参数化价值函数

本课程中解决的关键问题

- 之前所有模型的做法都是基于创建一个查询表，在表中维护状态值函数$V(s)$或状态-动作值函数$Q(s,a)$

- 当处理大规模马尔可夫决策过程（MDP）时，即：
  - 状态或者状态-动作空间非常大
  - 连续的状态或动作空间

  是否仍然需要为每一个状态维护$V(s)$或为每个状态-动作对维护$Q(s,a)$？
  - 例如
    - 围棋博弈（10^{170}的状态空间）
    - 直升机，自动驾驶汽车（连续的状态空间）

主要内容

- 大规模马尔可夫决策过程的解决方法
  - 对状态/动作进行离散化或分桶
  - 构建参数化的值函数估计

目录

Contents

ElitesAI

01

对状态/动作进行离散化

ElitesAI

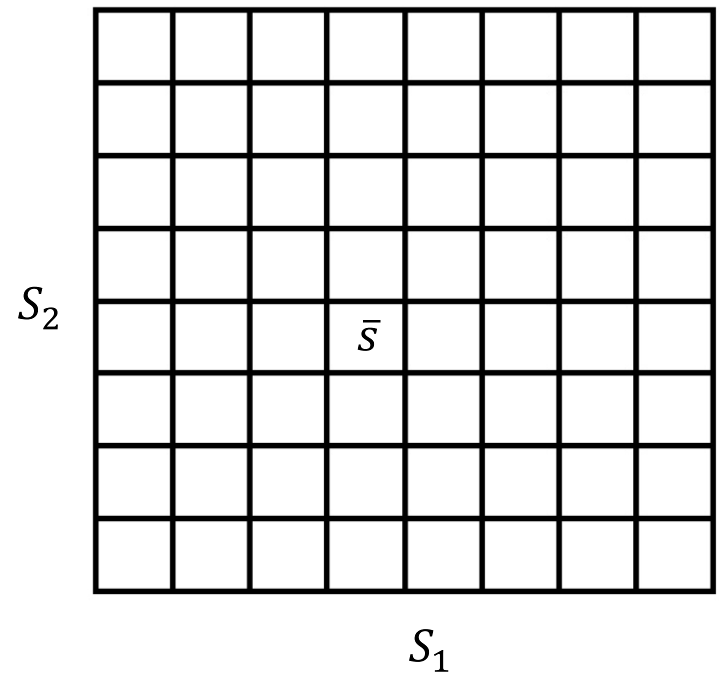
离散化连续马尔可夫决策过程

- 对于连续状态马尔可夫决策过程，我们可以对状态空间进行离散化

- 例如，如果用2维连续值 (s_1, s_2)表示状态，可以使用网格对状态空间进行切分从而转化为离散的状态值
- 记离散的状态值为$\bar{s}$
- 离散化的马尔可夫决策过程可以表示为：

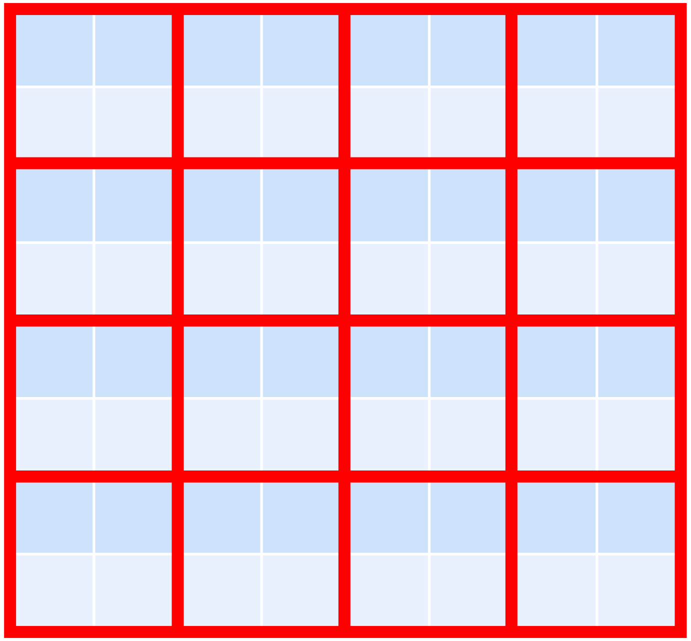
$$(\bar{S}, A, \{P_{\bar{s}a}\}, \gamma, R)$$

- 这样一来，就能够使用前述方法求解马尔可夫决策过程



对大型马尔可夫决策过程分桶

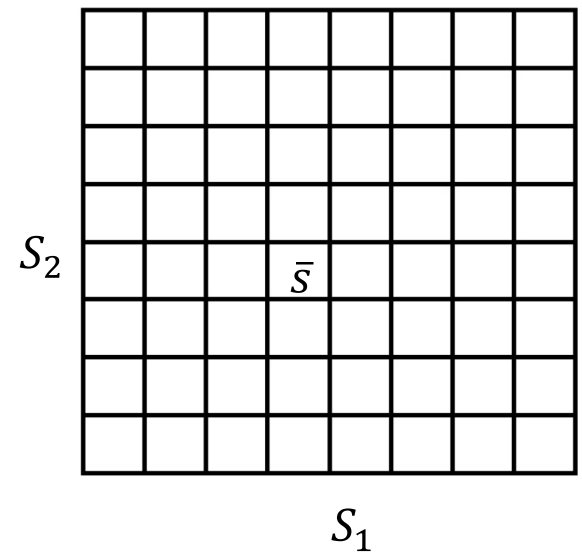
- 对于一个大型的离散状态马尔可夫决策过程，我们可以对状态值进一步分桶以进行采样聚合
  - 使用先验知识将相似的离散状态归类到一起
    - 例如，利用根据先验知识抽取出来的状态特征对状态进行聚类

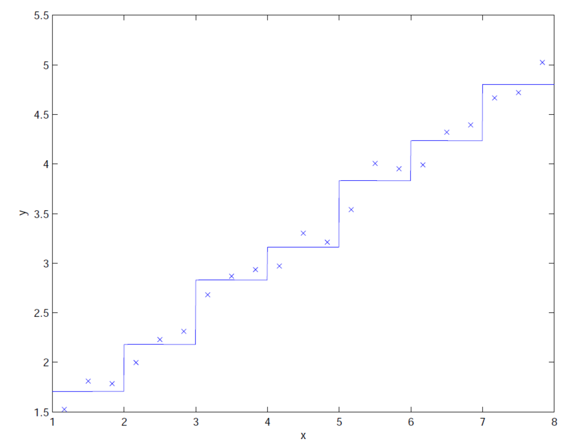


离散化/分桶

- 优点
  - 操作简洁直观
  - 高效
  - 在处理许多问题时能够达到较好效果
- 缺点
  - 过于简单地表示价值函数V
  - 可能为每个离散区间假设一个常数值
  - 维度灾难

$$S = R^n \Rightarrow \bar{S} = \{1, \ldots, k\}^n$$





02

参数化
价值函数

ElitesAI

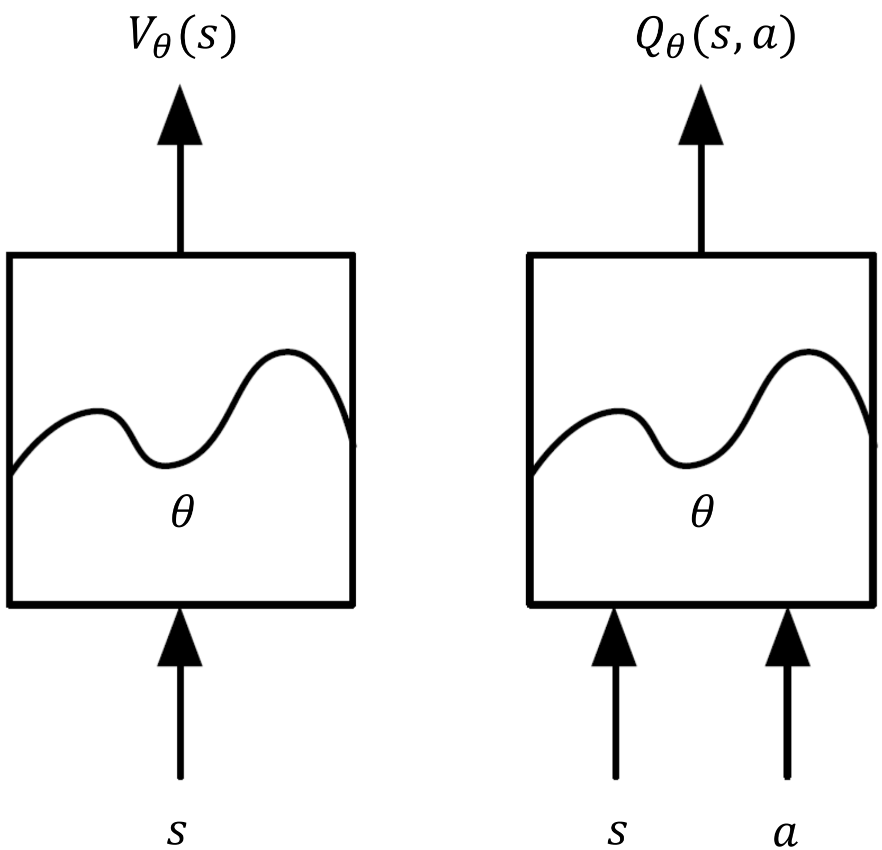
参数化值函数近似

- 构建参数化（可学习的）函数来近似值函数

$$V_\theta(s) \simeq V^\pi(s)$$
$$Q_\theta(s,a) \simeq Q^\pi(s,a)$$

  - θ是近似函数的参数，可以通过强化学习进行更新
  - 参数化的方法将现有可见的状态泛化到没有见过的状态上

值函数近似的主要形式



- 一些函数近似
  - （一般的）线性模型
  - 神经网络
  - 决策树
  - 最近邻
  - 傅立叶/小波基底
- 可微函数
  - （一般的）线性模型
  - 神经网络
- 我们希望模型适合在非稳态的，非独立同分布的数据上训练
  - 因此参数化模型比树模型更适合

基于随机梯度下降（SGD）的值函数近似

- 目标：找到参数向量θ最小化值函数近似值与真实值之间的均方误差

$$J(\theta) = \mathbb{E}_{\pi}[\frac{1}{2}\left(V^{\pi}(s) - V_{\theta}(s)\right)^{2}]$$

- 误差减小的梯度方向

$$-\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_{\pi}[\left(V^{\pi}(s) - V_{\theta}(s)\right)\frac{\partial V_{\theta}(s)}{\partial \theta}]$$

- 单次采样进行随机梯度下降

$$\begin{aligned} \theta &\leftarrow \theta - \alpha \frac{\partial J(\theta)}{\partial \theta} \\ &= \theta + \alpha\left(V^{\pi}(s) - V_{\theta}(s)\right)\frac{\partial V_{\theta}(s)}{\partial \theta} \end{aligned}$$

特征化状态

- 用一个特征向量表示状态

$$x(s) = \begin{bmatrix} x_1(s) \\ \vdots \\ x_k(s) \end{bmatrix}$$

- 以直升机控制问题为例
  - 3D位置
  - 3D速度（位置的变化量）
  - 3D加速度（速度的变化量）

价值函数近似算法

01

状态值函数近似

ElitesAI

线性状态值函数近似

- 用特征的线性组合表示价值函数

$$V_\theta(s) = \theta^{\mathrm{T}} x(s)$$

- 目标函数是参数θ的二次函数

$$J(\theta) = \mathbb{E}_\pi\left[\frac{1}{2}\left(V^\pi(s) - \theta^{\mathrm{T}} x(s)\right)^2\right]$$

- 因而随机梯度下降能够收敛到全局最优解上

$$\begin{aligned} \theta &\leftarrow \theta - \alpha \frac{\partial J(\theta)}{\partial \theta} \\ &= \theta + \underbrace{\alpha}_{\text{步长}} \underbrace{\left(V^\pi(s) - V_\theta(s)\right)}_{\text{预测误差}} \underbrace{x(s)}_{\text{特征值}} \end{aligned}$$

蒙特卡洛状态值函数近似

$$\theta \leftarrow \theta + \alpha\left(V^{\pi}(s) - V_{\theta}(s)\right)x(s)$$

- 我们用$V^{\pi}(s)$表示真实的目标价值函数
- 在“训练数据”上运用监督学习对价值函数进行预测

$$\langle s_1, G_1\rangle, \langle s_2, G_2\rangle, \dots, \langle s_T, G_T\rangle$$

- 对于每个数据样本$\langle s_t, G_t\rangle$

$$\theta \leftarrow \theta + \alpha\left(G_t - V_{\theta}(s)\right)x(s_t)$$

- 蒙特卡洛预测至少能收敛到一个局部最优解
    - 在价值函数为线性的情况下可以收敛到全局最优

时序差分状态值函数近似

$$\theta \leftarrow \theta + \alpha\big(V^{\pi}(s) - V_\theta(s)\big)x(s)$$

- 时序差分算法的目标$r_{t+1} + \gamma V_\theta(s_{t+1})$是真实目标价值$V_\pi(s_t)$的有偏采样

- 在“训练数据”上运用监督学习

$$\langle s_1, r_2 + \gamma V_\theta(s_2)\rangle, \langle s_2, r_3 + \gamma V_\theta(s_3)\rangle, \dots, \langle s_T, r_T\rangle$$

- 对于每个数据样本$\langle s_t, r_{t+1} + \gamma V_\theta(s_{t+1})\rangle$

$$\theta \leftarrow \theta + \alpha\big(r_{t+1} + \gamma V_\theta(s_{t+1}) - V_\theta(s)\big)x(s_t)$$

- 线性情况下时序差分学习（接近）收敛到全局最优解

02

状态-动作值函数近似

ElitesAI

状态-动作值函数近似

- 对动作-状态值函数进行近似

$$Q_\theta(s,a) \simeq Q^\pi(s,a)$$

- 最小均方误差

$$J(\theta) = \mathbb{E}_\pi\left[\frac{1}{2}\left(Q^\pi(s,a) - Q_\theta(s,a)\right)^2\right]$$

- 在单个样本上进行随机梯度下降

$$\begin{aligned}\theta &\leftarrow \theta - \alpha\frac{\partial J(\theta)}{\partial\theta} \\ &= \theta + \alpha\big(Q^\pi(s,a) - Q_\theta(s,a)\big)\frac{\partial Q_\theta(s,a)}{\partial\theta}\end{aligned}$$

线性状态-动作值函数近似

- 用特征向量表示状态-动作对

$$x(s,a)=\begin{bmatrix}x_1(s,a)\\ \vdots \\ x_k(s,a)\end{bmatrix}$$

- 线性情况下，参数化后Q函数

$$Q_\theta(s,a)=\theta^{\mathrm{T}}x(s,a)$$

- 利用随机梯度下降更新

$$\begin{aligned}\theta &\leftarrow \theta-\alpha\frac{\partial J(\theta)}{\partial\theta}\\ &=\theta+\alpha\left(Q^{\pi}(s,a)-\theta^{\mathrm{T}}x(s,a)\right)x(s,a)\end{aligned}$$

时序差分状态-动作值函数近似

$$\theta \leftarrow \theta + \alpha\left(Q^{\pi}(s,a) - Q_{\theta}(s,a)\right)\frac{\partial Q_{\theta}(s,a)}{\partial \theta}$$

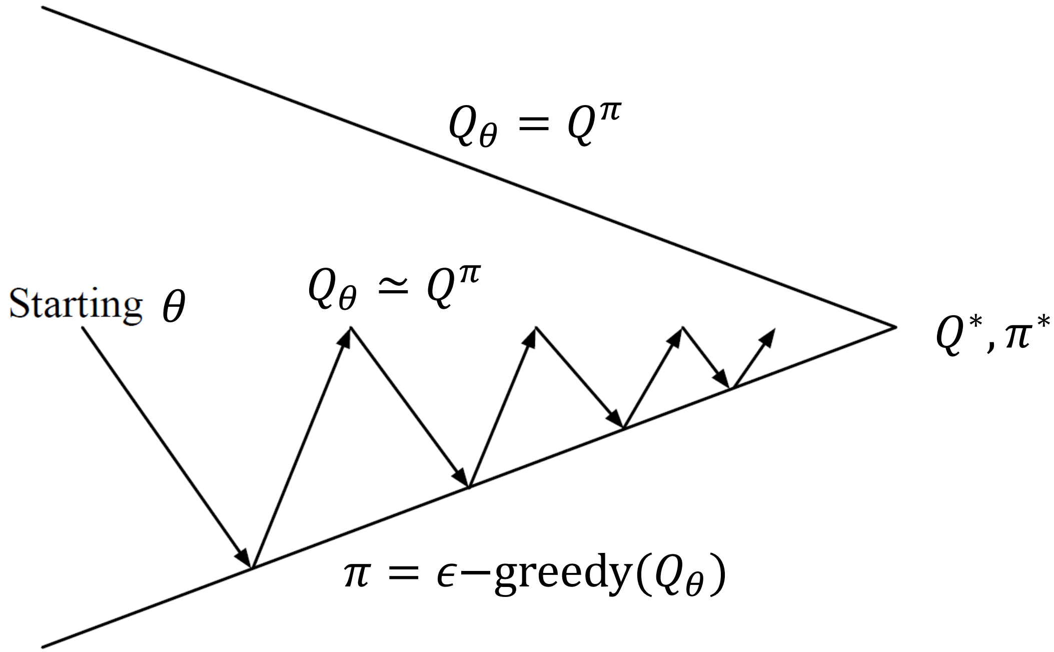
- 对于蒙特卡洛学习，目标是累计奖励G_t

$$\theta \leftarrow \theta + \alpha\left(G_t - Q_{\theta}(s,a)\right)\frac{\partial Q_{\theta}(s,a)}{\partial \theta}$$

- 对于时序差分学习，目标是$r_{t+1} + \gamma Q_{\theta}(s_{t+1}, a_{t+1})$

$$\theta \leftarrow \theta + \alpha\left(r_{t+1} + \gamma Q_{\theta}(s_{t+1}, a_{t+1}) - Q_{\theta}(s,a)\right)\frac{\partial Q_{\theta}(s,a)}{\partial \theta}$$

时序差分状态-动作值函数近似



- 策略评估：近似策略评估$Q_\theta \simeq Q^\pi$
- 策略改进：ϵ-贪心策略提升

时序差分学习参数更新过程

- 对于$TD(0)$，时序差分学习的目标是
  - 状态值函数

$$\theta \leftarrow \theta + \alpha\big(V^{\pi}(s_t) - V_\theta(s)\big)\frac{\partial V_\theta(s_t)}{\partial \theta}$$

$$= \theta + \alpha\big(r_{t+1} + \gamma V_\theta(s_{t+1}) - V_\theta(s)\big)\frac{\partial V_\theta(s_t)}{\partial \theta}$$

  - 动作-状态值函数

$$\theta \leftarrow \theta + \alpha\big(Q^{\pi}(s,a) - Q_\theta(s,a)\big)\frac{\partial Q_\theta(s,a)}{\partial \theta}$$

$$= \theta + \alpha\big(r_{t+1} + \gamma Q_\theta(s_{t+1},a_{t+1}) - Q_\theta(s,a)\big)\frac{\partial Q_\theta(s,a)}{\partial \theta}$$

- 虽然θ在时序差分学习的目标中出现，但是我们并不需要计算目标函数的梯度。想想这是为什么？

01

深度Q网络（DQN）

ElitesAI

Q 学习回顾

- 不直接更新策略
- 基于值的方法
- Q 学习算法学习一个由 θ 作为参数的函数 $Q_\theta(s,a)$
  - Target值 $y_\mathrm{t} = r_t + \gamma \max_{a'} Q_\theta(s_{t+1}, a')$
  - 更新方程 $Q_\theta(s_t, a_t) \leftarrow Q_\theta(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q_\theta(s_{t+1}, a') - Q_\theta(s_t, a_t))$
  - 优化目标

$$\theta^\star \leftarrow \arg\min_\theta \frac{1}{2} \sum_{(s_t,a_t)\in D} (Q_\theta(s_t, a_t) - (r + \gamma \max_{a'} \underbrace{Q_\theta(s_{t+1}, a')}_{\text{此处无梯度}}))^2$$

深度Q网络（DQN）

直观想法

- 使用神经网络来逼近上述 $Q_\theta(s,a)$
  - 算法不稳定
    - 连续采样得到的 $\{(s_t,a_t,s_{t+1},r_t)\}$ 不满足独立分布。
    - $\{(s_t,a_t,s_{t+1},r_t)\}$ 为状态-动作-下一状态-回报输入。
    - $Q_\theta(s,a)$ 的频繁更新。

解决办法

- 经验回放
- 使用双网络结构：评估网络（evaluation network）和目标网络（target network）

"Human-Level Control Through Deep Reinforcement Learning"，Mnih, Kavukcuoglu, Silver et al. (2015)

经验回放

经验回放

- 存储训练过程中的每一步 $e_t = (s_t, a_t, s_{t+1}, r_t)$ 于数据库 D 中，采样时服从均匀分布。

优先经验回放

衡量标准

- 以 Q 函数的值与 Target 值的差异来衡量学习的价值，即

$$p_t = |r_t + \gamma \max_{a'} Q_\theta(s_{t+1}, a') - Q_\theta(s_t, a_t)|$$

- 为了使各样本都有机会被采样，存储 $e_t = (s_t, a_t, s_{t+1}, r_t, p_t + \epsilon)$。

选中的概率

- 样本 e_t 被选中的概率为 $P(t) = \frac{p_t^\alpha}{\sum_k p_k^\alpha}$。

重要性采样（Importance Sampling）

- 权重为 $\omega_t = \frac{(N \times P(t))^{-\beta}}{\max_i \omega_i}$

"Prioritized Experience Replay"，Schaul et al. (2016)

经验回放

Algorithm 1 Deep Q-learning with Experience Replay

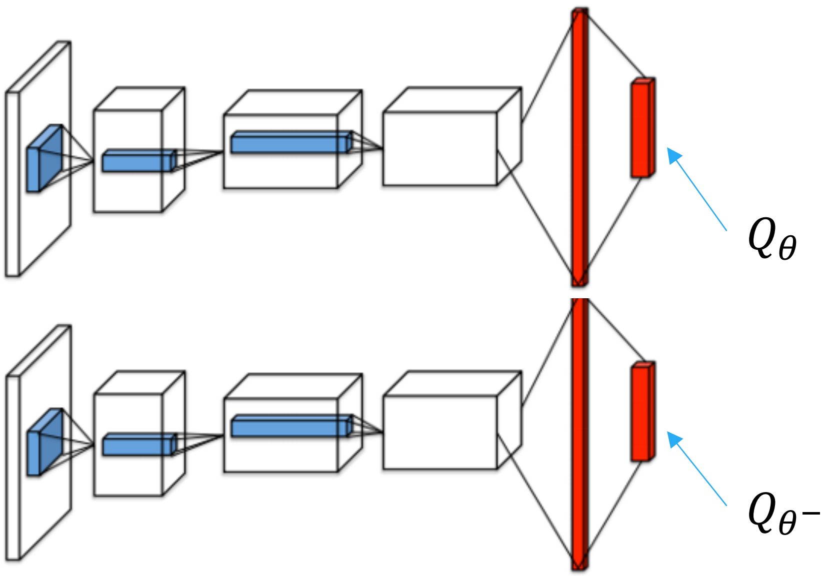
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

"Prioritized Experience Replay", Schaul et al. (2016)

目标网络

- 目标网络 $Q_{\theta^-}(s,a)$
  - 使用较旧的参数，记为 θ^-，每隔 C 步和训练网络的参数同步一次。
  - 第i次迭代的损失函数为

$$L_i(\theta_i) = \mathbb{E}_{s_t,a_t,s_{t+1},r_t,p_t\sim D}[\frac{1}{2}\omega_t(\underbrace{r_t + \gamma \max_{a'} Q_{\theta_i^-}(s_{t+1},a')}_{target} - Q_{\theta_i}(s_t,a_t))^2\ ]$$



“Human-Level Control Through Deep Reinforcement Learning”，Mnih, Kavukcuoglu, Silver et al. (2015)

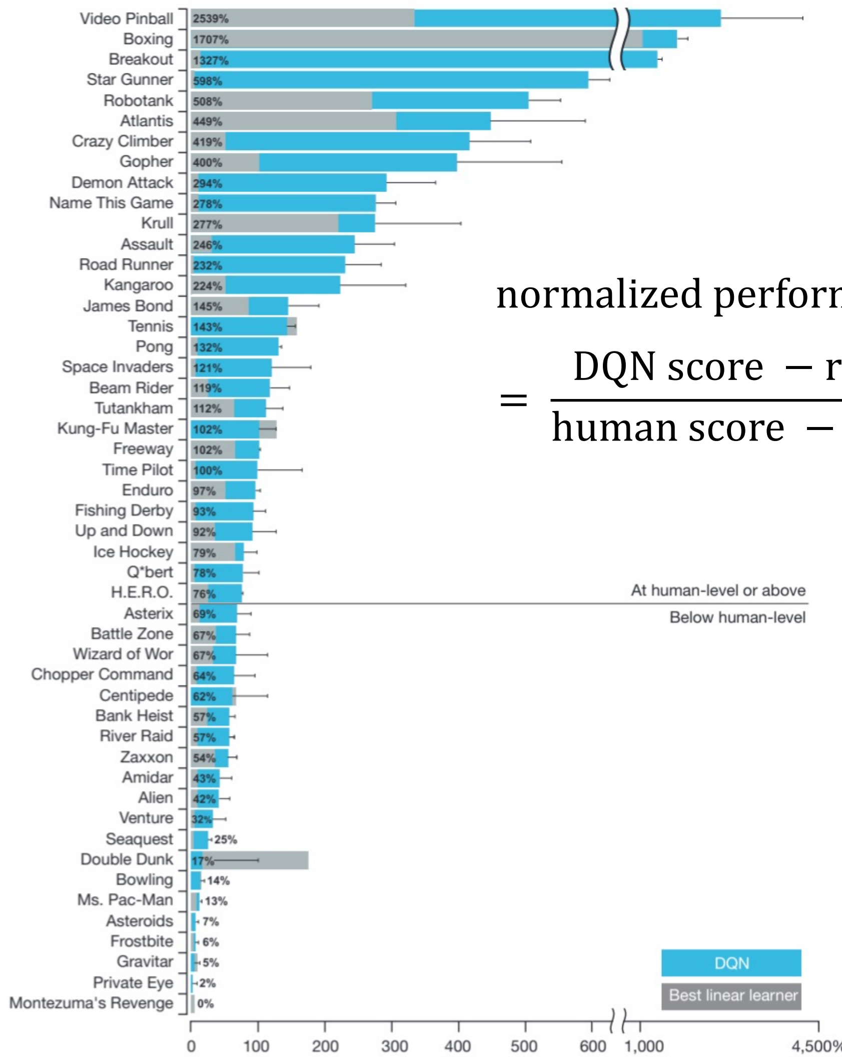
算法流程

1. 收集数据：使用 ϵ−greedy 策略进行探索，将得到的状态动作组 (s_t, a_t, s_{t+1}, r_t) 放入经验池（replay-buffer）

2. 采样：从数据库中采样 k 个动作状态组

3. 更新网络
   - 用采样得到的数据计算 $Loss$。
   - 更新 Q 函数网络 θ。
   - 每 C 次迭代（更新Q函数网络）更新一次目标网络 θ^-。

在 Atari 环境中的实验结果



normalized performance

$$= \frac{\text{DQN score} - \text{random play score}}{\text{human score} - \text{random play score}}$$

The performance of DQN is normalized with respect to a professional human games tester (that is, 100% level)

“Human-Level Control Through Deep Reinforcement Learning”，Mnih, Kavukcuoglu, Silver et al. (2015)

02

Double DQN

ElitesAI

Q-learning中的过估计

Q函数的过高估计

- Target值 $\mathrm{y_t} = r_t + \gamma \max_{a'} Q_\theta(s_{t+1}, a')$

max 操作使得 Q 函数的值越来越大，甚至高于真实值

过高估计的原因

- 假设有随机变量 X_1, X_2，有 $\mathbb{E}[\max(X_1, X_2)] \geq \max(\mathbb{E}[X_1], \mathbb{E}[X_2])$

$$\begin{aligned}\max_{a' \in A} Q_{\theta'}(s_{t+1}, a') &= Q_{\theta'}(s_{t+1}, \arg\max_{a'} Q_{\theta'}(s_{t+1}, a')) \\ &= \mathbb{E}[R | s_{t+1}, \arg\max_{a'} Q_{\theta'}(s_{t+1}, a'), \theta'] \\ &\geq \max(\mathbb{E}[R | s_{t+1}, a_1, \theta'], \mathbb{E}[R | s_{t+1}, a_2, \theta'], \cdots), a_i \in A\end{aligned}$$

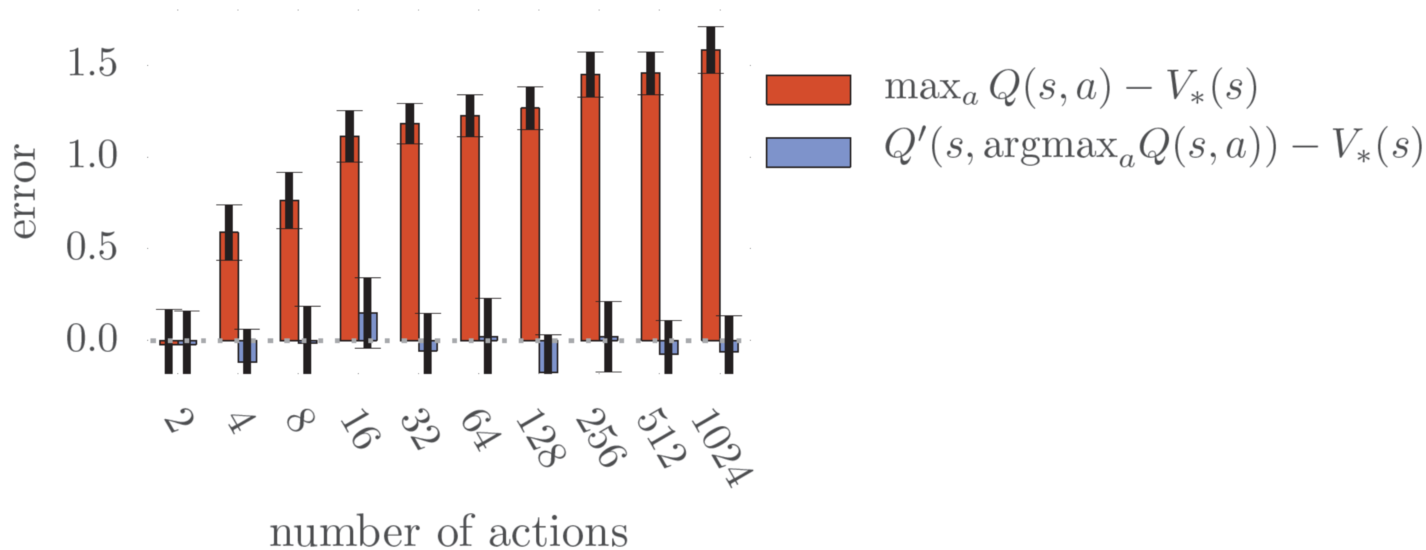
Q 函数的值被视作在状态 s'，动作 a' 下的回报期望值

我们真正想要得到的最大价值

“Double Reinforcement Learning with Double Q-Learning”，van Hasselt et al. (2016)
“Double Q-learning”，van Hasselt.(2010)

Q-learning中的过估计

- Q函数的过高估计程度随着候选行动数量增大变得更严重



- 其中$Q_t(s,a) - V_*(s)$设为在[-1,1]区间均匀分布
- Q'函数是另一组独立训练的价值函数

"Double Reinforcement Learning with Double Q-Learning", van Hasselt et al. (2016)
"Double Q-learning", van Hasselt.(2010)

Double DQN

- 使用不同的网络来估值和决策

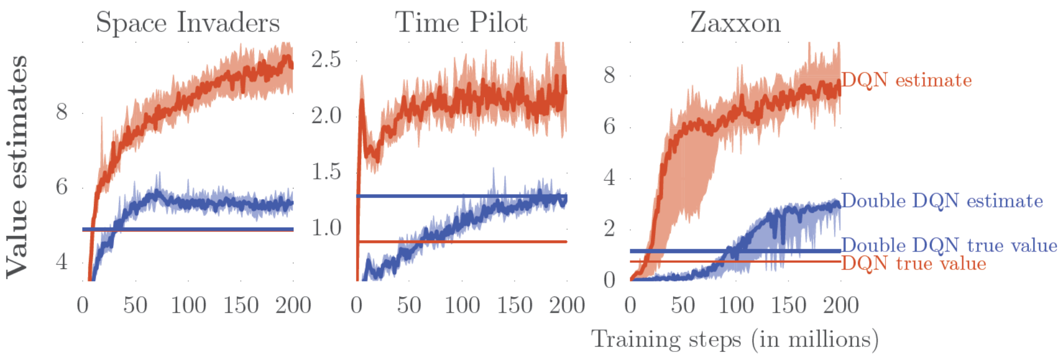
DQN $y_t = r_t + \gamma Q_\theta(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a'))$

Double DQN $y_t = r_t + \gamma \boxed{Q_{\theta'}}(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a'))$

"Double Reinforcement Learning with Double Q-Learning", van Hasselt et al. (2016)

在 Atari 环境中的实验结果

价值估计误差

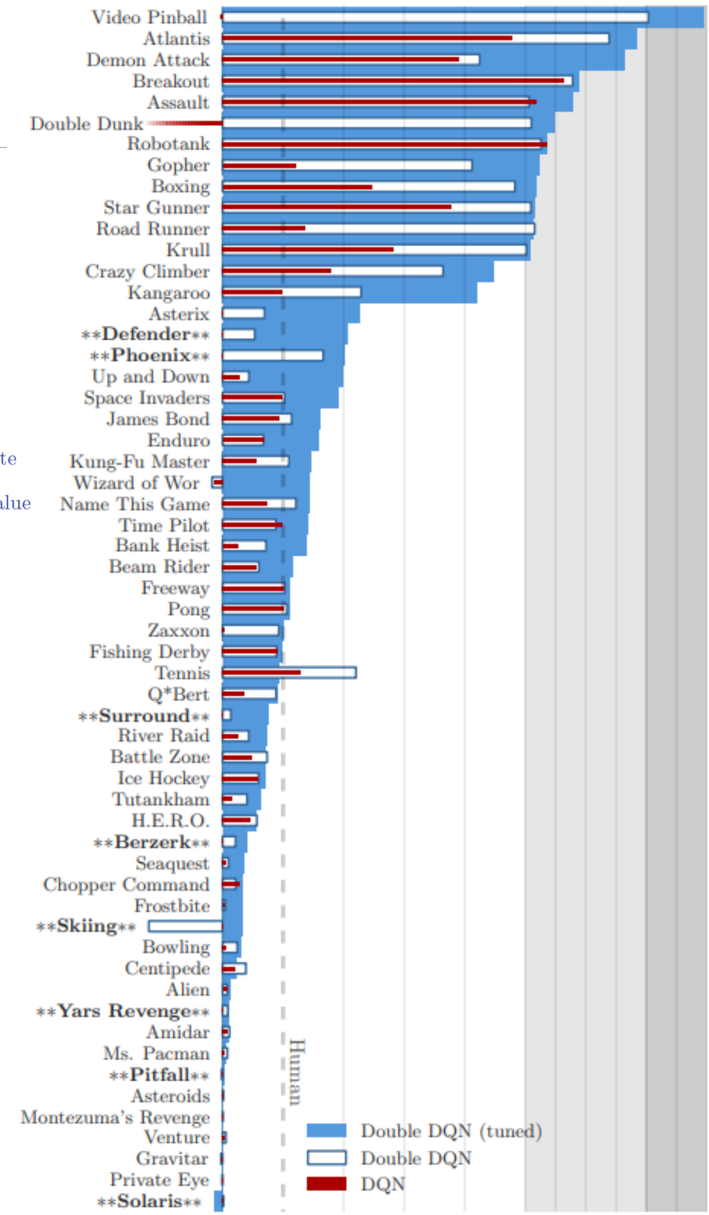


Atari Performance

	no ops		human starts		
	DQN	DDQN	DQN	DDQN	DDQN (tuned)
Median	93%	**115%**	47%	88%	**117%**
Mean	241%	**330%**	122%	273%	**475%**

normalized performance

$$= \frac{\text{DQN score } - \text{random play score}}{\text{human score } - \text{random play score}}$$



04

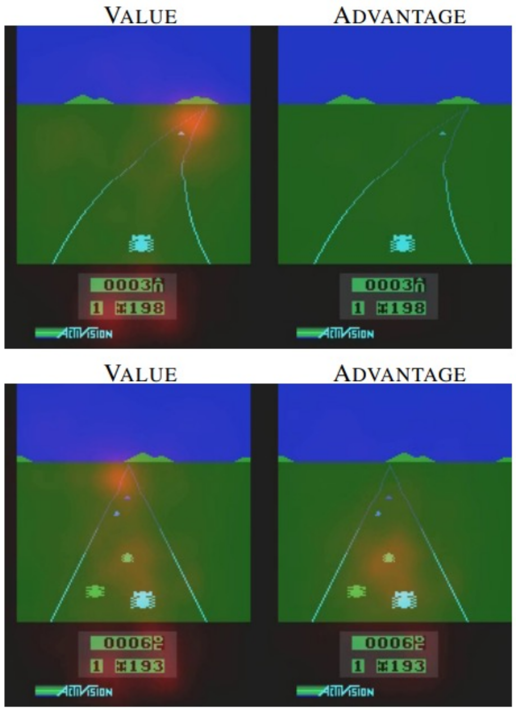
Dueling DQN

ElitesAI

优点

- 处理与动作关联较小的状态
- 状态值函数的学习较为有效：一个状态值函数对应多个Advantage函数

显著区域
(saliency maps)



可以在不考虑动作的影响下判断出该状态的好坏。

可以强调动作的重要性：advantage学会只在agent面前有车的时候会加强注意力

“Dueling Network Architectures for Deep Reinforcement Learning”，Wang et al. (2016)

Dueling DQN

假设动作值函数服从某个分布：

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

显然：$V(s) = \mathbb{E}[Q(s,a)] = \mu$

同样有：$Q(s,a) = \mu + \boxed{\varepsilon(s,a)}$

偏移量

问题

如何描述 $\varepsilon(s,a)$? → $\varepsilon(s,a) = Q(s,a) - V(s)$ → 也称为Advantage函数

“Dueling Network Architectures for Deep Reinforcement Learning”，Wang et al. (2016)

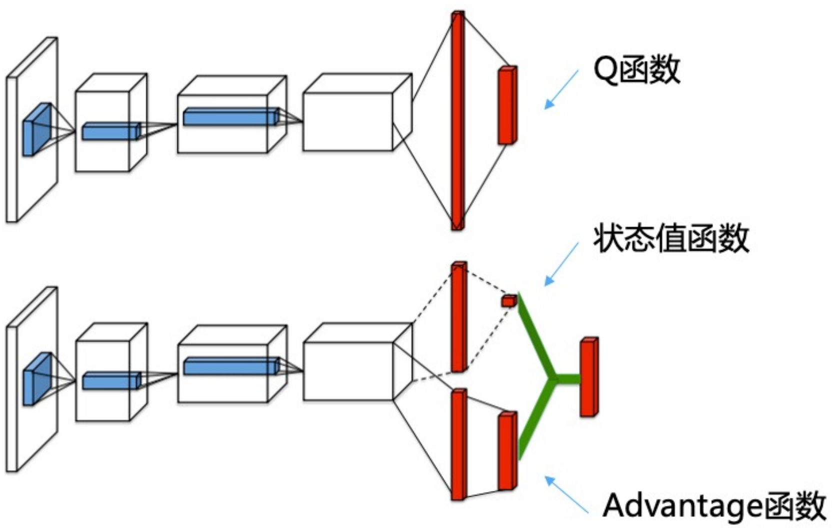
Dueling DQN

Advantage 函数

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

$$Q^{\pi}(s,a) = \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$$

$$V^{\pi}(s) = \mathbb{E}_{a \sim \pi(s)}[Q^{\pi}(s,a)]$$



$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha),$$

"Dueling Network Architectures for Deep Reinforcement Learning", Wang et al. (2016)

Dueling DQN

Advantage 函数

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

$$Q^{\pi}(s,a) = \mathbb{E}[R_t|s_t = s, a_t = a, \pi]$$

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi(s)}[Q^{\pi}(s,a)]$$

$$\mathbb{E}_{a\sim\pi(s)}\left[A^{\pi}(s,a)\right] = \ ?$$

$$A(s,a^*) = \ ?$$

Dueling DQN

Advantage 函数

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

$$Q^{\pi}(s,a) = \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$$

$$V^{\pi}(s) = \mathbb{E}_{a \sim \pi(s)}[Q^{\pi}(s,a)]$$

$$\mathbb{E}_{a \sim \pi(s)}\left[A^{\pi}(s,a)\right] = 0$$

$$A\left(s, a^{*}\right) = 0$$

"Dueling Network Architectures for Deep Reinforcement Learning" , Wang et al. (2016)

Dueling DQN

Advantage 函数

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

$$Q^{\pi}(s,a) = \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$$

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi(s)}[Q^{\pi}(s,a)]$$

不同的Advantage聚合形式

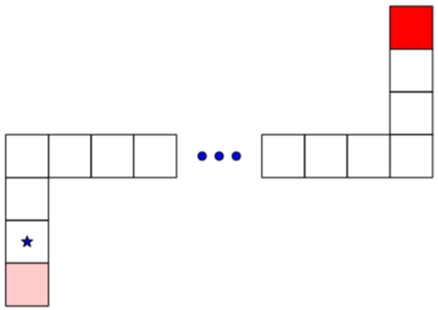
$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \boxed{\left(A(s,a;\theta,\alpha) - \max_{a'\in|A|} A(s,a';\theta,\alpha)\right)}$$

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \boxed{\left(A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)\right)}$$

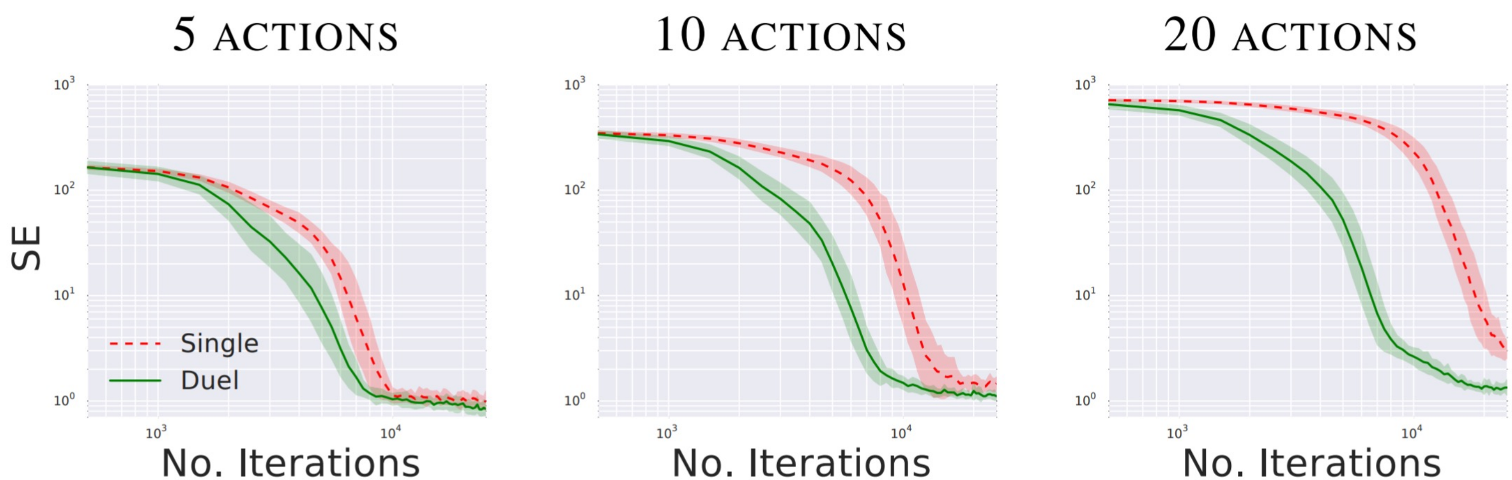
探索任务：走廊环境

- 走廊环境
  - *为起点状态
  - 行动：上、下、左、右、不动
  - 左下角有小的正向奖励
  - 右上角有大的正向奖励
- Q函数评估均方误差

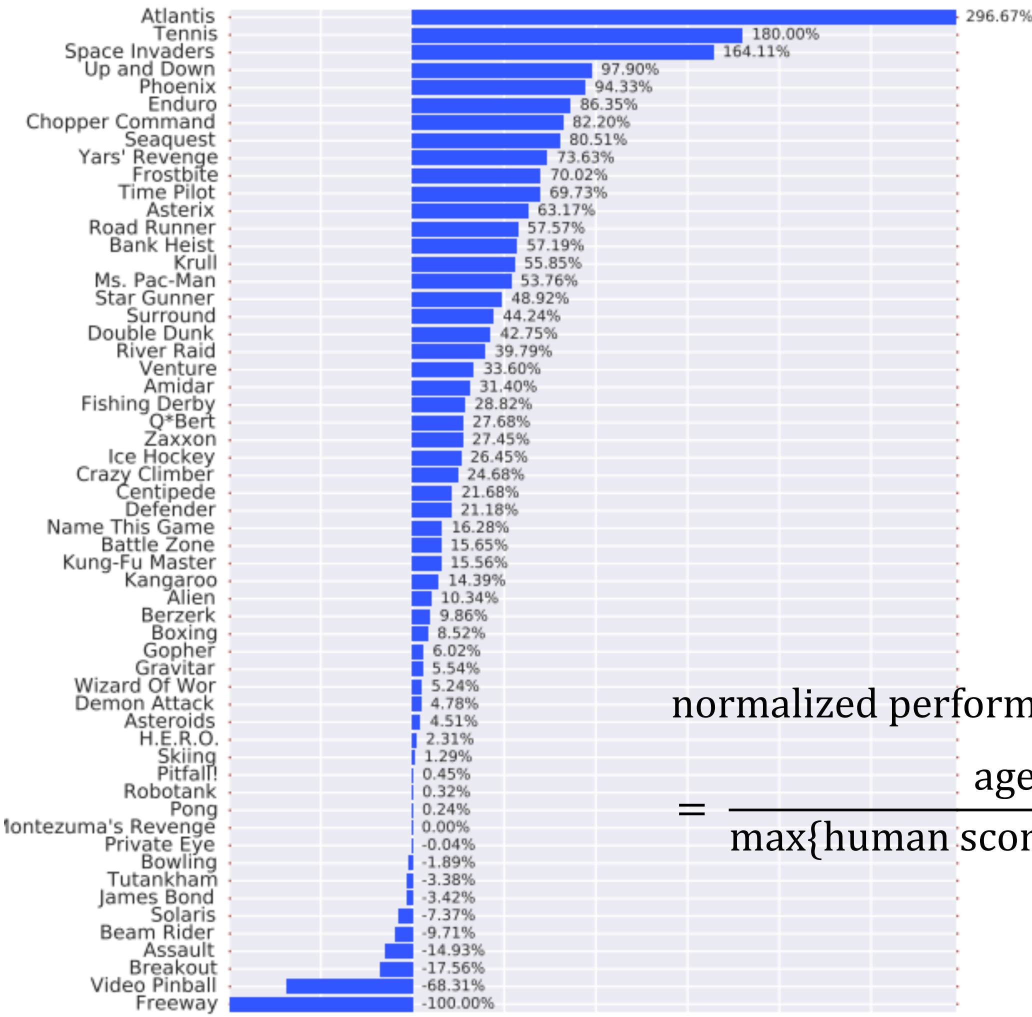
CORRIDOR ENVIRONMENT



更多的行动是‘不动’



在 Atari 环境中的实验结果I

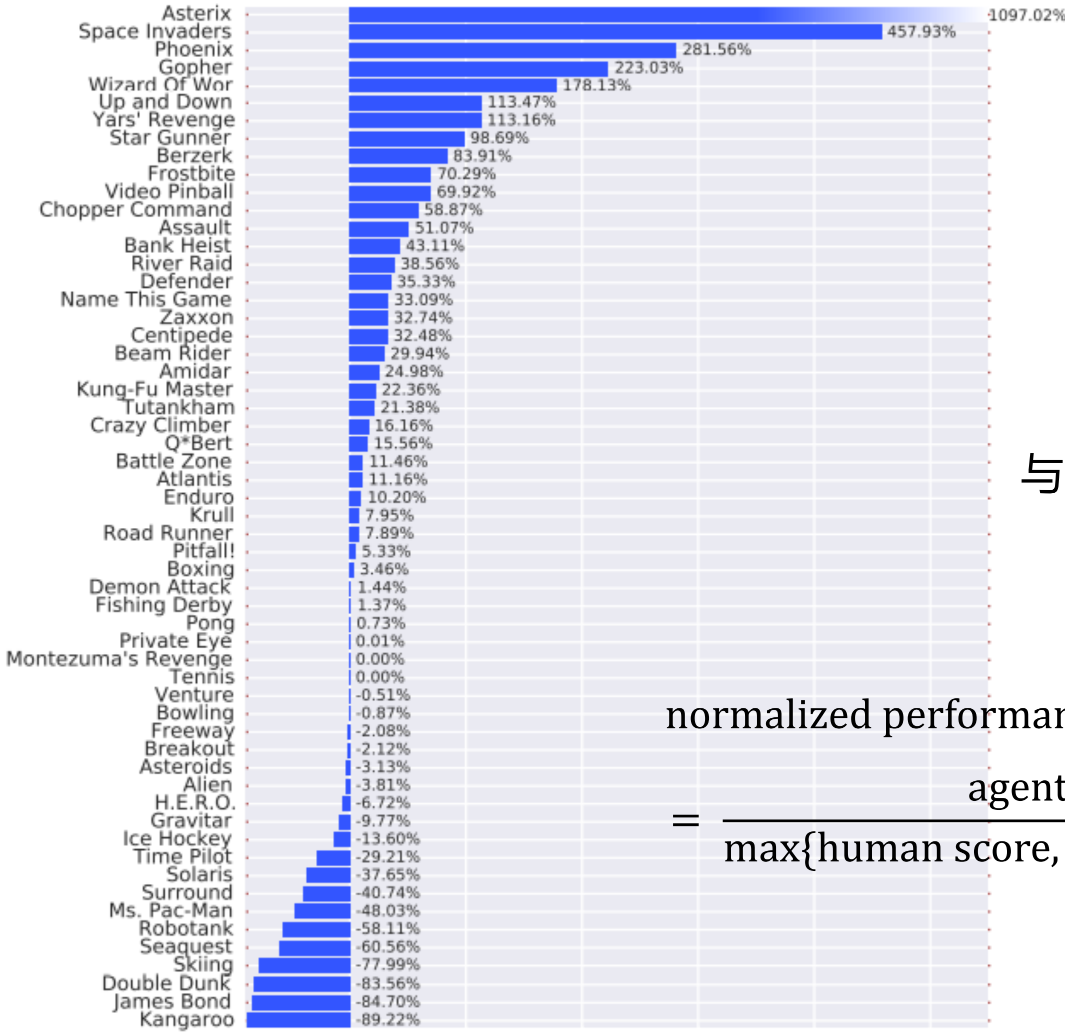


与DQN对比

$$\text{normalized performance} = \frac{\text{agent score} - \text{baseline score}}{\max\{\text{human score}, \text{baseline score}\} - \text{random play score}}$$

"Double Reinforcement Learning with Double Q-Learning" , van Hasselt et al. (2016)

在 Atari 环境中的实验结果II



与Prioritized Double DQN对比

$$\text{normalized performance} = \frac{\text{agent score} - \text{baseline score}}{\max\{\text{human score}, \text{baseline score}\} - \text{random play score}}$$

"Double Reinforcement Learning with Double Q-Learning", van Hasselt et al. (2016)

复习之前的深度强化学习算法

- DQN：一次输入多个行动Q值输出、目标网络、随机采样经验

$$y_t = r_t + \gamma Q_{\theta'}(s_{t+1}, \arg\max_{a'} Q_{\theta'}(s_{t+1}, a'))$$

"Human-Level Control Through Deep Reinforcement Learning", Mnih, Kavukcuoglu, Silver et al. Nature 2015.

- Double DQN：解耦合行动选择和价值估计、解决DQN过高估计问题

$$y_t = r_t + \gamma Q_{\theta'}(s_{t+1}, \arg\max_{a'} Q_{\theta}(s_{t+1}, a'))$$

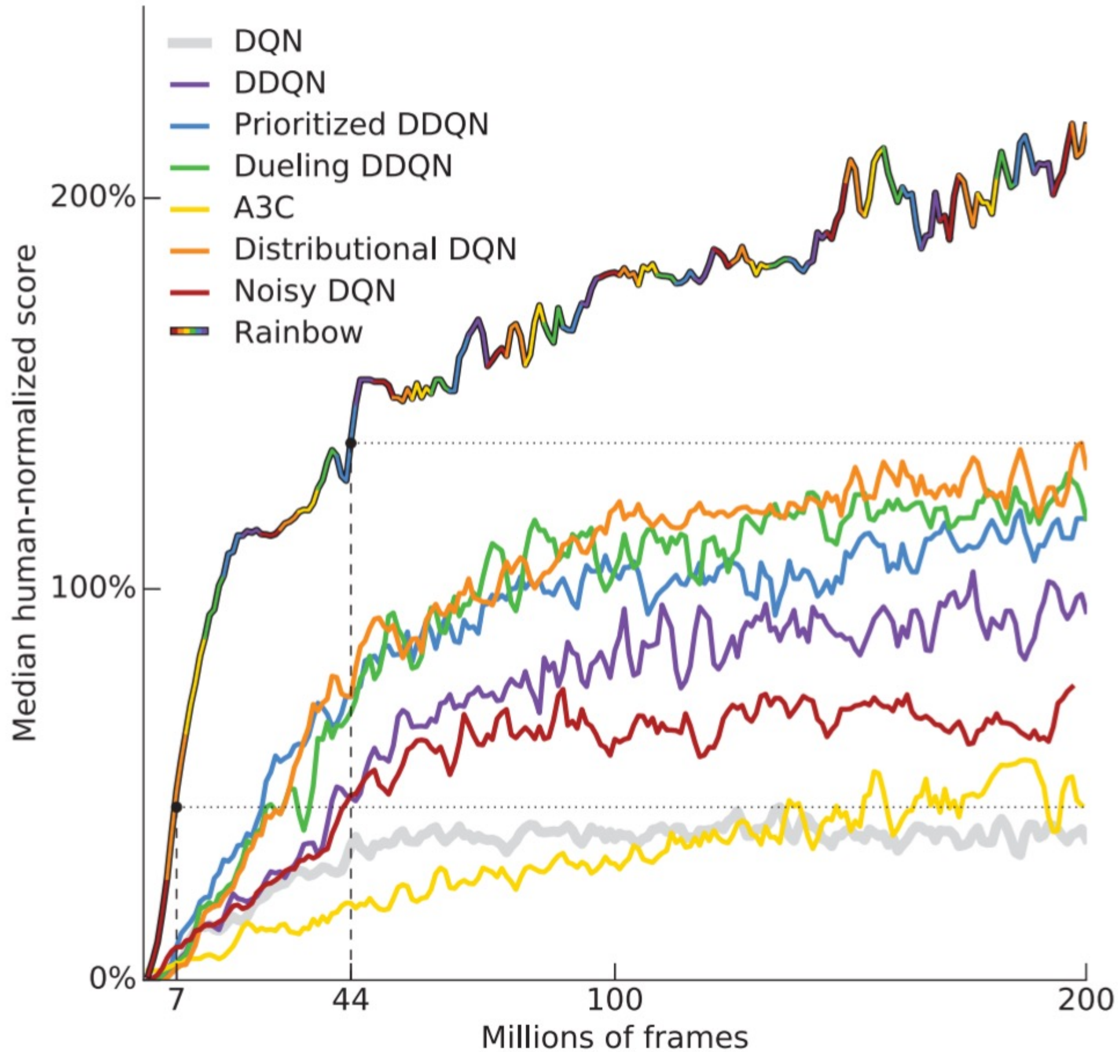
"Double Reinforcement Learning with Double Q-Learning", van Hasselt et al. AAAI 2016.

- Dueling DQN：精细捕捉价值和行动的细微关联、多种advantage函数建模

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)$$

"Dueling Network Architectures for Deep Reinforcement Learning", Wang et al. ICML 2016.

Rainbow: 结合众多Value-based DRL方法

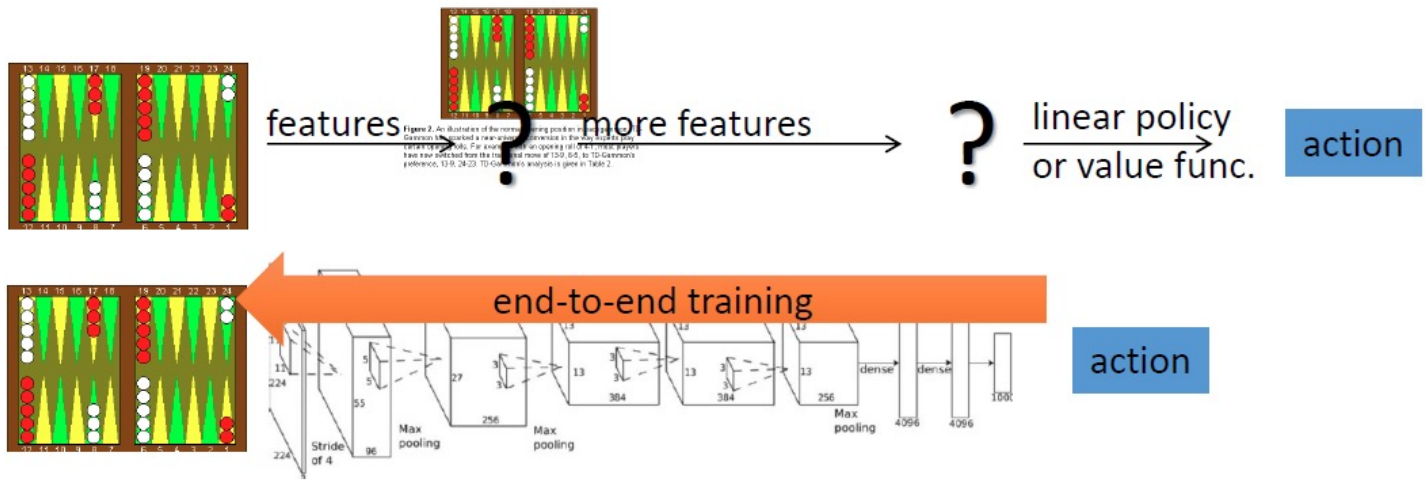


Hessel et al. Rainbow: Combining Improvements in Deep Reinforcement Learning. AAAI 2018.

深度强化学习总结

标准（传统）强化学习

深度强化学习



- 直面理解：深度学习+强化学习
- 深度强化学习使强化学习算法能够以端到端的方式解决复杂问题
- 真正让强化学习有能力完成实际决策任务
- 比强化学习和深度学习各自都更加难以驯化
- 基于价值函数的深度强化学习
  - DQN：一次输入多个行动Q值输出、目标网络、随机采样经验
  - Double DQN：解耦合行动选择和价值估计、解决DQN过高估计问题
  - Dueling DQN：精细捕捉价值和行动的细微关联、多种advantage函数建模

策略梯度

参数化策略

- 我们能够将策略参数化

$$\pi_\theta(a|s)$$

策略可以是确定性的

$$a = \pi_\theta(s)$$

也可以是随机的

$$\pi_\theta(a|s) = P(a|s;\theta)$$

- θ 是策略的参数
- 将可见的已知状态泛化到未知的状态上
- 在本课程中我们主要讨论的是模型无关的强化学习

基于策略的强化学习

优点

- 具有更好的收敛性质
- 在高维度或连续的动作空间中更有效
  - 最重要的因素：基于值函数的方法，通常需要取最大值
- 能够学习出随机策略

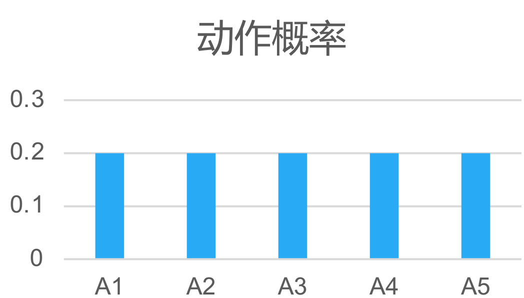
缺点

- 通常会收敛到局部最优而非全局最优
- 评估一个策略通常不够高效并具有较大的方差（variance）

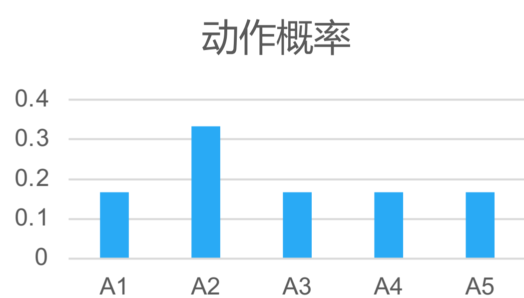
策略梯度

- 对于随机策略$\pi_\theta(a|s) = P(a|s;\theta)$
- 直觉上我们应该
  - 降低带来较低价值/奖励的动作出现的概率
  - 提高带来较高价值/奖励的动作出现的概率
- 一个离散动作空间维度为5的例子

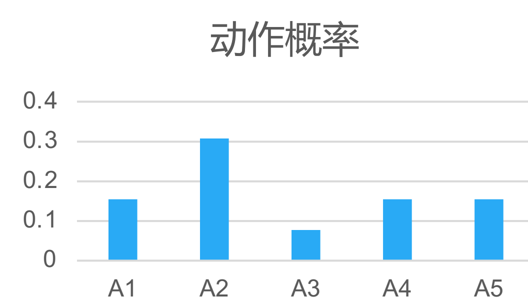
1. 初始化θ



3. 根据策略梯度更新θ



5. 根据策略梯度更新θ



2. 采取动作A2
观察到正的奖励

4. 采取动作A3
观察到负的奖励

单步马尔可夫决策过程中的策略梯度

- 考虑一个简单的单步马尔可夫决策过程
  - 起始状态为$s \sim d(s)$
  - 决策过程在进行一步决策后结束，获得奖励值为r_{sa}
- 策略的价值期望

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r] = \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(a|s) r_{sa}$$

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{s \in S} d(s) \sum_{a \in A} \frac{\partial \pi_\theta(a|s)}{\partial \theta} r_{sa}$$

似然比 (Likelihood Ratio)

- 似然比利用下列特性

$$\frac{\partial \pi_\theta(a|s)}{\partial \theta} = \pi_\theta(a|s)\frac{1}{\pi_\theta(a|s)}\frac{\partial \pi_\theta(a|s)}{\partial \theta}$$

$$= \pi_\theta(a|s)\frac{\partial \log \pi_\theta(a|s)}{\partial \theta}$$

- 所以策略的价值期望可以写成

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r] = \sum_{s\in S} d(s) \sum_{a\in A} \pi_\theta(a|s) \boxed{r_{sa}}$$

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{s\in S} d(s) \sum_{a\in A} \frac{\partial \pi_\theta(a|s)}{\partial \theta} r_{sa}$$

$$= \sum_{s\in S} d(s) \sum_{a\in A} \pi_\theta(a|s) \boxed{\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} r_{sa}}$$

$$= \mathbb{E}_{\pi\theta}\left[\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} r_{sa}\right]$$

这一结果可以通过从d(s)中采样状态s和从πθ中采样动作a来近似估计

策略梯度定理

- 策略梯度定理把似然比的推导过程泛化到多步马尔可夫决策过程
  - 用长期的价值函数 $Q^{\pi_\theta}(s,a)$代替前面的瞬时奖励 r_{sa}
- 策略梯度定理涉及
  - 起始状态目标函数 J_1，平均奖励目标函数 J_{avR}，和平均价值目标函数 J_{avV}
- 定理
  - 对任意可微的策略 $\pi_\theta(a|s)$，任意策略的目标函数 $J = J_1, J_{avR}, J_{avV}$，其策略梯度是

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

详细证明过程请参考:

1. Rich Sutton’s Reinforcement Learning: An Introduction (2nd Edition)第13章
2. 动手学强化学习策略梯度的附录
   https://hrl.boyuai.com/chapter/2/%E7%AD%96%E7%95%A5%E6%A2%AF%E5%BA%A6%E7%AE%97%E6%B3%95/

蒙特卡洛策略梯度（REINFORCE）

- 利用随机梯度上升更新参数
- 利用策略梯度定理
- 利用累计奖励值G_t作为$Q^{\pi_\theta}(s,a)$的无偏采样

$$\Delta\theta_t = \alpha \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta} G_t$$

- REINFORCE算法

initialize θ arbitrarily

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

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

    $$\theta \leftarrow \theta + \alpha \frac{\partial}{\partial \theta} \log \pi_\theta(a_t|s_t) G_t$$

  end for

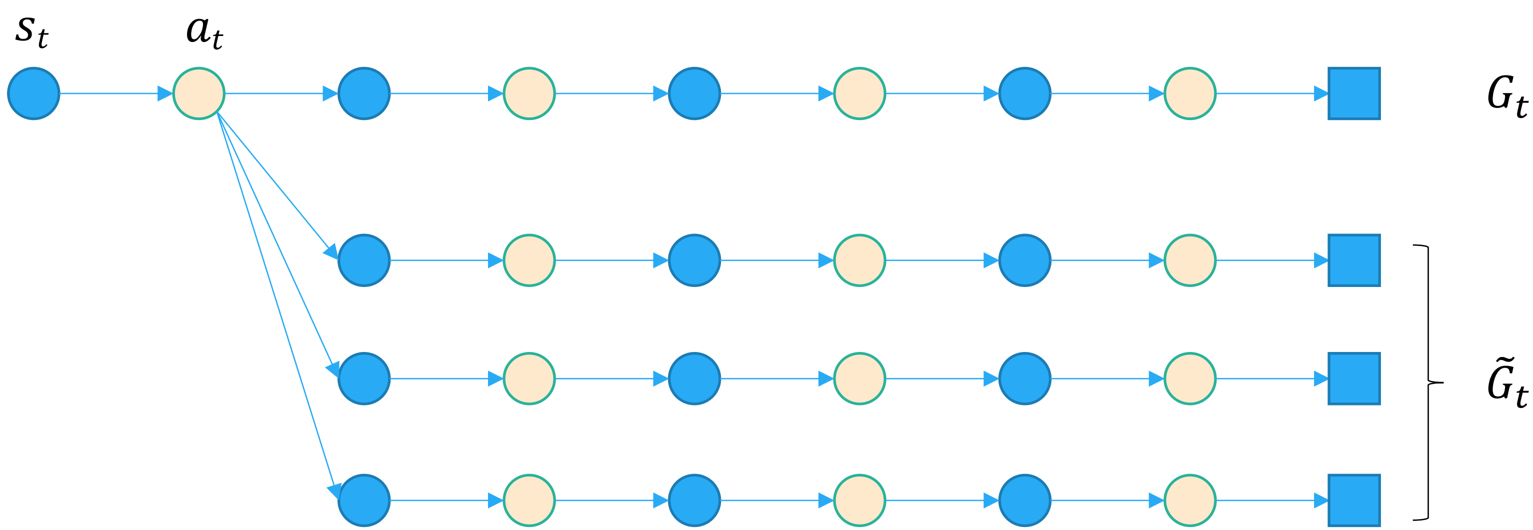
end for

return θ

蒙特卡洛策略梯度（REINFORCE）

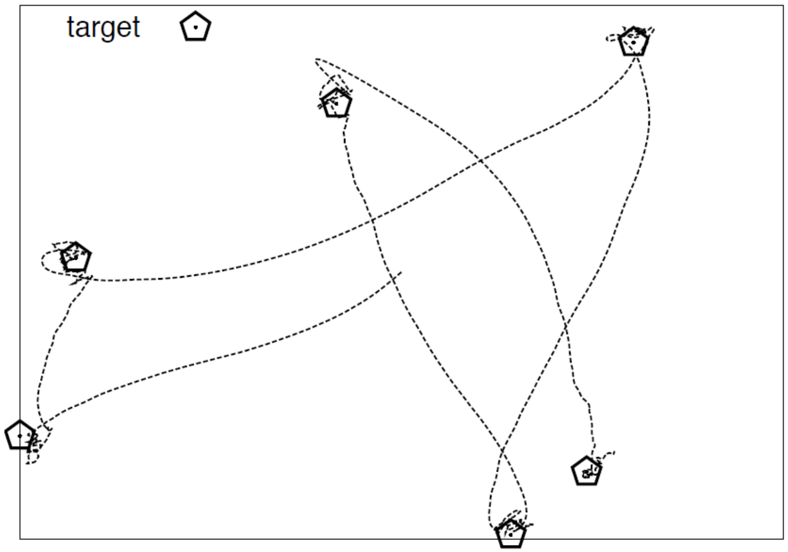
$$\Delta\theta_t = \alpha \frac{\partial \log \pi_\theta(a_t|s_t)}{\partial \theta} G_t$$

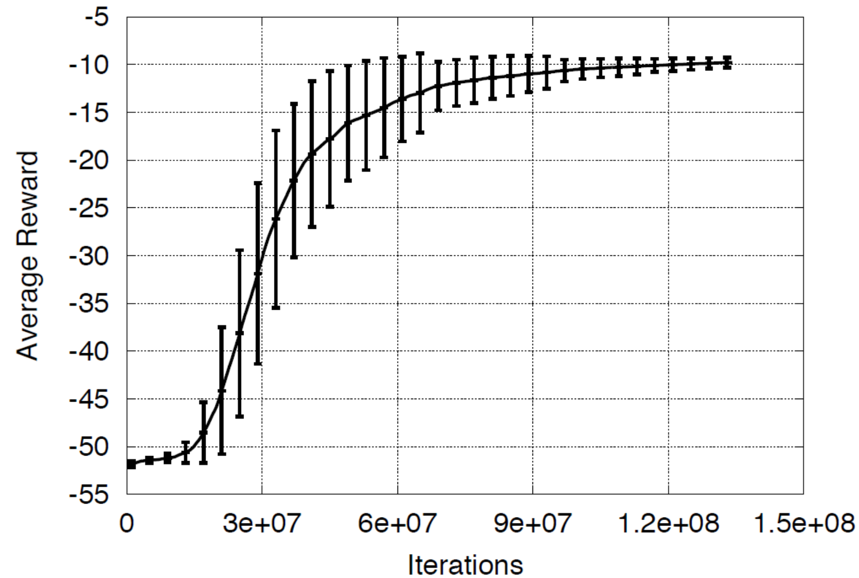
- 可通过多次roll-out的G_t平均值来逼近$Q(s_t, a_t)$



$$\tilde{G}_t = \frac{1}{N}\sum_{i=1}^{n} G_t^{(i)}$$

Puck World 冰球世界示例





- 连续的动作对冰球施加较小的力
- 冰球接近目标可以得到奖励
- 目标位置每30秒重置一次
- 使用蒙特卡洛策略梯度方法训练策略

Actor-Critic

REINFORCE 存在的问题

- 基于片段式数据的任务
  - 通常情况下，任务需要有终止状态，REINFORCE才能直接计算累计折扣奖励
- 低数据利用效率
  - 实际中，REINFORCE需要大量的训练数据
- 高训练方差（最重要的缺陷）
  - 从单个或多个片段中采样到的值函数具有很高的方差

Actor-Critic

- Actor-Critic的思想
  - REINFORCE策略梯度方法：使用蒙特卡洛采样直接估计(s_t, a_t)的值G_t
  - 为什么不建立一个可训练的值函数Q_Φ来完成这个估计过程？
- 演员（Actor）和评论家（Critic）

演员 $\pi_\theta(a|s)$

采取动作使评论家满意的策略

评论家 $Q_\Phi(s, a)$

学会准确估计演员策略所采取动作价值的值函数

Actor-Critic训练

- 评论家Critic: $Q_\Phi(s,a)$
  - 学会准确估计当前演员策略（actor policy）的动作价值

$$Q_\Phi(s,a) \simeq r(s,a) + \gamma \mathbb{E}_{s'\sim p(s'|s,a),a'\sim\pi_\theta(a'|s')}[Q_\Phi(s',a')]$$

- 演员Actor: $\pi_\theta(a|s)$
  - 学会采取使critic满意的动作

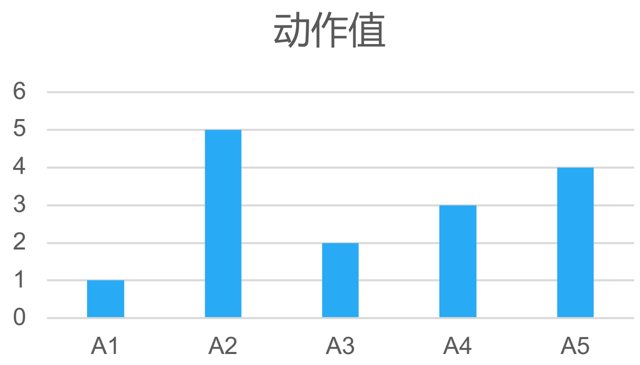
$$J(\theta) = \mathbb{E}_{s\sim p,\pi_\theta}[\pi_\theta(a|s)Q_\Phi(s,a)]$$

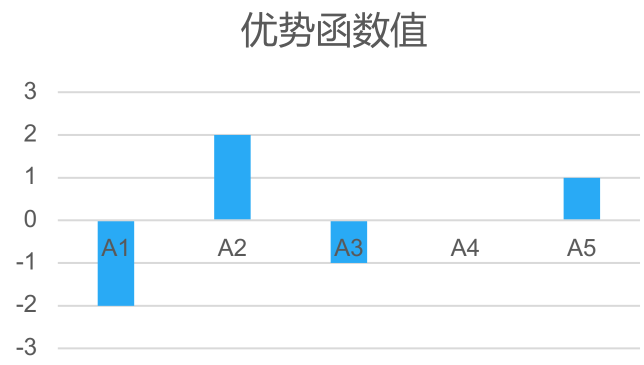
$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_{\pi_\theta}\left[\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} Q_\Phi(s,a)\right]$$

A2C： Advantageous Actor-Critic

- **思想**：通过减去一个基线函数来标准化评论家的打分
  - 更多信息指导：降低较差动作概率，提高较优动作概率
  - 进一步降低方差
- **优势函数（Advantage Function）**

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





A2C：Advantageous Actor-Critic

- 状态-动作值和状态值函数

$$
\begin{aligned}
Q^{\pi}(s,a) &= r(s,a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a), a' \sim \pi_\theta(a'|s')}[Q_\Phi(s',a')] \\
&= r(s,a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)}[V^{\pi}(s')]
\end{aligned}
$$

- 因此我们只需要拟合状态值函数来拟合优势函数

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

采样下一个状态s'

价值和策略的近似逼近方法总结

- 价值和策略的近似逼近方法是强化学习技术从‘玩具’ 走向 ‘现实’的第一步，是深度强化学习的基础设置
- 参数化的价值函数和策略
- 通过链式法则，价值函数的参数可以被直接学习
- 通过likelihood-ratio方法，可以用advantage对策略的参数进行学习
- Actor-critic框架同时学习了价值函数和策略，通过价值函数的Q（或Advantage）估计，以策略梯度的方式更新策略参数

A3C: 异步A2C方法

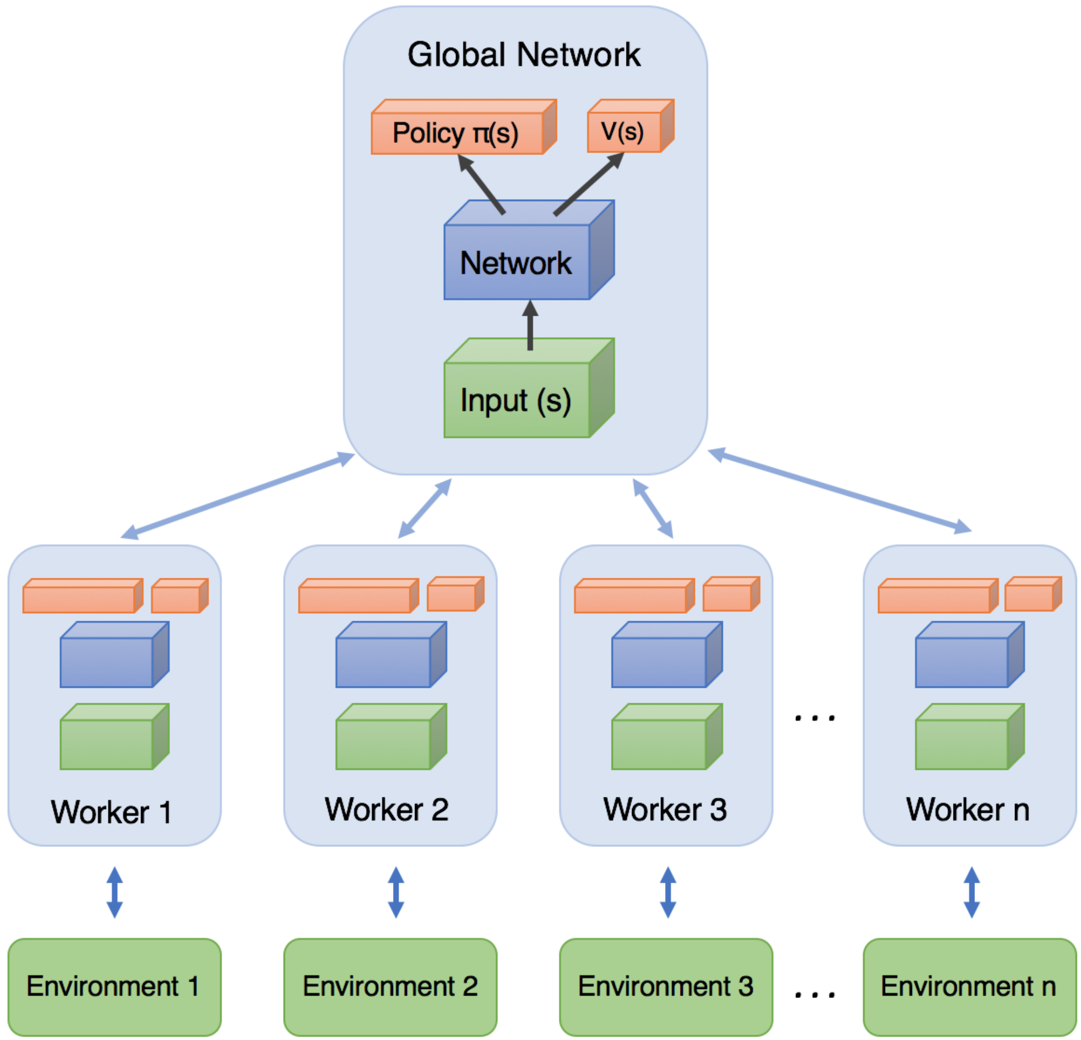
- A3C代表了异步优势动作评价（Asynchronous Advantage Actor Critic）
  - 异步（Asynchronous）：因为算法涉及并行执行一组环境
  - 优势（Advantage）：因为策略梯度的更新使用优势函数
  - 动作评价（Actor Critic）：因为这是一种动作评价（actor-critic）方法，它涉及一个在学得的状态值函数帮助下进行更新的策略

$$\nabla_{\theta'} \log \pi(a_t|s_t;\theta')A(s_t,a_t;\theta_v)$$

$$A(s_t,a_t;\theta_v)=\sum_{i=0}^{k-1}\gamma^i r_{t+i}+\gamma^k V(s_{t+k};\theta_v)-V(s_t;\theta_v)$$

https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2

A3C: 异步A2C方法



https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2

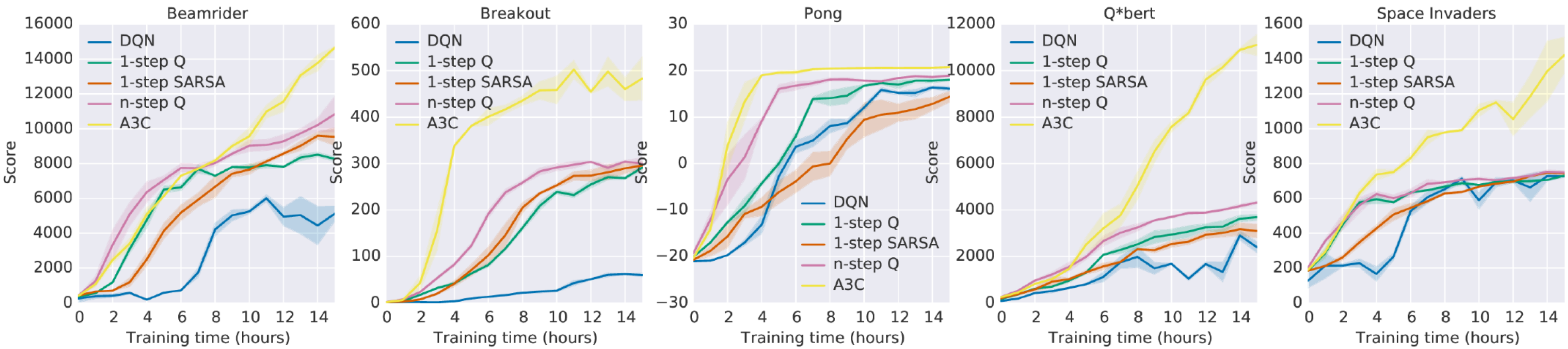
A3C算法

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial\left(R - V(s_i;\theta'_v)\right)^2 / \partial\theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Volodymyr Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. NIPS 2016.

A3C对比实验



a single Nvidia K40 GPU while the asynchronous methods were trained using 16 CPU cores

Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorila	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

Nvidia K40 GPUs

16 CPU cores and no GPU

Mean and median human-normalized scores on 57 Atari games

Volodymyr Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. NIPS 2016.

确定性策略梯度

随机策略与确定性策略

随机策略

对于离散动作 $$\pi(a|s;\theta) = \frac{\exp\{Q_\theta(s,a)\}}{\sum_{a'}\exp\{Q_\theta(s,a')\}}$$

对于连续动作 $$\pi(a|s;\theta) \propto \exp\left\{\left(a - \mu_\theta(s)\right)^2\right\}$$

确定性策略

对于离散动作 $$\pi(s;\theta) = \arg\max_a Q_\theta(s,a)$$ 不可微

对于连续动作 $$a = \pi(s;\theta)$$ 可微

确定性策略梯度

- 用于估计状态-动作值的评论家（critic）模块

$$Q^w(s,a) \simeq Q^\pi(s,a)$$

$$L(w) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}\left[\left(Q^w(s,a) - Q^\pi(s,a)\right)^2\right]$$

- 确定性策略
  - 确定性策略梯度定理

$$J(\pi_\theta) = \mathbb{E}_{s\sim\rho^\pi}[Q^w(s,a)]$$

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

在线策略　　　　链式法则

D. Silver et al. Deterministic Policy Gradient Algorithms. ICML 2014.

深度确定性策略梯度

DDPG：深度确定性策略梯度

- 对于确定性策略的梯度

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

- 在实际应用中，这种带有神经函数近似器的actor-critic方法在面对有挑战性的问题时是不稳定的
- 深度确定性策略梯度（DDPG）给出了在确定性策略梯度（DPG）基础上的解决方法
  - 经验重放（离线策略）
  - 目标网络
  - 在动作输入前批标准化Q网络
  - 添加连续噪声

Lillicrap et al. Continuous control with deep reinforcement learning. NIPS 2015.

THANK YOU