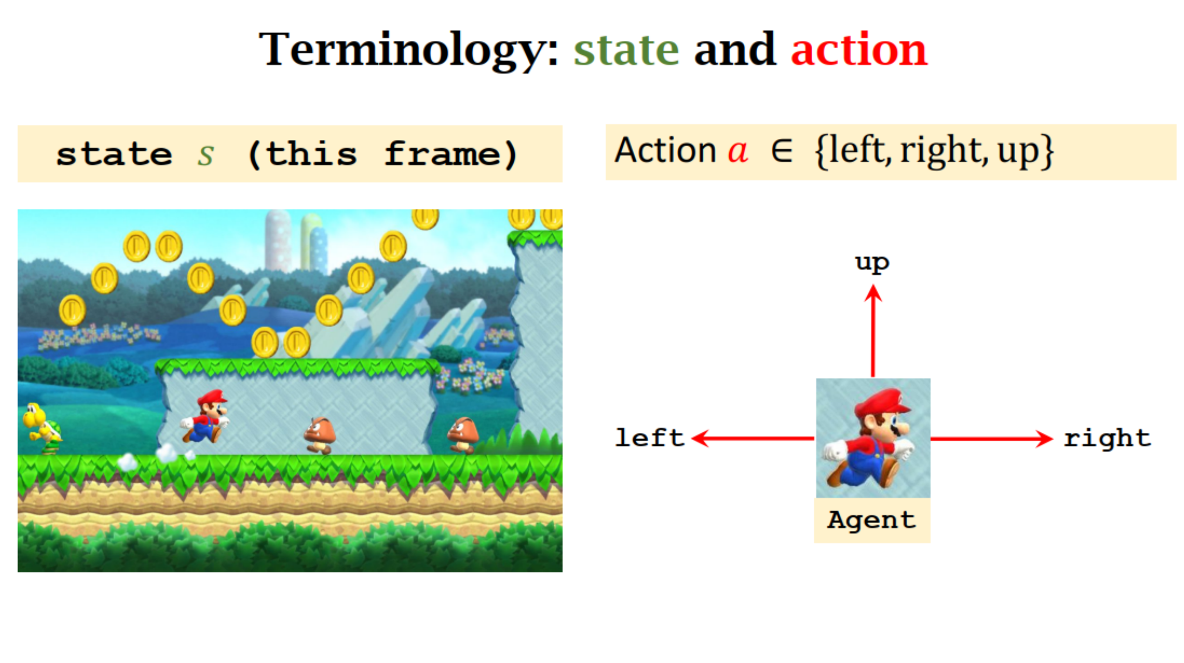
Reinforcement Learning

Terminologies

State and Action

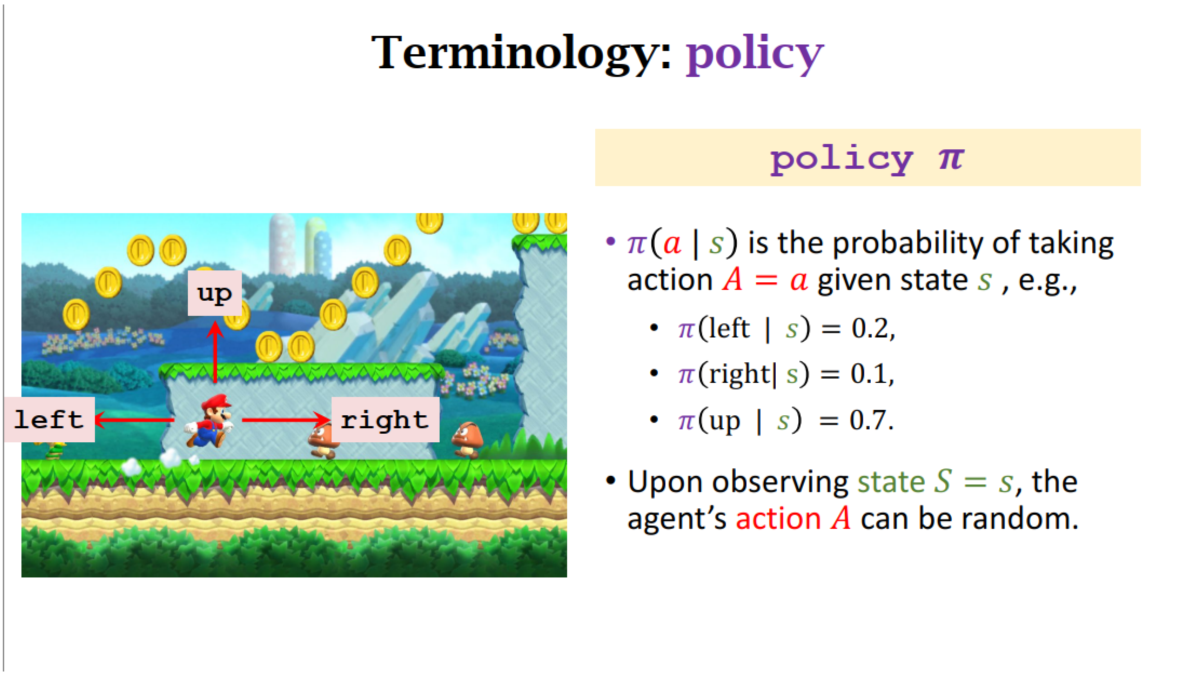


state是场景

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

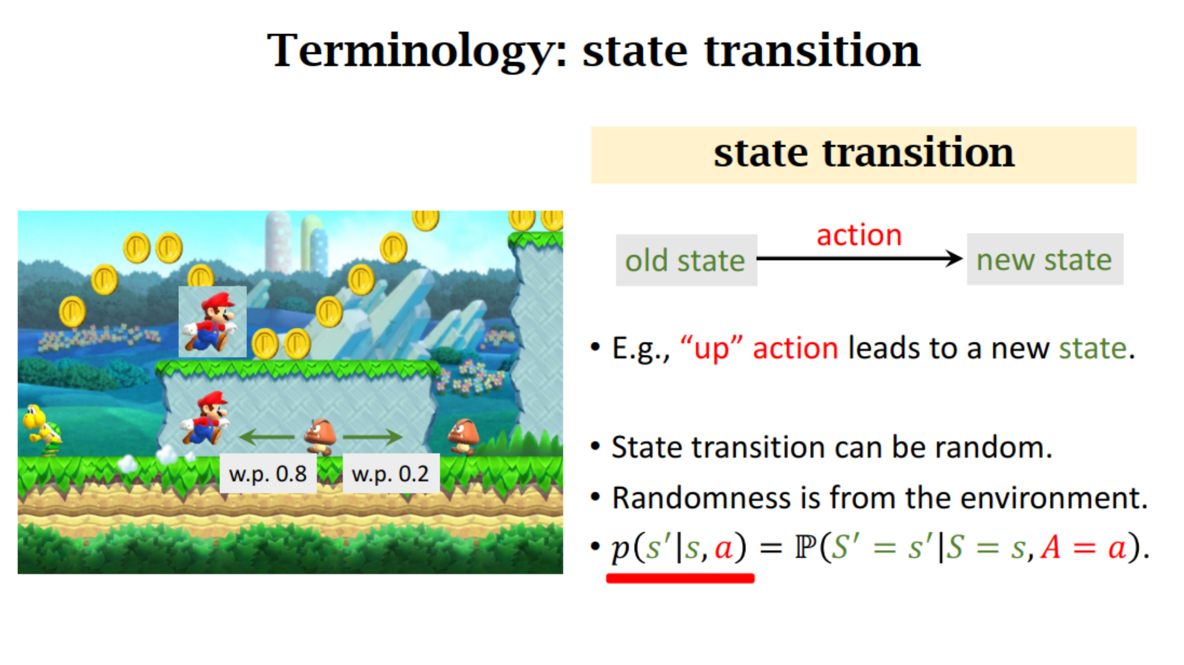


Policy函数，即策略函数，通常是一个概率分布函数，如该ppt中，在当前state下，马里奥选择往左走的策略的概率为0.2。该state下马里奥有三种策略，会在其中随机抽样，随机性使得policy更加灵活，难以被预测

Reward

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



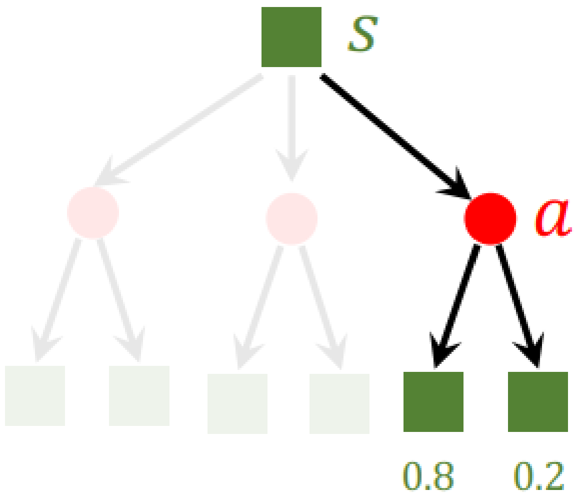
状态转移，顾名思义是old state变为new state，上图P为条件概率密度函数，意思是如果观测到当前的状态S以及动作A，下一个状态变成S一撇的概率

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:

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

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

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

Trajectory

整个过程可以认为是

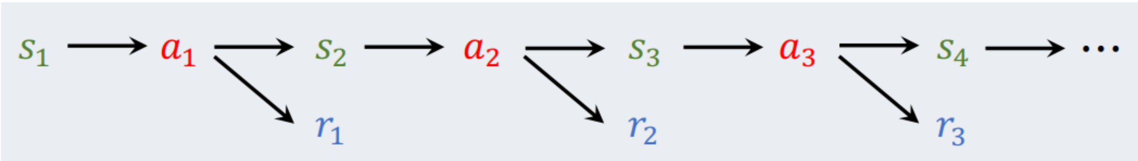
1. 观察state
2. 做出action
3. 观察到新的state并得到奖励（或惩罚）

Play game using AI

- (state, action, reward) trajectory:

$$s_1, a_1, r_1,\ \ s_2, a_2, r_2,\ \ \cdots\ ,\ \ s_n, a_n, r_n.$$

- One episode is from the 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

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

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).

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

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: $S_i \sim p(\cdot \mid s_{i-1}, a_{i-1})$.
- Actions can be random: $A_i \sim \pi(\cdot \mid s_i)$.
- 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, $R_t, \cdots, R_n$, are random, so the return U_t is random.

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

Value Function

价值函数

Action-value function

Ut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Ut是什么，所以我们可以对Ut求期望得到一个函数，即Action-value function，动作价值函数，Qpi

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

此时我们Ut未知，St和At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Ut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Qpi就只与当前时刻t的SA以及pi有关

Action-value function的实际意义，一直policy函数pi以及当前t时刻的s，则可以通过Action-value function Qpi对每个action打分，看做出哪个action，最终Ut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pi，SA有关，而我们有很多个policy函数pi，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pi，使得Action-value function最大，这样就可以消除pi对Action-value function的影响（因为policy函数pi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.

- $Q^\star(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t)$.

State-value function

Definition: State-value function.

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

Definition: State-value function.

- $V_\pi(s_t) = \mathbb{E}_A\left[Q_\pi(s_t, A)\right] = \sum_a \pi(a|s_t) \cdot Q_\pi(s_t, a)$. (Actions are discrete.)
- $V_\pi(s_t) = \mathbb{E}_A\left[Q_\pi(s_t, A)\right] = \underline{\int \pi(a|s_t) \cdot Q_\pi(s_t, a)\, da}$. (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: $Q_\pi(s, a) = \mathbb{E}\left[U_t | S_t = s, A_t = a\right]$.
- 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: $V_\pi(s) = \underline{\mathbb{E}_A\left[Q_\pi(s, A)\right]}$
- For fixed policy π, $V_\pi(s)$ evaluates how good the situation is in state s.
- $\underline{\mathbb{E}_S\left[V_\pi(S)\right]}$ 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 $Q^\star(s,a)$.

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

Value-Based Reinforcement Learing

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 (≈ maximize the total reward.)

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

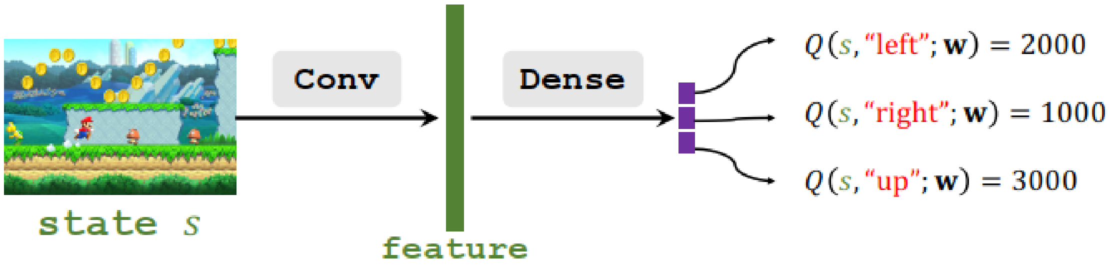
- Obviously, the best action is $a^\star = \underset{a}{\text{argmax}}\, Q^\star(s,a)$.

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

- Solution: Deep Q Network (DQN)
- Use neural network $Q(s,a;\mathbf{w})$ to approximate $Q^\star(s,a)$.

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

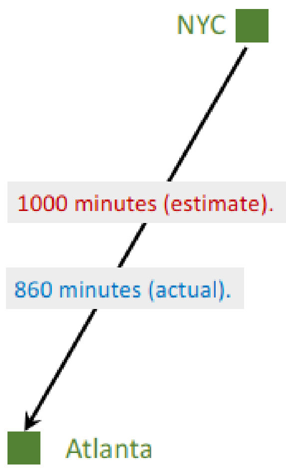
王老师举了一个预测开车时间的例子

Example

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

Question: How do I update the model?

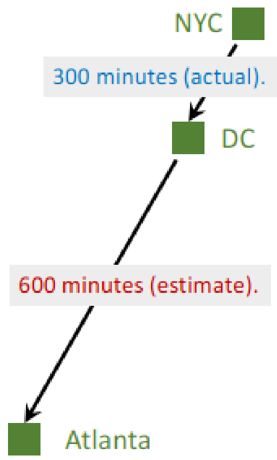
- Make a prediction: $q = Q(\mathbf{w})$, e.g., $q = 1000$.
- Finish the trip and get the target y, e.g., $y = 860$.
- Loss: $L = \frac{1}{2}(q - y)^2$.
- Gradient: $\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial q}{\partial \mathbf{w}} \cdot \frac{\partial L}{\partial q} = (q - y) \cdot \frac{\partial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



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: $\underbrace{300 + 600 = 900}_{\text{TD target.}}$ 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 learing，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:

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

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estimate = estimate+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

> $\boxed{\textbf{Identity: } U_t = R_t + \gamma \cdot U_{t+1}.}$
>
> - $U_t = R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \gamma^3 \cdot R_{t+3} + \gamma^4 \cdot R_{t+4} + \cdots$
> $= R_t + \gamma \cdot \underbrace{(R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \gamma^3 \cdot R_{t+4} + \cdots)}_{= U_{t+1}}$

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, $\underbrace{Q(s_t, a_t; \mathbf{w})}_{\text{Prediction}} \approx \underbrace{r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w})}_{\text{TD target}}$.

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

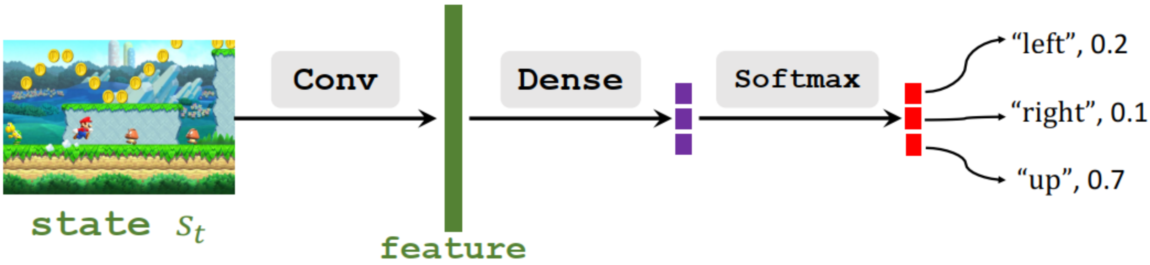
Policy-Based Reinforcement Learing

同样，策略函数pi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 $\pi(a|s; \boldsymbol{\theta})$ to approximate $\pi(a|s)$.

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

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



回顾之前学的state-value fuction Vpi，Vpi是Qpi对action求期望，可以表示在当前state下的胜算

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

Vpi可以写作下图形式

Definition: State-value function.

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

Approximate state-value function.

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

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

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 $\boldsymbol{\theta}$ that maximizes $J(\boldsymbol{\theta}) = \mathbb{E}_S[V(S;\boldsymbol{\theta})]$.

策略网络theta越好，J_theta越大

How to improve $\boldsymbol{\theta}$? Policy gradient ascent!

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

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.

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

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

- $$\frac{\partial V(s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_a \frac{\partial \pi(a|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s,a)$$
$$= \sum_a \pi(a|s;\boldsymbol{\theta}) \cdot \frac{\partial \log \pi(a|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s,a)$$
$$= \mathbb{E}_A\left[\frac{\partial \log \pi(A|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s,A)\right].$$

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

Two forms of policy gradient:

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

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 $\mathcal{A} = \{\text{"left"}, \text{"right"}, \text{"up"}\}$, ...

Use Form 1: $\frac{\partial V(s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_a \frac{\partial \pi(a|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \cdot Q_\pi(s,a)$.

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

This approach does not work for continuous actions.

对于action是连续形式，则需要积分，但Qpi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)

Calculate Policy Gradient

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

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

1. 随机抽样一个a_hat，抽样是根据概率密度函数pi抽的
2. 计算g（a_hat,theta）

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

可以知道g（A，theta）对A求期望即为V的导数

且g（a_hat,theta）是V求导的无偏估计（？）

则可以用g（a_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 $\pi(\cdot\,|s_t;\boldsymbol{\theta}_t)$.
3. Compute $q_t \approx Q_\pi(s_t,a_t)$ (some estimate).
4. Differentiate policy network: $\mathbf{d}_{\theta,t} = \frac{\partial \log \pi(a_t|s_t,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$.
5. (Approximate) policy gradient: $\mathbf{g}(a_t,\boldsymbol{\theta}_t) = q_t \cdot \mathbf{d}_{\theta,t}$.
6. Update policy network: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \beta \cdot \mathbf{g}(a_t,\boldsymbol{\theta}_t)$.

但还有一个问题没有解决，即由于Qpi无法得知，所以qt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

3. $\boxed{\text{Compute } q_t \approx Q_\pi(s_t, a_t) \text{ (some estimate).}}$ How?

Option 1: REINFORCE.

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

$$s_1, a_1, r_1, \; s_2, a_2, r_2, \quad \cdots \quad , s_T, a_T, r_T.$$

- 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