_\theta$ 到底表现得怎么样
- 如何更好地估计 action-value 函数?
  - MC policy evaluation
  - TD 学习
  - TD($\lambda$)
  - 批处理方法：least-square

## Action-Value Actor-Critic

- 一个简单的 actor-critic 算法可以使用基于行为价值的 critic，它使用一个线性价值函数来近似状态行为价值函数：

$$Q_w(s,a) = \phi(s,a)^\top w$$

- 其中 Critic 通过线性近似的 TD(0) 更新 $w$，Actor 通过策略梯度更新 $\theta$。具体算法流程如下：



```
function QAC
    Initialise $s$, $\theta$
    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
```

## Bias in Actor-Critic Algorithms

- 用特征的线性组合来近似 $Q_w(s, a)$ 进而求解策略梯度的方法引入了偏倚
- 一个偏倚的值函数下得到的策略梯度不一定能最后找到较好的解决方案
  - 例如当近似价值函数的 $Q_w(s, a)$ 使用可能会引起状态重名的特征时，还能解决那个格子世界问题吗（指前文提到的在格子世界里找钱袋子的问题），答案是不一定
- 不过幸运的是，如果我们小心设计近似的 $Q_w(s, a)$ 函数，是可以避免引入偏倚的，这样我们相当于遵循了准确的策略梯度

## 兼容近似函数（Compatible Function Approximation）

- 那么怎样才算是一个小心设计了的 $Q_w(s,a)$ 呢？需要满足下面两个条件:
  - 近似值函数的梯度完全等同于策略函数对数的梯度，即不存在重名情况：

$$\nabla_w Q_w(s,a) = \nabla_\theta log\, \pi_\theta(s,a)$$

  - 价值函数参数 $w$ 使得均方差最小：

$$\epsilon = E_{\pi_\theta}[(Q^{\pi_\theta}(s,a) - Q_w(s,a))^2]$$
$$\nabla_w \epsilon = 0$$

- 符合这两个条件，则认为策略梯度是准确的，此时：

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)]$$

## Reducing Variance Using Baseline

- 基本思想是从策略梯度里抽出一个基准函数 $B(s)$，要求这一函数仅与状态有关，与行为无关，因而不改变梯度本身
- $B(s)$ 的特点是能在不改变行为价值期望的同时降低其 Variance

$$
\begin{aligned}
E_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) B(s)\right] &= \sum_s d^{\pi_\theta}(s) \sum_s \nabla_\theta \pi_\theta(s,a) B(s) \\
&= \sum_s d^{\pi_\theta} B(s) \nabla_\theta \sum_a \pi_\theta(s,a) \\
&= \sum_s d^{\pi_\theta} B(s) \nabla_\theta 1 = 0
\end{aligned}
$$

- 原则上，和行为无关的函数都可以作为 $B(s)$。一个很好的 $B(s)$ 就是基于当前状态的状态价值函数：

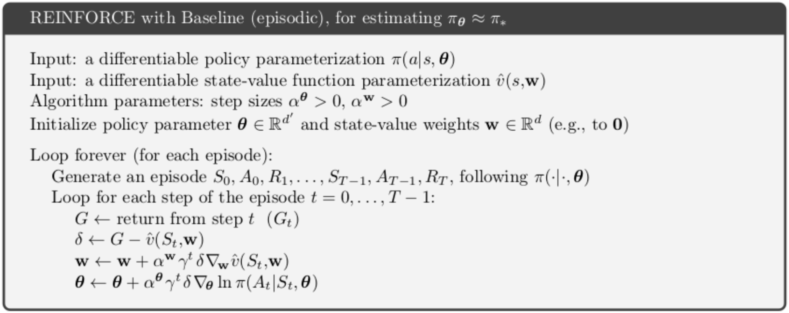
$$B(s) = V^{\pi_\theta}(s)$$

- 定义 advantage function，并且重写目标函数的梯度：

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

# REINFORCE with Baseline

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a)(G_t - V_w(s))]$$

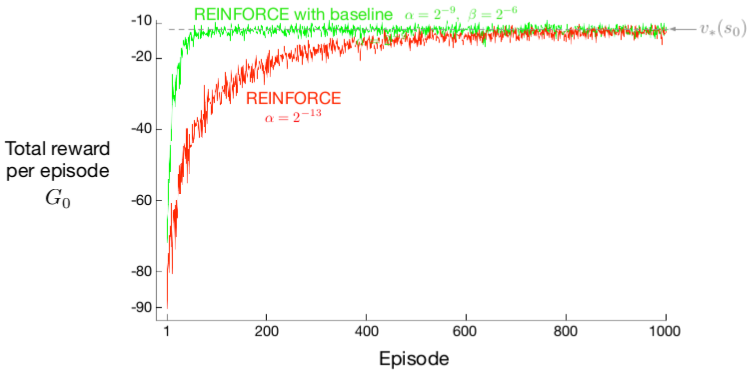


```
REINFORCE with Baseline (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    Loop for each step of the episode $t = 0, \ldots, T-1$:
        $G \leftarrow$ return from step $t$  $(G_t)$
        $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \gamma^t \delta \nabla_{\mathbf{w}} \hat{v}(S_t,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$
```

# REINFORCE with Baseline



## Estimating the Advantage Function

- Advantage 函数可以明显减少状态价值的变异性，因此算法的 Critic 部分可以去估计 advantage 函数而不是仅仅估计行为价值函数
- 在这种情况下，我们需要两个近似函数也就是两套参数，一套用来近似状态价值函数，一套用来近似行为价值函数，以便计算 advantage 函数，可以通过 TD 学习来更新这两个价值函数。数学表示如下：

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

- 不过实际操作时，并不需要这样做，有简化方式

## Estimating the Advantage Function

- 根据定义，TD 误差 $\delta^{\pi_\theta}$ 可以根据真实的状态值函数 $V^{\pi_\theta}(s)$ 算出：

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

- 这样得到的 TD 误差是 advantage 函数的无偏估计，这同样是根据行动值函数的定义推导成立的，即：

$$\begin{aligned} E_{\pi_\theta}[\delta^{\pi_\theta} \mid s, a] &= E_{\pi_\theta}[r + \gamma V^{\pi_\theta}(s') \mid s, a] - V^{\pi_\theta}(s) \\ &= Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s) \\ &= A^{\pi_\theta}(s, a) \end{aligned}$$

- 如此，我们就可以使用 TD 误差来计算策略梯度：

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a)\delta^{\pi_\theta}]$$

- 实际运用时，我们使用一个近似的 TD 误差，即用状态函数的近似函数来代替实际的状态函数：

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

## 针对 Critic 过程使用 TD($\lambda$)

通过计算不同时间范围内（步长）的 TD 误差来更新状态价值函数 $V_v(s)$，此时的 Critic 过程可以根据时间范围的的长短（步长的多少）来分为:

- MC，直至 Episode 结束：

$$\Delta v = \alpha(G_t - V_v(s))\phi(s)$$

- TD(0), 一步：

$$\Delta v = \alpha(r + \gamma V_v(s') - V_v(s))\phi(s)$$

- TD($\lambda$) 的前向视角，需要至 Episode 结束：

$$\Delta v = \alpha(G_t^\lambda - V_v(s))\phi(s)$$

- TD($\lambda$) 的后向视角，实时，具备频率记忆和近时记忆功能：

$$\begin{aligned}\delta_t &= r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)\\ e_t &= \gamma\lambda e_{t-1} + \phi(s_t)\\ \Delta v &= \alpha\delta_t e_t\end{aligned}$$

## 针对 Actor 过程使用 TD($\lambda$)

同样在 Actor 过程中也可以把时间范围考虑进去用来更新参数

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) A^{\pi_\theta}(s, a)]$$

- MC，直至 Episode 结束：

$$\Delta\theta = \alpha(G_t - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- TD(0)，一步：

$$\Delta\theta = \alpha(r + \gamma V_v(s_{t+1}) - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- TD($\lambda$) 的前向视角，需要至 Episode 结束：

$$\Delta\theta = \alpha(G_t^\lambda - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- TD($\lambda$) 的后向视角，实时，具备频率记忆和近时记忆功能：

$$\delta_t = r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)$$
$$e_t = \gamma\lambda e_{t-1} + \nabla_\theta \log \pi_\theta(s_t, a_t)$$
$$\Delta\theta = \alpha\delta_t e_t$$

对于 Critic 和 Actor，将 TD($\lambda$) 的后向视角算法应用于实际问题

## Actor-Critic with eligibility traces

- Critic with forward-view TD($\lambda$) 更新规则

$$\Delta v = \alpha(G_t^{\lambda} - V_v(s_t))\phi(s_t)$$

- Critic with backward-view TD($\lambda$) 更新规则

$$\begin{aligned}\delta_t &= r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)\\ e_t &= \gamma\lambda e_{t-1} + \phi(s_t)\\ \Delta v &= \alpha\delta_t e_t\end{aligned}$$

- Actor-critic with forward-view TD($\lambda$) 更新规则

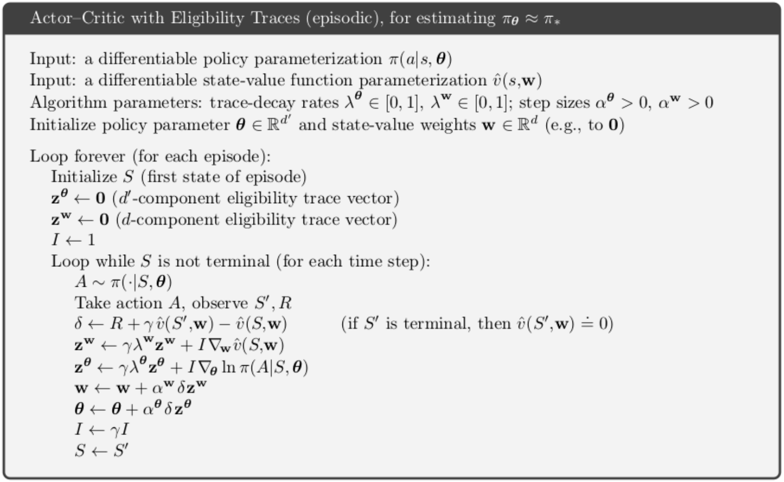
$$\Delta\theta = \alpha(G_t^{\lambda} - V_v(s_t))\nabla_\theta log\, \pi_\theta(s_t, a_t)$$

- Actor-Critic with backward-view TD($\lambda$) 更新规则

$$\begin{aligned}\delta_t &= r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)\\ e_t &= \gamma\lambda e_{t-1} + \nabla_\theta log\, \pi_\theta(s_t, a_t)\\ \Delta\theta &= \alpha\delta_t e_t\end{aligned}$$

- 这些方法都可以在线更新，用于非完整的片段

# actor-critic with eligibility traces



**Actor–Critic with Eligibility Traces (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$**

```
Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: trace-decay rates $\lambda^{\boldsymbol{\theta}} \in [0,1]$, $\lambda^{\mathbf{w}} \in [0,1]$; step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Initialize $S$ (first state of episode)
    $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$ ($d'$-component eligibility trace vector)
    $\mathbf{z}^{\mathbf{w}} \leftarrow \mathbf{0}$ ($d$-component eligibility trace vector)
    $I \leftarrow 1$
    Loop while $S$ is not terminal (for each time step):
        $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
        Take action $A$, observe $S', R$
        $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$        (if $S'$ is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{z}^{\mathbf{w}} \leftarrow \gamma\lambda^{\mathbf{w}}\mathbf{z}^{\mathbf{w}} + I\nabla_{\mathbf{w}}\hat{v}(S,\mathbf{w})$
        $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \gamma\lambda^{\boldsymbol{\theta}}\mathbf{z}^{\boldsymbol{\theta}} + I\nabla_{\boldsymbol{\theta}} \ln \pi(A|S, \boldsymbol{\theta})$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\mathbf{z}^{\mathbf{w}}$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\delta\mathbf{z}^{\boldsymbol{\theta}}$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$
```

# 小结

- 本章主要介绍了 policy gradient 算法，一种直接学习策略的方法

$$\pi_\theta(s,a) \approx P[a|s,\theta]$$

- Policy Gradient Theorem

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t \mid s_t) Q^{\pi_\theta}(s,a)]$$

- REINFORCE 利用 MC 去估计 $Q^{\pi_\theta}(s,a)$（高方差）
- Actor-Critic 算法，是值估计和策略梯度两种方法的混合体

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ v_t] && \text{REINFORCE}\\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)] && \text{Q Actor-Critic}\\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ A^w(s,a)] && \text{Advantage Actor-Critic}\\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ \delta] && \text{TD Actor-Critic}\\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ \delta e] && \text{TD}(\lambda)\text{ Actor-Critic}
\end{aligned}$$