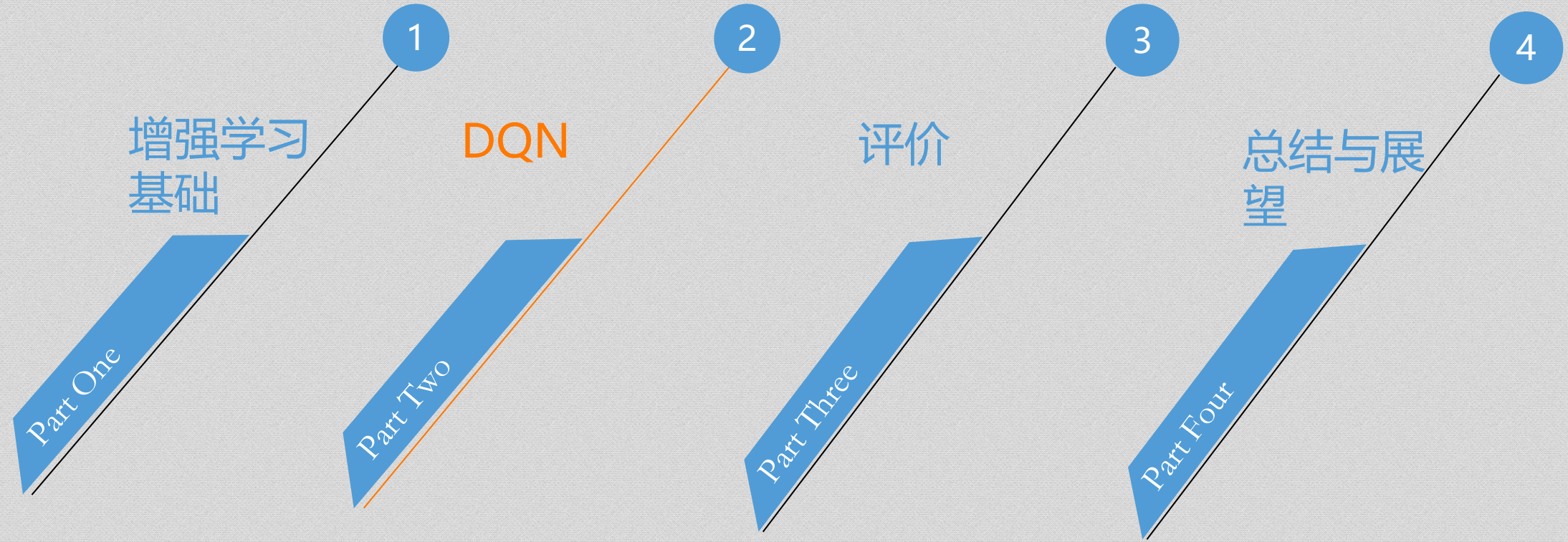
2015 Nature
Human-level control through deep reinforcement learning

韩玮光

2016-12-28


1
增强学习基础
Part One
2
DQN
Part Two
3
评价
Part Three
4
总结与展望
Part Four

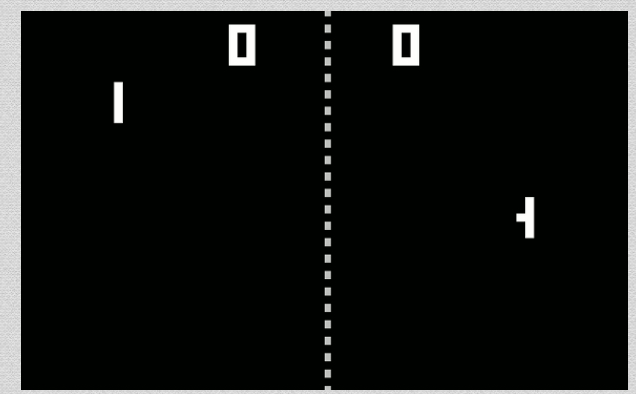
Part 1

增强学习基础

增强学习

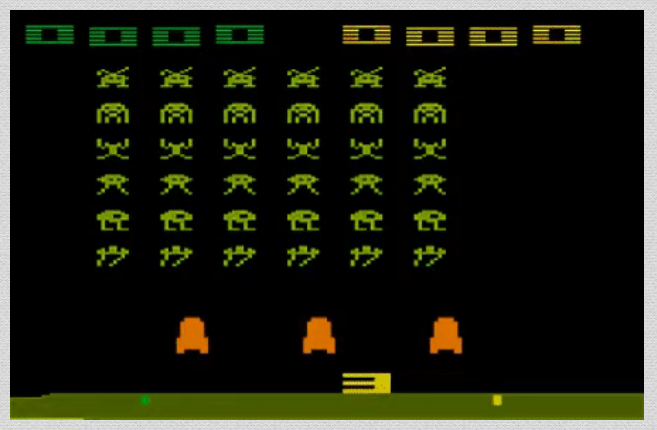
机器学习

- 有监督学习：每个训练样本都有标签
- **增强学习**：稀疏，并且具有时延的标签
- 无监督学习：完全没有标签



Pong

Breakout



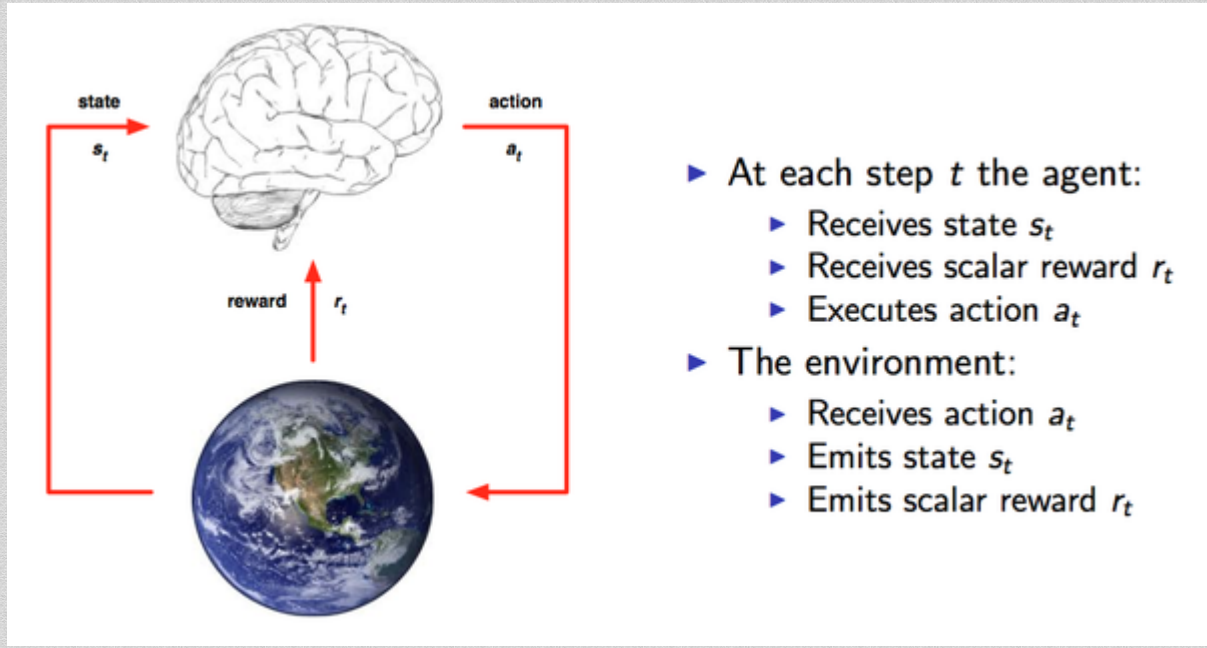
Space Invaders

Seaquest

增强学习

在人工智能领域，一般用**智能体Agent**来表示一个具备行为能力的物体，比如机器人，无人车，人等等。那么增强学习考虑的问题就是**智能体Agent**和**环境Environment**之间交互的任务。

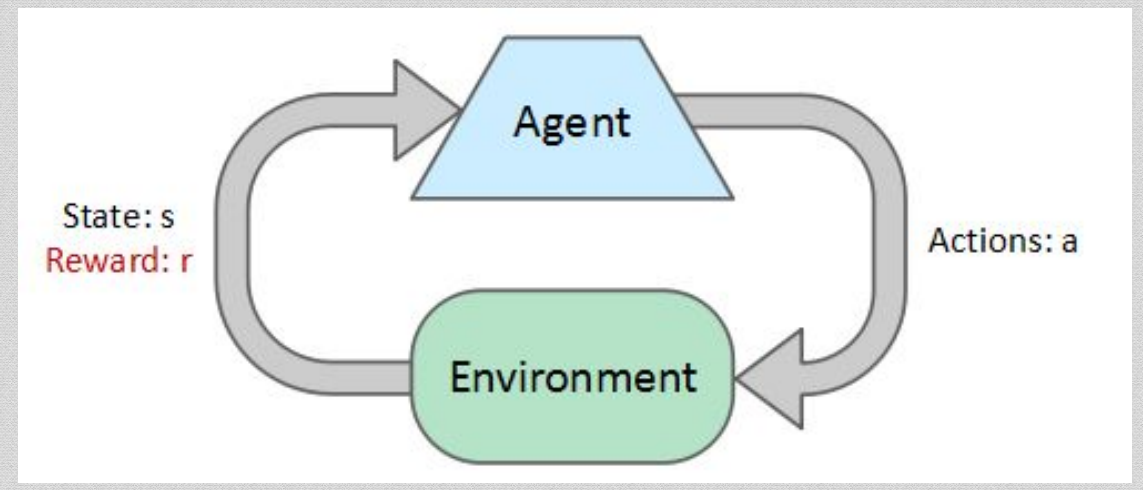
不管是什么样的任务，都包含了一系列的**动作Action**,**观察Observation**还有**反馈值Reward**。所谓的Reward就是Agent执行了动作与环境进行交互后，环境会发生变化，变化的好与坏就用Reward来表示。



所谓的Reward就是Agent执行了动作与环境进行交互后，环境会发生变化，变化的好与坏就用Reward来表示。

策略Policy

- **任务的目标：**获取尽可能多的Reward。
- **步骤：**每个时间片，Agent都是根据当前的观察来确定下一步的动作。观察Observation的集合就作为Agent的所处的**状态State**，因此，**状态State**和**动作Action**存在映射关系，也就是一个state可以对应一个action，或者对应不同动作的概率（常常用概率来表示，概率最高的就是最值得执行的动作）。



- **策略Policy：**状态与动作的关系其实就是输入与输出的关系，而状态State到动作Action的过程就称之为一个**策略Policy，**一般用π表示

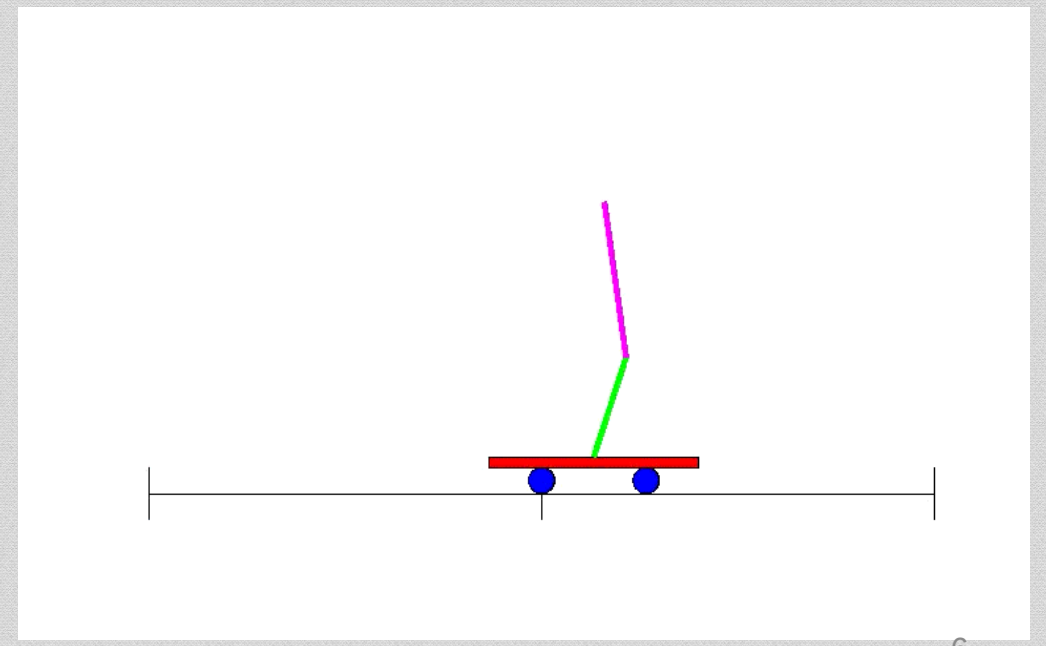
$$a = \pi(s)$$

或者

$$\pi(a|s)$$

- **样本Sample：** $\{s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_t, a_t, r_t\}$

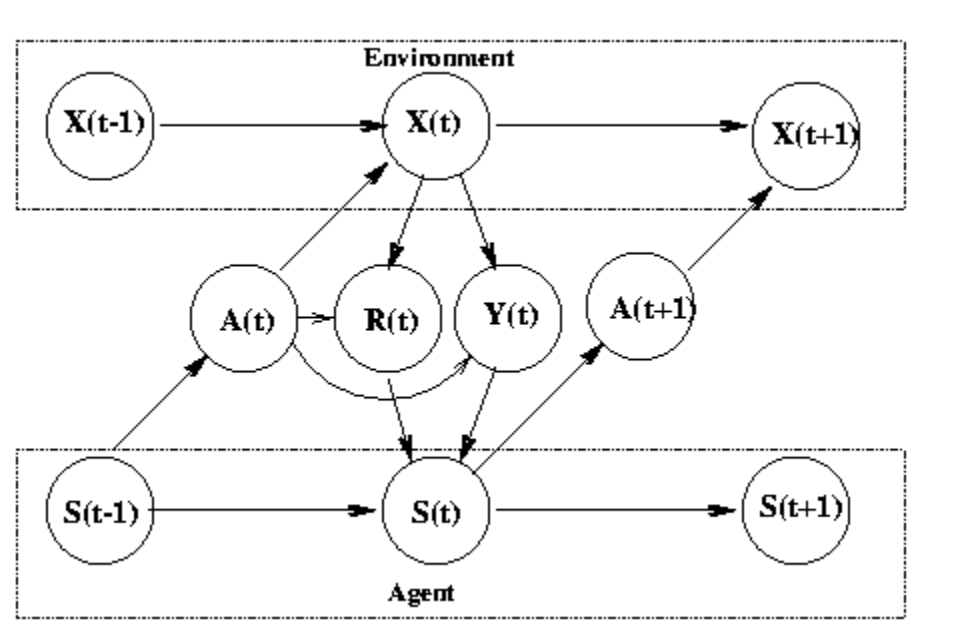
state
action
reward

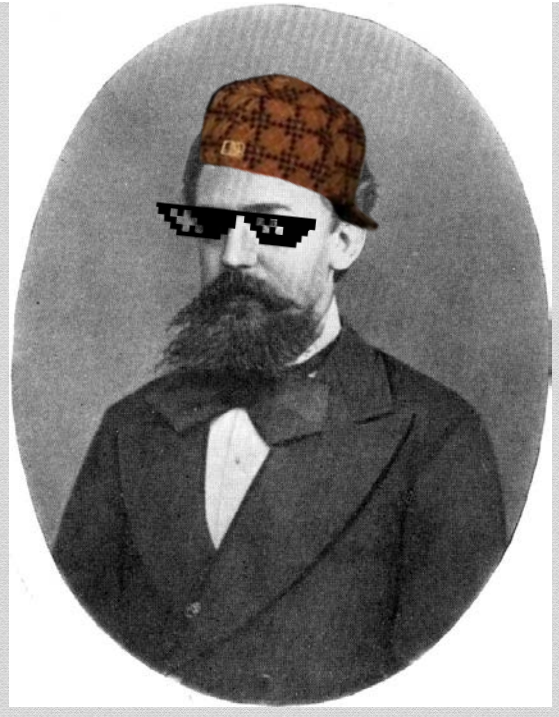


MDP(马尔科夫决策过程)

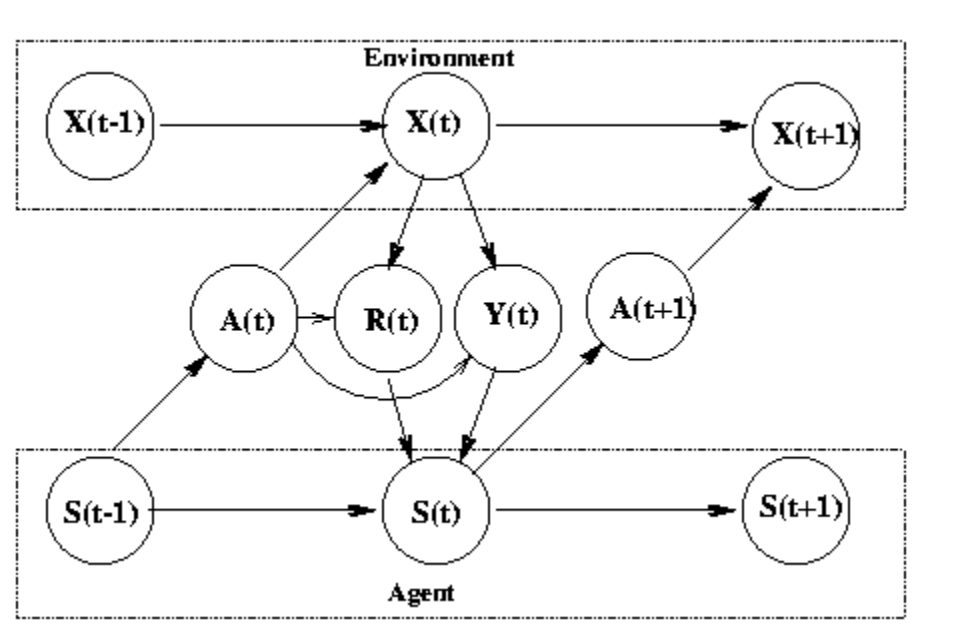
一个基本的MDP可以用（S,A,P）来表示，S表示状态，A表示动作，P表示状态转移概率，也就是根据当前的状态s_t和a_t转移到s_{t+1}的概率。

- **Model-based的方法**：通过模型来获取最优动作的方法。得到转移概率P，称为我们获得了**模型Model**，有了模型，未来就可以求解，那么获取最优的动作也就有可能。
- **Model-free的方法**：不通过模型来获取最优动作的方法。DQN就是一种Model-free的方法。





回报（Result）与价值（Value）



- **回报Result**：引入**回报Return**来表示某个时刻t的状态将具备的回报

$$G_t = R_{t+1} + \lambda R_{t+2} + \lambda^2 R_{t+3} + \cdots + \lambda^{n-t} R_n = \sum_{k=0}^{\infty} \lambda^k R_{t+k+1}$$

R：Reward反馈

λ：discount factor折扣因子，一般小于1，就是说一般当下的反馈是比较重要的，时间越久，影响越小

- **价值函数Value Function**：用value function $v(s)$来表示一个状态未来的潜在价值。状态的好坏可以用对未来回报的期望来描述

$$v(s) = \mathbb{E}[G_t | S_t = s]$$

- 获取最优的**策略Policy**：
  - 直接优化策略$\pi(a|s)$或者$a = \pi(s)$使得回报更高
  - 通过估计value function来间接获得优化的策略，DQN就是基于value function的算法
  - 融合上面的两种做法，actor-critic算法

Bellman方程

回报Result的基本定义：所有Reward的累加

$$G_t = R_{t+1} + \lambda R_{t+2} + \cdots = \sum_{k=0}^{\infty} \lambda^k R_{t+k+1}$$

Bellman方程：

$$\begin{aligned} v(s) &= \mathbb{E}[G_t | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \lambda R_{t+2} + \lambda^2 R_{t+3} + \cdots | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \lambda G_{t+1} | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \lambda v(S_{t+1}) | S_t = s] \end{aligned}$$

当前状态的价值和下一步的价值以及当前的反馈Reward有关

Value function： state对应的reward是多种动作对应的reward的期望值

Action-Value function：执行完动作action之后得到的reward

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}[r_{t+1} + \lambda r_{t+2} + \lambda^2 r_{t+3} + \cdots | s, a] \\ &= \mathbb{E}_{s'}[r + \lambda Q^{\pi}(s', a') | s, a] \end{aligned}$$

求解最优策略：$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = \mathbb{E}_{s'}[r + \lambda \max_{a'} Q^*(s', a') | s, a]$

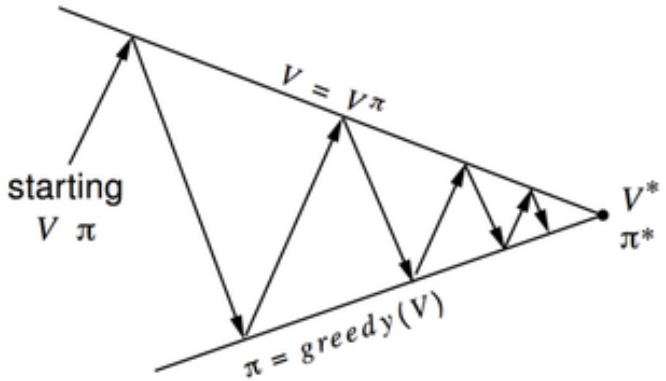
策略迭代

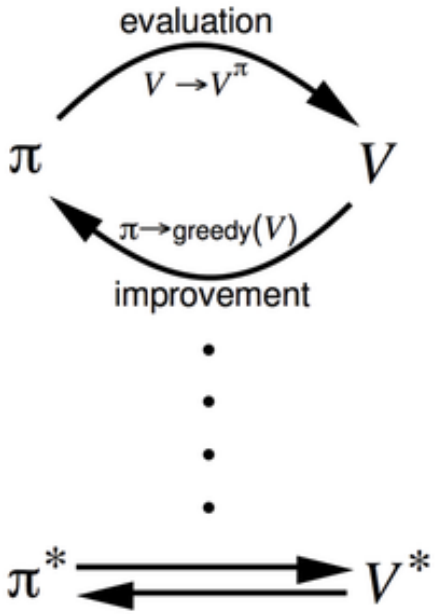
$$
\begin{aligned}
v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big],
\end{aligned}
$$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $a \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
   If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return V and π; else go to 2





Policy evaluation Estimate v_π
Any policy evaluation algorithm
Policy improvement Generate $\pi' \geq \pi$
Any policy improvement algorithm

策略迭代VS值迭代

$$
\begin{aligned}
v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big],
\end{aligned}
$$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
      $\Delta \leftarrow 0$
      For each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      $a \leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
      If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return V and π; else go to 2

修改成迭代形式：

$$
\begin{aligned}
v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_*(s')\Big]
\end{aligned}
$$

$$
\begin{aligned}
v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big],
\end{aligned}
$$

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
      $v \leftarrow V(s)$
      $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
   $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

对于动作价值函数，迭代公式修改成为：

$$Q_{i+1}(s,a) = \mathbb{E}[r + \lambda \max_{a'} Q_i(s', a')|s, a]$$

Q-Learning

之前求解最优策略的方法：

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = \mathbb{E}_{s'}[r + \lambda \max_{a'} Q^*(s',a') \,|s,a]$$

Q Learning提出了一种更新Q值的办法：

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\left(R_t + \lambda \max_a Q(S_{t+1},a) - Q(S_t,A_t)\right)$$

α：学习率

Off-policy：需要使用某一个policy来生成动作，但是这个policy不是优化的那个policy

Model-free：Q-Learning完全不考虑model模型也就是环境（P$(s',r|s,a)$这样一个概率分布）的具体情况，只考虑看到的环境及reward

选择Policy的方法：

- 随机的生成一个动作
- 根据当前的Q值计算出一个最优的动作，这个policyπ称之为greedy policy贪婪策略。

$$\pi(S_{t+1} = argmax_a(S_{t+1},a))$$

$\epsilon - greedy$策略：

ϵ是一个很小的值，作为选取随机动作的概率值。更改ϵ的值从而得到不同的exploration和exploitation的比例。

初始化$Q(s,a), \forall s \in S, a \in A(s)$,任意的数值，并且$Q(terminal-state,\cdot) = 0$

重复（对每一节episode）：

初始化 状态S

重复（对episode中的每一步）：

使用某一个policy比如（$\epsilon - greedy$)根据状态S选取一个动作执行

执行完动作后，观察reward和新的状态S'

$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha(R_t + \lambda \max_a Q(S_{t+1},a) - Q(S_t,A_t))$

$S \leftarrow S'$

循环直到S终止

Part 2

DQN

传统方式的烹饪

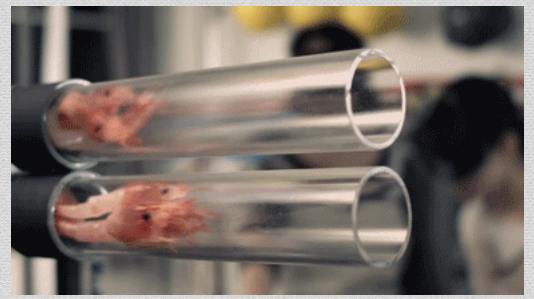
传统方式的烹饪

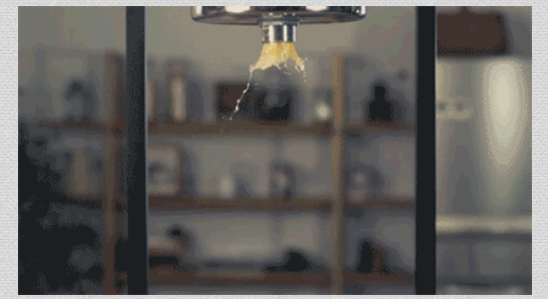
传统方式的烹饪

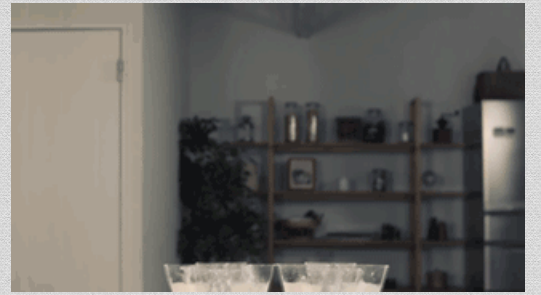
传统方式的烹饪

传统方式的烹饪

端到端的烹饪

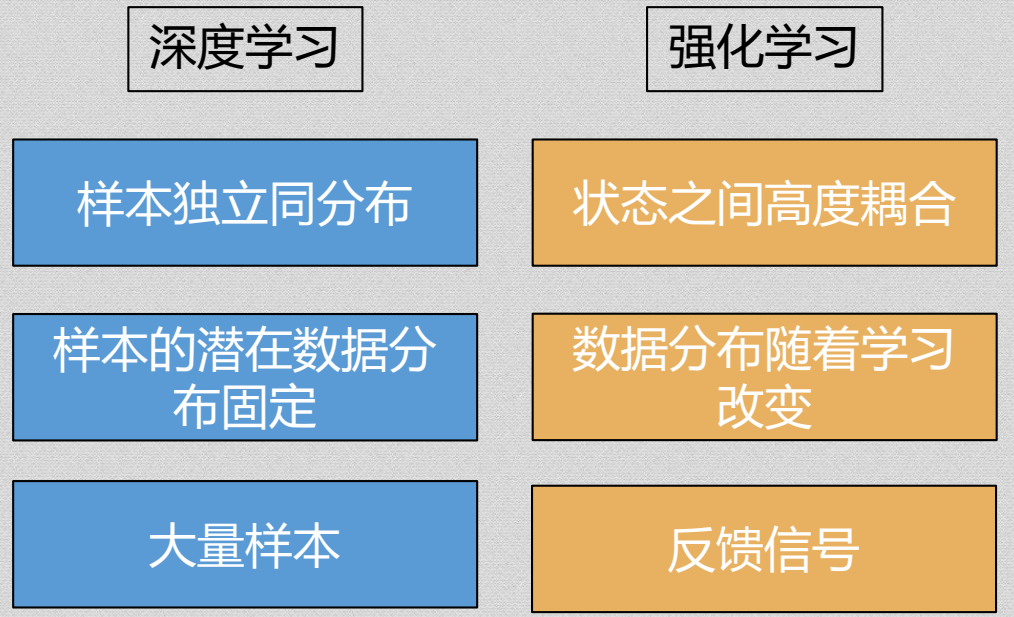






深度学习&增强学习

- 将深度学习应用在增强学习上的一些问题：
  - 一方面，很多成功的深度学习应用需要大量的手工标注的数据，另外一方面，强化学习需要从一个稀疏的、有噪声、有延迟的标量反馈信号中优化模型
  - 许多的深度学习算法假设每次采样的样本是独立的，而强化学习的样本序列包含了高度相关的状态 $\{s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_t, a_t, r_t\}$
  - 算法学习到的新的行为会改变数据的分布，而深度学习的方法假设样本是由一个潜在的固定不变的分布产生



问题：

- 学习的过程能否保证稳定，能否有较好的学习效果

解决：

- 经验重放(experience replay)
- 目标网络(target network)
- $\epsilon - greedy$

价值函数近似

- 维度灾难：原始图像状态数太多

$$256^{210\times160}$$

- 价值函数近似Value Function Approximation：对状态的维度进行压缩

$$Q(s,a)\approx f(s,a)$$

  - 线性函数：$Q(s,a)\approx w_1 s+w_2 a+b$
  - 非线性函数：$Q(s,a)\approx f(s,a;\theta)$，$\theta$为神经网络的参数

- Q函数的改造：
  - 状态s维度较高（原始图像），动作a维度较低（离散的动作，比如上下左右）
  - 对高维状态s进行降维，但不需要对动作a进行降维
  - 对每个动作a，都要重新计算一遍Q函数，需要较大的计算量

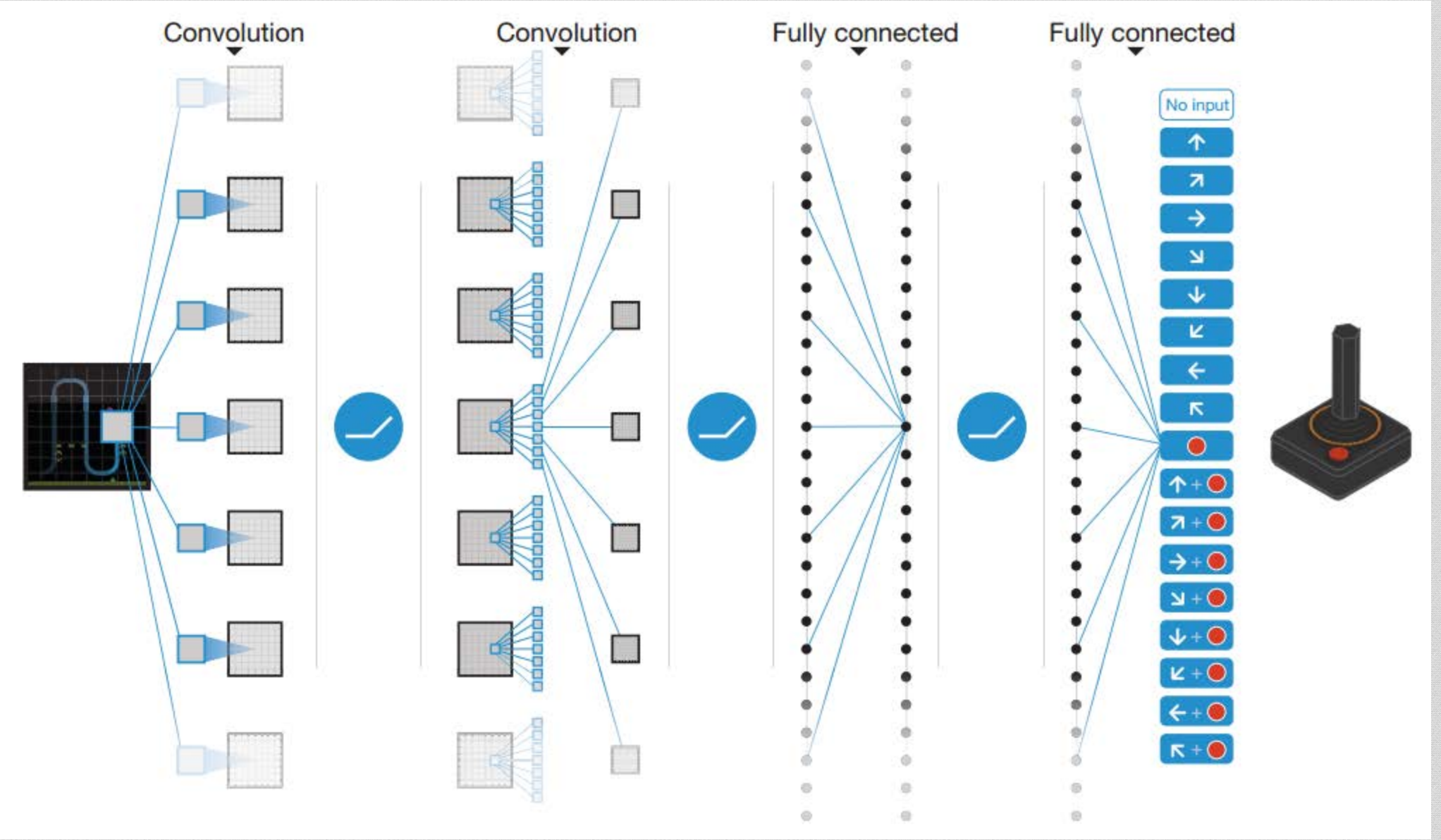
解决方案：改造Q函数，实现高维状态输入，低维动作输出

$$Q(s)\approx f(s;w)$$
$$=[Q(s,a_1),Q(s,a_2),Q(s,a_3),\dots,Q(s,a_n)]^T$$

使用固定长度的图像（比如连续的4帧画面）ϕ作为状态输入到神经网络之中，

$$Q(s)\approx f(\phi;\theta)$$

DQN


Convolution
Convolution
Fully connected
Fully connected
No input

经验重放&目标网络

Q-learning:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_t + \lambda \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right)$$

- 经验重放：

为了降低状态和状态之间的耦合，从agent玩游戏的经历中建立了数据集

1. 根据$\epsilon - greedy$策略，选择一个action a_t
2. 在玩游戏的过程中，将状态转移序列$< s_t, a_t, r_{t+1}, s_{t+1} >$存储在回放记忆（replay memory）$D$中
3. 从D中均匀采样一系列的状态转移序列$< s, a, r, s' >$作为mini-batch
4. 优化Q-network和target Q-network之间的MSE

$$L(\theta) = \mathbb{E}[(\underbrace{r + \gamma maxQ(s', a'; \theta)}_{Target} - Q(s, a; \theta)^2)]$$

s',a'是下一个时刻的状态和动作

- 目标网络：

修改上述的损失函数：

$$L(\theta) = \mathbb{E}[(\underbrace{r + \gamma maxQ(s', a'; \theta^-)}_{Target} - Q(s, a; \theta)^2)]$$

计算目标Q值的网络使用的参数是θ^-而不是θ,这个θ还是从训练的Q-network中来，训练了一段时间C将当前Q-network的参数值复制给目标Q-network

- 梯度更新：

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right]$$

算法

$\epsilon - greedy$策略

经验回放

目标网络

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For t = 1,T **do**

With probability ε select a random action a_t

otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

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 D

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

Every C steps reset $\hat{Q} = Q$

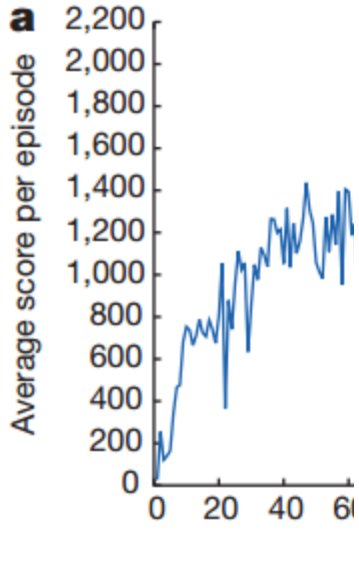
End For

End For

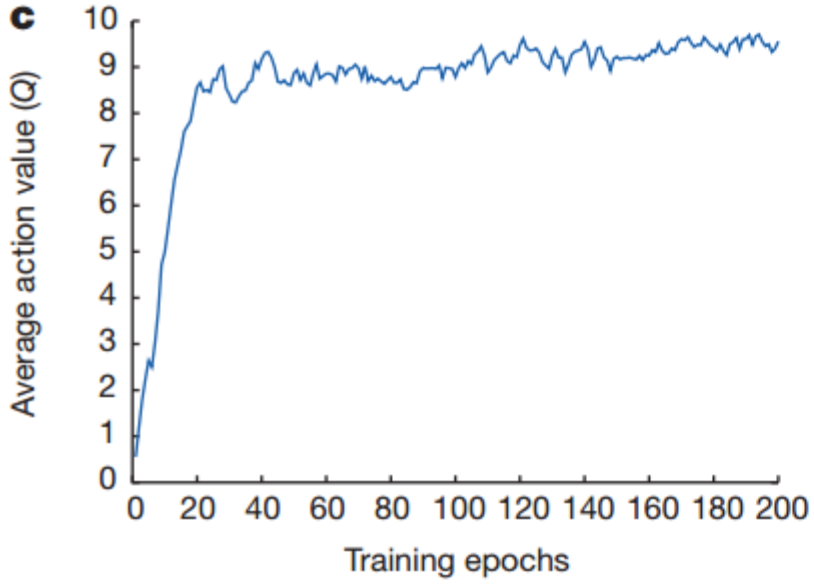
Part 3

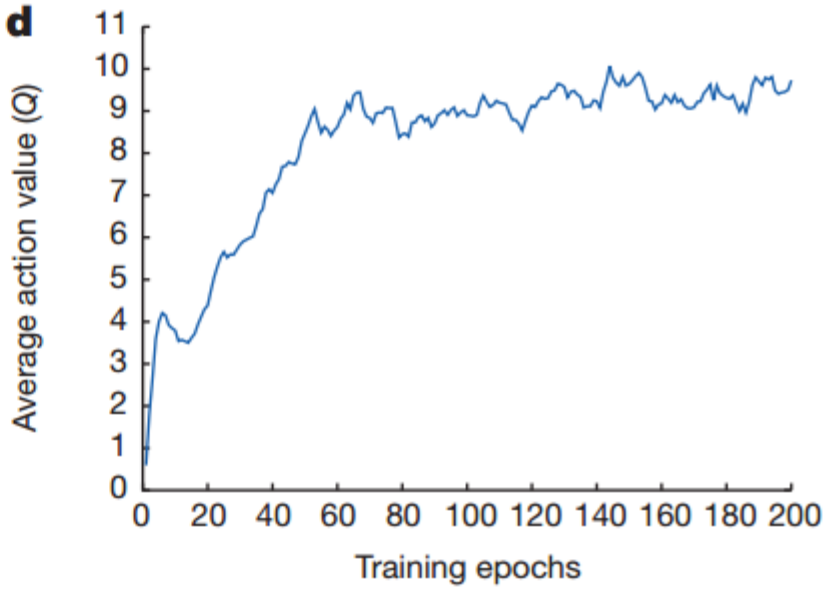
评价

训练的稳定性



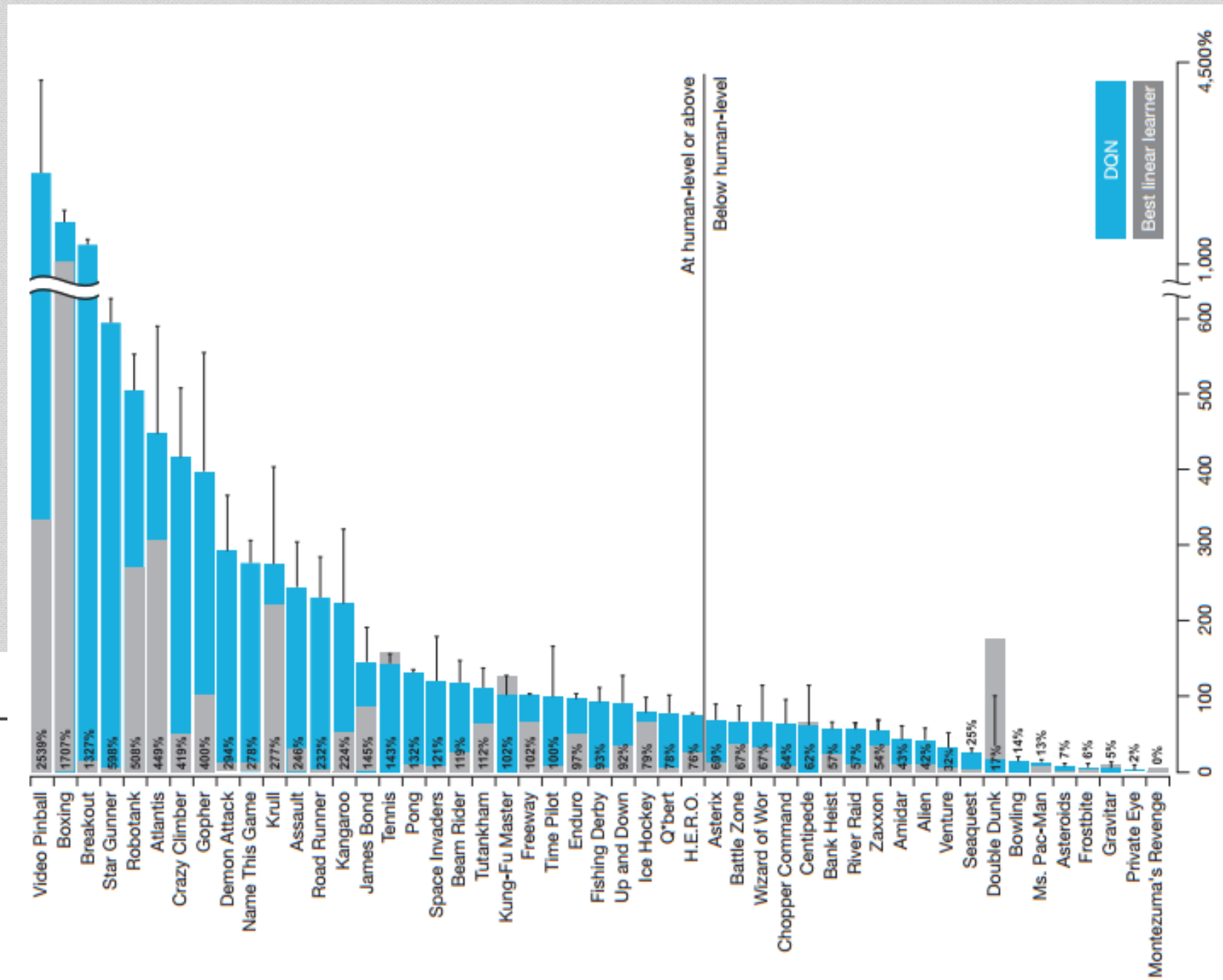
Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0



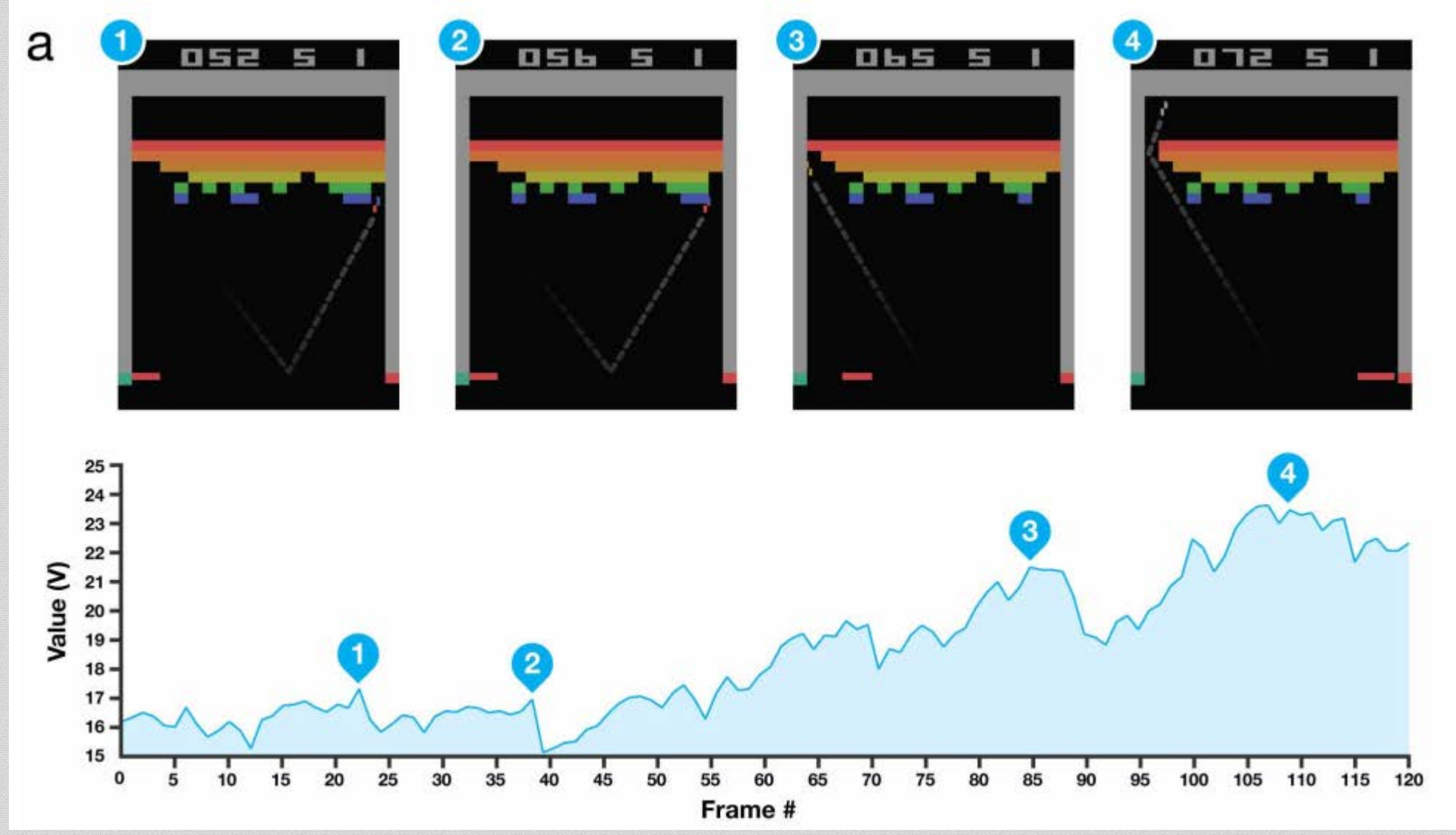


机器VS人

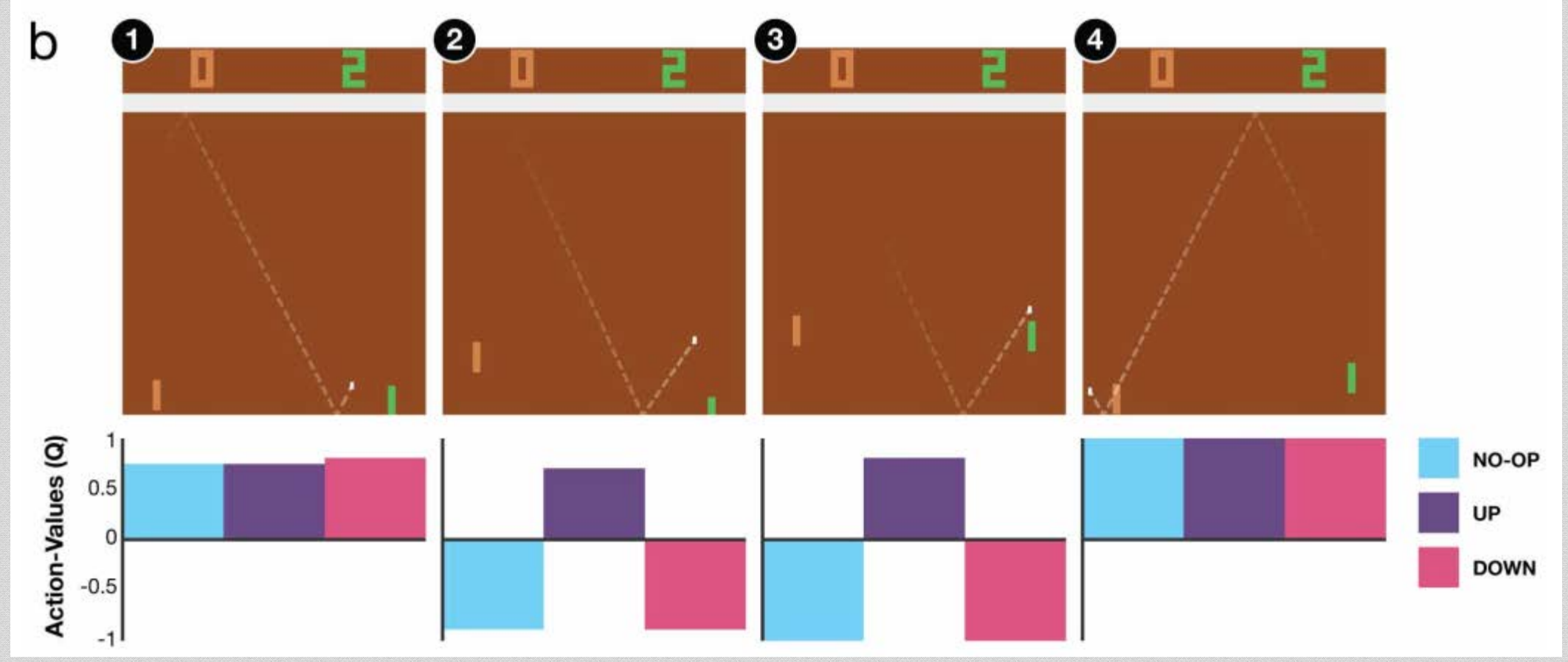
Game	DQN	Linear
Breakout	316.8	3.00
Enduro	1006.3	62.0
River Raid	7446.6	2346.9
Seaquest	2894.4	656.9
Space Invaders	1088.9	301.3



案例分析


a
1
052 5 1
2
056 5 1
3
065 5 1
4
072 5 1
Value (V)
25
24
23
22
21
20
19
18
17
16
15
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
105
110
115
120
Frame #

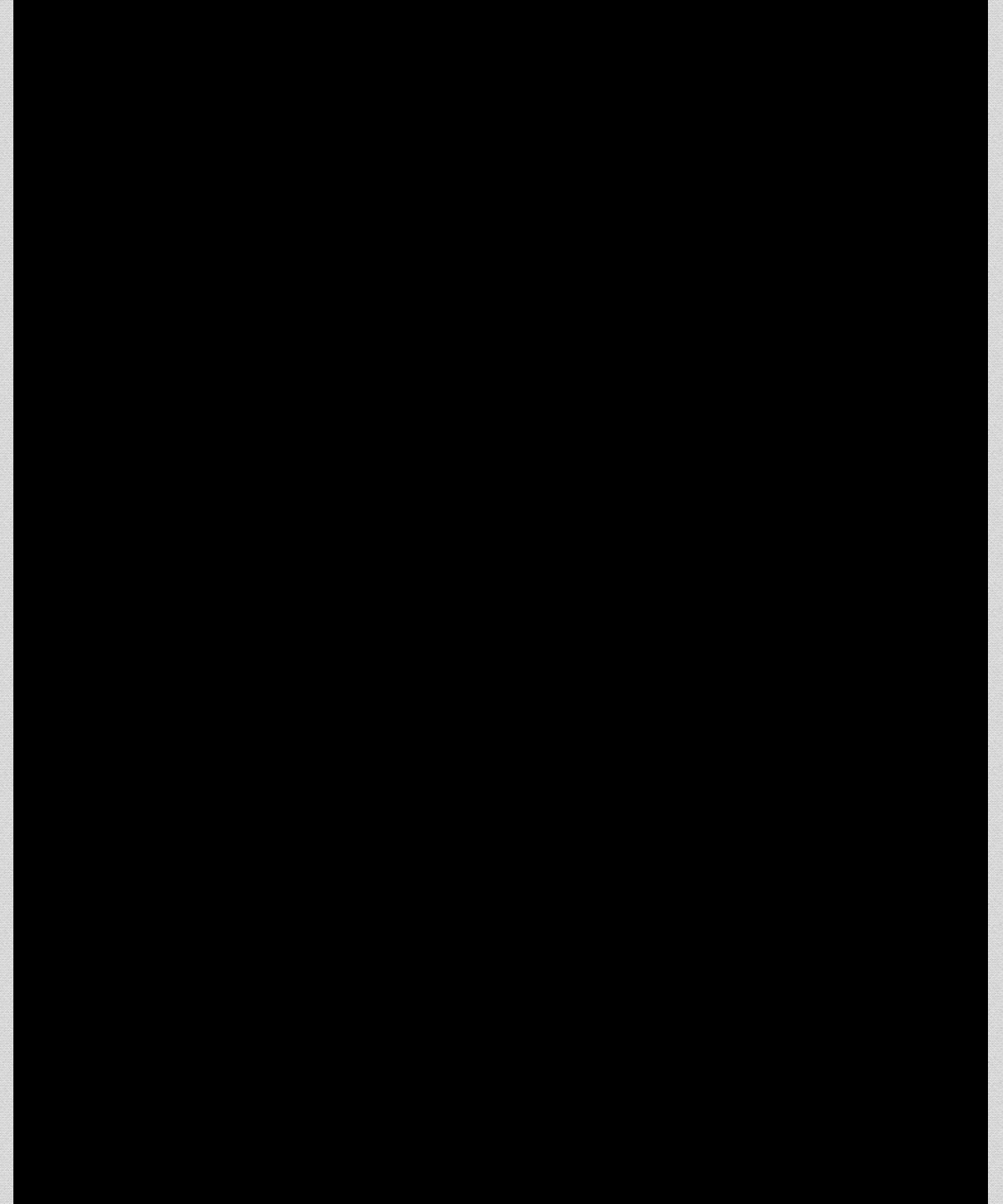
案例分析



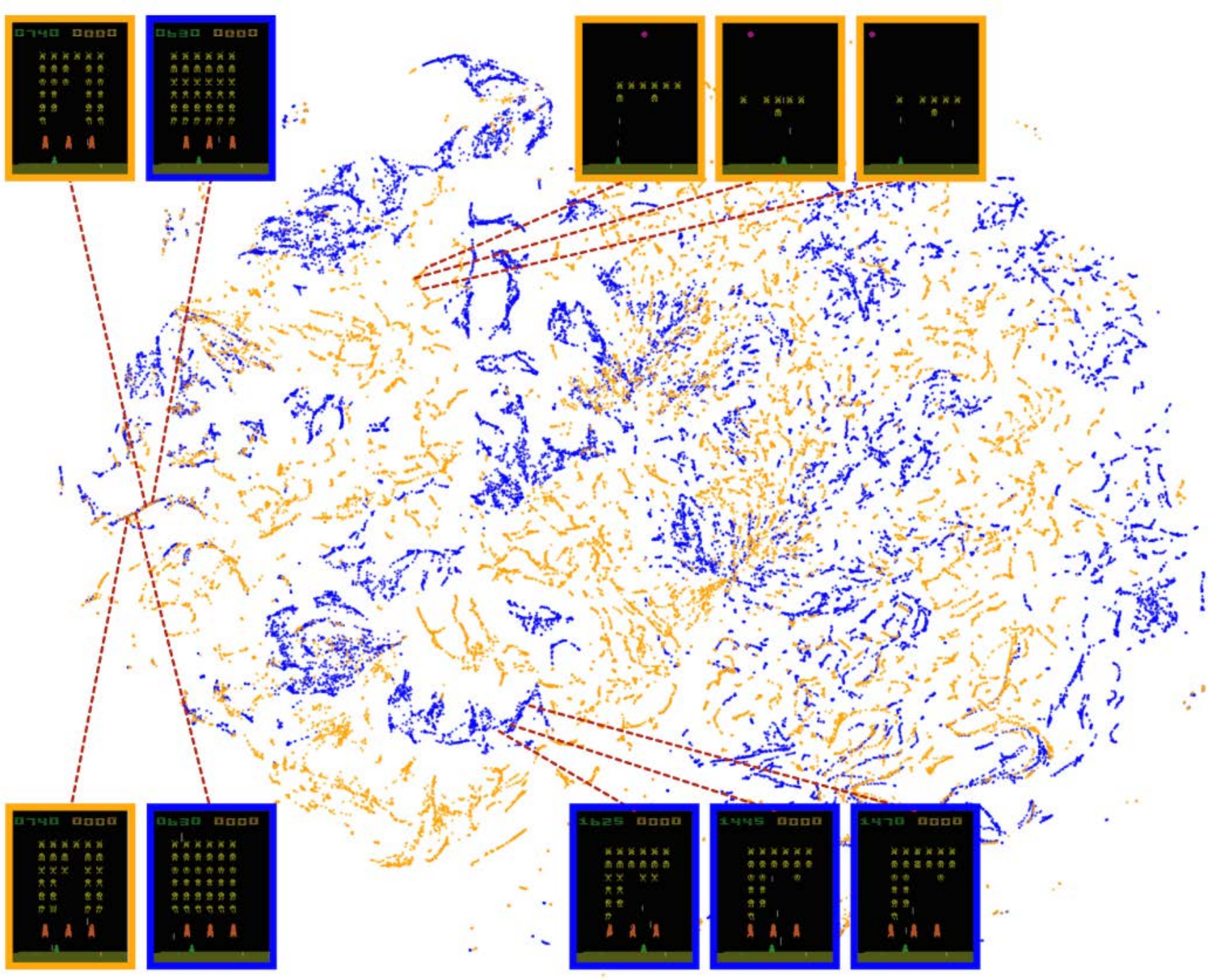
案例分析

Figure 3: The leftmost plot shows the predicted value function for a 30 frame segment of the game Seaquest. The three screenshots correspond to the frames labeled by A, B, and C respectively.

案例分析



隐含层的降维表示（可视化）



Part 4

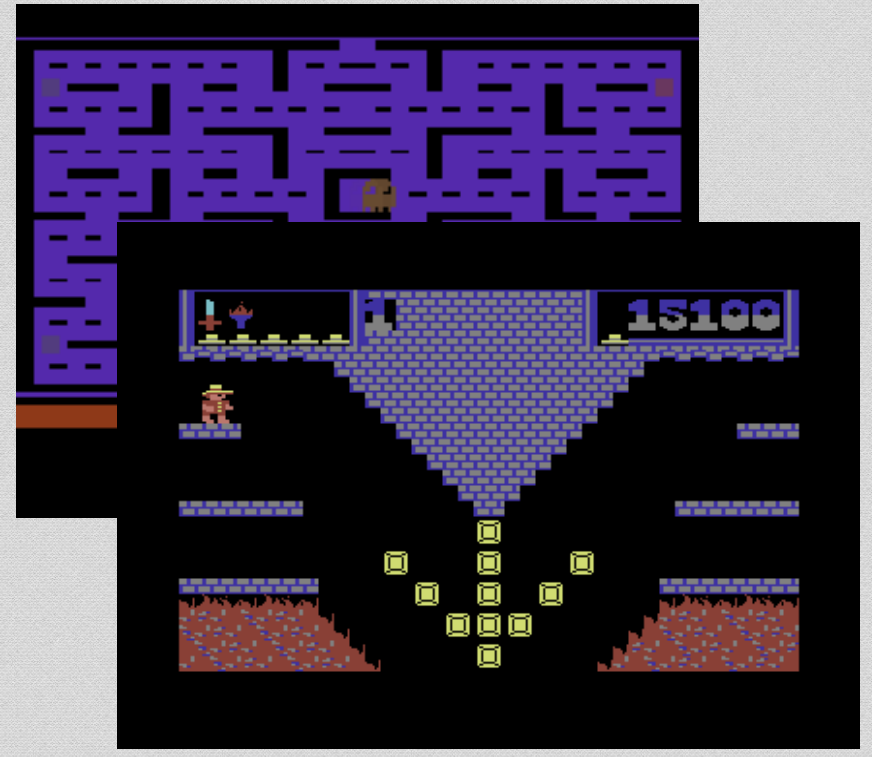
总结与展望

工作的不足

DQN擅长

DQN不擅长

temporally extended planning strategies



回顾与展望

回顾：

- 强化学习
  - 端到端的Q函数非线性近似
- 深度学习
  - 从原始的高维数据里提取出高质量的特征

展望：

- 对回放记忆的不均匀采样，寻找改变局势的显著部分，引入prioritized sweeping机制

目前深度增强学习在NLP领域的应用（2016）：

- （机器翻译NIPS2016）Dual Learning for Machine Translation
- （自动问答EMNLP2016）Deep Reinforcement Learning for Dialogue Generation
- （文本生成NIPS2015）Generating Text with Deep Reinforcement Learning
- （自动问答NIPS2015） Strategic Dialogue Management via Deep Reinforcement Learning
- （自动问答）End-to-end LSTM-based dialog control optimized with supervised and reinforcement learning
- Etc.

Reinforcement Learning + Deep Learning = **AI**

David Silver

Thank You