术语Monte Carlo通常被更广泛地用于任何依赖 重复随机抽样(repeated random sampling) 的估计方法。在RL中，蒙特卡洛方法允许我们直接从经验，从状态、行动和奖励的序列中估计数值。从经验中学习是引人注目的，因为代理可以准确地估计一个价值函数，而不需要事先了解环境的动态。

前面我们谈到了强化学习与动态编程的关系。要使用纯动态编程方法，代理人需要知道环境的过渡概率(transition probabilities)。在一些问题中，我们根本不知道这些概率。想象一下，一个气象学家正试图预测天气。天气如何变化取决于各种环境因素。我们根本不知道未来天气模式的确切概率。即使对于合理的任务，计算动态编程中使用的过渡概率也很困难。

想想预测12次掷骰子的结果，DP计算的数量和复杂性使得它们在编码和数字精度方面都很繁琐和容易出错。这里是蒙特卡洛方法可以帮助的地方。让我们试着找出12个骰子的平均和。蒙特卡洛方法并不像这里显示的那样详尽地扫过所有可能的结果。事实上，你不需要知道任何结果的概率就能使用蒙特卡洛方法。相反，蒙特卡洛方法是通过对大量随机样本的平均化来估计数值。让我们把12个骰子掷几次，看看结果。在这种情况下，平均数是41.57，相当接近于真实的平均数42。在强化学习中，我们要学习一个价值函数。价值函数代表预期收益。因此，学习价值函数的蒙特卡洛方法将首先观察同一状态下的多个回报。然后，它对这些观察到的回报进行平均，以估计该状态下的预期回报。随着样本数量的增加，平均值趋向于越来越接近预期收益。代理人从一个状态观察到的回报越多，样本平均值越有可能接近状态值。这些回报只有在一个事件结束时才能观察到。因此，我们将重点讨论用于偶发任务的蒙特卡洛方法。强化学习中的蒙特卡洛方法看起来有点像土匪方法。在强盗法中，一个手臂的价值是用拉动该手臂的平均报酬来估计的。蒙特卡洛方法考虑的是策略而不是手臂。在一个给定的政策下，价值状态S是使用从S到终止的该政策的平均回报采样来估计的。现在让我们来看看估计政策的状态价值函数的算法。蒙特卡洛算法必须跟踪多个观察到的回报。让我们引入一个列表，每个状态都有一个回报。每个列表持有从状态S观察到的回报，然后我们通过遵循我们的政策产生一个情节。对于情节中的每个日期，我们计算回报，并在回报列表中开始，但我们如何才能有效地做到这一点？

Monte Carlo methods estimate values by averaging over a large number of random samples

→統計的な数から大数を予知

Sum of Faces	Probability
12	?
13	?
14	?
…	
71	?
72	?

Samples

39	38	49	36	33	47	37	49	41	40	44	44
35	43	47	28	46	52	47	45	43	43	42	43
40	32	29	48	43	49	48	39	42	35	44	48
36	32	41	49	34	54	37	42	37	38	42	50
39	37	45	49	44	34	38	50	35	36	42	45

Average = 41.57

MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$ => Each list holds the returns observed from state S.

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$ => Generate an episode by following our policy. For each state in the episode, we compute the return and store in the list of returns.

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Append G to $Returns(S_t)$

=> On the previous time step t minus two, we calculate the return as before then added to the list of returns for S_t minus two.

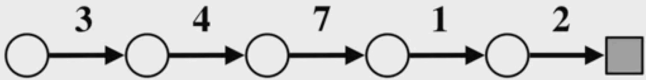
$V(S_t) \leftarrow \text{average}(Returns(S_t))$

Then, we set the value of S_t minus one to be the average of returns S_t minus one.

Finally we update the value of S_t minus two.

If we continue this loop until the end we'll, have updated the values for all the states visited in the current episode.

Computing returns efficiently



$\gamma = 0.5$

$$G_0 = R_1 + \gamma G_1$$

$$G_1 = R_2 + \gamma G_2$$

$$G_2 = R_3 + \gamma G_3$$

$$G_3 = R_4 + \gamma G_4$$

$$G_4 = R_5 + \gamma G_5 = 2 + 8 \cdot 0 = 2$$

因为 the episode ends at $t=5$, $\therefore G_5 = 0$

$$G_5 = 0 \Rightarrow$$

所以我们可以从 G_5 开始倒推.

MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Append G to $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

Incremental Update

$$NewEstimate \leftarrow OldEstimate + StepSize[Target - OldEstimate]$$

Summary

- We talked about how Monte Carlo methods learn directly from interaction
- We showed a Monte Carlo algorithm for learning state-values in episodic problems

Blackjack

Collect cards so that their sum is as large as possible without exceeding 21

= 10

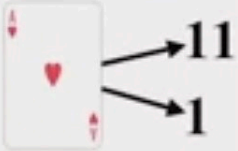
11

1

Blackjack

Collect cards so that their sum is as large as possible without exceeding 21

= 10



Problem Formulation

- Undiscounted MDP where each game of blackjack corresponds to an episode
- Reward: -1 for a loss, 0 for a draw, and 1 for a win
- Actions: Hit or Stick
- States (200 in total):
    - Whether the player has a usable ace (Yes or No)
    - The sum of the player's cards (12-21)
    - The card the dealer shows (Ace-10)
- Cards are dealt from a deck with replacement
- Policy: Stops requesting cards when the player's sum is 20 or 21 2)在牌的总和达到20或21时拒绝要牌.

S (Usable Ace, Sum, Dealer)	Returns(S)	V(S)

A = (NoAce, 20, 10)

State A

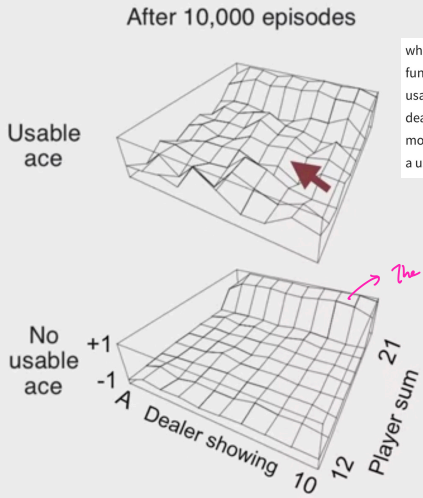
Player win. 얻은 R = +1

S (Usable Ace, Sum, Dealer)	Returns(S)	V(S)
A	Returns(A) = [1]	1
B	Returns(B) = [1]	1

A = (NoAce, 20, 10)

B = (NoAce, 13, 10)

Approximate state-value functions

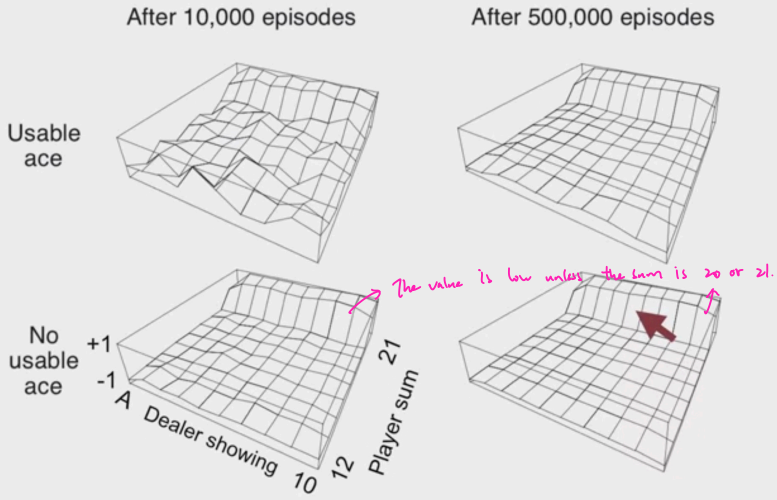


what would happen if the agent played many games of blackjack? Let's look at the value function the agent learns after 10,000 episodes. We'll plot one value function for states with a usable ace and one for states without a usable ace. The three axis of the plot are the card the dealer's showing, the agents sum, and the value of that state. The plot with usable aces is much more rough than the plot with no usable ace. That's because most blackjack games do not have a usable ace so the usable ace values are estimated with far fewer samples.

The value is low unless the sum is 20 or 21.

If we look at the agent's sum however, the value function is much higher in states where the agent has a 20 or 21. Why are these values so much higher than when the agent has a sum of 19? The answer has to do with the policy the agent is following. The policy always hits on sums of 19 or lower. So the agent is likely to bust when the sum is 19. On the other hand, when the sum is 20 or 21 then the agent will stick and is likely to win.

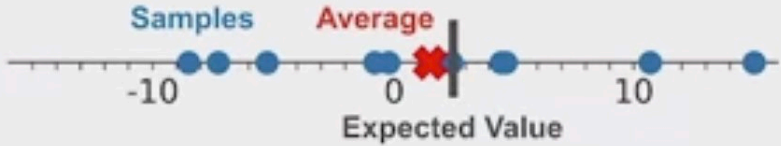
Approximate state-value functions



Monte Carlo methods in Reinforcement Learning

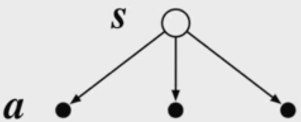
Recall that

$$q_\pi(s,a) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$$

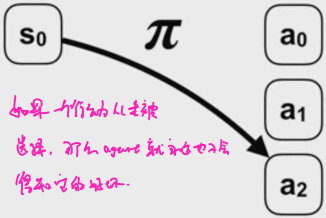


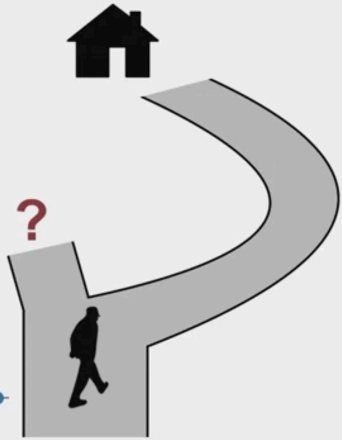
from that state. Conveniently, the same process works for action values. We collect returns following a policy from state-action pair and then take their average. But why do we care about

Action-values are useful for learning a policy



$$\underset{a}{\operatorname{argmax}}\, q_\pi(s,a)$$





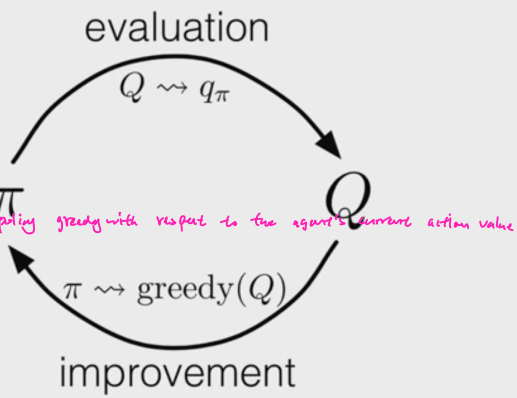
1:50 / 2:58

现在我们可以使用蒙特卡洛估计行动值，下一步是建立一个 通用的策略迭代算法(generalized policy iteration algorithm)。今天让我们来看看我们如何做到这一点。在本视频结束时，你将了解如何使用蒙特卡洛方法来实现GPI算法。

GPI包括一个策略评估(policy evaluation)和一个策略改进(policy improvement)步骤。GPI算法产生的政策序列(sequences of policies)至少与之前的政策一样好。对于政策改进步骤，我们可以使政策在代理人的当前行动值估计方面具有贪婪性。对于政策评估步骤，我们将使用蒙特卡洛方法来估计行动值。

Monte Carlo Generalized Policy Iteration

$$\pi_0 \rightarrow \pi_1 \rightarrow \pi_2 \rightarrow \dots$$



Improvement:

$$\pi_{k+1}(s) \doteq \underset{a}{\text{argmax}}\, q_{\pi_k}(s,a)$$

2) make the policy greedy with respect to the agent's current action value estimates.

Evaluation:

Monte Carlo Prediction

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):

Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0 → exploring start (随机选择 S_0 和 A_0)

Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

Then, the agent generates an episode by following his policy, keeping track of the states, actions, and rewards along the way. Once the episode is complete, it computes each return starting from the end of the episode.

The list of returns are then averaged to update the action value estimates for each state-action pair. This completes the policy evaluation step.

After policy evaluation, it's time to do policy improvement. We simply update the policy to take the greedy action with respect to our updated action values. We do this in every state observed during the episode.

1:24 / 2:32

现在我们有了蒙特卡洛控制的通用策略迭代算法，让我们在一个例子中使用它，看看它是如何工作的。在本视频结束时，你将能够应用Monte Carlo与Exploring Starts来解决一个MDP例子。让我们来学习一个策略。我们将使用Monte Carlo with Exploring Starts来训练我们的代理玩21点。探索开始要求情节从随机状态和行动开始。幸运的是，我们的21点版本自然是以随机状态开始的。然后，我们所要做的就是随机选择每集的第一个动作。也就是说，代理忽略了它认为在第一个状态下的最佳行动，并随机选择了打或粘。最初的政策是，当代理人的总和小于20时，就打，而当总和为20或21时，就坚持。让我们通过一个游戏，看看这个算法会做什么。

假设代理的牌显示总数为13，没有可用的A，庄家可见的牌是8。根据随机抽样的第一个动作，代理打出。代理人得到一张7，使总和达到20。在下一步，代理人根据其政策选择行动。所以它坚持了下来。现在，轮到庄家了。庄家抽到了9，超过了21，输掉了游戏，导致代理人获得了加一的奖励。很好。现在，让我们来看看我们在游戏中看到的状态动作对，从本集的结尾开始，向后看。在最后一个非终端状态中，代理人的总和是20，没有可用的A，而庄家有一张8。从这个状态开始，代理选择了坚持。代理人在其对应于该状态行动对的回报列表中加入加一。在这种状态下，行动坚持的估计值只是1。让我们看看这个状态下的两个动作的价值。代理人从未尝试过hit，所以它的值是零。坚持不出牌的值是1。我们刚刚计算过。所以，在这个状态下，相对于Q来说，贪婪的行动是坚持。它有最高的估计值。让我们再往前走一步，回到起始状态。代理人有13，没有可用的A，而庄家有8。随机选择的行动是击中。同样，代理人在该状态下的行动对之后的回报列表中加一。列表的平均值构成了行动值的估计。最后，在这个状态下，政策被更新为对行动值估计的贪婪。在这个状态下，我们从未尝试过坚持行动，其价值为零。但打的动作导致的回报是1。因此，贪婪的行动是打，政策被更新以反映这一点。然后我们在许多情节中重复这个过程。行动值和政策将接近它们的最优值。让我们看看代理在运行了很长一段时间后发现的最佳策略。注意当代理拥有可用的A时，它是如何玩的。对于大多数庄家的牌，代理一直打到它的总和接近19。有了可用的A，代理在计算其牌的总和方面有更大的灵活性，所以政策更加积极。如果没有可用的A，政策就更多地取决于庄家展示的牌。当代理的总和是13或更大，而庄家有一张低牌，如2或3时，代理就会坚持。这是最理想的，尽管在这种情况下看起来赢的机会会很低。在这个视频中，我们向你展示了如何在一个MDP的例子中应用Monte Carlo with Exploring Starts。

S,A	Returns(S,A)	Q(S,A)		π(S)
		Hit	Stick	
X,Stick	Returns(X,Stick) = [1]	0	1	

∴ the greedy action with respect to Q for this state is Stick.
it has higher estimated value.
→ the randomly chosen action at the beginning of the episode is Stick

Although the agent never tried the hit action, ∴ the value is 0.

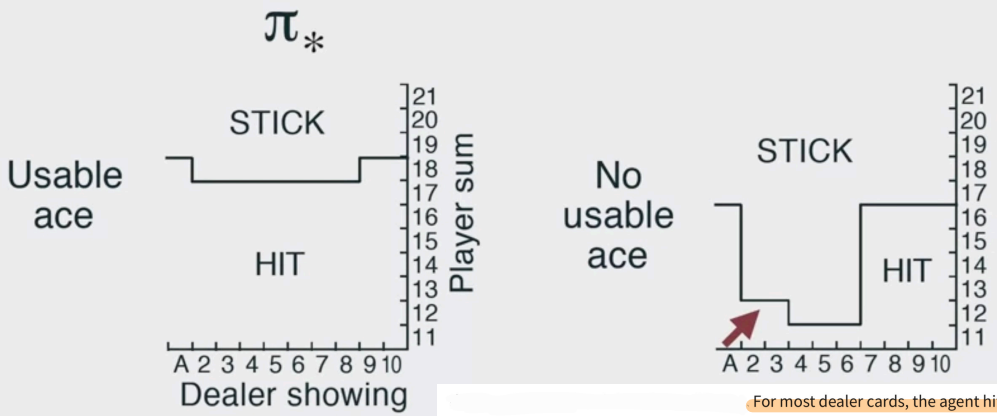
X = (NoAce, 20, 8) ⇒ In this state, the agent chose to stick.

S,A	Returns(S,A)	Q(S,A)		π(S)
		Hit	Stick	
X,Stick	Returns(X,Stick) = [1]	0	1	Stick
Y,Hit	Returns(Y,Hit) = [1]	1	0	

the randomly chosen action at the beginning of the episode is hit.

X = (NoAce, 20, 8)

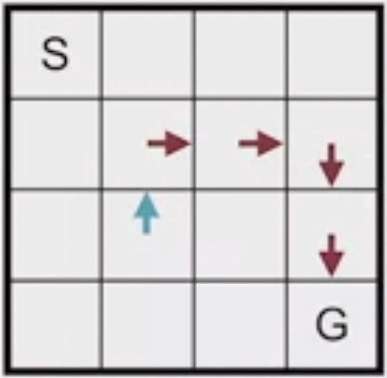
Y = (NoAce, 13, 8)



For most dealer cards, the agent hits until it has the sum near 19. With a usable ace, the agent has a lot more flexibility in calculating the sum of its cards, so the policy is much more aggressive. Without a usable ace, the policy depends a lot more on the cards the dealer is showing. The agent sticks when its sum is 13 or greater and the dealer has a low card, like a two or three. This is optimal even though it seems like the chance of winning will be low in this situation.

We can't always use Exploring Starts

Let's think about some situations where we cannot use exploring starts this algorithm which must be able to start from every possible State action pair. Otherwise the agent may not explore enough and could converge to a suboptimal solution in many problems. It can be difficult to randomly sample an initial State action pair.

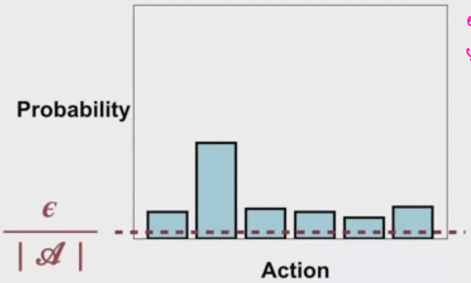


ϵ-Greedy Exploration

=) how can we learn all action values without exploring starts?

ϵ-Greedy policies $\subset$ ϵ-Soft policies

force the agent to continually explore. Always stochastic.

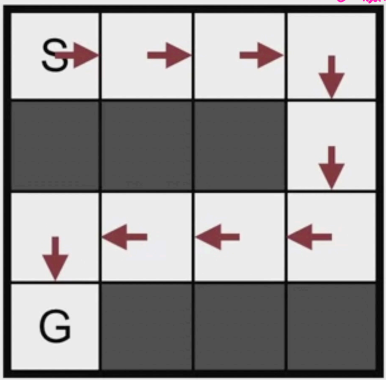


~~Exploring Starts~~

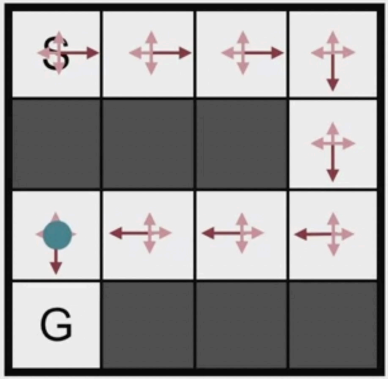
Epsilon soft policies are always stochastic. Deterministic policy specify a single action to take in each state, stochastic policies instead specify the probability of taking action in each state in epsilon. Soft policies: all actions have a probability of at least Epsilon over the number of actions. They will eventually try all the actions.

ϵ-greedy policies and deterministic policies

Deterministic



ϵ-greedy

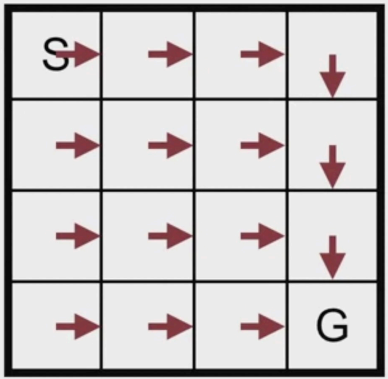


The Epsilon greedy policy has more arrows because every action has some small probability of being selected accordingly. The agent will probably follow a slightly different trajectory every episode.

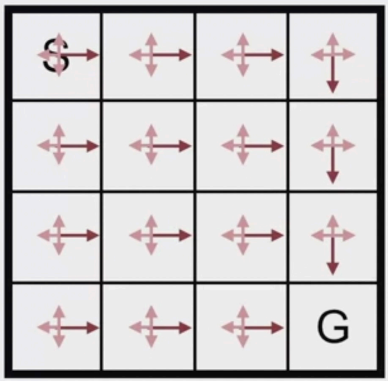
After enough episodes, it will have taken every action at least once in every state

ϵ-soft policies may not be optimal

π_*



Optimal ϵ-soft



If our policy always gives at least Epsilon probability to each action, it's impossible to converge to a deterministic optimal policy exploring starts can be used to find the optimal policy. But ~~that's on~~ soft policies can only be used to find the optimal Epsilon soft policy. That is the policy with the highest value in each state out of all the Epsilon soft policies.

This policy performs worse than the optimal policy in general. However, it often performs reasonably well and allows us to get rid of exploring starts.

MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy 2) one change to the initial conditions.

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ 2) one change to the policy evaluation step.

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$: 2) one change to the policy improvement step

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

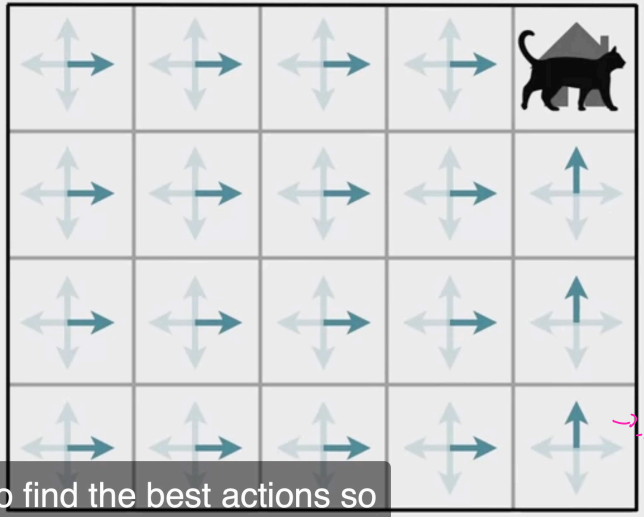
Summary

- We talked about how sampling an initial state-action pair for exploring starts is not always feasible
- We discussed Monte Carlo control with ϵ-soft policies

ϵ-soft policy

Epsilon-soft policies help with the continual exploration Problem by having some small probability of exploring on each time step.

- Values based on suboptimal policy
- Actions based on suboptimal policy



exploring to find the best actions so far throughout this course,

On-Policy and Off-Policy

- On-Policy: improve and evaluate the policy being used to select actions

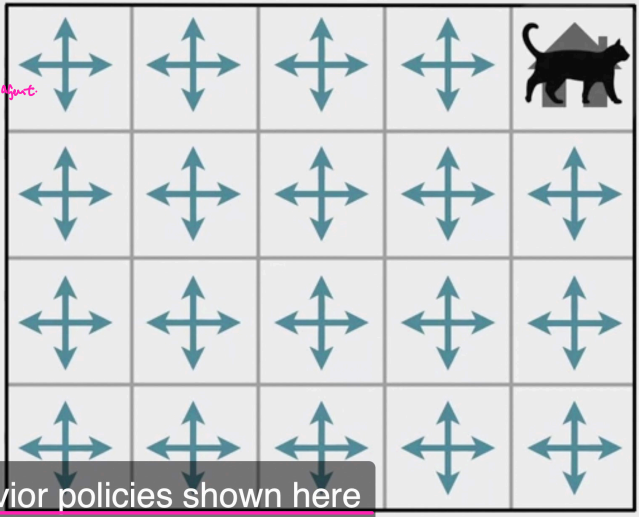
- Off-Policy: improve and evaluate a *different* policy from the one used to select actions

In on policy learning, the agent learns about the policy used to generate the data. What does it mean to learn about a policy and policy evaluation? We simply mean learning the value function in control. We mean learning the optimal policy In off policy learning the agent learns about a policy from data generated by following a different policy. That is the policy that we are learning is off the policy. They we are using for Action selection. For example, you could learn the optimal policy while following a totally random policy we call the policy that the agent is learning the target policy because it is the target of the agents learning the target policy is usually denoted by pie.

Behavior Policy

$b(a \mid s)$ ⇒ select actions for the agent

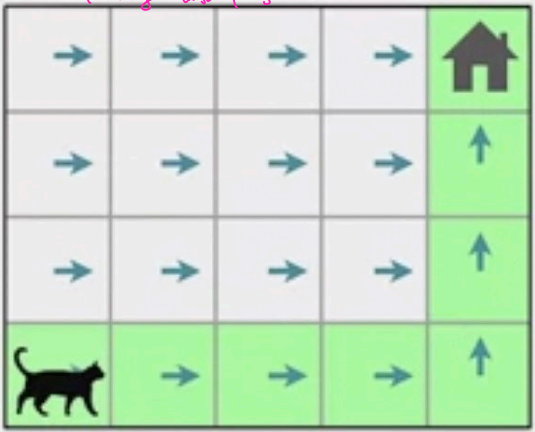
- Select **actions** from this policy
- Generally an **exploratory** policy



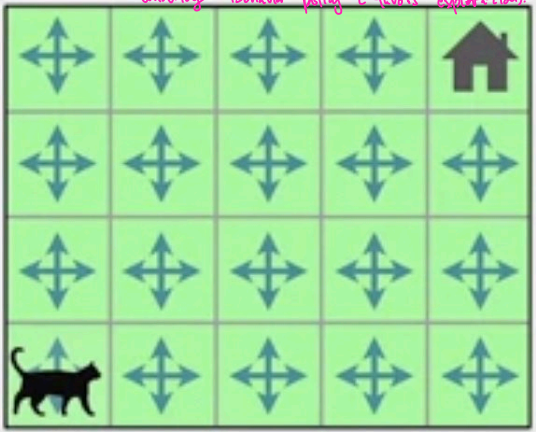
The behavior policies shown here is the uniform random policy.

Continual Exploration

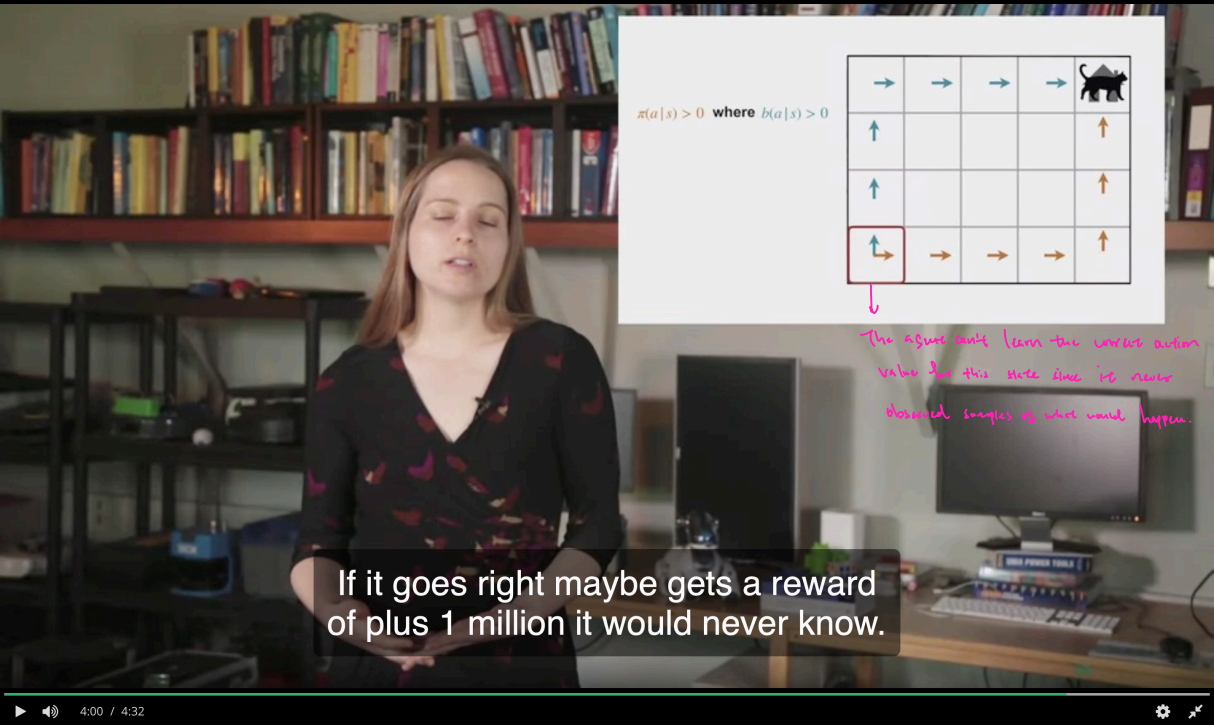
following target policy.



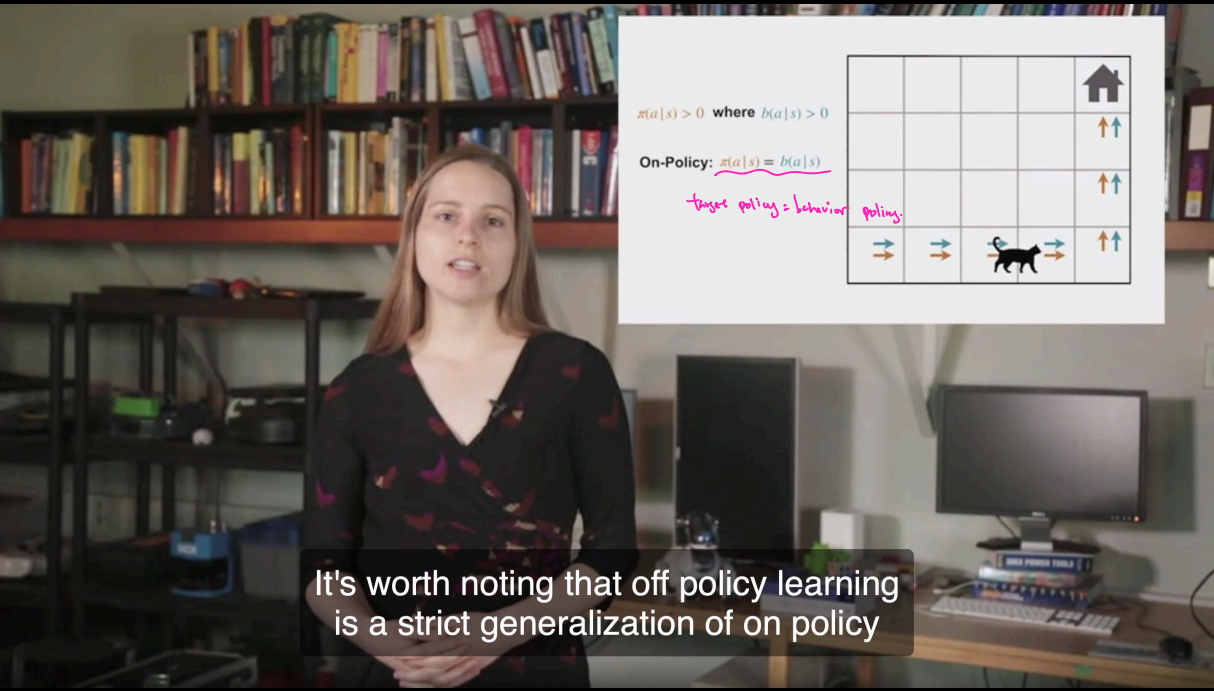
following Behavior policy (favors exploration).



So why are we to coupling the behavior from the target policy? Because it provides another strategy for continual exploration. If our agent behaves according to the Target policy it might only experience a small number of states. If our agent can behave according to a policy that favors exploration, It can experience a much larger number of states.


$\pi(a|s) > 0$ where $b(a|s) > 0$
The agent could learn the correct action value for this state since it never observed samples of what would happen.
If it goes right maybe gets a reward of plus 1 million it would never know.
4:00 / 4:32

One key rule of off policy learning is that the behavior policy must cover the target policy? In other words, if the target policy says the probability of selecting an action a given State s is greater than zero,then the behavior policy must say the probability of selecting that action in that state is greater than 0.there is a key mathematical reason for this that we will discuss in an upcoming video. But there's also an intuitive reason that we show here consider this state with the behavior policy always goes up but the target policy goes to the right,the agent cannot learn the correct action value for that state because a never observed samples of what would happen. If it goes right maybe gets a reward of plus 1 million it would never know.



It's worth noting that off policy learning
is a strict generalization of on policy

Summary

- Off-policy learning allows learning an optimal policy from suboptimal behavior
- The policy that we are learning is the target policy
- The policy that we are choosing actions from is the behavior policy

今天，我们将讨论 重要性采样(importance sampling)。正如我们在后面的视频中所看到的，重要性抽样允许我们进行非政策性学习(off-policy learning)，在遵循一个政策的同时学习另一个政策。在本视频中，我们将首先描述重要性采样的工作原理。

首先，让我们清楚地说明重要性抽样所解决的问题。我们有一些随机变量x正从一个概率分布b中抽样，我们想估计x的期望值，但要相对于目标分布Pi而言。因为x是从b中抽取的，我们不能简单地使用样本平均值来计算Pi下的期望值。这个样本平均数将给我们提供b下的期望值。

Derivation of Importance Sampling

Sample: $x \sim b$ 1) 随机变量 x 从 b 分布采样而来 b中抽样.

Estimate: $\mathbb{E}_{\pi}[X]$ 2) provide the expected value under b instead.

the expected value
under b instead.

让我们先来看看期望值(expected value)的定义。我们把所有可能的结果x的总和乘以根据Pi的概率。接下来，我们可以用x的b除以x的b，因为这个项等于1。x的b是观察到的结果x在b下的概率。我们通常把重要性抽样比率写成x的Rho。现在，让我们再进一步。如果我们把x乘以重要性抽样比率作为一个新的随机变量，乘以观察到x的概率，我们就可以把这个和改写成b下的期望值。我们知道如何使用重要性抽样来修正期望值，但我们如何使用它来估计数据的期望值呢？这其实很简单。我们只需要计算一个加权的样本平均值，以重要性抽样的比率作为权重，如幻灯片所示。注意，这些样本x_i是从b中抽取的，而不是Pi。这就是了。现在我们可以通过使用样本平均值，即从分布b中抽取的样本来估计分布Pi下的x的期望值。 让我们看一下使用重要性抽样来估计期望值的一个小例子。我们有两个相当不同的分布：b和Pi。我们将根据b抽取样本并试图估计Pi下的期望值。在右边，我们将跟踪我们当前对Pi下期望值的估计。作为参考，我们在线的中间位置显示真实的期望值。我们从b中抽取一个样本，得到x等于1，发生的概率为0.85。从Pi得到x等于1的概率是0.3。为了计算Pi的样本平均值，我们需要我们的重要性抽样公式。将观察到的数值插入，我们得到一个0.35的估计值。让我们再做一次。我们从b中抽取一个样本，这次x等于3，发生的概率为0.05。如果我们从Pi中取样，我们在3处看到的概率是0.1。我们将x添加到缓冲区，然后将观察到的值插入我们的公式中。我们把之前的估计值加上新的估计值，然后除以2，得到3.18。最后一个样本。我们再从b中抽取一个，并将其加入缓冲区，我们将观察到的数值插入重要性抽样比率公式中，得到2.2。仅仅从b中抽取样本，我们就能得到Pi下的期望值的一个相当好的估计。

Derivation of Importance Sampling

$$\mathbb{E}_{\pi}[X] \doteq \sum_{x \in X} x\pi(x)$$

$$= \sum_{x \in X} x\pi(x)\frac{b(x)}{b(x)}$$

$$= \sum_{x \in X} x\frac{\pi(x)}{b(x)}b(x)$$

Importance sampling ratio

Derivation of Importance Sampling

$$\mathbb{E}_{\pi}[X] \doteq \sum_{x \in X} x\pi(x)$$

$$= \sum_{x \in X} x\pi(x)\frac{b(x)}{b(x)}$$

$$= \sum_{x \in X} x\rho(x)b(x)$$

Derivation of Importance Sampling

$$\mathbb{E}_{\pi}[X] = \sum_{x \in X} x\rho(x)b(x)$$
$$= \mathbb{E}_{b}\left[X\rho(X)\right]$$

Derivation of Importance Sampling

$$\mathbb{E}[X] \approx \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\mathbb{E}_b\left[X\rho(X)\right] = \sum_{x \in X} x\rho(x)b(x)$$

$$\approx \frac{1}{n}\sum_{i=1}^{n} x_i\rho(x_i)$$

$$x_i \sim b$$

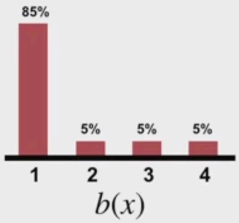
x_i は b から取ってくる、ということ

Derivation of Importance Sampling

$$\rho(x) = \frac{\pi(x)}{b(x)}$$

$$\mathbb{E}_{\pi}[X] \approx \frac{1}{n} \sum_{i=1}^{n} x_i \rho(x_i)$$

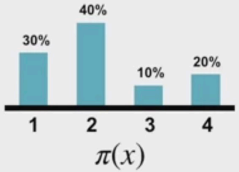
$$x_i \sim b$$



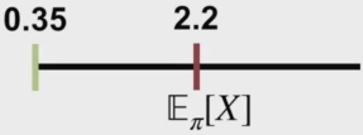
$$x = 1$$

$$b(x) = .85$$

x = [1]

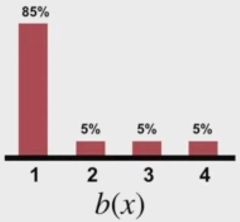


$$\pi(x) = .3$$



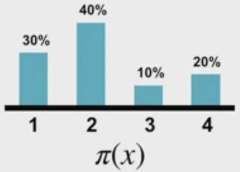
$$\frac{1}{n}\sum_{1}^{n} x\rho(x) \longrightarrow 1 \times \frac{.3}{.85} = 0.35$$

$$\rho(x) = \frac{\pi(x)}{b(x)}$$



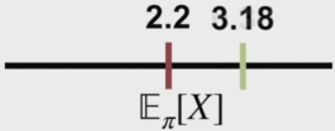
$$x = 3$$
$$b(x) = .05$$

x = [1, 3]



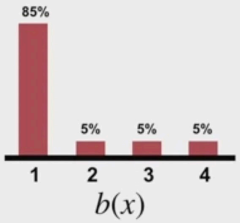
$$\pi(x) = .1$$

the previous estimate for X=1



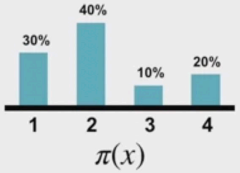
$$\frac{1}{n}\sum_{1}^{n} x\rho(x) \longrightarrow \frac{(1 \times \frac{.3}{.85}) + (3 \times \frac{.1}{.05})}{2} = 3.18$$

the estimate now for X=3

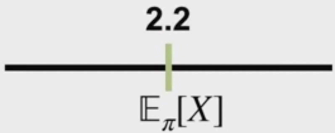


$$x = 1$$
$$b(x) = .85$$

x = [1, 3, 1]



$$\pi(x) = .3$$



$$\frac{1}{n}\sum_{1}^{n} x\rho(x) \longrightarrow \frac{(1 \times \frac{.3}{.85}) + (3 \times \frac{.1}{.05}) + (1 \times \frac{.3}{.85})}{2} = 2.24$$

Summary

- Importance sampling uses samples from one probability distribution to estimate the expectation of a *different* distribution

Graded Quiz

最新提交作业的评分 100%

1. 1 / 1 分

Which approach ensures continual (never-ending) exploration? **(Select all that apply)**

- [x] Exploring starts

  正确
  Correct! Exploring starts guarantee that all state-action pairs are visited an infinite number of times in the limit of an infinite number of episodes.

- [ ] On-policy learning with a **deterministic** policy

- [x] On-policy learning with an ϵ-soft policy

  正确
  Correct! ϵ-soft policies assign non-zero probabilities to all state-action pairs.

- [x] Off-Policy learning with an ϵ-soft behavior policy and a **deterministic** target policy

  正确
  Correct! ϵ-soft policies have non-zero probabilities for all actions in all states. The behavior policy is used to generate samples and should be exploratory.

- [ ] Off-Policy learning with an ϵ-soft target policy and a **deterministic** behavior policy

2. When can Monte Carlo methods, as defined in the course, be applied? (Select all that apply) 1 / 1 分

- [ ] When the problem is **continuing** and there are sequences of states, actions, and rewards
- [ ] When the problem is **continuing** and there is a model that produces samples of the next state and reward
- [x] When the problem is **episodic** and there are sequences of states, actions, and rewards

正确
Correct! Well-defined returns are available in episodic tasks.

- [x] When the problem is **episodic** and there is a model that produces samples of the next state and reward

正确
Correct! Well-defined returns are available in episodic tasks.

3. Which of the following learning settings are examples of off-policy learning? (Select all that apply) 1 / 1 分

- [x] Learning the optimal policy while continuing to explore

正确
Correct! An off-policy method with an exploratory behavior policy can assure continual exploration.

- [x] Learning from data generated by a human expert

正确
Correct! Applications of off-policy learning include learning from data generated by a non-learning agent or human expert. The policy that is being learned (the target policy) can be different from the human expert's policy (the behavior policy).

4. Which of the following is a requirement for using Monte Carlo policy evaluation with a behavior policy b for a target policy π? 1 / 1 分

- (●) For each state s and action a, if $\pi(a \mid s) > 0$ then $b(a \mid s) > 0$
- ( ) All actions have non-zero probabilities under π
- ( ) For each state s and action a, if $b(a \mid s) > 0$ then $\pi(a \mid s) > 0$

正确
Correct! Every action taken under π must have a non-zero probability under b.

5. When is it possible to determine a policy that is greedy with respect to the value functions v_π, q_π for the policy π? (Select all that apply) 1 / 1 分

- [x] When state values v_π and a model are available

正确
Correct! With state values and a model, one can look ahead one step and see which action leads to the best combination of reward and next state.

- [ ] When state values v_π are available but no model is available.
- [x] When action values q_π and a model are available

正确
Correct! Action values are sufficient for choosing the best action in each state.

- [x] When action values q_π are available but no model is available.

正确
Correct! Action values are sufficient for choosing the best action in each state.

6. Monte Carlo methods in Reinforcement Learning work by... 1 / 1 分

Hint: recall we used the term "sweep" in dynamic programming to discuss updating all the states systematically. This is not the same as visiting a state.

- ○ Performing **sweeps** through the state set
- ○ **Planning** with a model of the environment
- ◉ Averaging sample returns
- ○ Averaging sample rewards 对!! This is what reinforcement learning doing !!

✓ 正确
Correct! Monte Carlo methods in Reinforcement Learning sample and average returns much like bandit methods sample and average rewards.

7. Suppose the state s has been visited three times, with corresponding returns 8, 4, and 3. What is the current Monte Carlo estimate for the value of s? 1 / 1 分

- ○ 3
- ○ 15
- ◉ 5
- ○ 3.5

$\frac{8+4+3}{3} = \frac{15}{3} = 5$

✓ 正确
Correct! The Monte Carlo estimate for the state value is the average of sample returns observed from that state.

8. When does Monte Carlo prediction perform its first update? 1 / 1 分

- ○ After the first time step
- ○ After every state is visited at least once
- ◉ At the end of the first episode

正确
Correct! Monte Carlo Prediction updates value estimates at the end of an episode.

9. For Monte Carlo Prediction of state-values, the number of **updates** at the end of an episode depends on. 1 / 1 分

Hint: look at the innermost loop of the algorithm.

- ○ The number of states
- ○ The number of possible actions in each state
- ◉ The length of the episode

正确
Correct! Monte Carlo Prediction updates the estimated value of each state visited during the episode.

10. In an ϵ-greedy policy over $\mathcal{A}$ actions, what is the probability of the highest valued action if there are no other actions with the same value?

1 / 1 分

$1-\epsilon$: explore

ϵ : greedy.

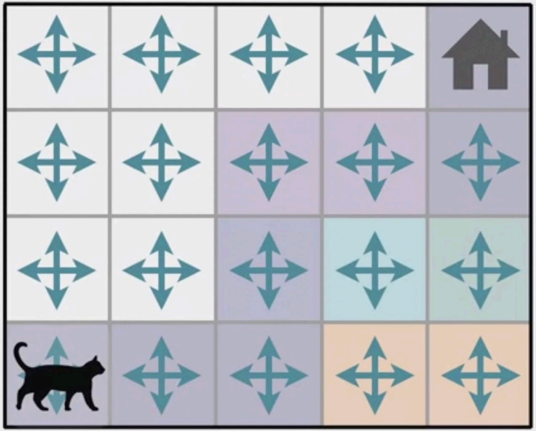
- ○ $1 - \epsilon$
- ○ ϵ
- ◉ $1 - \epsilon + \frac{\epsilon}{\mathcal{A}}$
- ○ $\frac{\epsilon}{\mathcal{A}}$

正确

Correct! The highest valued action still has a chance of being selected as an exploratory action.

在最近的视频中，我们已经讨论了非政策性学习。在这个视频中，我们将看到如何用蒙特卡洛做非政策性预测。回顾一下蒙特卡洛估计的目标。我们希望通过计算从该状态开始的回报的样本平均值来估计每个状态的价值。

Recall: Monte Carlo



Sample average over returns starting from that state.

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s]$$

$$\approx average(\; Returns[0],\; Returns[1],\; Returns[2],\;)$$

不同颜色代表不同的 state.

然而，当我们试图用行为策略b下的回报来估计目标策略Pi下的价值时，我们遇到了一个问题。如果我们简单地对回报进行平均，我们看到在行为b下的状态s，我们不会得到正确的答案。我们必须纠正平均数中的每个回报。这正是重要抽样的目的。我们所要做的就是找出每个抽样回报的Rho值。在这里，Rho是Pi下轨迹的概率除以b下轨迹的概率。这个Rho修正了整个轨迹的分布，因此也修正了回报的分布。利用这个修正，我们就能得到我们想要的东西，即Pi下的收益期望值。为了计算Rho，我们需要弄清楚如何计算一个政策下的轨迹的概率。让我们考虑轨迹的概率分布。我们把这个概率理解为，给定代理人在某个状态下为t，那么它采取行动A_t然后结束在状态S_t加1的概率是多少，然后，它采取行动A_t加1并结束在S_t加2，以此类推，直到在时间T终止？所有的行动都是根据行为b来取样的。

由于马尔可夫特性，我们可以把这个概率分布分成小块。第一块是代理人在状态S_t下喜欢行动A_t的概率乘以环境转入状态S_t的概率加1。第二块给出了下一个时间步长的概率，以此类推。我们可以用乘积符号重写这个乘积概率的列表。现在，我们已经定义了b下轨迹的概率，记住我们要去的地方。我们想用Pi下轨迹的概率和b下轨迹的概率来定义Rho，让我们把这些概率插入Rho的定义中。正如我们在之前的视频中看到的，我们可以把这些概率乘以重要性采样率。环境的过渡动态在每个时间步骤上都会抵消。这就使我们只有一个在每个时间步长的政策之间的比率的乘积。现在，让我们回到估计政策外的V_Pi。代理人观察到许多回报，每一个都是根据行为政策b。我们可以用这些回报来估计V_Pi，用Rho修正每个回报。现在，让我们来看看我们将如何实现这一点。

Off-Policy Monte Carlo

$$V_\pi(s) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right]$$

~~average(~~

$$\begin{aligned} &Returns[0], \\ &Returns[1], \quad \longleftarrow a \sim b \\ &Returns[2], \\ &) \end{aligned}$$

Off-Policy Monte Carlo

$$
\begin{aligned}
V_\pi(s) &\doteq \mathbb{E}_\pi\big[G_t \,|\, S_t = s\big] \\
&\approx average(\\
&\qquad \rho_0 Returns[0], \\
&\qquad \rho_1 Returns[1], \\
&\qquad \rho_2 Returns[2], \\
&\qquad \ldots)
\end{aligned}
$$

ρ corrects the trajectories over the entire trajectory, and to correct the distribution over returns.

Off-Policy Monte Carlo

$$\rho = \frac{\mathbb{P}(\textbf{trajectory under } \pi)}{\mathbb{P}(\textbf{trajectory under } b)}$$

$$V_\pi(s) = \mathbb{E}_b\left[\rho G_t \,|\, S_t = s\right]$$

Off-Policy Trajectories

Actions sampled according to behavior b

take action A_t then ended up in state S_{t+1}

take action A_{t+1} then ended up in state S_{t+2}

$$P(A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T})$$

Off-Policy Trajectories

$$P(A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T}) =$$

$$b(A_t \mid S_t)\, p(S_{t+1} \mid S_t, A_t)\, b(A_{t+1} \mid S_{t+1})\, p(S_{t+2} \mid S_{t+1}, A_{t+1}) \ldots p(S_T \mid S_{T-1}, A_{T-1})$$

Off-Policy Trajectories

$$\mathbb{P}(\textbf{trajectory under } b) = \prod_{k=t}^{T-1} b(A_k \mid S_k)\, p(S_{k+1} \mid S_k, A_k)$$

$$\rho_{t:T-1} \doteq \frac{\mathbb{P}(\textbf{trajectory under } \pi)}{\mathbb{P}(\textbf{trajectory under } b)}$$

$$\doteq \prod_{k=t}^{T-1} \frac{\pi(A_k \mid S_k)\, p(S_{k+1} \mid S_k, A_k)}{b(A_k \mid S_k)\, p(S_{k+1} \mid S_k, A_k)}$$

the 'important sampling ratio

Off-Policy Value

$$\rho_{t:T-1} \doteq \prod_{k=t}^{T-1} \frac{\pi(A_k \mid S_k)}{b(A_k \mid S_k)}$$

$$\mathbb{E}_b[\rho_{t:T-1} G_t \mid S_t = s] = v_\pi(s)$$

Off-policy every-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in S$

$Returns(s) \leftarrow$ an empty list, for all $s \in S$

Loop forever (for each episode):

Generate an episode following b: $S_0, A_0, R_1, S_1 \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0 \quad W \leftarrow 1$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$

$G \leftarrow \gamma W G + R_{t+1}$

Append G to $Returns(S_t)$

$V(S_t) \leftarrow$ average$(Returns(S_t))$

$W \leftarrow W \dfrac{\pi(A_t \mid S_t)}{b(A_t \mid S_t)}$ => W is the accumulating product of the importance sampling ratio on each time step of the episode.

Off-policy every-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, **arbitrarily, for all** $s \in S$

$Returns(s) \leftarrow$ **an empty list, for all** $s \in S$

Loop forever (for each episode):

Generate an episode following $b : S_0, A_0, R_1, S_1 \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$ $W \leftarrow 1$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$

$G \leftarrow \gamma W G + R_{t+1}$

Append G **to** $Returns(S_t)$

$V(S_t) \leftarrow$ **average**$(Returns(S_t))$

$$W \leftarrow W \frac{\pi(A_t \mid S_t)}{b(A_t \mid S_t)}$$

Computing $\rho_{t:T-1}$ incrementally

$$\rho_{t:T-1} \doteq \prod_{k=t}^{T-1} \frac{\pi(A_k \mid S_k)}{b(A_k \mid S_k)}$$

$$= \rho_t \rho_{t+1} \rho_{t+2} \cdots \rho_{T-2} \rho_{T-1}$$

$$\longleftarrow$$

$$W_1 \leftarrow \rho_{T-1}$$ → in first time step.

$$W_2 \leftarrow \rho_{T-1}\rho_{T-2}$$ → in second time step ... 以此类推.

$$W_3 \leftarrow \rho_{T-1}\rho_{T-2}\rho_{T-3}$$

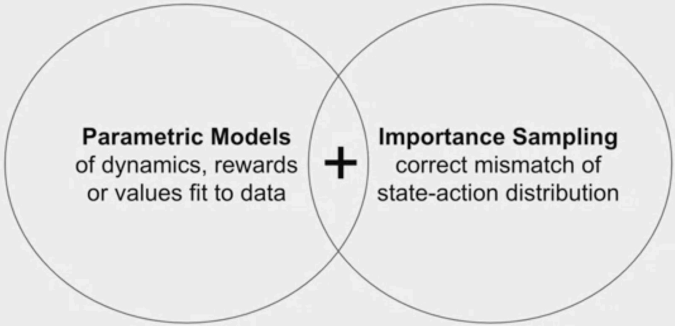
$$W_{t+1} \leftarrow W_t \rho_t$$

Summary

- Used importance sampling ratios to correct the returns
- Modified the on-policy Monte Carlo prediction algorithm for off-policy learning

首先，回顾一下 政策上的蒙特卡洛预测算法(on-policy Monte Carlo prediction algorithm)。这里需要做两个改变。首先，剧情不会在Pi之后产生，而是在b之后产生。其次，需要用重要的抽样比率的乘积来校正回报。通过这些改变，我们最终得到了 非政策性的蒙特卡洛预测算法(off-policy Monte Carlo prediction algorithm)。注意现在的情节是按照行为政策产生的。回报被一个新的项W修正，它是该事件每个时间步骤上重要抽样比率的累积产物。我们可以逐步计算从t到T减1的Rho。为了了解原因，让我们写出每个时间步骤的乘积。回想一下，蒙特卡洛算法是在时间步数上向后循环的。所以在算法的第一步，W被设置为最后一个时间步骤的Rho。在下一个时间步骤中，W是最后一个Rho乘以最后一个Rho，以此类推。每个时间步骤都会给乘积增加一个额外的项，并重复使用之前的所有项。我们可以递归地计算，而不需要存储所有过去的Rho值。这段视频就到此为止。在这段视频中，我们使用了重要的抽样比率来纠正行为政策产生的回报，我们修改了政策上的蒙特卡洛预测算法，用于非政策学习。

Doubly Robust



\+ Smaller variance than importance sampling
\+ Unbiased if either model realizable or behavior policy known

- Brought to multi-armed bandits (Dudik et al. 2011); and RL (Jiang & Li 2016), Lower variance extensions include (Thomas & Brunskill 2016)

Quest for Counterfactual/Batch Policy Optimization with Generalization Guarantees

$$\underbrace{\arg\max_{\pi \in H_i} \quad \max_{H_i \in \{H_1, H_2, \ldots\}}}_{\text{Policy Optimization}} \qquad \underbrace{\int_{s \in S_0} \hat{V}^{\pi}(s, \mathcal{D}) ds}_{\text{Policy Evaluation}} \quad - \quad \underbrace{\sqrt{\frac{f(VC(h), \ldots)}{n}}}_{\text{Error Bound}}$$

AAAI 2015, AAAI 2016, L@S 2017, UAI 2017, UAI 2019, (Nie, Brunskill, Wagner arxiv)

AAMAS 2014, AAAI 2015, AAAI 2016, ICML 2016, IJCAI 2016, AAAI 2017, NeurIPS 2017, NeurIPS 2018, ICML 2019

Nie, B, Wagner arxiv; Thomas, da Silva, Barto, Brunskill, arxiv

& many colleagues' work (Mannor, Ghavamzadeh, Thomas, Murphy, Jiang, Yue, Munos, Lazaric, Szepesvari, White, Sutton, Whiteson…)

干得好，你已经度过了第一周的讲座。这周的内容是 估计价值函数(estimating value functions)，并仅利用环境中的经验找到最佳政策。我们看到了 政策性(on-policy) 和 非政策性(off-policy) 的蒙特卡洛，并重温了土匪的探索问题。让我们回顾一下到目前为止的材料。

蒙特卡洛算法是基于样本的方法。当模型不可用或很难写下来时，它们可以被使用。蒙特卡洛算法通过对多个观察到的回报进行平均化来估计价值函数。它们在更新其数值之前会等待全部的回报。因此，我们只对偶发的MDPs使用蒙特卡洛。我们讨论了蒙特卡洛如何在广义的策略迭代中使用。这导致了我们第一个基于样本的控制算法，即带有探索性启动的蒙特卡洛(Monte Carlo with exploring starts)。蒙特卡洛算法不像动态编程那样对状态行动空间进行扫描，所以它们需要一种探索机制来确保它们了解每一个状态行动对。我们首先考虑了探索性开始。探索开始要求在每集开始时随机选择第一个状态和行动。使用探索性开始并不总是可行或安全的。试想一下，试图用一辆自动驾驶汽车做探索式启动。这种认识促使我们去研究其他的探索方法。

Monte Carlo algorithms are sample-based methods. They can be used when the model is unavailable or hard to write down. Monte Carlo algorithms estimate value functions by averaging over multiple observed returns. They wait for the full return before updating their values. Therefore, we use Monte Carlo only for episodic MDPs.

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):

Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0 =) exploration starts.

Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Append G to $Returns(S_t, A_t)$

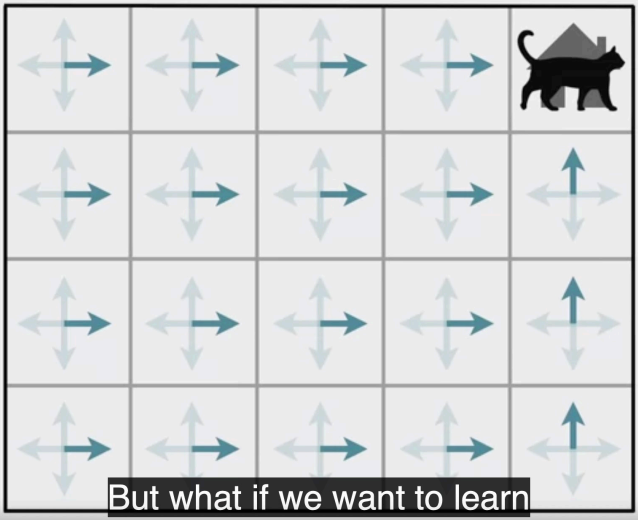
$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

or safe to use exploring starts.

我们涵盖了探索问题的另外两种策略。用Epsilon-soft政策进行政策学习(on policy learning) 和 非政策学习(off-policy learning)。对于第一个策略，代理人遵循并学习随机政策。它通常采取贪婪的行动。有一小部分时间它采取随机行动。这样一来，所有状态行动对的价值估计就能保证随着时间的推移不断提高。这种关于政策的策略迫使我们学习一个接近最优的政策，而不是一个最优的政策。但是，如果我们想学习一个最优策略，但仍然保持探索，该怎么办？答案就在于非政策性学习。我们为非政策性学习引入了一些新的定义，让我们回顾一下。行为政策是代理人用来选择行动的政策。目标政策是代理人在其价值函数中学习的政策。通过发送适当的探索性行为政策，代理人可以学习任何确定的目标政策。学习一种政策而放弃另一种政策的一种方法是使用重要性抽样。重要性抽样允许代理人从行为政策下的经验抽样中估计出目标政策下的预期收益。该比率对样本进行了重新加权。它增加了在Pi下更有可能出现的回报的重要性，减少了那些不太可能出现的回报。样本平均数有效地包含了每个回报的正确比例，因此在预期中，就像在Pi下抽样的回报一样。

Epsilon-soft policies



But what if we want to learn

We cover two other strategies for the exploration problem. Learning on policy with Epsilon-soft policies and learning off-policy. For the first strategy, the agent follows and learns about a stochastic policy. It usually takes the greedy action. A small fraction of the time it takes a random action. This way the value estimates for all state action pairs are guaranteed to continue to improve over time. This on policy strategy forced us to learn a near optimal policy instead of an optimal policy. But what if we want to learn an optimal policy but still maintain exploration? The answer lies with off-policy learning.

Importance Sampling

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \\
&\approx average(\\
&\qquad \rho_0 Returns[0], \\
&\qquad \rho_1 Returns[1], \\
&\qquad \rho_2 Returns[2], \\
&\quad)
\end{aligned}
$$

$$
\rho = \frac{\mathbb{P}(\textbf{trajectory under } \pi)}{\mathbb{P}(\textbf{trajectory under } b)}
$$

importance sampling. Importance sampling allows the agent to estimate the expected return under the target policy from experience sampled under the behavior policy. The ratio re-weights the samples. It increases the importance of returns that were more likely to be seen under Pi and it decreases those that were unlikely. The sample average effectively contains the right proportion of each return so that in in expectation it is as if the returns had been sampled under Pi. That's all for this module. Next we'll talk about one of the most fundamental concepts in