第七章　深度学习中的正则化

机器学习中的一个核心问题是设计不仅在训练数据上表现好，并且能在新输入上泛化好的算法。在机器学习中，许多策略显式地被设计来减少测试误差（可能会以增大训练误差为代价）。这些策略被统称为正则化。我们将在后文看到，深度学习工作者可以使用许多不同形式的正则化策略。事实上，开发更有效的正则化策略已成为本领域的主要研究工作之一。

第五章介绍了泛化、欠拟合、过拟合、偏差、方差和正则化的基本概念。如果你不熟悉这些概念，请参考该章节再继续阅读本章。

在本章中，我们会更详细地介绍正则化，重点介绍深度模型（或组成深度模型的模块）的正则化策略。

本章中的某些章节涉及机器学习中的标准概念。如果你已经熟悉了这些概念，可以随意跳过相关章节。然而，本章的大多数内容是关于这些基本概念在特定神经网络中的扩展概念。

在第 5.2.2 节中，我们将正则化定义为“对学习算法的修改——旨在减少泛化误差而不是训练误差”。目前有许多正则化策略。有些策略向机器学习模型添加限制参数值的额外约束。有些策略向目标函数增加额外项来对参数值进行软约束。如果我们细心选择，这些额外的约束和惩罚可以改善模型在测试集上的表现。有时侯，这些约束和惩罚被设计为编码特定类型的先验知识；其他时候，这些约束和惩罚被设计为偏好简单模型，以便提高泛化能力。有时，惩罚和约束对于确定欠定的问题是必要的。其他形式的正则化，如被称为集成的方法，则结合多个假说来解释训练数据。

在深度学习的背景下，大多数正则化策略都会对估计进行正则化。估计的正则化以偏差的增加换取方差的减少。一个有效的正则化是有利的“交易”，也就是能显著减少方差而不过度增加偏差。我们在第五章中讨论泛化和过拟合时，主要侧重模

型族训练的 3 个情形：（1）不包括真实的数据生成过程——对应欠拟合和含有偏差的情况，（2）匹配真实数据生成过程，（3）除了包括真实的数据生成过程，还包括许多其他可能的生成过程——方差（而不是偏差）主导的过拟合。正则化的目标是使模型从第三种情况转化为第二种情况。

在实践中，过于复杂的模型族不一定包括目标函数或真实数据生成过程，甚至也不包括近似过程。我们几乎从未知晓真实数据的生成过程，所以我们永远不知道被估计的模型族是否包括生成过程。然而，深度学习算法的大多数应用都是针对这样的情况，其中真实数据的生成过程几乎肯定在模型族之外。深度学习算法通常应用于极为复杂的领域，如图像、音频序列和文本，本质上这些领域的真实生成过程涉及模拟整个宇宙。从某种程度上说，我们总是持方枘（数据生成过程）而欲内圆凿（我们的模型族）。

这意味着控制模型的复杂度不是找到合适规模的模型（带有正确的参数个数）这样一个简单的事情。相反，我们可能会发现，或者说在实际的深度学习场景中我们几乎总是会发现，最好的拟合模型（从最小化泛化误差的意义上）是一个适当正则化的大型模型。

现在我们回顾几种策略，以创建这些正则化的大型深度模型。

7.1 参数范数惩罚

正则化在深度学习的出现前就已经被使用了数十年。线性模型，如线性回归和逻辑回归可以使用简单、直接、有效的正则化策略。

许多正则化方法通过对目标函数 J 添加一个参数范数惩罚 $\Omega(\boldsymbol{\theta})$，限制模型（如神经网络、线性回归或逻辑回归）的学习能力。我们将正则化后的目标函数记为 $\tilde{J}$：

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}), \tag{7.1}$$

其中 $\alpha \in [0, \infty)$ 是权衡范数惩罚项 Ω 和标准目标函数 $J(\boldsymbol{X}; \boldsymbol{\theta})$ 相对贡献的超参数。将 α 设为 0 表示没有正则化。α 越大，对应正则化惩罚越大。

当我们的训练算法最小化正则化后的目标函数 $\tilde{J}$ 时，它会降低原始目标 J 关于训练数据的误差并同时减小在某些衡量标准下参数 $\boldsymbol{\theta}$（或参数子集）的规模。选择不同的参数范数 Ω 会偏好不同的解。在本节中，我们会讨论各种范数惩罚对模型的

影响。

在探究不同范数的正则化表现之前，我们需要说明一下，在神经网络中，参数包括每一层仿射变换的权重和偏置，我们通常只对权重做惩罚而不对偏置做正则惩罚。精确拟合偏置所需的数据通常比拟合权重少得多。每个权重会指定两个变量如何相互作用。我们需要在各种条件下观察这两个变量才能良好地拟合权重。而每个偏置仅控制一个单变量。这意味着，我们不对其进行正则化也不会导致太大的方差。另外，正则化偏置参数可能会导致明显的欠拟合。因此，我们使用向量 $\boldsymbol{w}$ 表示所有应受范数惩罚影响的权重，而向量 $\boldsymbol{\theta}$ 表示所有参数 (包括 $\boldsymbol{w}$ 和无需正则化的参数)。

在神经网络的情况下，有时希望对网络的每一层使用单独的惩罚，并分配不同的 α 系数。寻找合适的多个超参数的代价很大，因此为了减少搜索空间，我们会在所有层使用相同的权重衰减。

7.1.1 L^2 参数正则化

在第 5.2 节中我们已经看到过最简单而又最常见的参数范数惩罚，即通常被称为**权重衰减**（weight decay）的 L^2 参数范数惩罚。这个正则化策略通过向目标函数添加一个正则项 $\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$，使权重更加接近原点[1]。在其他学术圈，$L^2$ 也被称为岭回归或 Tikhonov 正则。

我们可以通过研究正则化后目标函数的梯度，洞察一些权重衰减的正则化表现。为了简单起见，我们假定其中没有偏置参数，因此 $\boldsymbol{\theta}$ 就是 $\boldsymbol{w}$。这样一个模型具有以下总的目标函数：

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}), \tag{7.2}$$

与之对应的梯度为

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}). \tag{7.3}$$

使用单步梯度下降更新权重，即执行以下更新：

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon(\alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})). \tag{7.4}$$

[1]更一般地，我们可以将参数正则化为接近空间中的任意特定点，令人惊讶的是这样也仍有正则化效果，但是特定点越接近真实值结果越好。当我们不知道正确的值应该是正还是负时，零是有意义的默认值。由于模型参数正则化为零的情况更为常见，我们将只探讨这种特殊情况。

换种写法就是：

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}). \tag{7.5}$$

我们可以看到，加入权重衰减后会引起学习规则的修改，即在每步执行通常的梯度更新之前先收缩权重向量（将权重向量乘以一个常数因子）。这是单个步骤发生的变化。但是，在训练的整个过程会发生什么呢?

我们进一步简化分析，令 $\boldsymbol{w}^*$ 为未正则化的目标函数取得最小训练误差时的权重向量，即 $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$，并在 $\boldsymbol{w}^*$ 的邻域对目标函数做二次近似。如果目标函数确实是二次的 (如以均方误差拟合线性回归模型的情况)，则该近似是完美的。近似的 $\hat{J}(\boldsymbol{\theta})$ 如下

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*), \tag{7.6}$$

其中 $\boldsymbol{H}$ 是 J 在 $\boldsymbol{w}^*$ 处计算的 Hessian 矩阵 (关于 $\boldsymbol{w}$)。因为 $\boldsymbol{w}^*$ 被定义为最优，即梯度消失为 0，所以该二次近似中没有一阶项。同样地，因为 $\boldsymbol{w}^*$ 是 J 的一个最优点，我们可以得出 $\boldsymbol{H}$ 是半正定的结论。

当 $\hat{J}$ 取得最小时，其梯度

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) \tag{7.7}$$

为 0。

为了研究权重衰减带来的影响，我们在式 (7.7) 中添加权重衰减的梯度。现在我们探讨最小化正则化后的 $\hat{J}$。我们使用变量 $\tilde{\boldsymbol{w}}$ 表示此时的最优点:

$$\alpha\tilde{\boldsymbol{w}} + \boldsymbol{H}(\tilde{\boldsymbol{w}} - \boldsymbol{w}^*) = 0 \tag{7.8}$$

$$(\boldsymbol{H} + \alpha\boldsymbol{I})\tilde{\boldsymbol{w}} = \boldsymbol{H}\boldsymbol{w}^* \tag{7.9}$$

$$\tilde{\boldsymbol{w}} = (\boldsymbol{H} + \alpha\boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{w}^* \tag{7.10}$$

当 α 趋向于 0 时，正则化的解 $\tilde{\boldsymbol{w}}$ 会趋向 $\boldsymbol{w}^*$。那么当 α 增加时会发生什么呢? 因为 $\boldsymbol{H}$ 是实对称的，所以我们可以将其分解为一个对角矩阵 $\boldsymbol{\Lambda}$ 和一组特征向量的标准正交基 $\boldsymbol{Q}$，并且有 $\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top$。将其应用于式 (7.10)，可得：

$$\tilde{\boldsymbol{w}} = (\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top + \alpha\boldsymbol{I})^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^* \tag{7.11}$$

$$= [\boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})\boldsymbol{Q}^\top]^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^* \tag{7.12}$$

$$= \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^*. \tag{7.13}$$

我们可以看到权重衰减的效果是沿着由 $\boldsymbol{H}$ 的特征向量所定义的轴缩放 $\boldsymbol{w}^*$。具体来说，我们会根据 $\frac{\lambda_i}{\lambda_i+\alpha}$ 因子缩放与 $\boldsymbol{H}$ 第 i 个特征向量对齐的 $\boldsymbol{w}^*$ 的分量。（不妨查看图 2.3 回顾这种缩放的原理）。

沿着 $\boldsymbol{H}$ 特征值较大的方向 (如 $\lambda_i \gg \alpha$)正则化的影响较小。而 $\lambda_i \ll \alpha$ 的分量将会收缩到几乎为零。这种效应如图 7.1 所示。

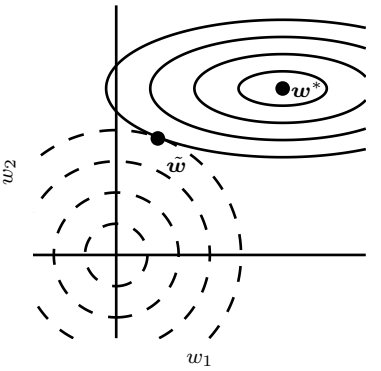

图 7.1: L^2（或权重衰减）正则化对最佳 $\boldsymbol{w}$ 值的影响。实线椭圆表示没有正则化目标的等值线。虚线圆圈表示 L^2 正则化项的等值线。在 $\tilde{\boldsymbol{w}}$ 点，这两个竞争目标达到平衡。目标函数 J 的 Hessian 的第一维特征值很小。当从 $\boldsymbol{w}^*$ 水平移动时，目标函数不会增加得太多。因为目标函数对这个方向没有强烈的偏好，所以正则化项对该轴具有强烈的影响。正则化项将 w_1 拉向零。而目标函数对沿着第二维远离 $\boldsymbol{w}^*$ 的移动非常敏感。对应的特征值较大，表示高曲率。因此，权重衰减对 w_2 的位置影响相对较小。

只有在显著减小目标函数方向上的参数会保留得相对完好。在无助于目标函数减小的方向（对应 Hessian 矩阵较小的特征值）上改变参数不会显著增加梯度。这种不重要方向对应的分量会在训练过程中因正则化而衰减掉。

目前为止，我们讨论了权重衰减对优化一个抽象通用的二次代价函数的影响。这些影响具体是怎么和机器学习关联的呢？我们可以研究线性回归，它的真实代价函数是二次的，因此我们可以使用相同的方法分析。再次应用分析，我们会在这种情况下得到相同的结果，但这次我们使用训练数据的术语表述。线性回归的代价函数是平方误差之和：

$$(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y})^\top(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y}). \tag{7.14}$$

我们添加 L^2 正则项后，目标函数变为

$$(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y})^\top(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y})+\frac{1}{2}\alpha\boldsymbol{w}^\top\boldsymbol{w}. \tag{7.15}$$

这将普通方程的解从

$$\boldsymbol{w}=(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} \tag{7.16}$$

变为

$$\boldsymbol{w}=(\boldsymbol{X}^\top\boldsymbol{X}+\alpha\boldsymbol{I})^{-1}\boldsymbol{X}^\top\boldsymbol{y}. \tag{7.17}$$

式 (7.16) 中的矩阵 $\boldsymbol{X}^\top\boldsymbol{X}$ 与协方差矩阵 $\frac{1}{m}\boldsymbol{X}^\top\boldsymbol{X}$ 成正比。L^2 正则项将这个矩阵替换为式 (7.17) 中的 $(\boldsymbol{X}^\top\boldsymbol{X}+\alpha\boldsymbol{I})^{-1}$ 这个新矩阵与原来的是一样的，不同的仅仅是在对角加了 α。这个矩阵的对角项对应每个输入特征的方差。我们可以看到，L^2正则化能让学习算法 “感知” 到具有较高方差的输入 $\boldsymbol{x}$，因此与输出目标的协方差较小（相对增加方差）的特征的权重将会收缩。

7.1.2 L^1 参数正则化

L^2权重衰减是权重衰减最常见的形式，我们还可以使用其他的方法限制模型参数的规模。一个选择是使用 L^1正则化。

形式地，对模型参数 $\boldsymbol{w}$ 的 L^1正则化被定义为：

$$\Omega(\boldsymbol{\theta})=\|\boldsymbol{w}\|_1=\sum_i|w_i|, \tag{7.18}$$

即各个参数的绝对值之和[2]。接着我们将讨论 L^1正则化对简单线性回归模型的影响，与分析 L^2正则化时一样不考虑偏置参数。我们尤其感兴趣的是找出 L^1 和 L^2正则化之间的差异。与 L^2权重衰减类似，我们也可以通过缩放惩罚项 Ω 的正超参数 α 来控制 L^1权重衰减的强度。因此，正则化的目标函数 $\tilde{J}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y})$ 如下所示

$$\tilde{J}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y})=\alpha\|\boldsymbol{w}\|_1+J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}), \tag{7.19}$$

对应的梯度 (实际上是次梯度)：

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y})=\alpha\text{sign}(\boldsymbol{w})+\nabla_{\boldsymbol{w}}J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}), \tag{7.20}$$

[2]如同 L^2正则化，我们能将参数正则化到其他非零值 $\boldsymbol{w}^{(o)}$。在这种情况下，L^1正则化将会引入不同的项 $\Omega(\boldsymbol{\theta})=\|\boldsymbol{w}-\boldsymbol{w}^{(o)}\|_1=\sum_i|w_i-w_i^{(o)}|$。

其中 sign($\boldsymbol{w}$) 只是简单地取 $\boldsymbol{w}$ 各个元素的正负号。

观察式 (7.20)，我们立刻发现 L^1 的正则化效果与 L^2 大不一样。具体来说，我们可以看到正则化对梯度的影响不再是线性地缩放每个 w_i；而是添加了一项与 sign(w_i) 同号的常数。使用这种形式的梯度之后，我们不一定能得到 $J(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{w})$ 二次近似的直接算术解（L^2正则化时可以）。

简单线性模型具有二次代价函数，我们可以通过泰勒级数表示。或者我们可以设想，这是逼近更复杂模型的代价函数的截断泰勒级数。在这个设定下，梯度由下式给出

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*), \tag{7.21}$$

同样，$\boldsymbol{H}$ 是 J 在 $\boldsymbol{w}^*$ 处的Hessian矩阵 (关于 $\boldsymbol{w}$)。

由于 L^1 惩罚项在完全一般化的 Hessian 的情况下，无法得到直接清晰的代数表达式，因此我们将进一步简化假设 Hessian 是对角的，即 $\boldsymbol{H} = \mathrm{diag}([H_{1,1}, \ldots, H_{n,n}])$，其中每个 $H_{i,i} > 0$。如果线性回归问题中的数据已被预处理（如可以使用 PCA），去除了输入特征之间的相关性，那么这一假设成立。

我们可以将 L^1正则化目标函数的二次近似分解成关于参数的求和：

$$\hat{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}^*; \boldsymbol{X}, \boldsymbol{y}) + \sum_i \left[\frac{1}{2} H_{i,i} (w_i - w_i^*)^2 + \alpha |w_i| \right]. \tag{7.22}$$

如下列形式的解析解（对每一维 i）可以最小化这个近似代价函数：

$$w_i = \mathrm{sign}(w_i^*) \max \left\{ |w_i^*| - \frac{\alpha}{H_{i,i}}, 0 \right\}. \tag{7.23}$$

对每个 i, 考虑 $w_i^* > 0$ 的情形，会有两种可能结果：

1. $w_i^* \leq \frac{\alpha}{H_{i,i}}$ 的情况。正则化后目标中的 w_i 最优值是 $w_i = 0$。这是因为在方向 i 上 $J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ 对 $\hat{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ 的贡献被抵消，L^1正则化项将 w_i 推至 0。

2. $w_i^* > \frac{\alpha}{H_{i,i}}$ 的情况。在这种情况下，正则化不会将 w_i 的最优值推至 0，而仅仅在那个方向上移动 $\frac{\alpha}{H_{i,i}}$ 的距离。

$w_i^* < 0$ 的情况与之类似，但是 L^1 惩罚项使 w_i 更接近 0(增加 $\frac{\alpha}{H_{i,i}}$) 或者为 0。

相比 L^2正则化，L^1正则化会产生更 **稀疏**（sparse）的解。此处稀疏性指的是最优值中的一些参数为 0。和 L^2正则化相比，L^1正则化的稀疏性具有本质的不同。

式 (7.13) 给出了 L^2正则化的解 $\tilde{\boldsymbol{w}}$。如果我们使用 Hessian 矩阵 $\boldsymbol{H}$ 为对角正定矩阵的假设（与 L^1正则化分析时一样），重新考虑这个等式，我们发现 $\tilde{w}_i = \frac{H_{i,i}}{H_{i,i}+\alpha} w_i^*$。如果 w_i^* 不是零，那么 $\tilde{w}_i$ 也会保持非零。这表明 L^2正则化不会使参数变得稀疏，而 L^1正则化有可能通过足够大的 α 实现稀疏。

由 L^1正则化导出的稀疏性质已经被广泛地用于**特征选择**（feature selection）机制。特征选择从可用的特征子集选择出有意义的特征，化简机器学习问题。著名的 LASSO (Tibshirani, 1995)（Least Absolute Shrinkage and Selection Operator）模型将 L^1 惩罚和线性模型结合，并使用最小二乘代价函数。L^1 惩罚使部分子集的权重为零，表明相应的特征可以被安全地忽略。

在第 5.6.1 节，我们看到许多正则化策略可以被解释为 MAP 贝叶斯推断，特别是 L^2正则化相当于权重是高斯先验的 MAP 贝叶斯推断。对于 L^1正则化，用于正则化代价函数的惩罚项 $\alpha\Omega(\boldsymbol{w}) = \alpha\sum_i |w_i|$ 与通过 MAP 贝叶斯推断最大化的对数先验项是等价的（$\boldsymbol{w} \in \mathbb{R}^n$ 并且权重先验是各向同性的拉普拉斯分布（式 (3.26)））：

$$\log p(\boldsymbol{w}) = \sum_i \log \text{Laplace}(w_i; 0, \frac{1}{\alpha}) = -\alpha \left\|\boldsymbol{w}\right\|_1 + n\log\alpha - n\log 2. \tag{7.24}$$

因为是关于 $\boldsymbol{w}$ 最大化进行学习，我们可以忽略 $\log\alpha - \log 2$ 项，因为它们与 $\boldsymbol{w}$ 无关。

7.2 作为约束的范数惩罚

考虑经过参数范数正则化的代价函数：

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}). \tag{7.25}$$

回顾第 4.4 节我们可以构造一个广义 Lagrange 函数来最小化带约束的函数，即在原始目标函数上添加一系列惩罚项。每个惩罚是一个被称为 **Karush–Kuhn–Tucker**（Karush–Kuhn–Tucker）乘子的系数以及一个表示约束是否满足的函数之间的乘积。如果我们想约束 $\Omega(\boldsymbol{\theta})$ 小于某个常数 k，我们可以构建广义 Lagrange 函数

$$\mathcal{L}(\boldsymbol{\theta}, \alpha; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha(\Omega(\boldsymbol{\theta}) - k). \tag{7.26}$$

这个约束问题的解由下式给出

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \max_{\alpha, \alpha \geq 0} \mathcal{L}(\boldsymbol{\theta}, \alpha). \tag{7.27}$$

如第 4.4 节中描述的，解决这个问题我们需要对 $\boldsymbol{\theta}$ 和 α 都做出调整。第 4.5 节给出了一个带 L^2 约束的线性回归实例。还有许多不同的优化方法，有些可能会使用梯度下降而其他可能会使用梯度为 0 的解析解，但在所有过程中 α 在 $\Omega(\boldsymbol{\theta}) > k$ 时必须增加，在 $\Omega(\boldsymbol{\theta}) < k$ 时必须减小。所有正值的 α 都鼓励 $\Omega(\boldsymbol{\theta})$ 收缩。最优值 α^* 也将鼓励 $\Omega(\boldsymbol{\theta})$ 收缩，但不会强到使得 $\Omega(\boldsymbol{\theta})$ 小于 k。

为了洞察约束的影响，我们可以固定 α^*，把这个问题看成只跟 $\boldsymbol{\theta}$ 有关的函数：

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \alpha^*) = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha^* \Omega(\boldsymbol{\theta}). \tag{7.28}$$

这和最小化 $\tilde{J}$ 的正则化训练问题是完全一样的。因此，我们可以把参数范数惩罚看作对权重强加的约束。如果 Ω 是 L^2 范数，那么权重就是被约束在一个 L^2 球中。如果 Ω 是 L^1 范数，那么权重就是被约束在一个 L^1 范数限制的区域中。通常我们不知道权重衰减系数 α^* 约束的区域大小，因为 α^* 的值不直接告诉我们 k 的值。原则上我们可以解得 k，但 k 和 α^* 之间的关系取决于 J 的形式。虽然我们不知道约束区域的确切大小，但我们可以通过增加或者减小 α 来大致扩大或收缩约束区域。较大的 α，将得到一个较小的约束区域。较小的 α，将得到一个较大的约束区域。

有时候，我们希望使用显式的限制，而不是惩罚。如第 4.4 节所述，我们可以修改下降算法（如随机梯度下降算法），使其先计算 $J(\boldsymbol{\theta})$ 的下降步，然后将 $\boldsymbol{\theta}$ 投影到满足 $\Omega(\boldsymbol{\theta}) < k$ 的最近点。如果我们知道什么样的 k 是合适的，而不想花时间寻找对应于此 k 处的 α 值，这会非常有用。

另一个使用显式约束和重投影而不是使用惩罚强加约束的原因是惩罚可能会导致目标函数非凸而使算法陷入局部极小 (对应于小的 $\boldsymbol{\theta}$)。当训练神经网络时，这通常表现为训练带有几个 ‘‘死亡单元’’ 的神经网络。这些单元不会对网络学到的函数有太大影响，因为进入或离开它们的权重都非常小。当使用权重范数的惩罚训练时，即使可以通过增加权重以显著减少 J，这些配置也可能是局部最优的。因为重投影实现的显式约束不鼓励权重接近原点，所以在这些情况下效果更好。通过重投影实现的显式约束只在权重变大并试图离开限制区域时产生作用。

最后，因为重投影的显式约束还对优化过程增加了一定的稳定性，所以这是另一个好处。当使用较高的学习率时，很可能进入正反馈，即大的权重诱导大梯度，然后使得权重获得较大更新。如果这些更新持续增加权重的大小，$\boldsymbol{\theta}$ 就会迅速增大，直到离原点很远而发生溢出。重投影的显式约束可以防止这种反馈环引起权重无限制地持续增加。Hinton *et al.* (2012c) 建议结合使用约束和高学习速率，这样能更快地探索参数空间，并保持一定的稳定性。

Hinton *et al.* (2012c) 尤其推荐由Srebro and Shraibman (2005) 引入的策略：约束神经网络层的权重矩阵每列的范数，而不是限制整个权重矩阵的 Frobenius 范数。分别限制每一列的范数可以防止某一隐藏单元有非常大的权重。如果我们将此约束转换成 Lagrange 函数中的一个惩罚，这将与 L^2 权重衰减类似但每个隐藏单元的权重都具有单独的 KKT 乘子。每个 KKT 乘子分别会被动态更新，以使每个隐藏单元服从约束。在实践中，列范数的限制总是通过重投影的显式约束来实现。

7.3 正则化和欠约束问题

在某些情况下，为了正确定义机器学习问题，正则化是必要的。机器学习中许多线性模型，包括线性回归和 PCA，都依赖于对矩阵 $\boldsymbol{X}^\top \boldsymbol{X}$ 求逆。只要 $\boldsymbol{X}^\top \boldsymbol{X}$ 是奇异的，这些方法就会失效。当数据生成分布在一些方向上确实没有差异时，或因为例子较少（即相对输入特征的维数来说）而在一些方向上没有观察到方差时，这个矩阵就是奇异的。在这种情况下，正则化的许多形式对应求逆 $\boldsymbol{X}^\top \boldsymbol{X} + \alpha \boldsymbol{I}$。这个正则化矩阵可以保证是可逆的。

相关矩阵可逆时，这些线性问题有闭式解。没有闭式解的问题也可能是欠定的。一个例子是应用于线性可分问题的逻辑回归。如果权重向量 $\boldsymbol{w}$ 能够实现完美分类，那么 $2\boldsymbol{w}$ 也会以更高似然实现完美分类。类似随机梯度下降的迭代优化算法将持续增加 $\boldsymbol{w}$ 的大小，理论上永远不会停止。在实践中，数值实现的梯度下降最终会达到导致数值溢出的超大权重，此时的行为将取决于程序员如何处理这些不是真正数字的值。

大多数形式的正则化能够保证应用于欠定问题的迭代方法收敛。例如，当似然的斜率等于权重衰减的系数时，权重衰减将阻止梯度下降继续增加权重的大小。

使用正则化解决欠定问题的想法不局限于机器学习。同样的想法在几个基本线性代数问题中也非常有用。

正如我们在第 2.9 节看到的，我们可以使用 Moore-Penrose 求解欠定线性方程。回想 $\boldsymbol{X}$ 伪逆 $\boldsymbol{X}^+$ 的一个定义：

$$\boldsymbol{X}^+ = \lim_{\alpha \searrow 0} (\boldsymbol{X}^\top \boldsymbol{X} + \alpha \boldsymbol{I})^{-1} \boldsymbol{X}^\top. \tag{7.29}$$

现在我们可以将第 7.29 节看作进行具有权重衰减的线性回归。具体来说，当正则化系数趋向 0 时，式 (7.29) 是式 (7.17) 的极限。因此，我们可以将伪逆解释为使用正则

化来稳定欠定问题。

7.4 数据集增强

让机器学习模型泛化得更好的最好办法是使用更多的数据进行训练。当然，在实践中，我们拥有的数据量是很有限的。解决这个问题的一种方法是创建假数据并添加到训练集中。对于一些机器学习任务，创建新的假数据相当简单。

对分类来说这种方法是最简单的。分类器需要一个复杂的高维输入 $\boldsymbol{x}$，并用单个类别标识 y 概括 $\boldsymbol{x}$。这意味着分类面临的一个主要任务是要对各种各样的变换保持不变。我们可以轻易通过转换训练集中的 $\boldsymbol{x}$ 来生成新的 $(\boldsymbol{x}, y)$ 对。

这种方法对于其他许多任务来说并不那么容易。例如，除非我们已经解决了密度估计问题，否则在密度估计任务中生成新的假数据是很困难的。

数据集增强对一个具体的分类问题来说是特别有效的方法：对象识别。图像是高维的并包括各种巨大的变化因素，其中有许多可以轻易地模拟。即使模型已使用卷积和池化技术（第九章）对部分平移保持不变，沿训练图像每个方向平移几个像素的操作通常可以大大改善泛化。许多其他操作如旋转图像或缩放图像也已被证明非常有效。

我们必须要小心，不能使用会改变类别的转换。例如，光学字符识别任务需要认识到 “b” 和 “d” 以及 “6” 和 “9” 的区别，所以对这些任务来说，水平翻转和旋转 180° 并不是合适的数据集增强方式。

能保持我们希望的分类不变，但不容易执行的转换也是存在的。例如，平面外绕轴转动难以通过简单的几何运算在输入像素上实现。

数据集增强对语音识别任务也是有效的 (Jaitly and Hinton, 2013)。

在神经网络的输入层注入噪声 (Sietsma and Dow, 1991) 也可以被看作是数据增强的一种方式。对于许多分类甚至一些回归任务而言，即使小的随机噪声被加到输入，任务仍应该是能够被解决的。然而，神经网络被证明对噪声不是非常健壮 (Tang and Eliasmith, 2010)。改善神经网络健壮性的方法之一是简单地将随机噪声添加到输入再进行训练。输入噪声注入是一些无监督学习算法的一部分，如去噪自编码器(Vincent *et al.*, 2008a)。向隐藏单元施加噪声也是可行的，这可以被看作在多个抽象层上进行的数据集增强。Poole *et al.* (2014) 最近表明，噪声的幅度被细心调整后，

该方法是非常高效的。我们将在第 7.12 节介绍一个强大的正则化策略 Dropout，该策略可以被看作是通过与噪声相乘构建新输入的过程。

在比较机器学习基准测试的结果时，考虑其采取的数据集增强是很重要的。通常情况下，人工设计的数据集增强方案可以大大减少机器学习技术的泛化误差。将一个机器学习算法的性能与另一个进行对比时，对照实验是必要的。在比较机器学习算法 A 和机器学习算法 B 时，应该确保这两个算法使用同一人工设计的数据集增强方案。假设算法 A 在没有数据集增强时表现不佳，而 B 结合大量人工转换的数据后表现良好。在这样的情况下，很可能是合成转化引起了性能改进，而不是机器学习算法 B 比算法 A 更好。有时候，确定实验是否已经适当控制需要主观判断。例如，向输入注入噪声的机器学习算法是执行数据集增强的一种形式。通常，普适操作（例如，向输入添加高斯噪声）被认为是机器学习算法的一部分，而特定于一个应用领域（如随机地裁剪图像）的操作被认为是独立的预处理步骤。

7.5 噪声鲁棒性

第 7.4 节已经提出将噪声作用于输入，作为数据集增强策略。对于某些模型而言，向输入添加方差极小的噪声等价于对权重施加范数惩罚 (Bishop, 1995a,b)。在一般情况下，注入噪声远比简单地收缩参数强大，特别是噪声被添加到隐藏单元时会更加强大。向隐藏单元添加噪声是值得单独讨论重要的话题；在第 7.12 节所述 Dropout 算法是这种做法的主要发展方向。

另一种正则化模型的噪声使用方式是将其加到权重。这项技术主要用于循环神经网络 (Jim *et al.*, 1996; Graves, 2011)。这可以被解释为关于权重的贝叶斯推断的随机实现。贝叶斯学习过程将权重视为不确定的，并且可以通过概率分布表示这种不确定性。向权重添加噪声是反映这种不确定性的一种实用的随机方法。

在某些假设下，施加于权重的噪声可以被解释为与更传统的正则化形式等同，鼓励要学习的函数保持稳定。我们研究回归的情形，也就是训练将一组特征 $\boldsymbol{x}$ 映射成一个标量的函数 $\hat{y}(\boldsymbol{x})$，并使用最小二乘代价函数衡量模型预测值 $\hat{y}(\boldsymbol{x})$ 与真实值 y 的误差：

$$J = \mathbb{E}_{p(x,y)}[(\hat{y}(\boldsymbol{x}) - y)^2]. \tag{7.30}$$

训练集包含 m 对标注样例 $\{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$。

现在我们假设对每个输入表示，网络权重添加随机扰动 $\epsilon_{\boldsymbol{w}} \sim \mathcal{N}(\boldsymbol{\epsilon}; 0, \eta \boldsymbol{I})$。想象我们有一个标准的 l 层 MLP。我们将扰动模型记为 $\hat{y}_{\epsilon_{\boldsymbol{W}}}(\boldsymbol{x})$。尽管有噪声注入，我们仍然希望减少网络输出误差的平方。因此目标函数变为：

$$\tilde{J}_{\boldsymbol{W}} = \mathbb{E}_{p(\boldsymbol{x},y,\epsilon_{\boldsymbol{W}})}[(\hat{y}_{\epsilon_{\boldsymbol{W}}}(\boldsymbol{x}) - y)^2] \tag{7.31}$$

$$= \mathbb{E}_{p(\boldsymbol{x},y,\epsilon_{\boldsymbol{W}})}[\hat{y}^2_{\epsilon_{\boldsymbol{W}}}(\boldsymbol{x}) - 2y\hat{y}_{\epsilon_{\boldsymbol{W}}}(\boldsymbol{x}) + y^2]. \tag{7.32}$$

对于小的 η，最小化带权重噪声（方差为 $\eta \boldsymbol{I}$）的 J 等同于最小化附加正则化项：$\eta \mathbb{E}_{p(\boldsymbol{x},y)}[\|\nabla_{\boldsymbol{W}} \hat{y}(\boldsymbol{x})\|^2]$ 的 J。这种形式的正则化鼓励参数进入权重小扰动对输出相对影响较小的参数空间区域。换句话说，它推动模型进入对权重小的变化相对不敏感的区域，找到的点不只是极小点，还是由平坦区域所包围的极小点 (Hochreiter and Schmidhuber, 1995)。在简化的线性回归中（例如，$\hat{y}(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$），正则项退化为 $\eta \mathbb{E}_{p(\boldsymbol{x})}[\|\boldsymbol{x}\|^2]$，这与函数的参数无关，因此不会对 $\tilde{J}_{\boldsymbol{w}}$ 关于模型参数的梯度有影响。

7.5.1 向输出目标注入噪声

大多数数据集的 y 标签都有一定错误。错误的 y 不利于最大化 $\log p(y \mid \boldsymbol{x})$。避免这种情况的一种方法是显式地对标签上的噪声进行建模。例如，我们可以假设，对于一些小常数 ϵ，训练集标记 y 是正确的概率是 $1 - \epsilon$，（以 ϵ 的概率）任何其他可能的标签也可能是正确的。这个假设很容易就能解析地与代价函数结合，而不用显式地抽取噪声样本。例如，**标签平滑**（label smoothing）通过把确切分类目标从 0 和 1 替换成 $\frac{\epsilon}{k-1}$ 和 $1 - \epsilon$，正则化具有 k 个输出的 softmax 函数 的模型。标准交叉熵损失可以用在这些非确切目标的输出上。使用 softmax 函数 和明确目标的最大似然学习可能永远不会收敛——softmax 函数 永远无法真正预测 0 概率或 1 概率，因此它会继续学习越来越大的权重，使预测更极端。使用如权重衰减等其他正则化策略能够防止这种情况。标签平滑的优势是能够防止模型追求确切概率而不影响模型学习正确分类。这种策略自 20 世纪 80 年代就已经被使用，并在现代神经网络继续保持显著特色 (Szegedy *et al.*, 2015)。

7.6 半监督学习

在半监督学习的框架下，$P(\mathbf{x})$ 产生的未标记样本和 $P(\mathbf{x}, \mathbf{y})$ 中的标记样本都用于估计 $P(\mathbf{y} \mid \mathbf{x})$ 或者根据 $\mathbf{x}$ 预测 $\mathbf{y}$。

在深度学习的背景下，半监督学习通常指的是学习一个表示 $\boldsymbol{h} = f(\boldsymbol{x})$。学习表示的目的是使相同类中的样本有类似的表示。无监督学习可以为如何在表示空间聚集样本提供有用线索。在输入空间紧密聚集的样本应该被映射到类似的表示。在许多情况下，新空间上的线性分类器可以达到较好的泛化 (Belkin and Niyogi, 2002; Chapelle *et al.*, 2003)。这种方法的一个经典变种是使用主成分分析作为分类前（在投影后的数据上分类）的预处理步骤。

我们可以构建这样一个模型，其中生成模型 $P(\mathbf{x})$ 或 $P(\mathbf{x}, \mathbf{y})$ 与判别模型 $P(\mathbf{y} \mid \mathbf{x})$ 共享参数，而不用分离无监督和监督部分。我们权衡监督模型准则 $-\log P(\mathbf{y} \mid \mathbf{x})$ 和无监督或生成模型准则（如 $-\log P(\mathbf{x})$ 或 $-\log P(\mathbf{x}, \mathbf{y})$）。生成模型准则表达了对监督学习问题解的特殊形式的先验知识 (Lasserre *et al.*, 2006)，即 $P(\mathbf{x})$ 的结构通过某种共享参数的方式连接到 $P(\mathbf{y} \mid \mathbf{x})$。通过控制在总准则中的生成准则，我们可以获得比纯生成或纯判别训练准则更好的权衡 (Lasserre *et al.*, 2006; Larochelle and Bengio, 2008a)。

Salakhutdinov and Hinton (2008) 描述了一种学习回归核机器中核函数的方法，其中建模 $P(\mathbf{x})$ 时使用的未标记样本大大提高了 $P(\mathbf{y} \mid \mathbf{x})$ 的效果。

更多半监督学习的信息，请参阅 Chapelle *et al.* (2006)。

7.7 多任务学习

多任务学习 (Caruana, 1993) 是通过合并几个任务中的样例（可以视为对参数施加的软约束）来提高泛化的一种方式。正如额外的训练样本能够将模型参数推向具有更好泛化能力的值一样，当模型的一部分被多个额外的任务共享时，这部分将被约束为良好的值（如果共享合理），通常会带来更好的泛化能力。

图 7.2 展示了多任务学习中非常普遍的一种形式，其中不同的监督任务（给定 $\mathbf{x}$ 预测 $\mathbf{y}^{(i)}$）共享相同的输入 $\mathbf{x}$ 以及一些中间层表示 $\boldsymbol{h}^{(\text{share})}$，能学习共同的因素池。该模型通常可以分为两类相关的参数：

1. 具体任务的参数（只能从各自任务的样本中实现良好的泛化）。如图 7.2 中的上层。
2. 所有任务共享的通用参数（从所有任务的汇集数据中获益）。如图 7.2 中的下层。

因为共享参数，其统计强度可大大提高（共享参数的样本数量相对于单任务模

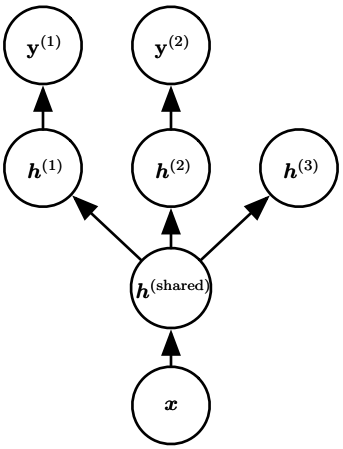

图 7.2: 多任务学习在深度学习框架中可以以多种方式进行，该图说明了任务共享相同输入但涉及不同目标随机变量的常见情况。深度网络的较低层（无论是监督前馈的，还是包括向下箭头的生成组件）可以跨这样的任务共享，而任务特定的参数（分别与从 $\boldsymbol{h}^{(1)}$ 和 $\boldsymbol{h}^{(2)}$ 进入和发出的权重）可以在共享表示 $\boldsymbol{h}^{(\text{shared})}$ 之上学习。这里的基本假设是存在解释输入 $\mathbf{x}$ 变化的共同因素池，而每个任务与这些因素的子集相关联。在该示例中，额外假设顶层隐藏单元 $\boldsymbol{h}^{(1)}$ 和 $\boldsymbol{h}^{(2)}$ 专用于每个任务（分别预测 $\mathbf{y}^{(1)}$ 和 $\mathbf{y}^{(2)}$），而一些中间层表示 $\boldsymbol{h}^{(\text{shared})}$ 在所有任务之间共享。在无监督学习情况下，一些顶层因素不与输出任务 ($\boldsymbol{h}^{(3)}$) 的任意一个关联是有意义的：这些因素可以解释一些输入变化但与预测 $\mathbf{y}^{(1)}$ 或 $\mathbf{y}^{(2)}$ 不相关。

式增加的比例），并能改善泛化和泛化误差的范围 (Baxter, 1995)。当然，仅当不同的任务之间存在某些统计关系的假设是合理（意味着某些参数能通过不同任务共享）时才会发生这种情况。

从深度学习的观点看，底层的先验知识如下：能解释数据变化（在与之相关联的不同任务中观察到）的因素中，某些因素是跨两个或更多任务共享的。

7.8 提前终止

当训练有足够的表示能力甚至会过拟合的大模型时，我们经常观察到，训练误差会随着时间的推移逐渐降低但验证集的误差会再次上升。图 7.3 是这些现象的一个例子，这种现象几乎一定会出现。

这意味着我们只要返回使验证集误差最低的参数设置，就可以获得验证集误差更低的模型（并且因此有希望获得更好的测试误差）。在每次验证集误差有所改善

后，我们存储模型参数的副本。当训练算法终止时，我们返回这些参数而不是最新的参数。当验证集上的误差在事先指定的循环次数内没有进一步改善时，算法就会终止。此过程在算法 7.1 中有更正式的说明。

这种策略被称为 **提前终止**（early stopping）。这可能是深度学习中最常用的正则化形式。它的流行主要是因为有效性和简单性。

算法 7.1 用于确定最佳训练时间量的提前终止元算法。这种元算法是一种通用策略，可以很好地在各种训练算法和各种量化验证集误差的方法上工作。

令 n 为评估间隔的步数。
令 p 为 “耐心 (patience)”，即观察到较坏的验证集表现 p 次后终止。
令 $\boldsymbol{\theta}_o$ 为初始参数。
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i^* \leftarrow i$
while $j < p$ **do**
  运行训练算法 n 步，更新 $\boldsymbol{\theta}$ 。
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
最佳参数为 $\boldsymbol{\theta}^*$，最佳训练步数为 i^*

我们可以认为提前终止是非常高效的超参数选择算法。按照这种观点，训练步

数仅是另一个超参数。我们从图 7.3 可以看到，这个超参数在验证集上具有 U 型性能曲线。很多控制模型容量的超参数在验证集上都是这样的 U 型性能曲线，如图 5.3 。在提前终止的情况下，我们通过控制拟合训练集的步数来控制模型的有效容量。大多数超参数的选择必须使用高代价的猜测和检查过程，我们需要在训练开始时猜测一个超参数，然后运行几个步骤检查它的训练效果。“训练时间” 是唯一只要跑一次训练就能尝试很多值的超参数。通过提前终止自动选择超参数的唯一显著的代价是训练期间要定期评估验证集。在理想情况下，这可以并行在与主训练过程分离的机器上，或独立的 CPU，或独立的 GPU 上完成。如果没有这些额外的资源，可以使用比训练集小的验证集或较不频繁地评估验证集来减小评估代价，较粗略地估算取得最佳的训练时间。

另一个提前终止的额外代价是需要保持最佳的参数副本。这种代价一般是可忽略的，因为可以将它储存在较慢较大的存储器上（例如，在 GPU 内存中训练，但将最佳参数存储在主存储器或磁盘驱动器上）。由于最佳参数的写入很少发生而且从不在训练过程中读取，这些偶发的慢写入对总训练时间的影响不大。

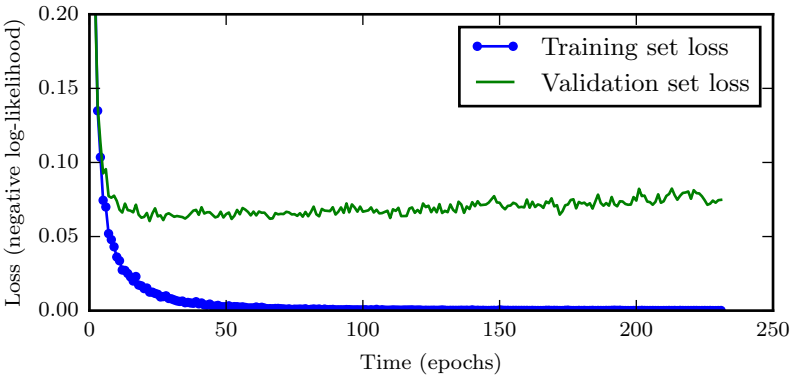

图 7.3: 学习曲线显示负对数似然损失如何随时间变化（表示为遍历数据集的训练迭代数，或 **轮数**（epochs））。在这个例子中，我们在 MNIST 上训练了一个 maxout 网络。我们可以观察到训练目标随时间持续减小，但验证集上的平均损失最终会再次增加，形成不对称的 U 形曲线。

提前终止是一种非常不显眼的正则化形式，它几乎不需要改变基本训练过程、目标函数或一组允许的参数值。这意味着，无需破坏学习动态就能很容易地使用提前终止。相对于权重衰减，必须小心不能使用太多的权重衰减，以防网络陷入不良局部极小点(对应于病态的小权重)。

提前终止可单独使用或与其他的正则化策略结合使用。即使为鼓励更好泛化，使用正则化策略改进目标函数，在训练目标的局部极小点达到最好泛化也是非常罕见的。

提前终止需要验证集，这意味着某些训练数据不能被馈送到模型。为了更好地利用这一额外的数据，我们可以在完成提前终止的首次训练之后，进行额外的训练。在第二轮，即额外的训练步骤中，所有的训练数据都被包括在内。有两个基本的策略都可以用于第二轮训练过程。

一个策略（算法 7.2 ）是再次初始化模型，然后使用所有数据再次训练。在这个第二轮训练过程中，我们使用第一轮提前终止训练确定的最佳步数。此过程有一些细微之处。例如，我们没有办法知道重新训练时，对参数进行相同次数的更新和对数据集进行相同次数的遍历哪一个更好。由于训练集变大了，在第二轮训练时，每一次遍历数据集将会更多次地更新参数。

另一个策略是保持从第一轮训练获得的参数，然后使用全部的数据继续训练。在这个阶段，已经没有验证集指导我们需要在训练多少步后终止。取而代之，我们可以监控验证集的平均损失函数，并继续训练，直到它低于提前终止过程终止时的目标值。此策略避免了重新训练模型的高成本，但表现并没有那么好。例如，验证集的目标不一定能达到之前的目标值，所以这种策略甚至不能保证终止。我们会在算法 7.3 中更正式地介绍这个过程。

提前终止对减少训练过程的计算成本也是有用的。除了由于限制训练的迭代次数而明显减少的计算成本，还带来了正则化的益处（不需要添加惩罚项的代价函数或计算这种附加项的梯度）。

算法 7.2 使用提前终止确定训练步数，然后在所有数据上训练的元算法。

令 $\boldsymbol{X}^{(\text{train})}$ 和 $\boldsymbol{y}^{(\text{train})}$ 为训练集。

将 $\boldsymbol{X}^{(\text{train})}$ 和 $\boldsymbol{y}^{(\text{train})}$ 分别分割为 $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ 和 $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$。

从随机 $\boldsymbol{\theta}$ 开始，使用 $\boldsymbol{X}^{(\text{subtrain})}$ 和 $\boldsymbol{y}^{(\text{subtrain})}$ 作为训练集，$\boldsymbol{X}^{(\text{valid})}$ 和 $\boldsymbol{y}^{(\text{valid})}$ 作为验证集，运行 (算法 7.1)。这将返回最佳训练步数 i^*。

将 $\boldsymbol{\theta}$ 再次设为随机值。

在 $\boldsymbol{X}^{(\text{train})}$ 和 $\boldsymbol{y}^{(\text{train})}$ 上训练 i^* 步。

算法 7.3 使用提前终止确定将会过拟合的目标值，然后在所有数据上训练直到再次达到该值的元算法。

令 $\boldsymbol{X}^{(\text{train})}$ 和 $\boldsymbol{y}^{(\text{train})}$ 为训练集。
将 $\boldsymbol{X}^{(\text{train})}$ 和 $\boldsymbol{y}^{(\text{train})}$ 分别分割为 ($\boldsymbol{X}^{(\text{subtrain})}$, $\boldsymbol{X}^{(\text{valid})}$) 和 ($\boldsymbol{y}^{(\text{subtrain})}$, $\boldsymbol{y}^{(\text{valid})}$)。
从随机 $\boldsymbol{\theta}$ 开始，使用 $\boldsymbol{X}^{(\text{subtrain})}$ 和 $\boldsymbol{y}^{(\text{subtrain})}$ 作为训练集，$\boldsymbol{X}^{(\text{valid})}$ 和 $\boldsymbol{y}^{(\text{valid})}$ 作为验证集，运行 (算法 7.1)。这会更新 $\boldsymbol{\theta}$。
$\epsilon \leftarrow J(\boldsymbol{\theta}, \boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{subtrain})})$
while $J(\boldsymbol{\theta}, \boldsymbol{X}^{(\text{valid})}, \boldsymbol{y}^{(\text{valid})}) > \epsilon$ **do**
  在 $\boldsymbol{X}^{(\text{train})}$ 和 $\boldsymbol{y}^{(\text{train})}$ 上训练 n 步。
end while

提前终止为何具有正则化效果： 目前为止，我们已经声明提前终止是一种正则化策略，但我们只通过展示验证集误差的学习曲线是一个 U 型曲线来支持这种说法。提前终止正则化模型的真正机制是什么呢? Bishop (1995a) 和 Sjöberg and Ljung (1995) 认为提前终止可以将优化过程的参数空间限制在初始参数值 $\boldsymbol{\theta}_0$ 的小邻域内。更具体地，想象用学习率 ϵ 进行 τ 个优化步骤 (对应于 τ 个训练迭代)。我们可以将 $\epsilon\tau$ 作为有效容量的度量。假设梯度有界，限制迭代的次数和学习速率能够限制从 $\boldsymbol{\theta}_0$ 到达的参数空间的大小，如图 7.4 所示。在这个意义上，$\epsilon\tau$ 的效果就好像是权重衰减系数的倒数。

事实上，在二次误差的简单线性模型和简单的梯度下降情况下，我们可以展示提前终止相当于 L^2正则化。

为了与经典 L^2正则化比较，我们只考察唯一的参数是线性权重 ($\boldsymbol{\theta} = \boldsymbol{w}$) 的简单情形。我们在权重 $\boldsymbol{w}$ 的经验最佳值 $\boldsymbol{w}^*$ 附近以二次近似建模代价函数 J：

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*), \tag{7.33}$$

其中 $\boldsymbol{H}$ 是 J 关于 $\boldsymbol{w}$ 在 $\boldsymbol{w}^*$ 点的Hessian。鉴于假设 $\boldsymbol{w}^*$ 是 $J(\boldsymbol{w})$ 的最小点，我们知道 $\boldsymbol{H}$ 为半正定。在局部泰勒级数逼近下，梯度由下式给出：

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*). \tag{7.34}$$

接下来我们研究训练时参数向量的轨迹。为简化起见，我们将参数向量初始化为原点[3]，也就是 $\boldsymbol{w}^{(0)} = 0$。我们通过分析 $\hat{J}$ 上的梯度下降来研究 J 上近似的梯度

[3]对于神经网络，我们需要打破隐藏单元间的对称平衡因此不能将所有参数都初始化为 $\mathbf{0}$ (如第 6.2 节所讨论的)。

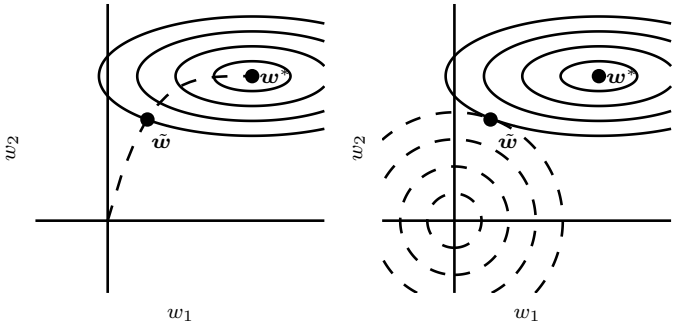

图 7.4: 提前终止效果的示意图。*(左)* 实线轮廓线表示负对数似然的轮廓。虚线表示从原点开始的 SGD 所经过的轨迹。提前终止的轨迹在较早的点 $\tilde{\boldsymbol{w}}$ 处停止，而不是停止在最小化代价的点 $\boldsymbol{w}^*$ 处。*(右)* 为了对比，使用 L^2正则化效果的示意图。虚线圆圈表示 L^2 惩罚的轮廓，L^2 惩罚使得总代价的最小值比非正则化代价的最小值更靠近原点。

下降的效果：

$$\boldsymbol{w}^{(\tau)} = \boldsymbol{w}^{(\tau-1)} - \epsilon \nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}^{(\tau-1)}) \tag{7.35}$$

$$= \boldsymbol{w}^{(\tau-1)} - \epsilon \boldsymbol{H}(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*), \tag{7.36}$$

$$\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon \boldsymbol{H})(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*). \tag{7.37}$$

现在让我们在 $\boldsymbol{H}$ 特征向量的空间中改写表达式，利用 $\boldsymbol{H}$ 的特征分解：$\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top$，其中 $\boldsymbol{\Lambda}$ 是对角矩阵，$\boldsymbol{Q}$ 是特征向量的一组标准正交基。

$$\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top)(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \tag{7.38}$$

$$\boldsymbol{Q}^\top(\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^*) = (\boldsymbol{I} - \epsilon\boldsymbol{\Lambda})\boldsymbol{Q}^\top(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \tag{7.39}$$

假定 $\boldsymbol{w}^{(0)} = 0$ 并且 ϵ 选择得足够小以保证 $|1 - \epsilon\lambda_i| < 1$，经过 τ 次参数更新后轨迹如下：

$$\boldsymbol{Q}^\top \boldsymbol{w}^{(\tau)} = [\boldsymbol{I} - (\boldsymbol{I} - \epsilon\boldsymbol{\Lambda})^\tau]\boldsymbol{Q}^\top \boldsymbol{w}^*. \tag{7.40}$$

现在，式 (7.13) 中 $\boldsymbol{Q}^\top \tilde{\boldsymbol{w}}$ 的表达式能被重写为：

$$\boldsymbol{Q}^\top \tilde{\boldsymbol{w}} = (\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^\top \boldsymbol{w}^*, \tag{7.41}$$

$$\boldsymbol{Q}^\top \tilde{\boldsymbol{w}} = [\boldsymbol{I} - (\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\alpha]\boldsymbol{Q}^\top \boldsymbol{w}^*. \tag{7.42}$$

然而，对于其他任何初始值 $\boldsymbol{w}_{(0)}$ 该论证都成立

比较式 (7.40) 和式 (7.42)，我们能够发现，如果超参数 ϵ, α 和 τ 满足如下：

$$(\boldsymbol{I} - \epsilon\boldsymbol{\Lambda})^{\tau} = (\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\alpha, \tag{7.43}$$

那么 L^2 正则化和提前终止可以被看作是等价的（至少在目标函数的二次近似下）。进一步取对数，使用 $\log\ (1+x)$ 的级数展开，我们可以得出结论：如果所有 λ_i 是小的（即 $\epsilon\lambda_i \ll 1$ 且 $\lambda_i/\alpha \ll 1$），那么

$$\tau \approx \frac{1}{\epsilon\alpha}, \tag{7.44}$$

$$\alpha \approx \frac{1}{\tau\epsilon}. \tag{7.45}$$

也就是说，在这些假设下，训练迭代次数 τ 起着与 L^2 参数成反比的作用，$\tau\epsilon$ 的倒数与权重衰减系数的作用类似。

在大曲率（目标函数）方向上的参数值受正则化影响小于小曲率方向。当然，在提前终止的情况下，这实际上意味着在大曲率方向的参数比较小曲率方向的参数更早地学习到。

本节中的推导表明长度为 τ 的轨迹结束于 L^2正则化目标的极小点。当然，提前终止比简单的轨迹长度限制更丰富；取而代之，提前终止通常涉及监控验证集误差，以便在空间特别好的点处终止轨迹。因此提前终止比权重衰减更具有优势，提前终止能自动确定正则化的正确量，而权重衰减需要进行多个不同超参数值的训练实验。

7.9 参数绑定和参数共享

目前为止，本章讨论对参数添加约束或惩罚时，一直是相对于固定的区域或点。例如，L^2正则化（或权重衰减）对参数偏离零的固定值进行惩罚。然而，有时我们可能需要其他的方式来表达我们对模型参数适当值的先验知识。有时候，我们可能无法准确地知道应该使用什么样的参数，但我们根据相关领域和模型结构方面的知识得知模型参数之间应该存在一些相关性。

我们经常想要表达的一种常见依赖是某些参数应当彼此接近。考虑以下情形：我们有两个模型执行相同的分类任务（具有相同类别），但输入分布稍有不同。形式地，我们有参数为 $\boldsymbol{w}^{(A)}$ 的模型 A 和参数为 $\boldsymbol{w}^{(B)}$ 的模型 B。这两种模型将输入映射到两个不同但相关的输出：$\hat{y}^{(A)} = f(\boldsymbol{w}^{(A)}, \boldsymbol{x})$ 和 $\hat{y}^{(B)} = f(\boldsymbol{w}^{(B)}, \boldsymbol{x})$。

我们可以想象，这些任务会足够相似（或许具有相似的输入和输出分布），因此我们认为模型参数应彼此靠近：$\forall i, w_i^{(A)}$ 应该与 $w_i^{(B)}$ 接近。我们可以通过正则化利用此信息。具体来说，我们可以使用以下形式的参数范数惩罚：$\Omega(\boldsymbol{w}^{(A)}, \boldsymbol{w}^{(B)}) = \left\| \boldsymbol{w}^{(A)} - \boldsymbol{w}^{(B)} \right\|_2^2$。在这里我们使用 L^2 惩罚，但也可以使用其他选择。

这种方法由Lasserre *et al.* (2006) 提出，正则化一个模型（监督模式下训练的分类器）的参数，使其接近另一个无监督模式下训练的模型（捕捉观察到的输入数据的分布）的参数。构造的这种架构使得分类模型中的许多参数能与无监督模型中对应的参数匹配。

参数范数惩罚是正则化参数使其彼此接近的一种方式，而更流行的方法是使用约束：强迫某些参数相等。由于我们将各种模型或模型组件解释为共享唯一的一组参数，这种正则化方法通常被称为 **参数共享**（parameter sharing）。和正则化参数使其接近（通过范数惩罚）相比，参数共享的一个显著优点是，只有参数（唯一一个集合）的子集需要被存储在内存中。对于某些特定模型，如卷积神经网络，这可能可以显著减少模型所占用的内存。

7.9.1 卷积神经网络

目前为止，最流行和广泛使用的参数共享出现在应用于计算机视觉的 **卷积神经网络**（CNN）中。

自然图像有许多统计属性是对转换不变的。例如，猫的照片即使向右边移了一个像素，仍保持猫的照片。CNN通过在图像多个位置共享参数来考虑这个特性。相同的特征（具有相同权重的隐藏单元）在输入的不同位置上计算获得。这意味着无论猫出现在图像中的第 i 列或 $i+1$ 列，我们都可以使用相同的猫探测器找到猫。

参数共享显著降低了CNN模型的参数数量，并显著提高了网络的大小而不需要相应地增加训练数据。它仍然是将领域知识有效地整合到网络架构的最佳范例之一。

我们将会在第九章中更详细地讨论卷积神经网络。

7.10 稀疏表示

前文所述的权重衰减直接惩罚模型参数。另一种策略是惩罚神经网络中的激活单元，稀疏化激活单元。这种策略间接地对模型参数施加了复杂惩罚。

我们已经讨论过（在第 7.1.2 节中）L^1 惩罚如何诱导稀疏的参数，即许多参数为零（或接近于零）。另一方面，表示的稀疏描述了许多元素是零（或接近零）的表示。我们可以线性回归的情况下简单说明这种区别：

$$\underbrace{\begin{bmatrix}18\\5\\15\\-9\\-3\end{bmatrix}}_{\boldsymbol{y}\in\mathbb{R}^m}=\underbrace{\begin{bmatrix}4&0&0&-2&0&0\\0&0&-1&0&3&0\\0&5&0&0&0&0\\1&0&0&-1&0&-4\\1&0&0&0&-5&0\end{bmatrix}}_{\boldsymbol{A}\in\mathbb{R}^{m\times n}}\underbrace{\begin{bmatrix}2\\3\\-2\\-5\\1\\4\end{bmatrix}}_{\boldsymbol{x}\in\mathbb{R}^n} \tag{7.46}$$

$$\underbrace{\begin{bmatrix}-14\\1\\19\\2\\23\end{bmatrix}}_{\boldsymbol{y}\in\mathbb{R}^m}=\underbrace{\begin{bmatrix}3&-1&2&-5&4&1\\4&2&-3&-1&1&3\\-1&5&4&2&-3&-2\\3&1&2&-3&0&-3\\-5&4&-2&2&-5&-1\end{bmatrix}}_{\boldsymbol{B}\in\mathbb{R}^{m\times n}}\underbrace{\begin{bmatrix}0\\2\\0\\0\\-3\\0\end{bmatrix}}_{\boldsymbol{h}\in\mathbb{R}^n} \tag{7.47}$$

第一个表达式是参数稀疏的线性回归模型的例子。第二个表达式是数据 $\boldsymbol{x}$ 具有稀疏表示 $\boldsymbol{h}$ 的线性回归。也就是说，$\boldsymbol{h}$ 是 $\boldsymbol{x}$ 的一个函数，在某种意义上表示存在于 $\boldsymbol{x}$ 中的信息，但只是用一个稀疏向量表示。

表示的正则化可以使用参数正则化中同种类型的机制实现。

表示的范数惩罚正则化是通过向损失函数 J 添加对表示的范数惩罚来实现的。我们将这个惩罚记作 $\Omega(\boldsymbol{h})$。和以前一样，我们将正则化后的损失函数记作 $\tilde{J}$：

$$\tilde{J}(\boldsymbol{\theta};\boldsymbol{X},\boldsymbol{y})=J(\boldsymbol{\theta};\boldsymbol{X},\boldsymbol{y})+\alpha\Omega(\boldsymbol{h}), \tag{7.48}$$

其中 $\alpha\in[0,\infty]$ 权衡范数惩罚项的相对贡献，越大的 α 对应越多的正则化。

正如对参数的 L^1 惩罚诱导参数稀疏性，对表示元素的 L^1 惩罚诱导稀疏的表示：$\Omega(\boldsymbol{h})=\|\boldsymbol{h}\|_1=\sum_i|h_i|$。当然 L^1 惩罚是使表示稀疏的方法之一。其他方法还包括从表示上的Student-t 先验导出的惩罚 (Olshausen and Field, 1996; Bergstra, 2011) 和KL 散度惩罚 (Larochelle and Bengio, 2008b)，这些方法对于将表示中的元素约束于单位区间上特别有用。Lee *et al.* (2008) 和Goodfellow *et al.* (2009) 都提供了正则化几个样本平均激活的例子，即令 $\frac{1}{m}\sum_i\boldsymbol{h}^{(i)}$ 接近某些目标值（如每项都是 .01 的向

量)。

还有一些其他方法通过激活值的硬性约束来获得表示稀疏。例如，**正交匹配追踪** (orthogonal matching pursuit)(Pati *et al.*, 1993) 通过解决以下约束优化问题将输入值 $\boldsymbol{x}$ 编码成表示 $\boldsymbol{h}$

$$\arg\min_{\boldsymbol{h}, \|\boldsymbol{h}\|_0 < k} \|\boldsymbol{x} - \boldsymbol{W}\boldsymbol{h}\|^2, \tag{7.49}$$

其中 $\|\boldsymbol{h}\|_0$ 是 $\boldsymbol{h}$ 中非零项的个数。当 $\boldsymbol{W}$ 被约束为正交时，我们可以高效地解决这个问题。这种方法通常被称为OMP-k，通过 k 指定允许的非零特征数量。Coates and Ng (2011) 证明OMP-1 可以成为深度架构中非常有效的特征提取器。

含有隐藏单元的模型在本质上都能变得稀疏。在本书中，我们将看到在各种情况下使用稀疏正则化的例子。

7.11 Bagging 和其他集成方法

Bagging（ bootstrap aggregating ）是通过结合几个模型降低泛化误差的技术 (Breiman, 1994)。主要想法是分别训练几个不同的模型，然后让所有模型表决测试样例的输出。这是机器学习中常规策略的一个例子，被称为**模型平均**（ model averaging)。采用这种策略的技术被称为集成方法。

模型平均（ model averaging ）奏效的原因是不同的模型通常不会在测试集上产生完全相同的误差。

假设我们有 k 个回归模型。假设每个模型在每个例子上的误差是 ϵ_i，这个误差服从零均值方差为 $\mathbb{E}[\epsilon_i^2] = v$ 且协方差为 $\mathbb{E}[\epsilon_i \epsilon_j] = c$ 的多维正态分布。通过所有集成模型的平均预测所得误差是 $\frac{1}{k}\sum_i \epsilon_i$。集成预测器平方误差的期望是

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i \left(\epsilon_i^2 + \sum_{j \neq i} \epsilon_i \epsilon_j\right)\right], \tag{7.50}$$

$$= \frac{1}{k}v + \frac{k-1}{k}c. \tag{7.51}$$

在误差完全相关即 $c = v$ 的情况下，均方误差减少到 v，所以模型平均没有任何帮助。在错误完全不相关即 $c = 0$ 的情况下，该集成平方误差的期望仅为 $\frac{1}{k}v$。这意味着集成平方误差的期望会随着集成规模增大而线性减小。换言之，平均上，集成至

少与它的任何成员表现得一样好，并且如果成员的误差是独立的，集成将显著地比其成员表现得更好。

不同的集成方法以不同的方式构建集成模型。例如，集成的每个成员可以使用不同的算法和目标函数训练成完全不同的模型。Bagging是一种允许重复多次使用同一种模型、训练算法和目标函数的方法。

具体来说，Bagging涉及构造 k 个不同的数据集。每个数据集从原始数据集中重复采样构成，和原始数据集具有相同数量的样例。这意味着，每个数据集以高概率缺少一些来自原始数据集的例子，还包含若干重复的例子（如果所得训练集与原始数据集大小相同，那所得数据集中大概有原始数据集 2/3 的实例）。模型 i 在数据集 i 上训练。每个数据集所含样本的差异导致了训练模型之间的差异。图 7.5 是一个例子。

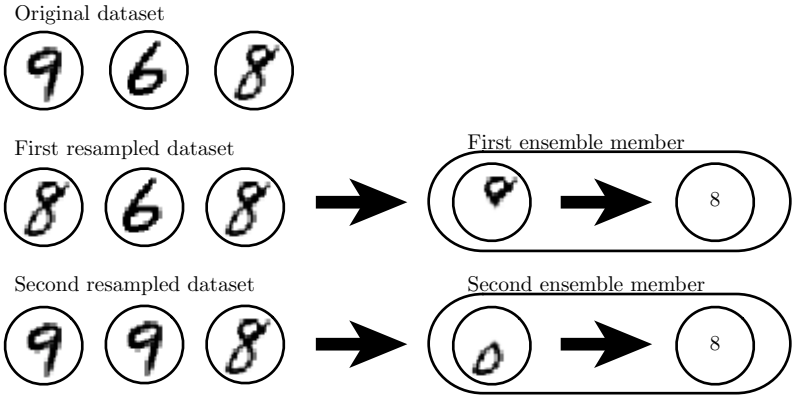

图 7.5: 描述Bagging如何工作的草图。假设我们在上述数据集（包含一个 8、一个 6 和一个 9）上训练数字 8 的检测器。假设我们制作了两个不同的重采样数据集。Bagging训练程序通过有放回采样构建这些数据集。第一个数据集忽略 9 并重复 8。在这个数据集上，检测器得知数字顶部有一个环就对应于一个 8。第二个数据集中，我们忽略 6 并重复 9。在这种情况下，检测器得知数字底部有一个环就对应于一个 8。这些单独的分类规则中的每一个都是不可靠的，但如果我们平均它们的输出，就能得到鲁棒的检测器，只有当 8 的两个环都存在时才能实现最大置信度。

神经网络能找到足够多的不同的解，意味着他们可以从模型平均中受益 (即使所有模型都在同一数据集上训练)。神经网络中随机初始化的差异、小批量的随机选择、超参数的差异或不同输出的非确定性实现往往足以使得集成中的不同成员具有部分独立的误差。

模型平均是一个减少泛化误差的非常强大可靠的方法。在作为科学论文算法的

基准时，它通常是不鼓励使用的，因为任何机器学习算法都可以从模型平均中大幅获益（以增加计算和存储为代价）。

机器学习比赛中的取胜算法通常是使用超过几十种模型平均的方法。最近一个突出的例子是Netflix Grand Prize(Koren, 2009)。

不是所有构建集成的技术都是为了让集成模型比单一模型更加正则化。例如，一种被称为 **Boosting** 的技术 (Freund and Schapire, 1996b,a) 构建比单个模型容量更高的集成模型。通过向集成逐步添加神经网络，Boosting已经被应用于构建神经网络的集成(Schwenk and Bengio, 1998)。通过逐渐增加神经网络的隐藏单元，Boosting也可以将单个神经网络解释为一个集成。

7.12 Dropout

Dropout (Srivastava *et al.*, 2014) 提供了正则化一大类模型的方法，计算方便但功能强大。在第一种近似下，Dropout可以被认为是集成大量深层神经网络的实用Bagging方法。Bagging涉及训练多个模型，并在每个测试样本上评估多个模型。当每个模型都是一个很大的神经网络时，这似乎是不切实际的，因为训练和评估这样的网络需要花费很多运行时间和内存。通常我们只能集成五至十个神经网络，如Szegedy *et al.* (2014a)集成了六个神经网络赢得 ILSVRC，超过这个数量就会迅速变得难以处理。Dropout提供了一种廉价的Bagging集成近似，能够训练和评估指数级数量的神经网络。

具体而言，Dropout训练的集成包括所有从基础网络除去非输出单元后形成的子网络，如图 7.6 所示。最先进的神经网络基于一系列仿射变换和非线性变换，我们只需将一些单元的输出乘零就能有效地删除一个单元。这个过程需要对模型（如径向基函数网络，单元的状态和参考值之间存在一定区别）进行一些修改。为了简单起见，我们在这里提出乘零的简单Dropout算法，但是它被简单修改后，可以与从网络中移除单元的其他操作结合使用。

回想一下Bagging学习，我们定义 k 个不同的模型，从训练集有放回采样构造 k 个不同的数据集，然后在训练集 i 上训练模型 i。Dropout的目标是在指数级数量的神经网络上近似这个过程。具体来说，在训练中使用Dropout时，我们会使用基于小批量产生较小步长的学习算法，如随机梯度下降等。我们每次在小批量中加载一个样本，然后随机抽样应用于网络中所有输入和隐藏单元的不同二值掩码。对于

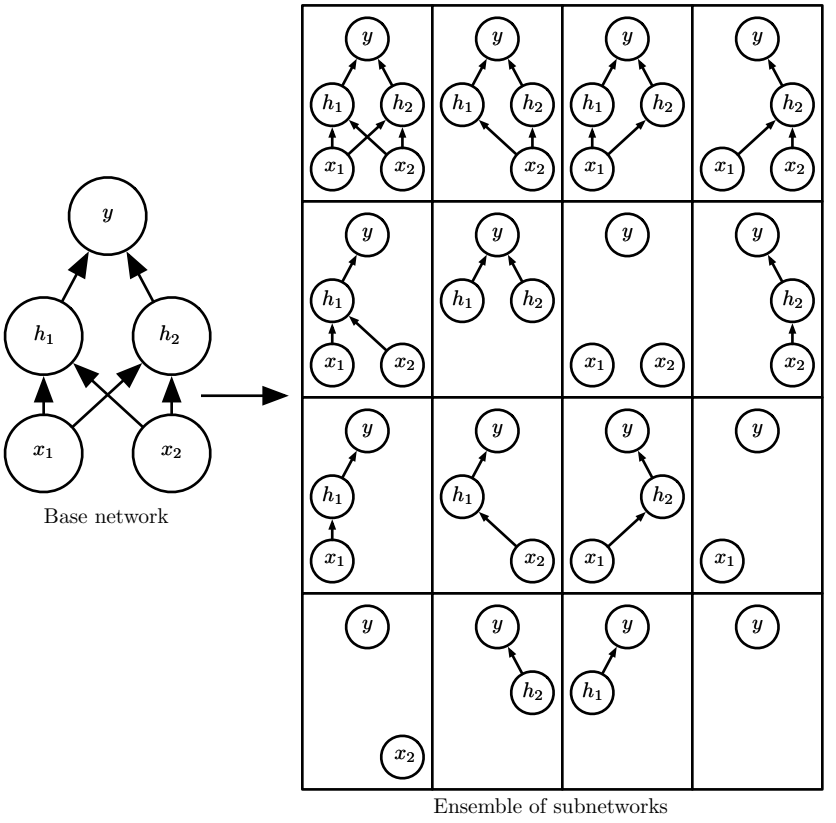

图 7.6: Dropout训练由所有子网络组成的集成，其中子网络通过从基本网络中删除非输出单元构建。我们从具有两个可见单元和两个隐藏单元的基本网络开始。这四个单元有十六个可能的子集。右图展示了从原始网络中丢弃不同的单元子集而形成的所有十六个子网络。在这个小例子中，所得到的大部分网络没有输入单元或没有从输入连接到输出的路径。当层较宽时，丢弃所有从输入到输出的可能路径的概率变小，所以这个问题不太可能在出现层较宽的网络中。

每个单元，掩码是独立采样的。掩码值为 1 的采样概率（导致包含一个单元）是训练开始前一个固定的超参数。它不是模型当前参数值或输入样本的函数。通常在每一个小批量训练的神经网络中，一个输入单元被包括的概率为 0.8，一个隐藏单元被包括的概率为 0.5。然后，我们运行和之前一样的前向传播、反向传播以及学习更新。图 7.7 说明了在Dropout下的前向传播。

更正式地说，假设一个掩码向量 $\boldsymbol{\mu}$ 指定被包括的单元，$J(\boldsymbol{\theta}, \boldsymbol{\mu})$ 是由参数 $\boldsymbol{\theta}$ 和掩码 $\boldsymbol{\mu}$ 定义的模型代价。那么Dropout训练的目标是最小化 $\mathbb{E}_{\boldsymbol{\mu}} J(\boldsymbol{\theta}, \boldsymbol{\mu})$。这个期望包含

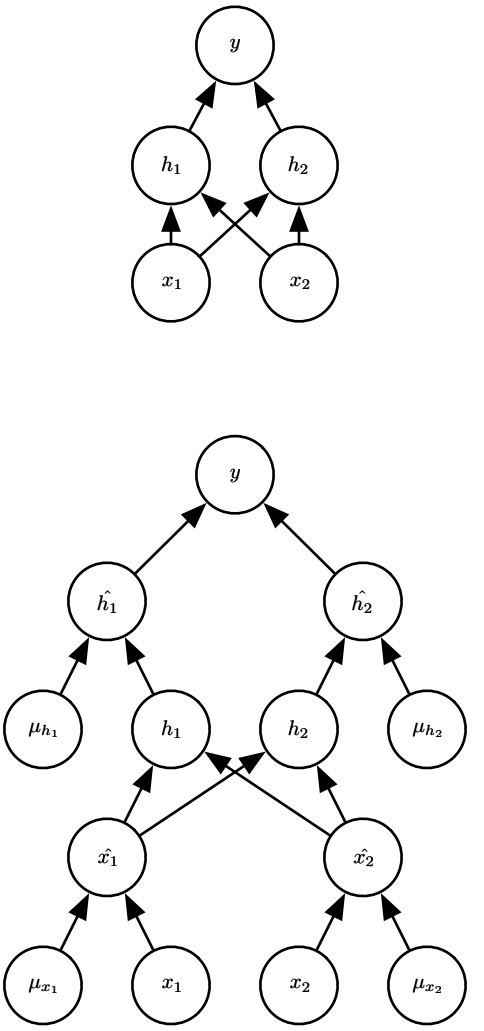

图 7.7: 在使用Dropout的前馈网络中前向传播的示例。(顶部) 在此示例中，我们使用具有两个输入单元，具有两个隐藏单元的隐藏层以及一个输出单元的前馈网络。(底部) 为了执行具有Dropout的前向传播，我们随机地对向量 $\boldsymbol{\mu}$ 进行采样，其中网络中的每个输入或隐藏单元对应一项。$\boldsymbol{\mu}$ 中的每项都是二值的且独立于其他项采样。超参数的采样概率为 1，隐藏层的采样概率通常为 0.5，输入的采样概率通常为 0.8。网络中的每个单元乘以相应的掩码，然后正常地继续沿着网络的其余部分前向传播。这相当于从图 7.6 中随机选择一个子网络并沿着前向传播。

多达指数级的项，但我们可以通过抽样 $\boldsymbol{\mu}$ 获得梯度的无偏估计。

Dropout训练与Bagging训练不太一样。在Bagging的情况下，所有模型都是独立

的。在Dropout的情况下，所有模型共享参数，其中每个模型继承父神经网络参数的不同子集。参数共享使得在有限可用的内存下表示指数级数量的模型变得可能。在Bagging的情况下，每一个模型在其相应训练集上训练到收敛。在Dropout的情况下，通常大部分模型都没有显式地被训练，因为通常父神经网络会很大，以致于到宇宙毁灭都不可能采样完所有的子网络。取而代之的是，在单个步骤中我们训练一小部分的子网络，参数共享会使得剩余的子网络也能有好的参数设定。这些是仅有的区别。除了这些，Dropout与Bagging算法一样。例如，每个子网络中遇到的训练集确实是有放回采样的原始训练集的一个子集。

Bagging集成必须根据所有成员的累积投票做一个预测。在这种背景下，我们将这个过程称为**推断**（inference）。目前为止，我们在介绍Bagging和Dropout时没有要求模型具有明确的概率。现在，我们假定该模型的作用是输出一个概率分布。在Bagging的情况下，每个模型 i 产生一个概率分布 $p^{(i)}(y \mid \boldsymbol{x})$。集成的预测由这些分布的算术平均值给出，

$$\frac{1}{k}\sum_{i=1}^{k} p^{(i)}(y \mid \boldsymbol{x}). \tag{7.52}$$

在Dropout的情况下，通过掩码 $\boldsymbol{\mu}$ 定义每个子模型的概率分布 $p(y \mid \boldsymbol{x}, \boldsymbol{\mu})$。所有掩码的算术平均值由下式给出

$$\sum_{\boldsymbol{\mu}} p(\boldsymbol{\mu}) p(y \mid \boldsymbol{x}, \boldsymbol{\mu}), \tag{7.53}$$

其中 $p(\boldsymbol{\mu})$ 是训练时采样 $\boldsymbol{\mu}$ 的概率分布。

因为这个求和包含多达指数级的项，除非该模型的结构允许某种形式的简化，否则是不可能计算的。目前为止，无法得知深度神经网络是否允许某种可行的简化。相反，我们可以通过采样近似推断，即平均许多掩码的输出。即使是 10 – 20 个掩码就足以获得不错的表现。

然而，一个更好的方法能不错地近似整个集成的预测，且只需一个前向传播的代价。要做到这一点，我们改用集成成员预测分布的几何平均而不是算术平均。Warde-Farley *et al.* (2014) 提出的论点和经验证据表明，在这个情况下几何平均与算术平均表现得差不多。

多个概率分布的几何平均不能保证是一个概率分布。为了保证结果是一个概率分布，我们要求没有子模型给某一事件分配概率 0，并重新标准化所得分布。通过几

何平均直接定义的非标准化概率分布由下式给出

$$\tilde{p}_{\text{ensemble}}(y \mid \boldsymbol{x}) = \sqrt[2^d]{\prod_{\boldsymbol{\mu}} p(y \mid \boldsymbol{x}, \boldsymbol{\mu})}, \tag{7.54}$$

其中 d 是可被丢弃的单元数。这里为简化介绍，我们使用均匀分布的 $\boldsymbol{\mu}$，但非均匀分布也是可以的。为了作出预测，我们必须重新标准化集成：

$$p_{\text{ensemble}}(y \mid \boldsymbol{x}) = \frac{\tilde{p}_{\text{ensemble}}(y \mid \boldsymbol{x})}{\sum_{y'} \tilde{p}_{\text{ensemble}}(y' \mid \boldsymbol{x})}. \tag{7.55}$$

涉及Dropout的一个重要观点 (Hinton *et al.*, 2012c) 是，我们可以通过评估模型中 $p(y \mid \boldsymbol{x})$ 来近似 p_{ensemble}：该模型具有所有单元，但我们将单元 i 的输出的权重乘以单元 i 的被包含概率。这个修改的动机是得到从该单元输出的正确期望值。我们把这种方法称为 **权重比例推断规则**（weight scaling inference rule）。目前还没有在深度非线性网络上对这种近似推断规则的准确性作任何理论分析，但经验上表现得很好。

因为我们通常使用 $\frac{1}{2}$ 的包含概率，权重比例规则一般相当于在训练结束后将权重除 2，然后像平常一样使用模型。实现相同结果的另一种方法是在训练期间将单元的状态乘 2。无论哪种方式，我们的目标是确保在测试时一个单元的期望总输入与在训练时该单元的期望总输入是大致相同的（即使近半单位在训练时丢失）。

对许多不具有非线性隐藏单元的模型族而言，权重比例推断规则是精确的。举个简单的例子，考虑softmax 函数回归分类，其中由向量 $\mathbf{v}$ 表示 n 个输入变量：

$$P(\mathrm{y} = y \mid \mathbf{v}) = \operatorname{softmax}\big(\boldsymbol{W}^\top \mathbf{v} + \boldsymbol{b}\big)_y. \tag{7.56}$$

我们可以根据二值向量 $\boldsymbol{d}$ 逐元素的乘法将一类子模型进行索引：

$$P(\mathrm{y} = y \mid \mathbf{v}; \boldsymbol{d}) = \operatorname{softmax}\big(\boldsymbol{W}^\top (\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}\big)_y. \tag{7.57}$$

集成预测器被定义为重新标准化所有集成成员预测的几何平均：

$$P_{\text{ensemble}}(\mathrm{y} = y \mid \mathbf{v}) = \frac{\tilde{P}_{\text{ensemble}}(\mathrm{y} = y \mid \mathbf{v})}{\sum_{y'} \tilde{P}_{\text{ensemble}}(\mathrm{y} = y' \mid \mathbf{v})}, \tag{7.58}$$

其中

$$\tilde{P}_{\text{ensemble}}(\mathrm{y} = y \mid \mathbf{v}) = \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} P(\mathrm{y} = y \mid \mathbf{v}; \boldsymbol{d})}. \tag{7.59}$$

为了证明权重比例推断规则是精确的，我们简化 $\tilde{P}_{\text{ensemble}}$：

$$\tilde{P}_{\text{ensemble}}(\mathrm{y} = y \mid \mathbf{v}) = \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} P(\mathrm{y} = y \mid \mathbf{v}; \boldsymbol{d})} \tag{7.60}$$

$$= \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} \text{softmax}(\boldsymbol{W}^\top (\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b})_y} \tag{7.61}$$

$$= \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} \frac{\exp(\boldsymbol{W}^\top_{y,:}(\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}_y)}{\sum_{y'} \exp(\boldsymbol{W}^\top_{y',:}(\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}_{y'})}} \tag{7.62}$$

$$= \frac{\sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} \exp(\boldsymbol{W}^\top_{y,:}(\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}_y)}}{\sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} \sum_{y'} \exp(\boldsymbol{W}^\top_{y',:}(\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}_{y'})}} \tag{7.63}$$

由于 $\tilde{P}$ 将被标准化，我们可以放心地忽略那些相对 y 不变的乘法：

$$\tilde{P}_{\text{ensemble}}(\mathrm{y} = y \mid \mathbf{v}) \propto \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} \exp(\boldsymbol{W}^\top_{y,:}(\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}_y)} \tag{7.64}$$

$$= \exp\left(\frac{1}{2^n} \sum_{\boldsymbol{d} \in \{0,1\}^n} \boldsymbol{W}^\top_{y,;}(\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}_y\right) \tag{7.65}$$

$$= \exp\left(\frac{1}{2} \boldsymbol{W}^\top_{y,:}\mathbf{v} + \boldsymbol{b}_y\right). \tag{7.66}$$

将其代入式 (7.58)，我们得到了一个权重为 $\frac{1}{2}\boldsymbol{W}$ 的softmax 函数分类器。

权重比例推断规则在其他设定下也是精确的，包括条件正态输出的回归网络以及那些隐藏层不包含非线性的深度网络。然而，权重比例推断规则对具有非线性的深度模型仅仅是一个近似。虽然这个近似尚未有理论上的分析，但在实践中往往效果很好。Goodfellow *et al.* (2013b) 实验发现，在对集成预测的近似方面，权重比例推断规则比蒙特卡罗近似更好（就分类精度而言）。即使允许蒙特卡罗近似采样多达 1000 子网络时也比不过权重比例推断规则。Gal and Ghahramani (2015) 发现一些模型可以通过二十个样本和蒙特卡罗近似获得更好的分类精度。似乎推断近似的最佳选择是与问题相关的。

Srivastava *et al.* (2014) 显示，Dropout比其他标准的计算开销小的正则化方法（如权重衰减、过滤器范数约束和稀疏激活的正则化）更有效。Dropout也可以与其他形式的正则化合并，得到进一步的提升。

计算方便是Dropout的一个优点。训练过程中使用Dropout产生 n 个随机二进制数与状态相乘，每个样本每次更新只需 $\mathcal{O}(n)$ 的计算复杂度。根据实现，也可能需要

$\mathcal{O}(n)$ 的存储空间来持续保存这些二进制数（直到反向传播阶段）。使用训练好的模型推断时，计算每个样本的代价与不使用Dropout是一样的，尽管我们必须在开始运行推断前将权重除以 2。

Dropout的另一个显著优点是不怎么限制适用的模型或训练过程。几乎在所有使用分布式表示且可以用随机梯度下降训练的模型上都表现很好。包括前馈神经网络、概率模型，如受限玻尔兹曼机(Srivastava *et al.*, 2014)，以及循环神经网络(Bayer and Osendorfer, 2014; Pascanu *et al.*, 2014a)。许多效果差不多的其他正则化策略对模型结构的限制更严格。

虽然Dropout在特定模型上每一步的代价是微不足道的，但在一个完整的系统上使用Dropout的代价可能非常显著。因为Dropout是一个正则化技术，它减少了模型的有效容量。为了抵消这种影响，我们必须增大模型规模。不出意外的话，使用Dropout时最佳验证集的误差会低很多，但这是以更大的模型和更多训练算法的迭代次数为代价换来的。对于非常大的数据集，正则化带来的泛化误差减少得很小。在这些情况下，使用Dropout和更大模型的计算代价可能超过正则化带来的好处。

只有极少的训练样本可用时，Dropout不会很有效。在只有不到 5000 的样本的Alternative Splicing数据集上 (Xiong *et al.*, 2011)，贝叶斯神经网络 (Neal, 1996) 比Dropout表现得更好 (Srivastava *et al.*, 2014)。当有其他未分类的数据可用时，无监督特征学习也比Dropout更有优势。

Wager *et al.* (2013) 表明，当Dropout作用于线性回归时，相当于每个输入特征具有不同权重衰减系数的 L^2权重衰减。每个特征的权重衰减系数的大小是由其方差来确定的。其他线性模型也有类似的结果。而对于深度模型而言，Dropout与权重衰减是不等同的。

使用Dropout训练时的随机性不是这个方法成功的必要条件。它仅仅是近似所有子模型总和的一个方法。Wang and Manning (2013) 导出了近似这种边缘分布的解析解。他们的近似被称为 **快速 Dropout**（fast dropout），减小梯度计算中的随机性而获得更快的收敛速度。这种方法也可以在测试时应用，能够比权重比例推断规则更合理地（但计算也更昂贵）近似所有子网络的平均。快速 Dropout在小神经网络上的性能几乎与标准的Dropout相当，但在大问题上尚未产生显著改善或尚未应用。

随机性对实现Dropout的正则化效果不是必要的，同时也不是充分的。为了证明这一点，Warde-Farley *et al.* (2014) 使用一种被称为 **Dropout Boosting** 的方法设计了一个对照实验，具有与传统Dropout方法完全相同的噪声掩码，但缺乏正则化效

果。Dropout Boosting训练整个集成以最大化训练集上的似然。从传统Dropout类似于Bagging的角度来看，这种方式类似于Boosting。如预期一样，和单一模型训练整个网络相比，Dropout Boosting几乎没有正则化效果。这表明，使用Bagging解释Dropout比使用稳健性噪声解释Dropout更好。只有当随机抽样的集成成员相互独立地训练好后，才能达到Bagging集成的正则化效果。

Dropout启发其他以随机方法训练指数量级的共享权重的集成。DropConnect是Dropout的一个特殊情况，其中一个标量权重和单个隐藏单元状态之间的每个乘积被认为是可以丢弃的一个单元 (Wan *et al.*, 2013)。随机池化是构造卷积神经网络集成的一种随机化池化的形式 (见第 9.3 节)，其中每个卷积网络参与每个特征图的不同空间位置。目前为止，Dropout仍然是最广泛使用的隐式集成方法。

一个关于Dropout的重要见解是，通过随机行为训练网络并平均多个随机决定进行预测，实现了一种参数共享的Bagging形式。早些时候，我们将Dropout描述为通过包括或排除单元形成模型集成的Bagging。然而，这种参数共享策略不一定要基于包括和排除。原则上，任何一种随机的修改都是可接受的。在实践中，我们必须选择让神经网络能够学习对抗的修改类型。在理想情况下，我们也应该使用可以快速近似推断的模型族。我们可以认为由向量 $\boldsymbol{\mu}$ 参数化的任何形式的修改，是对 $\boldsymbol{\mu}$ 所有可能的值训练 $p(y \mid \boldsymbol{x}, \boldsymbol{\mu})$ 的集成。注意，这里不要求 $\boldsymbol{\mu}$ 具有有限数量的值。例如，$\boldsymbol{\mu}$ 可以是实值。Srivastava *et al.* (2014) 表明，权重乘以 $\boldsymbol{\mu} \sim \mathcal{N}(\mathbf{1}, \boldsymbol{I})$ 比基于二值掩码Dropout表现得更好。由于 $\mathbb{E}[\boldsymbol{\mu}] = 1$，标准网络自动实现集成的近似推断，而不需要权重比例推断规则。

目前为止，我们将Dropout介绍为一种纯粹高效近似Bagging的方法。然而，还有比这更进一步的Dropout观点。Dropout不仅仅是训练一个Bagging的集成模型，并且是共享隐藏单元的集成模型。这意味着无论其他隐藏单元是否在模型中，每个隐藏单元必须都能够表现良好。隐藏单元必须准备好进行模型之间的交换和互换。Hinton *et al.* (2012d) 由生物学的想法受到启发：有性繁殖涉及到两个不同生物体之间交换基因，进化产生的压力使得基因不仅是良好的而且要准备好不同有机体之间的交换。这样的基因和这些特点对环境的变化是非常稳健的，因为它们一定会正确适应任何一个有机体或模型不寻常的特性。因此Dropout正则化每个隐藏单元不仅是一个很好的特征，更要在许多情况下是良好的特征。Warde-Farley *et al.* (2014) 将Dropout与大集成的训练相比并得出结论：相比独立模型集成获得泛化误差改进，Dropout会带来额外的改进。

Dropout强大的大部分原因来自施加到隐藏单元的掩码噪声，了解这一事实是重

要的。这可以看作是对输入内容的信息高度智能化、自适应破坏的一种形式，而不是对输入原始值的破坏。例如，如果模型学得通过鼻检测脸的隐藏单元 h_i，那么丢失 h_i 对应于擦除图像中有鼻子的信息。模型必须学习另一种 h_i，要么是鼻子存在的冗余编码，要么是像嘴这样的脸部的另一特征。传统的噪声注入技术，在输入端加非结构化的噪声不能够随机地从脸部图像中抹去关于鼻子的信息，除非噪声的幅度大到几乎能抹去图像中所有的信息。破坏提取的特征而不是原始值，让破坏过程充分利用该模型迄今获得的关于输入分布的所有知识。

Dropout的另一个重要方面是噪声是乘性的。如果是固定规模的加性噪声，那么加了噪声 ϵ 的整流线性隐藏单元可以简单地学会使 h_i 变得很大（使增加的噪声 ϵ 变得不显著）。乘性噪声不允许这样病态地解决噪声鲁棒性问题。

另一种深度学习算法——批标准化，在训练时向隐藏单元引入加性和乘性噪声重新参数化模型。批标准化的主要目的是改善优化，但噪声具有正则化的效果，有时没必要再使用Dropout。批标准化将会在第 8.7.1 节中被更详细地讨论。

7.13 对抗训练

在许多情况下，神经网络在独立同分布的测试集上进行评估已经达到了人类表现。因此，我们自然要怀疑这些模型在这些任务上是否获得了真正的人类层次的理解。为了探索网络对底层任务的理解层次，我们可以探索这个模型错误分类的例子。Szegedy *et al.* (2014b) 发现，在精度达到人类水平的神经网络上通过优化过程故意构造数据点，其上的误差率接近100%，模型在这个输入点 $\boldsymbol{x}'$ 的输出与附近的数据点 $\boldsymbol{x}$ 非常不同。在许多情况下，$\boldsymbol{x}'$ 与 $\boldsymbol{x}$ 非常近似，人类观察者不会察觉原始样本和**对抗样本**（adversarial example）之间的差异，但是网络会作出非常不同的预测。见图 7.8 中的例子。

对抗样本在很多领域有很多影响，例如计算机安全，这超出了本章的范围。然而，它们在正则化的背景下很有意思，因为我们可以通过**对抗训练**（adversarial training）减少原有独立同分布的测试集的错误率——在对抗扰动的训练集样本上训练网络 (Szegedy *et al.*, 2014b; Goodfellow *et al.*, 2014b)。

Goodfellow *et al.* (2014b) 表明，这些对抗样本的主要原因之一是过度线性。神经网络主要是基于线性块构建的。因此在一些实验中，它们实现的整体函数被证明是高度线性的。这些线性函数很容易优化。不幸的是，如果一个线性函数具有许多

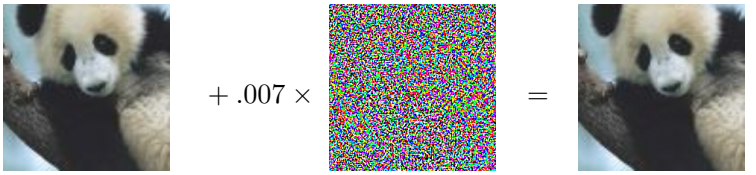

图 7.8: 在 ImageNet 上应用 GoogLeNet (Szegedy *et al.*, 2014a) 的对抗样本生成的演示。通过添加一个不可察觉的小向量（其中元素等于代价函数相对于输入的梯度元素的符号），我们可以改变 GoogLeNet 对此图像的分类结果。经Goodfellow *et al.* (2014b) 许可转载。

输入，那么它的值可以非常迅速地改变。如果我们用 ϵ 改变每个输入，那么权重为 $\boldsymbol{w}$ 的线性函数可以改变 $\epsilon \|\boldsymbol{w}\|_1$ 之多，如果 $\boldsymbol{w}$ 是高维的这会是一个非常大的数。对抗训练通过鼓励网络在训练数据附近的局部区域恒定来限制这一高度敏感的局部线性行为。这可以被看作是一种明确地向监督神经网络引入局部恒定先验的方法。

对抗训练有助于体现积极正则化与大型函数族结合的力量。纯粹的线性模型，如逻辑回归，由于它们被限制为线性而无法抵抗对抗样本。神经网络能够将函数从接近线性转化为局部近似恒定，从而可以灵活地捕获到训练数据中的线性趋势同时学习抵抗局部扰动。

对抗样本也提供了一种实现半监督学习的方法。在与数据集中的标签不相关联的点 $\boldsymbol{x}$ 处，模型本身为其分配一些标签 $\hat{y}$。模型的标记 $\hat{y}$ 未必是真正的标签，但如果模型是高品质的，那么 $\hat{y}$ 提供正确标签的可能性很大。我们可以搜索一个对抗样本 $\boldsymbol{x}'$，导致分类器输出一个标签 y' 且 $y' \neq \hat{y}$。不使用真正的标签，而是由训练好的模型提供标签产生的对抗样本被称为 **虚拟对抗样本**（virtual adversarial example）(Miyato *et al.*, 2015)。我们可以训练分类器为 $\boldsymbol{x}$ 和 $\boldsymbol{x}'$ 分配相同的标签。这鼓励分类器学习一个沿着未标签数据所在流形上任意微小变化都很鲁棒的函数。驱动这种方法的假设是，不同的类通常位于分离的流形上，并且小扰动不会使数据点从一个类的流形跳到另一个类的流形上。

7.14 切面距离、正切传播和流形正切分类器

如第 5.11.3 节所述，许多机器学习通过假设数据位于低维流形附近来克服维数灾难。

一个利用流形假设的早期尝试是**切面距离**（tangent distance）算法 (Simard *et al.*, 1993, 1998)。它是一种非参数的最近邻算法，其中使用的度量不是通用的欧几里德距离，而是根据邻近流形关于聚集概率的知识导出的。这个算法假设我们尝试分类的样本和同一流形上的样本具有相同的类别。由于分类器应该对局部因素（对应于流形上的移动）的变化保持不变，一种合理的度量是将点 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$ 各自所在流形 M_1 和 M_2 的距离作为点 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$ 之间的最近邻距离。然而这可能在计算上是困难的（它需要解决一个寻找 M_1 和 M_2 最近点对的优化问题），一种局部合理的廉价替代是使用 $\boldsymbol{x}_i$ 点处切平面近似 M_i，并测量两条切平面或一个切平面和点之间的距离。这可以通过求解一个低维线性系统（就流形的维数而言）来实现。当然，这种算法需要指定那些切向量。

受相关启发，**正切传播**（tangent prop）算法 (Simard *et al.*, 1992)（图 7.9）训练带有额外惩罚的神经网络分类器，使神经网络的每个输出 $f(\boldsymbol{x})$ 对已知的变化因素是局部不变的。这些变化因素对应于沿着的相同样本聚集的流形的移动。这里实现局部不变性的方法是要求 $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ 与已知流形的切向 $\boldsymbol{v}^{(i)}$ 正交，或者等价地通过正则化惩罚 Ω 使 f 在 $\boldsymbol{x}$ 的 $\boldsymbol{v}^{(i)}$ 方向的导数较小：

$$\Omega(f) = \sum_i \left((\nabla_{\boldsymbol{x}} f(\boldsymbol{x})^\top \boldsymbol{v}^{(i)}) \right)^2. \tag{7.67}$$

这个正则化项当然可以通过适当的超参数缩放，并且对于大多数神经网络，我们需要对许多输出求和 (此处为描述简单，$f(\boldsymbol{x})$ 为唯一输出)。与切面距离算法一样，我们根据切向量推导先验，通常从变换（如平移、旋转和缩放图像）的效果获得形式知识。正切传播不仅用于监督学习(Simard *et al.*, 1992)，还在强化学习(Thrun, 1995)中有所应用。

正切传播与数据集增强密切相关。在这两种情况下，该算法的用户通过指定一组应当不会改变网络输出的转换，将其先验知识编码至算法中。不同的是在数据集增强的情况下，网络显式地训练正确分类这些施加大量变换后产生的不同输入。正切传播不需要显式访问一个新的输入点。取而代之，它解析地对模型正则化从而在指定转换的方向抵抗扰动。虽然这种解析方法是聪明优雅的，但是它有两个主要的缺点。首先，模型的正则化只能抵抗无穷小的扰动。显式的数据集增强能抵抗较大的扰

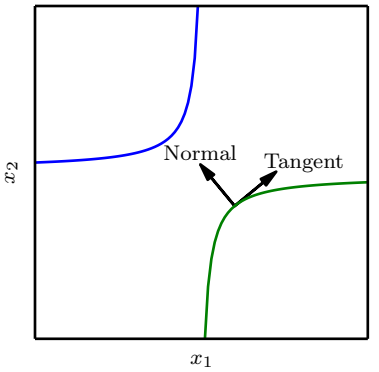

图 7.9: 正切传播算法 (Simard *et al.*, 1992) 和流形正切分类器主要思想的示意图 (Rifai *et al.*, 2011c)，它们都正则化分类器的输出函数 $f(\boldsymbol{x})$。每条曲线表示不同类别的流形，这里表示嵌入二维空间中的一维流形。在一条曲线上，我们选择单个点并绘制一个与类别流形（平行并接触流形）相切的向量以及与类别流形（与流形正交）垂直的向量。在多维情况下，可以存在许多切线方向和法线方向。我们希望分类函数在垂直于流形方向上快速改变，并且在类别流形的方向上保持不变。正切传播和流形正切分类器都会正则化 $f(\boldsymbol{x})$，使其不随 $\boldsymbol{x}$ 沿流形的移动而剧烈变化。正切传播需要用户手动指定正切方向的计算函数（例如指定小平移后的图像保留在相同类别的流形中），而流形正切分类器通过训练自编码器拟合训练数据来估计流形的正切方向。我们将在第十四章中讨论使用自编码器来估计流形。

动。其次，我们很难在基于整流线性单元的模型上使用无限小的方法。这些模型只能通过关闭单元或缩小它们的权重才能缩小它们的导数。它们不能像sigmoid或tanh单元一样通过较大权重在高值处饱和以收缩导数。数据集增强在整流线性单元上工作得很好，因为不同的整流单元会在每一个原始输入的不同转换版本上被激活。

正切传播也和双反向传播(Drucker and LeCun, 1992) 以及对抗训练(Szegedy *et al.*, 2014b; Goodfellow *et al.*, 2014b) 有关联。双反向传播正则化使Jacobian矩阵偏小，而对抗训练找到原输入附近的点，训练模型在这些点上产生与原来输入相同的输出。正切传播和手动指定转换的数据集增强都要求模型在输入变化的某些特定的方向上保持不变。双反向传播和对抗训练都要求模型对输入所有方向中的变化（只要该变化较小）都应当保持不变。正如数据集增强是正切传播非无限小的版本，对抗训练是双反向传播非无限小的版本。

流形正切分类器 (Rifai *et al.*, 2011d) 无需知道切线向量的先验。我们将在第十四章看到，自编码器可以估算流形的切向量。流形正切分类器使用这种技术来避免