

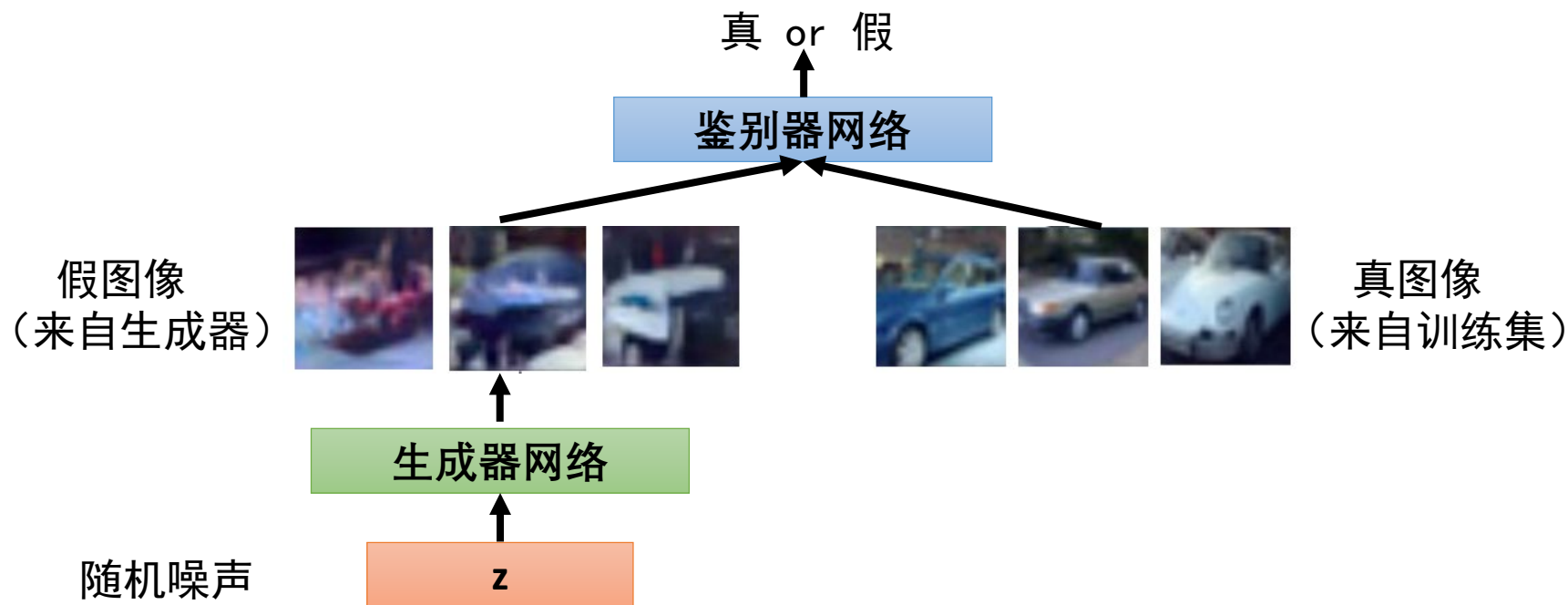
上节课回顾

- RNN的应用
- 生成式对抗网络及应用

训练GAN：双人游戏

生成器网络 (Generator)：尝试通过生成逼真的图像来欺骗鉴别器

鉴别器/判别器 (Discriminator) 网络：尝试区分真实和虚假的图像



算法

- 初始化判别器参数 θ_d 和生成器参数 θ_g
- 在每个训练的迭代中:

训练
D

- 从数据集中取样m个样本 $\{x^1, x^2, \dots, x^m\}$
- 从噪声先验分布中取样m个噪声样本 $\{z^1, z^2, \dots, z^m\}$
- 得到生成的样本 $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$, $\tilde{x}^i = G(z^i)$
- 更新判别器参数 θ_d 来最大化 \tilde{V}
 - $\tilde{V} = \frac{1}{m} \sum_{i=1}^m \log D_{\theta_d}(x^i) + \frac{1}{m} \sum_{i=1}^m \log (1 - D_{\theta_d}(G_{\theta_g}(z^i)))$
 - $\theta_d \leftarrow \theta_d + \eta \nabla \tilde{V}(\theta_d)$

训练
G

- 从噪声先验分布中取样m个噪声样本 $\{z^1, z^2, \dots, z^m\}$
- 更新生成器参数 θ_g 来最小化 \tilde{V}
 - $\tilde{V} = \frac{1}{m} \sum_{i=1}^m \log (D_{\theta_d}(G_{\theta_g}(z^i)))$
 - $\theta_g \leftarrow \theta_g + \eta \nabla \tilde{V}(\theta_g)$

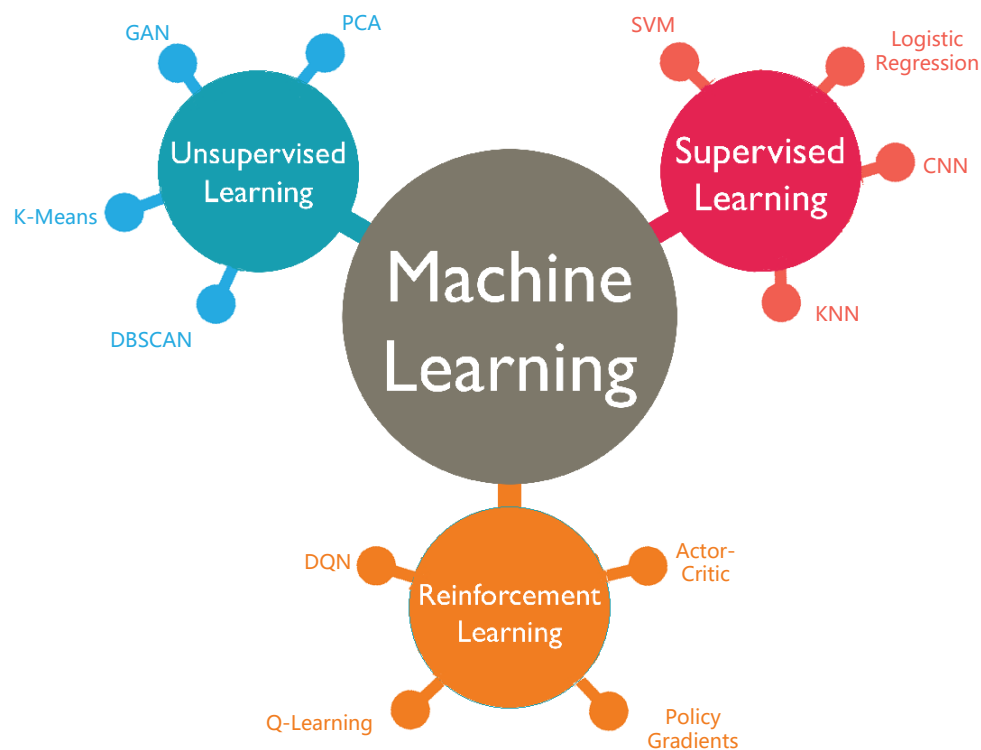
强化学习 (Reinforcement Learning)

1. 强化学习的背景简介
2. 强化学习的定义与方法介绍
3. Q-Learning算法介绍
4. Deep Q-Network (DQN) 方法

01

强化学习的背景

机器学习分类



Reinforcement Learning

强化学习需要大量的数据，因此它经常与有模拟数据的领域（游戏、机器人等）相关联。把研究论文的结果应用到实际应用中并不容易。即使对于强化学习研究人员来说，复制别人的研究结果也是一个挑战，尽管如此，依赖于强化学习的应用和产品已经出现。

- 机器人和工业自动化；
- 数据科学和机器学习；
- 文字，语音和对话系统；
- 医学和健康。

Reinforcement Learning

AlphaGo: 训练模型进行围棋对抗，首先要考虑围棋有数量难以估计的状态，为了克服这个问题，AlphaGo使用**强化学习**对**策略网络**进行学习，改善策略网络的性能，进一步训练价值网络，是模型能得到最佳的落子点。



Reinforcement Learning



机械臂操作：训练机械臂准确对特定的物体执行某项操作。

飞行器控制：使飞行器能自己调整角度、速度、高度保持飞行状态。

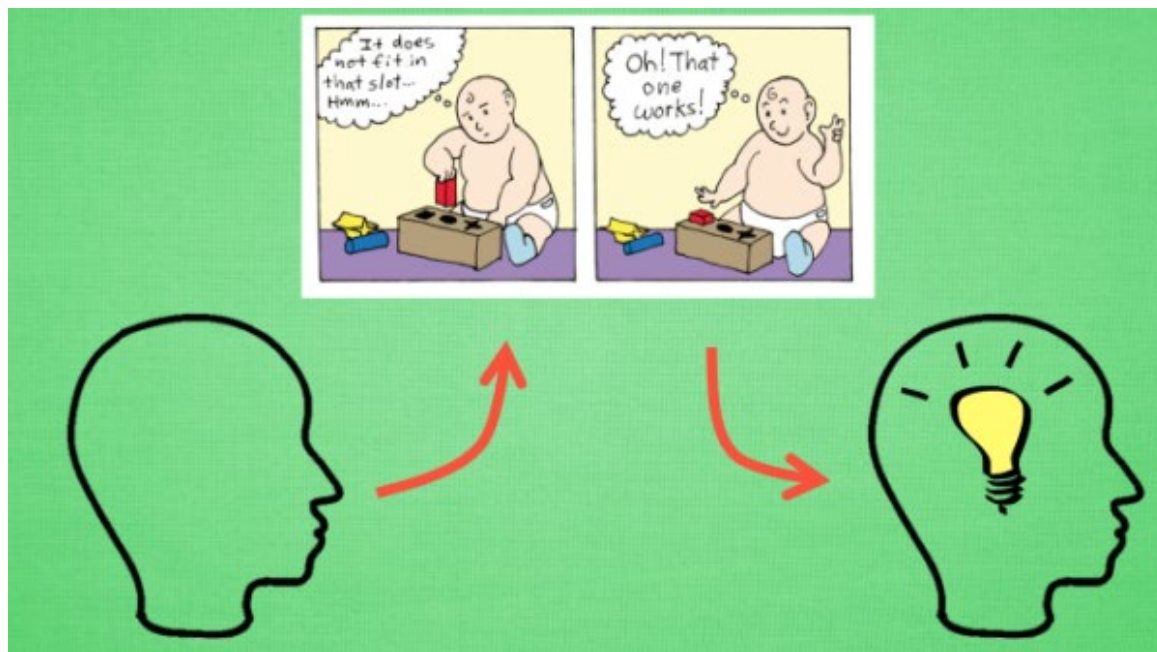
自动驾驶：可以训练一种自适应的系统来应对驾驶环境中出现的各种突发情况。

02

强化学习的定义与方法介绍

Reinforcement Learning

强化学习 又称再励学习、评价学习。是机器学习中的一个大大类，使用强化学习能够让参与者学会如何在环境中不断尝试，拿到高分，表现出优秀的成绩。



Reinforcement Learning

强化学习不同于监督和非监督学习

监督学习： 输入的数据有对应的正确标签，比如右图（积极的表情为高分，反之低分）。

无监督学习： 输入的数据没有相应标签。



强化学习： 没有输入数据和标签，需要通过在环境中不断尝试，获取数据及标签，再学习哪些数据对应哪些标签，通过学习到的这些规律，尽可能地选择带来高分的行为。

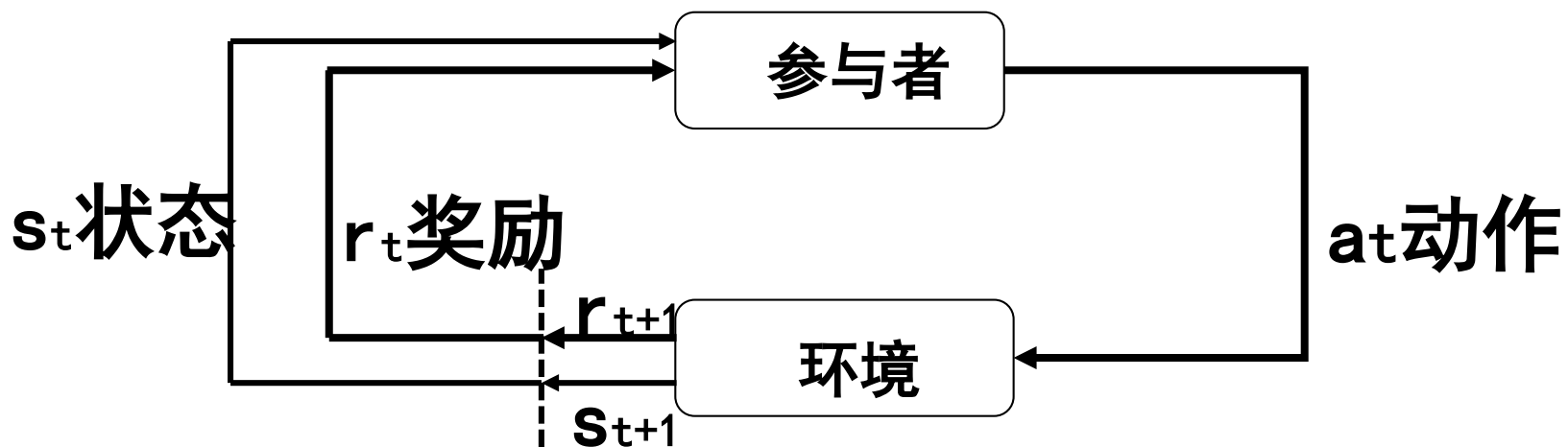
Reinforcement Learning

强化学习主要包含五个元素，**参与者 (agent)**，**环境 (environment)**、**状态 (state)**，**动作/行动 (action)**，**奖励 (reward)**。目标就是获得最多的累计奖励。

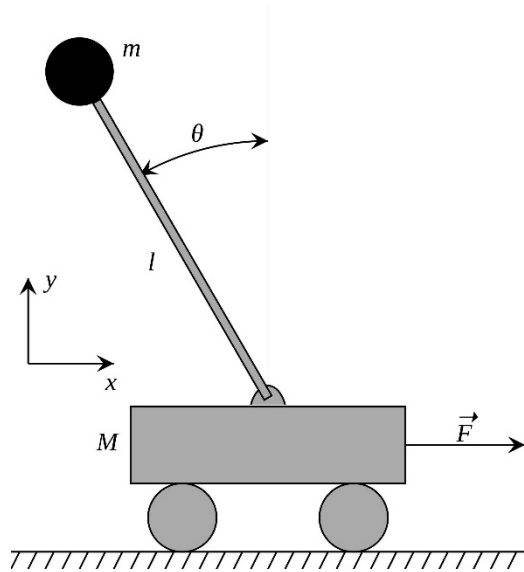
- **环境状态**：参与者对环境的感知，所有可能的状态称为状态空间；
- **行动空间**：参与者所采取的动作，所有动作构成动作空间；
- **转移概率**：当执行某个动作后，当前状态会以某种概率转移到另一个状态；
- **奖励函数**：在状态转移的同时，环境反馈给参与者一个奖赏。

Reinforcement Learning

各个要素之间的联系



车-杆平衡



目的 (Objective) : 控制小车使杆能够保持直立

状态 (State) : 杆的角度和角速度、车的水平速度

动作 (Action) : 作用在车上的水平力

奖励 (Reward) : 每一时刻杆能保持直立奖励1

Arari 游戏



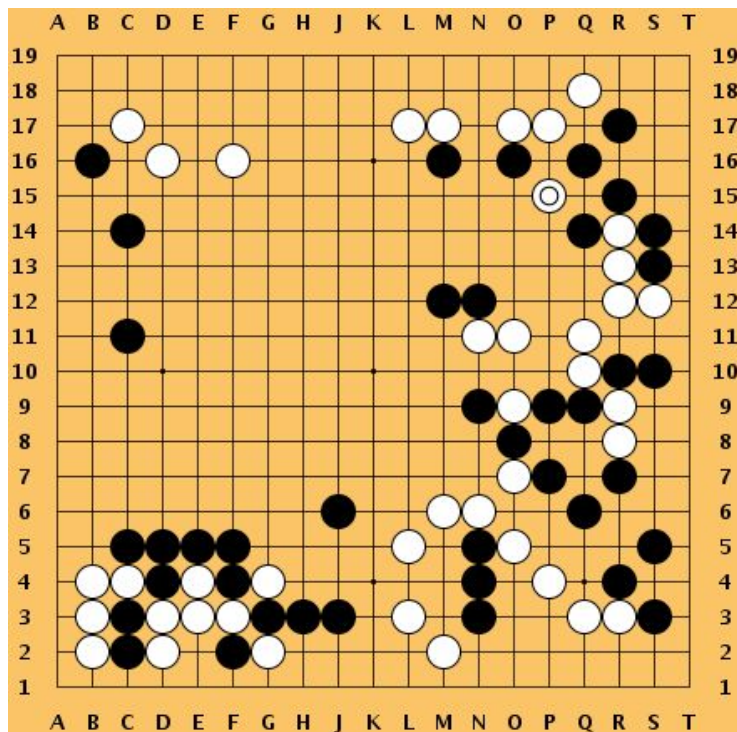
目的 (Objective) : 完成游戏并获得最高分数

状态 (State) : 当前游戏画面的图像状态

动作 (Action) : 游戏控制输入, 例如: 上、下、左、
右的动作

奖励 (Reward) : 每一步的游戏得分

围棋



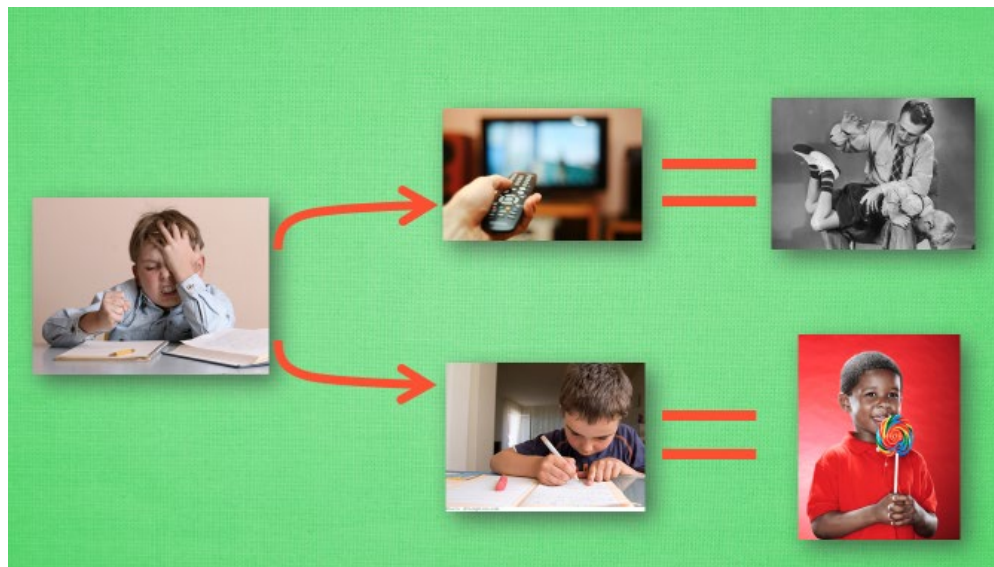
目的 (Objective) : 赢得胜利

状态 (State) : 所有棋子的位置

动作 (Action) : 下一个棋子在哪落位

奖励 (Reward) : 赢得游戏得到1的奖励, 反之奖励为0

Reinforcement Learning



一个**参与者**：正写作业的小孩子；

两种**状态**：写作业，看电视；

两个**行为**：去看电视，继续写作业；

两种**奖励**：糖或者受训。

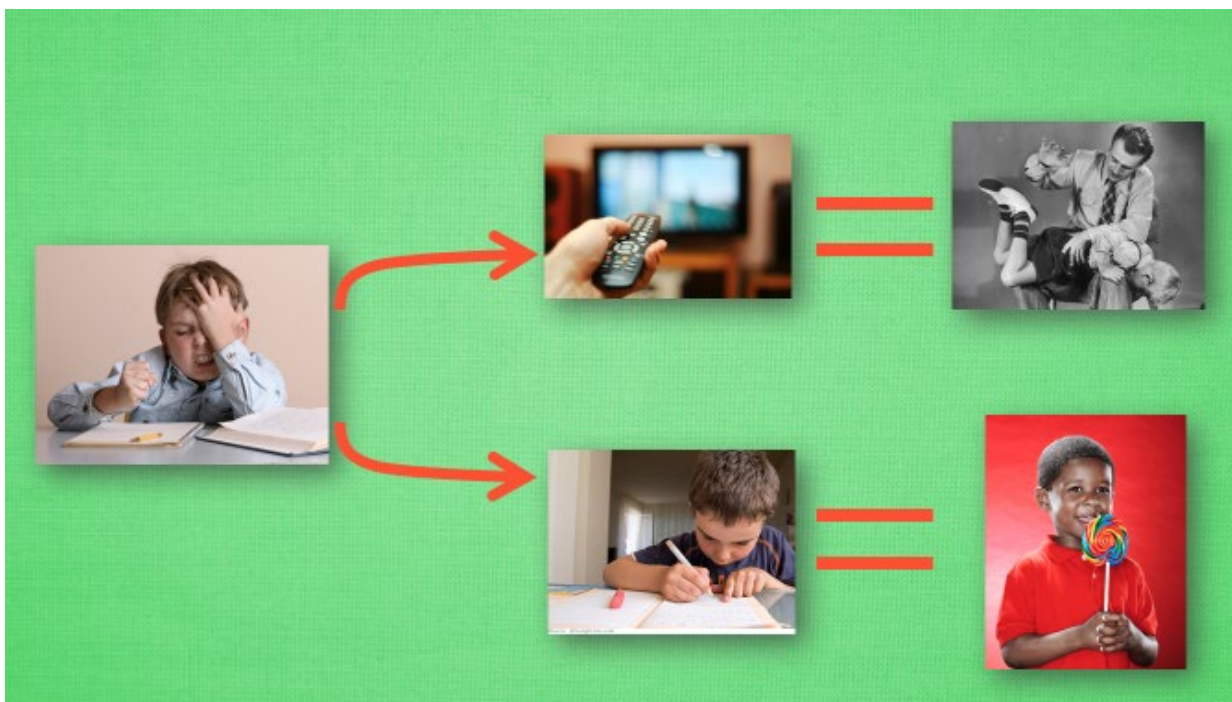
03

Q-Learning算法介绍

Reinforcement Learning

接下来介绍强化学习中一个经典算法Q-Learning。

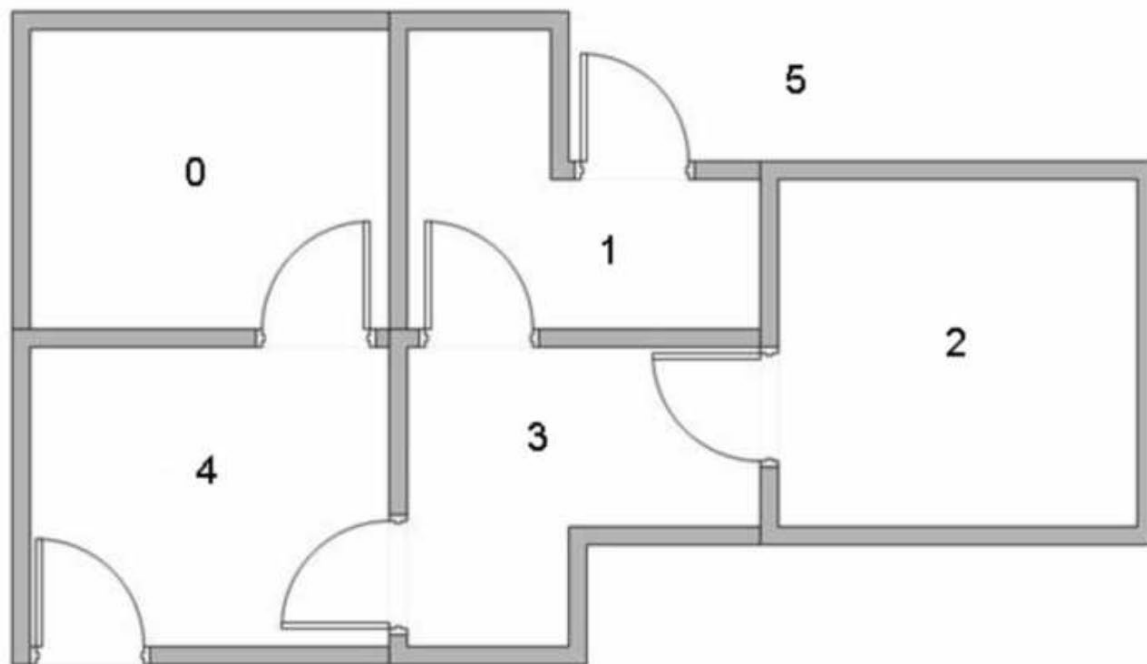
Q-Learning算法可根据既定的**行为准则**进行一个**决策**过程。



Reinforcement Learning

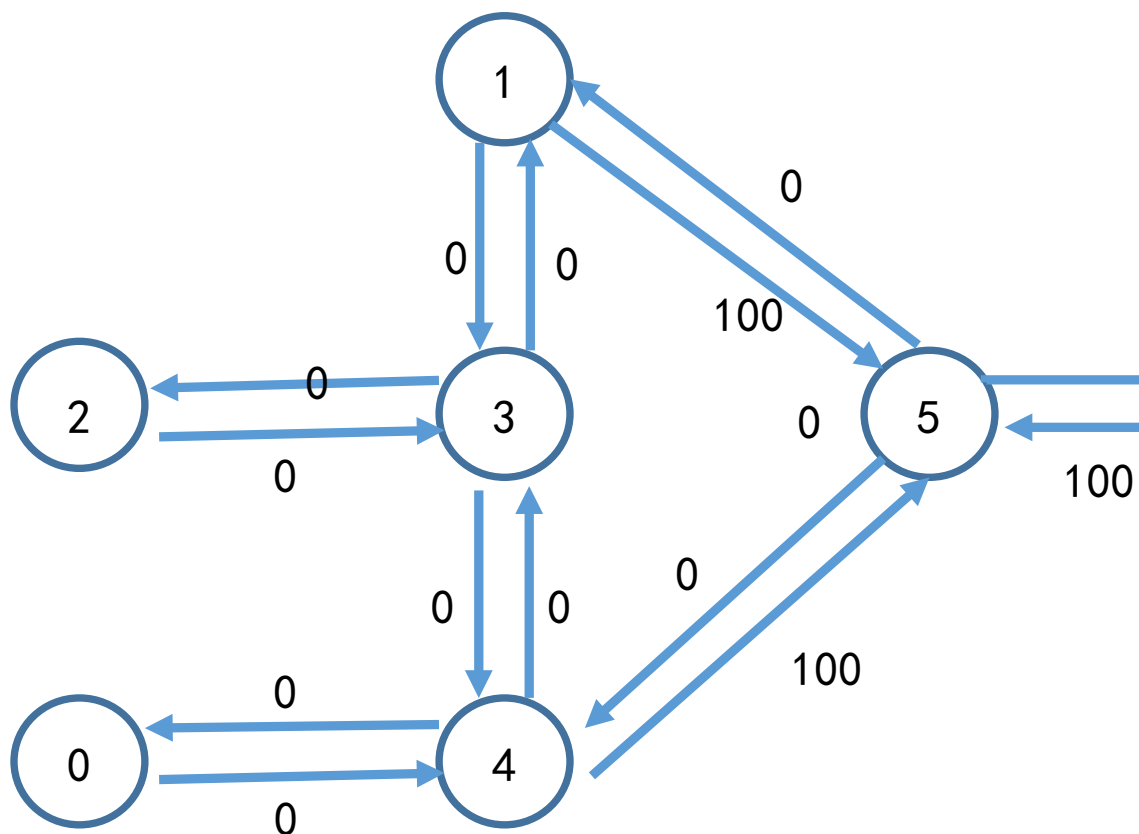
Q-Learning举例：

假设有一个这样的房间，参与者有个初始位置（在某个房间内），最终目的是需要走到房间5。

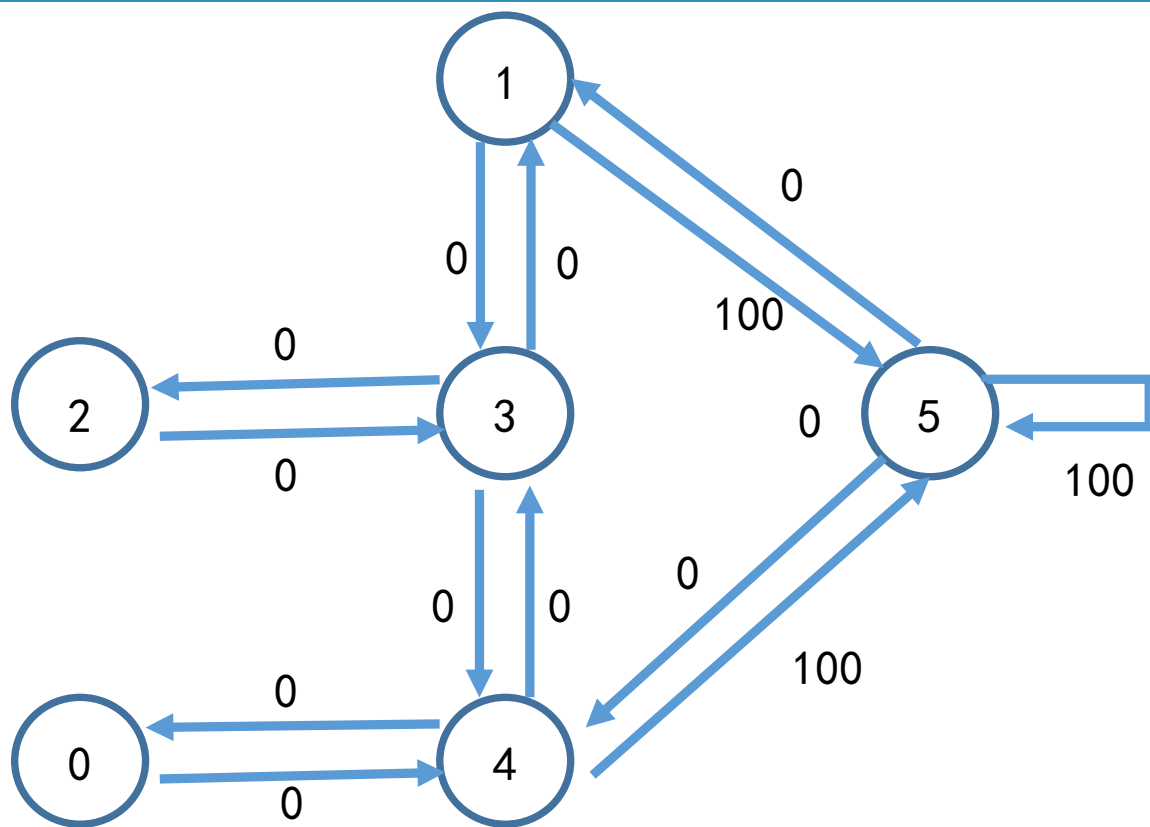


Reinforcement Learning

如果将房间表示成点，然后用房间之间的连通关系表示成线，将每个节点之间设置一定的权重，能够直接到达5（终点）的边设置为100，其他不能的设置0。如下图所示：



Reinforcement Learning



Q-Learning 中，最重要的就是“状态”和“动作”：

状态表示处于图中的某个节点，比如2节点，3节点等等。

动作是从一个节点到另一个节点的操作。

将参与者设置在任何一个位置，让他自己走动，直到走到5房间，表示成功。

Reinforcement Learning

首先我们生成一个奖励矩阵(R表)，矩阵中，-1表示不可以通过，0表示可以通过，100表示直接到达终点：

| | | Action | | | | | |
|-------|---|--------|----|----|----|----|-----|
| State | | 0 | 1 | 2 | 3 | 4 | 5 |
| R= | 0 | -1 | -1 | -1 | -1 | 0 | -1 |
| | 1 | -1 | -1 | -1 | 0 | -1 | 100 |
| | 2 | -1 | -1 | -1 | 0 | -1 | -1 |
| | 3 | -1 | 0 | 0 | -1 | 0 | -1 |
| | 4 | 0 | -1 | -1 | 0 | -1 | 100 |
| | 5 | -1 | 0 | -1 | -1 | 0 | 100 |

Reinforcement Learning

同时，我们创建一个Q表(Q-table)，表示学习到的经验，与R表同阶，初始化为0矩阵，表示从一个state到另一个state能获得的总奖励。

Q表示的是，在状态s下采取动作a能够获得的期望最大奖励，R是立即获得的收益，而未来一期的收益则取决于下一阶段的动作。

[illegible]

Reinforcement Learning

当前Q值计算：

$$Q(s, a) = [R(s, a) + \gamma \max_{a'} Q(s', a')]$$

$Q(s, a)$ 为在s状态下采取a动作对应Q表中的值。

$Q(s', a')$ 为s状态下采取a动作后的下一个状态为 s' ，下一个状态可采取的行动为 a' 。


γ 衰减因子， $0 < \gamma < 1$ ，一般设置为0.8。


Reinforcement Learning


Q-Learning中的Gamma（衰减因子）。使参与者在选择动作时不仅可以考虑到“眼前”的奖励，也可以考虑到“未来”的奖励。

$Q(s_1) = r_2 + \gamma Q(s_2) = r_2 + \gamma [r_3 + \gamma Q(s_3)] = r_2 + \gamma [r_3 + \gamma [r_4 + \gamma Q(s_4)]] = \dots$

$Q(s_1) = r_2 + \gamma r_3 + \gamma^2 r_4 + \gamma^3 r_5 + \gamma^4 r_6 + \dots$

$\gamma = 1$  $Q(s_1) = r_2 + 1 * r_3 + 1 * r_4 + 1 * r_5 + 1 * r_6 + \dots$

$\gamma = (0 \sim 1)$  $Q(s_1) = r_2 + \gamma r_3 + \gamma^2 r_4 + \gamma^3 r_5 + \gamma^4 r_6 + \dots$

$\gamma = 0$  $Q(s_1) = r_2$

Reinforcement Learning

$\epsilon - greedy$ (Epsilon贪婪) 策略:

选取动作是**基于当前所处状态下所有动作的Q值**进行判断。那在Q表初始化为0的情况下我们将使用 $\epsilon - greedy$ 策略，就是以 ϵ 概率随机选取行动。

- 开始阶段，**探索 (Exploration)** 速率 ϵ 设定为最大值1，**目的是找到更多关于环境的信息。**
- 生成一个随机数。如果这个数大于 ϵ ，那么将会进行**开发 (Exploitation)** 操作，即利用当前Q表选择动作，**目的是利用已知信息来得到最多的奖励。**
- 在刚开始学习Q 函数时，必须有一个大的 ϵ 。随着参与者对估算出的 Q 值更有把握，将**逐渐减小 ϵ** 。

Reinforcement Learning

根据 $\epsilon - \text{greedy}$ 策略，随机选择一个状态，比如1，查看状态1所对应的R表，也就是1可以到达3或5，随机地，我们选择5，根据转移方程（ $\gamma = 0.8$ ）：

$$\text{NewQ}(s, a) = [R(s, a) + \gamma \max_{a'} Q(s', a')]$$

$$\text{NewQ}(1, 5) = R(1, 5) + 0.8 * \max\{Q(5, 1), Q(5, 4), Q(5, 5)\}$$

$$= 100 + 0.8 * \max\{0, 0, 0\}$$

$$= 100$$

Reinforcement Learning

可得到Q表为：

$$Q = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 100 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

到达目标，一次尝试结束。

Reinforcement Learning

接下来再选择一个随机状态，比如3。状态3对应的下一个状态有1, 2, 4, 并且都是状态3对应的非负状态。随机地，我们选择1，这样根据算法更新：

$$\text{New}Q(3, 1) = R(3, 1) + 0.8 * \max\{Q(1, 3), Q(1, 5)\}$$

$$\begin{aligned} &= 0 + 0.8 * \max\{0, 100\} \\ &= 80 \end{aligned}$$

Reinforcement Learning

更新Q表：

$$Q = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 100 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 80 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Reinforcement Learning

经过1000次迭代，最后得到的Q表为：

```
[[ 0. 0. 0. 0. 124.44939 0. ]  
 [ 0. 0. 0. 119.41786 0. 124.98187 ]  
 [ 0. 0. 0. 121.78858 0. 0. ]  
 [ 0. 123.098785 112.5605 0. 123.30301 0. ]  
 [122.91019 0. 0. 113.726875 0. 124.71593 ]  
 [ 0. 0. 0. 0. 0. 0. ]]
```

部分测试结果：

```
Number:5Testing  
The agent at 3  
the robot goes to 4.  
the robot goes to 5.  
Number:6Testing  
The agent at 1  
the robot goes to 5.  
Number:7Testing  
The agent at 4  
the robot goes to 5.
```

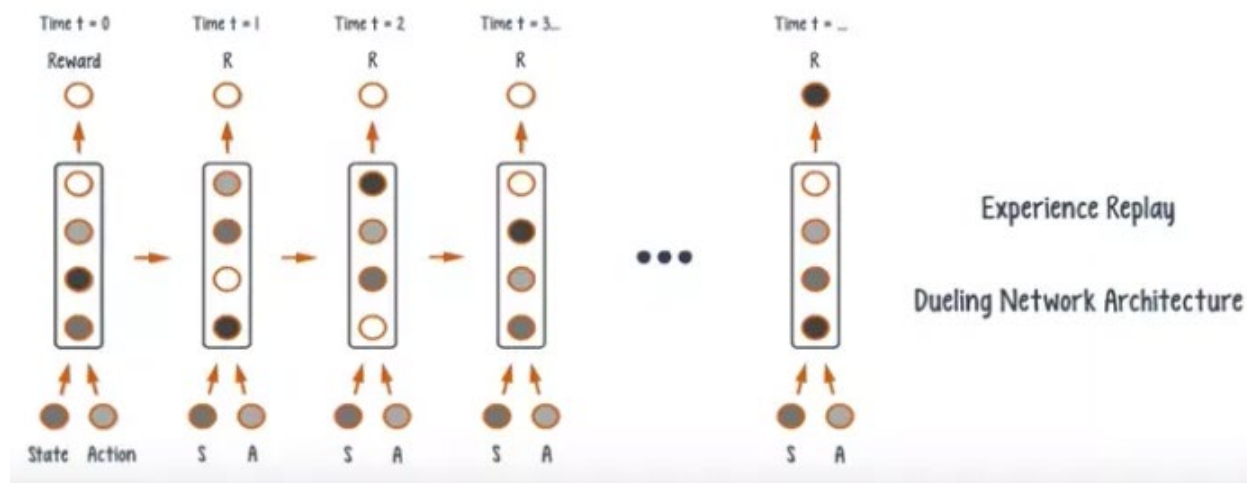
04

DQN方法

Reinforcement Learning – DQN

目前，在很多环境下往往有很多状态。例如围棋的状态总数是 (3^{19}) ，这样Q-Learning要计算更新的Q表大小就难以估量，根本不可行。这时候就需要DQN（Deep Q-Network）。实际上它就是Q-Learning和神经网络的结合，将Q-Learning的Q表变成了Q-Network。

Deep Q Net



Reinforcement Learning – DQN

DQN对状态的维度进行压缩的办法就是-价值函数近似（Value Function Approximation）。

价值函数近似就是用一个函数来表示 $Q(s, a)$ ，即：

$$Q(s, a) = f(s, a)$$

f 可以是任意类型的函数，比如线性函数：

$$Q(s, a) = \omega_1 + \omega_2 a + b, \text{ 其中 } \omega_1, \omega_2, b \text{ 是函数 } f \text{ 的参数}$$

因此可以解决状态维度过高的问题，最后都可以通过矩阵运算降维输出为单值的 Q 。

如果我们就用 w 来表示统一参数：

$$Q(s, a) = f(s, a, \omega)$$

称之为近似原因是因为我们并不知道 Q 值的实际分布情况，本质上就是用一个函数来近似 Q 值的分布，所以，更准确的表达是：

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

Reinforcement Learning – DQN

DQN作为基于深度网络的强化学习方法，确定合适的损失函数就非常重要。

Q-Learning算法：

$R_t + \gamma \max Q(s', a', \omega)$,其中 s' 为下一个状态， a' 为下一个动作。

因此，我们把Q目标Q值作为标签不就可以了？我们的目标不就是让Q值趋近于目标Q值吗？

所以，Q网络训练的损失函数就是：

$$L(\omega) = E[(R_t + \gamma \max Q(s', a', \omega) - Q(s, a, \omega))^2]$$

DQN的优缺点：

优点：

- 算法通用性强，可玩不同游戏；
- End-to-End 训练方式；
- 可生产大量样本供学习。

缺点：

- 无法应用于连续动作控制；
- 只能处理短时记忆问题；
- 无法处理需长时记忆问题（后续研究提出了使用LSTM等改进方法）；
- CNN难收敛，需精细调参。