9 在线策略预测和近似

在这章节，我们开始强化学习中函数近似的学习，通过考虑它在从在线策略数据估计状态-值函数中的使用，即从使用一个已知的策略 π 生成的经验来近似 v_π. 本章的新颖性是近似的值函数的表示不是用一张表，而是一个带有权重向量 $\mathbf{w} \in \mathbb{R}^d$ 的参数函数形式。我们会把给定权重向量 w 下的状态 s 的近似值写作 $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$. 比如说， $\hat{v}$ 可以是状态特征的一个线性函数，带有特征权重向量 $\mathbf{w}$. 更一般地， $\hat{v}$ 可以是由多层人工神经网络计算的函数，带有所有层中的连接权重向量 $\mathbf{w}$. 通过调整权重，大范围的不同函数中的任何一个都能被网络实现。或者 $\hat{v}$ 可以是由决策树计算的函数，其中 $\mathbf{w}$ 是所有定义树的分支点和叶节点值的数字。一般地，权重的数量 ($\mathbf{w}$ 的维度) 是要远远小于状态的数量 ($d \ll |\mathcal{S}|$)，改变一个权重会改变很多状态的估计值。因此，当一个单一状态被更新时，这个变化会从该状态泛化去影响到很多其它状态的值。这种泛化使得学习在潜在上更有力，同样也更困难管理和理解。

或许惊讶的是，把强化学习拓展到函数近似也使得它能够应用到部分可视问题上，也就是对于代理者不能到达所有的状态。如果对于 $\hat{v}$ 的参数方程形式不允许估计值去依赖状态的一些方面，那么它就只当那些方面是不可视的。实际上，所有为本书这部分使用函数近似表示的方法的理论结果都可以同等良好地应用到部分可视化的情形。函数近似不能做到的，却是用过去的观察的记忆来增强状态表示。这样的一些可能深入的拓展会在章节17.3简要讨论。

9.1 值函数近似

本书包括的所有预测方法，都可以被描述成到一个估计的值函数的更新，即值函数会改变在特定的状态的值，来趋于一个 "*backed-up* 值" ，或者对于这个状态的**更新目标** (*update target*). 我们用记号 $s \mapsto u$ 指代一次单独的更新，其中 s 是要更新的状态，u 是 s 的估计值改变趋向的更新目标。比如说，MC的值预测更新是 $S_t \mapsto G_t$ ，TD(0) 的更新是 $S_t \mapsto R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t)$, n 步TD的更新是 $S_t \mapsto G_{t:t+n}$. 在DP策略-评价更新中，$s \mapsto \mathbb{E}_\pi[R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) | S_t = s]$ ，任意一个状态 s 会被更新，尽管在其它情形中只有在实际经历中遇到的状态 S_t 会被更新。

自然地，可以把每次更新解释成在具体说明值函数的期望输入-输出性能的一个样例。在某种程度上，更新 $s \mapsto u$ 意味着对状态 s 的估计值应当更像更新目标 u. 目前为止，实际的更新变得不再重要：对 s 的估计值的表实体已经被简化变成趋于 u 的方式的一部分，而所有其它状态的估计值不会被改变。现在我们允许任意复杂先进的方法来实现更新，并且在 s 更新会泛化使得很多其它状态的估计值也会被改变。用这种方式来学习拟合输入-输出样例的机器学习方法被称为**监督学习** (*supervised learning*) 方法，而且当输出比如 u 是数字时，这个过程通常称为**函数近似** (*function approximation*). 函数近似方法期待得到它们试图近似的函数的想要的输入-输出性能的样例。我们对值预测使用这些方法，简单地通过传给它们每次更新的 $s \mapsto u$ 作为一个训练样例。然后我们解释它们生成的近似函数为一个估计值函数。

用这种方式把每次更新视作一个传统的训练样例，能够使我们能对值预测使用广泛的任何现有函数近似方法。原则上我们能使用从样例中使用监督学习的任何方法，包括人工神经网络，决策树，和各种类型的多元回归。然而，不是所有的函数近似方法都同等适用于强化学习。最复杂的人工神经网络和统计方法都假定在一个稳态的训练集上进行多次传递。但在强化学习中，学习能够在线进行是重要的，在代理者和它的环境或者它的环境模型交互时。为了做到这点需要能够从渐增获取的数据中能高效学习的方法。另外，强化学习通常需要函数近似方法能够处理非稳态目标函数 (目标函数会随时间变化). 比如，在基于GPI (*generalized policy iteration*) 的控制方法中，我们通常试图学习 q_π 当 π 改变时。即便策略保持不变，训练样例的目标值也是非稳态的，如果它们通过拔靴方法 (DP 和 TD 学习) 来生成样例。不能轻松处理这种非稳态性的方法不太适合强化学习。

9.2 预测目标 ($\overline{VE}$)

目前为止，我们没有具体指出对预测的一个明确的目标。在列表性情形下，预测质量的一个连续度量是不需要的，因为学习的值函数最终能精确等于正确的值函数。而且，每个状态的学习值是被分离的——在一个状态的更新不会影响其它状态。但用真正的近似，在一个状态的更新会影响到很多其它状态，且不可能使得所有状态的值都完全正确。假定我们有远多于权重的状态，那么使一个状态的估计始终更精确意味着使其它状态的估计更不精确。然后我们有义务说明哪些状态是我们最关心的。我们必须具体指出一个状态分布 $\mu(s) \geq 0, \sum_s \mu(s) = 1$，来表示我们对每个状态 s 中的误差关心的程度。我们用状态 s 中的误差来表示近似值 $\hat{v}(s, \mathbf{w})$ 和真值 $v_\pi(s)$ 之间的差值的平方。在状态空间上用 μ 对这加权，我们得到一个自然的目标函数，**均方值误差** (*mean square value error*), 记作 $\overline{VE}$:

$$\overline{VE}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s) \big[v_\pi(s) - \hat{v}(s, \mathbf{w}) \big]^2. \tag{9.1}$$

这个度量的方根，即根 $\overline{VE}$，给了一个对近似值和真值的差异大小的大致衡量，也经常被用于作图。通常 $\mu(s)$ 被选成在 s 上花费时间的分比。在线策略训练下这称为**在线策略分布** (*on-policy distribution*); 我们在本章完全集中于这种情形。在连续任务中，在线策略分布是在 π 下的平稳分布。

> ### 在*episodic*任务中的在线策略分布
>
> 在一个episodic任务中，在线策略分布会略微不同，因为它依赖于 episodes 的初始状态选择。令 $h(s)$ 表示一个 episode 在每个状态 s 开始的概率，再令 $\eta(s)$ 表示在一个单 episode 中每个状态 s 平均耗费的时间步数。时间耗费在状态 s 上，要么 episodes 从 s 开始，要么从一个先前状态 $\overline{s}$ 迁移到 s，则状态 s 的耗费时间为:
>
> $$\eta(s) = h(s) + \sum_{\overline{s}} \eta(\overline{s}) \sum_a \pi(a|\overline{s}) p(s|\overline{s}, a), \quad \text{for all } s \in \mathcal{S}. \tag{9.2}$$
>
> 解这个方程组能得到访问次数的期望 $\eta(s)$. 那么在线测率分布便是时间耗费在每个状态上的分比，分比被标准化使得和为1:
>
> $$\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}, \quad \text{for all } s \in \mathcal{S}. \tag{9.3}$$
>
> 这里是没有折扣的正常选择。如果有折扣 ($\gamma < 1$)，它应该被当作一个终止的形式，也就是简单地向 (9.2) 中的第二项加上因数 γ.

这两种情形，连续的和episodic，表现相似，但用近似下的正式分析中，它们必须被分别对待，如我们将在书的这部分中反复看到的那样。这完成了对学习目标的具体描述。

还不完全清楚 $\overline{VE}$ 是不是对强化学习真正的性能目标。记住我们的最终目的——我们学习一个值函数的原因——是为了找个一个更好的策略。为这个目的的最好的值函数不一定是为最小化 $\overline{VE}$ 的最好的。但是，尚未明确对值预测的一个更有用的替代目标应该是什么。所以现在我们将聚焦于 $\overline{VE}$.

在 $\overline{VE}$ 上一个理想目标是找到一个**全局最优** (*global optimum*)，一个权重向量 $\mathbf{w}^*$ 使得 $\overline{VE}(\mathbf{w}^*) \leq \overline{VE}(\mathbf{w})$ 对所有可能的 $\mathbf{w}$ 成立。达到这个目标有时候是可能的，对于简单的函数近似器比如线性的；但几乎不可能对于复杂函数近似器比如人工神经网络和决策树。达不到这点，复杂函数近似器便试图收敛到一个**局部最优** (*local optimum*)，一个权重函数 $\mathbf{w}^*$ 使得 $\overline{VE}(\mathbf{w}^*) \leq \overline{VE}(\mathbf{w})$ 对在 $\mathbf{w}^*$ 的某个邻域中所有 $\mathbf{w}$ 成立。尽管这个保证仅能稍微可靠的，但对于非线性函数近似器通常能说是最好的，而且往往这已经是足够的。然而，对于很多强化学习感兴趣的情形，没有保证能收敛到一个最优，甚至不能收敛到最优点的一个有界范围内。一些方法可能实际上发散，在极限下它们的 $\overline{VE}$ 会达到无穷。

这两个小节我们已经概括了一个框架，来把一个广泛的强化学习的值预测方法和一个广泛的函数近似方法结合起来，使用前者的更新为后者生成训练样例。我们也描述了 $\overline{VE}$ ，这些方法会想要最小化的一个性能衡量。可能的函数近似方法的范围过于巨大以致不能全部包含，而且不管怎样对它们的大多数了解得太少以致不能做一个可靠的评估或推荐。必要地，我们仅考虑一些可能性。在本章的剩余部分我们会集中在基于梯度准则的函数近似方法，特别是线性梯度下降方法。我们聚焦于这些方法，一部分因为我们认为它们特别有前景，而且因为它们揭露了关键的理论问题，另一部分是因为它们是简单的而且我们的空间是有限的。

9.3 随机梯度和半梯度方法

现在我们具体开发一类在值预测中对函数近似的学习方法，这些方法基于**随机梯度下降** (*stochastic gradient descent* , SGD). SGD方法是所有函数近似方法中使用最广泛的之一，并且特别适用于在线强化学习。

在梯度下降方法中，权重向量是一个有固定数量的实值分量的列向量 $\mathbf{w} \doteq (w_1, w_2, \dots, w_d)^T$ ，而近似值函数 $\hat{v}(s, \mathbf{w})$ 是一个关于 $\mathbf{w}$ 的可微方程，对于所有 $s \in \mathcal{S}$. 我们将会在一个离散时间步的序列中的每一步，$t = 0, 1, 2, 3, \dots$ ，更新 $\mathbf{w}$ ，所以我们需要一个记号 $\mathbf{w}_t$ 来表示在每步的权重向量。现在，我们假设在每一步观察一个新的样例 $S_t \mapsto v_\pi(S_t)$ 包含一个(尽可能随机选择的)状态 S_t 和它在策略下的真值。这些状态可能是相继的状态，从和环境的交互中得到的，但我们现在我们不这么假设。即便我们被给予完全且正确的值，$v_\pi(S_t)$ 对于每个 S_t ，仍然有一个困难的问题，因为我们的函数近似器只有有限的资源，因此只有有限的解决。特别地，一般没有 $\mathbf{w}$ 能够得到所有状态，或甚至所有的样例，完全正确的。此外，我们必须泛化到所有其它没有出现在样例中的状态。

我们假设出现在样例中的状态有相同的分布 μ ，在这分布上我们试图最小化 (9.1) 给出的 $\overline{VE}$. 这个情形下一个好的策略是在观察到的样例上最小化误差。**随机梯度下降** (SGD) 方法实现这点通过在每个样例后往最能够减少在该样例上的误差的方向上调整权重向量一小步：

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\Big[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\Big]^2 && (9.4)\\ &= \mathbf{w}_t + \alpha\Big[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\Big]\nabla\hat{v}(S_t, \mathbf{w}_t), && (9.5)\end{aligned}$$

其中 α 是一个正步长参数，$\nabla f(\mathbf{w})$ ，对是一个向量 (这里是 $\mathbf{w}$) 的函数的任何标量表达式 $f(\mathbf{w})$ ，表示该表达式关于向量分量的偏微分的列向量：

$$\nabla f(\mathbf{w}) \doteq \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \dots, \frac{\partial f(\mathbf{w})}{\partial w_d}\right)^T. \tag{9.6}$$

这个微分向量是 f 关于 $\mathbf{w}$ 的**梯度** (*gradient*). SGD方法是"梯度下降"方法，因为全部 $\mathbf{w}_t$ 中的步是正比于样例的方误差的负梯度。这是误差下降最快的方向。梯度下降方法被称作"随机的"，当更新就如这里仅在一个可能随机选择的单一样例上完成时。历经很多样例，走了很多小步，整体的效果是最小化一个平均性能度量比如 $\overline{VE}$.

也许不能立刻明白为什么SGD在梯度方向上只走一小步。我们难道不能在这个方向上一直移动，完全消除在样例上的误差吗？在很多情形下这能做到，但通常这是不需要的。记住我们不是试图或期待找到一个值函数能够对所有状态都是零误差，而仅仅是一个近似能够平衡不同状态之间的误差。如果我们完全修正每步中的样例，那我们就不会找到这样的一个平衡。实际上，SGD方法的收敛性结果假定 α 随时间递减。如果它按标准随机近似条件 (2.7) 的方式递减，那么SGD方法 (9.5) 能保证收敛到一个局部最优。

我们现在转向另一种情形，当 第 t 次训练样例 $S_t \mapsto U_t$ 的目标输出 $U_t \in \mathbb{R}$ 不是真值 $v_\pi(S_t)$ ，而是某个到它的可能随机的近似。比如，U_t 可能是 $v_\pi(S_t)$ 的一个参杂噪声版本，或者它可能是之前小节提到的使用 $\hat{v}$ 的拔靴目标之一。在这些情形中我们不能执行严格的更新 (9.5) 因为 $v_\pi(S_t)$ 是未知的，但我们能通过把 $v_\pi(S_t)$ 替换为 U_t 来近似它。这等价于下面的对状态-值预测的一般SGD方法：

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \Big[U_t - \hat{v}(S_t, \mathbf{w}_t) \Big] \nabla \hat{v}(S_t, \mathbf{w}_t). \tag{9.7}$$

如果 U_t 是一个**无偏**估计，也就是，如果 $\mathbb{E}[U_t | S_t = s] = v_\pi(s)$ 对每个 t 成立，那么 $\mathbf{w}_t$ 能保证收敛到一个局部最优，在 α 递减满足通常的随机近似条件 (2.7) 下。

比如，假设样例中的状态是使用策略 π 与环境交互 (或者仿真交互) 生成的状态。因为一个状态的真值是它之后的回报的期望值，所以MC目标 $U_t \doteq G_t$ 由定义是 $\hat{v}_\pi(S_t)$ 的一个无偏估计。用这个选择，一般SGD方法 (9.7) 收敛到 $v_\pi(S_t)$ 的一个局部最优近似。因此，MC状态-值预测的梯度下降版本是保证能找到一个局部最优解。一个完整的算法的伪代码见下面方框。

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
  Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \big[G_t - \hat{v}(S_t, \mathbf{w}) \big] \nabla \hat{v}(S_t, \mathbf{w})$

我们不能得到相同的保证如果 $\hat{v}_\pi(S_t)$ 的一个拔靴估计被用作在 (9.7) 中的目标 U_t. 拔靴目标比如 n 步返回 $G_{t:t+n}$ 或DP目标 $\sum_{a,s',r} \pi(a|S_t) p(s', r | S_t, a) [r + \gamma \hat{v}(s', \mathbf{w}_t)]$ 都依赖于权重向量 $\mathbf{w}_t$ 的当前值，也就意味着它们是有偏的且它们不会得到一个真正的梯度下降方法。看这个的一种方式是从 (9.4) 到 (9.5) 的关键步骤依赖于目标是独立于 $\mathbf{w}_t$ 的。如果使用一个拔靴估计来取代 $v_\pi(S_t)$，这个步骤就会无效。拔靴方法实际上不是真正的梯度下降的实例 (Barnard, 1993). 它们只考虑了改变权重向量 $\mathbf{w}_t$ 在估计上的影响，却忽略了在目标上的影响。因为它们只包含一部分的梯度，因此我们称它们为**半梯度方法** (*semi-gradient methods*).

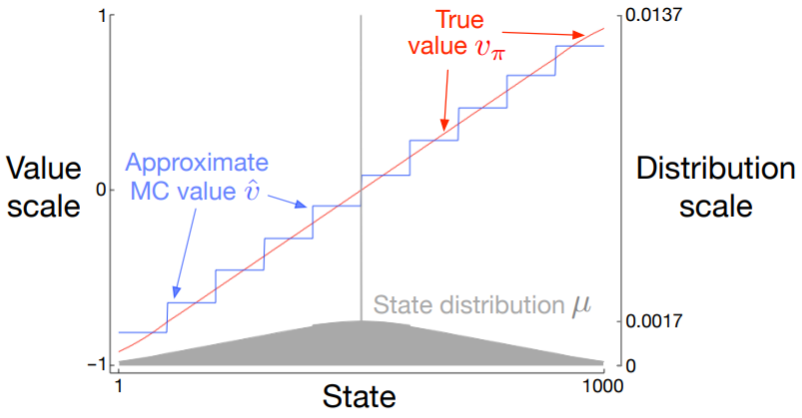
尽管半梯度 (拔靴) 方法不能像梯度方法那样稳定地收敛，它们在重要的情形下确实可靠地收敛，比如下一小节要讨论的线性情形。此外，它们提供了重要的优势，使得它们往往被优先考虑。一个原因是它们通常能够显著加快学习，如我们在章节 6 和 7 中看到。另一个原因是它们使得学习能连续和在线的，不需要等到一个 episode 的结束。这使得它们能能被用到连续性问题而且提供计算优势。一个典型的半梯度方法是半梯度TD(0)，使用 $U_t \doteq R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$ 作为它的目标。这个方法的完整伪代码见下面的方框。

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

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ 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'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \big[R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w}) \big] \nabla \hat{v}(S, \mathbf{w})$
    $S \leftarrow S'$
  until S is terminal

状态类聚 (*state aggregation*) 是泛化函数近似的一种简单形式，其中状态被分组在一起，每组有一个估计值 (权重向量 $\mathbf{w}$ 的一个分量). 状态的值当作它的组分量被估计，当状态被更新时，仅该分量会被更新。状态类聚是 SGD (9.7) 的一种特殊情形，其中梯度 $\nabla\hat{v}(S_t, \mathbf{w}_t)$ 对 S_t 的组分量是 1，对其它分量是 0.

> ### 例9.1：在1000状态的随机游走上的状态类聚
>
> 考虑一个1000状态版本的随机游走任务 (例6.2 和 7.1). 状态被从左到右，按1到1000编号，而且所有 episodes 从接近中心的状态500开始。状态转移是从当前状态到它的左边100个邻近状态中或右边100个邻近状态中的一个，全部都等概率的。当然，如果当前状态接近一端的边缘，那它的那侧会少于100个邻近状态。这种情形下，所有缺失邻近状态的概率会加到那侧的终止状态上 (比如，状态1有0.5的概率到左侧的终止状态，状态950有0.25的概率到右侧的终止状态). 和之前一样，终止在左侧生成一个 -1 的回报，终止在右侧生成一个 1 的回报。所有其它转移有一个 0 的回报。在这节我们至始至终把这个任务当作运行示例。
>
> 图 9.1 显示了这个任务的真值函数 v_π. 它接近一条直线，仅在两端的最后100个状态会略微趋于水平。同样显示了通过用状态类聚的梯度MC算法学习，在步长 $\alpha = 2 \times 10^{-5}$ 和 100,000个 episodes之后的最终近似值函数。对于状态类聚，1000个状态被按序分为10组，每组100个状态 (即状态1-100是一组，状态101-200是一组，以此类推). 图中展示的阶梯效应是典型的状态类聚；在每组中，近似值是恒定的，它在从一组到下一组时突变。这些近似值接近 $\overline{VE}$ 的全局最优点。
>
> 
>
>
> **Figure 9.1:** Function approximation by state aggregation on the 1000-state random walk task, using the gradient Monte Carlo algorithm (page 202).
>
> 近似值的一些细节最好用这次任务的状态分布 μ 来理解，见于图中的下半部分和一个右侧的标度。中心处的状态500，是每个episode的起始状态，但很少被再次访问。平均地，约有 1.37% 的时间步耗费在起始状态。从起始状态一步可到达的状态是第二多被访问的，约有 0.17% 的时间步耗费在它们的每个上。从这开始 μ 几乎线性下降，在极限状态1和1000时达到约 0.0147%. 分布最直观的效果是在最左边的组和最右边的组上，它们的值分别明显比该组真值的无偏均值要高和低。这是因为这些区域的状态在它们通过 μ 的权重上有最大的不对称性。比如，在最左边的组中，状态100的权重是状态1的 3 倍以上。因此该组的估计会偏向状态100的真值，其要高于状态1的真值。

9.4 线性方法

函数近似的最重要的情形之一是近似函数 $\hat{v}(\cdot, \mathbf{w})$ 是权重向量 $\mathbf{w}$ 的一个线性函数。对于每个状态 s，有一个实值向量 $\mathbf{x}(s) \doteq (x_1(s), x_2(s), \ldots, x_d(s))^T$，和 $\mathbf{w}$ 的分量个数相同。线性方法近似状态-值函数通过 $\mathbf{w}$ 和 $\mathbf{x}(s)$ 之间的内积：

$$\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^T\mathbf{x}(s) \doteq \sum_{i=1}^{d} w_i x_i(s). \tag{9.8}$$

这种情形中近似值函数被称作**线性于权重的** (*linear in the weights*)，或简化为**线性的** (*linear*).

向量 $\mathbf{x}(s)$ 被称为一个**特征向量** (*feature vector*) 来表示状态 s. $\mathbf{x}(s)$ 每个分量 $x_i(s)$ 是函数 $x_i : \mathcal{S} \to \mathbb{R}$ 的值。我们把一个**特征**作为其中一个函数的实体，并称它对状态 s 的值为 s 的一个**特征**。对于线性方法，特征是**基函数**因为它们对近似函数集建立了一个线性基。构造 d 维特征向量来表示状态是等价于选择一个 d 元基函数集。特征可以用很多不同方式定义；我们会在下一节包含一些可能性。

使用带有线性函数近似的SGD更新是自然的。这种情形下，近似值函数对于 $\mathbf{w}$ 的梯度是

$$\nabla \hat{v}(s, \mathbf{w}) = \mathbf{x}(s).$$

由此，在线性情形中，一般SGD更新 (9.7) 缩化成一个特别简单的形式：

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \Big[U_t - \hat{v}(S_t, \mathbf{w}_t) \Big] \mathbf{x}(S_t).$$

因为它太简单了，线性SGD情形是数学分析中最喜爱的之一。对各种学习系统有用的收敛性结果几乎所有都适用于线性 (或更简单的) 函数近似方法。

特别地，在线性情形下只有一个最优点 (或者，在退化情形下，一个同等好的最优点集)，因此所有保证能收敛到或接近一个局部最优的方法，自动能被保证收敛到或接近一个全局最优。比如，前节展示的梯度MC算法在线性函数近似下收敛到 $\overline{VE}$ 的全局最优，若 α 按通用条件随时间递减。

前节展示的半梯度TD(0) 算法在线性函数近似下也会收敛，但它与SGD的一般结果不同；需要单独的理论。权重向量也不是收敛到全局最优，而是一个局部最优附近的点。深入思考这种重要的情形是有用的，尤其是对连续情形。在每个时间 t 的更新是：

$$\begin{aligned} \mathbf{w}_{t+1} &\doteq \mathbf{w} + \alpha \Big(R_{t+1} + \gamma \mathbf{w}_t^T \mathbf{x}_{t+1} - \mathbf{w}^T \mathbf{x}_t \Big) \mathbf{x}_t \\ &= \mathbf{w}_t + \alpha \Big(R_{t+1} \mathbf{x}_t - \mathbf{x}_t (\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^T \mathbf{w}_t \Big), \end{aligned} \tag{9.9}$$

其中我们这里用到缩写记号 $\mathbf{x}_t = \mathbf{x}(S_t)$. 一旦到达稳定状态，对任意给定的 $\mathbf{w}_t$，下个权重向量的期望可以被写成

$$\mathbb{E}[\mathbf{w}_{t+1} | \mathbf{w}_t] = \mathbf{w}_t + \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t), \tag{9.10}$$

其中

$$\mathbf{b} \doteq \mathbb{E}[R_{t+1} \mathbf{x}_t] \in \mathbb{R}^d \quad \textit{and} \quad \mathbf{A} \doteq \mathbb{E}\big[\mathbf{x}_t (\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^T\big] \in \mathbb{R}^{d \times d} \tag{9.11}$$

从 (9.10) 可以清晰看到，若这方程收敛，它一定会收敛到权重向量 $\mathbf{w}_{TD}$ 在

$$\begin{aligned} & \mathbf{b} - \mathbf{A}\mathbf{w}_{TD} = \mathbf{0} \\ \Rightarrow \quad & \mathbf{b} = \mathbf{A}\mathbf{w}_{TD} \\ \Rightarrow \quad & \mathbf{w}_{TD} \doteq \mathbf{A}^{-1}\mathbf{b}. \end{aligned} \tag{9.12}$$

该点称为**TD不动点** (*TD fixed point*). 实际上线性半梯度TD(0) 收敛到这个点。证明它的收敛性和上述逆的存在性的理论的一部分，在下面给出。

> ## 线性$TD(0)$的收敛性证明
>
> 什么性质能确保线性 TD(0) 算法 (9.9) 的收敛性？可以有些领悟通过把 (9.10) 写作
>
> $$\mathbb{E}[\mathbf{w}_{t+1} | \mathbf{w}_t] = (\mathbf{I} - \alpha \mathbf{A})\mathbf{w}_t + \alpha \mathbf{b}. \tag{9.13}$$

注意到矩阵 $\mathbf{A}$ 乘的是权重向量 $\mathbf{w}_t$ 而不是 $\mathbf{b}$；只有 $\mathbf{A}$ 对于收敛性是重要的。为了加强灵感，考虑 $\mathbf{A}$ 是对角矩阵的特殊情形。一方面，如果对角元中任意一个为负数，那么 $\mathbf{I}-\alpha\mathbf{A}$ 对应的对角元会大于 1，进而 $\mathbf{w}_t$ 对应的分量会被放大，若一直持续会导致发散。另一方面，如果 $\mathbf{A}$ 的对角元全是正的，那么 α 可以选得比最大的对角元还小，使得 $\mathbf{I}-\alpha\mathbf{A}$ 是一个对角元全在 0 和 1 之间的对角矩阵。在这种情形下，更新的首项倾向缩小 $\mathbf{w}_t$，从而稳定性被确保。一般地，$\mathbf{w}_t$ 会朝 0 缩减只要 $\mathbf{A}$ 是正定的，也就是 $y^T\mathbf{A}y>0$ 对于任意向量 $y\neq 0$ 成立。正定性也保证逆 $\mathbf{A}^{-1}$ 存在。

对于线性TD(0)，在 $\gamma<1$ 的连续情形中，矩阵 $\mathbf{A}$ 可以被写作

$$\begin{aligned}
\mathbf{A} &= \sum_s \mu(s) \sum_a \pi(a|s) \sum_{r,s'} p(r,s'|s,a)\mathbf{x}(s)\big(\mathbf{x}(s)-\gamma\mathbf{x}(s')\big)^T \\
&= \sum_s \mu(s) \sum_{s'} p(s'|s)\mathbf{x}(s)\big(\mathbf{x}(s)-\gamma\mathbf{x}(s')\big)^T \\
&= \sum_s \mu(s)\mathbf{x}(s)\Big(\mathbf{x}(s)-\gamma\sum_{s'} p(s'|s)\mathbf{x}(s')\Big)^T \\
&= \mathbf{X}^T\mathbf{D}(\mathbf{I}-\gamma\mathbf{P})\mathbf{X},
\end{aligned}$$

其中 $\mu(s)$ 是在 π 下的平稳分布，$p(s'|s)$ 是在 π 下从 s 到 s' 的转移概率，$\mathbf{P}$ 是 $|\mathcal{S}|\times|\mathcal{S}|$ 转移概率矩阵，$\mathbf{D}$ 是 $|\mathcal{S}|\times|\mathbf{S}|$ 对角矩阵，以 $\mu(s)$ 为对角元，$\mathbf{X}$ 是 $|\mathcal{S}|\times d$ 矩阵，每列为 $\mathbf{x}(s)$. 从这清楚地看出内矩阵 $\mathbf{D}(\mathbf{I}-\gamma\mathbf{P})$ 是决定 $\mathbf{A}$ 的正定性的关键。

对这个形式的关键矩阵，正定性被确保若所有它的列和是一个非负数。这被 Sutton (1988, p. 27) 基于两个先前建立的理论所证明。一个理论是任何矩阵 $\mathbf{M}$ 是正定的当且仅当对称矩阵 $\mathbf{S}=\mathbf{M}+\mathbf{M}^T$ 是正定的 (Sutton 1988, 附录). 第二个理论是任何对称实矩阵 S 是正定的，当所有它的对角元是正的且大于对应的所有非对角元的绝对值之和。

$$s_{i,i} > \sum_{j\neq i}(|s_{i,j}|+|s_{j,i}|) \geq 0 \quad \text{for any } i$$

对于我们的关键矩阵 $\mathbf{D}(\mathbf{I}-\gamma\mathbf{P})$，对角元均是正的且非对角元都是负的，所以我们需要证明它的每个行和加上对应的列和是正的。行和都是正的因为 $\mathbf{P}$ 是一个随机矩阵且 $\gamma<1$. 因此只要证明它的列和是非负的。注意到任何矩阵 $\mathbf{M}$ 的列和的行向量可以写成 $\mathbf{1}^T\mathbf{M}$，其中 $\mathbf{1}$ 是所有元为 1 的列向量。令 $\boldsymbol{\mu}$ 表示 $\mu(s)$ 的 $|\mathcal{S}|$-向量，其中 $\boldsymbol{\mu}=\mathbf{P}^T\boldsymbol{\mu}$ 由于 μ 是平稳分布。我们关键矩阵的列行就变成：

$$\begin{aligned}
\mathbf{1}^T\mathbf{D}(\mathbf{I}-\gamma\mathbf{P}) &= \boldsymbol{\mu}^T(\mathbf{I}-\gamma\mathbf{P}) \\
&= \boldsymbol{\mu}^T-\gamma\boldsymbol{\mu}^T\mathbf{P} \\
&= \boldsymbol{\mu}^T-\gamma\boldsymbol{\mu}^T \qquad (\text{because } \mu \text{ is the stationary distrbution}) \\
&= (1-\gamma)\boldsymbol{\mu}^T,
\end{aligned}$$

它的所有分量都是非负的。因此，关键矩阵和它的矩阵 $\mathbf{A}$ 是正定的，在线策略TD(0) 是稳定的。(按概率1收敛需要额外的条件以及一个随时间减小 α 的方案。)

在一个TD不动点上，它也被证明 (在连续情形中) $\overline{VE}$ 是在最小可能误差的一个有界范围内：

$$\overline{VE}(\mathbf{w}_{TD}) \leq \frac{1}{1-\gamma}\min_{\mathbf{w}}\overline{VE}(\mathbf{w}). \tag{9.14}$$

也就是，TD方法的渐进误差是不超过最小可能误差的 $\frac{1}{1-\gamma}$ 倍，其中最小可能误差可以在极限情况下即MC方法中得到。因为 γ 经常接近 1，所以这个扩张因子会变得非常大，所以在TD的渐进性能上有巨大的潜在损失。另一方面，回忆TD方法经常会极大地减小方差相比于MC方法，因此会更快，如我们在章节 6 和 7 所见。哪种方法会最适合取决于近似和问题的特性，以及学习持续多久。

类似于 (9.14) 的边界也适用于其它在线策略拔靴方法。比如，线性半梯度DP (Eq. 9.7 用 $U_t \doteq \sum_a \pi(a|S_t) \sum_{s',r} p(s',r|S_t,a)[r + \gamma \hat{v}(s', \mathbf{w}_t)]$) 用按在线策略分布跟新也会收敛到TD不动点。一步半梯度行动-值(*action-value*)方法，比如下章包含的半梯度 Sarsa(0) 也会收敛到一个类似的不动点和一个类似的边界。对于episodic任务，有一个稍微不同但有关的边界(见 Bertsekas and Tsitsiklis, 1996). 在奖励，特征，和步长参数的递减上也有一些技术性条件，我们在这里忽略。全部的细节可以在原来的论文 (Tsitsiklis and Van Roy, 1997) 中找到。

对收敛性结果关键的是状态按照在线策略分布被更新。对于其它的更新分布，适用函数近似的拔靴方法可能实际上会发散到无穷。这样的例子和可能的解决方案的讨论在章节11给出。

例9.2：在1000状态随机游走上拔靴

状态类聚是线性函数近似的一种特殊情形，所以我们回到1000状态的随机游走来解释本章所做的一些观察。图 9.2 的左半区展示了通过半梯度TD(0) 算法用同例9.1相同的状态类聚来学习的最终值函数。我们看到近渐近式TD近似确实比图 9.1 展示的MC近似更远离真值。

但是，TD方法保持了巨大的潜在优势在学习速率上，而且推广了MC方法，如我们在章节 7 对 n 步TD方法深入探讨的那样。图 9.2 的右半区展示了使用状态类聚的 n 步半梯度TD方法在1000状态随机游走上的结果，这些和我们在之前用列表性方法和19状态随机游走中得到的结果 (图7.2) 惊人地相似。为了得到这样性质上相似的结果，我们把状态类聚变成了20组，每组50个状态。这20组在在性质上接近列表性问题的19个状态。特别地回忆状态转移是到到左或右最多100个状态。一个典型的转移会变成到左或右最多50个状态，这和19状态列表性系统的单状态的状态转移性质上相似。为了完成匹配，我们使用相同的性能衡量——一个在所有状态上的无偏的RMS平均，遍历前10次episodes——而不是一个 $\overline{VE}$ 目标，虽然它在使用函数近似时更加合适。

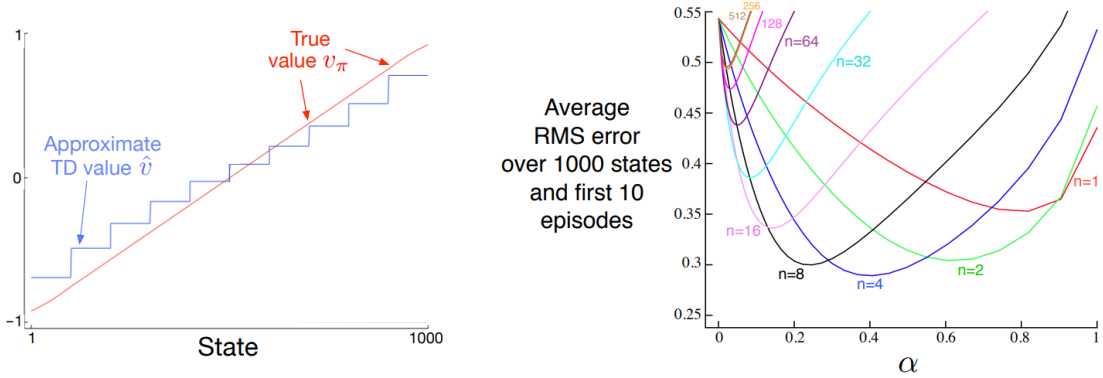


Figure 9.2: Bootstrapping with state aggregation on the 1000-state random walk task. *Left*: Asymptotic values of semi-gradient TD are worse than the asymptotic Monte Carlo values in Figure 9.1. *Right*: Performance of n-step methods with state-aggregation are strikingly similar to those with tabular representations (cf. Figure 7.2). These data are averages over 100 runs.

在示例中适用的半梯度 n 步TD算法是章节7展示的列表性 n 步TD算法到半梯度函数近似的自然拓展。伪代码如下：

n-step semi-gradient TD for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameters: step size $\alpha > 0$, a positive integer n
Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
All store and access operations (S_t and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then $T \leftarrow t+1$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n \hat{v}(S_{\tau+n},\mathbf{w})$ $(G_{\tau:\tau+n})$
  | $\mathbf{w} \leftarrow \mathbf{w} + \alpha\left[G - \hat{v}(S_\tau,\mathbf{w})\right] \nabla \hat{v}(S_\tau,\mathbf{w})$
  Until $\tau = T - 1$

这个算法的关键等式，类似于 (7.2)，是

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha[G_{t:t+n} - \hat{v}(S_t, \mathbf{w}_{t+n-1})]\nabla \hat{v}(S_t, \mathbf{w}_{t+n-1}), \quad 0 \leq t < T, \quad (9.15)$$

其中 n 步回报由 (7.1) 一般化成

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}(S_{t+n}, \mathbf{w}_{t+n-1}), \quad 0 \leq t < T - n. \quad (9.16)$$

> 练习9.1
>
> 证明列表性方法比如本书的第一部分展示的是线性函数近似的一种特殊情形。特征向量应该是什么?

9.5 线性方法的特征构造

线性方法是有趣不仅因为它们的收敛性保证，而且因为实际上它们能在数据和计算上都非常高效。而是不是这样关键依赖于状态是怎样用特征来表示的，也就是我们在这一大节中所探究的。选择适合于任务的特征是添加先验知识到强化学习系统中的重要方式。直观上，特征应该对应于状态空间的一些方面，沿着这些方面泛化可能是恰当的。如果我们对几何物体赋值，比如，我们可能想有每种可能的形状，颜色，尺寸或功能的特征。如果我们正在对一个遥控机器人的状态赋值，那么我们可能想要位置，剩余电量程度，最近的声纳读物等特征。

线性形式的一个局限是它不能考虑到特征之间的交互，比如特征 i 的存在只会在特征 j 不在时是好的。比如说，在平衡杆任务中 (例3.4)，高角速度可以是好的或者坏的，取决于角度。如果角度是高的，那么高角速度意味着会有掉落 (一个坏状态) 的紧迫危险；反之，如果角度是低的，那么高角速度意味着杆在校正自身 (一个好状态). 一个线性值函数不能够表示这个如果它的状态被独立编码成角度和角速度。它而是需要，或者附加的特征来表示这两个状态维度之间的潜在关联。在下面的几个子节中我们会考虑一系列的普遍方式来实现这个。

9.5.1 多项式

很多问题的状态最初被表示成数字，比如杆平衡任务 (例3.4) 中的位置和速度，Jack租车问题 (例4.2) 中的每个车库中的车数，或者在赌徒问题 (例4.3) 中赌徒的资产。在这些类型的问题中，用于强化学习的函数近似和熟悉的插值与回归任务有很多相似之处。各种常用于插值和回归的特征簇也可以被用于强化学习中。多项式构成了用于插值和回归的最简单的特征簇之一。尽管我们这里讨论的基本多项式特征并不能像其它特征一样在强化学习中工作良好，但是它们能作为一个好的引言因为它们是简单且熟悉的。

作为一个例子，假设一个强化学习问题有两个数值维度的状态。对一个单一的代表状态 s，令它的两个数值是 $s_1 \in \mathbb{R}$ 和 $s_2 \in \mathbb{R}$. 你可能会选择表示 s 简单地用它的两个状态维度，也就是 $\mathbf{x}(s) = (s_1, s_2)^T$，但之后你就不能考虑这些维度之间的任何交互了。 另外，如果 s_1 和 s_2 都是零，那么近似值必须也是零。这两个局限都可以通过用一个四维特征向量 $\mathbf{x}(s) = (1, s_1, s_2, s_1 s_2)$ 表示 s 来克服。最初的 1 特征允许在原状态数值上的放射函数表示，而最后的积特征，能够把交互考虑进去。或者你可以选择用更高维的特征向量像 $\mathbf{x}(s) = (1, s_1, s_2, s_1 s_2, s_1^2, s_2^2, s_1 s_2^2, s_1^2 s_2, s_1^2 s_2^2)^T$ 来考虑更加复杂的交互。这样的特征向量能使近似作为状态数值的任意二次函数——即便近似仍然是线性于必须要学的权重的。把这个例子中的 2 个推广到 k 个，我们可以表示问题的状态维度之间的高度复杂的交互：

> 假设每个状态 s 对应于 k 个数字，$s_1, s_2, \ldots, s_k$，其中每个 $s_i \in \mathbb{R}$. 对于这个 k 维状态空间，每个 n 阶的多项式基特征 x_i 能够被写成
>
> $$x_i(s) = \Pi_{j=1}^{k} s_j^{c_{i,j}}, \tag{9.17}$$
>
> 其中每个 $c_{i,j}$ 是在集合 $\{0, 1, \ldots, n\}$ 中的一个整数，对于整数 $n \geq 0$. 这些特征构成了 k 维的 n 阶多项式基，包含 $(n+1)^k$ 个不同特征。

高阶多项式基允许对更复杂的函数有更高精度的近似。但是因为在一个 n 阶多项式基中的特征数会以自然状态空间的维度 k 指数级增长 (如果 $n > 0$)，它通常需要挑选它们的一个子集来进行函数近似。这可以使用关于要被近似的函数的特性的先验知识来完成，而且开发用于多项式回归的一些自动化选择方法也能够被改编用来处理强化学习的递增和非平稳的特性。

> 练习9.2
>
> 为什么 (9.17) 对于维度 k 定义了 $(n+1)^k$ 个不同的状态?

> 练习9.3
>
> 什么样的 n 和 $c_{i,j}$ 生成了特征向量 $\mathbf{x}(s) = (1, s_1, s_2, s_1 s_2, s_1^2, s_2^2, s_1 s_2^2, s_1^2 s_2, s_1^2 s_2^2)^T$?

9.5.2 傅里叶基

另一种线性函数近似方法是基于时域傅里叶级数，它用不同频率的正弦和余弦基函数 (特征) 的加权和来表示周期函数。(一个函数 f 是周期的，如果 $f(x) = f(x+\tau)$ 对所有 x 和某些 τ 成立。) 傅里叶级数和更一般的傅里叶变换在应用科学中被广泛应用，部分原因是，如果要近似的函数已知，那么基函数权重就能通过简单的公式给出，而且，只要有足够的基函数，任何函数都能被近似到任何想要的精度。

在强化学习中，要近似的函数是未知的，傅里叶基函数是有趣的因为它们便于使用而且能在一大片强化学习问题中表现良好。

首先考虑一维情形。周期为 τ 的一维函数的通常傅里叶级数表示将函数表示为正弦和余弦函数的线性组合，这些正弦余弦函数每个都有整除 τ 的周期 (换句话说，它们的频率是一个基础频率 $1/\tau$ 的整数倍). 但如果你对近似一个定义在有界区间上的非周期函数感兴趣，那么你可以用这些傅里叶特征，并把 τ 设置成区间的长度。感兴趣的函数也就变成了正弦和余弦特征的周期线性组合的其中一个周期。

更多地，如果你设置 τ 为兴趣区间长度的两倍，并且限制关注在半区间 $[0, \tau/2]$ 上的近似，那么你只需要用余弦特征。这是可以的因为你可以只用余弦基表示任意的**偶函数**，即任意关于原点轴对称的函数。所以在半周期 $[0, \tau/2]$ 上的任何函数都能用足够的余弦特征近似到任意想要的精度。(说 "任意函数" 是不完全正确的因为函数必须是数学上良定义的，但我们这里忽略这个学术细节。) 或者，也可以只用正弦函数，它的线性组合都是**奇函数**，即关于原点中心对称的函数。但通常最好只保留余弦特征，因为"半偶"函数往往比"半奇"函数更容易近似，因为后者通常在原点不连续。当然，不排除在区间 $[0, \tau/2]$ 上同时使用正弦和余弦特征来近似的可能，在有些情形下这种方式可能更有优势。

在这逻辑下，令 $\tau = 2$ 使得特征是定义在半区间 $[0, 1]$ 上，一维 n 阶傅里叶余弦基包含 $n + 1$ 个特征

$$x_i(s) = \cos(i\pi s), s \in [0, 1], \quad \text{for } i = 0, \ldots, n.$$

图9.3展示了一维傅里叶余弦特征 x_i，对于 $i = 1, 2, 3, 4$；x_0 是一个常值函数。

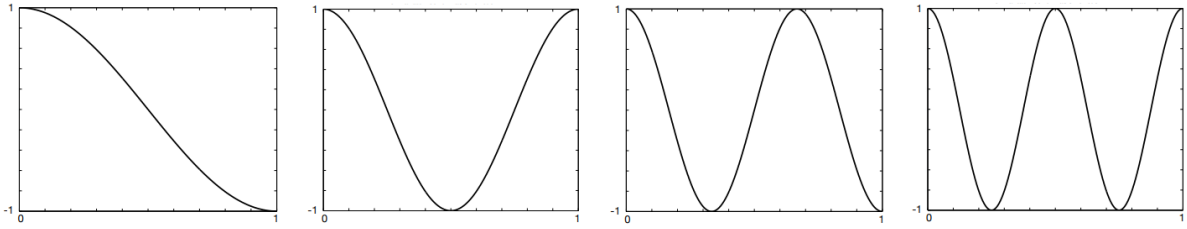


Figure 9.3: One-dimensional Fourier cosine-basis features x_i, $i = 1, 2, 3, 4$, for approximating functions over the interval $[0, 1]$. After Konidaris et al. (2011).

相同的推理适用于在多维情形下的傅里叶余弦级数近似，如下方框所描述的。

> 假设每个状态 s 对应于一个 k 维数值向量，$\mathbf{s} = (s_1, s_2, \ldots, s_k)^T$，其中每个 $s_i \in [0, 1]$. n 阶傅里叶余弦基的第 i 个向量可以被写成
>
> $$x_i(s) = \cos\left(\pi \mathbf{s}^T \mathbf{c}^i\right), \tag{9.18}$$
>
> 其中 $\mathbf{c}^i = (c_1^i, \ldots, c_k^i)^T$，且 $c_j^i \in \{0, \ldots, n\}$ 对每个 $j = 1, \ldots, k$ 和 $i = 1, \ldots, (n+1)^k$. 这对 $(n+1)^k$ 个可能的整数向量 $\mathbf{c}^i$ 中的每一个定义了一个特征。内积 $\mathbf{s}^T \mathbf{c}^i$ 有把 $\{0, \ldots, n\}$ 中的整数分配到 $\mathbf{s}$ 的每个维度上的效果。如在一维情形中，这个整值决定了在该维度上的特征频率。特征当然可以被移动和缩放，以适应特定应用中的有界状态空间。

作为一个例子，考虑 $k = 2$ 情形，其中 $\mathbf{s} = (s_1, s_2)^T$，每个 $\mathbf{c}^i = (c_1^i, c_2^i)^T$. 图9.4展示了一组六个傅里叶余弦特征，每个标签用定义它的向量 $\mathbf{c}^i$ (s_1 是横轴且 $\mathbf{c}^i$ 被表示成去掉上标 i 的行向量). 在 $\mathbf{c}$ 中的任何 0 意味着特征沿该状态维度是常值。所以如果 $\mathbf{c} = (0, 0)^T$，特征在两个维度上都是常数；如果 $\mathbf{c} = (c_1, 0)^T$ 特征在第二个维度上是常数且在第一个维度上按取决于 c_1 的频率变化；类似的有 $\mathbf{c} = (0, c_2)^T$. 当 $\mathbf{c} = (c_1, c_2)^T$ 没有一个 $c_j = 0$，特征在两个维度上都变化，而且表示了两个状态变量之间的一个交互。c_1 和 c_2 的值决定了沿各自维度上的频率，它们的比值给出了交互的方向。

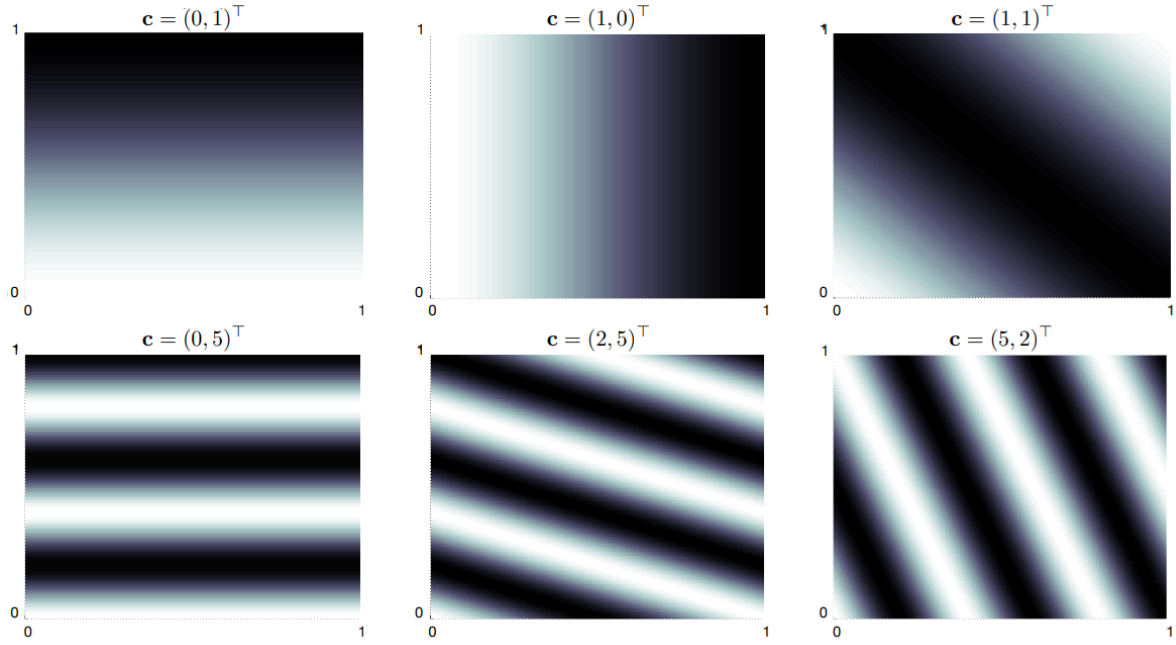


Figure 9.4: A selection of six two-dimensional Fourier cosine features, each labeled by the vector $\mathbf{c}^i$ that defines it (s_1 is the horizontal axis, and $\mathbf{c}^i$ is shown with the index i omitted). After Konidaris et al. (2011).

当傅里叶余弦级数用在一个学习算法中如 (9.7)，半梯度TD(0)，或半梯度Sarsa，对每个特征使用不一样的步长参数可能会有帮助。如果 α 是基本步长参数，那么 Konidaris, Osentoski, 和 Thomas (2011) 建议对特征 x_i 的步长参数设置为 $\alpha_i = \alpha/\sqrt{(c_1^i)^2 + \cdots + (c_k^i)^2}$ (除了当每个 $c_j^i = 0$，此时 $\alpha_i = \alpha$).

傅里叶余弦特征在Sarsa中能产生良好的表现，相比于几个其它的基函数族，包括多项式核径向基函数。不惊讶的是，傅里叶特征在处理非连续性上存在问题，因为它很难避免在非连续点周围的"毛刺"除非包含非常高的频率基函数。

在 n 阶傅里叶基中特征数随特征向量的维度指数级增长，但如果维度足够小 (比如 $k \leq 5$)，那么我们可以选择 n 使得所有 n 阶傅里叶特征能被使用。这使得特征的选择或多或少地自动化。但对于高维状态空间，有必要选择这些特征的一个子集。这可以使用要近似的函数特性的相关先验知识完成，而且一些自动化选择方法能被改变来处理强化学习中渐增核非平稳的特性。在这方面上傅里叶基特征的一个优势是它能简单选择特征通过设置 $\mathbf{c}^i$ 向量来考虑状态变量之间存疑的交互，以及通过限制 $\mathbf{c}^j$ 向量中的值使得近似可以过滤掉被认为是噪音的高频分量。另一方面，因为傅里叶特征是在整个状态空间上非零的(除了少量的几个零点)，它们表示的是状态的全局性质，使得它们很难找到好的方式表示局部性质。

图9.5展示了用来比较傅里叶和多项式基在1000状态随机游走示例上的学习曲线。一般地，我们不推荐在在线学习中使用多项式。

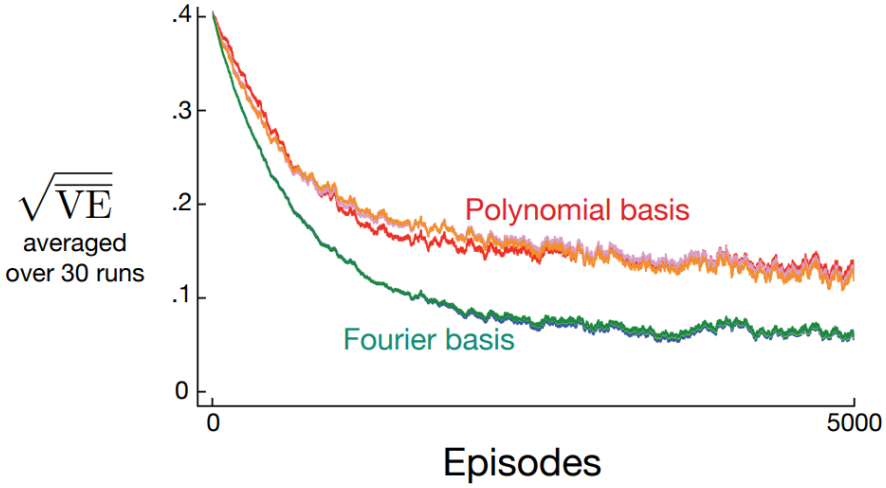


Figure 9.5: Fourier basis vs polynomials on the 1000-state random walk. Shown are learning curves for the gradient Monte Carlo method with Fourier and polynomial bases of order 5, 10, and 20. The step-size parameters were roughly optimized for each case: $\alpha = 0.0001$ for the polynomial basis and $\alpha = 0.00005$ for the Fourier basis. The performance measure (y-axis) is the root mean square value error (9.1).

9.5.3 粗编码

考虑一个任务，其中状态集的自然表示是一个连续的二维空间。对于这种情形一种表示法是由对应于状态空间的圆的特征来组建，如图所示。

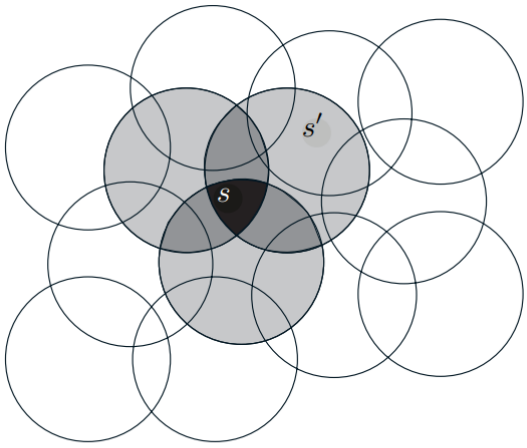


Figure 9.6: Coarse coding. Generalization from state s to state s' depends on the number of their features whose receptive fields (in this case, circles) overlap. These states have one feature in common, so there will be slight generalization between them.

如果状态在一个圆中，那么对应的特征值为1且被称为**在场的** (*present*)；否则特征值为0且被称为**不在的** (*absent*). 这种1-0赋值的特征被称为一个二元特征。给定一个状态，二元特征是在场的表明该状态在对应的圆之中，因此能粗编码它的位置。用这种方式重叠的特征 (尽管它们不需要是圆或二元的) 来表示一个状态被称作是**粗编码** (*coarse coding*).

在线性梯度下降方法近似假设下，考虑圆的尺寸和密度的效果。对应于每个圆的是一个单一权重 (权重 $\mathbf{w}$ 的一个分量) 会通过学习改变。如果我们在一个状态上训练，那么所有包含该状态的圆对应的权重都会被改变。因此，通过 (9.8)，近似值函数会被这些圆的并集中的所有状态所改变，其中一个点与该状态"共同的"圆越多，受到影响越大，如图9.6所示。如果圆都是小的，那么泛化会持续一个短的距离，如图9.7(左) 中所示，反之如果它们是大的，泛化会持续一个大的距离，如图9.7 (中) 所示。更多地，特征

的形状也会决定泛化的特性。比如说，如果它们不是严格圆的，而是在一个方向上被拉长，那么泛化也会被相似地影响，如图9.7 (右) 所示。

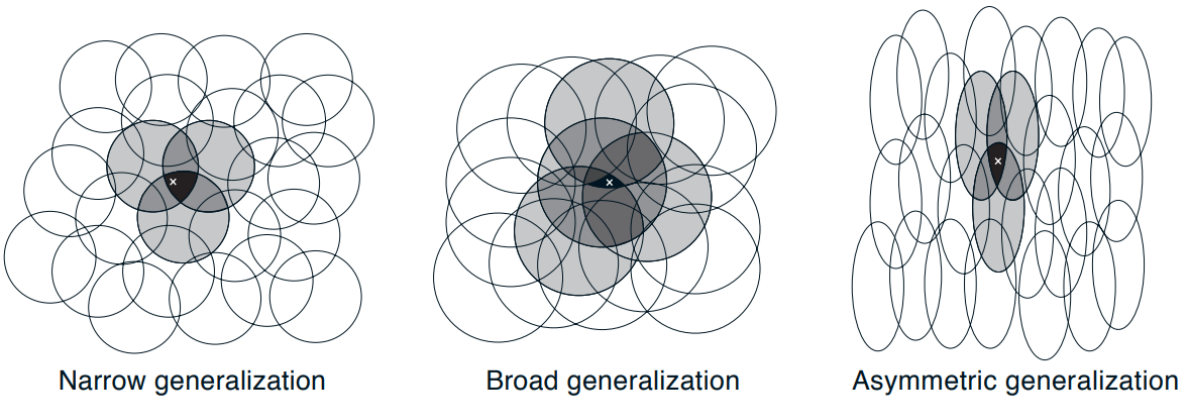


Figure 9.7: Generalization in linear function approximation methods is determined by the sizes and shapes of the features' receptive fields. All three of these cases have roughly the same number and density of features.

有很大感受野的特征会提供广阔的泛化，但也似乎会局限学习函数在一个粗近似上，不能够得到比它的感受野的宽度更好的分辨度。幸运的是，情况不是这样。初始的从一个点到另一点的泛化确实被感受野的尺寸和形状所控制，但敏锐度，最终可能的最好分辨度，更多的是被特征总数所控制的。

例9.3粗编码的粗糙度

这个例子解释了在粗编码中感受野的大小在学习上的影响。线性函数近似基于粗编码以及用 (9.7) 来学习一个一维方波函数 (展示在图9.8的顶部). 这个函数的值被用作目标 U_t. 由于仅有一个维度，感受野是区间而不是圆。学习会被重复在三个不同的区间长度上：窄，中，宽，就如图的底部所示。所有三种情形都有相同的特征密度，比要学习的函数范围大50左右。步长参数是 $\alpha = \frac{0.2}{n}$，其中 n 是在一个时间中在场的特征数 。图9.8展示了在所有三种学习情况下的学习函数。注意到特征的宽度在学习的初期由一个强的影响。用宽特征，泛化倾向于变宽；用窄的特征，只有每个训练点的紧邻会改变，使得学习的函数变得更加坎坷。然而，最终的学习函数仅被特征的宽度轻微地影响。感受野的形状往往对泛化有一个强的影响但对渐进解的质量几乎没有影响。

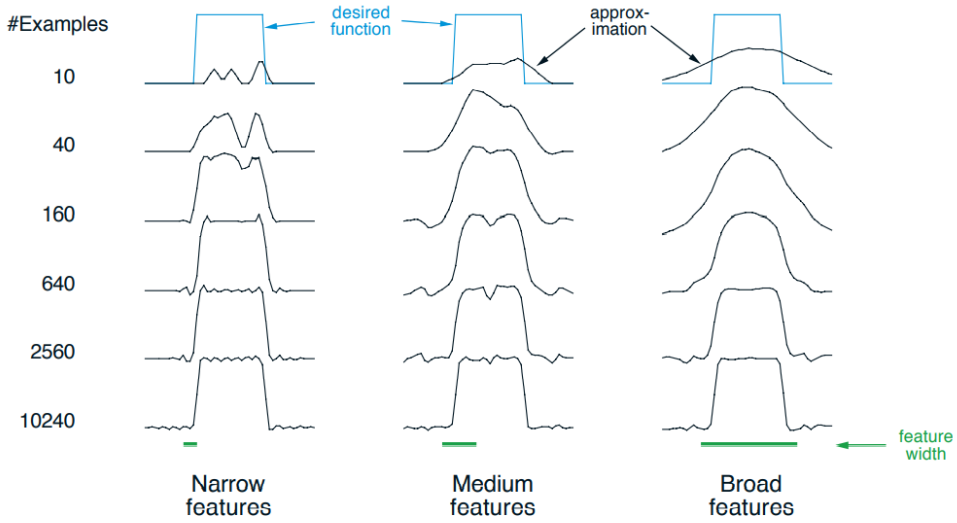


Figure 9.8: Example of feature width's strong effect on initial generalization (first row) and weak effect on asymptotic accuracy (last row). ■

9.5.4 平铺编码

平铺编码是多维连续空间中的一种粗编码形式，具有灵活性和计算效率。它可能是现代序列化数字计算机 (sequential digital computers) 最实用的特征表示。

在平铺编码中特征的感受野被分组成状态空间的分划。每个这样的分划称为一个**平铺** (*tiling*)，分划中的每个成员称为一个**平铺块** (*tile*). 比如，最简单的二维状态空间的平铺是一个均匀网格如图9.9的左侧所示。这里的平铺块或感受野是正方形而不是图9.6中的圆。如果仅使用一个平铺，那么用白点标出的状态将会被它所落在的平铺块的单一特征所表示；泛化将适用于所有在相同的平铺块中的所有状态，而不适用于在它之外的状态。只用一个平铺，我们将得到的不是粗编码而只是一个状态类聚的情形。

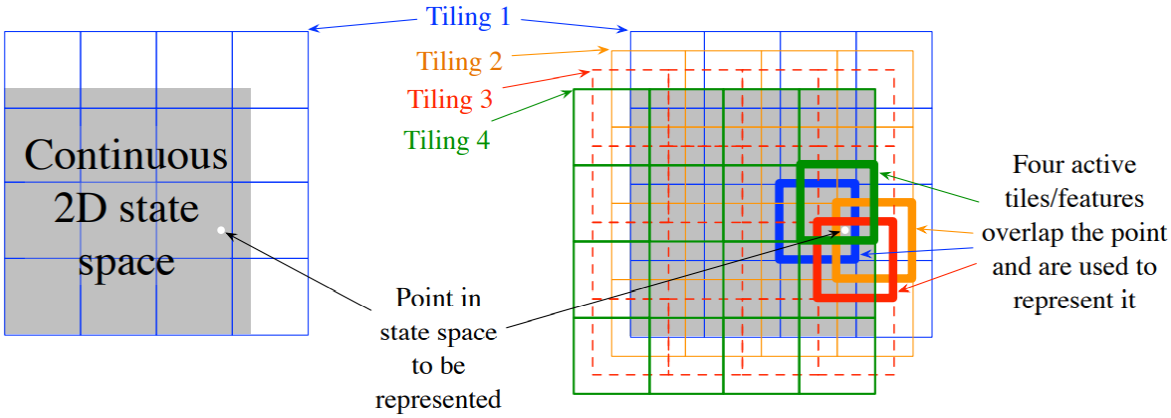


Figure 9.9: Multiple, overlapping grid-tilings on a limited two-dimensional space. These tilings are offset from one another by a uniform amount in each dimension.

为了得到粗编码的优势需要重叠感受野，而由定义一个分划中的平铺块是互不重叠的。为了用平铺编码得到真正的粗编码，需要使用多重平铺，每个偏移一个平铺块的分比。一个四平铺的简单情形展示在图9.9的右侧。每个状态，比如白点所指示的，都恰好落在四个平铺中每个的一个平铺块中。这四个平铺块对应于四个特征，在状态发生时激活。特别地，特征向量 $\mathbf{x}(s)$ 对每个平铺中的每个平铺块有一个分量。在这个例子中有 $4 \times 4 \times 4 = 64$ 个分量，它们所有都是 0 除了四个对应于 s 坐落的平铺块的分量。图9.10展示了多重偏移平铺 (粗编码) 比起单一平铺在1000状态随机游走示例中的优势。

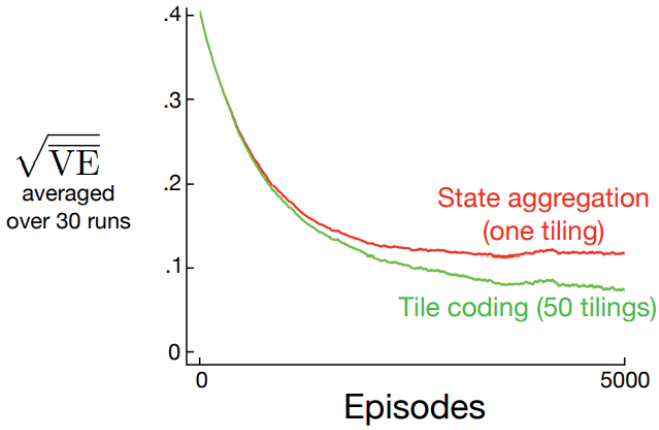


Figure 9.10: Why we use coarse coding. Shown are learning curves on the 1000-state random walk example for the gradient Monte Carlo algorithm with a single tiling and with multiple tilings. The space of 1000 states was treated as a single continuous dimension, covered with tiles each 200 states wide. The multiple tilings were offset from each other by 4 states. The step-size parameter was set so that the initial learning rate in the two cases was the same, $\alpha = 0.0001$ for the single tiling and $\alpha = 0.0001/50$ for the 50 tilings.

平铺编码的一个立刻实用的优势是，因为它用分划来工作，在同一时间对于任何状态活跃的特征的总数是相同的。准确地讲只有一个特征在每个平铺中在场，所以在场的特征总数始终和平铺的数目相同。这使得步长参数 α，能以简单、直观的方式被设置。比如，选 $\alpha = \frac{1}{n}$，其中 n 是平铺的数目，能生成完全的单次试验学习 (one-trial learning). 如果在样例 $s \mapsto v$ 上训练，那么无论先前估计 $\hat{v}(s, \mathbf{w}_t)$ 是什

么，新的估计将会是 $\hat{v}(s, \mathbf{w}_{t+1}) = v$. 通常我们想要比这改变得更慢些，来允许目标输出中存在泛化和随机变量。比如， 我们会选择 $\alpha = \frac{1}{10n}$，这种情形中每次更新训练状态的估计值会朝目标方向的移动1/10，而相邻的状态会被移动的更少，正比于它们之间共同的平铺块的个数。

平铺编码也从它的二元特征向量的使用中得到计算上的优势。因为每个分量是0或者1，构成近似值函数 (9.8) 的权重和计算几乎是平凡的。不是执行 d 次乘法和加法，而是简单地计算 $n \ll d$ 个活跃特征的下标，然后将它们对应的权重向量的分量加起来即可。

泛化会发生在训练状态外的状态上，如果这些状态坐落在任何一个相同的平铺块中，且 (泛化程度) 正比于它们共同的平铺块数目。甚至设置平铺相互之间的偏移的选择也会影响到泛化。如果它们在每个维度一致偏移，如图9.9中所示，那么不同的状态能以不同性质的方式泛化，如图9.11中的上半部分所示。八个子图中的每个展示了从训练状态到邻近点的一种泛化模式。在这个例子中有八个平铺，那么有一个平铺块中有64个子区域能不同地泛化，但所有都按照这八个模式中的一种。注意一致偏移是怎样导致在很多模式中会沿对角线产生很强的影响的。这些伪影能被避免如果平铺用非对称性偏移，如图的下半部所示。下面的这些泛化模式是更好地因为它们都很好地以训练状态为中心没有明显的非对称性。

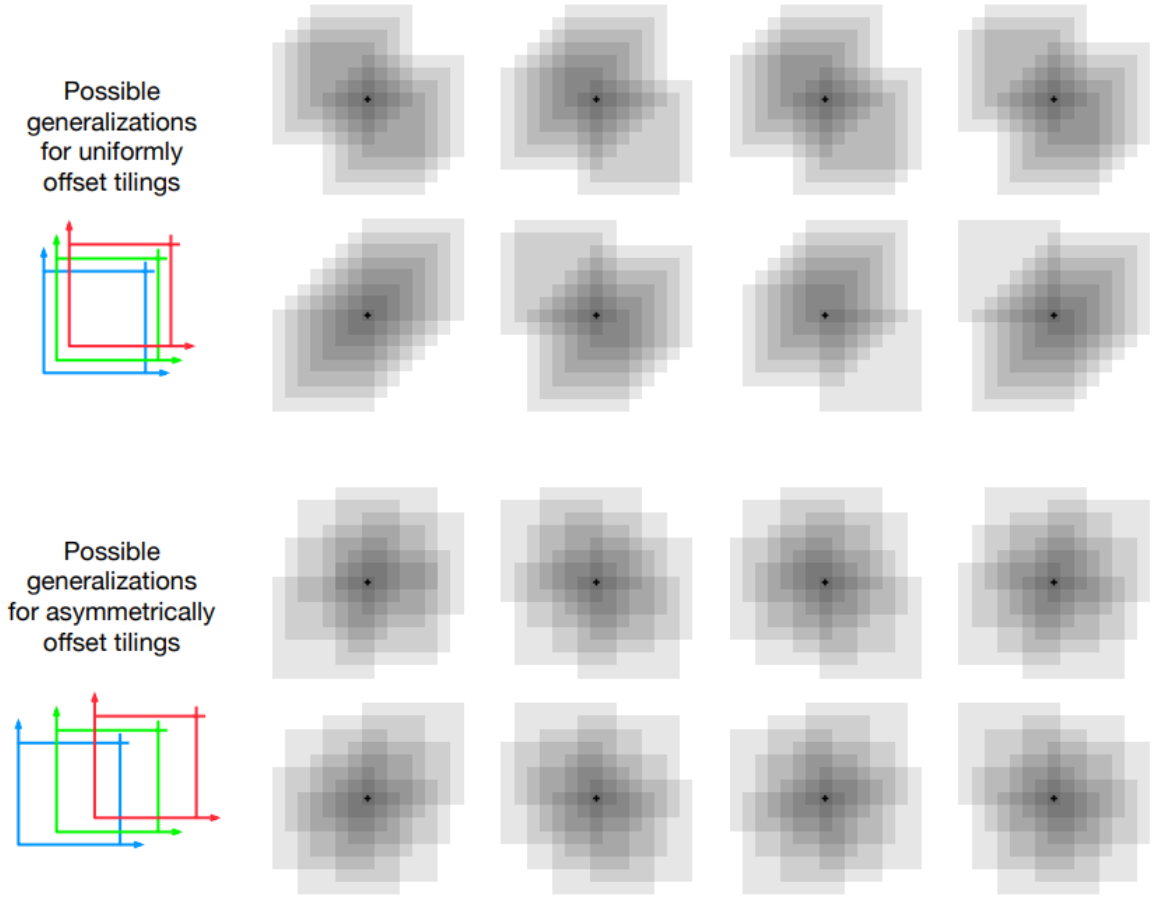


Figure 9.11: Why tile asymmetrical offsets are preferred in tile coding. Shown is the strength of generalization from a trained state, indicated by the small black plus, to nearby states, for the case of eight tilings. If the tilings are uniformly offset (above), then there are diagonal artifacts and substantial variations in the generalization, whereas with asymmetrically offset tilings the generalization is more spherical and homogeneous.

在所有情形中平铺是互相偏移一个平铺块宽度在每个维度上的分比。如果用 w 表示平铺块宽度，n 是平铺数目，那么 $\frac{w}{n}$ 是一个基本单位。在边长为 $\frac{w}{n}$ 的小正方形中，所有状态都活跃在相同的平铺块，有相同的特征表示，以及相同的近似值。如果一个状态在任意笛卡尔方向上按 $\frac{w}{n}$ 移动，那么特征表示也会按一个分量/平铺块进行改变。一致偏移平铺是相互之间偏移全部是这个单位距离。对于一个二维空间，我们说每个平铺相互之间按位移向量 $(1, 1)$ 偏移，意思是它们与之前的平铺的偏移是 $\frac{w}{n}$ 乘以这个向量。在这种形式下，图9.11的下半部分展示的非对称偏移平铺是按一个位移向量 $(1, 3)$ 偏移的。

已经对不同位移向量在平铺编码的泛化上的效果做了广泛的研究 (...)，评估它们的同态性和对角线伪影的趋势，就像在位移向量 $(1,1)$ 中所看到的那些。基于这些工作，Miller 和Glanz (1996) 推荐使用包含最前面的奇数的位移向量。特别地，对于一个 k 维连续空间，一个好的选择是使用最前面的奇数 $(1,3,5,7,\ldots,2k-1)$，和 n (平铺的数目) 设置成一个 2 的整数倍且大于等于 $4k$. 这也是我们生成图9.11的下半部分的平铺所做的，也就是 $k=2, n=2^3 \geq 4k$，且位移向量是 $(1,3)$. 在一个三维情形中，最前面的四个平铺将会从基位置总共位移 $(0,0,0),(1,3,5),(2,6,10),(3,9,15)$. 能够对任意 k 高效生成像这样的平铺的开源软件随时可用。

在选择一个平铺策略时，我们必须选择平铺的数目和平铺块的形状。平铺块的数目，以及平铺块的形状，决定了渐进近似的分辨度 (resolution) 或精细度 (fineness)，就如一般的粗编码和图9.8中解释的那样。平铺块的形状会决定泛化的特性，如图9.7中。 方形平铺块会在每个维度几乎同等地泛化如图9.11 (下) 所表明的那样。平铺块在某个维度上被拉伸，如图9.12 (中)，会促进在该维度上的泛化。在图9.12 (中) 平铺在左边的更密更细，促进沿水平维度上的低值的分辨度。在图9.12 (右) 中的对角带状平铺会促进沿一条对角线的泛化。在高维中，轴向条带 (axis-aligned stripes) 对应于忽略在一些平铺中的一些维度。即，对应于超平面切片 (hyperplanar slices). 不规则平铺比如图9.12 (左) 中展示的也是有可能的，尽管很少用于实践而且超出标准软件的能力。

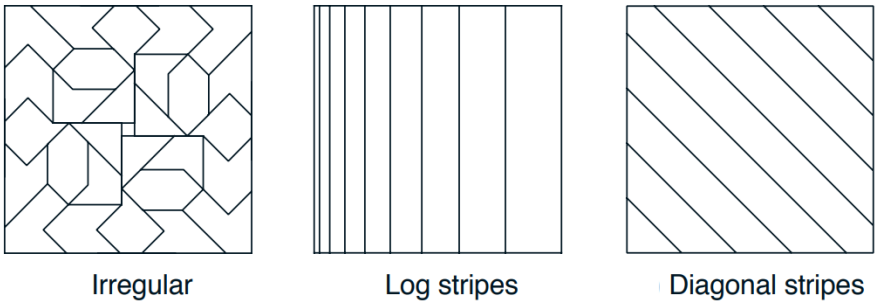
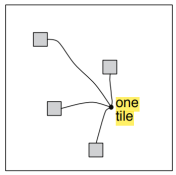


Figure 9.12: Tilings need not be grids. They can be arbitrarily shaped and non-uniform, while still in many cases being computationally efficient to compute.

实际上，通常需要在不同的平铺中使用不同形状的平铺块。比如，我们可能会使用一些垂直条带和一些水平条带。这会鼓励在各自的维度上泛化。然而仅用条带平铺就不能学习水平和垂直坐标的特定组合具有的独特的值 (无论学到什么，都会进入具有相同水平和垂直坐标的状态). 对此我们需要联合矩形平铺块比如原先在图9.9中展示的。用多重平铺——一些水平的，一些垂直的，一些联合的——我们可用得到所有事物：既偏重沿各维度上的泛化，也有能力学习组合的特殊值 (比如见 Sutton, 1996). 平铺的选择决定了泛化，直到这个选择能被高效地自动化，重要的是平铺编码能够使得选择能被灵活地做出，而且以人们有意义的方式。

另一个对减少存储需求有用的技巧是**哈希** (*hashing*)——一个将大的平铺一致伪随机散列成一个小得多的平铺块的集合。哈希生成由不连续，不相交的区域组成的平铺，其中这些区域在整个状态空间上随机分布，但仍然会形成一个彻底的分划。比如，一个平铺块会包含四个子块如下图所示。



通过哈希，内存需求通常可以大幅度减少，而性能损失很少。这是可能的因为只有状态空间的一小部分需要高分辨率。哈希把我们从维数诅咒解放出来意味着内存需求不需要维数的指数级，而是仅仅匹配任务的现实需求。平铺编码的开源实现通常包含高效的哈希。

> 练习9.4
>
> 假设我们相信两个状态维度中的一个比另一个对值函数有更有影响，也就是泛化应该基本跨这个维度而不是沿着它(前者维度). 哪种平铺可以被使用来利用这个先验知识?

9.5.5 径向基函数

径向基函数 (*radial basis functions*, RBFs) 是粗编码向连续值特征的自然拓展。每个特征不再是 0 或者 1，而是可以为区间 $[0,1]$ 中的任何值，反映特征存在的不同程度。一个典型的RBF特征 x_i，有一个高斯响应 $x_i(s)$ 只取决于状态 s 到特征的典范或中心的状态 c_i 之间的距离，以及和特征的宽度 σ_i 有关：

$$x_i(s) \doteq \exp\left(-\frac{||s-c_i||^2}{2\sigma_i^2}\right).$$

标准和距离度量当然可以按任何看起来适合于手头上的状态和任务的方式来选择。下图展示了一个一维的用欧几里得距离度量的示例。

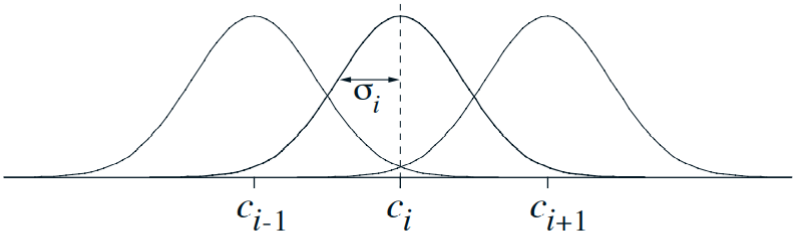


Figure 9.13: One-dimensional radial basis functions.

RBFs比起二元特征最重要的优势是它们生成的近似函数是平滑变化且可微的。尽管这很吸引人，但在大多数情形下它没有实际意义。但是，在平铺编码的背景下已经对分级响应函数比如RBFs进行了广泛的研究。所有这些方法都需要大量的额外计算复杂度(相较于平铺编码)，并且当有两个以上状态维度时经常会降低性能。在高维下，平铺块的边界会变得尤为重要，而且已经被证明很难在边界附近得到良控制的分级平铺块响应。

一个**RBF网络**是一个使用RBFs作为其特征的线性函数近似器。学习通过等式 (9.7) 和 (9.8) 来定义，完全和其它线性函数近似器一样。另外，一些对RBF网络的学习方法也会改变特征的中心和宽度，使得它们进入非线性函数近似器的领域。非线性函数可能能够更精确地匹配目标函数。RBF网络的缺点，尤其在非线性RBF网络中，是巨大的计算复杂度，而且通常更多的学习前人工调节，是具有鲁棒性和高效的。

9.6 手工选择步长参数

大多数SGD方法需要设计者选择一个合适的步长参数α. 理想的是这选择是被自动化的，而在一些情形下这已经做到了，但是对于大多数情形，仍然需要普遍尝试来手工设置它。为了做到这点，并且更好地理解算法，发展一些对步长参数的作用的直观感觉是有用的。我们能一般论述怎么设置它吗?

理论上的考虑不幸是没有帮助的。随机近似理论给我们在一个缓慢递减的步长序列上的条件(2.7)，它能够充分保证收敛，但这往往导致学习过慢。传统的选择是$\alpha_t = 1/t$, 在列表性MC方法中提供样本平均，是不适合TD方法，非稳态问题，或任何用函数近似的方法的。对于线性方法，有回归最小二乘法能够设置一个最优的矩阵步长，而且这些方法能够拓展到TD学习，就像在章节9.8中讲述的LSTD方法，但是这需要$O(d^2)$步长参数，或者我们正在学习的参数的 d 倍。因此我们把它们从函数近似最需要的大问题中排除使用。

为了得到一些直观感觉在如何手工设置步长参数，最好暂时回到可列表情形中。那里我们能够理解一个步长$\alpha = 1$会导致完全忽略在一个目标后的样本误差(见(2.4)中步长是1)。在201页中讨论的那样，我们往往想比这学得更慢。在可列表情形下，一个步长$\alpha = \frac{1}{10}$会取大约10次经历来近似收敛到它们的平均目标，如果我们想要在100次经历学习，我们会取$\alpha = \frac{1}{100}$. 一般地，如果$\alpha = \frac{1}{\tau}$, 那么一个状态的可列表估计，会在大约 τ 次经历这个状态后，到达它的目标的平均值，其中最近的目标有最大的影响。

在一般的函数近似中，没有一个明确的记号表示一个状态的经历数，因为每个状态和其他状态在不同程度上是相似的或者不同的。但是，在线性函数近似情形下，有一个相似的规则能够提供相似的表现。假设你想要在大约 τ 次经历用基本相同的特征向量学习。一个好的经验法则来设置线性SGD方法的步长参数便是

$$\alpha \doteq (\tau \mathbb{E}[\mathbf{x}^T\mathbf{x}])^{-1}, \tag{9.19}$$

其中 $\mathbf{x}$ 是一个随机特征向量，按和SGD中可能的输入向量的相同分布中选出的。这个方法在特征向量的长度改变得不大时能工作地最好；理想的是 $\mathbf{x}^T\mathbf{x}$ 是一个常数。

> 练习9.6
>
> 如果 $\tau = 1$ 且 $\mathbf{x}(S_t)^T\mathbf{x}(S_t) = \mathbb{E}[\mathbf{x}^T\mathbf{x}]$，证明 (9.19) 和 (9.7) 和线性函数近似在一次更新的误差上会减少到 0 。

9.7 非线性函数近似：人工神经网络

人工神经网络 (*artificial neural networks*, ANNs) 在非线性函数近似中广泛被使用。一个ANN是一个互联单元组建的网络，其中这些单元和神经系统中的主元，神经元，有着一些相同的性质。ANNs有悠久的历史，并且在训练深层(deeply-layered)ANNs(深度学习)中的最新成果，能负责包含强化学习系统的机器学习系统的最印象深刻的能力。在章节16中我们会讲述几个强化学习中使用ANN函数近似的印象深刻的示例。

图9.14显示了一个通常的**前馈**(*feedforward*) ANN，意味着网络里面没有循环，也就是网络中没有路径能使一个单元的输出可以影响它的输入。图中的网络有一个包含两个输出单元的输出层，一个包含四个输入单元的输入层，和两个"隐藏层"：既不是输入层也不是输出层的层。每个连线都表示一个实值权重。一个权重近似对应于真实神经网络中的一个突触连接的效力 (见章节15.1)。如果一个ANN在它的连接中至少有一个循环，那么它是一个**循环**(recurrent)ANN而不是前馈ANN。尽管前馈和循环ANNs都已经在强化学习中被使用，这里我们仅考虑更简单的前馈情形。

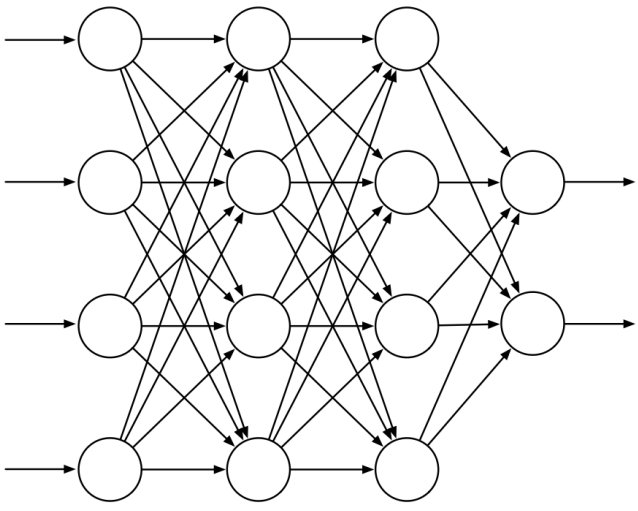


Figure 9.14: A generic feedforward ANN with four input units, two output units, and two hidden layers.

单元(图9.14中的圆)是典型的半线性单元，意味着它们计算它们的输入信号的一个权重和然后应用到一个非线性函数的结果中，其中非线性函数也称为**激活函数**(*activation function*)，来生成单元的输出，或者激活。不同的激活函数在被使用，但它们一般是S形的，或者sigmoid，也就是形如逻辑函数 $f(x) = 1/(1 + e^{-x})$ 的函数，甚至有些时候非线性整流器 $f(x) = max(0, x)$ 也会被使用。一个阶跃函数，形如 $f(x) = 1$ 若 $x \geq \theta$, 否则 0, 会得到一个阈值为 θ 的二元单元。在网络的输入层中的单元的激活有点不同，它们的激活要设置成外部提供值，也就是网络要近似的函数的输入。

一个前馈ANN的每个输出单元的激活是网络的输入单元上的激活模式的一个非线性函数。函数被网络的连接权重参数化。一个ANN没有隐藏层只能表示非常小的一部分可能的输入-输出函数。但是一个ANN只要有一层包含足够多数量的sigmoid单元的隐藏层，就能在网络的输入空间的一个紧区域上，以任意精度近似任何连续函数(Cybenko, 1989)。这也对其它满足弱条件(mild condition)的非线性函数都成立，但函数的非线性是必须的：如果在一个多层前馈ANN中所有单元都是线性激活函数，那整个网络等价于一个没有隐藏层的网络(因为线性函数的线性函数还是线性的)。

尽管单隐藏层ANNs有"通用近似"的性质，经验和理论都证明对很多人工智能需要的复杂函数的近似的简化——确实应该需要——由很多低级抽象按层次组成的抽象，也就是深度结构，比如有很多隐藏层的ANNs，生成的抽象。(更详细的观念见Bengio, 2009.) 深层ANN的连接层(successive layer)要计算网络的"原始"输入的愈加抽象的表示，其每个单元提供一个有助于网络整体的输入-输出函数的层次表示的特征。

训练ANN的隐藏层也就为自动构造合适于给定问题的特征提供了一种方式，使得层次表示的生成能够不单独依赖于手工制造的特征。这本来是人工智能一直以来的挑战，也解释了为什么用隐藏层的ANNs的学习算法多年来一直受到广泛关注。ANN一般用随机梯度方法(章节9.3)学习。每个权重在一个方向上被调整，目的是提高网络整体的性能，性能通过一个要被最大化或最小化的目标函数来衡量。在大多数一般的监督学习情形中，目标函数是在一个标签化的训练集上的期望误差，或者损失。在强化学习中，ANNs能使用TD误差来学习值函数，或者它们能目标最大化期望回报，就像在一个赌博机中(章节2.8)，或者一个策略-梯度算法(章节13). 在所有这些情形中，有必要估计每个连接权重的改变怎么影响网络的整体性能，换句话说，在所有网络权重的当前值下，估计一个目标函数关于每个权重的偏微分。梯度就是这些偏微分的向量。

为有隐藏层 (提供的每个单元有可微的激活函数) 的ANNs做到这点，最成功的方式是**反向传播**(*backpropagation*)算法，它包含了网络中可选的向前和向后的传递。每个前传计算在网络输入单元的当前激活下，每个单元的激活。在每个前传之后，一个后传高效地计算每个权重的偏微分。(就像在其它随机梯度学习算法中，这些偏微分的向量是一个梯度真值的估计。) 在章节15.10中，我们会讨论训练有隐藏层的ANNs的方法，它们使用强化学习准则而不是反向传播。这些方法比起反向传播算法效率要低，但它们可能更接近真实神经网络怎样学习的。

反向传播算法能够为包含 1 或 2 隐藏层的浅层网络产生好的结果，但它可能在更深的ANNs中工作得不好。实际上，训练有 $k + 1$ 隐藏层的网络会导致比训练有 k 隐藏层的网络更差的表现，即便更深的网络能够表示所有浅层网络能表示的函数(Bengio, 2009)。这样的结果解释起来不简单，但有几个因素是重要的。第一，一个普通的深层ANN中的大数量的权重，会很难避免对问题的**过度匹配**(overfitting), 也就是，不能正确概括网络没有被训练的情形的问题。第二，反向传播不能良作用于深度ANNs，因为用它的后传计算的偏微分，要么会向网络的输入方向迅速衰减，使得深层的学习极度缓慢，要么会向网络的输入方向迅速增走，使得学习不稳定。使用深层ANNs的系统最近取得的很多引人注目的成果，离不开能处理这些问题的方法。

过度匹配是一个问题，对于任何函数近似方法，基于有限的训练数据，用大量的自由度调整函数。它在在线强化学习中可能会少一些，因为它不依赖于有限的训练集，但高效地泛化仍然是一个重要的问题。一般过度匹配是对ANNs的一个问题，但尤其对深度ANNs，因为它们倾向于有非常大数目的权重。很多为减少过度匹配的方法已经被开发。这包括：当在不同于训练数据的检验数据(交叉检验)上的性能开始减少时停止训练；改善目标函数抑制近似(正则化)的复杂度；引入权重之间的依赖度来减少自由度的数目(比如，权重共享)。

一个特别有效的方法来减少深度ANNs产生的过度匹配，是 ***dropout*** 方法，由...提出(2014). 在学习时，随机从网络中移除单元(dropped out)以及它们的连接。这可以被想成训练一个大数目的"精简(thinned)"网络。将这些精简的网络的结果在测试时间联合是一种能提高泛化表现的方式。dropout方法通过将一个单元的每个传出(outgoing)权重乘这个单元在训练中留存的概率来高效地近似这个联合。Srivastava et al. 发现这个方法能显著提高泛化性能。它鼓励单独的隐藏单元能学习能够和其他特征的集合中工作良好的特征，能够在随机其它特征的组合中工作好的。这增加了由隐藏单元建立的特征的多样性，使得网络不会过度特化到很少发生的情形。

Hinton, Osindero 和 Tec(2006) 在解决一个深度ANN的深层训练的问题中做出了主要的一步，在他们的工作中用了深度信赖网络，分层网络(与这里讨论的深度ANNs是紧密相关的)。在它们的方法中，最深层每次用一个非监督学习算法训练一层。不依赖于整体的目标函数，非监督学习能够提取捕获了从输入流的统计规律的特征。最深层最先被训练，然后由训练层提供的输入，第二个最深的层被训练，以此类推，直到所有，或者大多数网络层的权重都已经设置好值，这时能够作为监督学习的初始值。之后网络在整体的目标函数意义下，能通过反向传递来被良调整。研究表明这个方法比随机初始化权重的反向传递能工作地好很多。以这种方式初始化权重训练的网络有更好的性能可能由于多个因素，但一个想法是这个方法把网络放置到一个能够使得基于梯度算法取得好的成果的权重空间的区域。

批量标准化(Ioffe and Szegedy, 2015) 是另外一个技巧，能够使得训练深度ANNs变得简单。很早就知道ANN学习会更简单，如果网络的输入被标准化，比如，通过调整每个输入变量有零期望和单位方差。批量标准化对于训练深度ANNs，将深层的输出标准化在它们被送给下一层之前。Ioffe and Szegedy(2015) 使用从训练样例中的子集，或"小批量(mini-batches)"得到的统计数据来规范化这些层间信号，从而提高深度ANNs的学习速率。

另一个对训练深度ANNs的有用的技巧是**深度残差学习**(*deep residual learning*). 有时候学习一个函数如何不同于恒等函数比学习函数本身更简单。然后把这个差异，或者残差函数，加到输入中生成想要的函数。在深度ANNs中，一个层区块可以被用来学习一个残差函数，简单地通过短接或跳过块周围的连接。这些连接把区块的输入加到它的输出，不需要加额外的权重。He et al.(2016) 评估这种方法通过使用深度卷积网络并跳过每对邻接层周围的连接，发现比起没有用跳过连接的网络在图像检测分类任务中有重大提升。批量标准化和深度残差学习都有使用，在16章会讲述的强化学习在Go游戏中的应用中。

一种深度ANN的类型，已经在包含很多引人注目的强化学习应用(章节16)的应用中被证明很成功，它是**深度卷积网络**(*deep convolutional network*). 这种类型的网络专门处理以空间矩阵排列的高维数据，比如图像。它启发于早期视觉处理在大脑中的工作方式(...)。凭借它的特殊结构，一个深层卷积网络能够被反向传播训练，而不需要诉诸上面讲述的那些方法来训练深层。

图9.15显示了一个深层卷积网络的结构。这个示例，来源于LeCun et al. (1998), 是设计成分辨手写字母。它包含交替的**卷积** (*convolutional*) 层和**子抽样** (*subsampling*) 层，随后是几个最后的**全连接** (*fully connected*) 层。每个卷积层产生很多**特征图** (*feature map*). 一个特征图是一个在一组单元上的行为模式，其中每个单元在它的**感受野** (*receptive field*) 中的数据上执行相同的操作，感受野是从先前层(或者对于第一个卷积层是从额外的输入)中能"看到"的部分数据。一个特征图的单元之间是一样的，除了它们的感受野，感受野有相同的大小和形状，会随着输入数据阵列上移动到不同的位置。在相同的特征图中单元共享相同的权重。这意味着一个特征图都会检测到相同的特征，无论它位于输入阵列何处。在图9.15的网络中，比如，第一个卷积层生成了6个特征图，每个包含 28×28 个单元。每个特征图中的每个单元有一个 5×5 的感受野，而且这些感受野会重叠(这里是四行和四列会重叠)。因此，6 个特征图中的每一个被仅 25 个可调整的权重描述。

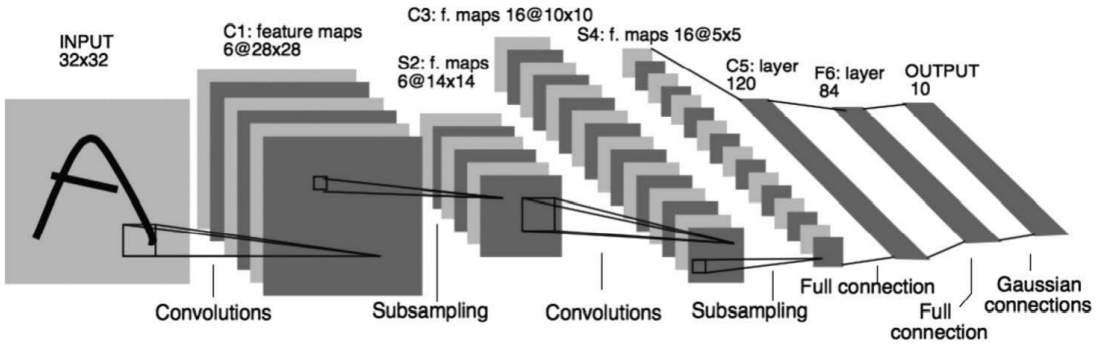


Figure 9.15: Deep Convolutional Network. Republished with permission of Proceedings of the IEEE, from Gradient-based learning applied to document recognition, LeCun, Bottou, Bengio, and Haffner, volume 86, 1998; permission conveyed through Copyright Clearance Center, Inc.

一个深度卷积网络的子抽样层减少特征图的空间分辨率。在一个子抽样层中，每个特征图由先前卷积层的特征图中的单元的感受野的均值的单元组成。比如，6个特征图中的每一个的每个单元在图9.15的网络中的第一个子抽样层中，在第一个卷积层中生成的特征图中之一的一个 2×2 的不重叠感受野进行平均，生成 6 个 14×14 特征图。子抽样层减少了在特征检测的空间位置中的网络灵敏度，也就是，这有助于使得网络响应是空间不变的。这是有用的因为一个在图像的一个地方检测的特征很可能在其它地方也有用。

在设计和训练ANNs的最新进展中——我们只提及了一小部分——都有助于强化学习。尽管现在的强化学习理论大多局限在使用可列表或线性函数近似的方法中，著名的强化学习应用的引人注目的表现要把成功归功于通过多层ANNs的非线性函数近似。我们将会在章节16中讨论这些应用中的一些。

9.8 最小二乘 TD

目前为止，我们在本章讨论的所有方法都需要在每时间步长正比于参数数量的计算量。如果有更多的计算量，这能做得更好。在这一节中我们会展示一个对线性近似的方法，可被证明是在这种情形下能做得最好的。

就像我们在章节 9.4 确认的，线性函数近似的TD(0)渐进地 (通过恰当地减小步长) 收敛到TD不动点:

$$\mathbf{w}_{TD} = \mathbf{A}^{-1}\mathbf{b},$$

其中

$$\mathbf{A} \doteq \mathbb{E}[\mathbf{x}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^T] \quad and \quad \mathbf{b} \doteq \mathbb{E}[R_{t+1}\mathbf{x}_t].$$

可能会问，为什么我们必须要迭代计算这个解？这是浪费数据！我们难道不能通过计算 $\mathbf{A}$ 和 $\mathbf{b}$ 的估计，然后直接计算TD不动点来做得更好吗？**最小二乘TD算法**(*Least-Squares TD algorithm*)，通常记作 ***LSTD*** ，完全是这么做的。它形成了自然估计

$$\widehat{\mathbf{A}}_t \doteq \sum_{k=0}^{t-1} \mathbf{x}_k(\mathbf{x}_k - \gamma\mathbf{x}_{k+1})^T + \varepsilon\mathbf{I} \quad and \quad \widehat{\mathbf{b}}_t \doteq \sum_{k=0}^{t-1} R_{k+1}\mathbf{x}_k, \tag{9.20}$$

其中 $\mathbf{I}$ 是单位矩阵，而 $\varepsilon\mathbf{I}$, 取小的 $\varepsilon > 0$, 保证 $\widehat{\mathbf{A}}_t$ 始终可逆。这看起来可能这些估计都需要除以 t ，而且它们确实需要；正如这里定义的，这些估计的确是 $\mathbf{A}$ 的 t 倍和 $\mathbf{b}$ 的 t 倍。然而，额外的因数 t 会抵消掉，当LSTD使用这些估计来估计TD不动点:

$$\mathbf{w}_t \doteq \widehat{\mathbf{A}}_t^{-1}\widehat{\mathbf{b}}_t. \tag{9.21}$$

这个算法是线性TD(0)的最高效使用数据的形式，但它计算开支也更大。回想一下半梯度TD(0)需要的内存和每步计算量仅是 $O(d)$.

LSTD 有多复杂？正如上面所写的，复杂度似乎随 t 增加，然而在 (9.20) 中的两个近似能够自增表示，在使用我们之前介绍的技巧(比如， 在章节2) ，这使得它们每步能够在常数时间内完成。即便如此，对 $\widehat{\mathbf{A}}_t$ 的更新会包含一个外积(一个行向量乘一个列向量)，因此会变成一个矩阵更新；它的计算复杂度会是 $O(d^2)$, 当然需要用来存放矩阵 $\widehat{\mathbf{A}}_t$ 的内存也会是 $O(d^2)$.

一个潜在地更严重的问题是我们的最终计算 (9.21) 使用了 $\widehat{\mathbf{A}}_t$ 的逆，而一个一般的逆计算的计算复杂度是 $O(d^3)$.幸运的是，在我们的特殊形式下的矩阵的逆——一个外积和——能够被自增更新仅用 $O(d^2)$ 的计算量，就如

$$\begin{aligned}\widehat{\mathbf{A}}_t^{-1} &= \left(\widehat{\mathbf{A}}_{t-1} + \mathbf{x}_{t-1}(\mathbf{x}_{t-1} - \gamma\mathbf{x}_t)^T\right)^{-1} && \text{(from (9.20))} \\ &= \widehat{\mathbf{A}}_{t-1}^{-1} - \frac{\mathbf{A}_{t-1}^{-1}\mathbf{x}_{t-1}(\mathbf{x}_{t-1} - \gamma\mathbf{x}_t)^T\widehat{\mathbf{A}}_{t-1}^{-1}}{1 + (\mathbf{x}_{t-1} - \gamma\mathbf{x}_t)^T\widehat{\mathbf{A}}_{t-1}^{-1}\mathbf{x}_{t-1}}, && (9.22)\end{aligned}$$

对于 $t > 0$, 且 $\widehat{\mathbf{A}}_0 \doteq \varepsilon\mathbf{I}$. 尽管等式 (9.22), 被称为 ***Sherman-Morrison*** 公式，表面上看起来是复杂的，它仅包含向量-矩阵和向量-向量乘法，因此仅有 $O(d^2)$. 因此我们可以存储逆矩阵 $\widehat{\mathbf{A}}_t^{-1}$, 用 (9.22) 控制它，然后在 (9.21) 中使用它，所有仅需要 $O(d^2)$ 的存储和每步计算量。整个算法在下面的方框中给出。

LSTD for estimating $\hat{v} = \mathbf{w}^\top\mathbf{x}(\cdot) \approx v_\pi$ ($O(d^2)$ version)

Input: feature representation $\mathbf{x} : \mathcal{S}^+ \to \mathbb{R}^d$ such that $\mathbf{x}(terminal) = \mathbf{0}$
Algorithm parameter: small $\varepsilon > 0$

$\widehat{\mathbf{A}^{-1}} \leftarrow \varepsilon^{-1}\mathbf{I}$ — A $d \times d$ matrix
$\widehat{\mathbf{b}} \leftarrow \mathbf{0}$ — A d-dimensional vector
Loop for each episode:
  Initialize S; $\mathbf{x} \leftarrow \mathbf{x}(S)$
  Loop for each step of episode:
    Choose and take action $A \sim \pi(\cdot|S)$, observe R, S'; $\mathbf{x}' \leftarrow \mathbf{x}(S')$
    $\mathbf{v} \leftarrow \widehat{\mathbf{A}^{-1}}^\top(\mathbf{x} - \gamma\mathbf{x}')$
    $\widehat{\mathbf{A}^{-1}} \leftarrow \widehat{\mathbf{A}^{-1}} - \left(\widehat{\mathbf{A}^{-1}}\mathbf{x}\right)\mathbf{v}^\top / \left(1 + \mathbf{v}^\top\mathbf{x}\right)$
    $\widehat{\mathbf{b}} \leftarrow \widehat{\mathbf{b}} + R\mathbf{x}$
    $\mathbf{w} \leftarrow \widehat{\mathbf{A}^{-1}}\widehat{\mathbf{b}}$
    $S \leftarrow S'$; $\mathbf{x} \leftarrow \mathbf{x}'$
  until S' is terminal

当然，$O(d^2)$ 仍然比半梯度TD的 $O(d)$ 要显著高开支。LSTD的更良好的数据效率是不是值得这个计算开支，取决于 d 有多大，它学得快的重要性，以及系统其它部分的开支。事实上，LSTD不需要步长参数有时候也被吹捧，但这个优势很可能被夸大了。LSTD确实不需要一个步长，但它需要 ε; 如果 ε 被选得过小，逆序列可能会变化失控，如果 ε 选得过大，那么学习就会减慢。另外，LSTD缺少步长参数意味着它从不会遗忘。这有时候是需要的，但当目标策略 π 改变，就像它在强化学习和GPI中那样时，这反而是有问题的。在控制应用中，LSTD一般必须联合一些其它手段来引发遗忘，以体现不需要步长参数的初始优势。

9.9 基于记忆的函数近似

直到现在我们已经讨论了参数化方式来近似值函数。在这种方式下，一个学习算法调整一个函数形式的参数，为了在问题的整个状态空间上来近似值函数。每次更新，$s \mapsto g$, 是一个训练样例，它使用学习算法来改变参数，目的是减少近似误差。在更新后，训练样例可以被丢弃 (尽管它可能会被保存以再次使用) 。当一个状态 (称为**查询状态**(*query state*)) 的近似值被需要时，使用学习算法生成的最新的参数，在该状态下简单评估函数。

基于记忆 (*memory-based*) 的函数近似是很不一样的。它们简单地保存训练样例在记忆中 (或者至少保存样例的子集) 而不修改任何参数，在它们到达这些训练样例时。然后，无论何时一个查询状态的值估计被需要时，会从记忆中取出一组样例，然后被用来计算该查询状态的值估计。这种方式有时候称为**懒惰学习** (*lazy learning*), 因为加工训练样例被延迟直到系统被查询以提供一个输出。

基于记忆的函数近似方法是**非参数** (*nonparametric*) 方法的典型例子。不同于参数化方法，近似函数的形式没有被限制在一个固定的参数化的函数类中，比如线性函数或者多项式，而是取决于训练样例本身，以及一些把它们结合来输出查询状态的估计值的方法。随着更多的训练样例在记忆中积累，我们期待非参数方法能对任何目标函数生成越来越精确的近似。

有很多不同的基于记忆的方法。这取决于怎么存储寻选择的训练样例和它们怎么被使用来响应一个查询。这里，我们聚焦于**局部学习** (*local-learning*) 方法，它只在当前的查询状态的邻域中局部近似一个值函数。这些方法从记忆中取出一组训练样例，它们的状态被判定为和查询状态最相关的，其中相关性通常取决于状态之间的距离：一个训练样例的状态离查询状态越近，它就被认为越相关，其中距离能用很多不同方式来定义。在给查询状态值后，局部近似会被丢弃。

基于记忆的方法的最简单的示例是**最邻近** (*nearest neighbor*) 方法，它简单地在记忆中寻找其状态最近于查询状态的样例，然后返回该样例的值作为查询状态的近似值。换句话说，如果查询状态是 s, 那么 $s' \mapsto g$ 是在记忆中的样例，其中 s' 是最近于 s 的状态，而 g 被返回作为 s 的近似值。稍微有点复杂的是**权重平均** (*weight average*) 方法，它们取出一组最近的相邻样例然后返回一个它们目标值的加权平均，其中权重一般随它们的状态到查询状态的距离增大而减小。**局部加权回归** (*locally weighted regression*) 是类似的，但是它把曲面拟合到一组最近状态通过最小化权重误差度量的参数化近似方法比如 (9.1), 得到的值，其中权重取决于到查询状态的距离。返回值是在查询状态上的局部拟合曲面的估计，之后局部近似曲面会被丢弃。

作为非参数的，基于记忆方法有超过参数化方法的优势，它不需要限制近似到预定义的函数形式。这允许当更多数据积累时提高精确度。基于记忆的局部 (*local)* 近似方法有其它性质使得它们能更适合强化学习。因为强化学习中轨迹采样 (*trajectory sampling*) 是相当重要的，就像在章节8.6中讨论的，基于记忆的局部方法能够聚焦函数近似，在实际或仿真轨迹访问的状态 (或状态-行动对) 的局部邻域上。可能不需要全局近似，因为状态空间的很多区域会从不 (或几乎从不) 被到达。并且，基于近似方法允许代理者的经验有一个相对即刻的效果在当前状态的邻域中的值估计上，与此相对，一个参数化方法需要逐渐调整全局近似的参数。

避免全局近似也是一种解决维数诅咒的方式。比如，对于一个有 k 维的状态空间，一个列表化方法存储一个全局近似需要 k 的指数级的空间，另一方面，在存储样例到一个基于记忆方法中，每个样例需要正比于 k 的空间，而且需要存储的空间是线性于 n ，即 n 个样例。哪个都不需要 k 或 n 的指数级。当然，关键的存在问题是一个基于记忆方法能不能回应查询足够快以至对代理者有用。一个相关的担心是当记忆规模增长时速度是如何衰减的。在一个大数据库中寻找最近的邻点会耗时过长以致不能实用在很多应用中。

基于记忆方法的支持者已经开发出加速最近邻搜索的方式。使用并行计算机或特定目的的硬件是一种方式；另一种是使用特殊的多维数据结构来存储训练数据。为这种应用研究出的一种数据结构是 k **维树** (k-d tree) (k-*dimensional tree* 的缩写), 它递归将一个 k 维空间拆分成作为二叉树节点排列的区域。 根据数据的数量和它在状态空间上的分布方式，最近邻搜索使用 k-d树能够在搜索邻点中很快消除空间中大区域，使得搜索在原本简单搜索耗时过长的问题中变得可行。

局部权重回归还需要快速的方式来完成局部回归计算，这些计算需要不断重复来回应每次查询。研究者已经开发出很多方式来解决这些问题，包括遗忘实体为了保持数据库的有界规模的方法。在本章最后的 Bibliographic and Historical Comments 小节中会提出一些相关的文献，包含一些讲述了强化学习中基于记忆的应用的论文。

9.10 基于核函数近似

基于记忆方法比如上面讲述的权重平均和局部权重回归方法，取决于根据 s' 到查询状态 s 的距离来分配权重到数据库中的示例 $s' \mapsto g$. 分配这些权重的函数称为**核函数** (*kernel function*)，或简单的**核** (*kernel*). 在权重平均核局部权重回归方法中，比如，一个核函数 $k : \mathbb{R} \to \mathbb{R}$ 分配权重到状态之间的距离。更一般地，权重不一定要依赖于距离；它们可以依赖于一些其它状态之间的相似性的度量。在这种情形下， $k : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$, 使得 $k(s, s')$ 是由 s' 的数据对响应 s 的查询的影响来给出的权重。

稍微从不同的角度来看， $k(s, s')$ 是一个从 s' 到 s 的泛化强度的度量。核函数数值化地表现了任何状态的知识与任何其它状态的**相关** (*relevant*) 程度。 举个例子，对于图9.11 展示的平铺编码的泛化强度对应于从一致且不对称的平铺偏移生成的不同核函数。尽管平铺编码没有在它的操作中显式地使用一个核函数，但是它是根据一个核函数进行泛化的。实际上，如我们在更下面讨论的，由线性参数化函数近似得到的泛化强度总能够描述成一个核函数。

核回归 (*kernel regression*) 是基于记忆方法，它计算所有存储在记忆中的样例目标的一个核权重平均，把结果赋值给查询状态。如果 $\mathcal{D}$ 是存储样例的集合， $g(s')$ 表示在一个存储样例中状态 s' 的目标，那么核回归近似到目标函数，在这种情形下是一个依赖于 $\mathcal{D}$ 的值函数, 如

$$\hat{v}(s, \mathcal{D}) = \sum_{s' \in \mathcal{D}} k(s, s') g(s'). \tag{9.23}$$

上面讲述的权重平均方法是一个特殊情形，其中 $k(s, s')$ 是非零的仅当 s 和 s' 是互相邻近的，使得和不需要在整个 $\mathcal{D}$ 上计算。

一个通用核是 ***Gaussian radial basis function*** (*RBF*)， 在章节9.5.5中讲述的RBF函数近似中有使用。那里讲的方法中，RBFs是特征，其中心和宽度要么在开始时固定，中心集中在期待有很多样例进入的区域，要么在学习中以某些方式进行调整。不考虑调整中心和宽度的方法，这是一种线性参数化方法，它的参数是每个RBD的权重，通常通过随机梯度下降，或者半梯度下降来学习。这种近似的形式是一个预定义的RBFs的线性组合。用一个RBF核的核回归则不同于这，体现在两个方面。第一，它是基于记忆的：RBFs是以存储样例的状态为中心。第二，它是非参数的：没有参数需要学习；一个查询的响应由 (9.23) 给出。

当然，核回归的实际实现不得不要解决很多问题，而这些问题超出我们简要讨论的范围。然而，事实证明，任何线性参数回归方法就像在章节9.4讲述的那些，其状态通过特征向量 $\mathbf{x}(s) = \big(x_1(s), x_2(s), \ldots, x_d(s)\big)^T$ 表示，能够重新转换为核回归，其中 $k(s, s')$ 是 s 和 s' 表示的特征向量的内积；即

$$k(s, s') = \mathbf{x}(s)^T \mathbf{x}(s'). \tag{9.24}$$

用这核函数的核回归会生成和一个线性参数方法相同的近似，如果它使用相同的特征向量且用相同的训练数据学习。

我们跳过这个的数学证明，它可以在任何现代机器学习文章中找到，比如 Bishop (2006), 然后简单指出一个重要的意义。不用为线性参数函数近似器构造特征，而是可以直接构造核函数，不需要考虑特征向量。不是所有的核函数都能够像 (9.24) 那样表示成特征向量的内积，但能被这样表示的核函数比起等价的参数化方法有显著的优势。对于很多特征向量集， (9.24) 有一个紧致的函数形式，使得它不需要在 d 维特征空间中进行计算就能被估计。在这些情形中，核回归比直接使用一个状态用这些特征向量表示的线性参数化方法复杂度要显著降低。这也称为"*kernel trick*"， 也就是允许在高维的拓展特征空间中有效工作，而实际上只用存储的训练样例工作。核技巧是很多机器学习方法的基础，研究者已经证明了在有些时候它是怎样有利于强化学习的。

9.11 深入研究在线策略学习：兴趣和重要性

直到现在我们在本章已经考虑的算法是平等对待所有遇到的状态，即认为它们同等重要。在一些情形下，然而，我们对一些状态相对于其它状态更感兴趣。在折扣 episodic 问题中，比如，我么可能对需要精确计算的在 episode 中的早期状态更感兴趣，比起在后面的状态，折扣会使得它们的回报对于起始状态的值明显不重要。或者，如果一个行动-值函数正在被学习，那么精确评价那些值明显小于贪婪行动的表现差的行动就不那么重要。函数近似资源总是有限的，如果它能以更有目标性的方式被使用，那么性能就能被提升。

我们把遇到的所有状态同等对待的一个原因是我们按照在线策略分布更新，对于这种分布，半梯度方法可以得到更强的理论结果。回忆一下，在线策略分布是按遵循目标策略生成的MDP中遇到的状态的分布来定义的。现在我们将对这一概念进行重要概括。对MDP，不只有一个在线策略分布，而是有很多。它们所有都有一个共同点，即它们都是遵循目标策略中生成的轨迹中遇到的状态的分布，但它们的轨迹在启动上在某种意义上会不一样。

我们现在介绍一些新的概念。首先我们介绍一个非负标量度量，一个随机变量 I_t 称作**兴趣** (*interest*)，表明我们在时间 t 对精确评价状态 (或状态-行动对) 的感兴趣程度。如果我们一点都不关心这个状态，那么兴趣应该是 0 ；如果我们完全关心，他应该是 1 ，尽管形式上我们可以取任何非负值。兴趣可以被设置成任何有因果的方式；举个例子，它可以依赖于直到时间 t 的轨迹或者在时间 t 的学习参数。在 $\overline{VE}$ (9.1) 中的分布 μ 就被定义成遵循目标策略时遇到的状态的分布，被兴趣赋权重。其次，我们介绍另一个非负标量随机变量，**重要性** (*emphasis*) M_t. 这个标量乘以学习的更新，以此强调或者不强调在时间 t 完成的学习。一般的 n 步学习规则，取代 (9.15) ，是

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha M_t [G_{t:t+n} - \hat{v}(S_t, \mathbf{w}_{t+n-1})] \nabla \hat{v}(S_t, \mathbf{w}_{t+n-1}), \quad 0 \leq t < T, \quad (9.25)$$

用到 (9.16) 给出的 n 步回报，并且重要性由兴趣递归确定，通过：

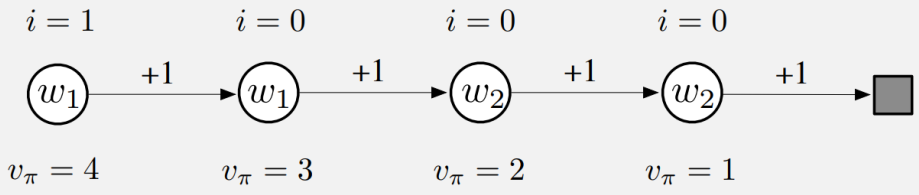
$$M_t = I_t + \gamma^n M_{t-n}, \quad 0 \leq t < T, \quad (9.26)$$

其中 $M_t \doteq 0$ ，对所有 $t < 0$. 这些等式也可以用来包括蒙特卡洛情形，其中 $G_{t:t+n} = G_t$ ，所有的更新在episode的最后进行， $n = T - t$ ，和 $M_t = I_t$.

例9.4 会解释兴趣和重要性可以如何生成更精确的值估计。

例9.4：兴趣和重要性

为了发现使用兴趣和重要性的潜在好处，考虑一个四状态的马尔可夫回报过程，如下：



Episodes 从最左边的状态开始，然后从一个状态转移到右边的状态，在每一步有一个 $+1$ 的报酬，直到到达终止状态。由此，第一个状态的真值是4，第二个状态是3，以此类推，图中状态的下方有标出。这些是真值；估计值只能近似到这些因为它们受到参数化的限制。参数向量 $\mathbf{w} = (w_1, w_2)^T$ 有两个分量，参数化如每个状态内部所写的那样。前两个状态的估计值被 w_1 单独给出，因此必须是相同的，即便它们的真值是不同的。同样的，第三个和第四个状态的估计值被 w_2 单独给出，也必须是相同的即便它们的真值是不一样的。假定我们只对精确评价最左边的状态感兴趣；我们赋给它一个 1 的兴趣，而其它的所有状态的兴趣被赋给一个 0 的兴趣，如图的上方所示。

> 首先考虑对这个问题应用蒙特卡罗算法。本章先前展示的算法并没有考虑兴趣和重要性 (在 (9.7) 和 页202的方框中) 会收敛到 (用递减的步长) 参数向量 $\mathbf{w}_\infty = (3.5, 1.5)$，给第一个状态——我们仅感兴趣的一个——一个 3.5 的值 (即，第一个和第二个状态的真值的中间值). 本章节展示的方法确实使用了兴趣和重要性，另一方面，会完全正确地学习第一个状态的值； w_1 会收敛到 4 而 w_2 不会被更新因为除了最左边的状态以外所有状态的重要性是0。
>
> 现在考虑应用 2 步半梯度TD方法。本章先前的方法没有用兴趣和重要性 (在 (9.15) 和 (9.16) 和 页209的方框中) 会再次收敛到 $\mathbf{w}_\infty = (3.5, 1.5)$，而用兴趣和重要性的方法会收敛到 $\mathbf{w}_\infty = (4, 2)$. 后者生成了对于第一个和第三个 (第一个状态拔靴来源) 状态完全正确的值，但不会对于第二个和第四个状态作任何更新。

9.12 小结

强化学习系统必须具有泛化能力，如果它们要应用到人工智能或者大型工程问题。为了实现这点，一个宽泛的现有的监督学习函数近似方法中的任何一种都能简单地通过把每次更新作为一个训练样例来使用。

也许最合适的监督学习方法是那些使用参数化函数近似的，也就是策略被一个权重向量 $\mathbf{w}$ 参数化。尽管权重向量有很多分量，但状态空间仍要比这个大得多，我们必须满足于一个近似解。我们定义**均方值误差** (*mean square value error*)，$\overline{VE}(\mathbf{w})$，为在**在线策略分布** (*on-policy distribution*), μ 下，权重向量 $\mathbf{w}$ 的值 $v_{\pi_\mathbf{w}}(s)$ 的误差的一个度量。$\overline{VE}$ 给我们一个清晰的方式来对在线策略情形下不同的值函数近似进行评级。

为了找到一个好的权重向量，最主流的方法是**随机梯度下降** (*stochastic gradient descent* (SGD)) 的变体。在本章中我们聚焦于用固定策略的在线策略情形，也称作策略评价或预测；对这个情形一个自然的学习算法是 n **步半梯度TD** (n-*step semi-gradient TD*)，包含梯度蒙特卡罗和半梯度TD(0) 算法，分别作为 $n = \infty$ 和 $n = 1$ 下的特例。半梯度TD方法不是真正的梯度方法。在类似拔靴方法中 (包括DP)，权重向量在更新目标中出现，却没有在计算梯度时考虑进去——因此它们是**半梯度**方法。因此，它们不能依赖传统的SGD结果。

但是，在线性函数近似的特殊情形下，即其值估计是特征乘相应权重的和，半梯度方法能得到好的结果。线性情形是理论上最好理解的，而且实际上在提供恰当地特征时能工作得很好。选择特征是加入先验知识 (*prior domain knowledge*) 到强化学习系统中的最重要方式之一。它们可以被选成**多项式**，但这种情形在强化学习通常考虑的在线学习环境下泛化很差。更好地是选按照**傅里叶基**，或者按照一些有稀疏的重叠感受野的**粗编码**的形式得到的特征。**平铺编码** (*tile coding*) 是粗编码的一种形式，它在计算上特别高效灵活。**径向基函数** (*radial basis function*) 在一维或二维下一个平滑变化的响应是重要的任务中是有用的。**LSTD** 是数据效率最高的线性TD预测方法，但是需要正比于权重数量的平方的计算量，而所有其它方法的复杂度是权重数量的线性级。非线性函数近似包括用**反向传播**和 SGD 的变体训练的**人工神经网络**；这些方法在最近非常主流，也就是深度强化学习 (*deep reinforcement learning*).

线性半梯度 n 步TD在标准条件下保证收敛到在最优误差范围内的 $\overline{VE}$ (MC方法则能渐进地达到)，对于所有的 n. 这个边界对更高的 n 总会更紧，当 $n \to \infty$ 时到达 0. 但是，实际上很高的 n 会导致学习很慢，而且一定程度的拔靴 ($n < \infty$) 通常更推荐，就像我们在章节7列表性 n 步方法的比较，以及章节6列表性TD和MC方法的比较中看到的那样。

> 练习9.7
>
> ---
>
> 一种最简单的人工神经网络包含单一半近似单元用 logistic nonlinearity. 用这种形式处理近似值函数的需求普遍见于最后会赢或输的游戏中，在这种情形中状态的值可以被解释成赢得概率。从 (9.7) 推导这种情形的学习算法，使得没有梯度符号出现。
>
> ---

练习9.8

按理说，用来推导 (9.7) 的平方误差不适合前面练习中处理的情形，正确的误差度量是交叉熵损失 (cross-entropy loss, 可以在Wikipedia上找到). 使用交叉熵损失而不是 (9.4) 中的平方误差，重复在章节9.3的推导，直到没有梯度或对数符号的显式形式。你的最终形式是比你之前练习中得到的更复杂还是更简单?