机器学习
Machine Learning

强化学习-2

Reinforcement Learning-2

（函数逼近、策略梯度、连续动作）

1. 值函数逼近
Value Function Approximation

大规模强化学习问题
Large-Scale Reinforcement Learning

状态取值空间 $\mathcal{S} = \{s^{(1)}, s^{(2)}, \cdots, s^{(K_{\mathcal{S}})}\}$

动作的取值空间 $\mathcal{A} = \{a^{(1)}, a^{(2)}, \cdots, a^{(K_{\mathcal{A}})}\}$

状态空间和/或动作空间巨大甚至取值连续的情况下，经典的表格方法不再适用，这时可用函数逼近的方法表示值函数。

1.1 值函数逼近解大规模MDP问题

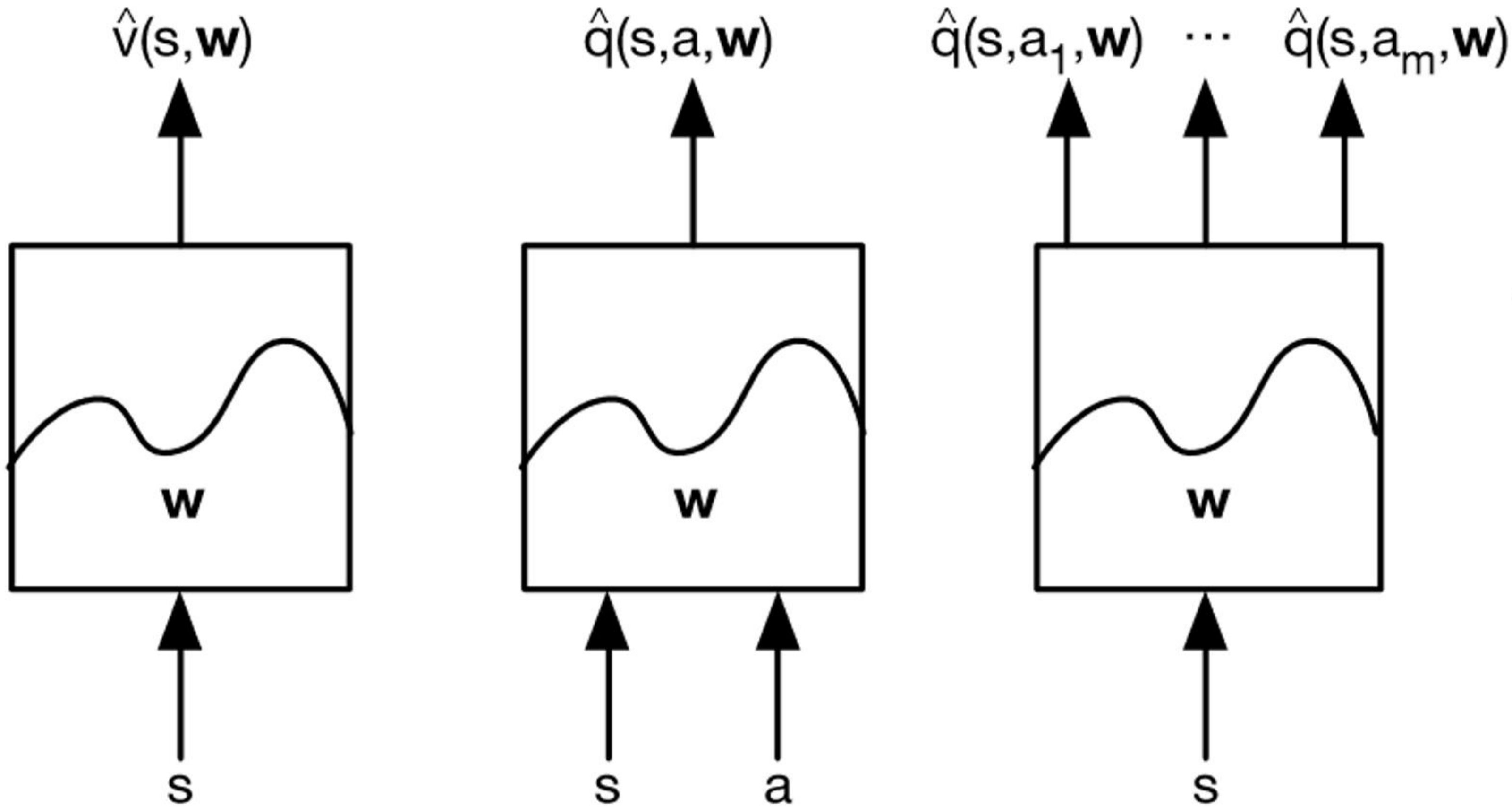
值函数逼近：用一种参数化的函数分别表示值函数和动作-值函数

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

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

值函数逼近的几种类型

可用监督学习中的参数回归模型表示一种值函数



注：其中第3种情况是状态空间巨大，但只有很少动作的情况下，可针对每一动作给出“动作值函数”

1.2 值函数逼近的随机梯度方法

- 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

特征向量
Feature Vector

为了有效表示值函数，用特征向量表示状态

- 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

线性值函数逼近
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 $\mathbf{w}$

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

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

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

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

Update $=$ *step-size* $\times$ *prediction error* $\times$ *feature value*

例：表方法可认为是线性值函数逼近的特例

- Table lookup is a special case of linear value function approximation
- Using *table lookup features*

$$\mathbf{x}^{table}(S) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix}$$

- Parameter vector $\mathbf{w}$ gives value of each individual state

$$\hat{v}(S, \mathbf{w}) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix}^\top \cdot \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_n \end{pmatrix}$$

1.3 增量类值函数预测算法

- 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

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

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

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

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

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

MC-增量类值函数预测算法

- 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*

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

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

TD（0）-增量类值函数预测算法

- 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":

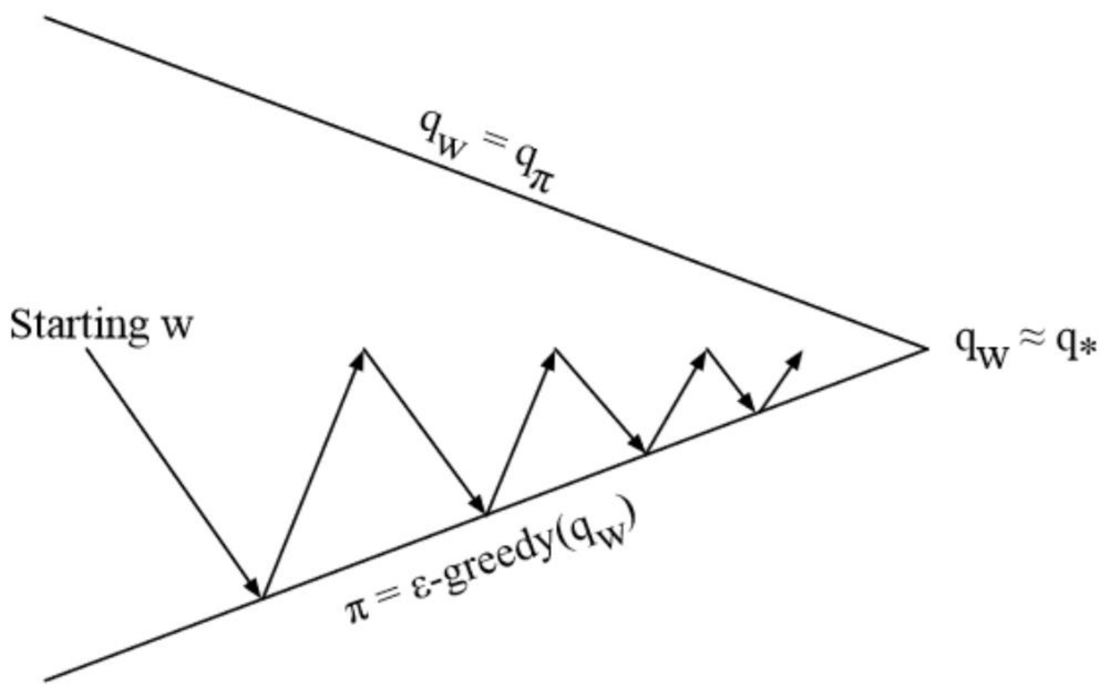
$$\langle S_1, R_2 + \gamma \hat{v}(S_2, \mathbf{w})\rangle, \langle S_2, R_3 + \gamma \hat{v}(S_3, \mathbf{w})\rangle, ..., \langle S_{T-1}, R_T\rangle$$

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

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

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

1.4 增量类控制算法



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

Policy improvement ϵ-greedy policy improvement

实际中用“动作-值函数Q”取代值函数V

动作-值函数逼近

- 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

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

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

线性动作-值函数逼近

- 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

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

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

线性动作-值函数的增量逼近算法

- Like prediction, we must substitute a *target* for $q_\pi(S, A)$
  - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

  - For TD(0), the target is the TD target $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

  - For forward-view TD(λ), target is the action-value λ-return

$$\Delta \mathbf{w} = \alpha(q_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

  - For backward-view TD(λ), equivalent update is

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}) \\
E_t &= \gamma\lambda E_{t-1} + \nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w}) \\
\Delta \mathbf{w} &= \alpha\delta_t E_t
\end{aligned}$$

结合贪婪改进策略算法，构造控制算法。

函数逼近情况下的策略改进

选择当前逼近函数下的最优动作

$$A^{*}=\underset{a}{\arg\max}\left\{\hat{q}\left(S_{t},a,\boldsymbol{w}\right)\right\}$$

$\varepsilon-$ 贪婪策略

$$\pi\left(a \middle| A_{t}\right)=\begin{cases}1-\varepsilon+\dfrac{\varepsilon}{\|\mathcal{A}\|}, & 若 a=A^{*}\\ \dfrac{\varepsilon}{\|\mathcal{A}\|}, & 其他\end{cases}$$

1.5 基本函数逼近方法的收敛性

值函数预测的收敛性

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

控制过程的收敛性

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

(✓) = chatters around near-optimal value function

1.6 神经网络Q函数逼近和学习

Given experience consisting of $\langle state, value \rangle$ pairs

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

Repeat:

1. Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

2. Apply stochastic gradient descent update

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

Converges to least squares solution

$$\mathbf{w}^\pi = \underset{\mathbf{w}}{\operatorname{argmin}}\ LS(\mathbf{w})$$

Deep Q-Networks： DQN

为保证收敛，引入经验回放（experience replay）和目标Q网络

DQN uses experience replay and fixed Q-targets

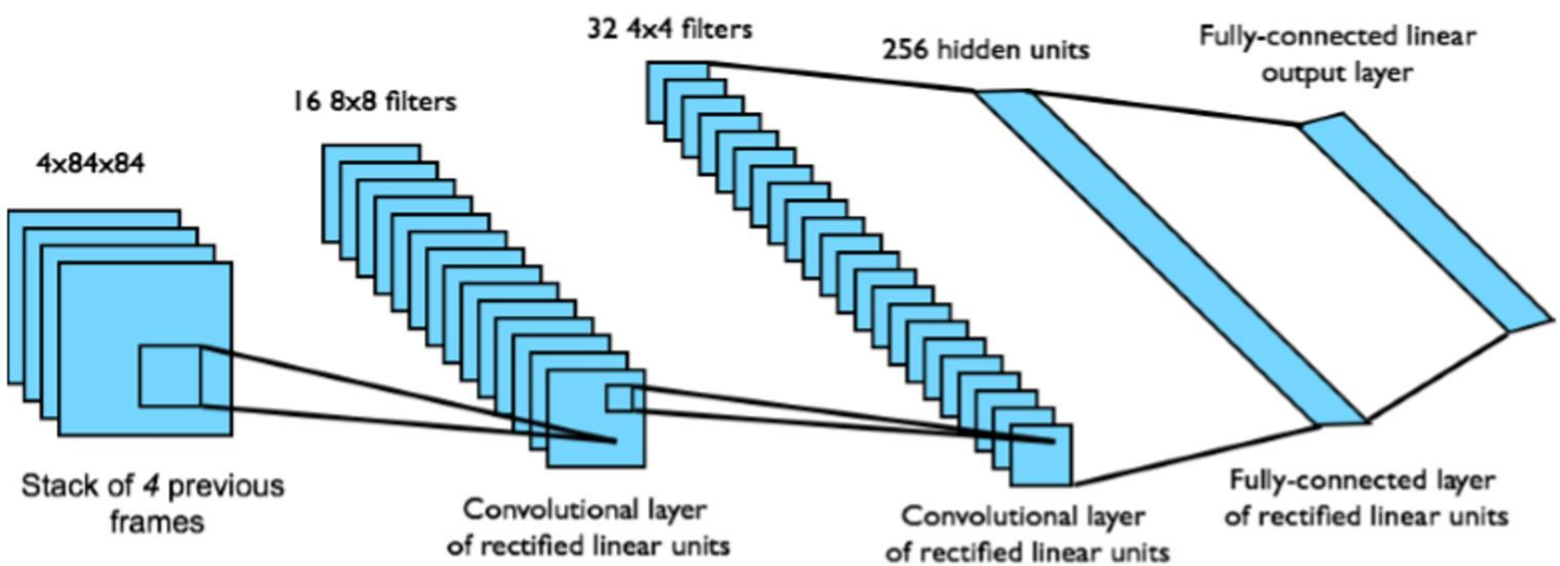
- Take action a_t according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters w^-
- Optimise MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

- Using variant of stochastic gradient descent

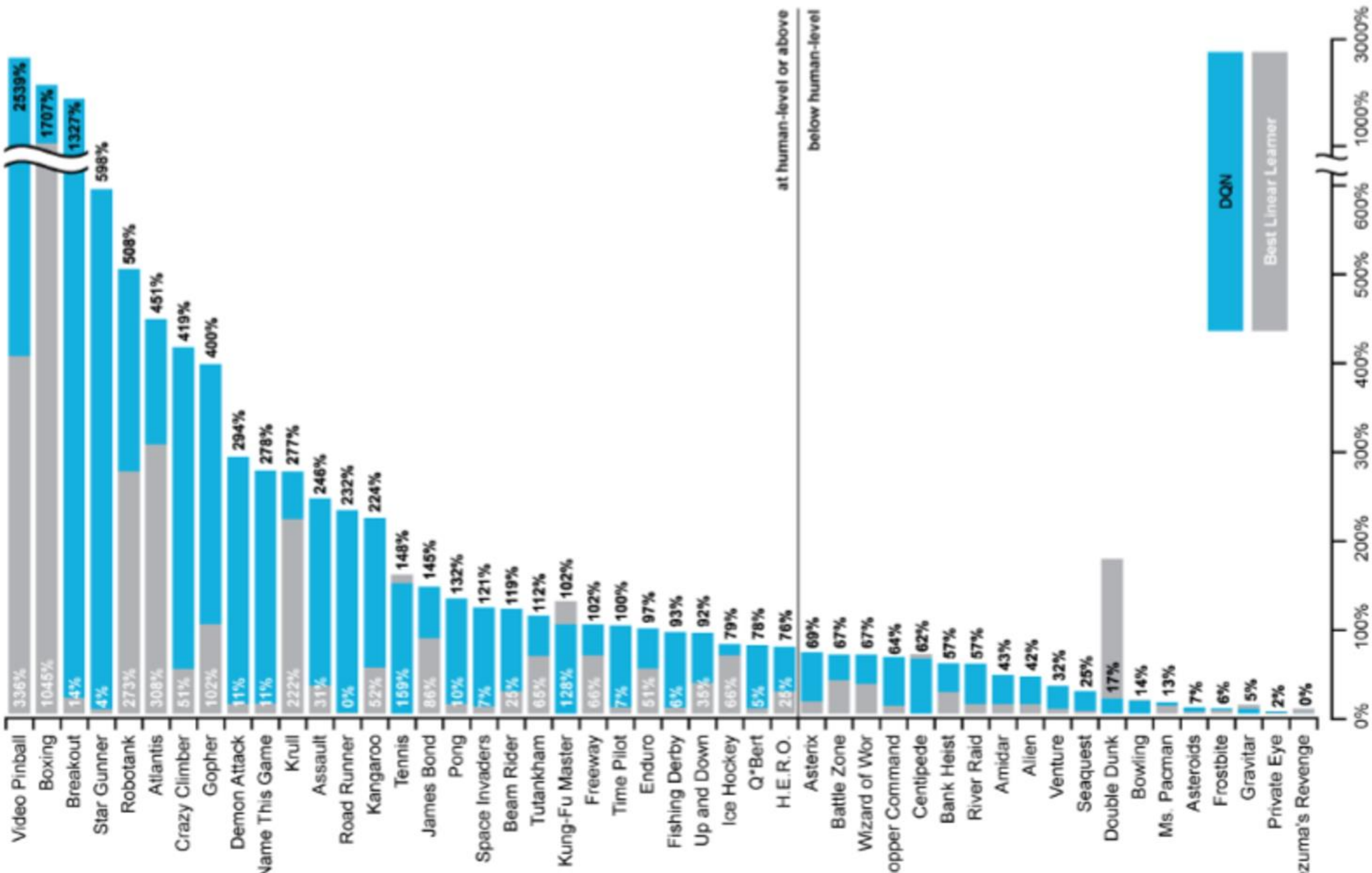
示例：针对Atari游戏的DQN

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



Network architecture and hyperparameters fixed across all games

DQN的实验结果（Atari）



2. 策略梯度算法

通过学习直接得到一个参数化的策略函数
假设动作是离散的，策略是随机的
一个策略表示

$$\pi(a|s,\boldsymbol{\theta}) \doteq \Pr\{A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$$

性能评价函数为 $\eta(\boldsymbol{\theta})$

策略梯度算法表示为迭代过程

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \widehat{\nabla \eta(\boldsymbol{\theta}_t)},$$

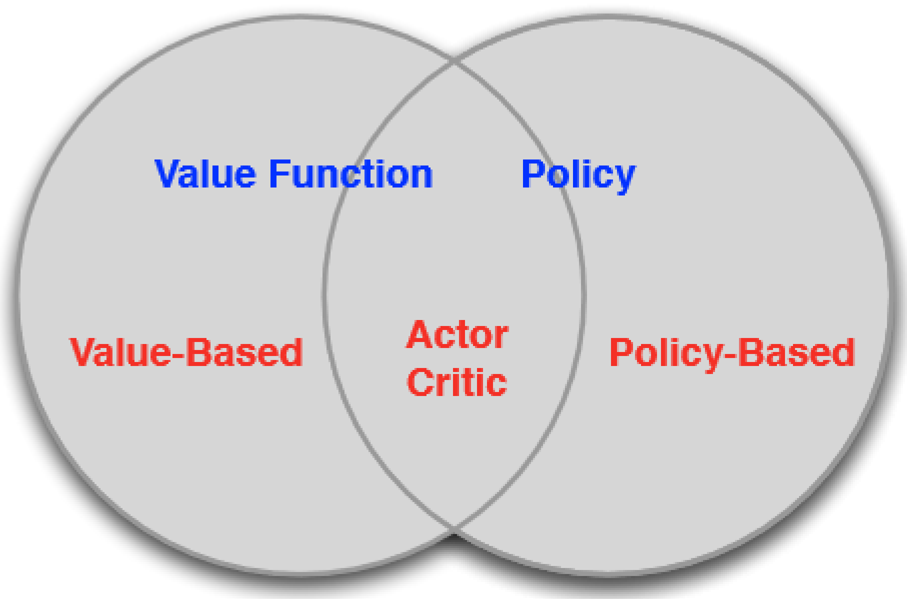
策略梯度算法
Policy Gradient

directly parametrise the policy

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

策略学习的三种基本方法

- 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



策略函数举例

常用离散策略函数用Softmax表示

$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{\exp(h(s,a,\boldsymbol{\theta}))}{\sum_b \exp(h(s,b,\boldsymbol{\theta}))}$$

这里

parameterized numerical preferences $h(s,a,\boldsymbol{\theta}) \in \mathbb{R}$

一个例子是线性函数逼近

$$h(s,a,\boldsymbol{\theta}) \doteq \boldsymbol{\theta}^\top \boldsymbol{\phi}(s,a),$$

feature vectors $\boldsymbol{\phi}(s,a) \in \mathbb{R}^n$

2.1 策略梯度方法的目标函数

- 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

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

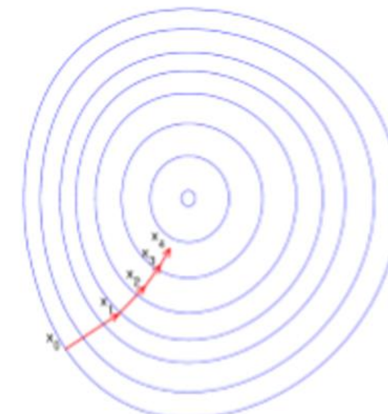
- Or the average reward per time-step

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

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

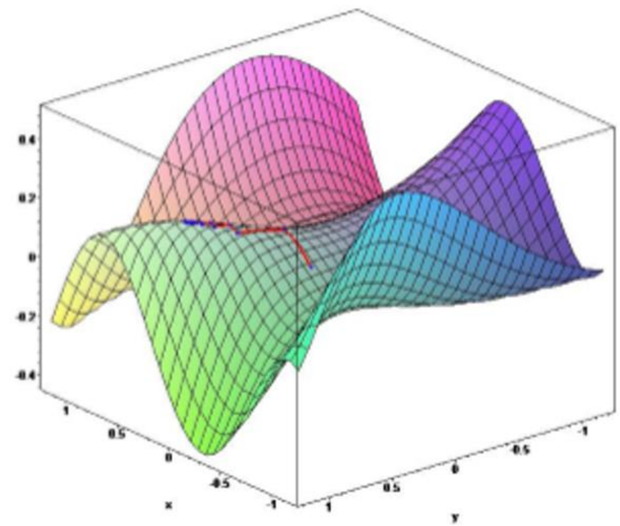
策略梯度方法的梯度算法

- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a *local* maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ



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

- Where $\nabla_\theta J(\theta)$ is the policy gradient



$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

- and α is a step-size parameter

2.2 策略梯度定理

预备知识：记分函数（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 $\nabla_\theta \log \pi_\theta(s, a)$

特例说明：梯度定理

一步MDPs作为说明

- 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

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

策略梯度定理

- 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

梯度定理

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

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

基于梯度定理的基本算法：Reinforce

$$\mathbb{E}_\pi[G_t|S_t,A_t] = q_\pi(S_t,A_t)$$

以G_t取代梯度定理的Q函数

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

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 $\mathbf{0}$)

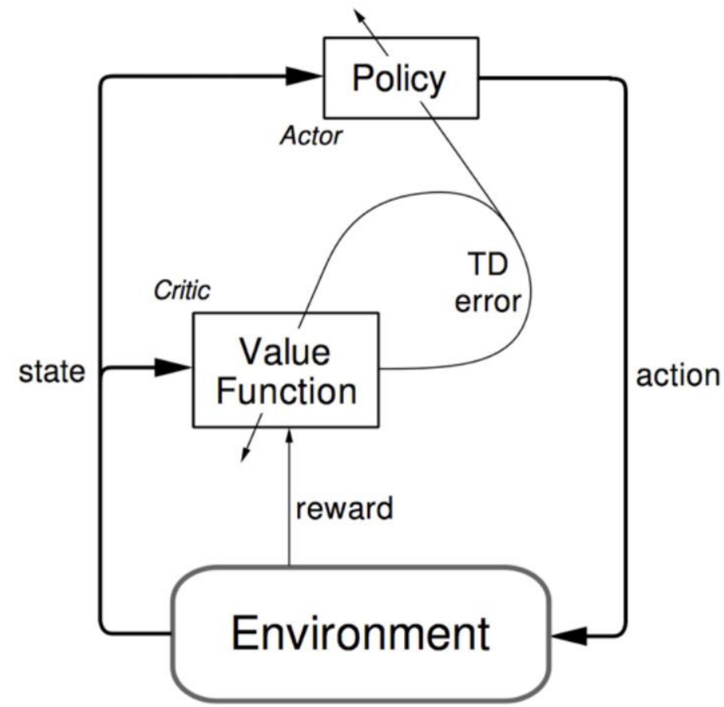
Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow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

2.3 Actor-Critic方法

- We use a critic to estimate the action-value function

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

- Actor-critic algorithms
  - Updates action-value function parameters
  - Updates policy parameters θ, in direction suggested by critic



Actor-Critic算法

加入一个Critic降低方差

- 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

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

动作-值函数Actor-Critic算法描述

- 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

3. DRL中连续动作空间的策略梯度算法进展

- 确定策略梯度算法（DPG，2014）
  - 确定策略（针对随机策略）、Actor-Critic、线性Q函数逼近，连续动作空间，确定策略函数
- 深度Q网络（Deep Q-Network，2015）
  - 用深度CNN网络逼近Q函数和策略函数，解决不收敛问题（replay algorithm）
  - 离散动作空间

连续动作策略梯度定理

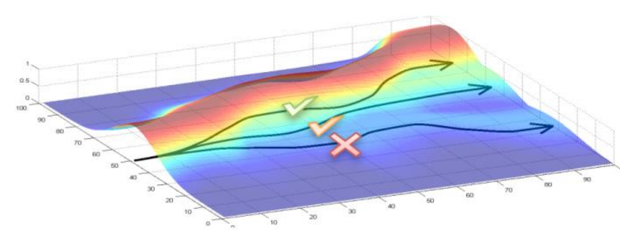
- Optimization Problem: Find θ that maximizes expected total reward.
  - The gradient of a stochastic policy $\pi_\theta(a|s)$ is given by

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

  - The gradient of a deterministic policy $a = \mu_\theta(s)$is given by

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s)\ \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\right]$$

- Gradient tries to
  - Increase probability of paths with positive R
  - Decrease probability of paths with negative R



连续动作空间的策略梯度算法（续）

- Deep DPG（DDPG，2016）
  - 深度CNN网络，连续动作空间
- Recurrent DPG（RDPG，2016）
  - 解决POMDP问题
  - RNN网络+DPG，LSTM更优先采用
- Fast-RDPG（2017/18）
  - 解决POMDP问题
  - On-line实现，收敛效率高，LSTM表示函数逼近（我们近期的一个工作）