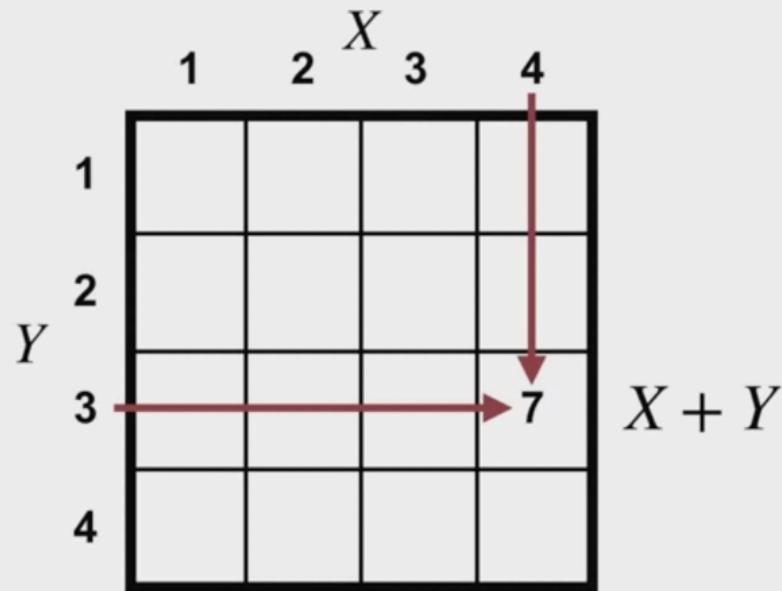


到现在为止，我们已经了解了表格式方法 tabular methods。这些方法存储了包含单独学习值的表格，用于每个可能的状态。在现实世界的问题中，这些表格将变得难以处理的大。想象一下，一个通过摄像头看世界的机器人。我们显然不能为每一个可能的图像都存储一个表项。幸运的是，这并不是唯一的可能性。今天，我们将讨论更多关于近似值函数的一般方法。在本视频结束时，你将能够理解我们如何使用参数化函数来近似值，解释线性函数的近似，认识到表格的情况是线性值函数近似的一个特例，并理解有许多方法来参数化近似值函数。

在以前的课程中，我们花了很多时间来思考如何估计价值函数。到目前为止，我们的值近似总是有相同的形式。对于每个状态，我们在一个表格中存储单独的值。我们查看数值，并随着学习的进展在表中修改它们。但是，这并不是对价值函数进行近似的唯一方法。原则上，我们可以使用任何接受一个状态，并产生一个实数的函数。例如，在这样的网格中，我们的价值函数近似可以采取X和Y的位置，并把它们加在一起，产生一个价值估计。这只是为了让你的直觉更清晰。我们不想把它作为一个价值函数，因为我们不能修改这个近似值。所以我们没有办法进行学习。

A value function can be represented in many different ways

State	Value
$S_1$	-4
$S_2$	6
$S_3$	12
$S_4$	5
$S_5$	53
...	
$S_{16}$	-9



## Parameterizing the Value Function

$$\hat{v}(s, \mathbf{W}) \approx v_{\pi}(s)$$

weights



这就是参数化函数的概念的来源。我们加入了一组实值权重 real valued weights，我们可以调整这些权重来改变函数。例如，在这个网格中，我们可以使用一个  $w_1$  乘以  $X$ ，再加上  $w_2$  乘以  $Y$  的函数，而不是使用一个固定的  $X$  和  $Y$  的总和。它们允许我们改变该函数产生的输出。为了表示这个函数接近于真实的价值函数，我们使用符号  $\hat{v}$ 。  $W$  是一个向量，包含所有对近似进行参数化的权重。我们不需要存储一整张表的数值。我们只需要存储两个权重来表示我们的价值函数近似。你可以想象这种紧凑的表示法对大的状态空间是多么有用。表格表示法修改了一个状态的值估计，使其他状态没有变化。而对于参数化的函数来说，情况就不再是这样了。注意当我们改变权重  $w_1$  时，每个状态的值估计会发生什么。如果我们改变  $w_1$  或  $w_2$ ，我们将改变每个状态的价值估计。现在，我们的学习结果将修改权重，而不是各个状态的值。

## How Changes to the Weight Impact the Value Function

$$w_1 = 4$$

$$w_2 = 1$$

if we change

either  $w_1$  and  $w_2$ ,

We change all approximation  
value in each state.

	1	2	$X$	3	4
1	5	9	13	17	
2	6	10	14	18	
3	7	11	15	19	
4	8	12	16	20	

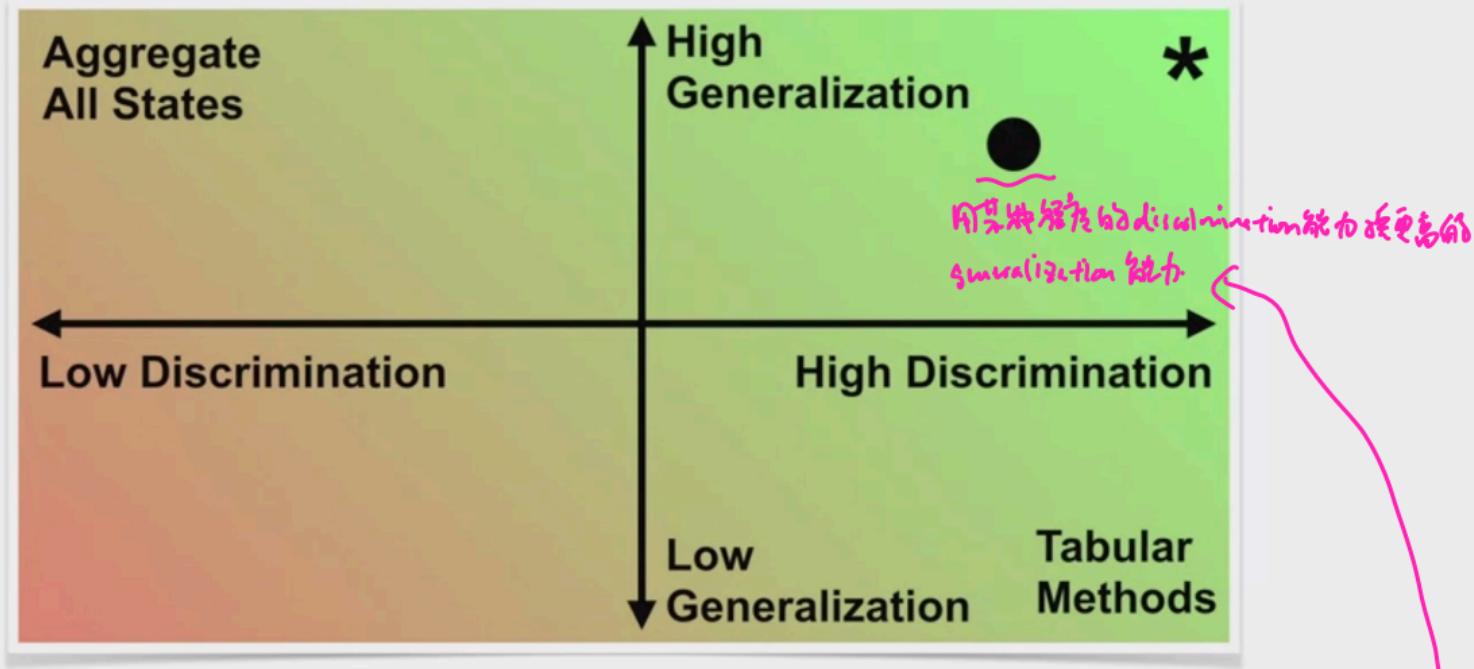
我们刚才讨论的例子是一种特殊情况，叫做 线性价值函数逼近 linear value function approximation。在这种情况下，每个状态的值由权重的线性函数表示。这只是意味着每个状态的值，被计算为权重的总和乘以状态的一些固定属性，称为特征 features。我们可以紧凑地表达这一点。我们写道，近似值是由这个特征向量和权重向量的内积给出的。我们将使用S的粗体X来表示特征向量。我们对特征的选择影响了我们可以表示的价值函数的种类。例如，考虑我们到目前为止讨论过的近似值。用它，我们只能表示作为特征X和Y的函数而线性变化的价值函数。考虑真正的价值函数是由这个网格上的数字给出的情况。我们不能将其表示为X和Y的线性函数。要使这些外在的数值正确， $w_1$ 和 $w_2$ 都需要为零。然而，这意味着这些内部值是不正确的，因为 $w_1$ 和 $w_2$ 中至少有一个必须是非零的，才能使之等于5。但我们不一定用X和Y作为特征。线性函数近似依赖于有好的特征。在这里，X和Y对这个问题来说不是好的特征。但正如你在后面所看到的，有许多强大的方法来构建特征。线性函数逼近实际上是非常普遍的。事实上，即使是表格表示也是线性函数逼近的一个特例。

为了构建一个线性值函数来表示这个表格值函数，我们需要定义特征。让我们选择我们的特征作为特定状态的指标函数。对于状态 $S_i$ ，特征 $i$ 为1，其余特征为0。我们有16个特征，每个状态一个。让我们用这些特征计算出一个状态的近似值。因为除了其中一个特征外，所有的特征都是零，所以内积等于与该状态相关的单一权重。由于每个状态的值都由一个单独的权重表示，这就相当于一个表格的价值函数。这些参数化的值是通用的，我们可以考虑很多不同类型的函数。神经网络是状态的非线性函数的一个例子。网络的输出是我们对一个给定状态的近似值。状态被传递给网络作为输入。网络中的所有连接都对应着实值权重。当数据通过一个连接时，权重会乘以其输入。这个过程通过一连串的层对输入状态进行转换，最终产生价值估计。||

也许在函数近似中最重要的考虑因素是它如何在不同状态之间进行概括。归纳是人类自然而然做的事情。一旦一个人学会了如何驾驶一辆车，他们就不必再从头开始学习如何驾驶另一辆车。他们也不必在不同的街道或下雨时从头开始。我们希望我们的代理人也能归纳总结。在本视频结束时，你将能够理解泛化 generalization 和 辨别 discrimination 的含义，了解泛化如何有益，并解释为什么我们希望从函数近似中获得泛化和辨别。泛化直观地意味着应用关于特定情况的知识来得出关于更多情况的结论。当我们在政策评估的背景下谈论泛化时，我们的意思是，对一个状态的价值估计的更新会影响到其他状态的价值。想象一下，一个负责收集罐子的机器人，通过一组距离传感器观察世界。在许多地方，开到最近的罐子需要同样的时间。即使它们对应于不同的传感器读数，这些地点也有类似的值。因此，我们可能希望价值函数能在这些状态中泛化。泛化可以通过更好地利用我们所拥有的经验来加速学习。如果我们能从类似的状态中学习它的价值，你可能就不需要经常访问每个状态来获得这个价值的正确性。

另一方面，辨别能力 discrimination 是指使两个状态的数值不同，以区分这两个状态的数值的能力。回到机器人收集易拉罐的例子，想象它处于一个易拉罐在三英尺外，但在一堵墙后面的状态。与此相对照的是，一个罐子在三英尺之外，但有一条清晰的路径可以到达。机器人会想给这些状态分配不同的值。因此，虽然在与最近的罐子有类似距离的状态之间进行归纳是有用的，但当其他信息可能影响到它们的价值时，我们根据其他信息来区分不同的状态也是很重要的。我们可以用泛化和区分的二维图来形象地描述可能的方法空间。我们到目前为止所讨论的表格方法就躺在这里。它们能完美地区分不同的状态，但完全没有泛化学习值，每个值都是独立表示的。在另一个极端，我们可以把所有的状态都看作是一样的。每个更新都会泛化到所有的状态，但完全不能进行区分。充其量，我们将能够学习与当前状态无关的平均回报，这不是很有用。我们真正想要的是一种能够实现良好概括和良好区分的学习方法。这样的方法可以将学习值泛化到类似的状态，让它学习得更快，但它也可以对不同的状态进行区分。意思是说，有了更多的数据，价值函数的近似值可以准确地代表价值。在实践中，我们更有可能在这里得到一个点，即用某种程度的区分来换取泛化。

# Categorizing Methods Based on Generalization and Discrimination



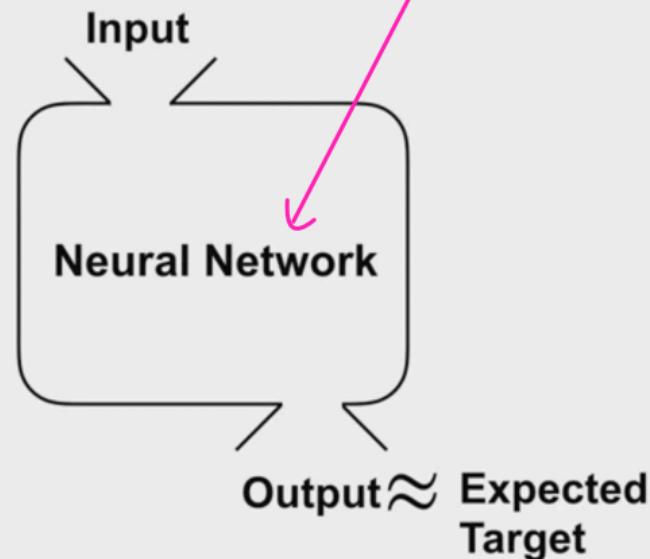
例如，我们可能会把类似的状态结合在一起，用一个数字表示它们的值。为了建立更多的直觉，让我们看一下国际象棋的具体例子。以极端的情况为例，我们把所有的状态都看作是一样的。这个数值对应的是无论游戏的状态如何都能获胜的概率。在棋手平等的情况下，这个数字可能是50%。在另一个极端，我们有表格的情况，即我们把每个状态都视为完全不同。这对小问题来说是可以的，但在象棋这样的游戏中，列举所有可能的状态是不切实际的。46种状态中大约有10种。此外，想象一下，单独学习所有这些状态的价值需要多长时间。显然，我们想要的是介于两者之间的东西，即在具有类似获胜概率的状态之间进行概括。识别这些相似性以获得这样的分组是一个困难的问题，没有单一的答案。我们如何归纳会对我们算法的性能产生重大影响，也是机器学习和强化学习的一个核心话题。这段视频就到此为止。你现在应该明白，表格表征提供了良好的辨别能力，但没有概括能力。**泛化对于快速学习很重要，同时拥有泛化和辨别力是最理想的。**虽然在实践中，通常会有一个交易。在接下来的几个模块中，我们将在研究特定函数逼近器时访问这些概念。//

监督学习方法 Supervised learning methods 从一组输入目标实例中学习一个函数。这与强化学习有很大不同。但监督学习方法对于处理强化学习的部分问题是有效的。在本视频结束时，你将能够理解价值估计是如何被设定为监督学习问题的，并认识到并非所有的函数逼近方法都适合强化学习。

监督学习涉及到对一个输入目标对 input target pairs 的数据集进行函数逼近。例如，设想我们有一个房价清单，以及每所房子的一组属性。我们可以使用监督学习来训练一个函数，将房子的属性作为输入，并估计出房子的预期价格。我们希望所学到的函数也能泛化为不在训练集中的房屋的预期价格的近似值。这个参数化的函数可以用各种方式表示。例如，一个神经网络。强化学习中的政策评估问题也可以用类似的方式来构建。这种相似性在蒙特卡洛方法的情况下最为明显。请记住，蒙特卡洛方法使用回报的样本来估计价值函数。我们可以把它看作是监督学习问题的一个例子，输入是状态，目标是回报。通过对足够多的例子进行训练，我们希望我们学习函数的输出将是对预期收益的近似。换句话说，就是每个状态下的价值。TD也可以被看作是监督学习的框架。在这种情况下，目标是一步引导的回报 one-step bootstrap return。

# Supervised Learning

$\{(Input=(2 \text{ bed, } 103 \text{ m}^2, \dots), Target=\$300000),$   
 $(Input=(1 \text{ bed, } 80 \text{ m}^2, \dots), Target=\$170000),$   
 $\dots,$   
 $(Input=(2 \text{ bed, } 103 \text{ m}^2, \dots), Target=\$515000)\}$



原则上，任何来自监督学习的函数近似技术都可以应用于政策评估任务 policy evaluation task。然而，并非所有的技术都同样适合。让我们来看看原因。在强化学习中，代理人与环境互动并不断产生新的数据。这通常被称为在线环境 online setting。为了区别于离线环境 offline setting (从一开始就有完整的数据集，并在整个学习过程中保持固定)，如果我们想使用一种函数近似技术，我们应该确保它能在在线环境下工作。

一些方法与在线环境不兼容，因为它们要么是为固定的一批数据设计的，要么不是为时间上相关的数据设计的，而强化学习的数据总是相关的。在应用监督学习的技术时，TD方法引入了一个额外的复杂问题。TD方法使用 bootstrapping，这意味着我们的目标现在取决于我们自己的估计。这些估计随着学习的进展而变化，因此我们的目标也在不断变化。这与监督学习不同，在监督学习中，我们可以获得一个地面真实标签作为目标。例如，房子的价格并不取决于我们的估计，也不会因为我们改变估计而改变。监督学习方法通常不是为改变目标或从代理人自己的估计中计算出来的目标而设计的。但是，并不是所有监督学习的方法都是强化学习的理想选择，我们需要与在线更新和引导相兼容的方法。//

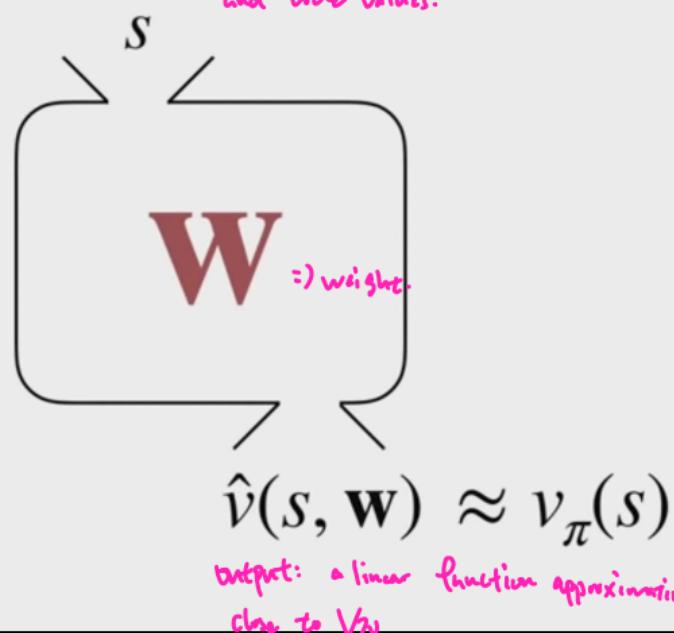
我们讨论了如何将 价值估计 value estimation 作为一个 监督学习问题 Supervised Learning problem 来看待。在监督学习中，一个重要的步骤是 找到要优化的目标。在这段视频中，我们将更精确地说明我们想要优化的目标。看完本视频后，你将能够：理解政策评估的平均平方值误差目标，并解释状态分布在目标中的作用。

让我们先想象一下一个理想化的场景。我们得到一连串的 状态和真值对 pairs of states and true values。我们想利用这些数据找到一个参数化的函数，该函数近似于  $V_{\text{Pi}}$ 。我们将通过调整权重来做到这一点，以便使函数的输出与给定状态的相关值密切匹配。一般来说，我们不能期望在所有的状态上都能完全匹配价值函数。我们选择的函数大约会限制我们可以表示的价值函数。为了使我们的目标精确，我们需要指定一些衡量我们的近似值与价值函数的接近程度。考虑一个线性价值函数的近似，用  $V_{\text{hat}}$  表示。这里为了保持可视化的简单，状态是一个单维的、连续的。让我们绘制我们对价值函数的估计。

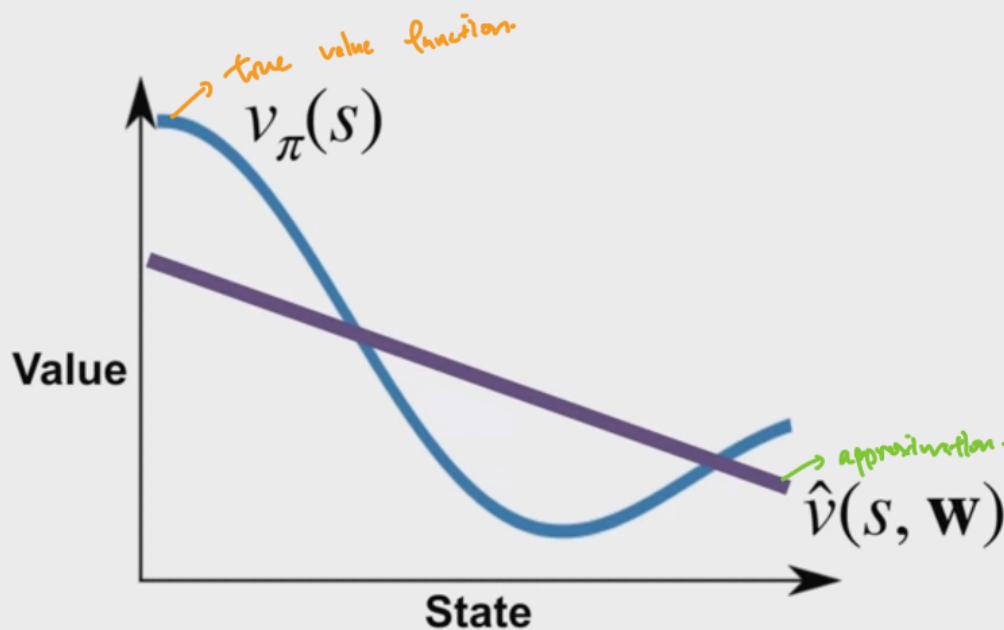
现在，想象一下真正的价值函数是这样的。显然，我们对  $V_{\text{Pi}}$  的近似值并不完美，但它到底有多大的偏差？让我们更精确地定义它。让我们先定义一个衡量状态的值和近似值之间的误差。一个自然的选择是 平方误差 squared error。就是说，数值 和 我们的近似值 approximation 之间的平方差。

## An Idealized Scenario

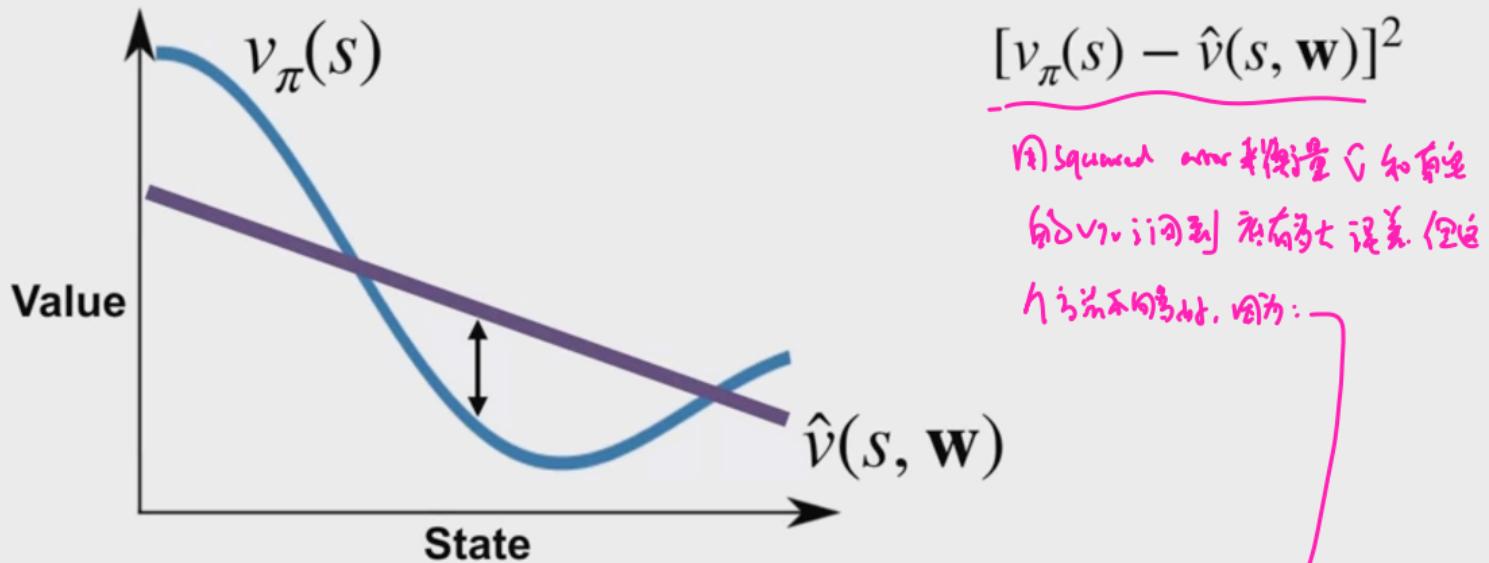
$\{(S_1, v_\pi(S_1)), (S_2, v_\pi(S_2)), (S_3, v_\pi(S_3)), \dots\}$  *input: a sequences of pairs of states and their values.*



## The Mean Squared Value Error Objective



## The Mean Squared Value Error Objective



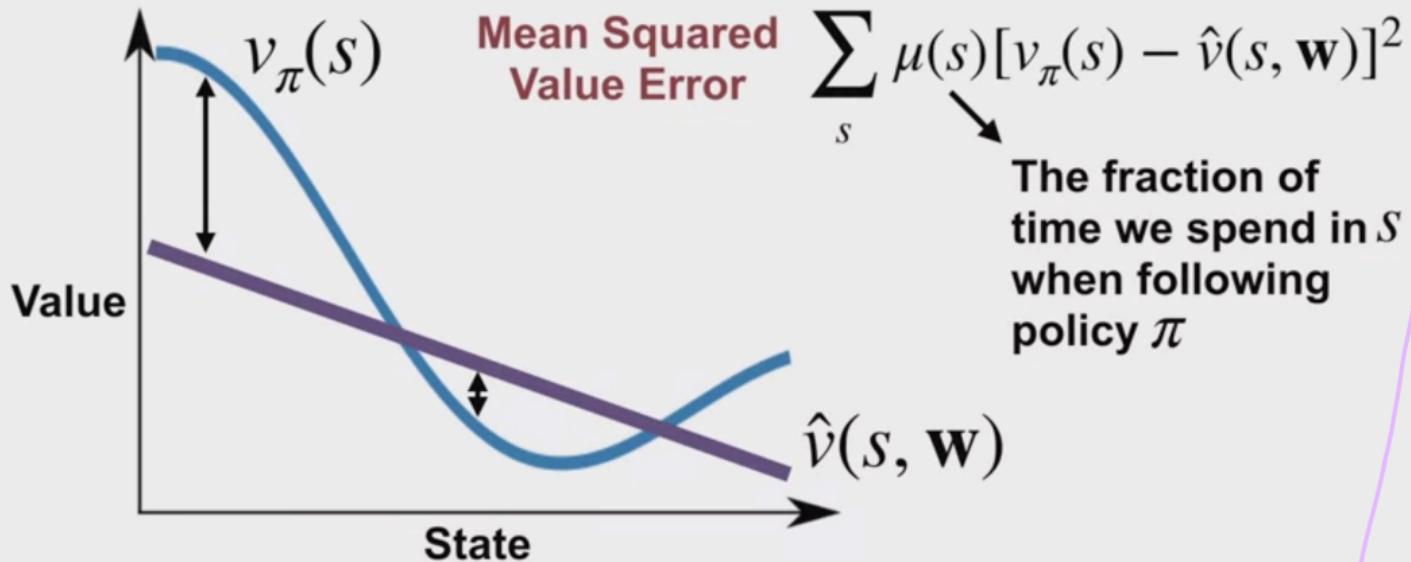
然而, 这还不足以定义一个函数近似的目。在一个状态下使估计值更准确, 往往意味着在另一个状态下使其更不准确。出于这个原因, 我们需要说明我们对每个状态下的数值的正确性的关注程度。我们将其称为  $S$  的  $Mu$ 。现在我们可以把我们的全部目标写成整个状态空间的平方误差之和, 其中每个状态都由  $Mu$  加权。我们称这个目标为 平均平方值误差 Mean Squared Value Error。

回到  $Mu$  上。我们应该从  $S$  的  $Mu$  中选择什么? 记住,  $Mu$  of  $S$  应该告诉我们对每个状态的关心程度。一个自然的衡量标准是政策下每个状态被访问的时间比例。这意味着我们要使我们在遵循  $Pi$  时访问的状态的平均价值误差最小。政策花费更多时间的状态在目标中的权重更高。我们不太关心政策较少访问的状态的误差。 $S$  的  $Mu$  是一个概率分布, 如幻灯片上所示。在这个例子中, 政策在状态空间的极端处花的时间很少。这意味着, 在平均平方值误差下, 我们允许价值近似在这些状态下有更高的误差。现在记住定义这个目标的目的。我们要调整或加权, 使平均平方值误差尽可能地低。从现在开始, 我们将这个目标称为  $VE$ 。以一种方式改变权重可能会增加值误差, 而以另一种方式改变权重可能会减少值误差。

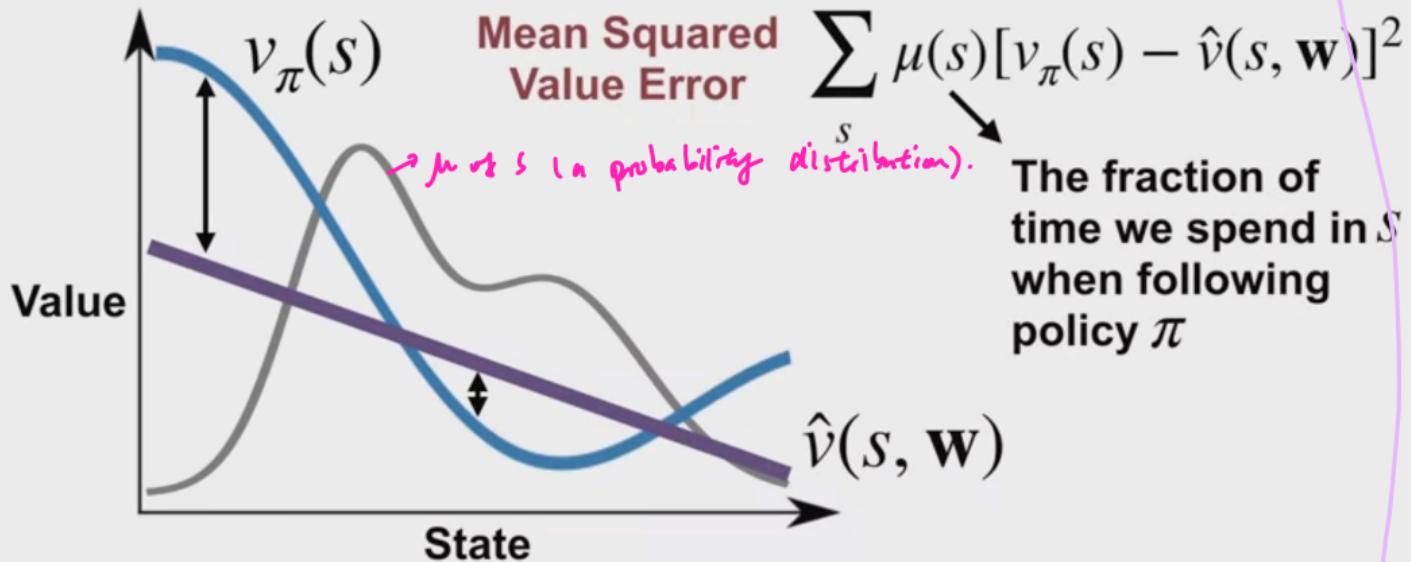
在下一讲中, 我们会讲到如何使用梯度下降来逐步调整权重。你现在应该明白为什么函数近似下的策略评估需要我们指定一个目标。平均平方值误差是一个可能的目标。//

## The Mean Squared Value Error Objective

所求值函数



## The Mean Squared Value Error Objective



## Adapting the Weights to Minimize the Mean Squared Value Error Objective

$$\overline{VE} = \sum_s \mu(s) [v_\pi(s) - \hat{v}(s, \mathbf{w})]^2$$

*if this is the last weight  
then overall value error is it.*

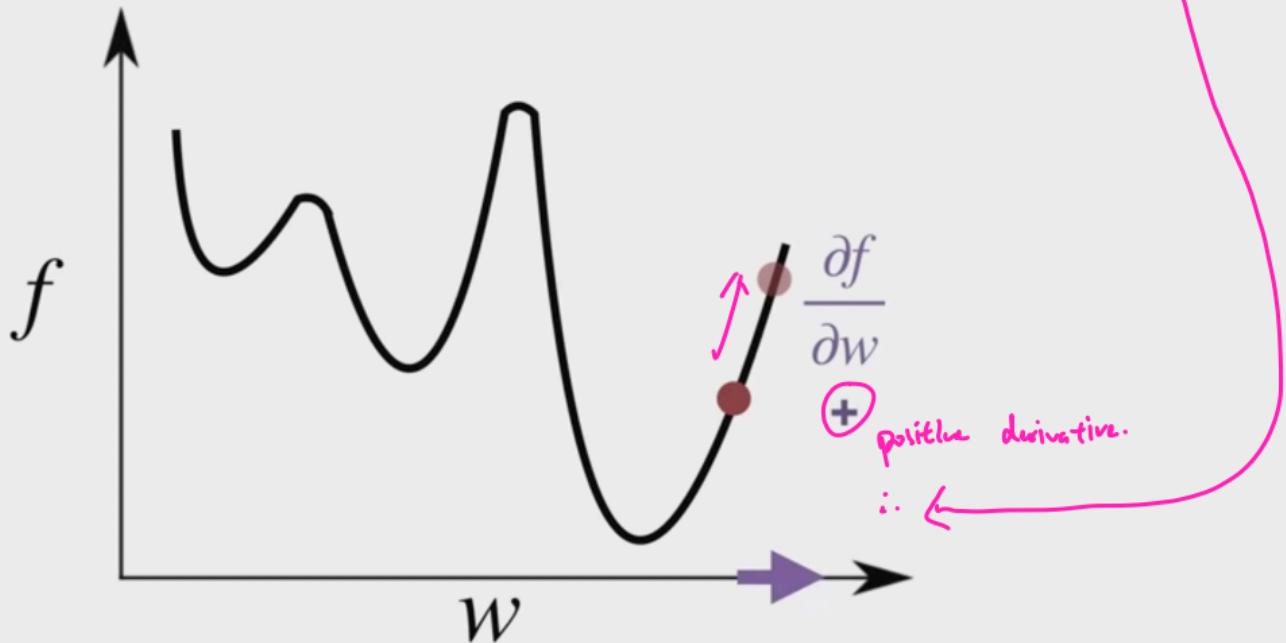
之前我们研究了价值函数的估计如何能被视为监督学习。我们通过指定一个目标使其精确。现在我们想弄清楚如何找到这个目标的解决方案。今天，我们将谈论一个最小化目标的一般策略，称为梯度下降。在接下来的视频中，我们将介绍强化学习中目标的具体算法。看完这段视频后，你将能够，理解梯度下降的概念，并了解梯度下降会收敛到静止点。我们希望最小化平均平方值误差，使我们的价值估计接近真实价值函数。

回顾一下，我们的价值估计是由状态的函数给出的，参数是一组实值权重  $W$ 。这些方式决定了每个状态的价值是如何计算的，改变权重将修改许多状态的价值估计。我们必须考虑如何改变权重以最小化整体价值误差。要做到这一点，我们需要一点微积分。希望你还记得微积分中导数的概念。这里我们绘制了一个函数  $f$ ，标量参数  $W$ 。导数告诉我们如何局部改变  $W$  来增加或减少  $f$ 。 $f$  在某一点  $W$  的导数的符号，表明改变  $W$  以增加  $f$  的方向。导数的大小表示函数  $f$  在  $W$  点的斜率，也就是说，当我们改变  $W$  时， $f$  的变化有多快。如果我们在这个方向上移动  $W$ ，我们就会增加  $f$ 。

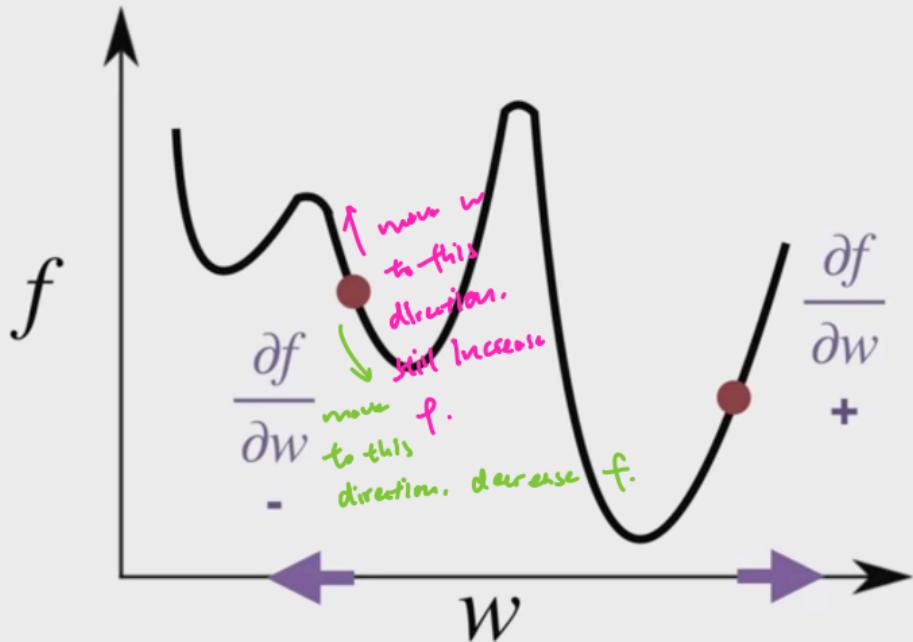
## Recap: Learning Parameterized Value Functions

$$\sum_s \mu(s)[v_\pi(s) - \hat{v}(s, \mathbf{w})]^2 \quad \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_d \end{bmatrix}$$

## Understanding Derivatives



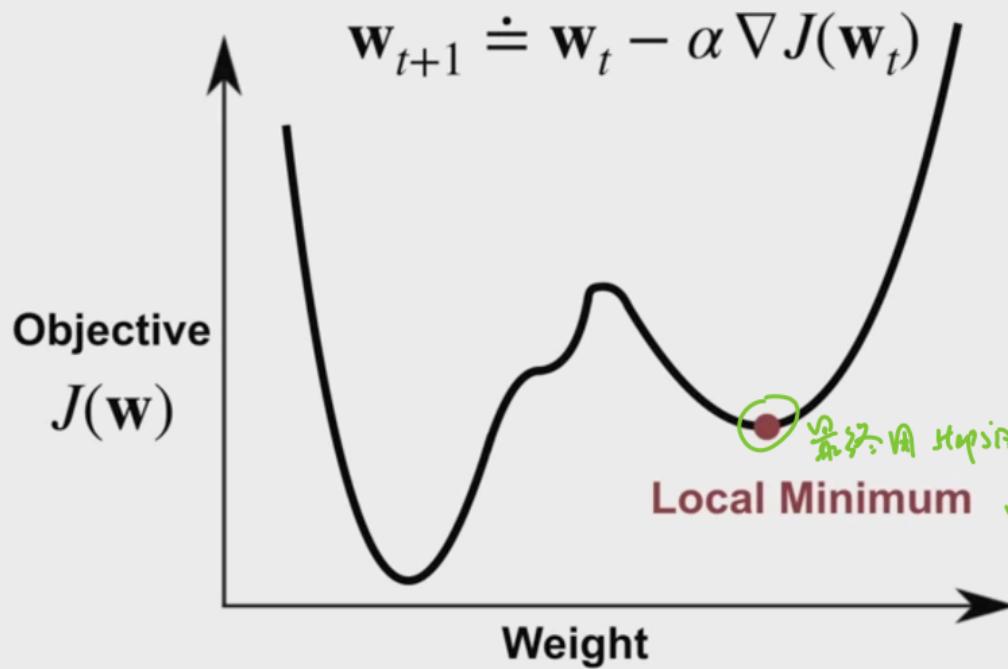
# Understanding Derivatives



在另一点上，导数是负的，它指向减少 $w$ 的方向，但这个方向仍然表明如何增加 $f$ 。如果我们把 $w$ 朝这个方向移动，它就会增加 $f$ 。如果我们把它朝这个方向的负数移动，它就会减少 $f$ 。如果 $f$ 被一个以上的变量所私有化，那么 $w$ 就是一个矢量。在这种情况下，我们需要引入梯度的概念来描述 $f$ 如何随着矢量 $w$ 的变化而变化。梯度是一个偏导数的向量，表明 $w$ 的每个分量的局部变化是如何影响函数的。请注意，梯度的大小必须与权重向量相同。梯度的每个分量的符号指定了改变 $w$ 的相关分量的方向，以便增加 $f$ 。这个方向要么是正的，要么是负的。分量的大小指定了当 $w$ 向该方向移动时 $f$ 变化的速度。如果函数在一个维度上非常陡峭，其幅度会很高。如果它非常平坦，那么该维度上的梯度就会很低。不管幅度如何，梯度给出了最陡峭的下降方向。它提供了改变 $w$ 的方向，从而使局部的 $f$ 得到最大的增加。

为了举一个具体的例子，让我们来推导一个线性值函数的梯度。记住，线性值的近似值只是权重与状态的特征向量的内积。该函数相对于单个权重的偏导，只是与该权重相关的特征。记住，特征本身并不取决于权重。这意味着线性值近似的梯度只是该状态的特征向量。我们的目标是权重的一个函数。例如，平均平方值误差是权重的一个函数，因为它是  $V \hat{}$  的一个函数，而  $V \hat{}$  是权重的一个函数。我们的目标是调整权重，使目标变小。这里我们有一个假想的目标图。为了简单起见，我们假设我们的函数是由一个单一的权重来参数化的。x轴对应于这个权重，y轴对应于这个权重的目标值。因为我们有一个单一的权重，这意味着我们的梯度只是一个标量导数。如果我们想减少我们的目标函数，我们应该在梯度的负值方向移动权重。这就是梯度下降的理念。下面是梯度下降的更新规则，抓住了这个直觉。我们在最能减少目标的方向上做小的改变。我们使用一个步长参数，即  $\alpha$ ，来控制我们的移动距离，否则就会有步长过大的风险。这是因为向这个方向移动只能保证局部减少目标。通过在足够小的  $\alpha$  下反复做这样的更新，我们最终会收敛到一个梯度为0的静止点。这意味着这些权重比附近的所有权重都要好，但不一定是最好的。

## Gradient Descent

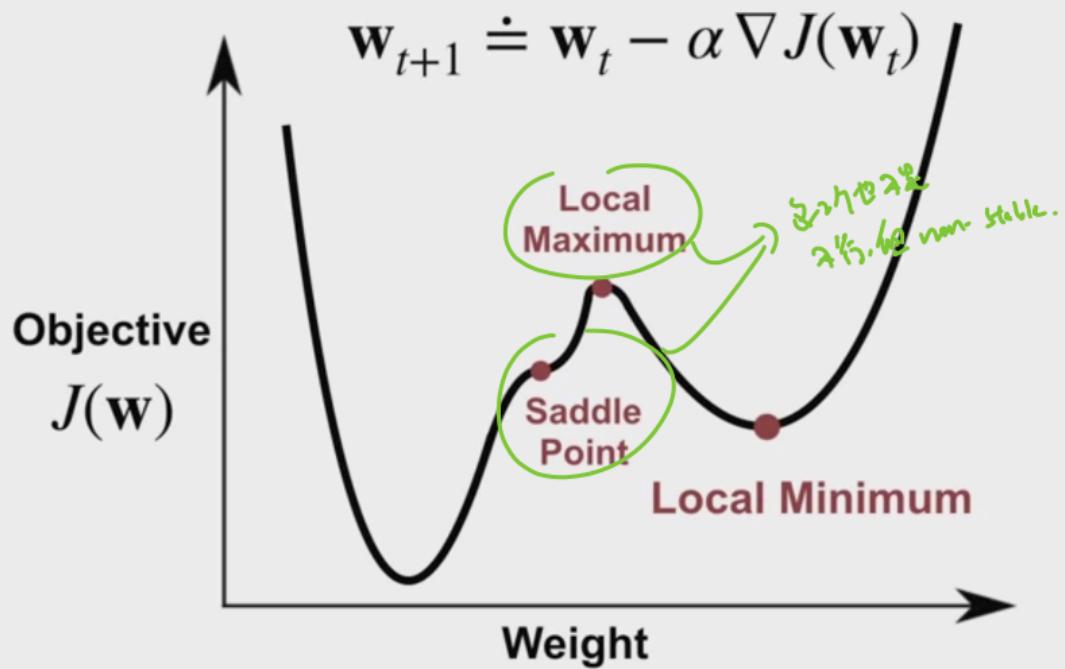


其他可能的 静止点 stationary points 是 局部最大值 local maxima 和 鞍点 saddle points。这些都是不稳定的解决方案，我们算法中固有的随机性通常足以防止卡在这些可怜的静止点。另一方面，局部最小点是稳定的，所以即使是随机算法也会倾向于在那里收敛。出于这个原因，我们通常谈论我们的算法收敛到局部最小值。

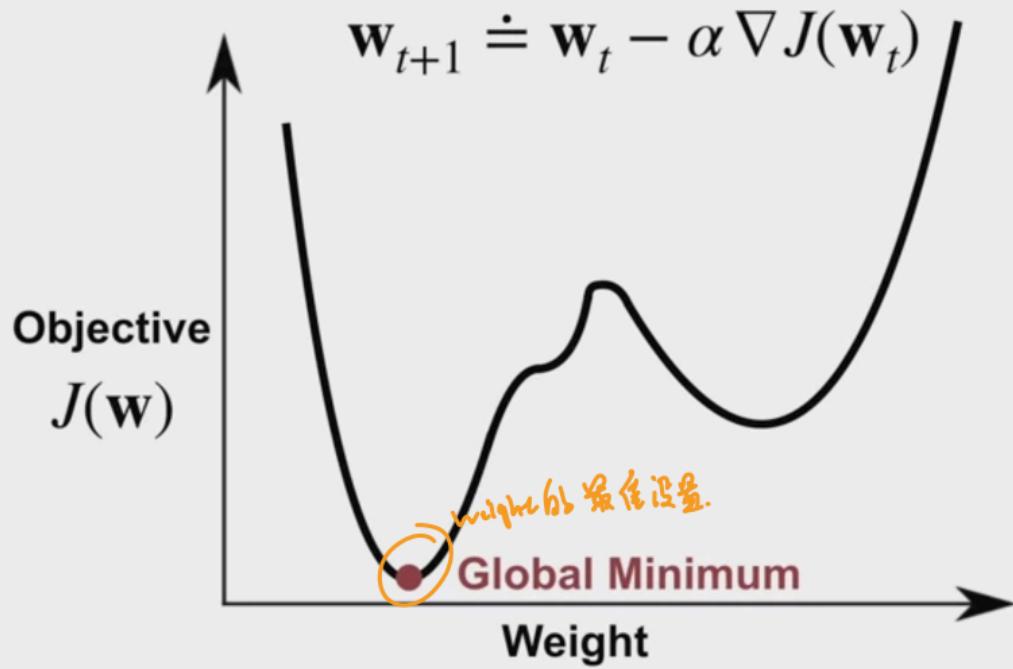
在某些情况下，梯度下降可以保证收敛到全局最小值，也就是目标的权重的最佳设置。例如，对于线性函数近似的平均平方值误差就是如此。对于更复杂的函数近似误差，如神经网络，静止点不一定是全局最小值。请注意，全局最小值并不一定对应于真实的函数。它受限于我们对函数参数化的选择，并取决于我们对目标的选择。想象一下，一个只包含一个元素的特征向量，无论你处于什么状态，它总是一个。使均值误差最小化的近似值函数，将收敛于所有状态下的平均值。这不是一个很好的价值函数。然而，就价值误差目标而言，这仍然是我们在该参数化下所能做到的最好的。

这段视频就到此为止。你现在应该明白梯度下降法如何被用来寻找目标的静止点，而且这些解决方案并不总是全局最优的。下一次我们将讨论如何专门为我们的误差使用梯度下降。到时见。||

## Gradient Descent



## Gradient Descent



你现在知道了一个最小化目标的策略，梯度下降，我们为政策评估制定了一个明确的目标，即价值误差 ( $\text{value error}$  (差值  $\rightarrow$  minimized))。我们现在应该能够把这些放在一起进行近似取值。让我们回到我们的老朋友--蒙特卡洛，看看如何用它来最小化价值误差。看完这段视频后，你将能够理解如何使用梯度下降和随机梯度下降来最小化数值误差，并概述了用于数值估计的梯度蒙特卡洛算法。

我们刚刚看到如何使用梯度下降法来最小化目标。要使用这种方法，第一步是要找到你的目标的梯度。因此，让我们从计算平均平方值误差的梯度开始，相对于我们近似值的权重而言。记住，这个目标是所有状态下的平方误差的加权和。按照微积分的规则，我们可以把梯度拉到总和里面。然后，我们取和内每项的梯度，这需要使用链式规则。 $v_{-Pi}$ 的梯度减去 $v_{\text{hat}}$ ，等于 $v_{\text{hat}}$ 的梯度，因为 $v_{-Pi}$ 不是 $w$ 的函数，换句话说，改变 $w$ 不会改变 $s$ 的 $v_{-Pi}$ 。在线性函数近似的情况下，梯度特别容易计算，正如我们在之前的视频中展示的那样。它只是该状态下的特征向量。这个梯度是有意义的。值函数近似的梯度表明如何改变权重以增加该状态的值。我们把这个方向乘以真实值和我们的估计值之间的差异。如果差值是正的，意味着真值比我们的估计值高，所以我们应该朝增加估计值的方向改变权重。如果当前的误差是负的，我们应该朝相反的方向改变权重。计算平均平方值误差的梯度需要对所有状态进行求和。这通常是不可行的。而且，我们很可能不知道 $Mu$ 的分布。

## Gradient of the Mean Squared Value Error Objective

$$\begin{aligned} & \nabla \sum_{s \in \mathcal{S}} \mu(s) [v_{\pi}(s) - \hat{v}(s, \mathbf{w})]^2 & \hat{v}(s, \mathbf{w}) \doteq \langle \mathbf{w}, \mathbf{x}(s) \rangle \\ &= \sum_{s \in \mathcal{S}} \mu(s) \nabla [v_{\pi}(s) - \hat{v}(s, \mathbf{w})]^2 & \nabla \hat{v}(s, \mathbf{w}) = \mathbf{x}(s) \\ &= - \sum_{s \in \mathcal{S}} \mu(s) 2 \underbrace{[v_{\pi}(s) - \hat{v}(s, \mathbf{w})]}_{\substack{\uparrow \\ + \\ -}} \nabla \hat{v}(s, \mathbf{w}) \\ \Delta \mathbf{w} &\propto \sum_{s \in \mathcal{S}} \mu(s) [v_{\pi}(s) - \hat{v}(s, \mathbf{w})] \nabla \hat{v}(s, \mathbf{w}) \end{aligned}$$

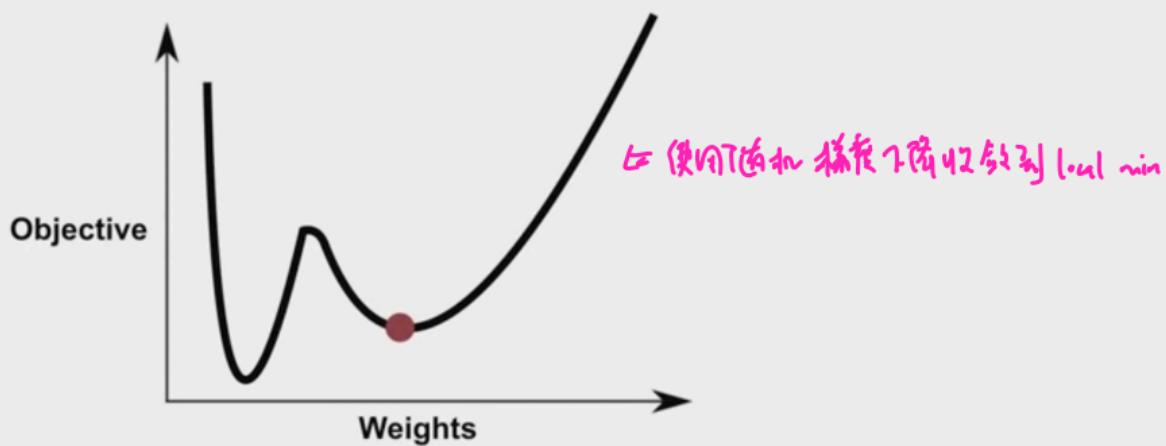
相反，让我们对这个梯度进行近似计算。设想一个理想化的环境，我们可以获得 $v_{Pi}$ 。尽管我们没有明确的 $Mu$ ，但我们可以从遵循政策中抽取状态。让我们取其中的一个状态， $S_1$ ，它是在遵循政策时发生的，目标是 $S_1$ 的 $v_{Pi}$ 。我们可以用这一对来进行更新，以减少该例子的误差。这里是单个状态的梯度。我们可以用这个梯度做梯度下降，以减少该状态下的错误。所以这里是 $S_1$ 的相应梯度更新规则。我们可以执行这个更新来减少这一对的误差。然后我们可以对另一个状态， $S_2$ ，在跟踪 $Pi$ 时观察到的状态再做一次。通过在改善每一对误差的方向上进行小的更新，我们有时可能会增加整个目标的误差。但总的趋势将是在整个目标上取得进展。这种更新方法被称为随机梯度下降，因为它只使用梯度的随机估计。

事实上，每个随机梯度的期望值等于目标的梯度。你可以把这种随机梯度看作是对梯度的一种嘈杂的近似，它的计算成本要低得多，但仍然可以稳定地达到最小。随机梯度下降使我们能够通过对梯度的抽样来有效地更新每一步的权重。然而，还有一个实际问题。我们无法获得 $v_{Pi}$ 。让我们看看如何从我们的更新规则中去掉它。让我们用一个估计值取代 $v_{Pi}$ 。一个选择是使用每个访问状态的回报样本 $S_t$ ，就像我们对蒙特卡洛方法所做的那样。这很有意义，因为价值函数是这些样本的期望值。事实上，当我们用采样的返回值代替真实值时，梯度的期望值仍然等于均方值误差的梯度。这就给我们带来了估计 $v_{Pi}$ 的梯度蒙特卡洛算法。我们采取任何我们希望评估的政策 $Pi$ ，我们选择一些函数，将状态和权重映射到一个在权重上可微调的实数。对于一个给定的权重向量，这个 $v$ 帽子是一个产生近似值的状态的函数。我们选择一个步长 $Alpha$ 的值，并以我们喜欢的方式初始化我们估计的权重参数。例如，为零。在每个迭代中，代理人与环境互动，产生一个完整的情节。我们计算每个访问状态的采样回报。然后，我们循环浏览剧情中的每一步，并根据采样的回报进行随机梯度和更新。这段视频就到此为止。你现在应该明白如何用随机梯度下降来估计一个值函数，以及梯度蒙特卡洛算法是如何工作的。

## From Gradient Descent to Stochastic Gradient Descent

$$\sum_{s \in \mathcal{S}} \mu(s) 2[v_\pi(s) - \hat{v}(s, \mathbf{w})] \nabla \hat{v}(s, \mathbf{w})$$

$(S_1, v_\pi(S_1)), (S_2, v_\pi(S_2)), (S_3, v_\pi(S_3)), \dots$



所有的剧情都将在队伍的中间开始，在500号状态。有两个动作，左边和右边。左边的动作使代理人向左移动，但不一定只是一个地方。

代理人向左跳到100个相邻状态内的任何状态，均匀地随机进行。同样地，右边的动作也是向右跳，跳到邻近的100个状态中的一个。让我们评估一下统一的随机策略，它以相等的概率向左或向右移动。靠近终点状态的国家，在左边，有少于100个邻居在左边。

所有会跳过链条末端的动作都会转到终端状态。同样，在右边也是如此。

左边的结束给的奖励是减1，右边的结束给的奖励是加1。所有其他转换的奖励为0，折扣伽马为1。我们可能会从某种函数的近似中受益。

在这里我们将使用一种叫做 **状态聚合 state aggregation** 的技术。顾名思义，状态聚合将某些状态视为相同的。在这个由八个状态组成的表格中，我们可能会选择将状态以四组的形式聚合在一起。所以，现在我们的值函数表不是有八个条目，而是只有两个。当我们更新第一组中任何一个状态的值时，该组中所有其他状态的值都会被更新。

状态聚合是线性函数近似的另一个例子。每组状态都有一个特征。如果当前状态属于相关组，每个特征将为1，否则为0。一个状态的近似值是与该状态所属的组相关的权重。

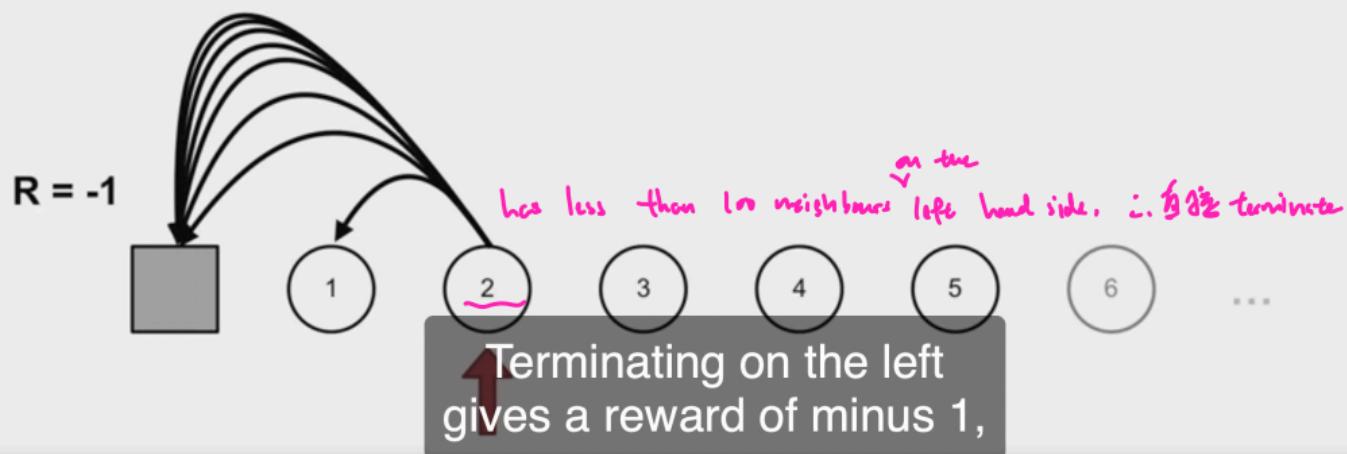
让我们来看看带有状态聚合的梯度蒙特卡罗算法。我们已经知道如何计算一个给定状态的值了。现在我们需要考虑如何计算值函数的梯度，以便我们可以使用我们的密钥更新公式。状态聚合是线性函数近似的一个例子，所以梯度等于特征向量。

更新规则只修改了与当前活动组对应的权重。让我们思考一下这个更新规则是如何改变权重的。如果估计值小于返回 $G_t$ 的样本，那么权重就会增加。如果估计值大于回报率 $G_t$ 的样本，那么权重就会减少。

让我们回到随机漫步，看看我们如何应用状态聚合。首先，我们必须选择如何聚合状态。状态聚合迫使同一组中的状态使用相同的值估计。因此，理想的情况是，如果我们有理由相信各州的价值会相似，我们就应该把它们组合在一起。

# Random Walk Example

Policy  
50%  $\longleftrightarrow$  50%



# Random Walk Example

Policy  
50%  $\longleftrightarrow$  50%

$$\gamma = 1$$

Learn how states spend lot of time

in we use state aggregation which make certain state as the same.

$$R = +1$$



terminating the right  
gives a reward of plus 1.

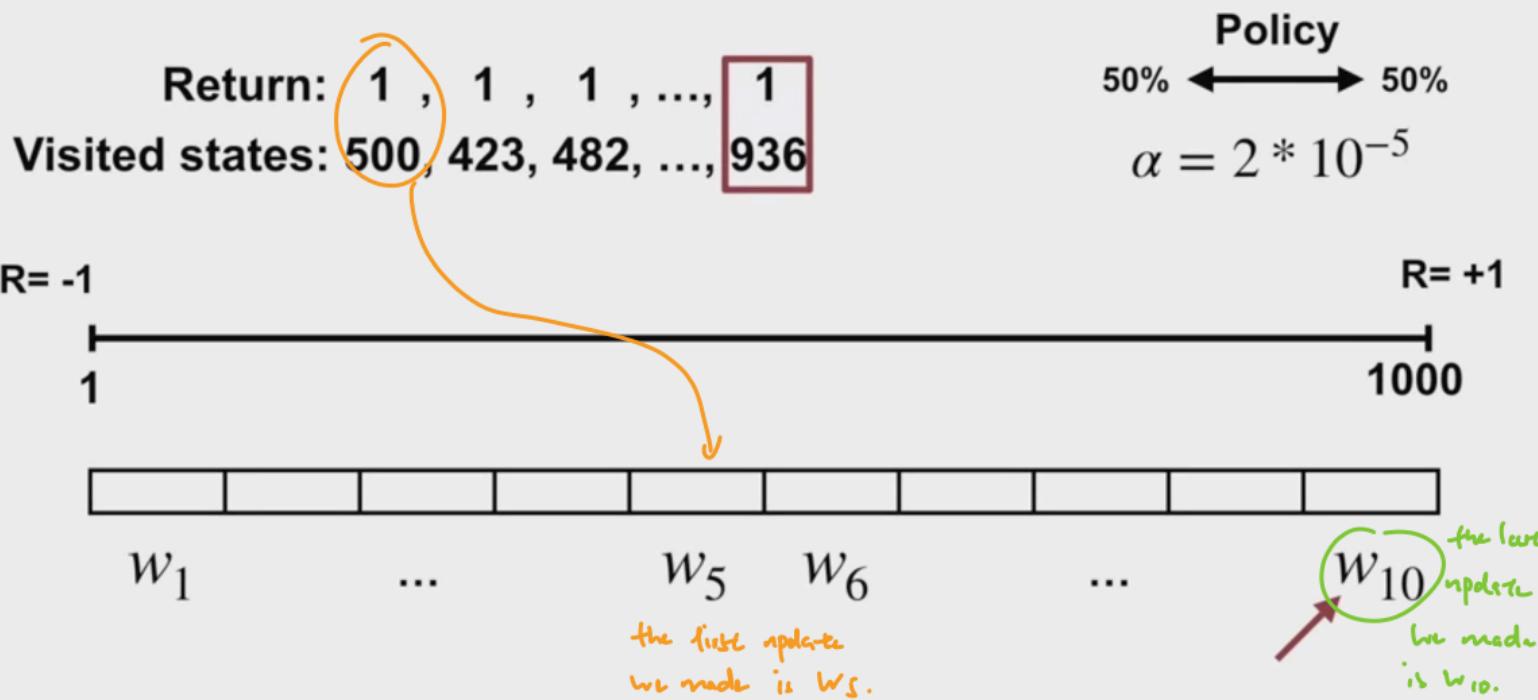
# Constructing a State Aggregation for the Random Walk

States: 1 1000

1-100 901-1000

将 8192 个 state 用 state aggregation 归为 10 个组, 对一个物理块 1000 h state 重新解决.

## Monte Carlo Updates for a Single Episode



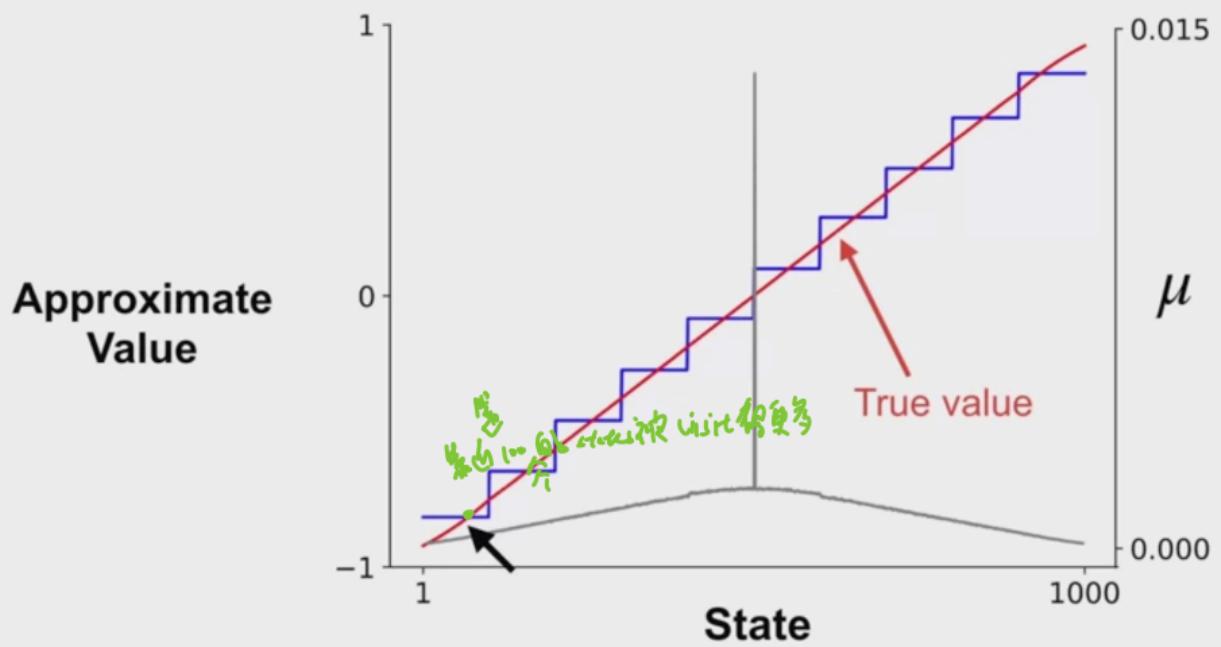
After running for many episodes, we get state values that look something like this. Each step in the plot corresponds to our group of states that share the same approximate value. For comparison, here's the true value function. Although we have heavily approximated by aggregating many states together, our value estimate is not that far off. Why doesn't the red line pass directly through the center of all the blue steps? This is where  $\mu$  plays an important role. Recall  $\mu$  of  $s$  is the visitation frequency for state  $s$ . States near the center of the chain are visited more often than those near the terminal states, as depicted here. For example, let's look at the leftmost group of states, states 1 to 100. States near state 100 are visited much more than states near state 1. The approximate value is skewed towards the true value for states near state 100. This is because  $\mu$  weights these states higher in the value error.

That's it for this video. You should now have an idea of how gradient Monte Carlo works in practice when combined with state aggregation. See you next time.

在运行了许多集之后，我们得到的状态值看起来像这样。图中的每一步都对应于我们这组具有相同近似值的状态。作为比较，这里是真正的价值函数。尽管我们通过将许多状态聚集在一起进行了大量的近似，但我们的价值估计并没有那么大的偏差。为什么红线没有直接穿过所有蓝色台阶的中心？这就是 $\mu$ 发挥重要作用的地方。回想一下， $s$ 的 $\mu$ 是状态 $s$ 的访问频率。靠近链中心的状态比靠近终端状态的状态被更频繁地访问，如这里所描述的。例如，让我们看一下最左边的一组状态，即状态1到100。靠近状态100的状态比靠近状态1的状态被访问得多。状态100附近的状态的近似值是偏向真实值的。这是因为 $\mu$ 在数值误差中对这些状态的权重更高。

这段视频就到此为止。你现在应该对梯度蒙特卡洛与状态聚合结合时的实际工作有了一个概念。下回见。

## Final Value Estimates



TD学习是建立在代理人可以使用自己对价值函数的估计来更新其预测的想法上。我们已经在表格的情况下看到了TD学习。今天我们将描述它是如何与函数近似工作的。

在本视频结束时，你将能够理解函数近似的TD更新，并概述价值估计的半梯度TD算法。回顾一下梯度蒙特卡洛更新方程。它将我们当前的价值估计更新为更接近回报率 $G_t$ 的样本，但我们可以考虑用其他目标代替回报率。事实上，我们可以在这个更新中用任何价值估计代替回报。

我们把这个估计值称为 $U_t$ 。如果 $U_t$ 是对真实价值的无偏估计，那么我们的函数近似器将在适当的条件下收敛到一个局部最优。

对于回报率来说就是这样，但是我们也可以用一个~~引~~<sup>bootstrap</sup>目标代替 $U_t$ ，比如一步TD目标。这仍然是对收益的估计，但在这种情况下，估计是有偏差的。TD目标使用我们当前的价值估计，这很可能不等于真正的价值函数。正因为如此，我们不能保证这个算法会收敛到价值误差的局部最小值。

好处是，TD目标往往比回报的样本有更低的方差。这意味着TD会在较少的更新中趋于收敛。TD更新实际上不是一个随机梯度下降更新。为了了解原因，我们把误差的梯度与一个样本的权重联系起来。

记住 $U_t$ 等于TD的目标。使用链式规则，我们得到一个样本的平方误差梯度的扩展表达式。但是，等一下。这看起来并不像TD更新。这两个表达式只有在 $U_t$ 的梯度=0的情况下才会相等，但TD不是这样的。**在TD中，目标包含一个估计值，它取决于权重。这意味着 $U_t$ 的梯度不是这里所示的0，因此TD不是对平方误差进行梯度下降更新。**我们称它为半梯度方法。尽管如此，在我们关心的许多情况下，TD还是收敛了。我们将在接下来的视频中讨论TD的收敛特性。这里是半梯度TD的伪代码。它实际上与表格设置的TD参数非常相似。**这个专辑不需要等到一集结束时再进行更新。TD在每一步都进行更新，这与无线电蒙特卡洛不同。**在情节的每一步，**代理人在状态S中选择一个行动A。我们以这种方式继续下去，直到我们到达终端状态，它被定义为值为零，然后我们开始一个新的情节，就这样。**这其实与表格中的TD没有什么不同。//

## TD is a semi-gradient method

$$\nabla \frac{1}{2} [U_t - \hat{v}(S_t, \mathbf{w})]^2$$

$$U_t \doteq R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$$

$$= (U_t - \hat{v}(S_t, \mathbf{w}))(\nabla U_t - \nabla \hat{v}(S_t, \mathbf{w})) \neq - (U_t - \hat{v}(S_t, \mathbf{w})) \nabla \hat{v}(S_t, \mathbf{w}) \quad \nabla U_t = 0$$

The TD Update

For TD:

$$\nabla U_t = \nabla (R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}))$$

$$= \gamma \nabla \hat{v}(S_{t+1}, \mathbf{w})$$

$$\neq 0$$

## Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy  $\pi$  to be evaluated

Input: a differentiable function  $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$  such that  $\hat{v}(\text{terminal}, \cdot) = 0$

Algorithm parameter: step size  $\alpha > 0$

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

Loop for each episode: 

    Initialize  $S$

    Loop for each step of episode: 

        Choose  $A \sim \pi(\cdot | S)$

        Take action  $A$ , observe  $R, S'$

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

$S \leftarrow S'$

    until  $S$  is terminal

$\hat{v}(\text{terminal}, \cdot) = 0$

$\Rightarrow$  reach the terminal state.

我们刚刚谈到了带函数近似的蒙特卡洛和带函数近似的TD。但是这些算法有什么不同呢？今天，我们将更多地讨论TD更新中的偏差，并在一个小实验中把它与蒙特卡洛进行比较。看完这段视频后，你将能够：理解TD收敛到一个偏置值的估计，并理解TD可以比梯度蒙特卡洛学习得更快。

在合理的假设下，梯度蒙特卡罗 Gradient Monte Carlo 将随着越来越多的样本接近平均平方值误差的局部最小值。这是因为它使用了对价值误差梯度的无偏估计。理论上，我们需要运行算法很长的时间，并衰减步长参数来获得这种收敛。在实践中，我们使用一个恒定的步长。因此，该算法围绕局部最小值进行振荡。

另一方面，TD目标取决于我们对下一个状态下的数值的估计。这意味着我们的更新可能是有偏见的，因为我们目标中的估计可能不准确。由于我们的数值近似永远不会是完美的，即使在极限状态下，目标可能仍然是有偏见的。我们不能保证半梯度TD会收敛到平均平方值误差的局部最小值。当然，这种偏差会随着其估计的改善而减少。但是，所有这些理论上的担忧如何影响实践中的表现呢？

让我们回到1000个状态的随机漫步的例子，看看TD更新中的偏差的影响。这一次，我们将使用半梯度TD而不是蒙特卡罗来学习近似值。在运行TD很长时间后，比如我们的价值估计已经停止变化，这就是我们得到的结果。在许多状态下，值估计与真实值不太一致。TD的估计值不如Monte Carlo做的估计值准确。在这个领域，即状态聚合，Monte Carlo在长期性能方面有优势。

但是，学习的速度呢？TD和Monte Carlo哪个能最好地利用有限的样本？让我们做一个不同的实验来弄清这个问题。为了专注于早期学习，我们只运行30集。请记住，在上一个实验中，我们使用了1,000个事件。两种算法的性能都取决于我们如何设置步长参数Alpha。这两种算法可能需要非常不同的Alpha值。为了做一个公平的比较，我们用100个均匀分布的Alpha值来测试每个算法，这些值在0和1之间。我们为每种算法选取表现最好的Alpha值，并比较其性能。最好的Alpha被定义为在30次训练后产生最低值误差的那个。对于这项任务，我们发现TD的最佳阿尔法是0.22。Monte Carlo的最佳Alpha值要小得多，为0.01。让我们绘制每个算法的性能。y轴显示的是1000个状态的均方根值误差，平均100次运行。X轴显示了我们的集数。我们期望每种算法的值误差开始时很高，并随着越来越多的插曲逐渐减少。让我们看一下结果。我们看到TD更快地达到一个较低的误差。这不是一个一次性的结果。TD往往比Monte Carlo学习得更快。这是因为TD可以在事件中学习，并且有较低的方差更新。蒙特卡洛在长期性能上更好，这并不总是主要关注的。我们永远不可能在实验中达到渐进性能。早期学习在实践中也许更重要。

你现在已经看到了函数近似的TD更新是如何有偏差的。但无论如何，我们常常喜欢TD学习而不是蒙特卡罗，因为它能更快地学习。//

原则上，我们可以使用任何种类的函数近似来进行强化学习。线性函数近似是一个重要的特例。它很简单，我们可以很好地理解它，但它在一般情况下仍然是相当强大的。事实上，用TD的线性函数逼近 linear function approximation with TD 已经被用来建立Atari代理，在许多游戏中超过人类的表现。看完这段视频后，你将能够推导出线性函数逼近的TD更新，理解表格TD是线性半梯度TD的一个特例，并理解为什么我们关心线性TD这个特例。

回顾 半梯度TD semi gradient TD 的更新。它调整权重和TD空气的方向乘以近似值函数相对于权重的梯度。在线性情况下，一个状态的近似值的梯度只是该状态的特征向量。所以半梯度TD的更新是这样的。权重按照特征向量乘以TD空气的方向更新。如果一个特征很大，那么相应的权重就会对预测产生很大影响。另一方面，如果该特征为零，那么该权重对预测没有影响，因此梯度为零。专家设计特征给出的固定基础对更新有很大影响。如果设计得好，我们可以通过简单的更新得到有效的价值函数近似。我们已经看到了线性价值函数逼近是对表格价值函数逼近的严格概括。我们可以证明，线性TD是表格TD和带状态聚合的TD的严格概括。

让我们仔细看一下算法，明确地看到表格式TD的这一点。想象一下，我们有一个表格式的状态表示。只有一个特征将等于1，对应于当前的状态，其余的都是0。一个状态的值近似等于与当前状态相关的权重。因此，我们可以认为权重向量只是一个数值表，每个状态都有一个。下面是带有这些表格式特征的半梯度TD的更新。在更新中， $ST$ 的特征向量 $X$ 选择了一个与当前状态相关的单一权重。这个权重只是该状态的价值估计。所以这个更新对应于我们在以前的课程中看到的表格式TD更新。我们可以用同样的分析来说明，带有状态聚合的TD也是线性TD的一个特例。

## Tabular TD is a special case of linear TD

without aggregation TD to ?.

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})] \mathbf{x}(S_t)$$

$$w_i \leftarrow w_i + \alpha [R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})] \mathbf{1}$$

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

*i-th element*

$$\hat{v}(s_i, \mathbf{w}) = w_i$$

但为什么我们如此关心线性函数逼近的特例呢？线性方法在数学上更容易理解和分析。TD学习的大部分理论都是针对线性设置的。我们将在下一个视频中进一步讨论这个问题。在一些应用中，你可能有机会获得专家知识来帮助你设计好的特征。也许，你是领域专家，也许你可以和对你的应用非常了解的人交谈。如果我们能够很好地设计这些特征，那么我们就可以期望线性方法能够快速学习并达到良好的预测精度。特征设计提供了一种结合领域知识的方法，可以在实践中获得良好的性能。

今天就说到这里。我们谈到了线性TD更新，表格TD是线性半梯度TD的一个特例，以及为什么线性函数近似可能是有用的。//

回顾一下，在表格设置中，我们将TD描述为解决贝尔曼方程的一种基于样本的方法。线性TD类似于对贝尔曼方程的解进行逼近，使所谓的预测贝尔曼误差最小化。这个目标的细节超出了本课程的范围，但你可以参考课程课本以了解更多细节。关键的启示是，即使TD没有收敛到均方值误差的最小值，但它确实收敛到了基于贝尔曼方程的原则性目标的最小值。尽管如此，我们仍然想了解TD找到的解决方案和最小值误差解决方案之间的关系。我们可以用这个方程来正式描述这种关系。如果Gamma接近于1，那么TD固定点和最小值误差解之间的差异会很大。另一方面，如果Gamma非常接近于零，那么TD固定点就非常接近最小值误差解。这个界限也取决于特征的质量。如果特征是有限的，最小均值误差和价值，即TD固定点 (TD fixed point)，都可能很大。如果我们能完美地表示价值函数，那么无论Gamma如何，TD固定点都等同于最小价值误差解 minimum value error solution。这是因为左手边和右手边都会是零。

那么一般来说，为什么TD固定点不等于最小值误差解呢？这是因为函数近似下的~~bootstraping~~。如果我们对下一个状态的估计由于函数的逼近而持续不准确，那么TD就会永远朝着不准确的目标更新。另一方面，如果我们的函数逼近器非常好，那么我们对下一个状态的估计将变得非常准确。所以从这个估计值上~~是~~是<sup>bootstraping</sup>没有问题的，而且TD解决方案的误差也接近于最小值误差。

今天就说到这里。你现在应该知道更多关于为什么线性半梯度TD能保证收敛到一个叫做TD固定点的固定点，以及TD固定点与最小均方值误差的关系。//

本周，我们学到了很多关于如何将表格情况下的强化学习扩展到函数近似。在许多现实世界的问题中，在表格中列举所有可能的状态是不现实的。因此，转向函数近似是使强化学习更广泛适用的一大步。在这段视频中，我们将对我们所学到的知识做一个快速回顾。

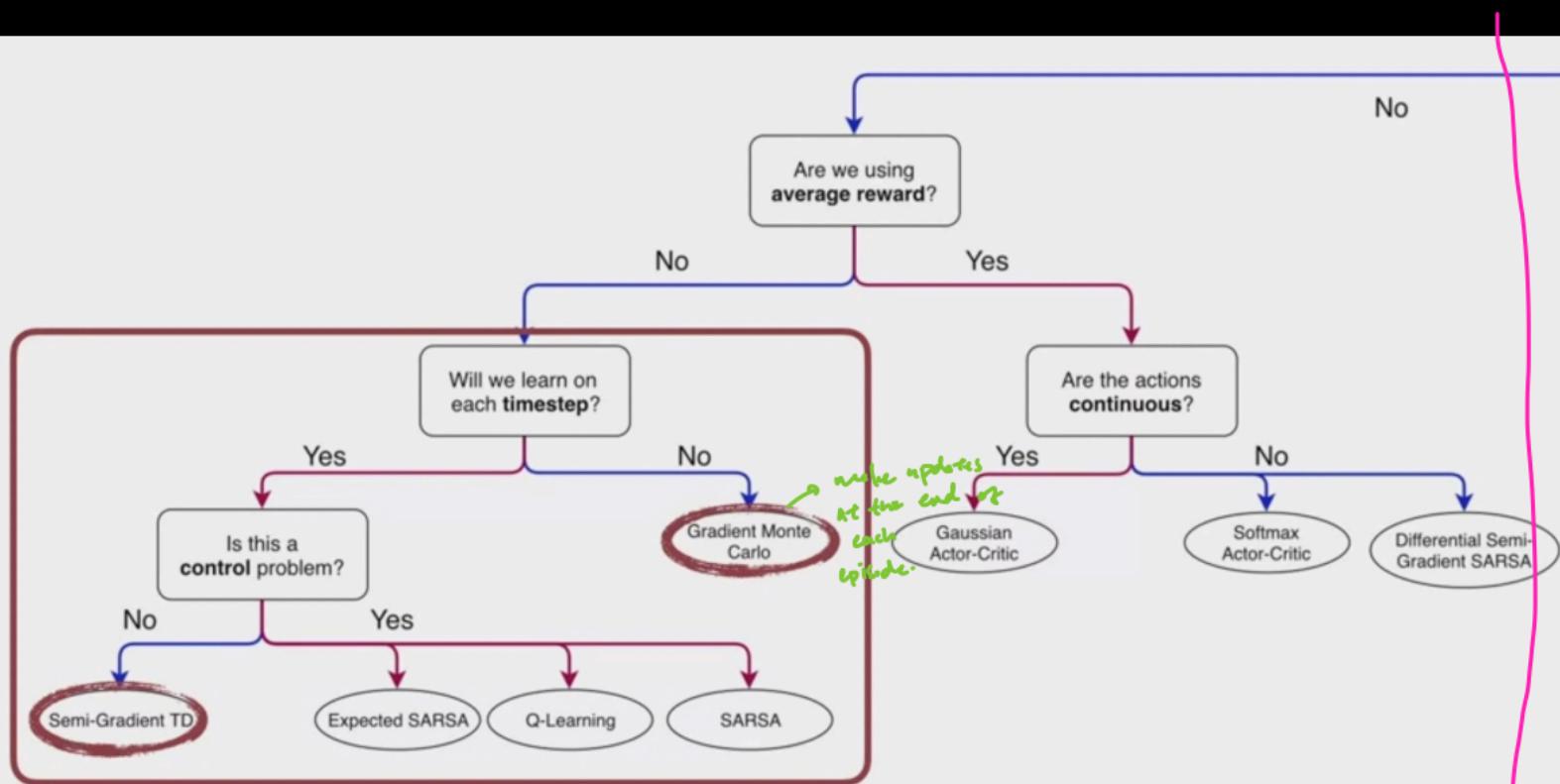
我们已经涵盖了很多材料，很容易被所有不同的概念所迷惑。选择退一步，看看本周的主题是如何融入大局的。

这个地图显示了我们在这个专业中所涉及的所有不同算法，以及它们之间的关系。记住，地图只显示了我们在专业课中提出的项目。本周的主要概念变化是转向参数化函数近似的框架。这意味着，我们不再使用表格来存储每个状态的值。这使我们处于地图的左侧。我们对平均报酬的讨论将在后面进行。现在，让我们把重点放在这里。

本周，我们重点讨论政策评估问题。我们涵盖两种算法。梯度蒙特卡洛 Gradient Monte Carlo 和 半梯度TD (Semi-Gradient TD)。梯度蒙特卡洛在每集结束时进行更新。TD使用价值估计和下一个时间步骤进行 bootstraps。因此，它不需要等到一集结束时再学习。

现在我们已经有了大的图景，让我们快速回顾一下一些细节。我们把一个参数化的价值函数描述为一个从状态到实数的映射。输出由一个实值权重  $W$  的向量控制。我们用  $S$  和  $W$  的符号  $V^{\hat{w}}$  来表示近似的价值函数。这表明价值估计是由当前状态和学习权重向量决定的。参数化价值函数的一个例子是固定或专家设计特征的线性函数。

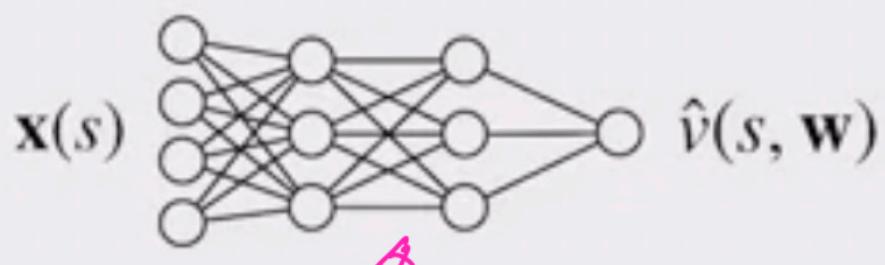
*is linear function of fixed or expert designed features*



## Parameterized Value Function Approximation



$$\hat{v}(s, \mathbf{w}) \doteq \langle \mathbf{w}, \mathbf{x}(s) \rangle$$



另一个例子是神经网络 neural network。我们谈到了函数近似的两个关键属性，即泛化 generalization 和 分辨 discrimination。更新一个状态的值可以改善其他状态的值估计。这种泛化可以使学习更快。我们还希望我们的价值函数在状态非常不同的时候分配不同的值。我们把这称为分辨。接下来，我们讨论了如何使用监督学习的思想来学习近似的价值函数。我们不能保证每个状态的值都是完美的近似值。所以我们需要定义一个目标，一个衡量我们的近似值和真实值之间的距离。我们使用 平均平方值 Mean Squared value 或 目标 objective。每个状态的值与我们的近似值之间的平方误差之和，由访问频率 Mu 加权。我们讨论了如何使用 随机梯度下降 stochastic gradient descent 来优化其目标。梯度蒙特卡洛算法使用采样的回报更新权重来执行随机梯度下降。这意味着，它可以从代理人与世界互动产生的经验中学习近似值。我们还引入了半梯度 TD 作为随机梯度下降的近似方法。半梯度 TD 利用引导的优势，在实践中可能比蒙特卡洛收敛得更快。半梯度这个名字的一部分提醒我们，它不是一个梯度下降算法。最后，我们讨论了线性半梯度 TD 算法或带有线性函数近似的 TD。线性 TD 可以证明收敛到一个很好理解的点。

你现在有了一个很好的基础来理解带有参数化函数的强化学习。接下来，我们将讨论构建状态的不同方法，以及状态动作特征。//