

## 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.,  $\mathbf{w} = \mathbf{0}$ )

Loop for each episode:

    Initialize  $S$

    Loop for each step of episode:

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

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

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

$S \leftarrow S'$

    until  $S$  is terminal

*TD error*      *approximation of next state*      *gradient of current state*  
*approximation of the current state.*

该功能用于构建价值估计是强化学习代理的最重要部分之一。今天，我们将讨论一种简单而有效的方法。在本视频结束时，你将能够描述 ~~课程~~ 编码 ~~course~~ coding 并说明它与 状态聚合 state aggregation 的关系。

course

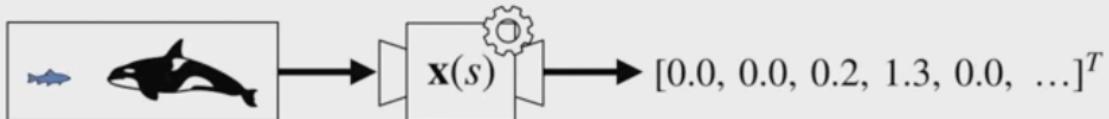
粗糙

让我们来谈谈为 线性值函数 linear value functions 创建特征的新方法。回想一下，近似的价值函数是由权重向量  $W$  参数化的。为了计算一个状态的价值，我们首先计算特征并构建特征向量  $X(S)$ 。然后，状态的值由权重向量和特征向量的 DOT 乘积来近似。回顾一下，表格表示可以表示为一个二进制特征向量。每个状态都与一个不同的特征相关。如果代理处于一个状态，那么对应于该状态的特征为1，所有其他的特征为0。表格的情况只是线性函数近似的一个特例，其中特征向量是一个指标或状态的一个热编码。当然，当状态空间的大小变得比代理人的可用内存大得多时，这是不可行的。假设我们想表示这条在池塘里游泳的鱼的位置，这是一个二维的状态。这条鱼可以是无限多的位置之一。用一个有限的查找表来表示所有这些位置是不可能的。

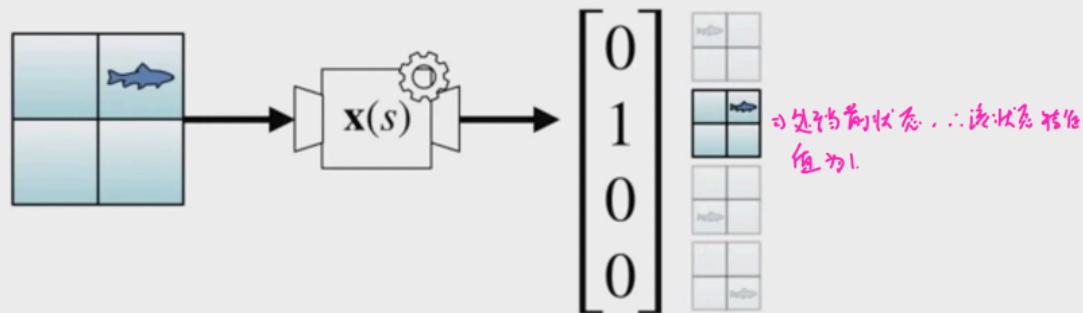
回顾一下，我们可以使用状态聚合来将附近的状态与同一特征联系起来。这就好比把每个方格内的所有状态都当作同一个状态。在这个例子中，所有的组都有大致相同的形状，但是没有必要使用相同的形状。一般来说，我们可以使用任何我们想要的形状来聚合状态，只要这些形状没有任何空隙或重叠。状态聚合通常不允许形状重叠。但这种限制是没有必要的。事实上，通过允许重叠，我们得到了一类更灵活的特征表示，称为课程编码。让我们来看看鱼儿在池塘中的当前位置的特征向量的例子。请记住，特征向量 feature vector 中的每个 索引 index 都对应于其中一个形状。如果鱼在圆圈内，对应于圆圈的特征就会被激活或设置为1，否则该特征设置为0。当鱼移动到一个新的位置时，通常会被一组不同的圆圈覆盖。附近的状态会有类似的 特征激活 feature activations，但它们也可能有不同的 组件活跃 <sup>components</sup> active features，包括不同数量的活跃特征。在这个例子中，总是至少有一个活跃的特征，最多有三个活跃的特征。到目前为止，我们所讨论的所有想法都不限于二维状态空间。课程编码也可以应用于更高维度的输入。

## Recall linear value function approximation

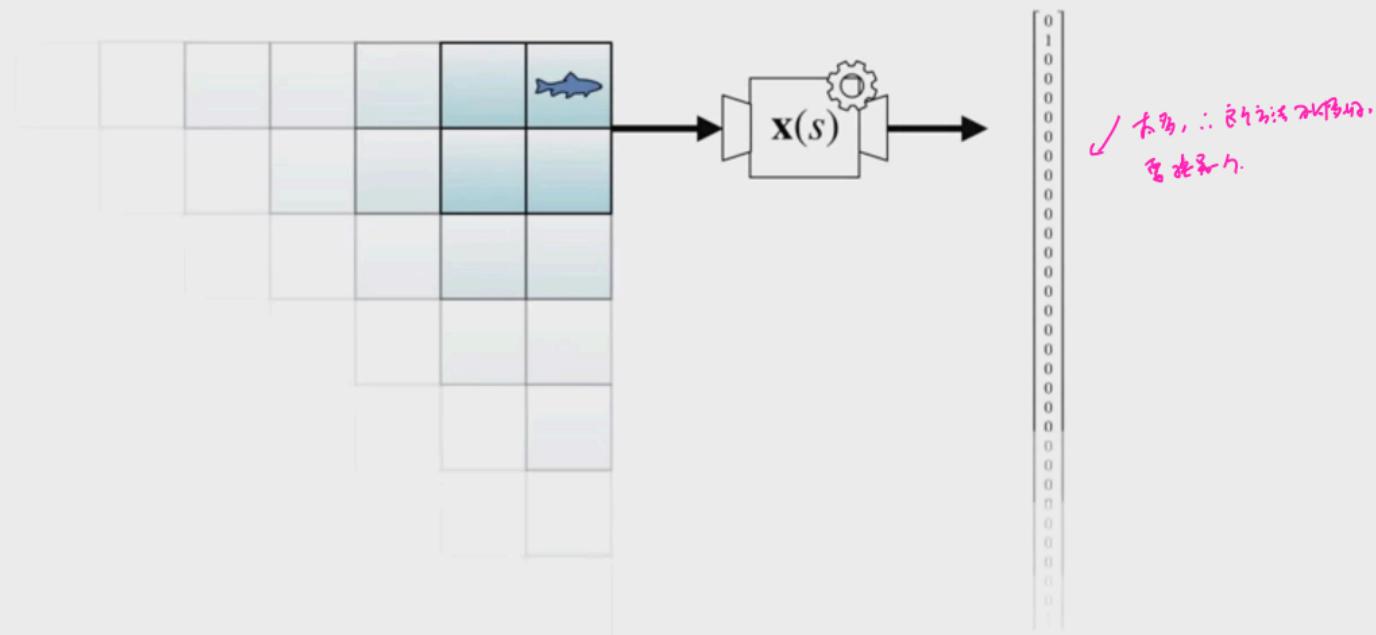
$$v_{\pi}(s) \approx \hat{v}(s, \mathbf{w})$$



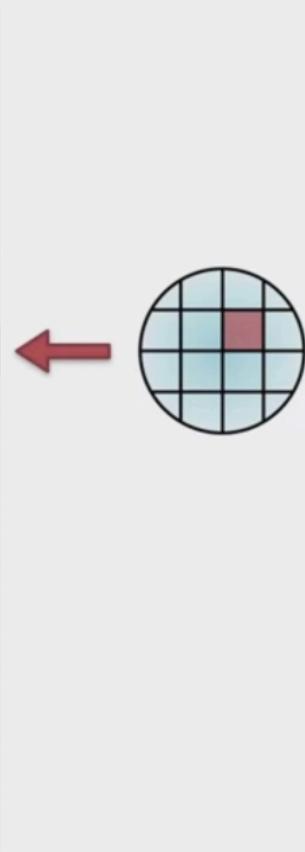
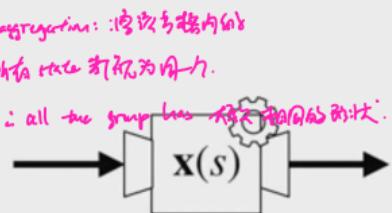
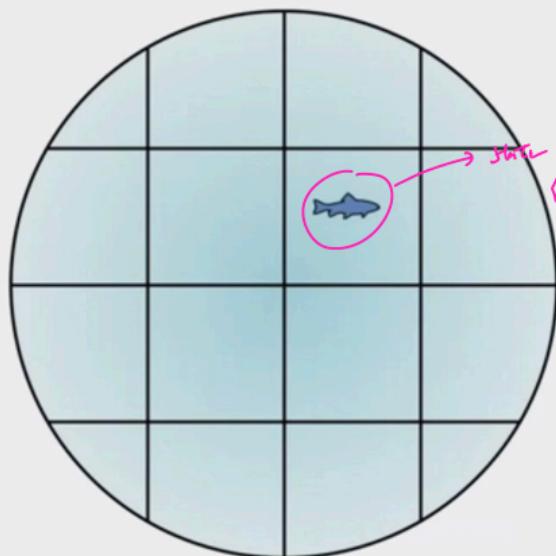
## Recall linear value function approximation



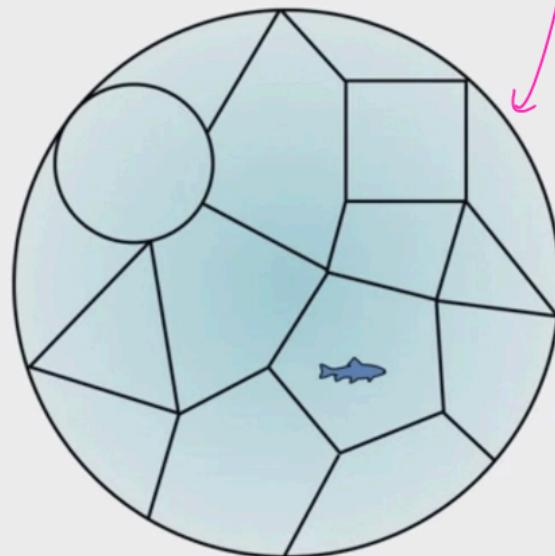
## Recall linear value function approximation



## Recall state aggregation

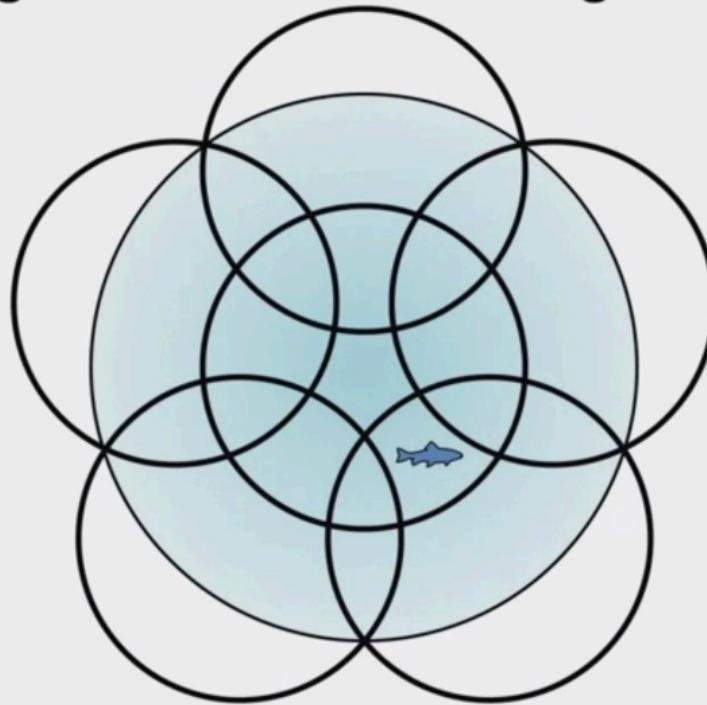


## State aggregation → coarse coding



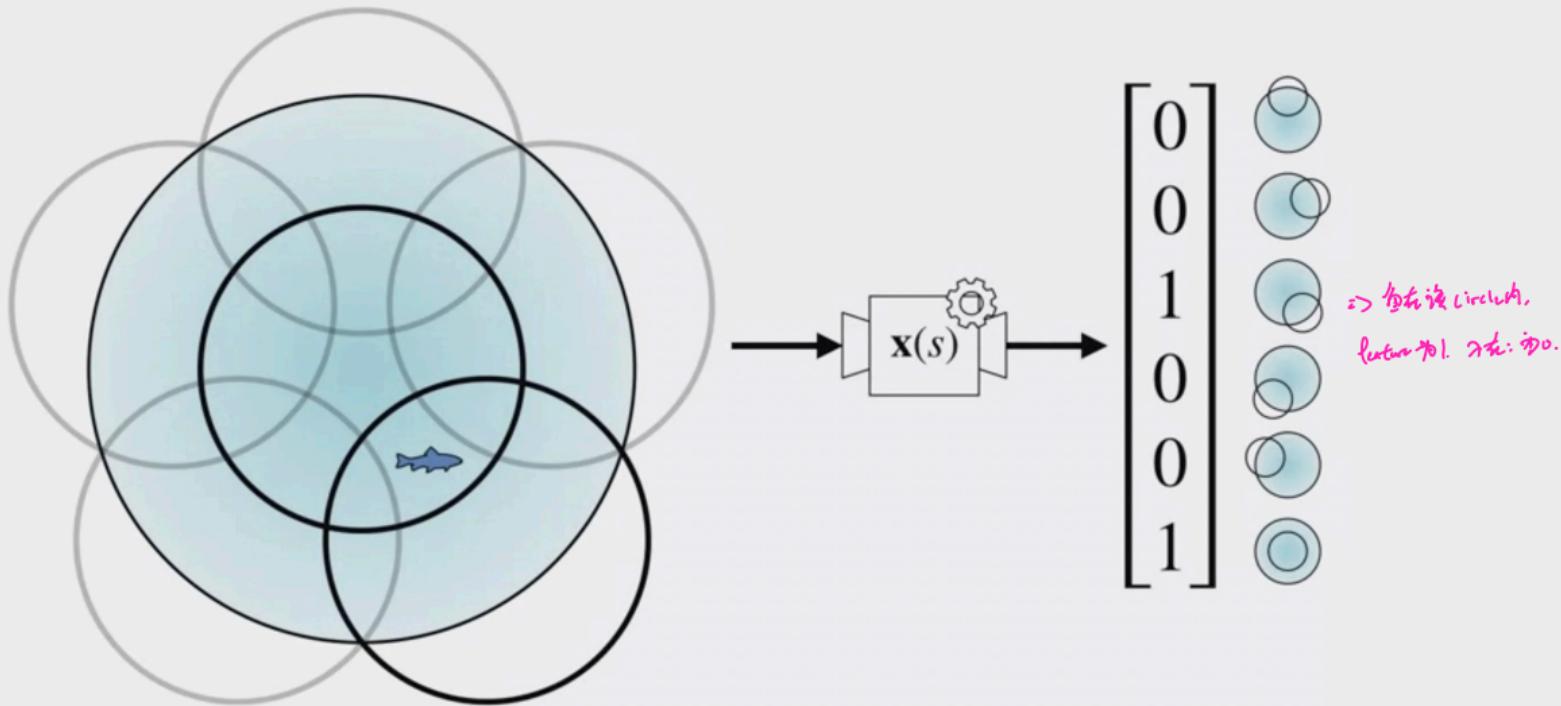
但是是我們沿此每一塊用相同的向量，得到更靈活的  
feature 矢量。

## State aggregation → coarse coding



⇒ 7個圓的「粗」over-lap, 粗到什麼程度  
lecture 22-.

## Coarse coding



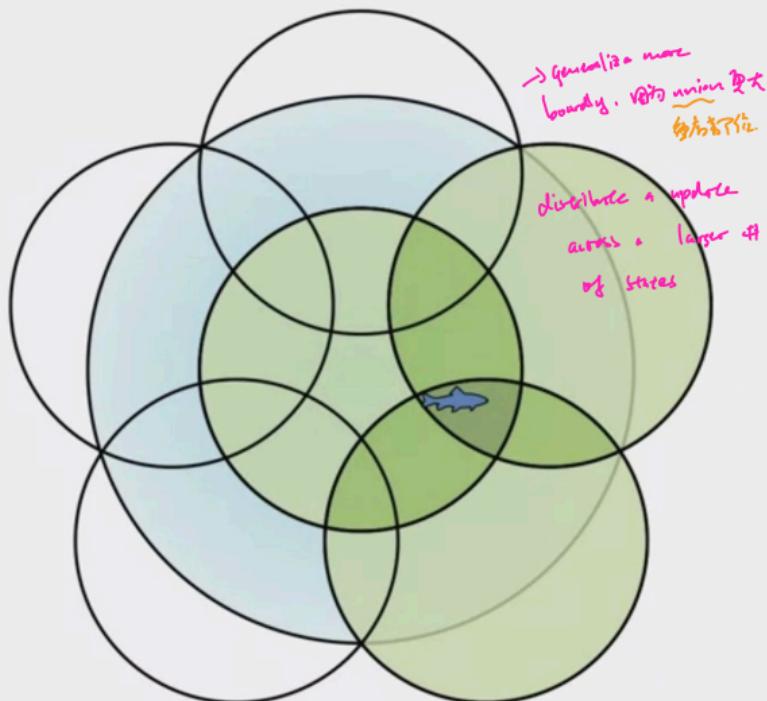
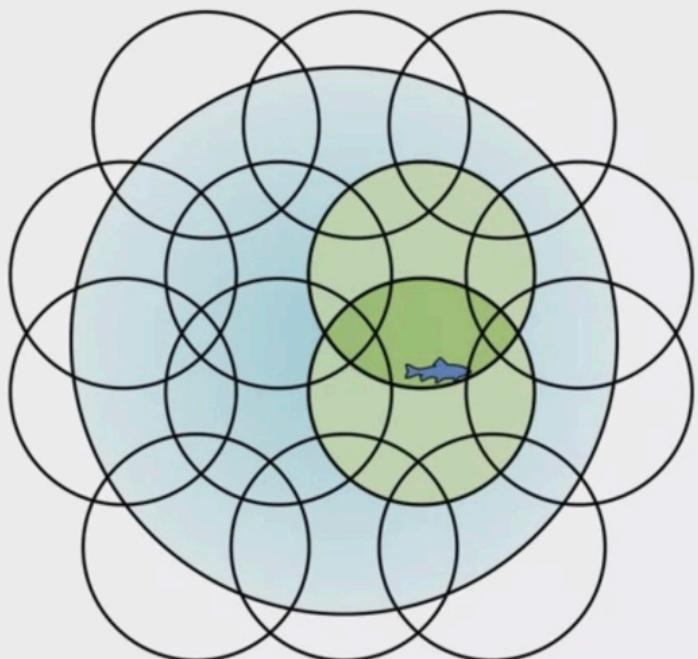
之前，我们谈到了几种不同的特征表示方法。表格式 Tabular，状态聚合 State Aggregation，和 粗略编码 Coarse Coding。但我们并没有告诉你为什么粗略编码可能是有用的。今天，我们将讨论改变粗略编码的属性如何影响概括 generalization 和 辨别 discrimination。

我们已经谈到了粗编码如何将状态分组为 任意形状 arbitrary shapes 和 大小 sizes 的特征。它们可以是圆形、椭圆、正方形或不同形状的组合。让我们来看看改变特征的形状和大小是如何影响泛化和辨别的，从而影响学习的速度和我们可以表示的价值函数。请注意，对一个状态下的权重进行更新，会改变活动特征的接受域内所有状态的价值估计。如果活动特征的感受野的联合是大的，那么特征表示就会有更多的概括。反之，如果联合体很小，则泛化程度就很低。这里，右边的大圆圈泛化得更广泛，将更新分布在更多的状态中。

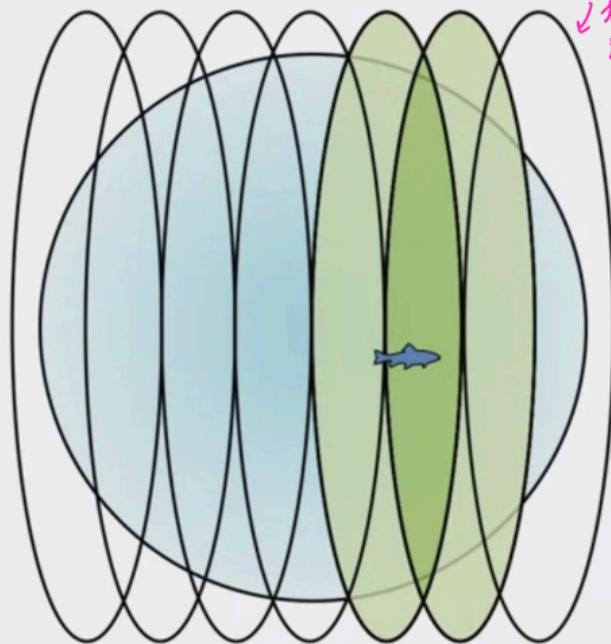
然而，泛化不只是一个标量。在粗糙的编码中使用不同的形状也可以改变泛化的方向。让我们把前面的例子与圆圈相比较，看看粗编码如何用垂直拉长的椭圆来概括。由椭圆构成的感受野比它们的宽度要长。对于这些椭圆，粗略编码主要在垂直维度上进行概括。所以我们已经谈到了 接受区 receptive fields 的形状和大小是如何影响泛化的，从而影响学习的速度。但是，我们估计的最终准确性如何呢？这就是辨别力的作用。

回顾一下，区分两个不同状态的数值的能力被称为辨别力。在粗略的编码中，圆圈之间的重叠决定了分辨力的水平。我们不可能做到完美的区分，因为我们不可能在不影响其他状态的值的情况下更新一个状态的值。彩色的形状描述了这个特定的粗略编码的辨别能力。我们只强调了几个区域，以保持可视化的简单。同一个彩色形状内的每个状态都有完全相同的特征向量。因此，它们都必须有相同的近似值。这些区域越小，我们就能更好地进行分辨。有了很多圆圈，区域就会变小，我们可以更精细地分辨不同状态的值，或者我们可以让圆圈变小。所以，特征的大小、数量和形状都会影响表征的判别能力。

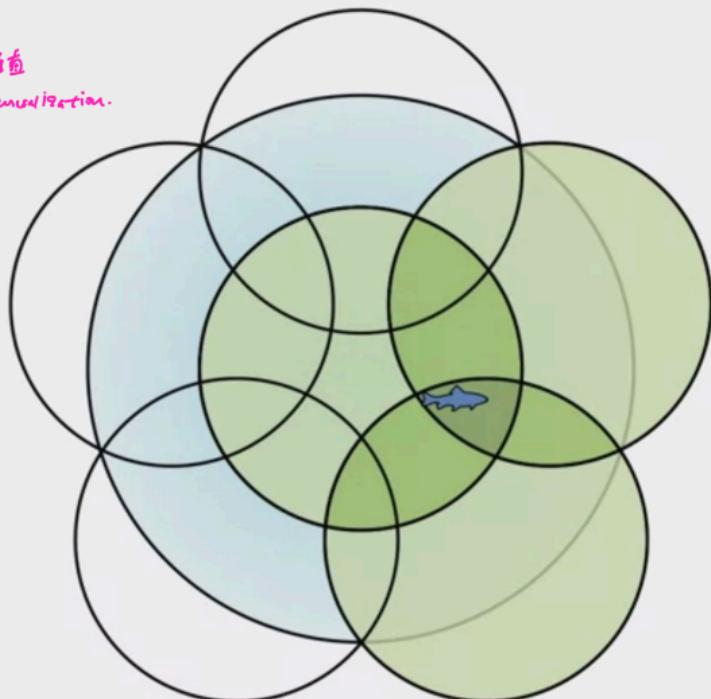
## Broadness of generalization



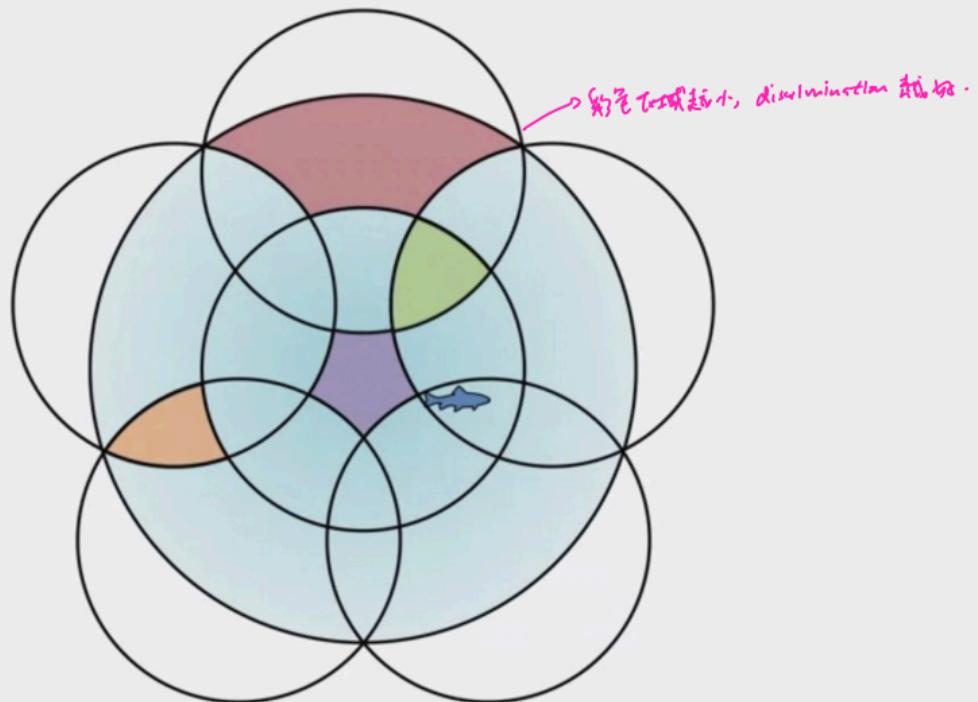
## Direction of generalization



↑ 仰向圖示的泛化  
方向上圖示 generalization.



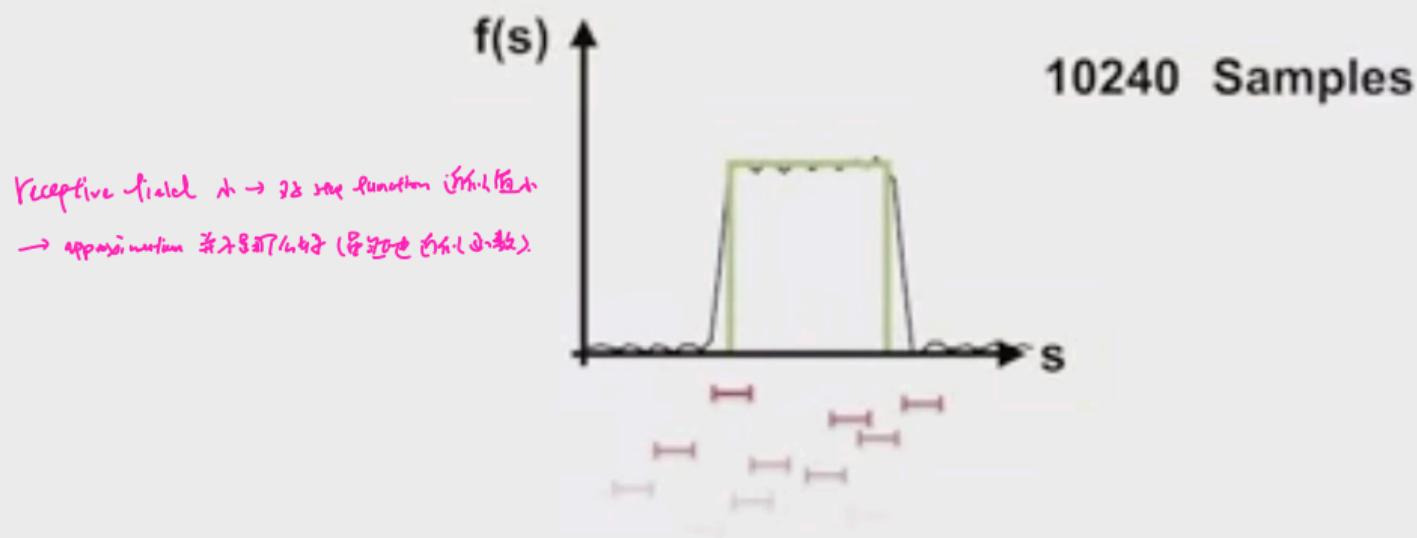
# State discrimination



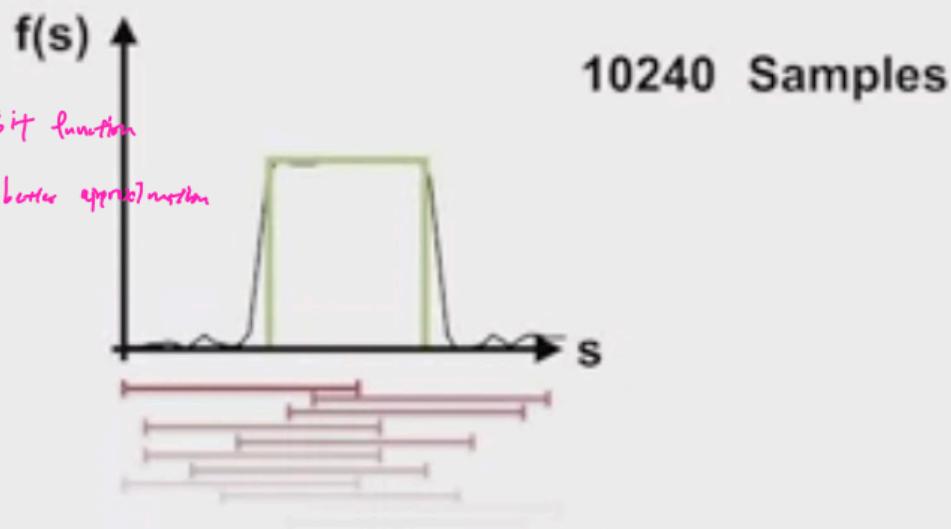
让我们看一个简单的例子，有一个一维的输入空间。考虑学习阶梯函数的近似值。让我们假设我们可以对真实的函数值进行采样，以便更新我们的估计值。这个例子应该能帮助你更好地理解表示法的选择如何影响学习的速度和最终近似的质量。对于我们的一维函数，每个特征的感受野将被表示为重叠的区间。让我们从这个相对较短的区间开始。我们将铺设大约50个这样的区间，使它们在我们的函数域上随机地重叠起来。让我们看看，当我们对函数的真值进行随机抽样时，我们对函数的估计是如何变化的。我们从最初的零估计开始，描绘成一条平线。每个特征的感受野 receptive field 是相当小的。因此，即使经过多次采样，我们对阶梯函数 step function 的近似值也不是那么大。经过更多的训练，我们的近似值最终获得了与真实函数的近似值。但它并不完美。通过检查函数顶部的近似值可以很容易看出这一点。它没有那么平坦或光滑。让我们用更长的时间间隔再试一下。每个特征的接受域都相当大。这意味着我们可以用相对较少的样本来估计函数的大致形状。随着我们对函数进行更多的采样，我们的估计形成了对真实函数越来越好的近似。长区间的广泛概括使学习更快。我们需要更少的样本来获得一个好的近似值。大量较长的区间也导致了更好的辨别力，对真实函数更好的最终近似。在这个例子中，较长的区间最终实现了更好的概括和辨别。但情况可能并不总是这样。每个任务可能需要不同的特征属性，没有一个通用的解决方案。

在这段视频中，我们谈到了特征的大小、数量和形状是如何影响概括的，以及由此产生的形状交叉点是如何影响辨别能力的。粗略的编码是一种非常普遍的表征类型。了解它在学习过程中是如何泛化和辨别的，将有助于我们理解其他表征，包括神经网络。//

# 1D Example



# 1D Example

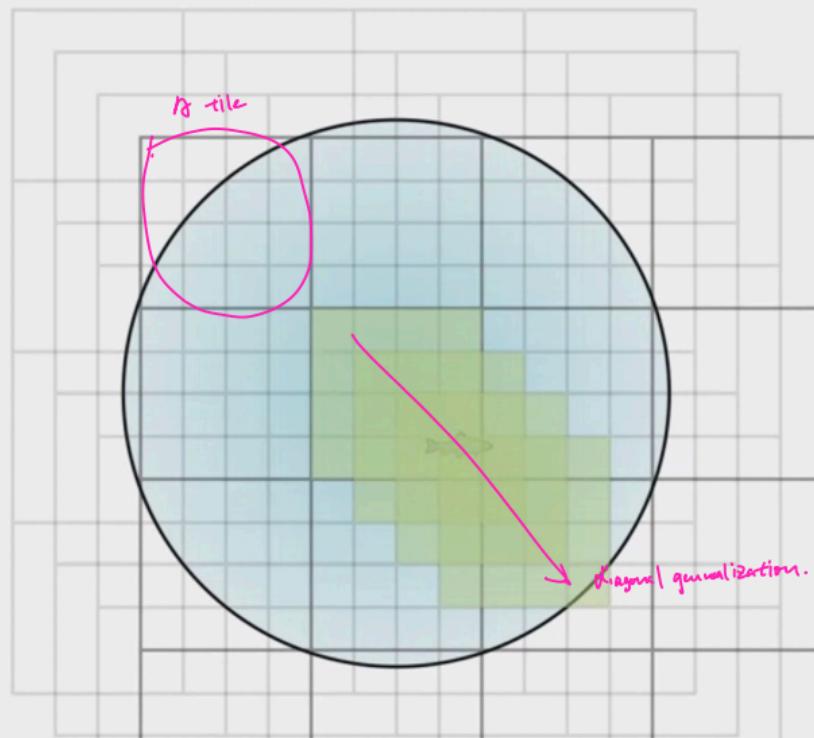


我们之前谈到了课程编码。今天，让我们来看看一种计算效率高的方法来执行课程编码，即 瓦片编码 Tile Coding。在这段视频结束时，你将能够；解释瓦片编码是如何实现泛化和区分的，并理解瓦片编码的好处和限制。

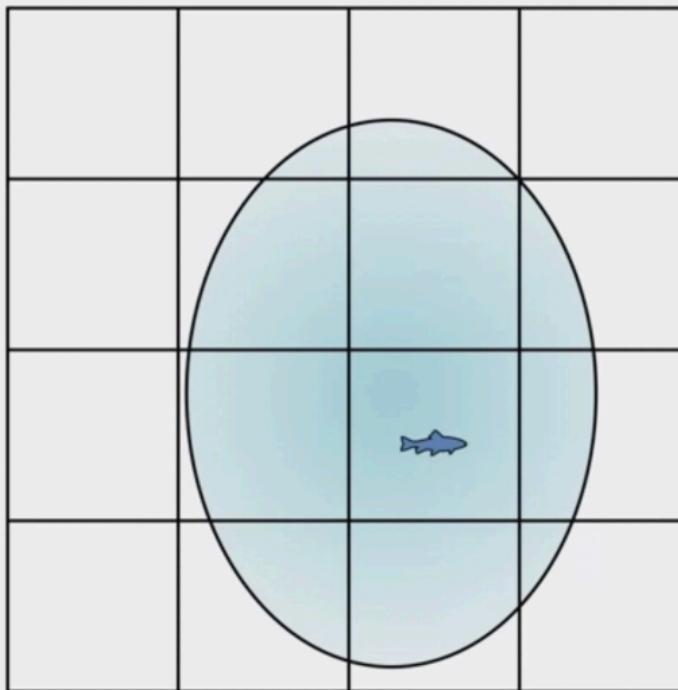
瓦片编码是课程编码的一种特殊类型。瓦片编码使用重叠的网格对状态空间进行了详尽的划分。这也许是通过一个例子最好的理解。让我们再来看看我们的鱼朋友，他生活在一个二维的连续状态空间，池塘。与使用任意形状的课程编码不同，瓦片编码使用方块。什么是最方便的方式来布置空间上的一堆方块？是一个网格 grid。让我们把这个网格称为瓦片。到目前为止，使用这个只是状态聚合。较大的瓦片会导致泛化的增加。虽然理想的瓦片大小取决于具体问题，但一般来说，使用较大的瓦片是个好主意。为了提高我们的瓦片编码的判别能力，我们可以把几个瓦片放在彼此的上面。每个瓦片都会有少量的偏移。每层瓦片的偏移都会产生许多小的交叉点。这导致了更好的辨别。

在实践中，使用大量的瓦片是很有用的。对于一个瓦片，泛化只发生在正方形内，但对于多个瓦片，这个状态的更新会泛化到这些其他状态。这种情况下的泛化是对角线式的。如果我们使用了随机偏移 random offsets，那么泛化将是更多的球形和同质的。这在教科书中会有更多讨论。让我们来谈谈如何进一步控制泛化的特性。我们可以通过创建一个长方形而不是正方形的网格来做到这一点。一个有效的方法是缩放状态空间的每个维度。这里的环境似乎已经被压扁了。实际上，在这个被压扁的环境上铺设正方形，就像在未被压扁的环境上铺设矩形。通过使用矩形，我们可以在状态空间的每个维度上控制泛化的广度。瓦片编码可以代表广泛的功能，但它的效用并不限于此。瓦片编码在计算上也是高效的。由于网格是统一的，所以很容易计算出当前状态在哪个单元。由于其计算效率高，瓦片编码可用于在低维环境中快速运行初步实验。然而，随着维数的增长，所需的瓦片数量也呈指数级增长。因此，可能有必要对输入维度单独进行贴码 tile。输入维度 input dimensions 是否可以独立处理，取决于具体问题。

## Shape of generalization

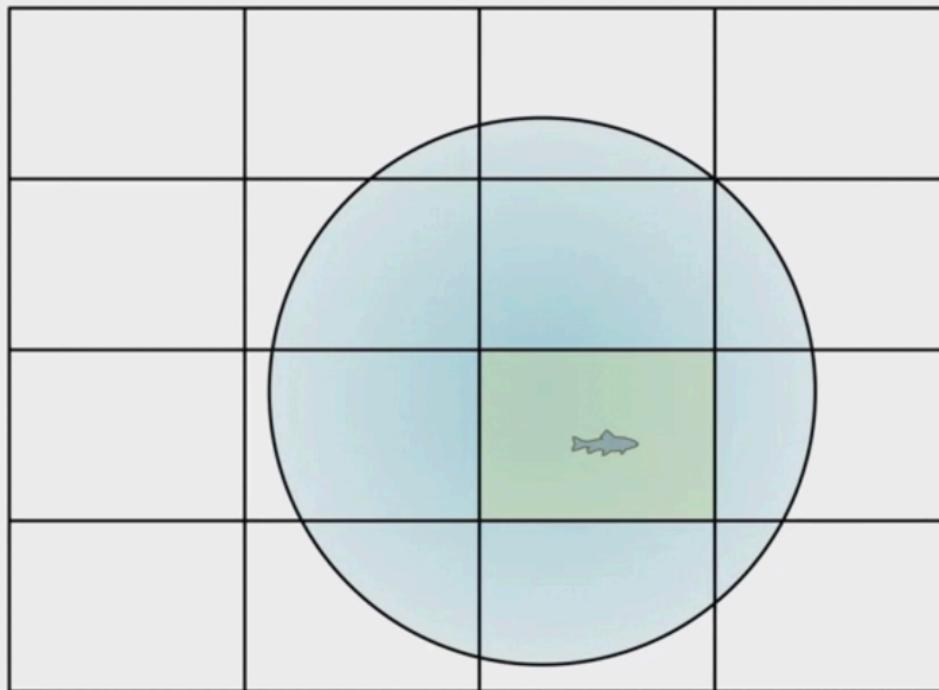


## Direction of generalization



From top to bottom.

## Direction of generalization



→ *From the specific environment*

到目前为止，你已经了解了瓦片编码，但我们还没有把它与强化学习联系起来。今天，我们将解释如何将瓦片编码与线性TD一起使用。在本视频结束时，你将能够解释如何将瓦片编码与时差学习结合起来使用，并确定瓦片编码表示的重要属性。

活动瓦片 active tiles 的数量总是明显少于 总瓦片 total tiles 的数量。我们可以利用这一点来有效地计算 价值函数 value function。回顾一下，在线性函数近似 linear function approximation 的情况下，价值函数是一个 权重向量 weight vector 和一个 特征向量 feature vector 之间的 点积 dot product。让我们看看这个点积在稀疏的二进制特征向量中是什么样子的。

如果我们将两个向量进行元素相乘，许多元素相乘的产物将为零。这意味着我们只需要考虑特征向量中非零元素的权重，因为特征是二进制的，非零特征的权重被乘以1。用通常的方法计算点积会很昂贵。相反，我们可以直接将与活动特征相对应的权重相加。请注意，这需要一定的时间，因为活动特征的数量在每个状态下是相同的。通过瓦片编码产生的特征向量可能会在价值函数中查询到便宜的计算结果。让我们来看看一个带有瓦片编码特征的价值函数的例子。

假设我们回到旧池塘，有两个各由四块瓷砖组成的倾斜面。我们对瓷砖进行了颜色编码，每个瓷砖方块的索引如图所示。如果鱼的位置和一个重量向量如图所示，那么鱼的位置值是多少？我们可以看到，鱼与属于紫色瓦片的一号方格和属于橙色瓦片的四号方格相交。这就得出了这个状态的下降特征向量。现在我们只需用点积来计算该状态的值。特征向量为二进制。所以我们可以直接把具有非零特征的位置的权重加起来。这就给了我们一个2的值。

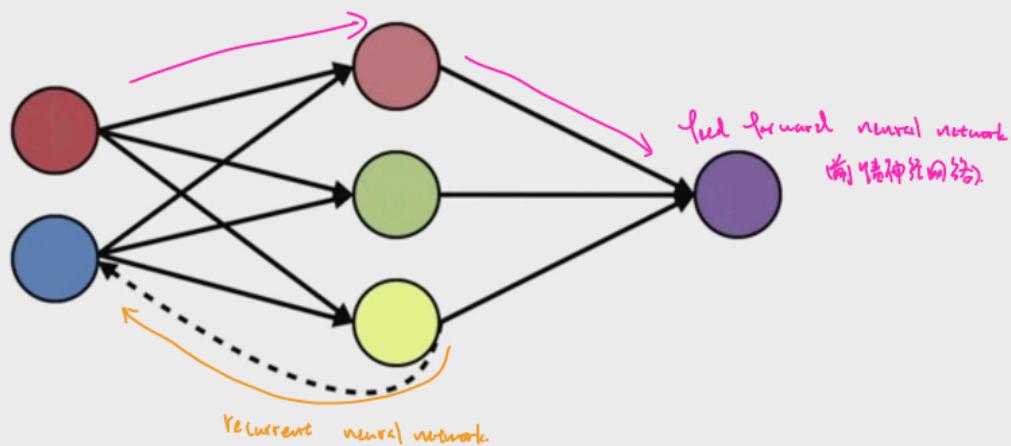
现在，让我们用瓦片编码进行实验，并将其与状态聚合进行比较。回顾一下，瓦片编码实际上是状态聚合或倾斜 state aggregation or tilings 的多个实例相互叠加。我们的环境是1,000个状态的随机行走。状态从1-1,000编号，所有事件都从状态500开始。在每个时间步骤中，代理人随机地过渡到两侧200个相邻的状态之一。逾越状态1或1000将导致过渡到终端状态。例如，看一下状态二左边的可能过渡，一个会去到状态一，其余199个会去到终端状态。让我们来谈谈如何在这个问题上使用函数近似。我们先从状态聚合开始。让我们使用有五个组的状态聚合，每个组对应200个状态。现在，让我们为这个问题设计一个有50个瓦片的瓦片代码。我们把1000个状态当作一个区间，每个瓦片对应200个状态。由于每个瓦片都有200个状态，你可能认为我们只需要5个瓦片就能覆盖1000个状态的全部空间。然而，每块瓷砖都有轻微的偏移或偏离。需要一个额外的瓦片来覆盖未覆盖的空间。正因为如此，我们的每一个平铺图都包含六块瓷砖，而不是五块。让我们用梯度蒙特卡洛来比较这两个函数近似器。步长参数 Alpha 是由活动特征的数量来缩放的；瓦片编码为50，状态聚集为1。让我们看一下结果。我们绘制了每一集后30次运行的平均值的数值误差。由于积极的泛化，两个代理都学得很快。然而，瓦片编码表示法能够更好地地区分不同的状态，并实现较低的值误差。有趣的是，即使瓦片编码器有更多的参数，它也能像核心状态聚合一样快速学习。但它也取得了更好的辨别能力，因为多个重叠的瓦片所产生的小的交叉点。

在这段视频中，我们谈到了瓦片编码如何用于近似价值函数和政策估值，我们在简单的例子中比较了瓦片编码和状态聚合。下一次，我们将超越固定表征 fixed representations，开始用神经网络学习表征。//

今天，我们将讨论 神经网络 Neural Networks，这是一类灵活而强大的 非线性函数近似器 nonlinear function approximators。用神经网络进行的强化学习已经被用来创造世界围棋冠军，创造出能够实现超人表现的 雅达利代理 Atari agents，甚至是自主汽车的帮助。在本视频结束时，你将能够理解前馈神经网络来寻找激活函数，并理解神经网络是如何成为一个参数化的函数 parameterized function。

让我们走过一个简单的神经网络。这些圆圈中的每一个都代表网络中的 节点 nodes。每条线代表节点之间的连接。节点被组织成层。当新的数据进来时，它从输入层开始，通过连接被发送到下一层的节点。该层对数据进行一些计算，然后将结果发送到下一层。这个过程一直持续到最后一层产生最终的输出。我们称其为前馈神经网络，因为数据总是通过各层向前移动。没有任何连接会返回到网络中。如果他们这样做，那么节点的输出就会影响其自身的输入。这样的神经网络被称为递归神经网络。当数据通过一个连接时，一个权重被应用。然后，节点将其每个加权输入相加，并对这个总和应用一些 激活函数 activation function。

# Simple neural network



Input  
Layer

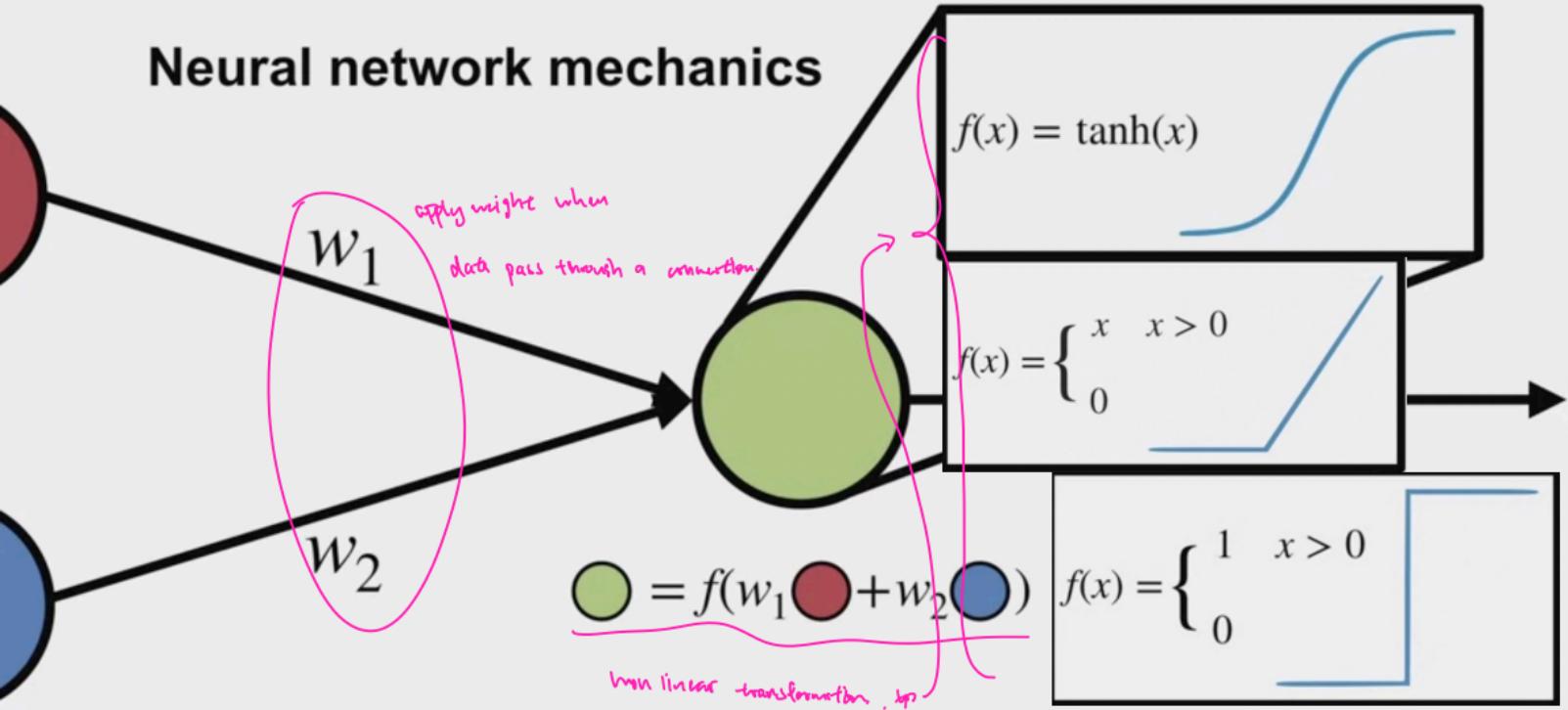
Hidden  
Layer

Output  
Layer

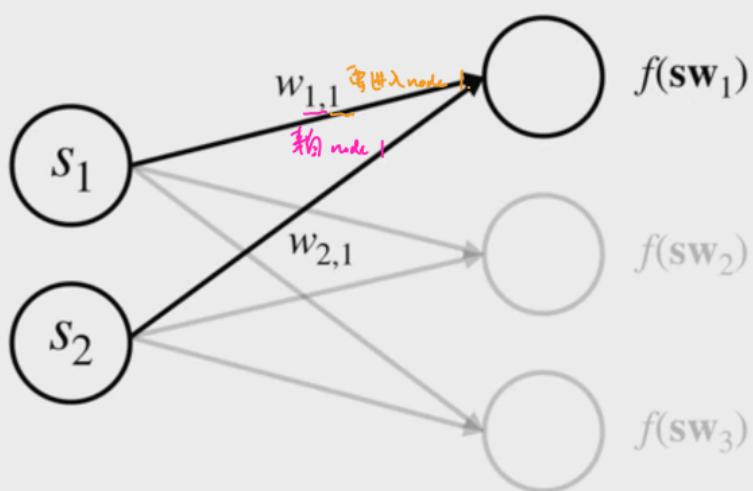
激活函数通常是一个非线性变换 nonlinear transformation。常见的激活函数包括sigmoidal函数，如 $\tanh$ 或logistic函数，整流线性单元或ReLU，甚至是阈值单元 thresholding units。让我们来看看如何用数学方法编写神经网络的前馈传递 forward pass。在每个节点，我们有两个向量，输入 inputs 和每个输入的权重 the weights for each input。第一个下标指的是该连接来自的节点。第二个下标指的是该连接进入的节点。我们不将权重与输入相乘，而是将结果通过激活函数传递。注意这里 $S$ 是一个行向量，所以我们不需要用转置来写点积。每一层都由许多这样的节点组成。一个神经网络只是一个参数化的函数，尽管把它看作是一个连接节点的集合，这可能有点难看。为了更明确地看到这一点，让我们用矩阵向量乘法重写网络中的操作。我们可以把一个层的输入看作是一个行向量，而该层的权重则是一个矩阵。换句话说，就是这个二维数组。我们可以用矩阵乘法来计算整个层的输出。然后我们将激活函数应用于输出向量的每个元素。如果这个输出不是最后一层，这个输出向量就会成为下一层的输入，然后我们重复这个过程。

这段视频就到此为止。我们解释了神经网络是如何由一个处理和传递信息的节点网络组成的，并讨论了前馈神经网络是如何一个参数化的函数。||

## Neural network mechanics



# Neural network implementation



$$\mathbf{s} = [s_1, s_2]$$

$$\mathbf{W} = [\mathbf{w}_1 \quad \mathbf{w}_2 \quad \mathbf{w}_3]$$

$$= \begin{bmatrix} w_{1,1} & w_{2,1} & w_{3,1} \\ w_{1,2} & w_{2,2} & w_{3,2} \end{bmatrix}$$

$$\text{outputs} = f(\mathbf{sW})$$

the inputs  
 $\mathbf{s} = [s_1, s_2]$

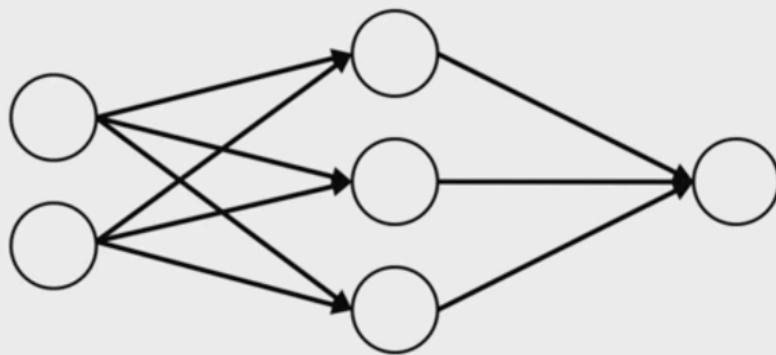
the weights for each inputs.  
 $\mathbf{w}_1 = [w_{1,1}, w_{2,1}]^T$

瓦片编码 Tile coding 是为预测创造一套固定特征的一种方法。神经网络提供了一种学习有用特征集的策略。今天我们将讨论神经网络如何创建这些特征。在本视频结束时，你将了解神经网络如何进行特征构建，你将了解神经网络是一个非线性的状态函数。

让我们重新审视一下前馈神经网络。当我们第一次构建神经网络时，我们需要指定初始权重。我们初始化权重的方式很重要，但后面会有更多的内容。现在，让我们假设这些权重是从某个随机分布中提取的。

让我们来看看，当我们把一个给定的输入向量通过网络时，会发生什么。让我们先看一下网络的一个节点。每一个输入都要乘以它们相应的权重。然后，这些输入的加权和被传递给一个非线性激活函数 non-linear activation function，从而形成一个输入的非线性函数 non-linear function of the inputs。这个过程对该层中的每个节点都要反复进行。每个节点都有一组不同的权重，所以会产生不同的输出，我们称之为特征 feature。所有这些新的特征，统统被认为是新的表征。这个过程实际上与瓦片编码没有什么不同，我们把输入传给瓦片编码机，然后得到一个新的表示。因此，在这两种情况下，我们都构建了输入的非线性映射来产生特征。在这两种情况下，我们对表示进行线性组合以产生输出，即当前状态的近似值。让我们通过与瓦片编码的对比来更深入地了解神经网络的表示。

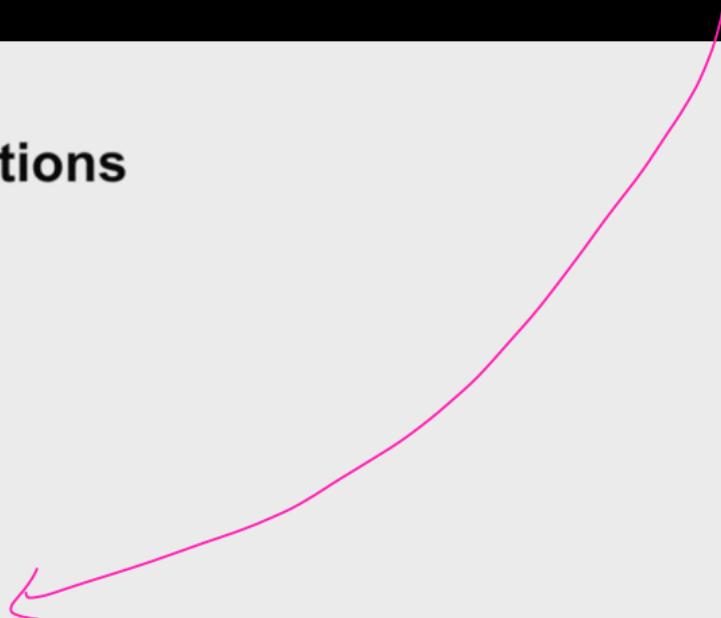
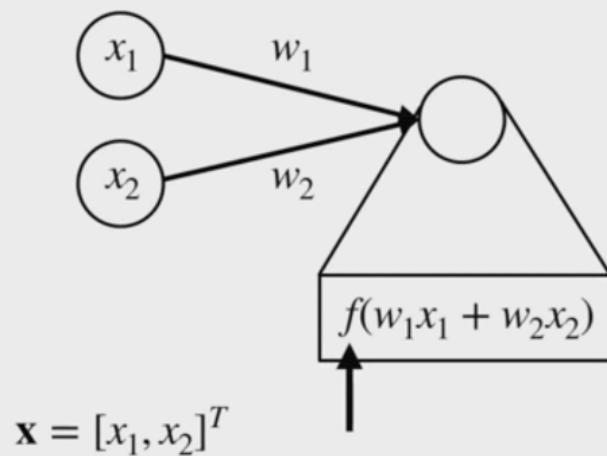
## Non-linear representations



$$\underline{\mathbf{w}_{init}} \sim \mathcal{N}(\mu, \sigma)$$

initial weight

## Non-linear representations



回顾一下，当我们创建瓦片编码时，我们必须设置几个参数，瓦片的大小和形状以及瓦片的数量。这些参数在学习之前是固定的。在神经网络中，我们有类似的参数，对应于层数、每层的节点数和激活函数。这些通常都是在学习之前固定的。在这个意义上，两者都使用先验知识来帮助构建特征。然而，除此之外，神经网络也有可调整的参数，在学习过程中改变特征。神经网络可以使用数据来改进特征，而瓦片编码器则不能从数据中吸收新的信息。神经网络中更多的瓦片编码产生的特征在输入空间是非线性的。我们可以通过可视化预训练的神经网络的隐藏层来更好地理解这一点。我们在一个连续的二维空间中训练了一个代理，该空间包含两堵墙之间的一个狭窄的走廊。在这里，我们绘制了网络学习的单个特征的感受野。轴对应于x、y位置，这是MDP的状态空间。图中的每一点都对应于该xy状态的特征值，越深意味着激活越大。如果是白色，说明该特征对该状态下是活跃的，也就是说，这个特征的幅度高于一个小的阈值。因此，在更高的层次上，这个图显示了这个特征是如何泛化的。对其权重的更新，改变了所有这些状态的值。看一下几个特征的感受野，我们可以看到，不同的特征以不同的方式泛化，并形成复杂的非线性形状。激活并不像瓦片编码中那样有硬性的边界。这些学习特征可以有平稳变化的边界。这个特征根据状态的不同，有不同程度的泛化。这也有点意思，这个网络中的特征激活，向我们展示了两面墙的位置。

这段视频就到此为止。你学会了如何将神经网络视为构建特征的一种方式，以及神经网络是sState的非线性函数。//

神经网络架构的选择会对性能产生很大影响。这包括 节点的数量 number of nodes、激活函数 the activation functions, 以及 节点的排列和连接方式 how the nodes are arranged and connected。今天, 我们将提供一些关于深度在网络中可以发挥的作用的直觉。在本视频结束时, 你将了解深度神经网络是如何由许多层组成的, 并理解深度可以通过组成和抽象促进学习特征。

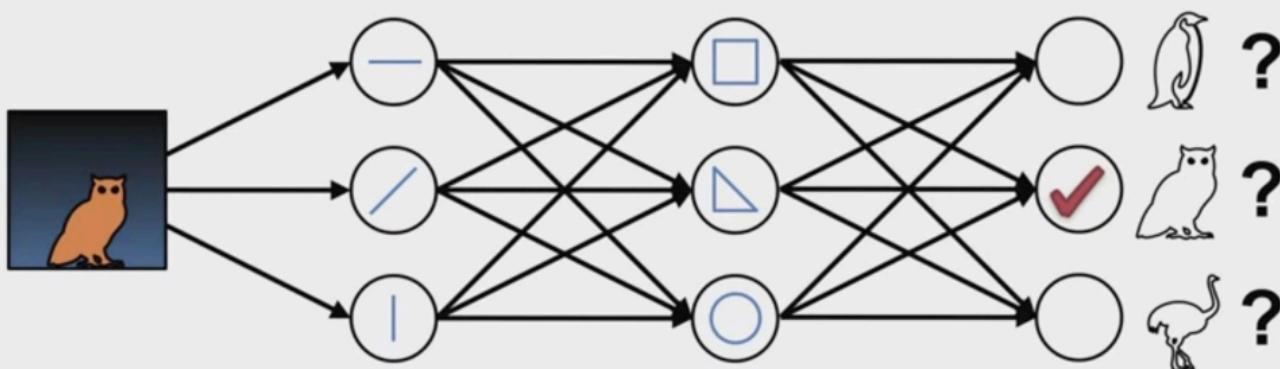
我们可以把神经网络看作是一个模块化系统, 每一层都是一个 模块 module。我们可以添加和删除层, 我们可以改变我们使用的层的类型。网络的深度由网络中隐藏层的数量来定义。从理论上讲, 一个神经网络不需要很深。一个只有一个 隐藏层 hidden layer 的神经网络可以近似于任何连续的函数, 只要足够宽。我们把这称为 普遍近似特性 universal approximation property。然而, 实际经验和理论表明, 深度神经网络可能使复杂函数的近似更容易。其中一个原因是深度允许特征的组合。组合可以通过组合模块化组件产生更多的专业化特征。

考虑一个直观的例子。输入是一张图像的原始像素。前面的层可能会学习捕捉低层次的特征, 如不同角度的线条。下一层可能会学习将这些线条组合成各种形状。根据这些, 网络可能能够检测图像中的物体或动物。这通常比试图直接从原始像素中推断出这些要好。通过添加更多的层或更多的单元到每一层, 我们可以表示更复杂的功能。深度也可以帮助获得抽象。深度神经网络由许多低级别的抽象层组成, 每一个连续的层都有助于获得越来越抽象的表示。考虑一下前面的例子。该网络最终可能用一个比特来表示猫头鹰的概念。零表示图像不包含猫头鹰, 一表示图像包含猫头鹰。在这个层面上, 图像上所有的额外细节都已经被删除。我们不再知道猫头鹰的颜色, 甚至不知道背景里有什么。事实上, 我们可以明确地设计网络以去除输入的不必要的细节。例如, 可以设计一个带有 瓶颈层 bottleneck layer 的网络。这个想法很简单。每一个连续的层包含的节点都比之前的层少。表示的是节点最少的一层, 包含预测所需的关键细节。总的来说, 网络中的深度可以显著提高我们代理学习特征的能力。

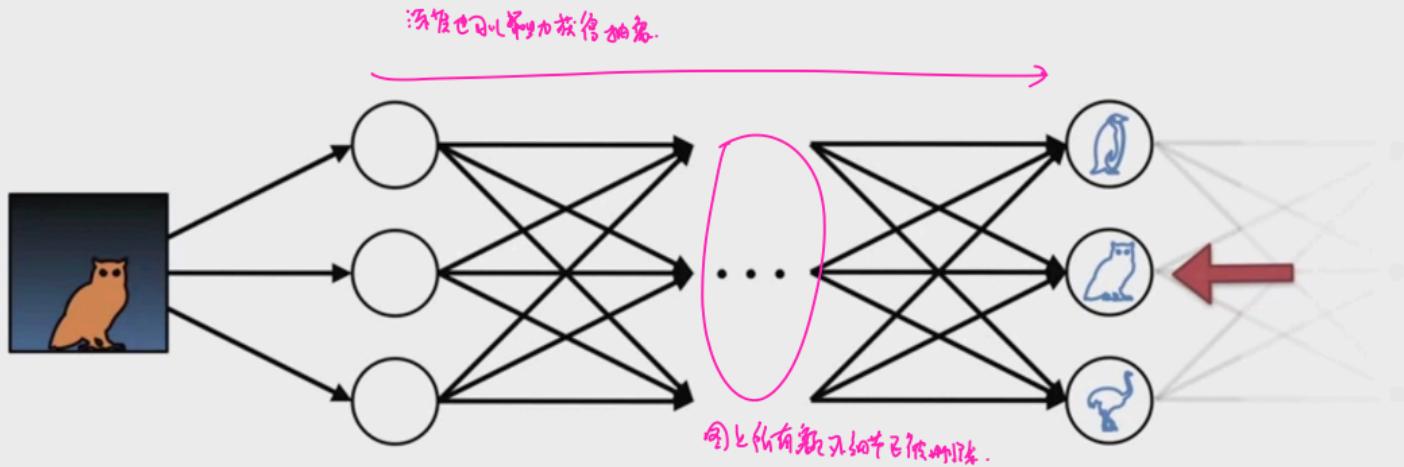
# Compositional features

深度神经网络使复杂函数的近似更容易

(将 features 组合)



# Levels of abstraction

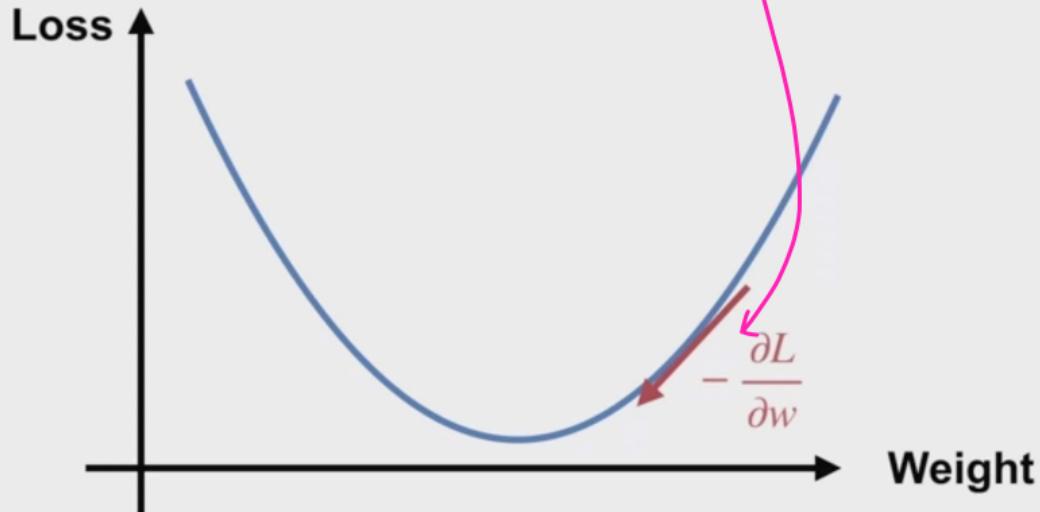


我们的算法的更新一直很简单，大多数都是基于梯度下降的。神经网络的情况也是如此。后向传播算法 back propagation algorithm 实际上并不复杂。事实上，它只是梯度下降。但是，由于 嵌套函数 nested functions 的存在，梯度就显得有些复杂了。在本视频结束时，你将能够推导出神经网络的梯度并在神经网络上实现梯度下降。

与 线性函数逼近 linear function approximation 一样，第一步是定义神经网络参数的损失，然后推导梯度。损失函数规定了网络预测距离正确的程度。我们训练神经网络的目标是找到最小化这个损失函数的参数。一个函数的梯度指向最大的上升方向。通过向与梯度相反的方向移动，我们就会向损失最小化最快的方向移动。那么，我们如何计算神经网络损失函数的梯度？在我们开始之前，让我们建立一些符号。网络的输入是  $S$ ，输出是  $y$  hat。隐蔽层 hidden layer 是学到的特征。权重  $A$  产生特征，权重  $B$  线性加权  $x$  以产生输出。网络的每一层的输出可以表示为一个向量。这里，就是  $x$  和  $y$  hat。

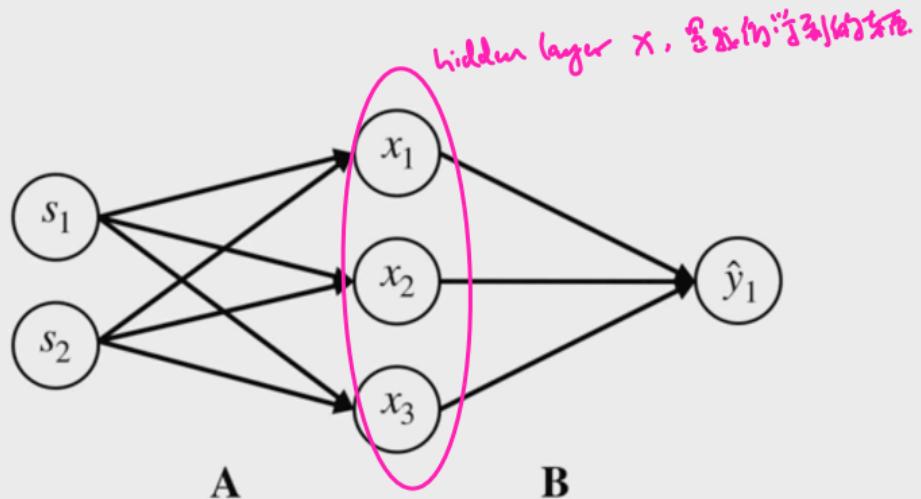
权重矩阵 weight matrices 的第一个索引是指该层的输入，第二个索引是指输出。我们现在假设我们有一个通用的损失  $L$ ，所以我们可以描述基本的想法。 $L$  的一个例子是误差的平方。在推导过程中，我们将保持  $L$  的通用性。我们将在后面给你具体的更新与平方误差。在我们深入研究推导之前，让我们先了解一下方向。虽然推导过程有点复杂，但我们会发现最终的更新是非常简单的。每个参数矩阵都有一个更新，看起来像一个误差项  $\Delta$  乘以该层的输入。对于  $A$  来说，输入是  $S$ ，对于  $B$  来说，输入是  $x$ 。此外，我们会发现  $\Delta_A$  可以被有效地计算为  $\Delta_B$  的函数。来自网络输出的错误  $\Delta_B$  被向后传播到这个早期层，以帮助确定  $A$  在产生该错误中的作用。

## Recap on Gradient Descent

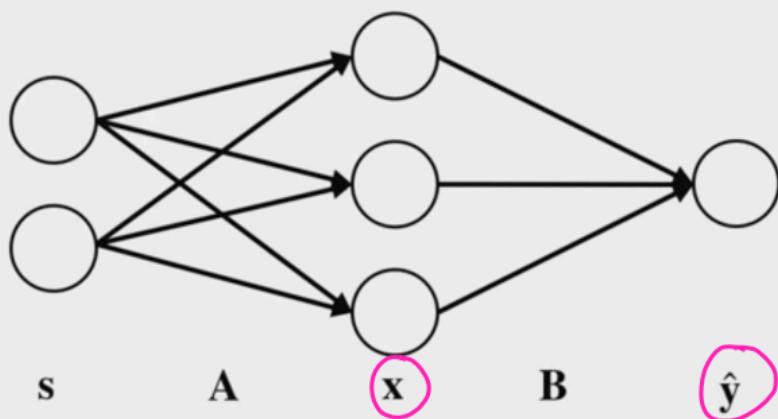


$$w \leftarrow w - \alpha \frac{\partial L}{\partial w}$$

# Notation

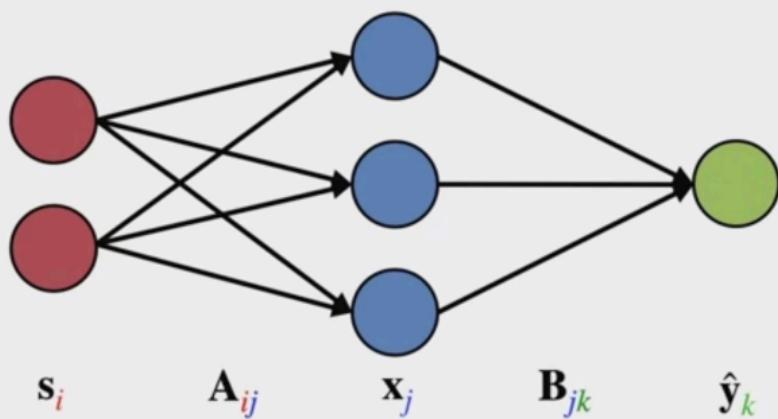


## Notation



神经网络用一层的输出.

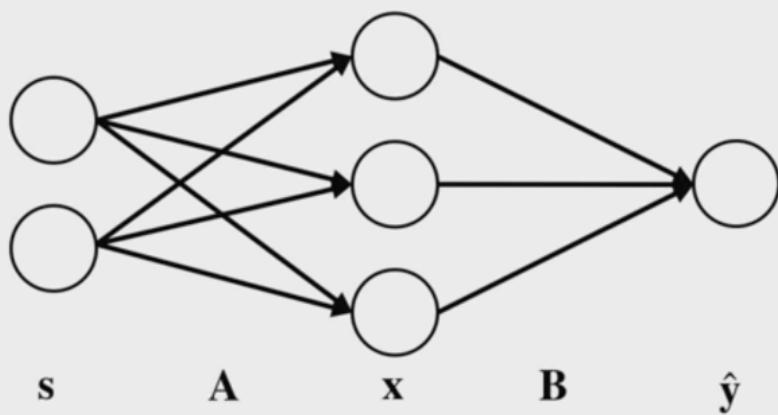
# Notation



$i$  =  $i$  index: input.

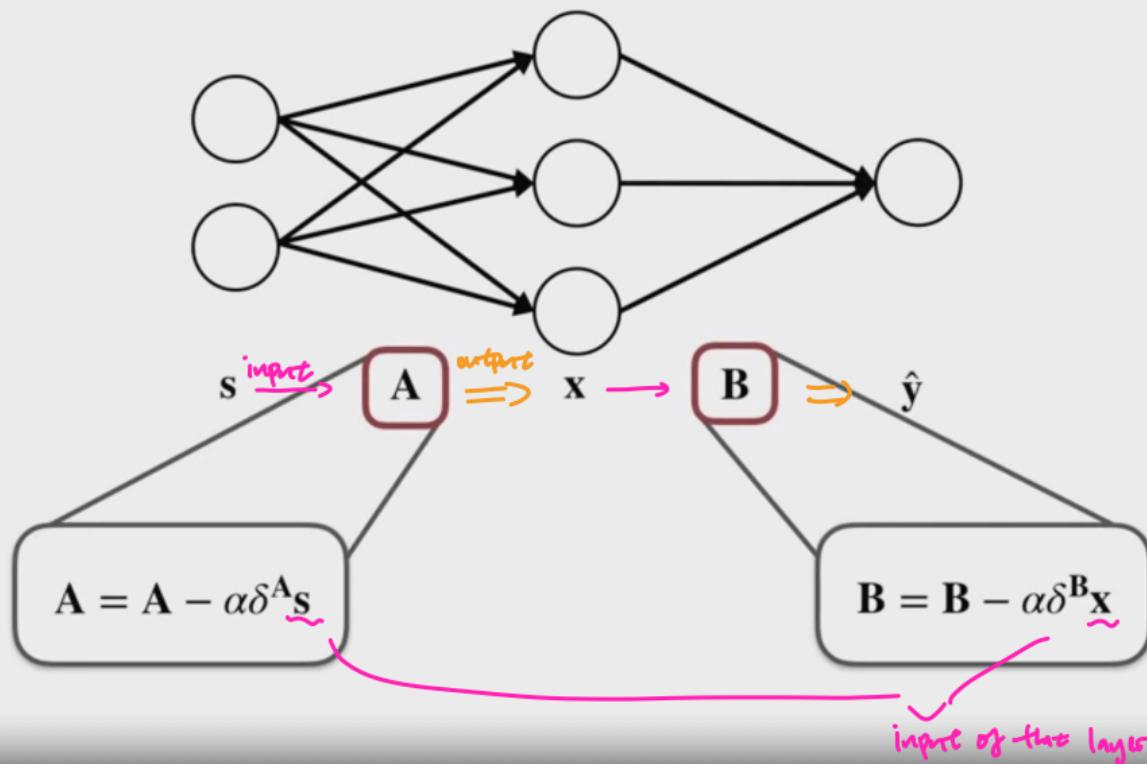
$j$  =  $j$  index: output.

## Notation



$$\underbrace{L(\hat{y}_k, y_k)}_{\text{generic loss}} = (\hat{y}_k - y_k)^2$$

# Goal



input of this layer.

那么，现在我们如何获得Delta A和Delta B？为了好玩，当然也是为了教育目的，我们现在来做个完整的推导。让我们从网络的输出开始，向后推导。这是有原因的，你会看到。我们从损失函数相对于第一组权重B的部分导数开始。我们使用链式规则，给出L相对于y hat的导数乘以y hat相对于B的导数。为了使之更容易，让我们引入一个新的变量，Theta。Theta是隐藏层的输出乘以最后一组权重。让我们把它写在旁边，以便我们记住它。

现在，让我们用Theta重写y hat。让我们展开y hat相对于B的导数。使用连锁规则，我们得到 $f_B$ 相对于Theta的导数乘以Theta相对于B的导数。现在我们已经完成了B的梯度推导，接下来我们将进行A的梯度推导。但在此之前，让我们对损失和激活做一些选择，以便我们可以看到B的这个一般方程的一个具体实例。损失相对于y hat的导数是这样的。输出层的激活的导数是1。这是因为 $f_B$ 是相同的，Theta相对于它的导数是1。这是损失相对于B的通用梯度。将其插入相对于B的梯度，我们得到y hat减去y乘以1乘以x。现在，为了计算相对于A的梯度，我们需要经历与B相同的步骤，主要区别在于我们有一个额外的链式规则步骤，因为权重A也会影响x。让我们从这里开始。让我们用连锁规则展开这个导数。在我们展开下一个项之前，让我们引入另一个辅助变量，Psi等于输入s乘以A。

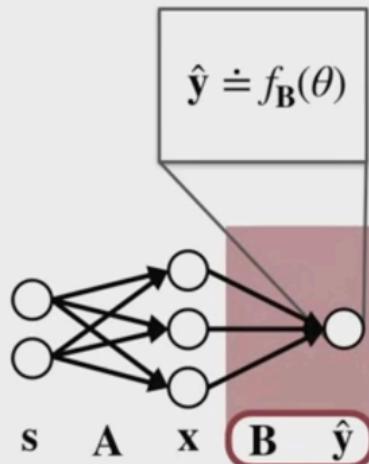
## Deriving the gradient

$$\frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \mathbf{B}_{jk}} = \frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \hat{\mathbf{y}}_k} \frac{\partial \hat{\mathbf{y}}_k}{\partial \mathbf{B}_{jk}}$$

$$= \frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \hat{\mathbf{y}}_k} \frac{\partial f_{\mathbf{B}}(\theta_k)}{\partial \theta_k} \frac{\partial \theta_k}{\partial \mathbf{B}_{jk}}$$

$$= \frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \hat{\mathbf{y}}_k} \frac{\partial f_{\mathbf{B}}(\theta_k)}{\partial \theta_k} \mathbf{x}_j \quad \text{对 } \mathbf{B} \text{ 的梯度疏导}$$

$$\begin{aligned}\mathbf{x} &\doteq f_{\mathbf{A}}(\mathbf{s}\mathbf{A}) \\ \theta &\doteq \mathbf{x}\mathbf{B} \\ \hat{\mathbf{y}} &\doteq f_{\mathbf{B}}(\theta)\end{aligned}$$



让我们把它扔到一边，重写 $x$ 来使用它。然后我们展开下一个项。再次使用连锁法则，我们可以得到 $\Psi_i$ 的 $f_A$ 相对于 $\Psi_i$ 的导数乘以 $\Psi_i$ 相对于 $A$ 的导数。因为 $\Psi_i$ 是 $s$ 乘以 $A$ ，我们得到 $d \Psi_i dA = s$ 。现在，相对于 $A$ 的导数是 $\Delta A$ 乘以其输入 $s$ 。它们都有一个 $\Delta$ 项，包含一个误差信号乘以它们的输入。让我们简单看一下实现随机梯度下降的Backprop算法的一些伪代码。对于我们数据集中的每个数据点 $s, y$ ，我们首先从网络中得到我们的预测值 $y \hat{}$ 。这通常被称为前向传递，然后我们从输出开始计算梯度。我们首先计算 $\Delta B$ 和 $B$ 的梯度，然后我们用这个梯度来更新参数 $B$ ，即步长 $\text{Alpha}_B$ 。它只是用这种有效的策略来计算梯度的梯度下降。然后，我们计算 $A$ 的梯度，并用这个梯度更新 $A$ ，使用步长 $\text{Alpha}_A$ 。前一层的梯度同样是使用下一层的梯度来递归计算的。每一层的更新都是 $\Delta$ 乘以该层的输入的形式。本质上，网络将错误信息存储在每一层的 $\Delta$ 中。为了不那么抽象，让我们看看一个特定网络的反推算法。如果我们在隐藏层上使用ReLU激活，在输出上使用线性单元，那么这里是伪代码。首先，我们计算输出层的误差，然后计算ReLU单元相对于 $\Psi_i$ 的导数，最后，我们用输出层的空中信号  $\text{aerial signal}$  和你一起计算隐藏层的空中信号，其余的不变。

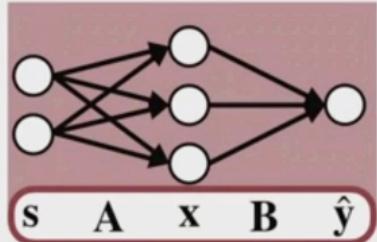
这段视频就到此为止。今天，我们推导出了神经网络的梯度。我们还讨论了Backprop的主要思想，即从网络的输出开始计算梯度可以节省计算量。这是一个相当详细的推导。你实际上不需要记住所有这些步骤。目的是让你对神经网络的更新有一些了解，以及在损失、激活和层的不同选择下可能发生的变化。在你的评估中，你将实现本视频末尾给你的伪代码。在下一个视频中，我们将给你一些更多的工具，使你的神经网络在实践中更有效地实现。//

# Deriving the gradient

$$\frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \mathbf{A}_{ij}} = \delta_j^{\mathbf{A}} \mathbf{s}_i$$
$$\frac{\partial L(\hat{\mathbf{y}}_k, \mathbf{y}_k)}{\partial \mathbf{B}_{jk}} = \delta_k^{\mathbf{B}} \mathbf{x}_j$$

.. 它们的梯度下降都是 Data  $\times$  该层的 input.

$$\psi \doteq \mathbf{s} \mathbf{A}$$
$$\mathbf{x} \doteq f_{\mathbf{A}}(\psi)$$
$$\theta \doteq \mathbf{x} \mathbf{B}$$
$$\hat{\mathbf{y}} \doteq f_{\mathbf{B}}(\theta)$$

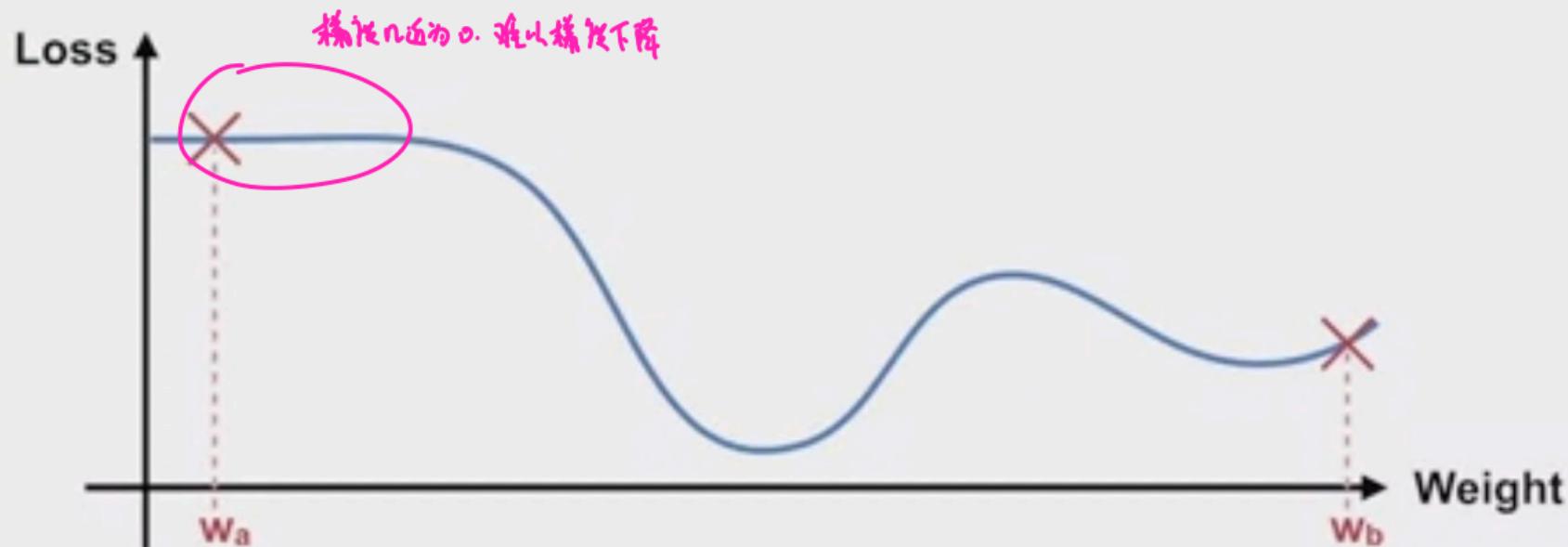


基于深度神经网络的监督学习系统，现在是图像分类、语音识别和自然语言处理的首选解决方案。部分原因是由于训练数据和可负担得起的计算的急剧增加。但是为了真正利用这些数据和计算的增加，需要改进和训练。几个简单的优化策略，使训练网络变得更加容易，并有助于加速采用。我们今天就来谈谈这些。在本视频结束时，你将能够理解初始化对神经网络的重要性，并描述训练神经网络的优化技术。

像许多机器学习方法一样，神经网络的训练是使用一个迭代的过程。这个过程，必须有一些起始点。这个起点的选择，可以对神经网络的性能起到很大的作用。让我们来看看一个只有一个权重的例子。在Y轴上，我们将显示损失函数。在X轴上，我们将显示权重的值。正如你所知道的，梯度下降将迭代地将权重向最近的静止点移动。但神经网络的损失函数并不那么简单。如果我们从这个几乎平坦的区域开始，由于梯度接近零，梯度下降就很难取得任何进展。如果我们从这个小碗里开始，那么我们可以很快找到局部最优。

但是，如果它能从这里的某个地方开始，我们可以迅速找到一个更好的最优点，那不是很好吗？一个简单而有效的初始化策略是，从一个小方差的正态分布中随机抽取初始权重。这样一来，每个神经元的输出都与本层的其他神经元不同。这就提供了一套更多样化的潜在特征。通过保持小的变异，我们确保每个神经元的输出与它的邻居在同一范围内。这种策略的一个缺点是，当我们向一个神经元添加更多的输入时，输出的变异会增长。我们可以通过缩放权重的方差来解决这个问题，缩放量为输入数的平方根的1。在我们选择了网络的起点之后，我们开始使用随机梯度下降步骤逐步对权重进行小的改进。另一种改进训练的方法是考虑更复杂的更新机制。两种常见的策略是使用重球法，也称为动量和矢量步长适应。我们先来谈谈动量。

It matters where you start



想象一下，这是随机梯度下降下的二维波矢量 two-dimensional wave vector 的轨迹。这里是随机梯度下降的更新规则，这里是修改后的更新，包括动量。注意，它类似于常规的随机梯度下降更新，再加上一个额外的项，叫做动量M。动量项用衰减率为Lambda的梯度总和来总结梯度的历史。如果最近的梯度都在类似的方向上，那么我们就在这个方向上获得了动量。这意味着，我们在这个方向上迈出了一大步。如果最近的更新有冲突的方向，那么就会扼杀动量。动量项对更新的影响不大，我们会做一个常规的梯度下降步骤。事实证明，动量可以加速学习，这意味着它能更快地到达一个静止的点。另一个潜在的改进是为网络中的每个权重使用单独的步骤大小。到目前为止，我们只谈到了一个全局标量的步长。众所周知，这是有问题的，因为这可能导致对某些权重的更新过大，而对其他权重的更新过小。根据对实践中学习过程的统计，调整每个权重的步长，可以获得更好的性能。现在，更新是如何改变的？这个变化非常简单。不是用一个标量Alpha来更新，而是用一个以t为索引的步长向量来表示它可以在每个时间步长上变化。梯度的每个维度，都由其相应的步长而不是全局的步长进行缩放。有多种方法来调整步长的矢量。你将在你的作业中实现一个。

本视频就到此为止。今天，我们讨论了改进神经网络训练的方法。具体来说，我们讨论了在神经网络中初始化权重的一种策略，以及如何利用动量和步长适应来加速学习。现在，你已经准备好用神经网络实现一个TD代理了。//

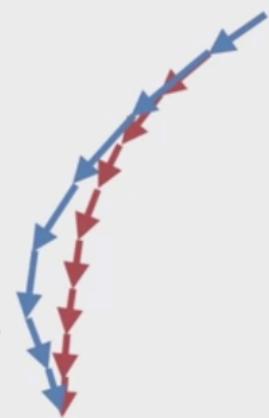
## Update momentum

梯度的 weight 更新 (紅線)

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \alpha \nabla_{\mathbf{w}} L(\mathbf{w}_t) + \lambda \mathbf{M}_t$$

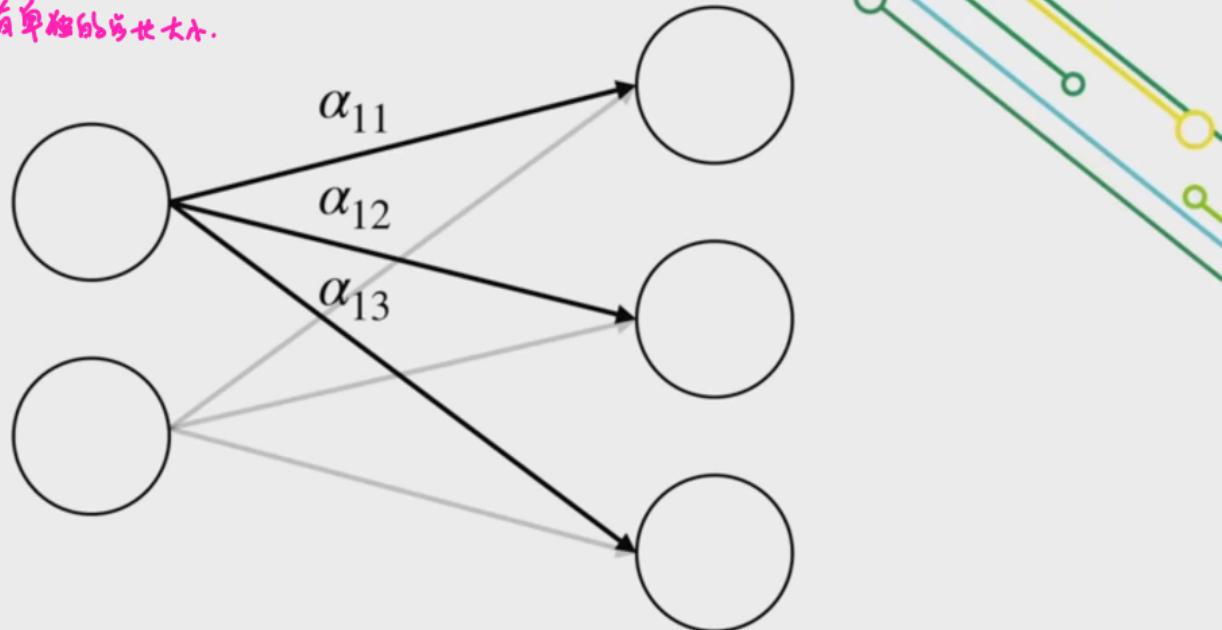
$\mathbf{M}_{t+1} \leftarrow \lambda \mathbf{M}_t - \alpha \nabla_{\mathbf{w}} L$

Not momentum is for the weight更新 (藍線).



## Vector step sizes

每个权重都有单独的步伐大小。



本周，我们讨论了代表大型、不可能连续的状态空间的方法。构建特征的方法。表征是代理人对状态的内部编码，代理人构建特征来总结当前的输入。每当我们在谈论特征和表征学习时，我们都是在函数近似的领域。这就把我们带到了我们的课程地图的左边。

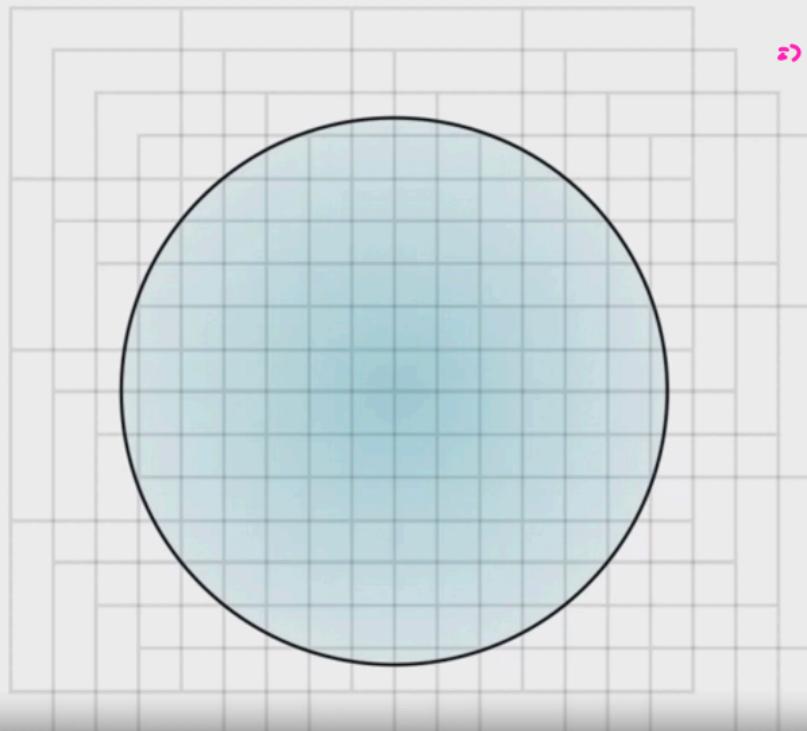
首先，我们介绍粗糙编码，粗糙编码与状态聚合有关，它将相邻的状态分组，每个分组可以有一个任意的形状，二维粗糙编码的一个特定例子由这些重叠的圆圈表示。每个圆圈都是一个特征，当状态在圆圈内时为1，而当状态在圆圈外时为0。

接下来，我们讨论一种特殊的粗糙编码类型，即瓦片编码。瓦片编码使用一组重叠的网格生成特征，每个网格被称为瓦片。一个平铺本身没有重叠或方格之间的空间，一次只能有一个特征被激活。通过堆叠多个偏移的瓦片，我们可以分辨出不同的状态。瓦片的形状、大小和数量帮助我们平衡泛化、区分和计算效率。然后我们讨论了一种用神经网络在线学习表征的方法。通过粗糙编码技术，表征在学习之前是固定的。

一个前馈神经网络 feed-forward neural network 使用一系列的层来产生一个表征。在每一层中，多个神经元接收相同的输入并产生不同的输出。然后这些输出被送入下一层，这个过程重复进行。每个神经元通过对输入进行加权求和并通过激活函数来计算其输出。

为了训练一个神经网络，我们使用了一个叫做梯度下降的迭代过程。我们将输入传入网络以产生预测，然后我们将这些预测与输出进行比较并计算损失函数。最后，我们计算损失函数的导数，并将我们的学习规则应用于权重。下周，我们将讨论如何通过函数近似来实现奖励最大化的学习，届时见。

# Tile Coding



⇒ discrimination: 圆柱重叠多个  
相邻的 tile 来实现

# Constructing Features for Prediction

最新提交作业的评分 100%

1. Which of the following is TRUE about coarse coding? (Select all that apply)

1 / 1 分

- In coarse coding, generalization occurs between states that have features with overlapping receptive fields.

 正确

Correct.

- In coarse coding, generalization between states depend on the size and shape of the receptive fields.

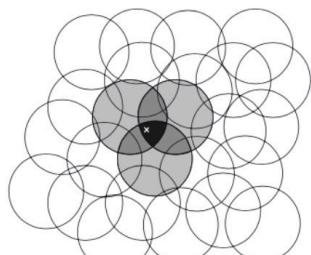
 正确

Correct.

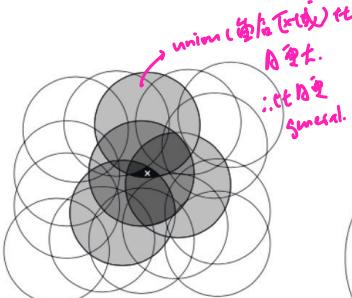
- When using features with large receptive fields, the function approximator cannot make discriminations that are finer than the width of the receptive fields.

- When training at one state, the learned value function will be updated over all states within the intersection of the receptive fields.

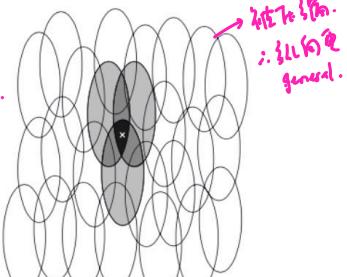
2. Consider a continuous two-dimensional state space. Assuming linear function approximation with the coarse-codings in either A, B or C, which of the following is TRUE? (Select all that apply)



A



B



C

- Generalization is broader in case A as compared to case B.
- In case B, when updating the state marked by an 'x' , the value function will be affected for a larger number of states as compared to case A.



正确

Correct. In case B, the receptive fields of the features are larger and include a larger number of states.

- In case C, each update results in more generalization along the vertical dimension, as compared to horizontal dimension.



正确

Correct. Updates to the state marked by the 'x' change the values for more states further away in the vertical dimension, as indicated by the greyed areas.

- In case C, each update results in more generalization along the horizontal dimension, as compared to vertical dimension.

3. Which of the following affects generalization in tile coding? (Select all that apply)

1/1分

- The shape of the tiles.

正确

Correct.

- The number of tiles.

正确

Correct.

- The number of tilings.

正确

Correct.

- How the tilings are offset from each other.

正确

Correct.

4. When tile coding is used for feature construction, the number of active or non-zero features

1/1分

- is the number of tiles.
- is the number of tilings.
- is the number of tilings multiplied by the number of tiles.
- depends on the state.



正确

Correct.

5. Which of the following is TRUE about neural networks (NNs) ? (Select all that apply)

1/1 分

- A NN is feedforward if there are no paths within the network by which a unit's output can influence its input.

 正确

Correct.

- Hidden layers are layers that are neither input nor output layers.

 正确

Correct.

- The output of the units in NNs are typically a linear function of their input signals.

- NNs are parameterized functions that enable the agent to learn a nonlinear value function of state.

 正确

Correct.

- The nonlinear functions applied to the weighted sum of the input signals are called the activation function.

 正确

Correct.

6. Which of the following is the rectified linear activation function?

1/1分

- $f(x) = \frac{1}{1+e^{-x}}$
- $f(x) = 1$  if  $x > 0$  and 0 otherwise
- $f(x) = \max(0, x)$
- $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

✓ 正确

Correct.

7. Which of the following is TRUE about neural networks (NNs)?

1/1分

- A NN with a single hidden layer can represent a smaller class of functions compared to a NN with two hidden layers.
- The universal approximation property of one-hidden-layer NNs is not true when linear activation functions are used for the hidden layer.
- Given the universal approximation property of one-hidden-layer NNs, there is no benefit to including more layers in the network.

✓ 正确

Correct.

8. Which of the following is TRUE about backpropagation? (Select all that apply)

1/1分

- Backpropagation corresponds to updating the parameters of a neural network using gradient descent.

正确

Correct. Neural networks are commonly trained using gradient descent. Backpropagation is an efficient way to compute and apply the gradient update.

- Backpropagation involves computing the partial derivatives of an objective function with respect to the weights of the network.

正确

Correct.

- The forward pass in backpropagation updates the weights of the network using the partial derivatives computed by the backward passes.
- Backpropagation computes partial derivatives starting from the last layer in the network, to save computation.

正确

Correct. Because of the nested structure in the neural network, the partial derivatives for earlier layers have some shared components with layers near the output. Starting from the output layer means we can cache some of these shared computations, and avoid needlessly recomputing them.

9. Training neural networks (NNs) with backpropagation can be challenging because (Select all that apply)

- the loss surface might have flat regions, or poor local minima, meaning gradient descent gets stuck at poor solutions.



正确

Correct.

- the initialization can have a big impact on how much progress the gradient updates can make and on the quality of the final solution.



正确

Correct.

- neural networks cannot accurately represent most functions, so the loss stays large.

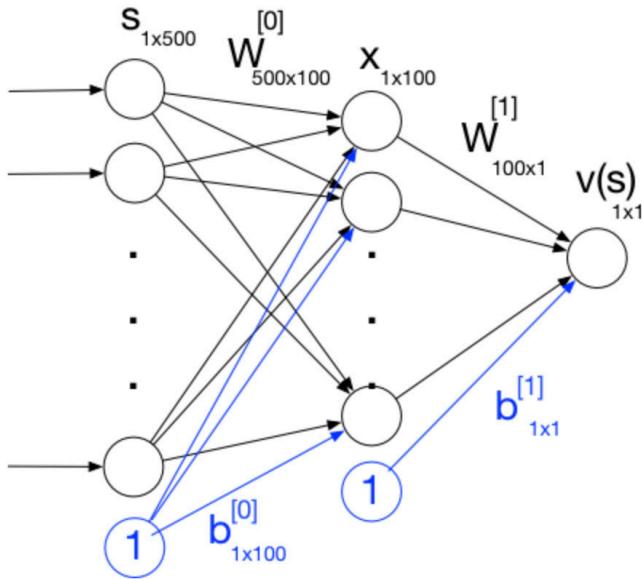
- learning can be slow due to the vanishing gradient problem, where if the partial derivatives for later nodes in the network are zero or near zero then this causes earlier nodes in the network to have small or near zero gradient updates.



正确

Correct.

10. Consider the following network:



where for a given input  $s$ , value of  $s$  is computed by:

$$\psi = sW^{[0]} + b^{[0]}$$

$$x = \max(0, \psi)$$

$$v = xW^{[1]} + b^{[1]}$$

What is the partial derivative of  $v(s)$  with respect to  $W_{ij}^{[0]}$ ?

$s_i$

$W_j^{[1]}s_i$  if  $x_j > 0$  and 0 otherwise

$x_j$

$W_j^{[1]}x_j$  if  $x_j > 0$  and 0 otherwise

正确

Correct.

11. Which of the following is TRUE? (Select all that apply)

1/1分

- When using stochastic gradient descent, we often completely eliminate the error for each example.
- The difference between stochastic gradient descent methods and batch gradient descent methods is that in the former the weights get updated using one random example whereas in the latter they get updated based on batches of data.

正确

Correct.

- Adagrad, Adam, and AMSGrad are stochastic gradient descent algorithms with adaptive step-sizes.

正确

Correct.

- Setting the step-size parameter for stochastic gradient descent can be challenging because a small step-size makes learning slow and a large step-size can result in divergence.

正确

Correct.

12. Which of the following is TRUE about artificial neural networks (ANNs)? (Select all that apply)

1/1分

- It is best to initialize the weights of a NN to large numbers so that the input signal does not get too small as it passes through the network.
- It is best to initialize the weights of a NN to small numbers so that the input signal does not grow rapidly as it passes through the network.
- If possible, it would be best to initialize the weights of an NN near the global optimum.

✓ 正确

Correct. Then the solution is just a few gradient descent steps away!

- A reasonable way to initialize the NN is with random weights, with each weight sampled from a normal distribution with the variance scaled by the number of inputs to the layer for that weight.

✓ 正确

Correct. This is the initialization strategy we discussed. It is by no means optimal, and how to improve the initialization is the subject of ongoing research.