

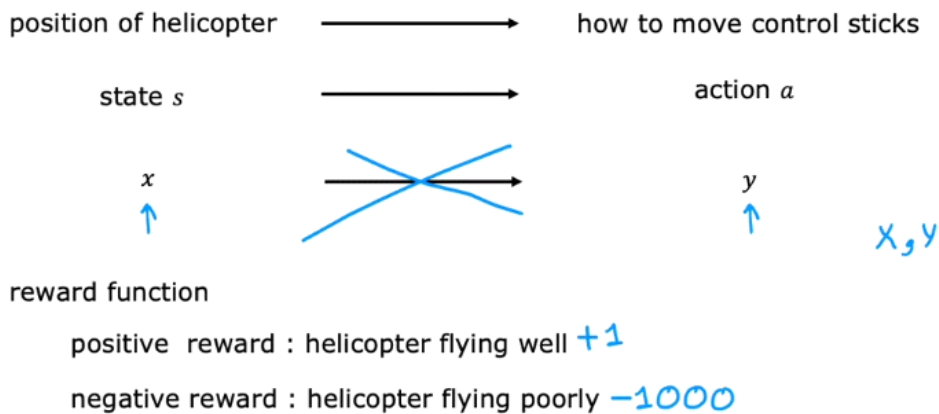
Reinforcement Learning

2023年5月31日 9:02

1. 强化学习简介

关键思想是利用reward function给予算法激励，从而让算法自己寻找每一种状态下合理的行动，而非自己告诉算法什么是合适的输出

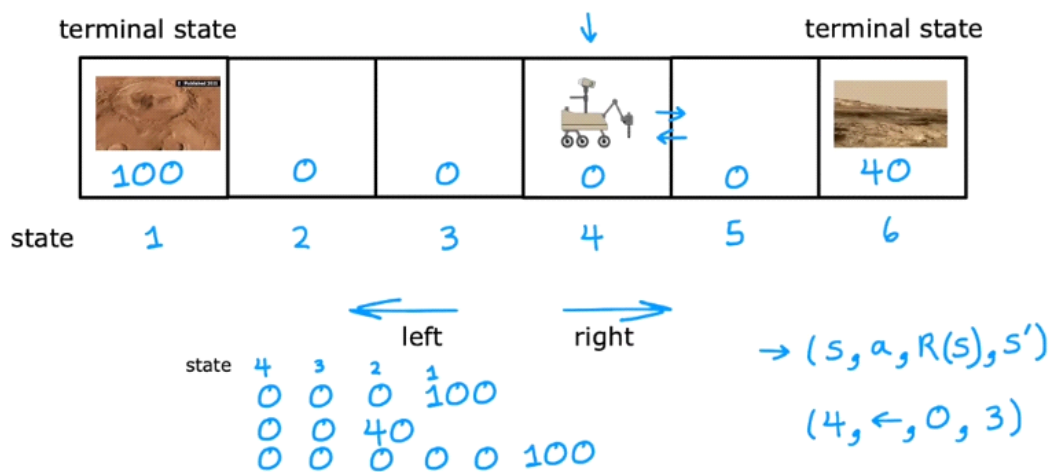
Reinforcement Learning



屏幕剪辑的捕获时间: 2023/6/11 16:50

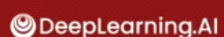
四个要素: state、action、reward、next state

Mars Rover Example



[Credit: Jagriti Agrawal, Emma Brunskill]

在决定如何采取行动时看看。



look at when deciding how to take actions.

Andrew Ng

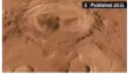


屏幕剪辑的捕获时间: 2023/6/12 20:05

2. 强化学习中的"回报"(Return)

引入一个折减系数 γ ，使算法越往后，得到的reward显得价值越小

一般 γ 会选择很接近1的数字（比1略小）， γ 越大说明算法越“不耐烦”，即每多走一步都会让回报更快下降

Return

					
100	0	0	0	0	40
state 1	2	3	4	5	6

$\text{Return} = 0 + (0.9)0 + (0.9)^2 0 + (0.9)^3 100 = 0.729 \times 100 = 72.9$
 $\text{Return} = R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$ (until terminal state)
 Discount Factor $\gamma = 0.9$ 0.99 0.999
 $\gamma = 0.5$
 $\text{Return} = 0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100 = 12.5$

到了最终状态，结果是 12.5 的回报。

and this turns out to be a return of 12.5.

DeepLearning.AI

Stanford University

Andrew Ng

屏幕剪辑的捕获时间: 2023/6/12 20:14

Example of Return

100	50	25	12.5	6.25	40	$\gamma = 0.5$
100	0	0	0	0	40	
1	2	3	4	5	6	

← return
← reward

The return depends on the actions you take.

100	2.5	5	10	20	40	$0 + (0.5)0 + (0.5)^2 40 = 10$
100	0	0	0	0	40	
1	2	3	4	5	6	

100	50	25	12.5	20	40	$0 + (0.5)40 = 20$
100	0	0	0	0	40	
1	2	3	4	5	6	

20 如果您采取此处显示的操作。

20 if you take actions shown here.

DeepLearning.AI

Stanford University

Andrew Ng







屏幕剪辑的捕获时间: 2023/6/12 20:18

强化学习的回报是根据折减系数折减后的奖励总和，其中折减系数的幂次随步数增加而变大

如果引入负的reward，算法会倾向于使负收益推后

3. 强化学习中的一些概念整合

强化学习的目标是找到一个策略(policy)，来指导算法对于每一个状态(state)，该采取何种行动(action)，使得回报(return)最大化

	Mars rover 	Helicopter 	Chess 
states	6 states	position of helicopter	pieces on board
actions	\longleftrightarrow	how to move control stick	possible move
rewards	100, 0, 40	+1, -1000	+1, 0, -1
discount factor γ	0.5	0.99	0.995
return	$R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$	$R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$	$R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$
policy π		Find $\pi(s) = a$ 	Find $\pi(s) = a$ 

这种强化学习应用程序的形式实际上有

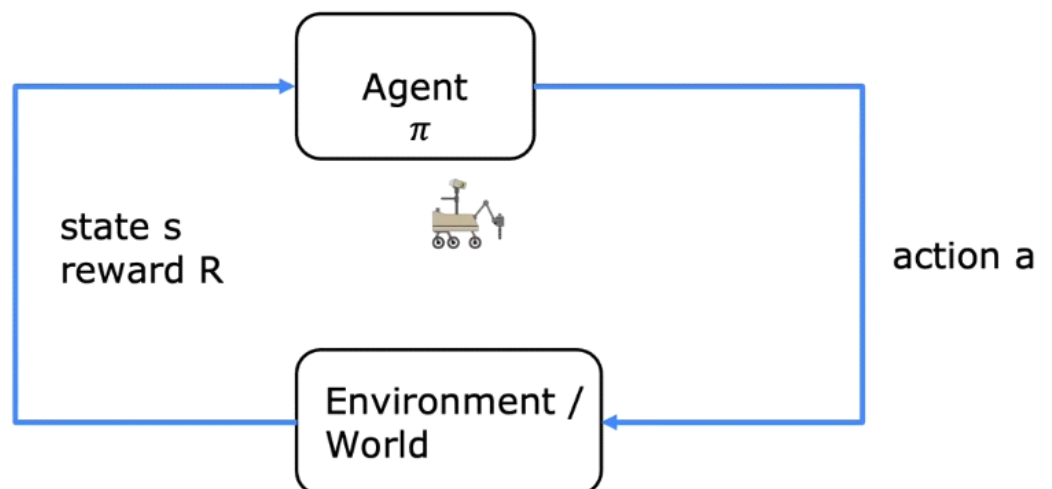
DeepLearning.AI This formalism of a reinforcement learning application Andrew Ng

屏幕剪辑的捕获时间: 2023/6/12 21:31

马尔可夫决策过程(Markov Decision Process)

是指未来只取决于当前状态，而非任何前置过程

Markov Decision Process (MDP)



您有时会看到不同的作者使用这样的图

DeepLearning.AI You sometimes see different authors use a diagram like Andrew Ng

屏幕剪辑的捕获时间: 2023/6/12 21:35

4.动作状态价值函数(State action value function)

将这个函数记为 $Q(s,a)$,其中 s 指状态, a 指行动,这个函数返回一个回报值,这个值是指在状态 s 下采取行动 a 之后,以后每一步如果都按照最佳策略行动,最终会得到的回报值

State action value function (Q-function)

$Q(s, a)$ = Return if you

- start in state s .
- take action a (once).
- then behave optimally after that.

$Q(s, a)$
↑ ↑

100	50	25	12.5	20	40
100	0	0	0	0	40

← return
← action
← reward

$Q(2, \leftarrow)$ $Q(2, \rightarrow)$

100	50	12.5	25	6.25	12.5	10	6.25	20	40	40
100	0	0	0	0	0	0	0	0	0	40
1	2	3	4	5	6					

$$Q(2, \rightarrow) = 12.5$$

$$0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100$$

$$Q(2, \leftarrow) = 50$$

$$0 + (0.5)100$$

$$Q(4, \leftarrow) = 12.5$$

$$0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100$$

什么是你的回报或真正的价值是什么?

it tells you what are your returns or really what is the value? Andrew Ng

屏幕剪辑的捕获时间: 2023/6/13 9:21

状态 s 下的最大回报是这个状态下 Q function 的最大值; 所以状态 s 下的最佳行动就是使得 Q function 最大的行动

5. 贝尔曼方程

Bellman Equation

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

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

100	100	50	12.5	25	6.25	12.5	10	6.25	20	40	40
100	0	0	0	0	0	0	0	0	0	0	40
1	2	3	4	5	6						

$s = 2$
 $a = \rightarrow$
 $s' = 3$

$$Q(2, \rightarrow) = R(2) + 0.5 \max_{a'} Q(3, a')$$

$$= 0 + (0.5)25 = 12.5$$

$$Q(4, \leftarrow) = R(4) + 0.5 \max_{a'} Q(3, a')$$

$$= 0 + (0.5)25 = 12.5$$

$s = 4$
 $a = \leftarrow$
 $s' = 3$

素数, 所以第二项会消失。

no state S prime and so that second term would go away. Andrew Ng

屏幕剪辑的捕获时间: 2023/6/14 11:44

其中 s' 指下一个状态, a' 指下一个状态中采取的行动

Explanation of Bellman Equation

$Q(s, a)$ = Return if you

- start in state s .
- take action a (once).
- then behave optimally after that.

$s \rightarrow s'$

→ The best possible return from state s' is $\max_{a'} Q(s', a')$

$$Q(s, a) = \underbrace{R(s)}_{\text{Reward you get right away}} + \underbrace{\gamma \max_{a'} Q(s', a')}_{\text{Return from behaving optimally starting from state } s'}$$

$$Q(s, a) = R_1 + \gamma [R_2 + \gamma R_3 + \gamma^2 R_4 + \dots]$$

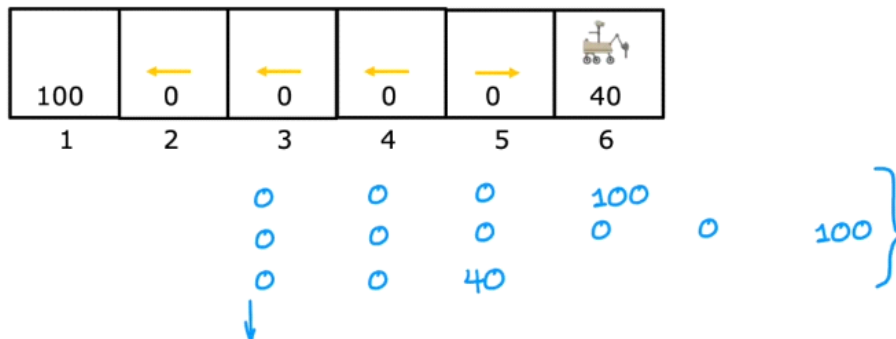
屏幕剪辑的捕获时间: 2023/6/14 11:48

贝尔曼方程相当于把总的回报分解为当前状态立即可得的Reward与下一个状态下可获得的回报之和，即分解为当下与未来的回报之和

6. 随机环境

期望回报：随机环境下，算法每一个状态下的实际行动可能因为外界干扰等因素而与指令不同，这种情况下无法确实给出最大回报，所以需要有一个Policy使得期望回报最大

Expected Return



$$\text{Expected Return} = \text{Average}(R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \dots)$$

$$= E[R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \dots]$$

屏幕剪辑的捕获时间: 2023/6/14 12:04

Expected Return

Goal of Reinforcement Learning:

Choose a policy $\pi(s) = a$ that will tell us what action a to take in state s so as to maximize the expected return.

Bellman Equation:

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

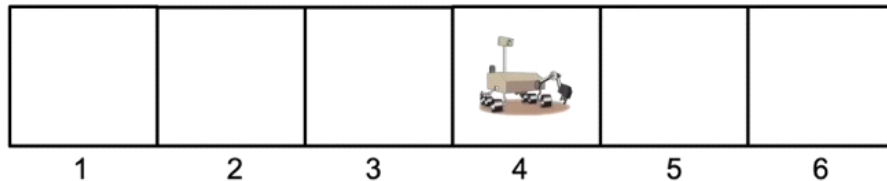
修正后的Bellman Equation, 未来回报换为未来期望的回报

7.连续状态空间应用

之前的火星探测器案例中, 只有六个离散的状态, 即他只能处于这六个状态之一, 然而大多数机器都可以处于许多的状态, 每个状态也不一定只有一个数据来描述

Discrete vs Continuous State

Discrete State:



Continuous State:



$$s = \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}$$

屏幕剪辑的捕获时间: 2023/6/14 17:54

Autonomous Helicopter



$$s = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \omega \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\omega} \end{bmatrix}$$

屏幕剪辑的捕获时间: 2023/6/14 17:55

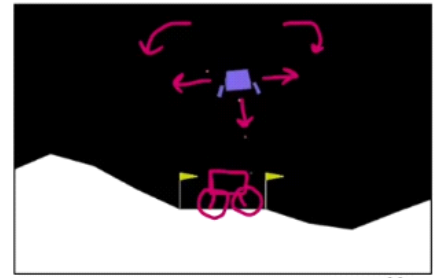
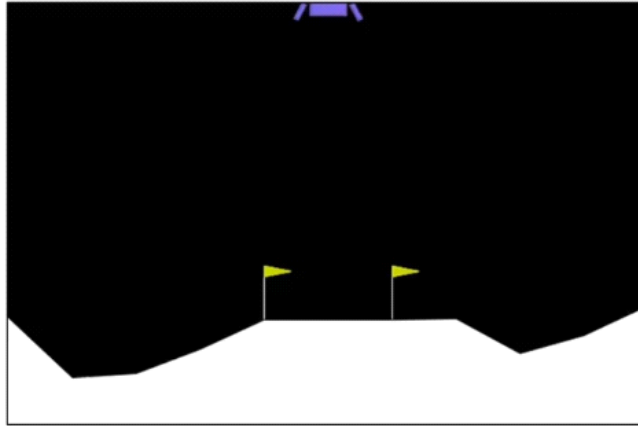
其中字母上方的点表示求导, 也即这些量变化的快慢

8.Deep Reinforcement Learning

示例: 登月器模型

其中 x, y 代表水平、竖直位置, θ 代表倾角, L, r 代表机器的左/右脚是否着地, 即 L, r 是二元的值0或1, 其余为连续值

Lunar Lander



$$s = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ l \\ r \end{bmatrix}$$

l 和 r 将是二进制值，并且只能取零值或

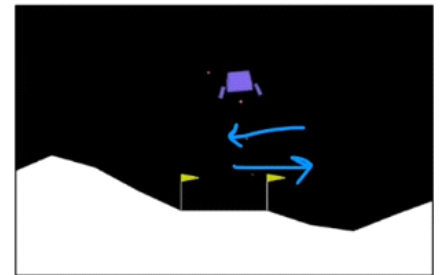
DeepLearning.AI will be binary valued and can take on only values zero or 1 Andrew Ng

屏幕剪辑的捕获时间: 2023/6/14 20:53

Reward Function设置如下:

Reward Function

- Getting to landing pad: 100 – 140
- Additional reward for moving toward/away from pad.
- Crash: -100
- Soft landing: +100
- Leg grounded: +10
- Fire main engine: -0.3
- Fire side thruster: -0.03

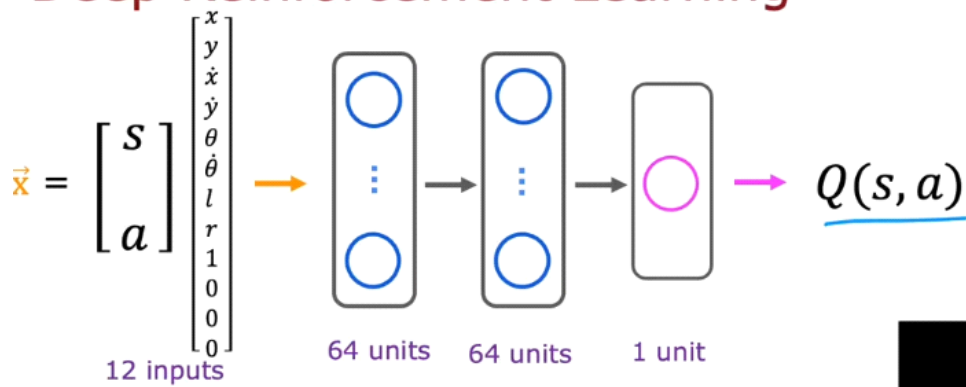


屏幕剪辑的捕获时间: 2023/6/14 20:53

所以任务就是：对于任何一个给定的状态 s ，需要找到一个policy，使得按照这个决策选择的行动让回报最大

利用神经网络来学习状态值函数 $Q(s,a)$ ，方法是输入一组(state,action)数据，让神经网络输出 $Q(s,a)$

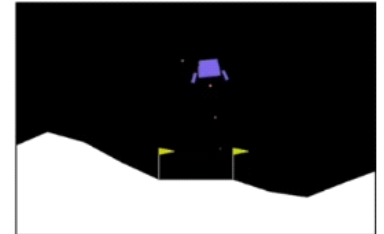
Deep Reinforcement Learning



In a state s , use neural network to compute

$Q(s, \text{nothing})$, $Q(s, \text{left})$, $Q(s, \text{main})$, $Q(s, \text{right})$

Pick the action a that maximizes $Q(s, a)$



屏幕剪辑的捕获时间: 2023/6/20 9:59

如何构建这样的神经网络?

①. 让登月器在随机的状态采取随机的行动, 得到许多数组:

$(s, a, R(s), s')$

②. 储存最近10000次的数组 (*Replay Buffer*)

③. 利用贝尔曼方程训练网络, 创建 (x, y) 训练集, 其中:

$x = (s, a)$

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

注意要训练Q函数, 所以给Q函数一个初始的估计, 每次训练后都会得到一个新的Q函数, 再将其用于下一次的训练, 慢慢优化Q函数

Bellman Equation

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

$$f_{w, \theta}(x) \approx y$$

$$(s, a, R(s), s')$$

$$(s^{(1)}, a^{(1)}, R(s^{(1)}), s'^{(1)}) \leftarrow$$

$$(s^{(2)}, a^{(2)}, R(s^{(2)}), s'^{(2)}) \leftarrow$$

$$(s^{(3)}, a^{(3)}, R(s^{(3)}), s'^{(3)}) \leftarrow$$

$$y^{(1)} = R(s^{(1)}) + \gamma \max_{a'} Q(s'^{(1)}, a')$$

$$y^{(2)} = R(s^{(2)}) + \gamma \max_{a'} Q(s'^{(2)}, a')$$

x	y
$x^{(1)} = (s^{(1)}, a^{(1)})$	$y^{(1)}$
$x^{(2)} = (s^{(2)}, a^{(2)})$	$y^{(2)}$
$x^{(10,000)}$	$y^{(10,000)}$

我们将训练一个新网络, 例如

屏幕剪辑的捕获时间: 2023/6/20 9:54

Learning Algorithm

Initialize neural network randomly as guess of $Q(s, a)$.

Repeat {

Take actions in the lunar lander. Get $(s, a, R(s), s')$.

Store 10,000 most recent $(s, a, R(s), s')$ tuples.

Replay Buffer

Train neural network:

Create training set of 10,000 examples using

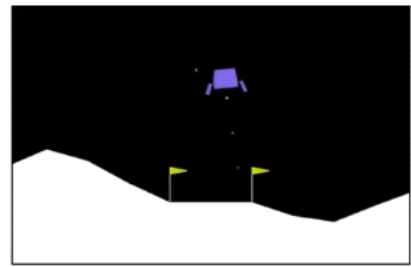
$$x = (s, a) \text{ and } y = R(s) + \gamma \max_{a'} Q(s', a')$$

Train Q_{new} such that $Q_{new}(s, a) \approx y$.

Set $Q = Q_{new}$.

$$f_{w, B}(x) \approx y$$

函数的估计。



x, y

x'', y''

x^{10000}, y^{10000}

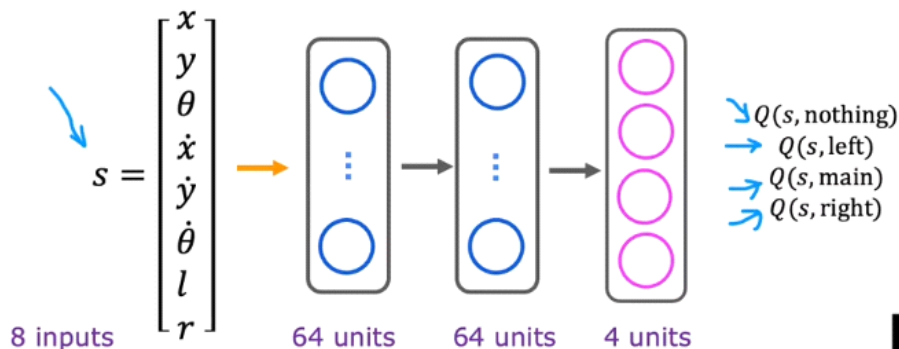
屏幕剪辑的捕获时间: 2023/6/20 9:55

9. 算法改进

1. 优化神经网络架构

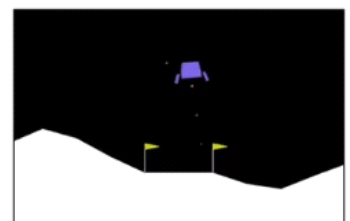
之前的网络对于每种状态都要进行多次预测，很低效，如果改变输出层，使其同时计算一种状态下，采取不同可能的action后Q函数的值，选出最大的Q值对应的action即可

Deep Reinforcement Learning



In a state s , input s to neural network.

Pick the action a that maximizes $Q(s, a)$. $R(s) + \gamma \max_{a'} Q(s', a')$



计算它的效率更高

much more efficient to compute this

Andrew Ng

屏幕剪辑的捕获时间: 2023/6/20 10:02

2. ϵ -greedy策略

在让登月器采取行动以获得Training Set时（已经假设了一个Q函数），有两种选择：

- ①. 始终选择令 $Q(s, a)$ 最大的行动
- ②. 有 ϵ 的可能性随机选择行动， $(1-\epsilon)$ 的可能性选择使Q最大的行动

在一些情况中，可能对于一个状态 s ，按照①所说，采取某一个特定行动 a' 对应的Q函数值可能一直很小，但是由于Q函数仍然在学习过程，一直以最大化当前Q函数为目标可能会让算法永远不会选择行动 a' ，这时方法②就会给算法 ϵ 的概率让算法进行**explore**，即探索更多可能性

而使Q最大化的行动选择则称为**exploit**，即试图最大化回报，所以许多情况下需要对**exploration**和**exploitation**进行权衡

可以采取的策略是：在学习初期使 ϵ 很大，探索更多可能性，随着学习的深入，Q函数越发精确，可以适当降低 ϵ ，狠狠压榨算法XD

PS：强化学习对于参数的选择十分敏感（相比于监督学习），这与强化学习研究程度较浅有关，参数选不好往往会导致学习速度十分慢，调参很重要

3. Mini-Batch and Soft Update

Mini-Batch:

在数据集很大时，同时对所有数据进行训练可能会非常慢，这时可以把数据集分成许多小的子集，每计算一次就更新一次模型，然后用于下一次的计算，这同样可以用于监督学习

在强化学习中，以上述登月器为例，在重放缓冲区中放入10000组数据后，也可以将其分为10批次，每次计算后更新Q函数，进行下一次的计算

How to choose actions while still learning?

x	y
2104	400
1416	232
1534	315
852	178
...	...
3210	870

100,000,000

$$J(w, b) = \frac{1}{2m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})^2$$

$$m = 100,000,000$$

$$m' = 1,000$$

$$\text{repeat } \left\{ \begin{array}{l} w = w - \alpha \frac{\partial}{\partial w} \left[\frac{1}{2m'} \sum_{i=1}^{m'} (f_{w,b}(x^{(i)}) - y^{(i)})^2 \right] \\ b = b - \alpha \frac{\partial}{\partial b} \left[\frac{1}{2m'} \sum_{i=1}^{m'} (f_{w,b}(x^{(i)}) - y^{(i)})^2 \right] \end{array} \right\}$$

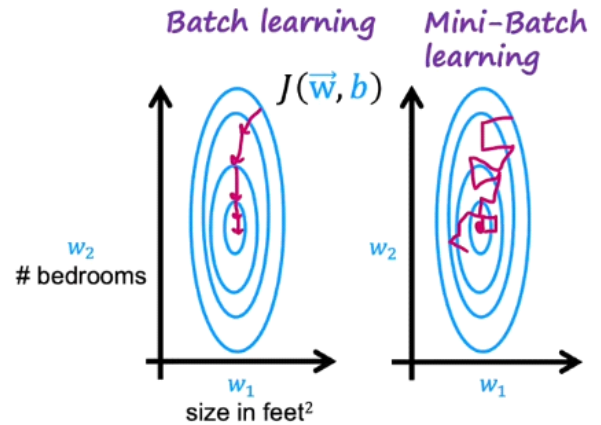
屏幕剪辑的捕获时间: 2023/6/21 9:32

Mini-Batch方法的特点是：

①. 目标函数趋向全局最小值的过程十分“嘈杂”“曲折”，但是每次的计算都非常快

Mini-batch

x	y
2104	400
1416	232
1534	315
852	178
...	...
3210	870



屏幕剪辑的捕获时间: 2023/6/21 9:32

②. Mini-Batch在每次计算后都会更新目标函数（如强化学习中的Q函数），然后用于下一次计算，但是这种更新是一种很“突然”的变化，有可能这样得到的新的函数性能还不如原来的，所以需要一些“柔和”的变换方式



Soft Update

软更新是一种可以让目标函数的更新更加丝滑的方法，通过对目标函数的参数进行新老加权的方式来更新函数

Soft Update

$$\text{Set } Q = Q_{\text{new}} \quad \leftarrow \quad Q(s, a)$$

\nwarrow \swarrow
 w, B $w_{\text{new}}, B_{\text{new}}$

$$W = 0.01 W_{\text{new}} + 0.99 W$$

$$B = 0.01 B_{\text{new}} + 0.99 B$$

$$w = 1 W_{\text{new}} + 0 W$$

屏幕剪辑的捕获时间: 2023/6/21 9:38