

Lecture 4: 强化学习

强化学习基础

强化学习和监督学习的区别

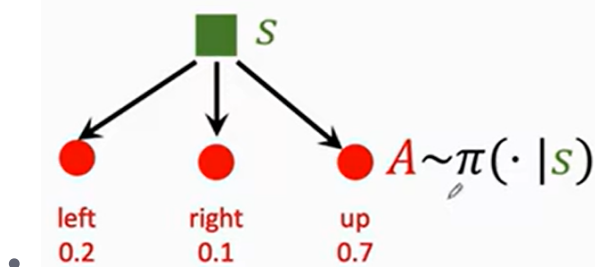
概念：强化学习和监督学习的区别

- (1) 训练数据中没有标签，只有奖励函数 (Reward Function)
- (2) 训练数据不是现成给定，而是由行为 (Action) 获得；
 - 比如下棋的过程当中，训练的数据是棋盘上的棋子，即整个环境当中的动作决定的，而不是一开始就确定的；
- (3) 现在的行为 (Action) 不仅影响后续训练数据的获得，也影响奖励函数 (Reward Function) 的取值
 - 比如在围棋中，走好的棋和不好的棋，最后训练数据的活动会不一样；
- (4) 训练的目的是构建一个“状态→行为”的函数，其中状态 (State) 描述了目前内部和外部的环境，在此情况下，要使一个智能体 (Agent) 在某个特定的状态下，通过这个函数，决定此时应该采取的行为。希望采取这些行为后，最终获得最大的奖励函数值；
 - 所有的内部和外部的环境，构成了状态；

强化学习基础概念与假设

概念：强化学习基础概念

- R_t : t 时刻的奖励函数值
 - 在围棋中，它每一个时刻都是为 0，只有在游戏结束的时候、才会返回 1 或者 0；
 - 比如在游戏中，获得一个金币， $R = +1$ ，赢得游戏， $R = +1000$
- S_t : t 时刻的状态
 - 比如：某一个时刻，当前围棋的棋盘的状态，以及你是执黑子还是白子；
- A_t : t 时刻的行为
 - 根据一个 S_t 到 A_t 的映射，产生的哪一个行为；
 - 其中，Action 是由策略控制得到的。当给定一个状态 s 时，行为可以被随机：



e.g., .

- $\pi(\text{"left"}|s) = 0.2,$
- $\pi(\text{"right"}|s) = 0.1,$
- $\pi(\text{"up"}|s) = 0.7.$

- **policy** : 策略 π

$$\pi(a | s) = \mathbb{P}(A = a | S = s).$$

- 目的：

- 强化学习的目的，就是学习 $\pi(a|s)_\rho$ 或者是 $Q^*(s, a)$.

概念：三个基本假设

- 假设：马尔科夫假设

- 在 $t+1$ 时刻的状态，只和 t 时刻的状态有关，和其他时刻无关；

$$\mathbb{P}[S_{t+1} | S_t] = \mathbb{P}[S_{t+1} | S_1, \dots, S_t]$$

- 假设：下一个时刻的状态，只和这一个时刻的状态，以及这一个时刻的行为有关；

$$P_{ss'}^a = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

- 但是，如果是在自动驾驶当中，下一个时刻的状态，除了和自己这个时刻的内部状态有关，还可能和外部的环境有关；

- 假设：下一个时刻的奖励函数值，只与这一时刻的状态及这一时刻的行为有关

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

补充：一个常用的强化学习库 - Gym

- Gym is a toolkit for developing and comparing reinforcement learning algorithms.

马尔可夫过程

概念：MARKOV DECISION PROCESS - MDP

- (1) 在 $t=0$ 时候，环境给出一个初始状态 $s_0 \sim p(s_0)$
 - 比如围棋的空白棋盘；
 - 但是如果是无人驾驶时，这里的初始状态可能不是 0，而是需要一些初始的设置；
- (2) for $t=0$: end
 - 智能体选择行为 a_t
 - 根据已有状态选择行为；
 - 环境采样奖励函数 $r_t \sim R(\cdot | s_t, a_t)$
 - 有了 s 有了 a ，需要给出奖励函数；
 - 环境产生下一个状态： $s_{t+1} \sim P(\cdot | s_t, a_t)$
 - 环境以一定的概率、产生下一个状态；
 - 但是在无人驾驶时，可能会有不一样的情况；
 - 智能体获得奖励函数 r_t 和下一个状态 s_{t+1}
- 我们需要学习一个策略 (Policy): π^* ;
- 这是一个从状态到行为的映射函数，并且最大化累积的奖励；

概念：待优化目标函数 - Return

- 概念：Return

- 将之后的奖励加在一起；

$$U_t = R_t + R_{t+1} + R_{t+2} + R_{t+3} + \dots$$

- 概念：折扣回报 - Discounted return

- 增强学习中的待优化目标函数是累积奖励，即一段时间内的奖励函数加权平均值，以下是 Discounted return：

- $$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- 在这里，GAMMA 是一个衰减项；

- 解释：

- 需要评估未来状态的奖励，并且为了让最近的状态所带来的影响最大，因此使用加权平均值、保证最近的状态权值更大，但同时后续的步骤也有影响；
- 并且由于马尔可夫过程的存在，因此在当前 t 时刻，就可以计算出后续若干时刻的奖励函数；

- 概念：return 的随机性

- 在某一状态 t，当前的回报是随机的，随机性有两个来源：
 - 1. 行为可以被随机： $\mathbb{P}[A = a | S = s] = \pi(a|s)$
 - 2. 新的状态可以被随机： $\mathbb{P}[S' = s' | S = s, A = a] = p(s'|s, a)$
- 而 U_t 随机性的来源，就是未来的所有行为以及状态的随机性；
- 由于 U_t 是一个随机变量，并不能评估当前状态下的局势，因此需要定义估值函数来判断；

估值函数与 Q-learning 函数

概念：估值函数与 Q-learning 函数

- 补充：路径

- 根据一个决策机制 (Policy)，我们可以获得一条路径：

- $$s_0, a_0, r_0, s_1, a_1, r_1, \dots$$

- 定义：Q 函数 - 动作价值函数

- 概念：

- Q 函数是衡量某个状态下采取某个行为后，最终能获得多少累积奖励的函数；

- $$Q^{\pi}(s, a) = Q_{\pi}(s_t, a_t) = \mathbb{E}[U_t | S_t = s_t, A_t = a_t] = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

- 即：Q 是 s 确定、a 确定下（行为确定），算出来的估计值；

- 含义：

- Q 函数告诉了我们，在给定策略 π 的情况下，在状态 s 的情况下，做动作 a 的行为是好还是坏，因此 Q 函数可以给当前状态 s 下的所有行为 a 打分，选择最好的行为；

- 定义：最优动作价值函数 - Optimal action-value function

- $$Q^*(s_t, a_t) = \max_{\pi} Q_{\pi}(s_t, a_t).$$

- 定义：状态价值函数 - State-value function

- 状态价值函数 (Value Function) 是衡量某个状态最终能获得多少累积奖励的函数；

- $$V_{\pi}(s_t) = \mathbb{E}_A [Q_{\pi}(s_t, A)] = \sum_a \pi(a|s_t) \cdot Q_{\pi}(s_t, a).$$

- $$V_{\pi}(s_t) = \mathbb{E}_A [Q_{\pi}(s_t, A)] = \int \pi(a|s_t) \cdot Q_{\pi}(s_t, a) da$$

- 即：V 是只有 s 确定（状态确定）、a 不确定的情况下，算出来的估计值；
- 因此 V 函数是对所有的状态进行遍历，计算其在不同状态下的概率以及 Q，求和得到 V；
- 因此状态价值函数可以用于评估当前形势怎么样，是要赢了还是要输了；
- 并且 V 函数还可以用于评估策略函数，如果 V 越大，则策略越好；

分析：公式推导

- $$\begin{aligned} v^{\pi}(s) &= E_{\pi} \left[\sum_{t=0}^{+\infty} \gamma^t r_t \mid S_0 = S, \pi \right] \\ &= E_{\pi} \left[r_0 + \sum_{t=0}^{+\infty} \gamma^t r_{t+1} \mid S_0 = S, \pi \right] \\ &= \sum_{a \in A} \pi(S, a) \sum_{S' \in S} P_{SS'}^a (R_S^a + \gamma V^{\pi}(S')) \end{aligned}$$

其中 $\pi(S, a) = p(a|S)$

- $$\begin{aligned} Q^{\pi}(s, a) &= \sum_{S' \in S} P_{SS'}^a (R_S^a + \gamma V^{\pi}(S')) \\ \pi(S, a) &= \begin{cases} 1 : & \text{若 } a = \operatorname{argmax} Q(S, a) \\ 0 : & \text{其他} \end{cases} \\ \text{则有 } V^{\pi}(s) &= \sum_{a \in A} \pi(S, a) Q^{\pi}(s, a) \end{aligned}$$

强化学习代码示例

代码

```
import gym
env = gym.make('CartPole-v0')

state = env.reset() # 重置环境

for t in range(100):
    env.render() # 渲染
    print(state)

    action = env.action_space.sample() # 随机均匀抽样的选择动作
    state, reward, done, info = env.step(action) # 环境执行一次动作，其中done表示当
```

前游戏是否结束

```
if done:
    print('Finished')
    break
```

```
env.close()
```

价值学习

DQN - 神经网络近似 Q 函数

概念: DQN

- 引入: 为什么需要价值学习

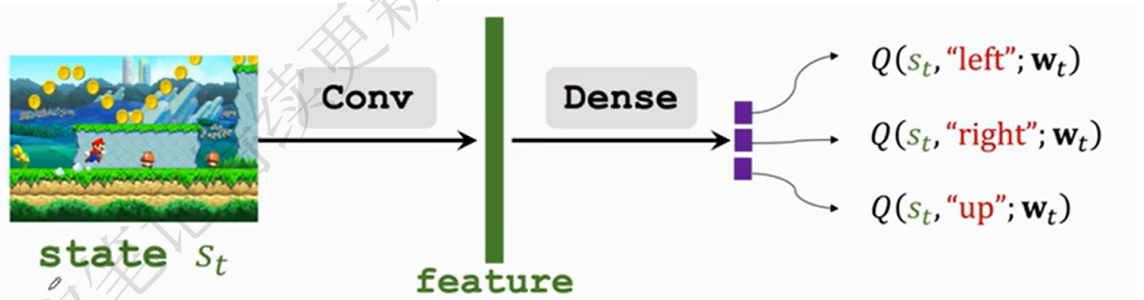
- 如果目标是赢得游戏;
- 则如果当前知道 Q^* 函数, 则最优的选择为

$$a^* = \underset{a}{\operatorname{argmax}} Q^*(s, a)$$

- 即: 根据当前选择的平均回报, 选择平均回报最大的那个动作, 真正的回报不一定是这个, 但还是应该选择平均的选择;
- 而价值学习的思路, 就是通过神经网络、学习出一个 $Q^*(s, a)$;
- DQN 利用神经网络, 学习出 $Q(s, a; W)$ 来近似 $Q^*(s, a)$;

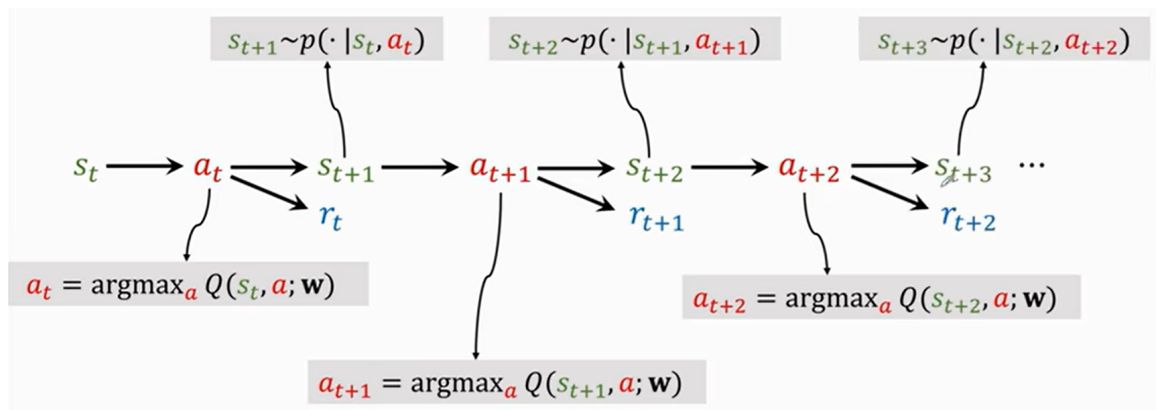
- 举例:

- 图示:



- 分析:

- 画面作为输入, 利用卷积将图片变为特征向量, 然后再使用全连接层将特征映射到一个输出的向量;
- 这个输出的向量就是打分, 每一个向量元素对应一个动作;
- 比如: 向左的分数为 2000, 向右为 3000 分, 则需要选择向右;



- 重复以上过程;

训练 DQN 算法 - TD 学习

概念: Temporal Difference (TD) Learning

• 引入:

- $T_{\text{NYC} \rightarrow \text{ATL}} \approx T_{\text{NYC} \rightarrow \text{DC}} + T_{\text{DC} \rightarrow \text{ATL}}$
- 其中左边是从纽约到亚特兰大的总时间，右边是两段分别的时间;
- 理想情况下，这公式的两端应当是相等的，如果计算的结果和实际的情况有偏差，就更新;

• 公式:

- $Q(s_t, a_t; \mathbf{w}) \approx r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w})$.
- 其中 $Q(s_t, a_t; \mathbf{w})$ 是 t 时刻估计的值;
- r_t 是奖励， $Q(s_{t+1}, a_{t+1}; \mathbf{w})$ 是在 $t+1$ 时刻的估计值;

• 分析:

- 由 return 的定义可知:

$$\begin{aligned}
 U_t &= R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \gamma^3 \cdot R_{t+3} + \gamma^4 \cdot R_{t+4} + \dots \\
 &= R_t + \gamma \cdot (R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \gamma^3 \cdot R_{t+4} + \dots) \\
 &= U_{t+1}
 \end{aligned}$$

- 因此，可以得到一个等式:

$$U_t = R_t + \gamma \cdot U_{t+1}.$$

- 这个等式反映了相邻两个折扣回报之间的关系;

• 概念: TD for DQN

- DQN 的输出 $Q(s_t, a_t; \mathbf{w})$ 是 $\mathbb{E}[U_t]$ 的估计;
- DQN 的输出 $Q(s_{t+1}, a_{t+1}; \mathbf{w})$ 是 $\mathbb{E}[U_{t+1}]$ 的估计;
- 因此得到:

$$\underbrace{Q(s_t, a_t; \mathbf{w})}_{\text{之前的预测}} \approx \underbrace{r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w})}_{\text{TD更加接近的新预测}} = y_t.$$

- 并且将两者之间的差值，定义为损失函数：

$$L_t = \frac{1}{2} [Q(s_t, a_t; \mathbf{w}) - y_t]^2$$

- 然后使用梯度下降，来更新参数 W ：

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial L_t}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}.$$

- 做梯度下降的目的，就是为了使得 **Loss** 变小；

TD 算法过程

概念： TD 算法过程

- (1) 观察当前的状态 S_t 和动作 a_t
- (2) 用 DQN 进行一次计算，预测价值，把输出作为 q_t ，即： $q_t = Q(s_t, a_t; \mathbf{w}_t)$
- (3) 用反向传播对神经网络求导，计算其梯度：

$$\mathbf{d}_t = \frac{\partial Q(s_t, a_t; \mathbf{w})}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$$

- (4) 环境更新状态 S_{t+1} 并给出奖励 r_t ；
- (5) 计算出 **TD** 目标值：

$$y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}_t).$$

- (6) 做一次梯度下降：

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot (q_t - y_t) \cdot \mathbf{d}_t.$$