

Deep Reinforcement Learning

[深度强化学习基础](#)

Terminologies

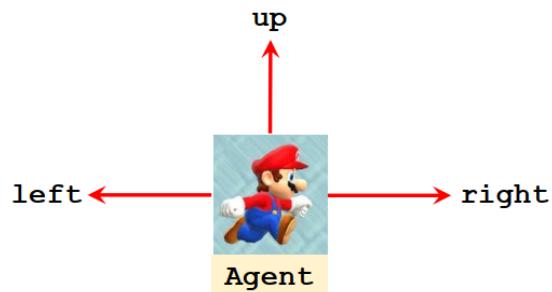
State and Action

Terminology: state and action

state s (this frame)



Action $a \in \{\text{left}, \text{right}, \text{up}\}$



state是场景

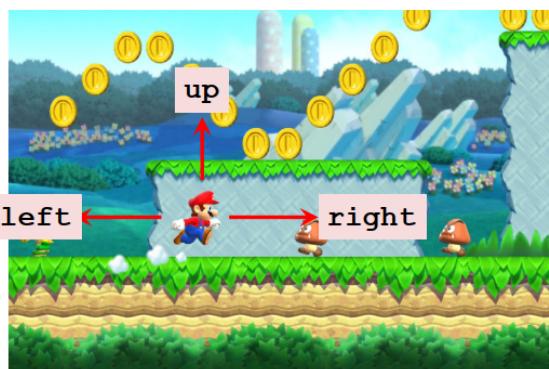
agent为智能体，也就是执行操作的对象，此处是马里奥，也可以是机器人，车等

action为agent做出的动作

Policy

Terminology: policy

policy π



- $\pi(a | s)$ is the probability of taking action $A = a$ given state s , e.g.,
 - $\pi(\text{left} | s) = 0.2$,
 - $\pi(\text{right} | s) = 0.1$,
 - $\pi(\text{up} | s) = 0.7$.
- Upon observing state $S = s$, the agent's action A can be random.

Policy函数，即策略函数，通常是一个概率分布函数，如该ppt中，在当前state下，马里奥选择往左走的策略的概率为0.2。该state下马里奥有三种策略，会在其中随机抽样，随机性使得policy更加灵活，难以被预测

Reward

Terminology: reward

reward R



- Collect a coin: $R = +1$
- Win the game: $R = +10000$
- Touch a Goomba: $R = -10000$ (game over).
- Nothing happens: $R = 0$

reward为奖励函数，如此处吃到金币+1分，通关+1w分，碰到敌人（game over）扣1w分

State Transition

Terminology: state transition

state transition



- old state → action → new state
- E.g., “up” action leads to a new state.
 - State transition can be random.
 - Randomness is from the environment.
 - $p(s'|s, a) = \mathbb{P}(S' = s' | S = s, A = a)$.

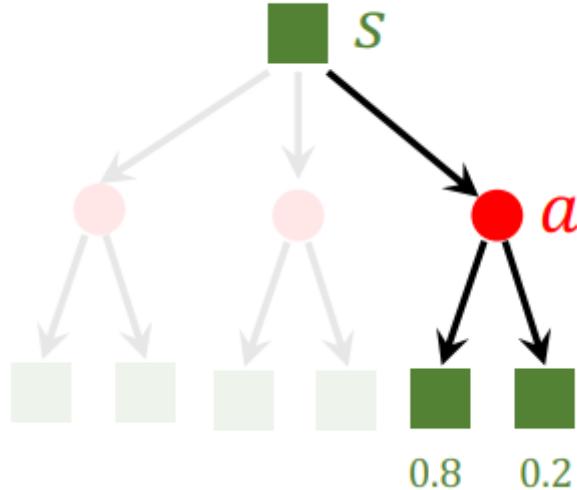
状态转移，顾名思义是old state变为new state，上图 P 为条件概率密度函数，意思是如果观测到当前的状态 S 以及动作 A ，下一个状态变成 S' 的概率

由于env以及action的随机性，所以状态转移具有随机性

Two Sources of Randomness

整个过程中的随机性主要来源于两点：

1. 是action的随机性，这个很好理解
2. 是state的随机性，在马里奥的例子中可以理解为敌人移动也是随机的，无法被agent知晓



- The randomness in **action** is from the policy function:

$$A \sim \pi(\cdot | s).$$

- The randomness in **state** is from the state-transition function:

$$S' \sim p(\cdot | s, a).$$

Trajectory

整个过程可以认为是

1. 观察state
2. 做出action
3. 观察到新的state并得到奖励（或惩罚）

Play game using AI

- (state, action, reward) trajectory:

$$\underline{s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_n, a_n, r_n}.$$

- One episode is from the beginning to the end (Mario wins or dies).



Rewards and Returns

returns 定义为未来所有 cumulative future reward 未来的累积奖励

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

但其实之后的奖励对当前时刻不是同等重要的，如选择现在给你100元和一年后给你100元，大部分人会选择立刻得到一百

所以引入折扣回报 Discounted return

Definition: Discounted return (at time t).

$$\bullet U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots + \gamma^{n-t} R_n.$$

gamma 介于0到1，为超参数

R 与 S 和 A 有关

- Reward R_i depends on S_i and A_i .
- States can be random: $S_i \sim p(\cdot | s_{i-1}, a_{i-1})$.
- Actions can be random: $A_i \sim \pi(\cdot | s_i)$.
- If either S_i or A_i is random, then R_i is random.

所以 U 与未来所有 $S A$ 有关

At time t , the rewards, R_t, \dots, R_n , are **random**, so the return U_t is **random**.

- Reward R_i depends on S_i and A_i .
- U_t depends on $\underline{R_t, R_{t+1}, \dots, R_n}$.
- → U_t depends on $\underline{S_t, A_t, S_{t+1}, A_{t+1}, \dots, S_n, A_n}$.

Value Function

价值函数

Action-value function

U_t 其实是一个随机变量，他依赖于之后的所有动作和状态， t 时刻我们并不知道 U_t 是什么，所以我们可以对 U_t 求期望得到一个函数，即Action-value function，动作价值函数， Q_{π}

Definition: Action-value function for policy π .

- $Q_{\pi}(s_t, a_t) = \mathbb{E} [U_t | S_t = s_t, A_t = a_t].$



- Return U_t depends on actions $A_t, A_{t+1}, A_{t+2}, \dots$ and states $S_t, S_{t+1}, S_{t+2}, \dots$
- Actions are random: $\mathbb{P}[A = a | S = s] = \pi(a|s)$. (Policy function.)
- States are random: $\mathbb{P}[S' = s' | S = s, A = a] = p(s'|s, a)$. (State transition.)

此时我们 U_t 未知， S_t 和 A_t 是变量，且他们的概率分布已知 (S -t, A -t分布)，则可以用积分的方式把将来的SA对当前时刻 U_t 的影响通过积分求期望的方式获得

将 t 时刻之后的随机变量 A, S 都用积分积掉，之后得到的 Q_{π} 就只与当前时刻 t 的SA以及 π 有关

Action-value function的实际意义，一直policy函数 π 以及当前 t 时刻的 s ，则可以通过Action-value function Q_{π} 对每个action打分，看做出哪个action，最终 U_t 的期望最高

Optimal Action-value function

最优动作价值函数

之前说的Action-value function与 π , SA有关，而我们有很多个policy函数 π ，我们要使得Action-value function最大，则可以做一个动态规划，取得最优的policy函数 π^* ，使得Action-value function最大，这样就可以消除 π 对Action-value function的影响（因为policy函数 π 已经确定了，不再是变量），得到一个最优的Action-value function，即Optimal Action-value function

Definition: Optimal action-value function.

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

Definition: State-value function.

- $V_\pi(s_t) = \mathbb{E}_A [Q_\pi(s_t, A)]$

Definition: State-value function.

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

将Qpi对A求期望 (将A当作随机变量) , 从而消掉A

物理意义在于可以评估目前状态的胜算

Conclusion

Understanding the Value Functions

- Action-value function: $Q_\pi(s, a) = \mathbb{E} [U_t | S_t = s, A_t = a]$.
- Given policy π , $Q_\pi(s, a)$ evaluates how good it is for an agent to pick action a while being in state s .
- State-value function: $V_\pi(s) = \mathbb{E}_A [Q_\pi(s, A)]$
- For fixed policy π , $V_\pi(s)$ evaluates how good the situation is in state s .
- $\mathbb{E}_S [V_\pi(s)]$ evaluates how good the policy π is.

How does AI control the agent

1. policy-based learning 策略学习: 已知pi, S, 可以求得每个A的概率, 再随机抽样
2. value-based learning 价值学习: 求Optimal Action-value function Q-star

How does AI control the agent?

Suppose we have a good policy $\pi(a|s)$.

- Upon observe the state s_t ,
- random sampling: $a_t \sim \pi(\cdot | s_t)$.

Suppose we know the optimal action-value function $Q^*(s, a)$.

- Upon observe the state s_t ,
- choose the **action** that maximizes the value: $a_t = \operatorname{argmax}_a Q^*(s_t, a)$.

Value-Based Reinforcement Learning

Deep Q-Network (DQN)

回顾上节课讲的**Optimal Action-value function** Q_{star} , Q_{star} 的作用是判断在当前state下，哪个action带来的未来reward总和的期望越大。而 Q_{star} 往往是不能直接得到的，价值学习的基本想法就是通过学习一个函数（神经网络）来近似 Q_{star}

Approximate the Q Function

Goal: Win the game (\approx maximize the total reward.)

Question: If we know $Q^*(s, a)$, what is the best **action**?

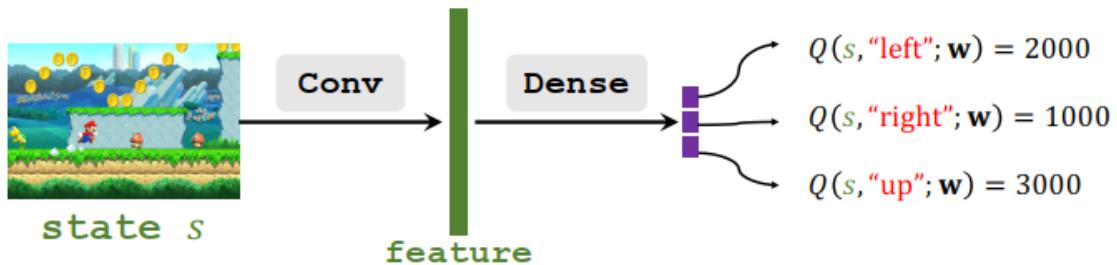
- Obviously, the best action is $a^* = \operatorname{argmax}_a Q^*(s, a)$.

Challenge: We do not know $Q^*(s, a)$.

- Solution: Deep Q Network (DQN)
- Use neural network $Q(s, a; w)$ to approximate $Q^*(s, a)$.

Deep Q Network (DQN)

- Input shape: size of the screenshot.
- Output shape: dimension of action space.



Question: Based on the predictions, what should be the **action**?

Temporal Difference (TD) Learning

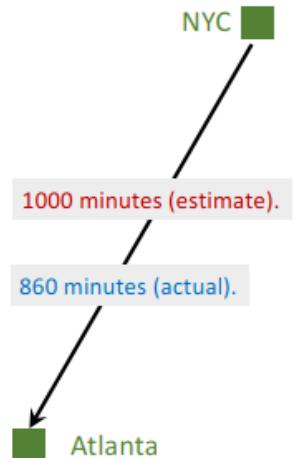
王老师举了一个预测开车时间的例子

Example

- I want to drive from NYC to Atlanta.
- Model $Q(w)$ estimates the time cost, e.g., 1000 minutes.

Question: How do I update the model?

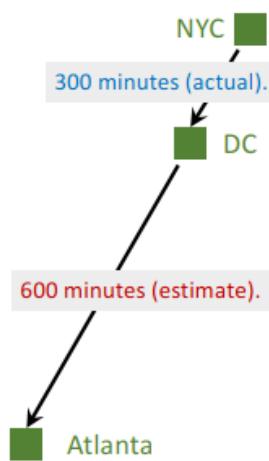
- Make a prediction: $q = Q(w)$, e.g., $q = 1000$.
- Finish the trip and get the target y , e.g., $y = 860$.
- Loss: $L = \frac{1}{2}(q - y)^2$.
- Gradient: $\frac{\partial L}{\partial w} = \frac{\partial q}{\partial w} \cdot \frac{\partial L}{\partial q} = (q - y) \cdot \frac{\partial Q(w)}{\partial w}$.
- Gradient descent: $w_{t+1} = w_t - \alpha \cdot \frac{\partial L}{\partial w} \Big|_{w=w_t}$.



如图用梯度下降法，比较naive，需要完成一次旅程才能update model

Temporal Difference (TD) Learning

- Model's estimate: $Q(\mathbf{w}) = 1000$ minutes.
- Updated estimate: $300 + 600 = 900$ minutes.
- TD target $y = 900$ is a more reliable estimate than 1000.
- Loss: $L = \frac{1}{2}(Q(\mathbf{w}) - y)^2$.
- Gradient: $\frac{\partial L}{\partial \mathbf{w}} = (1000 - 900) \cdot \frac{\partial Q(\mathbf{w})}{\partial \mathbf{w}}$.
- Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial L}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$.



使用TD learning，走到路程中间（300min处），再使用模型预测一次，预测值为600min，容易想到离终点越接近，该估计会越准，所以可以认为 $300+600=900$ 的估计比一开始的1000更准，1000与900的差称为TD error

How to apply TD learning to DQN?

- In the “driving time” example, we have the equation:

$$T_{\text{NYC} \rightarrow \text{ATL}} \approx T_{\text{NYC} \rightarrow \text{DC}} + T_{\text{DC} \rightarrow \text{ATL}}.$$

Model's estimate
Actual time
Model's estimate

- In deep reinforcement learning:

$$Q(s_t, a_t; \mathbf{w}) \approx r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w}).$$

TD learning可以运用的场景，即可以写作 $\text{estimate} = \text{estimate} + \text{actual}$ 的形式，其中的等于是我们最理想的情况，即TD error等于0

回顾discount return

$$\boxed{\text{Identity: } U_t = R_t + \gamma \cdot U_{t+1}.}$$

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

How to apply TD learning to DQN?

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

TD learning for DQN:

- DQN's output, $Q(s_t, a_t; \mathbf{w})$, is an estimate of U_t .
- DQN's output, $Q(s_{t+1}, a_{t+1}; \mathbf{w})$, is an estimate of U_{t+1} .
- Thus,
$$Q(s_t, a_t; \mathbf{w}) \approx r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w}).$$

Prediction TD target

Train DQN using TD learning

- Prediction: $Q(s_t, a_t; \mathbf{w}_t)$.

- TD target:

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

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

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

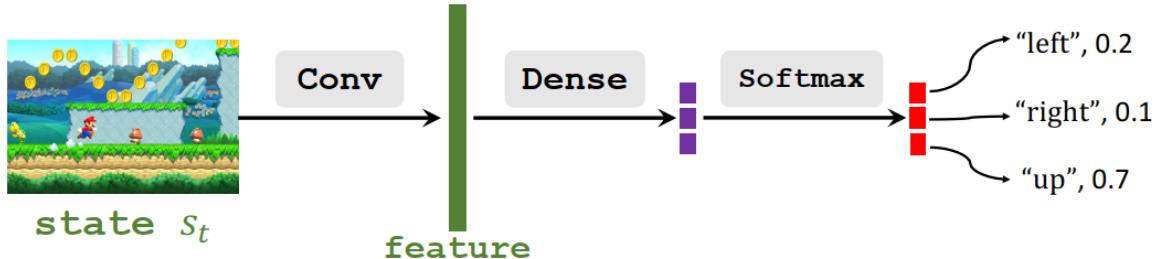
Policy-Based Reinforcement Learning

同样，策略函数 π 也是难以直接获得的，所以需要通过神经网络来近似，此神经网络被称为policy network

- Use policy network $\pi(a|s; \Theta)$ to approximate $\pi(a|s)$.

Policy Network $\pi(a|s; \theta)$

- $\sum_{a \in \mathcal{A}} \pi(a|s; \theta) = 1$.
- Here, $\mathcal{A} = \{\text{"left", "right", "up"}\}$ is the set all actions.
- That is why we use softmax activation.



回顾之前学的state-value function V_{π} , V_{π} 是 Q_{π} 对action求期望，可以表示在当前state下的胜算

$$\bullet V_{\pi}(s_t) = \mathbb{E}_A [Q_{\pi}(s_t, A)]$$

V_{π} 可以写作下图形式

Definition: State-value function.

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

Approximate state-value function.

- Approximate policy function $\pi(a|s_t)$ by policy network $\pi(a|s_t; \theta)$.
- Approximate value function $V_{\pi}(s_t)$ by:

$$V(s_t; \theta) = \sum_a \pi(a|s_t; \theta) \cdot Q_{\pi}(s_t, a).$$

$V(s, \theta)$ 可以度量状态S和策略网络theta的好坏，给定状态s，策略网络theta越好，则V越大

所以我们可以把目标函数设置为

Policy-based learning: Learn θ that maximizes $J(\theta) = \mathbb{E}_S [V(S; \theta)]$.

策略网络theta越好，J_theta越大

How to improve θ ? Policy gradient ascent!

- Observe state s .
- Update policy by: $\theta \leftarrow \theta + \beta \cdot \frac{\partial V(s; \theta)}{\partial \theta}$

Policy gradient

此处使用的是梯度上升算法，因为我们是想要目标函数越大越好（对比loss函数）

Policy gradient

此处Policy gradient的求导会用到高数和概率论，不太好记录，建议多看看ppt和视频

Policy Gradient

Definition: Approximate state-value function.

$$\bullet V(s; \theta) = \sum_a \pi(a|s; \theta) \cdot Q_\pi(s, a).$$

Policy gradient: Derivative of $V(s; \theta)$ w.r.t. θ .

$$\begin{aligned} \bullet \frac{\partial V(s; \theta)}{\partial \theta} &= \sum_a \frac{\partial \pi(a|s; \theta)}{\partial \theta} \cdot Q_\pi(s, a) \\ &= \sum_a \pi(a|s; \theta) \cdot \underbrace{\frac{\partial \log \pi(a|s; \theta)}{\partial \theta}}_{\text{blue box}} \cdot Q_\pi(s, a) \\ &= \mathbb{E}_A \left[\underbrace{\frac{\partial \log \pi(A|s; \theta)}{\partial \theta}}_{\text{blue box}} \cdot Q_\pi(s, A) \right]. \end{aligned}$$

The expectation is taken w.r.t. the random variable $A \sim \pi(\cdot | s; \theta)$.

Two forms of policy gradient:

- **Form 1:** $\frac{\partial V(s; \theta)}{\partial \theta} = \sum_a \frac{\partial \pi(a|s; \theta)}{\partial \theta} \cdot Q_\pi(s, a).$
- **Form 2:** $\frac{\partial V(s; \theta)}{\partial \theta} = \mathbb{E}_{A \sim \pi(\cdot | s; \theta)} \left[\frac{\partial \log \pi(A|s; \theta)}{\partial \theta} \cdot Q_\pi(s, A) \right].$

我们得到了以上两种方式来表示policy gradient

对于动作是离散形式，可以使用Form1枚举计算

Calculate Policy Gradient for Discrete Actions

If the actions are **discrete**, e.g., action space $\mathcal{A} = \{\text{"left"}, \text{"right"}, \text{"up"}\}, \dots$

Use **Form 1:** $\frac{\partial V(s; \theta)}{\partial \theta} = \sum_a \underbrace{\frac{\partial \pi(a|s; \theta)}{\partial \theta}}_{\text{blue box}} \cdot Q_\pi(s, a).$

1. Calculate $f(a, \theta) = \frac{\partial \pi(a|s; \theta)}{\partial \theta} \cdot Q_\pi(s, a)$, for every action $a \in \mathcal{A}$.
2. Policy gradient: $\frac{\partial V(s; \theta)}{\partial \theta} = f(\text{"left"}, \theta) + f(\text{"right"}, \theta) + f(\text{"up"}, \theta).$

This approach does not work for continuous actions.

对于action是连续形式，则需要积分，但Q_{pi}是一个神经网络非常复杂，不能直接积分得到解析解，所以需要使用蒙特卡洛算法近似（此处需要补概率论...）

具体可见[蒙特卡洛近似 - 知乎\(zhihu.com\)](#)

[蒙特卡洛近似的一些例子](#)

[数学理论——蒙特卡洛近似](#)

Calculate Policy Gradient

$$\text{Policy Gradient: } \frac{\partial V(s; \theta)}{\partial \theta} = \mathbb{E}_{A \sim \pi(\cdot | s; \theta)} \left[\frac{\partial \log \pi(A | s; \theta)}{\partial \theta} \cdot Q_\pi(s, A) \right].$$

1. Randomly sample an action \hat{a} according to $\pi(\cdot | s; \theta)$.
2. Calculate $\mathbf{g}(\hat{a}, \theta) = \frac{\partial \log \pi(\hat{a} | s; \theta)}{\partial \theta} \cdot Q_\pi(s, \hat{a})$.
3. Use $\mathbf{g}(\hat{a}, \theta)$ as an approximation to the policy gradient $\frac{\partial V(s; \theta)}{\partial \theta}$.

1. 随机抽样一个 a_{hat} , 抽样是根据概率密度函数 π 抽的
2. 计算 $\mathbf{g}(a_{\text{hat}}, \theta)$

- By the definition of \mathbf{g} , $\mathbb{E}_A[\mathbf{g}(A, \theta)] = \frac{\partial V(s; \theta)}{\partial \theta}$.
- $\mathbf{g}(\hat{a}, \theta)$ is an unbiased estimate of $\frac{\partial V(s; \theta)}{\partial \theta}$.

可以知道 $\mathbf{g}(A, \theta)$ 对 A 求期望即为 V 的导数

且 $\mathbf{g}(a_{\text{hat}}, \theta)$ 是 V 求导的无偏估计 (?)

则可以用 $\mathbf{g}(a_{\text{hat}}, \theta)$ 来近似 V 求导

蒙特卡洛算法就是通过抽取一个或多个样本对期望进行近似

整个流程如下图所示

Algorithm

1. Observe the state s_t .
2. Randomly sample action a_t according to $\pi(\cdot | s_t; \theta_t)$.
3. Compute $q_t \approx Q_\pi(s_t, a_t)$ (some estimate).
4. Differentiate policy network: $\mathbf{d}_{\theta, t} = \frac{\partial \log \pi(a_t | s_t; \theta)}{\partial \theta} \Big|_{\theta=\theta_t}$.
5. (Approximate) policy gradient: $\mathbf{g}(a_t, \theta_t) = q_t \cdot \mathbf{d}_{\theta, t}$.
6. Update policy network: $\theta_{t+1} = \theta_t + \beta \cdot \mathbf{g}(a_t, \theta_t)$.

但还有一个问题没有解决, 即由于 Q_π 无法得知, 所以 q_t 不能直接得到, 有如下两种方法来近似

1. Reinforce算法

因为Qpi的Ut的期望，所以可以用ut来近似Qpai，即近似qt，该方法需要完整玩完一局游戏才能对策略函数进行更新

Compute $q_t \approx Q_\pi(s_t, a_t)$ (some estimate). **How?**

Option 1: REINFORCE.

- Play the game to the end and generate the trajectory:

$$s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_T, a_T, r_T.$$

- Compute the discounted return $u_t = \sum_{k=t}^T \gamma^{k-t} r_k$, for all t .
- Since $Q_\pi(s_t, a_t) = \mathbb{E}[U_t]$, we can use u_t to approximate $Q_\pi(s_t, a_t)$.
- → Use $q_t = u_t$.

2. actor-critic method

用神经网络近似qt，下节课具体讲解

Actor-Critic Methods

State-Value Function Approximation

Definition: State-value function.

$$V_\pi(s) = \sum_a \pi(a|s) \cdot Q_\pi(s, a) \approx \sum_a \pi(a|s; \theta) \cdot q(s, a; w).$$

Policy network (actor):

- Use neural net $\pi(a|s; \theta)$ to approximate $\pi(a|s)$.
- θ : trainable parameters of the neural net.

Value network (critic):

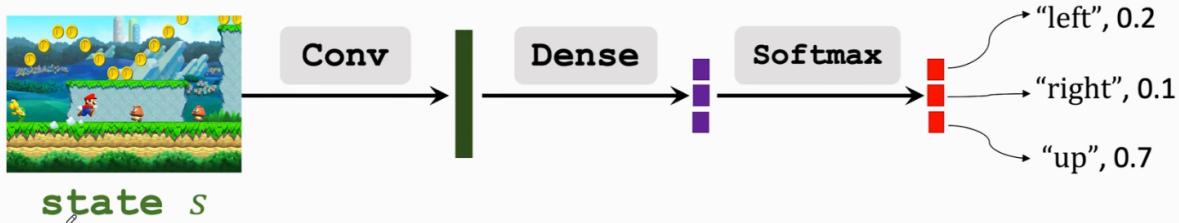
- Use neural net $q(s, a; w)$ to approximate $Q_\pi(s, a)$.
- w : trainable parameters of the neural net.

之前学习的Policy network，策略网络，使用net去近似policy函数pi，他控制agent的运动action，被称为actor

Value network不直接控制agent，而是对动作打分，被称为critic

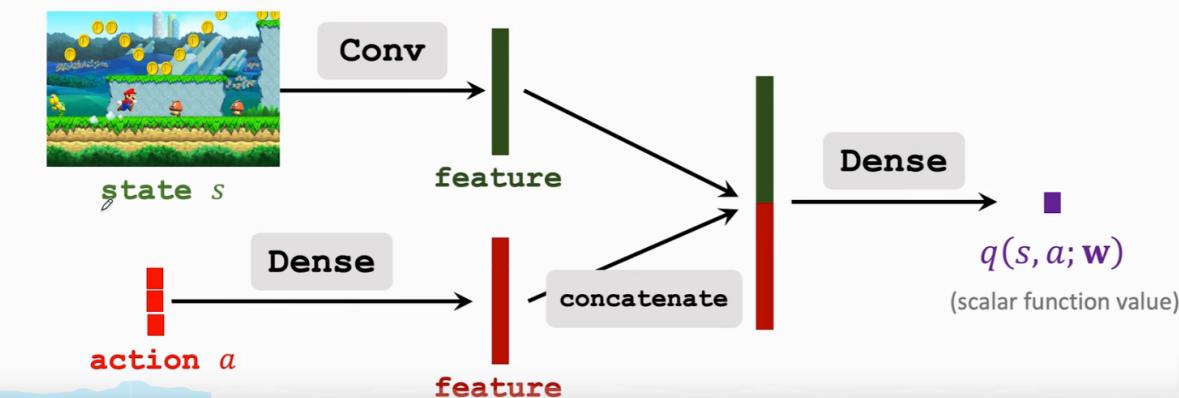
Policy Network (Actor): $\pi(a|s, \theta)$

- Input: state s , e.g., a screenshot of Super Mario.
- Output: probability distribution over the actions.
- Let \mathcal{A} be the set all actions, e.g., $\mathcal{A} = \{\text{"left"}, \text{"right"}, \text{"up"}\}$.
- $\sum_{a \in \mathcal{A}} \pi(a|s, \theta) = 1$. (That is why we use softmax activation.)



Value Network (Critic): $q(s, a; w)$

- Inputs: state s and action a .
- Output: approximate action-value (scalar).



更新两个network的目标是不同的

1. 更新policy network的目标是增大V的值 (因为V是对未来return的期望)
2. 更新value network的目标是使得q的打分更准

- Update policy network $\pi(a|s; \theta)$ to increase the state-value $V(s; \theta, w)$.
 - Actor gradually performs better.
 - Supervision is purely from the value network (critic).
- Update value network $q(s, a; w)$ to better estimate the return.
 - Critic's judgement becomes more accurate.
 - Supervision is purely from the rewards.

1. Observe the state s_t .
2. Randomly sample action a_t according to $\pi(\cdot | s_t; \theta_t)$.
3. Perform a_t and observe new state s_{t+1} and reward r_t .
4. Update w (in value network) using temporal difference (TD).
5. Update θ (in policy network) using policy gradient.

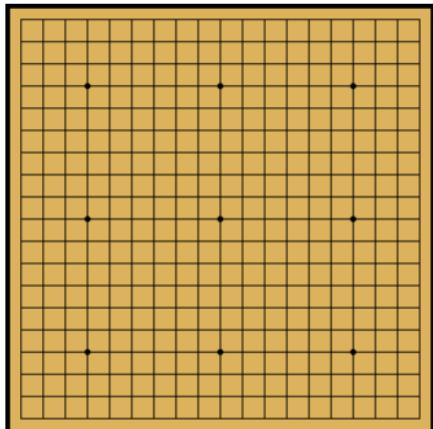
Summary

Summary of Algorithm

1. Observe state s_t and randomly sample $a_t \sim \pi(\cdot | s_t; \theta_t)$.
2. Perform a_t ; then environment gives new state s_{t+1} and reward r_t .
3. Randomly sample $\tilde{a}_{t+1} \sim \pi(\cdot | s_{t+1}; \theta_t)$. (Do not perform \tilde{a}_{t+1} !)
4. Evaluate value network: $q_t = q(s_t, a_t; w_t)$ and $q_{t+1} = q(s_{t+1}, \tilde{a}_{t+1}; w_t)$.
5. Compute TD error: $\delta_t = q_t - (r_t + \gamma \cdot q_{t+1})$.
6. Differentiate value network: $d_{w,t} = \frac{\partial q(s_t, a_t; w)}{\partial w} |_{w=w_t}$.
7. Update value network: $w_{t+1} = w_t - \alpha \cdot \delta_t \cdot d_{w,t}$.
8. Differentiate policy network: $d_{\theta,t} = \frac{\partial \log \pi(a_t | s_t; \theta)}{\partial \theta} |_{\theta=\theta_t}$.
9. Update policy network: $\theta_{t+1} = \theta_t + \beta \cdot q_t \cdot d_{\theta,t}$.

AlphaGO

Go Game



- The standard Go board has a 19×19 grid of lines, containing 361 points.
- **State:** arrangement of black, white, and space.
 - State s can be a $19 \times 19 \times 2$ tensor of 0 or 1.
 - (AlphaGo actually uses a $19 \times 19 \times 48$ tensor to store other information.)
- **Action:** place a stone on a vacant point.
 - Action space: $\mathcal{A} \subset \{1, 2, 3, \dots, 361\}$.
- Go is very complex.
 - Number of possible sequence of actions is 10^{170} .

Training and Execution

Training in 3 steps:

1. Initialize policy network using behavior cloning.
(Supervised learning from human experience.)
2. Train the policy network using policy gradient. (Two policy networks play against each other.)
3. After training the policy network, use it to train a value network.

Execution (actually play Go games):

- Do Monte Carlo Tree Search (MCTS) using the policy and value networks.

Imitation learning, 模仿学习

TD Learning

先总结一下Sarsa和Q learning

Q learning需要用到st at rt st+1

Sarsa需要用到st at rt st+1 at+1

个人认为Q learning更加贪婪，他每次都直接选择最大的Qpi (更容易陷入局部最优?)

从代码上看，Q learning是选择了t时间动作之后，更新table后，再选择下一步动作

```
1 while True:  
2     # fresh env  
3     env.render()  
4  
5     # RL choose action based on observation  
6     action = RL.choose_action(str(observation))  
7  
8     # RL take action and get next observation and reward  
9     observation_, reward, done = env.step(action)  
10  
11    # RL learn from this transition  
12    RL.learn(str(observation), action, reward, str(observation_))  
13  
14    # swap observation  
15    observation = observation_  
16  
17    # break while loop when end of this episode  
18    if done:  
19        break
```

而Sarsa是在更新table前就选好了at和at+1

```
1     while True:
2         # fresh env
3         env.render()
4
5         # RL take action and get next observation and reward
6         observation_, reward, done = env.step(action)
7
8         # RL choose action based on next observation
9         action_ = RL.choose_action(str(observation_))
10
11        # RL learn from this transition (s, a, r, s, a) ==> Sarsa
12        RL.learn(str(observation), action, reward, str(observation_), 
13        action_)
14
15        # swap observation and action
16        observation = observation_
17        action = action_
18
19        # break while loop when end of this episode
20        if done:
21            break
```

第一点区别在于，Q-learning在s2的时候无论如何都选最大的action的Q值作为反馈，但是Sarsa就会不会改变主意选最大的，而是就选了自己随便选的那个step走走看再更新。

第二点区别在于：区别就在a'那里，Q-learning预选一个a'但是下一回合并不一定执行a'（因为下一回合Q表就更新了）。可是Sarsa下一回合却一定会执行a'

Sarsa

TD Target

- Assume R_t depends on (S_t, A_t, S_{t+1}) .
- $$\begin{aligned} Q_\pi(s_t, a_t) &= \mathbb{E}[U_t | s_t, a_t] \\ &= \mathbb{E}[R_t + \gamma \cdot U_{t+1} | s_t, a_t] \\ &= \mathbb{E}[R_t | s_t, a_t] + \gamma \mathbb{E}[U_{t+1} | s_t, a_t] \\ &= \mathbb{E}[R_t | s_t, a_t] + \gamma \cdot \mathbb{E}[Q_\pi(S_{t+1}, A_{t+1}) | s_t, a_t]. \end{aligned}$$

$\mathbb{E}(U_{t+1}) = Q_{\pi^*}$, 又 $\mathbb{E}(\mathbb{E}(\cdot)) = \mathbb{E}(\cdot)$

所以可以写作：

Identity: $Q_\pi(s_t, a_t) = \mathbb{E}[R_t + \gamma \cdot Q_\pi(S_{t+1}, A_{t+1})]$, for all π .

右侧的期望做蒙特卡洛近似为yt (TD target)

$$\approx r_t + \gamma \cdot Q_\pi(s_{t+1}, a_{t+1})$$

TD target y_t

TD learning的目的就是让动作价值函数 Q_{π} 去接近 y_t , 这是因为 Q_{π} 完全是估计, 而 y_t 一部分基于真实 reward, 所以认为 y_t 更可信

具体算法

表格形式, 通常用在 S 和 A 较少的情况下, 表格中的值即为 Q_{π} , 需要更新 (随机初始化, 可中途加入新的 State)

	Action a_1	Action a_2	Action a_3	Action a_4	...
State s_1					
State s_2					
State s_3					
:					

算法流程

Sarsa (tabular version)

- Observe a transition (s_t, a_t, r_t, s_{t+1}) .
- Sample $a_{t+1} \sim \pi(\cdot | s_{t+1})$, where π is the policy function.
- TD target: $y_t = r_t + \gamma \cdot Q_\pi(s_{t+1}, a_{t+1})$.
- TD error: $\delta_t = Q_\pi(s_t, a_t) - y_t$.
- Update: $\underline{Q_\pi(s_t, a_t)} \leftarrow Q_\pi(s_t, a_t) - \alpha \cdot \delta_t$.

其中 Q_{π} 是通过查表得到的

神经网络形式

TD Error & Gradient

- TD target: $y_t = \textcolor{blue}{r}_t + \gamma \cdot q(\textcolor{brown}{s}_{t+1}, \textcolor{red}{a}_{t+1}; \mathbf{w})$.
- TD error: $\delta_t = q(\textcolor{brown}{s}_t, \textcolor{red}{a}_t; \mathbf{w}) - y_t$.
- Loss: $\delta_t^2 / 2$.
- Gradient: $\frac{\partial \delta_t^2 / 2}{\partial \mathbf{w}} = \delta_t \cdot \frac{\partial q(\textcolor{brown}{s}_t, \textcolor{red}{a}_t; \mathbf{w})}{\partial \mathbf{w}}$.
- Gradient descent: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \delta_t \cdot \frac{\partial q(\textcolor{brown}{s}_t, \textcolor{red}{a}_t; \mathbf{w})}{\partial \mathbf{w}}$.

q为神经网络，被称为Value network

Q learning

Sarsa对应Qpi，而Q learning 对应Q_star(最优动作价值函数)

同理可以写出他的TD Target

Derive TD Target

- We have proved that for all π ,

$$Q_\pi(\textcolor{brown}{s}_t, \textcolor{red}{a}_t) = \mathbb{E}[\textcolor{blue}{R}_t + \gamma \cdot Q_\pi(\textcolor{brown}{S}_{t+1}, \textcolor{red}{A}_{t+1})].$$

- If π is the optimal policy π^* , then

$$Q_{\pi^*}(\textcolor{brown}{s}_t, \textcolor{red}{a}_t) = \mathbb{E}[\textcolor{blue}{R}_t + \gamma \cdot Q_{\pi^*}(\textcolor{brown}{S}_{t+1}, \textcolor{red}{A}_{t+1})].$$

- Q_{π^*} and Q^* both denote *the optimal action-value function*.

Identity: $Q^*(\textcolor{brown}{s}_t, \textcolor{red}{a}_t) = \mathbb{E}[\textcolor{blue}{R}_t + \gamma \cdot Q^*(\textcolor{brown}{S}_{t+1}, \textcolor{red}{A}_{t+1})]$.

又根据下述等式

- The action A_{t+1} is computed by

$$A_{t+1} = \underset{\mathbf{a}}{\operatorname{argmax}} Q^*(S_{t+1}, \mathbf{a}).$$

- Thus $Q^*(S_{t+1}, \underline{A_{t+1}}) = \underset{\mathbf{a}}{\max} Q^*(S_{t+1}, \mathbf{a})$.

即 A_{t+1} 是最优动作，他可以最优化 Q_{star} 函数

表格形式，与 Sarsa 不同的是，这里取一行中最大值对应的 a

- TD target: $y_t = r_t + \gamma \max_{\mathbf{a}} Q^*(S_{t+1}, \mathbf{a})$.

	Action a_1	Action a_2	Action a_3	Action a_4	...
State s_1					
State s_2					
State s_3					
:					