

State-values to action-values

$$v_{\pi}(s) \approx \hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^T \mathbf{x}(s)$$

↑
weight vector
↓
feature vector (one-hot)

$$q_{\pi}(s, a) \approx \hat{q}(s, a, \mathbf{w}) \doteq \mathbf{w}^T \mathbf{x}(s, a)$$

represent action as well.

Representing actions

$$\mathbf{x}(s) = \begin{bmatrix} x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \end{bmatrix}$$



$$\mathcal{A}(s) = \{a_0, a_1, a_2\}$$

$$\left. \begin{bmatrix} x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \end{bmatrix} \right\} a_0$$
$$\left. \begin{bmatrix} x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \end{bmatrix} \right\} a_1$$
$$\left. \begin{bmatrix} x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \end{bmatrix} \right\} a_2$$

$$\mathbf{x}(s, a_0) =$$

$$\begin{bmatrix} x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \end{bmatrix}$$

⇒ only features of respective action will be active, the other action will be set to 0.

Computing action-values

$$\mathbf{x}(s_0) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\mathcal{A}(s) = \{a_0, a_1, a_2\}$$

$$\mathbf{w} = \begin{bmatrix} 0.7 \\ 0.1 \\ 0.4 \\ 0.3 \\ 2.2 \\ 1.0 \\ 0.6 \\ 1.8 \\ 1.3 \\ 1.1 \\ 0.9 \\ 1.7 \end{bmatrix}$$

$$\mathbf{x}(s_0, a_2) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\hat{q}(s_0, a_0, \mathbf{w}) = 0.7 + 0.3 = 1$$

$$\hat{q}(s_0, a_1, \mathbf{w}) = 2.2 + 1.8 = 4$$

$$\hat{q}(s_0, a_2, \mathbf{w}) = 1.3 + 1.7 = 3$$

因此，只有指定动作的特征将被激活，而其他动作的特征将被设置为0。让我们看一个例子，说明如何从一个给定的状态中计算出动作值。比方说，有四个特征和三个动作。权重系数看起来是这样的。通过叠加特征，我们得到以下状态s中动作0的特征向量，我们0出其他动作的特征。所以我们提取与每个动作对应的权重向量的段。然后，动作值是权重向量的每一段与特征向量之间的点积。你可能会认为，堆叠特征来创建动作值是线性函数逼近所特有的，但事实并非如此。例如，用神经网络表示动作值的常见方法是生成多个输出，每个动作值一个。然而，这等同于我们刚刚描述的堆叠程序。神经网络输入的是状态，最后一个隐藏层产生的是状态特征。每个动作值都是通过使用这些状态特征的独立权重集来计算的。 *state feature*

一个动作值的权重不会与另一个动作值的权重相互影响，就像在堆叠中一样。

我们可能想对行动进行泛化，这与对状态进行泛化的原因相同。现在，这在神经网络中如何运作？

我们会把状态和动作都输入到网络中。只会有一个输出。该状态和行动的近似行动值。我们可以通过将状态和动作作为输入，对瓷砖涂层做类似的事情。好了，我们现在知道如何用函数近似法处理动作值了。让我们来谈谈用Sarsa来控制函数逼近的问题。这个算法与标签分流相当类似，所以我们只需回顾一下其中的区别。

我们使用参数化的动作值函数进行动作值估计。更新也改变为使用梯度来更新权重，类似于类似梯度TD。就这样了。这就是带有函数近似的Sarsa控制算法。在这段视频中，我们谈到了动作相关的特征，并介绍了带有函数近似的Episodic Sarsa。在接下来的讲座中，我们将介绍更多的控制算法control algorithms。//

Episodic Sarsa with function approximation

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$

Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$

Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:

$S, A \leftarrow$ initial state and action of episode (e.g., ε -greedy)

Loop for each step of episode:

Take action A , observe R, S'

If S' is terminal:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R - \hat{q}(S, A, \mathbf{w})] \nabla \hat{q}(S, A, \mathbf{w}) \quad \leftarrow$$

Go to next episode

Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε -greedy)

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R + \gamma \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})] \nabla \hat{q}(S, A, \mathbf{w}) \quad \leftarrow$$

$$S \leftarrow S'$$

$$A \leftarrow A'$$

在这段视频中，我们可以看到连续状态域中的偶发性sarsa在行动。我们将直观地看到学习值，并获得一些关于sarsa学习的解决方案的直觉。在本视频结束时，你将获得分析近似TD控制方法性能的经验。

让我们来看看一个经典控制领域 "山地车" 上的偶发性sarsa的例子。山地车是一个偶发任务，目标是驾驶一辆动力不足的汽车爬上山的一侧。重力比汽车的发动机更强，所以代理人不能直接开车上山。逃跑的唯一方法是首先向后开上左边的斜坡。然后开车下山，使汽车有足够的动力开上右边的斜坡，然后出去。本集开始时，汽车在谷底附近的一个随机位置。当汽车到达山顶的旗帜时，它就结束了。为了鼓励代理人尽快完成这个情节，每个时间段的奖励都是减一，而且不使用折扣。代理人可以观察汽车的当前位置和速度。这是一个二维连续值的状态。代理人可以选择三种行动：向前加速，向后加速，或沿海。在这个例子中，我们联合talco位置和速度来产生特征。我们使用八个~~瓦片~~^{tile}，这意味着我们对二维输入空间使用八乘八的网格。我们使用八个~~瓦片~~^{tile}_(重叠网格)，这意味着我们有八个重叠的网格。我们通过使用堆叠的特征表示法来独立处理动作。我们将权重初始化为零。这种初始化实际上是乐观的。这是因为每一步的奖励是减一，所以任何政策下的值都会小于零。在这个问题上，这些乐观的初始值导致了广泛和系统的探索。正因为如此，我们可以贪婪地行动而不需要任何额外的随机探索。

让我们看看在我们运行代理很长一段时间后，价值函数是什么样子的。理想情况下，我们希望绘制每个状态的价值函数。然而，有一个无法计数的无限多的状态。所以我们必须对状态集进行抽样。我们将使用每个样本状态下的最大值。这个数字告诉我们，在贪婪政策下，代理人认为它需要多少步才能逃脱。每一步的奖励是减一，所以我们否定这个数字，以产生代理人对从每个状态出发的步骤数的估计。这是代理人对9000个事件后的预期步骤数的估计。对于这个问题来说，这是一个非常长的时间，所以我们非常确定这些值的估计是最好的。但由于函数的近似，它们并不完美。

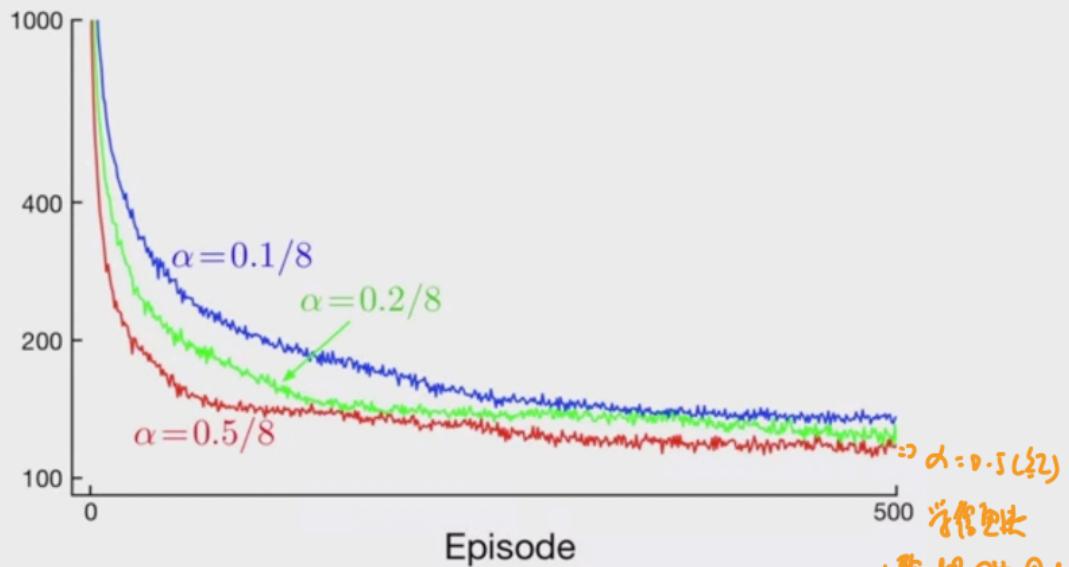
绿线代表目标位置，不取决于速度。在目标附近，如果速度足够大，代理人可以直接开出去。然而，如果代理人在目标附近，但速度太低，它将需要很多步才能逃脱。这是因为代理人必须先回到山下，再从左边山上山，以获得足够的动力来逃跑。

峰值对应的是开始状态，它需要大约120步才能到达旗子。这条绿色轨迹显示了学习策略在状态空间中的路径。值函数看起来很有趣，但让我们看看一些学习曲线，以便更好地了解学习的速度。让我们用三个不同的Alpha值来尝试sarsa，所以算法的三个变种。在这里，我们对达到目标的步骤感兴趣，在这种情况下，它对应于每集的回报。我们希望每集的步骤数随着学习而减少。X轴上的低点对应于更好的政策，即能够以平均最少的步骤达到目标的政策。像往常一样，我们对许多独立运行的性能进行平均。下面是结果。到500集时，每集的步骤数已大致稳定。所有的学习曲线都呈现出熟悉的指数曲线。较小的步长参数值0.1导致了较慢的学习，而0.5的Alpha值允许代理更快地学习，并在500次事件中找到更好的政策。注意我们把每个Alpha值除以8。如果我们不使用步长的矢量，我们通常用特征向量的常数来缩放步长参数。在这里，我们在瓦片编码器中使用了八个瓦片。这意味着特征向量中1的数量始终是8。作为一个简单的练习，向你自己证明，8对应于特征向量的L₁规范。

在这段视频中，我们在山地车中用线性函数近似法评估了episodic sarsa。下次见。//

Learning curves

Mountain Car
Steps per episode
log scale
averaged over 100 runs



tile, = one-hot vector \times (1, 1, 1)
中16的数是1, 其他都是0.

平衡探索 exploration 和 利用 exploitation 的需要是顺序决策问题的决定性特征之一。我们已经谈到了在bandits和表格强化学习中促进探索的几种简单方法。那函数逼近呢？那里的探索有什么特别之处吗？今天我们就来了解一下。在本视频结束时，你将能够描述乐观的初始值和Epsilon greedy是如何用于函数逼近的。

让我们简单复习一下我们如何在表格设置中使用乐观的初始值。我们把我们的值初始化为大于真实值。这就像代理想象它通过采取该行动可以得到比现实中更多的奖励。通常，以这种方式初始化价值函数会使代理人系统地探索状态行动空间。随着代理人的价值变得越来越准确，他们受这种初始化的影响也越来越小。这在表格设置中很容易实现，因为每个状态行动对的更新都与所有其他状态行动对无关。在函数近似中，乐观的初始值对应于初始化权重，使其结果值是乐观的。在某些情况下，这是直接的，例如，当特征是二进制时，我们简单地将每个权重初始化为可能的最大回报。然后，只要每个状态至少有一个特征处于激活状态，其值就会是乐观的，而且很可能是过度乐观的。然而，在许多情况下，很难做到乐观地初始化。例如，在一个神经网络中，最终值和特征之间的关系可能相当复杂。

想象一下一个由tanh激活函数组成的网络。即使初始权重为正，该网络也可以输出负值。但这并不是故事的全部。取决于我们的特征如何泛化，乐观的初始值可能不会导致我们在表格案例中看到的那种系统性探索。考虑一个极端的例子，我们只有一个永远是一的特征。我们可以乐观地初始化，但每次更新都会改变所有状态的值。这意味着在某些状态被访问之前，数值就已经下降了，以至于不再乐观了。为了促进系统的探索，对价值函数的改变需要更加本地化。例如，使用瓦片编码的函数近似可以产生这种局部更新。神经网络和也提供局部更新，但神经网络也可能积极地泛化。在实践中，如果没有特别的考虑，神经网络会比较快地失去他的乐观性 *optimism*。Epsilon greedy普遍适用，即使在有非线性函数近似的情况下也容易使用。Epsilon贪婪唯一需要的是 动作值估计 *action value estimates*，与它们的初始化或近似方式无关。然而，Epsilon贪婪并不是一种定向探索方法。它依靠随机性来发现当前政策所遵循的状态附近的更好的行动。因此，它不像依赖乐观主义的探索方法那样系统。在函数近似设置中改进探索仍然是一个开放的研究问题。所以在本课程中，我们将坚持采用这种简单的策略。

在这个视频中，我们谈到了在结合乐观的初始值和函数近似时有很多微妙之处，以及Epsilon贪婪如何与任何函数近似相结合。下回见。//

在持续任务中，我们可能对极长的水平线表现感兴趣。到目前为止，我们已经用贴现和持续问题来平衡短期表现和长期收益。然而，这并不是制定问题的唯一方法。今天，我们将学习一种制定持续问题的新方法，称为平均奖励制定。*Average reward formulation*

在本视频结束时，你将能够描述平均报酬的设定，解释什么时候平均报酬最优政策与贴现下获得的政策不同，并理解差值函数。

今天是关于持续任务的。想象一下一个简单的任务，其中的状态被安排在两个相交的环中。让我们把这称为近视的MDP。

near sighted

在大多数状态下，只有一个行动，所以没有决定要做。只有一个状态是可以做决定的。在这个状态下，代理人可以决定穿越哪个环。这意味着有两个决定性的政策，即穿越左环或穿越右环。除了在每个环的一个过渡中，其他地方的奖励都是零。在左环中，状态S之后的奖励是+1；在右环中，状态S之前的奖励是+2。但是，价值函数会告诉我们该怎么做呢？

如果我们使用贴现，在这两个不同的政策下，状态S的价值是什么？选择左边行动的政策的价值是1比1减去gamma的5，我们怎么算出来呢？如果你写出无限折现的回报，你会发现这是一个相当直接的几何数列，有一个封闭式的解决方案。看看你是否能得到和我们一样的答案。

右边

选择正确行动的政策，其价值是4的2倍伽玛2，而不是1减去伽玛的5。让我们想想在这两部分政策下，在特定的伽玛值下，状态S的价值。如果gamma为0.5，VL大约为1，VR大约为0.1。这意味着在这种比较近视的折扣下，采取左边行动的政策是比较好的。让我们尝试一个更大的伽马值，即0.9。

现在VL约为2.4, VR约为3.2。所以现在我们更喜欢另一个政策。事实上,我们可以算出伽马的最小值,使代理人更喜欢向右走的政策。伽马至少需要是0.841。所以这里的问题是,折扣幅度取决于问题。对于这个例子来说,0.85就足够大了。但如果环状物各有100个状态,这个折扣系数就需要超过0.99。

一般来说,确保代理人的行动在一段时间内获得最大回报的唯一方法是不断增加折扣系数,使之趋于1。根据问题的不同,我们可能需要gamma相当大。记住,我们不能在一个持续的环境中把它设置为1,因为那样的话回报可能是无限的。现在,更大的gamma有什么问题呢?较大的gamma值也会导致更大和更多的变量和,这可能是难以学习的。那么,有没有一种替代方法呢?让我们来讨论一个新的目标,叫做平均奖励。想象一下,代理人已经与世界互动了H步。这是它在这H个步骤中获得的平均奖励。换句话说,它的奖励率。如果代理人的目标是使这个平均奖励最大化,那么它对附近和远处的奖励同样关心。我们用 R_{pi} 来表示一个政策的平均奖励。更一般地说,我们可以用状态访问量(mu)来写平均奖励。这个内项是政策 pi 下一个状态的预期奖励。外项是对政策在该状态下出现频率的期望值。合在一起,我们就得到了跨州的预期奖励。换句话说,就是一个政策的平均奖励。在近视的例子中,两个确定性的可能政策无限期地访问左循环或右循环。在这两种情况下,每个循环中的五个状态被访问的次数相同。在左边的循环中,除了一个状态得到+1外,所有状态的即时预期报酬都是+0。这导致每5步的平均奖励为1或0.2。

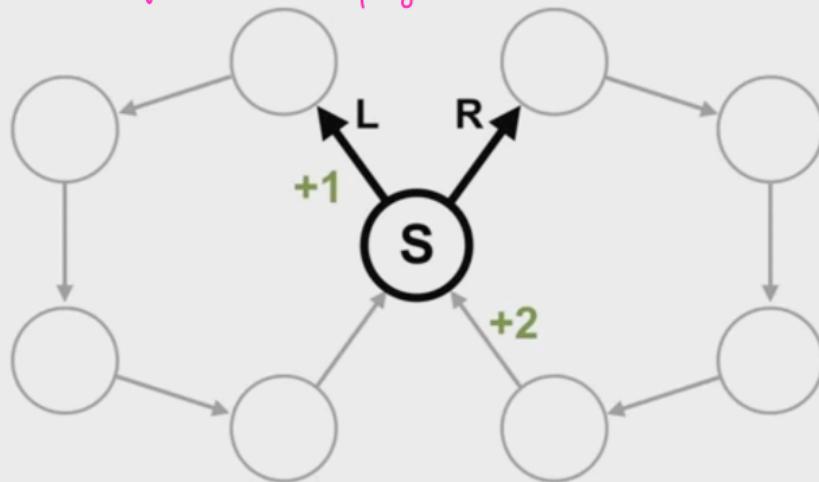
右边循环中的大多数状态也有+0的预期奖励。但是这一次,最后一个日期得到了+2。这使得每5步的平均奖励为2,即0.4。

我们可以看到,平均奖励把优先权放在总的获得更多奖励的政策上,而不必考虑越来越大的折扣。

The Average Reward objective

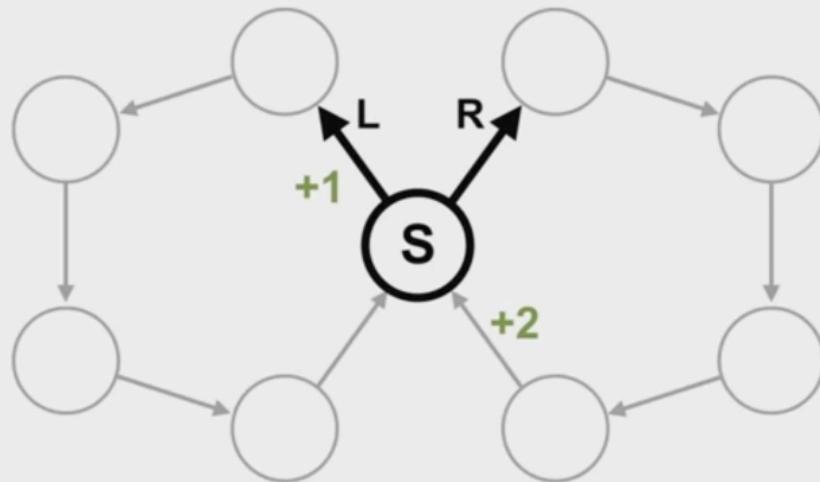
$$r(\pi) \doteq \lim_{h \rightarrow \infty} \frac{1}{h} \sum_{t=1}^h \mathbb{E}[R_t | S_0, A_{0:t-1} \sim \pi]$$

the average reward of the policy.



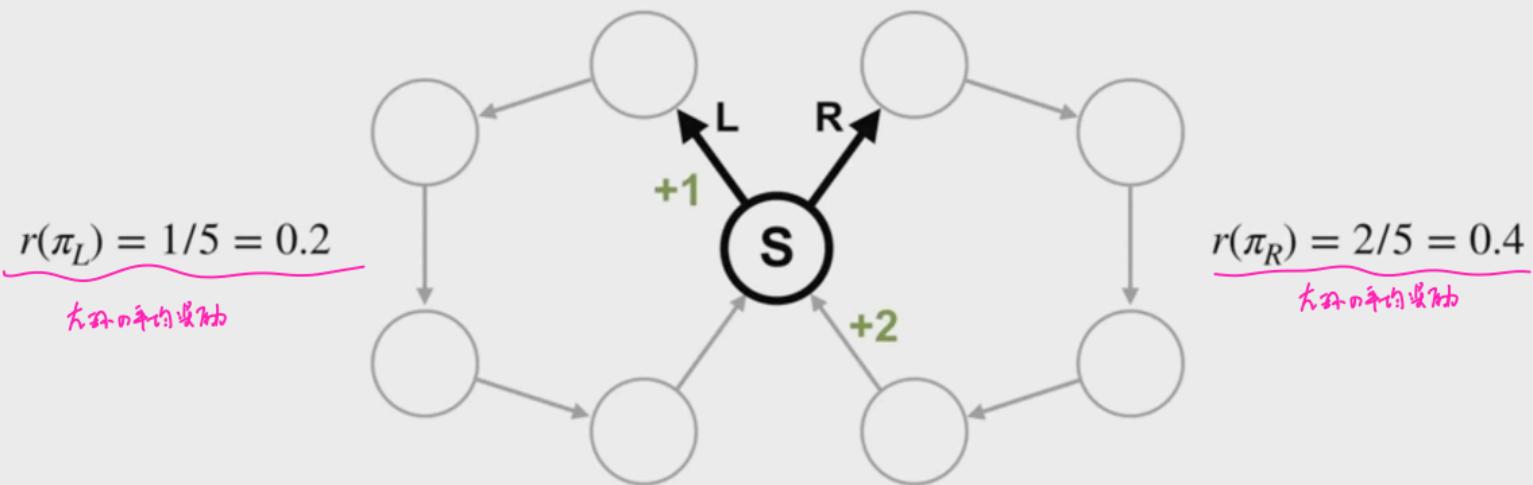
The Average Reward objective

$$r(\pi) = \sum_s \mu_\pi(s) \sum_a \pi(a | s) \sum_{s',r} p(s', r | s, a) r$$



The Average Reward objective

$$r(\pi) = \sum_s \mu_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)r$$



平均奖赏的定义对于说一个政策是否比另一个政策好是很直观的，但我们如何决定一个国家的哪些行动是更好的呢？我们需要的是这个新设定的行动值。第一步是要弄清楚回报是什么。在平均奖励设置中，回报是以奖励和平均奖励 R_{pi} 之间的差异来定义的。这就是所谓的差额回报。让我们看看在我们的近视MDP中差额回报是什么。差额回报表示代理人在行动中从当前状态获得的奖励与政策的平均奖励相比多多少。让我们看一下从状态 S 开始，首先选择行动 L ，然后跟随 pi_L 之后的差额回报。这个政策的平均回报是 0.2。差额回报是未来回报的总和，每个回报都要减去平均回报。我们可以通过对某个有限水平线 H 进行求和来计算它。我们用这个极限符号稍微简化了一些事情。虽然符号提供者在许多情况下是有效的，但当环境是周期性的，我们需要使用不同的技术。在这种情况下，我们使用一个更普遍的极限来计算回报，称为 Cesaro 和，但这个技术细节并不关键。这里主要是指直觉。我们发现，差额回报率为 0.4。现在，让我们来看看另一个动作。这一次，我们可以把差额回报分成两部分。首先是通过右环的单个轨迹的总和。我们可以明确地写出这个总和，它等于 1。然后是无限期地采取左边的行动所对应的总和。这个和与我们刚才计算的差额回报相同，为 0.4。将这两部分相加，我们发现差额回报是 1.4。

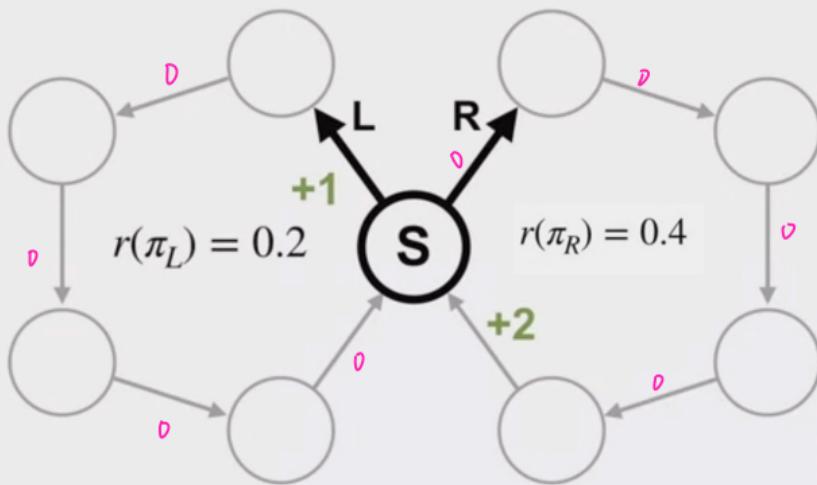
因此，如果代理人的政策是总是采取左边的行动，它可以观察其差额回报，并意识到它应该转而采取右边的行动。

现在，让我们来看看如果代理人的政策是总是采取正确的行动，那么差额回报。这个政策导致的平均回报是 0.4，在状态 S 下采取正确行动的政策的差额回报是 -0.8。现在，在 S 状态下采取左边的行动，然后无限期地采取右边的行动，其差额回报是多少呢？像以前一样，我们把总和分成两部分，采取一次左循环的结果是在前五个时间步骤中的总和是 -1。再加上从 S 状态跟随 pi_R 的差额回报，我们发现是 -0.8，结果我们的答案是 -1.8。再一次，我们看到正确的行动是首选。

Returns for Average Reward

$$G_t = [R_{t+1} - r(\pi)] + [R_{t+2} - r(\pi)] + [R_{t+3} - r(\pi)] + \dots$$

$$\begin{aligned}
 G_t &= 1 - 0.2 + \\
 &\quad 0 - 0.2 + \\
 &\quad \vdots \quad H \rightarrow \infty \\
 &= 0.4
 \end{aligned}$$



$$G_t = 0 - 0.2 + 0 - 0.2 + 0 - 0.2 + 0 - 0.2 + 2 - 0.2 + 0.4 = 1.4$$

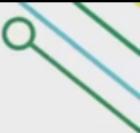
你可能已经注意到，尽管 $P_i R$ 的平均回报较高，但 $P_i R$ 的差额回报比 $P_i L$ 的差额回报低。这是因为差额回报代表了在某一状态下采取某种行动比在某一政策下的平均水平要好多少。只有在随后的时间步骤中遵循相同的政策时，才能用差额回报来比较行动。为了比较政策，应该使用它们的平均回报来代替。

有趣的是，只有当被减去的常数等于真正的平均报酬时，差额回报才是一个收敛的总和。如果减去一个较低或较高的数字，总和将发散到正或负的无限大。

现在我们对平均报酬的回报有了一个有效的定义，我们以通常的方式定义价值函数，即预期回报。同样地，我们也可以将差值函数定义为来自给定状态或状态行动对的政策下的预期差值回报。这个数量反映了代理人从某一特定状态开始得到的回报比它在所有状态下遵循固定政策平均得到的回报多多少。

像在贴现设置中一样，差分价值函数可以写成贝尔曼方程。方便的是，它们看起来和我们之前看到的那些一样。它们的不同之处在于，它们从即时奖励中减去了 R_{pi} ，而且没有折扣。

许多贴现情况下的算法可以被改写成适用于平均奖励的情况。例如，差分 Sarsa 与你以前见过的 Sarsa 算法非常相似。让我们来看看两者的区别。一个关键的区别是，差分 Sarsa 必须跟踪其策略下的平均奖励的估计值，并在更新中从样本奖励中减去它。这个实现是用我们在整个课程中看到的增量平均技术来做的。考虑到这个估计值，它就在更新中从抽样奖励中减去 R 吧。在实践中，我们可以通过对这个算法的轻微修改来获得更好的性能。我们不使用奖励的指数平均值来计算 R_{bar} ，而是使用这个更新，它的方差较低。



Differential Sarsa

Differential semi-gradient Sarsa for estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$

Algorithm parameters: step sizes $\alpha, \beta > 0$

Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Initialize average reward estimate $\bar{R} \in \mathbb{R}$ arbitrarily (e.g., $\bar{R} = 0$)

Initialize state S , and action A

Loop for each step:

 Take action A , observe R, S'

 Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε -greedy)

$\delta \leftarrow R - \bar{R} + \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})$ *→ 满足动作的估价值.*

$\bar{R} \leftarrow \bar{R} + \beta \delta$ *→ 平均奖励的估计值.*

$\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \nabla \hat{q}(S, A, \mathbf{w})$

$S \leftarrow S'$

$A \leftarrow A'$

本周，我们谈到了使用函数近似法时如何做控制，让我们复习一下主要的观点。

首先，我们向你展示了如何用函数近似法估计动作值。如果动作空间是离散的，可能最简单的就是堆积状态特征。如果动作空间是连续 continuous 的，或者你想对动作进行泛化，那么动作可以像其他状态变量一样作为一个输入被传递。让我们通过查看算法图来了解一些关于模块下一部分的背景。函数逼近使我们处于地图的左侧，第一讲的重点是左下角的控制算法 control algorithms: SARSA、预期 SARSA 和 Q-Learning。这些都是我们在课程 2 中所涉及的表格控制算法的扩展，这些算法与表格中的对应算法的唯一区别是更新方程。更新都是以同样的方式适应于函数近似，使用梯度来更新权重。我们还看到了如何用偶发性 SARSA 来解决山地车问题。在这种情况下，较大的步长是 0.5，能够更快地学习。

接下来，我们谈到了探索，乐观的初始化 optimistic initialization 可以用于一些结构化的特征表示，如瓦片编码。但一般来说，如何用神经网络这样的非线性函数近似器进行乐观的初始化值并不清楚，它的表现可能不符合预期，例如乐观主义可能消退得太快。Epsilon-greedy 可以不受函数逼近器的影响而使用。

最后，我们谈到了一种思考持续控制问题 continuing control problem 的新方法。我们可以考虑最大化一个政策在一段时间内获得的平均回报，而不是最大化当前状态下的折现回报。我们定义了差额回报和差额价值，这些使代理人能够在平均回报的环境下评估行动的相对价值。最后，我们引入了差分半梯度 SARSA，它近似于差分值 differential values 来学习政策。差分 SARSA 也在算法图的左半部分，但与我们在本周早些时候介绍的算法不同，它使用的是平均奖励框架 average reward framework。

本周的内容就到此为止，我们将表格控制算法扩展到函数近似，讨论了探索如何变化，并引入了思考控制问题的新方法。下周，我们将讨论如何在不学习价值函数的情况下进行强化学习。到时见。

Representing actions

$$\mathbf{x}(s) = \begin{bmatrix} x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \end{bmatrix} \longrightarrow \mathbf{x}(s, a) =$$

$$\mathcal{A}(s) = \{a_0, a_1, a_2\}$$

$$\left. \begin{array}{c} x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \end{array} \right\} a_0$$
$$\left. \begin{array}{c} x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \end{array} \right\} a_1$$
$$\left. \begin{array}{c} x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \end{array} \right\} a_2$$

\Rightarrow episodic task

1 action/episode feature vectors

Representing actions

$$\mathbf{x}(s) = \begin{bmatrix} x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \end{bmatrix} \longrightarrow \mathbf{x}(s, a) =$$

$$\mathcal{A}(s) = \{a_0, a_1, a_2\}$$

$$\begin{bmatrix} x_0(s, a) \\ x_1(s, a) \\ x_2(s, a) \\ x_3(s, a) \\ x_4(s, a) \\ x_5(s, a) \\ x_6(s, a) \\ x_7(s, a) \\ x_8(s, a) \\ x_9(s, a) \\ x_{10}(s, a) \\ x_{11}(s, a) \end{bmatrix}$$

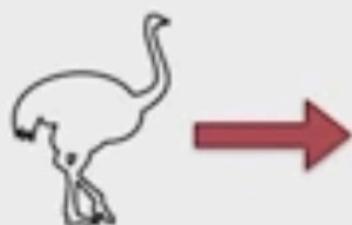
↳ continuing talk

Exploration with Approximation

ε - study:

$$1 - \epsilon$$

$$\epsilon$$



$$A_t = \operatorname{argmax}_a \hat{q}(S_t, a, \mathbf{w})$$

A_t = Random action

Average Reward

$$r(\pi) = \sum_s \mu_\pi(s) \sum_a \pi(a | s) \sum_{s',r} p(s', r | s, a) r$$

$$G_t = \boxed{R_{t+1} - r(\pi)} + \boxed{R_{t+2} - r(\pi)} + \boxed{R_{t+3} - r(\pi)} + \dots \Rightarrow \text{differential return}$$

$$q_\pi(s, a) = \sum_{s',r} p(s', r | s, a) \left(r - r(\pi) + \sum_{a'} \pi(a' | s') q_\pi(s', a') \right) \Rightarrow \text{differential value}$$