第四章　数值计算

机器学习算法通常需要大量的数值计算。这通常是指通过迭代过程更新解的估计值来解决数学问题的算法，而不是通过解析过程推导出公式来提供正确解的方法。常见的操作包括优化（找到最小化或最大化函数值的参数）和线性方程组的求解。对数字计算机来说实数无法在有限内存下精确表示，因此仅仅是计算涉及实数的函数也是困难的。

4.1　上溢和下溢

连续数学在数字计算机上的根本困难是，我们需要通过有限数量的位模式来表示无限多的实数。这意味着我们在计算机中表示实数时，几乎总会引入一些近似误差。在许多情况下，这仅仅是舍入误差。舍入误差会导致一些问题，特别是当许多操作复合时，即使是理论上可行的算法，如果在设计时没有考虑最小化舍入误差的累积，在实践时也可能会导致算法失效。

一种极具毁灭性的舍入误差是**下溢**（underflow）。当接近零的数被四舍五入为零时发生下溢。许多函数在其参数为零而不是一个很小的正数时才会表现出质的不同。例如，我们通常要避免被零除（一些软件环境将在这种情况下抛出异常，有些会返回一个非数字 (not-a-number, NaN) 的占位符）或避免取零的对数（这通常被视为 $-\infty$，进一步的算术运算会使其变成非数字）。

另一个极具破坏力的数值错误形式是**上溢**（overflow）。当大量级的数被近似为 ∞ 或 $-\infty$ 时发生上溢。进一步的运算通常会导致这些无限值变为非数字。

必须对上溢和下溢进行数值稳定的一个例子是**softmax 函数**（softmax func-

tion)。softmax 函数经常用于预测与 Multinoulli 分布相关联的概率，定义为

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}. \tag{4.1}$$

考虑一下当所有 x_i 都等于某个常数 c 时会发生什么。从理论分析上说，我们可以发现所有的输出都应该为 $\frac{1}{n}$。从数值计算上说，当 c 量级很大时，这可能不会发生。如果 c 是很小的负数，$\exp(c)$ 就会下溢。这意味着 softmax 函数的分母会变成 0，所以最后的结果是未定义的。当 c 是非常大的正数时，$\exp(c)$ 的上溢再次导致整个表达式未定义。这两个困难能通过计算 $\text{softmax}(\boldsymbol{z})$ 同时解决，其中 $\boldsymbol{z} = \boldsymbol{x} - \max_i x_i$。简单的代数计算表明，softmax 解析上的函数值不会因为从输入向量减去或加上标量而改变。减去 $\max_i x_i$ 导致 exp 的最大参数为 0，这排除了上溢的可能性。同样地，分母中至少有一个值为 1 的项，这就排除了因分母下溢而导致被零除的可能性。

还有一个小问题。分子中的下溢仍可以导致整体表达式被计算为零。这意味着，如果我们在计算 $\log \text{softmax}(\boldsymbol{x})$ 时，先计算 softmax 再把结果传给 log 函数，会错误地得到 $-\infty$。相反，我们必须实现一个单独的函数，并以数值稳定的方式计算 $\log \text{softmax}$。我们可以使用相同的技巧来稳定 $\log \text{softmax}$ 函数。

在大多数情况下，我们没有明确地对本书描述的各种算法所涉及的数值考虑进行详细说明。底层库的开发者在实现深度学习算法时应该牢记数值问题。本书的大多数读者可以简单地依赖保证数值稳定的底层库。在某些情况下，我们有可能在实现一个新的算法时自动保持数值稳定。Theano (Bergstra *et al.*, 2010a; Bastien *et al.*, 2012a) 就是这样软件包的一个例子，它能自动检测并稳定深度学习中许多常见的数值不稳定的表达式。

4.2 病态条件

条件数表征函数相对于输入的微小变化而变化的快慢程度。输入被轻微扰动而迅速改变的函数对于科学计算来说可能是有问题的，因为输入中的舍入误差可能导致输出的巨大变化。

考虑函数 $f(\boldsymbol{x}) = \boldsymbol{A}^{-1}\boldsymbol{x}$。当 $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ 具有特征值分解时，其条件数为

$$\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|. \tag{4.2}$$

这是最大和最小特征值的模之比[1]。当该数很大时，矩阵求逆对输入的误差特别敏感。

这种敏感性是矩阵本身的固有特性，而不是矩阵求逆期间舍入误差的结果。即使我们乘以完全正确的矩阵逆，病态条件的矩阵也会放大预先存在的误差。在实践中，该错误将与求逆过程本身的数值误差进一步复合。

4.3 基于梯度的优化方法

大多数深度学习算法都涉及某种形式的优化。优化指的是改变 $\boldsymbol{x}$ 以最小化或最大化某个函数 $f(\boldsymbol{x})$ 的任务。我们通常以最小化 $f(\boldsymbol{x})$ 指代大多数最优化问题。最大化可经由最小化算法最小化 $-f(\boldsymbol{x})$ 来实现。

我们把要最小化或最大化的函数称为**目标函数**（objective function）或**准则**（criterion）。当我们对其进行最小化时，我们也把它称为**代价函数**（cost function）、**损失函数**（loss function）或**误差函数**（error function）。虽然有些机器学习著作赋予这些名称特殊的意义，但在这本书中我们交替使用这些术语。

我们通常使用一个上标 $*$ 表示最小化或最大化函数的 $\boldsymbol{x}$ 值。如我们记 $\boldsymbol{x}^* = \arg\min f(\boldsymbol{x})$。

我们假设读者已经熟悉微积分，这里简要回顾微积分概念如何与优化联系。

假设我们有一个函数 $y = f(x)$，其中 x 和 y 是实数。这个函数的**导数**（derivative）记为 $f'(x)$ 或 $\frac{dy}{dx}$。导数 $f'(x)$ 代表 $f(x)$ 在点 x 处的斜率。换句话说，它表明如何缩放输入的小变化才能在输出获得相应的变化：$f(x+\epsilon) \approx f(x) + \epsilon f'(x)$。

因此导数对于最小化一个函数很有用，因为它告诉我们如何更改 x 来略微地改善 y。例如，我们知道对于足够小的 ϵ 来说，$f(x - \epsilon \text{sign}(f'(x)))$ 是比 $f(x)$ 小的。因此我们可以将 x 往导数的反方向移动一小步来减小 $f(x)$。这种技术被称为**梯度下降**（gradient descent）(Cauchy, 1847)。图 4.1 展示了一个例子。

当 $f'(x) = 0$，导数无法提供往哪个方向移动的信息。$f'(x) = 0$ 的点称为**临界点**（critical point）或**驻点**（stationary point）。一个**局部极小点**（local minimum）意味着这个点的 $f(x)$ 小于所有邻近点，因此不可能通过移动无穷小的步长来减小 $f(x)$。一个**局部极大点**（local maximum）意味着这个点的 $f(x)$ 大于所有邻近点，因此不可能通过移动无穷小的步长来增大 $f(x)$。有些临界点既不是最小点也不是最大

[1] 译者注：与通常的条件数定义有所不同。

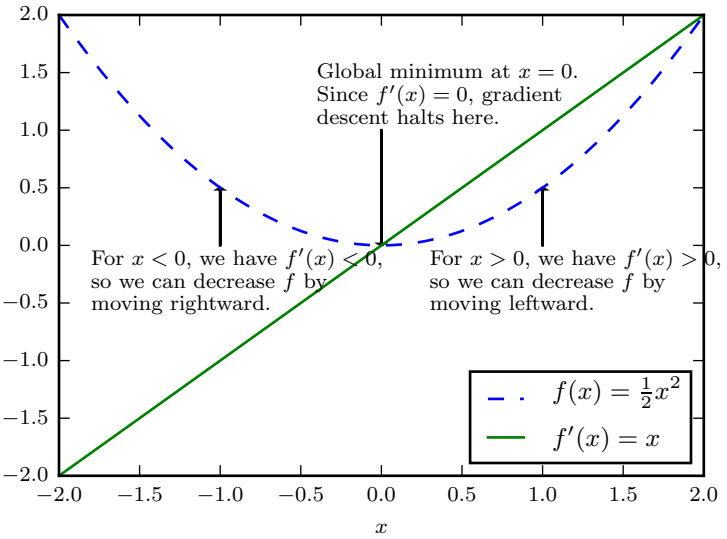

图 4.1: 梯度下降。梯度下降算法如何使用函数导数的示意图，即沿着函数的下坡方向（导数反方向）直到最小。

点。这些点被称为 **鞍点**（saddle point）。见图 4.2 给出的各种临界点的例子。

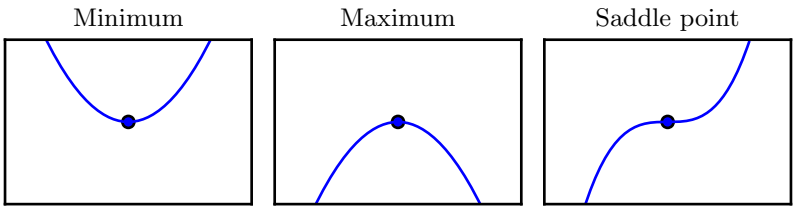

图 4.2: 临界点的类型。一维情况下，三种临界点的示例。临界点是斜率为零的点。这样的点可以是 **局部极小点**（local minimum），其值低于相邻点; **局部极大点**（local maximum），其值高于相邻点; 或鞍点，同时存在更高和更低的相邻点。

使 $f(x)$ 取得绝对的最小值（相对所有其他值）的点是 **全局最小点**（global

minimum)。函数可能只有一个全局最小点或存在多个全局最小点，还可能存在不是全局最优的局部极小点。在深度学习的背景下，我们要优化的函数可能含有许多不是最优的局部极小点，或者还有很多处于非常平坦的区域内的鞍点。尤其是当输入是多维的时候，所有这些都将使优化变得困难。因此，我们通常寻找使 f 非常小的点，但这在任何形式意义下并不一定是最小。见图 4.3 的例子。

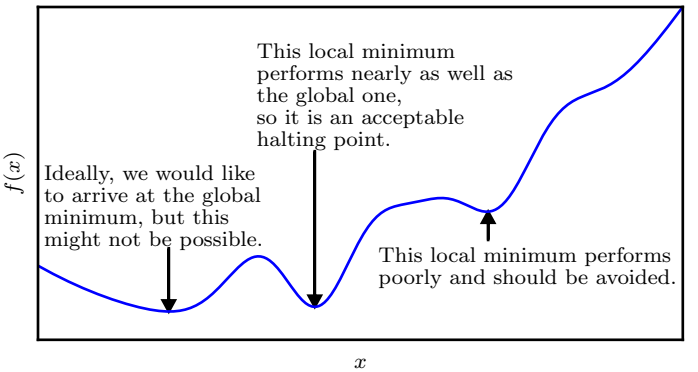

图 4.3: 近似最小化。当存在多个局部极小点或平坦区域时，优化算法可能无法找到全局最小点。在深度学习的背景下，即使找到的解不是真正最小的，但只要它们对应于代价函数显著低的值，我们通常就能接受这样的解。

我们经常最小化具有多维输入的函数：$f : \mathbb{R}^n \to \mathbb{R}$。为了使 “最小化” 的概念有意义，输出必须是一维的 (标量)。

针对具有多维输入的函数，我们需要用到 **偏导数**（partial derivative）的概念。偏导数 $\frac{\partial}{\partial x_i} f(\boldsymbol{x})$ 衡量点 $\boldsymbol{x}$ 处只有 x_i 增加时 $f(\boldsymbol{x})$ 如何变化。**梯度**（gradient）是相对一个向量求导的导数:f 的导数是包含所有偏导数的向量，记为 $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$。梯度的第 i 个元素是 f 关于 x_i 的偏导数。在多维情况下，临界点是梯度中所有元素都为零的点。

在 $\boldsymbol{u}$（单位向量）方向的 **方向导数**（directional derivative）是函数 f 在 $\boldsymbol{u}$ 方向的斜率。换句话说，方向导数是函数 $f(\boldsymbol{x} + \alpha \boldsymbol{u})$ 关于 α 的导数（在 $\alpha = 0$ 时取得）。使用链式法则，我们可以看到当 $\alpha = 0$ 时，$\frac{\partial}{\partial \alpha} f(\boldsymbol{x} + \alpha \boldsymbol{u}) = \boldsymbol{u}^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$。

为了最小化 f，我们希望找到使 f 下降得最快的方向。计算方向导数：

$$\min_{\boldsymbol{u},\boldsymbol{u}^\top\boldsymbol{u}=1} \boldsymbol{u}^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{4.3}$$

$$= \min_{\boldsymbol{u},\boldsymbol{u}^\top\boldsymbol{u}=1} \|\boldsymbol{u}\|_2 \|\nabla_{\boldsymbol{x}} f(\boldsymbol{x})\|_2 \cos\theta \tag{4.4}$$

其中 θ 是 $\boldsymbol{u}$ 与梯度的夹角。将 $\|\boldsymbol{u}\|_2 = 1$ 代入，并忽略与 $\boldsymbol{u}$ 无关的项，就能简化得到 $\min_{\boldsymbol{u}} \cos\theta$。这在 $\boldsymbol{u}$ 与梯度方向相反时取得最小。换句话说，梯度向量指向上坡，负梯度向量指向下坡。我们在负梯度方向上移动可以减小 f。这被称为**最速下降法** (method of steepest descent) 或 **梯度下降**（gradient descent）。

最速下降建议新的点为

$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{4.5}$$

其中 ϵ 为 **学习率**（learning rate），是一个确定步长大小的正标量。我们可以通过几种不同的方式选择 ϵ。普遍的方式是选择一个小常数。有时我们通过计算，选择使方向导数消失的步长。还有一种方法是根据几个 ϵ 计算 $f(\boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}))$，并选择其中能产生最小目标函数值的 ϵ。这种策略被称为线搜索。

最速下降在梯度的每一个元素为零时收敛（或在实践中，很接近零时）。在某些情况下，我们也许能够避免运行该迭代算法，并通过解方程 $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = 0$ 直接跳到临界点。

虽然梯度下降被限制在连续空间中的优化问题，但不断向更好的情况移动一小步（即近似最佳的小移动）的一般概念可以推广到离散空间。递增带有离散参数的目标函数被称为 **爬山**（hill climbing）算法 (Russel and Norvig, 2003)。

4.3.1 梯度之上：Jacobian 和 Hessian 矩阵

有时我们需要计算输入和输出都为向量的函数的所有偏导数。包含所有这样的偏导数的矩阵被称为 **Jacobian** 矩阵。具体来说，如果我们有一个函数：$\boldsymbol{f}: \mathbb{R}^m \to \mathbb{R}^n$，$\boldsymbol{f}$ 的 Jacobian 矩阵 $\boldsymbol{J} \in \mathbb{R}^{n\times m}$ 定义为 $J_{i,j} = \frac{\partial}{\partial x_j} f(\boldsymbol{x})_i$。

有时，我们也对导数的导数感兴趣，即 **二阶导数**（second derivative）。例如，有一个函数 $f: \mathbb{R}^m \to \mathbb{R}$，$f$ 的一阶导数(关于 x_j) 关于 x_i 的导数记为 $\frac{\partial^2}{\partial x_i \partial x_j} f$。在一维情况下，我们可以将 $\frac{\partial^2}{\partial x^2} f$ 为 $f''(x)$。二阶导数告诉我们，一阶导数将如何随着输入的变化而改变。它表示只基于梯度信息的梯度下降步骤是否会产生如我们预期的那

样大的改善，因此它是重要的。我们可以认为，二阶导数是对曲率的衡量。假设我们有一个二次函数（虽然很多实践中的函数都不是二次的，但至少在局部可以很好地用二次近似）。如果这样的函数具有零二阶导数，那就没有曲率。也就是一条完全平坦的线，仅用梯度就可以预测它的值。我们使用沿负梯度方向大小为 ϵ 的下降步，当该梯度是 1 时，代价函数将下降 ϵ。如果二阶导数是负的，函数曲线向下凹陷 (向上凸出)，因此代价函数将下降的比 ϵ 多。如果二阶导数是正的，函数曲线是向上凹陷 (向下凸出)，因此代价函数将下降的比 ϵ 少。从图 4.4 可以看出不同形式的曲率如何影响基于梯度的预测值与真实的代价函数值的关系。

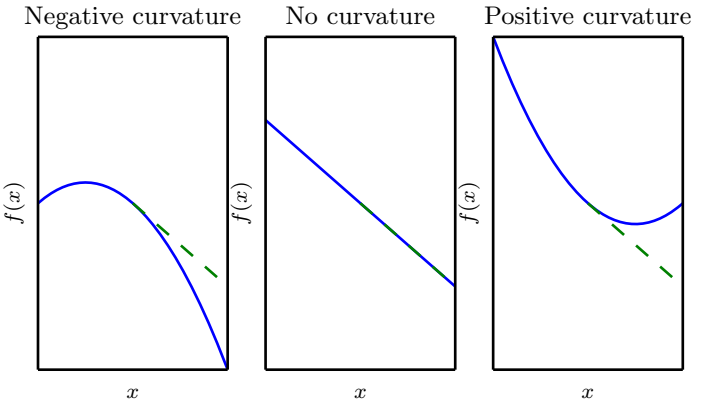

图 4.4: 二阶导数确定函数的曲率。这里我们展示具有各种曲率的二次函数。虚线表示我们仅根据梯度信息进行梯度下降后预期的代价函数值。对于负曲率，代价函数实际上比梯度预测下降得更快。没有曲率时，梯度正确预测下降值。对于正曲率，函数比预期下降得更慢，并且最终会开始增加，因此太大的步骤实际上可能会无意地增加函数值。

当我们的函数具有多维输入时，二阶导数也有很多。我们可以将这些导数合并成一个矩阵，称为 **Hessian** 矩阵。Hessian 矩阵 $\boldsymbol{H}(f)(\boldsymbol{x})$ 定义为

$$\boldsymbol{H}(f)(\boldsymbol{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x}). \tag{4.6}$$

Hessian 等价于梯度的 Jacobian 矩阵。

微分算子在任何二阶偏导连续的点处可交换，也就是它们的顺序可以互换：

$$\frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x}) = \frac{\partial^2}{\partial x_j \partial x_i} f(\boldsymbol{x}). \tag{4.7}$$

这意味着 $H_{i,j} = H_{j,i}$，因此 Hessian 矩阵在这些点上是对称的。在深度学习背景下，我们遇到的大多数函数的 Hessian 几乎处处都是对称的。因为 Hessian 矩阵是实对称的，我们可以将其分解成一组实特征值和一组特征向量的正交基。在特定方向 $\boldsymbol{d}$ 上的二阶导数可以写成 $\boldsymbol{d}^\top \boldsymbol{H}\boldsymbol{d}$。当 $\boldsymbol{d}$ 是 $\boldsymbol{H}$ 的一个特征向量时，这个方向的二阶导数就是对应的特征值。对于其他的方向 $\boldsymbol{d}$，方向二阶导数是所有特征值的加权平均，权重在 0 和 1 之间，且与 $\boldsymbol{d}$ 夹角越小的特征向量的权重越大。最大特征值确定最大二阶导数，最小特征值确定最小二阶导数。

我们可以通过（方向）二阶导数预期一个梯度下降步骤能表现得多好。我们在当前点 $\boldsymbol{x}^{(0)}$ 处作函数 $f(\boldsymbol{x})$ 的近似二阶泰勒级数：

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{g} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}^{(0)}), \tag{4.8}$$

其中 $\boldsymbol{g}$ 是梯度，$\boldsymbol{H}$ 是 $\boldsymbol{x}^{(0)}$ 点的 Hessian。如果我们使用学习率 ϵ，那么新的点 $\boldsymbol{x}$ 将会是 $\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}$。代入上述的近似，可得

$$f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \epsilon \boldsymbol{g}^\top \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^\top \boldsymbol{H}\boldsymbol{g}. \tag{4.9}$$

其中有 3 项：函数的原始值、函数斜率导致的预期改善、函数曲率导致的校正。当最后一项太大时，梯度下降实际上是可能向上移动的。当 $\boldsymbol{g}^\top \boldsymbol{H}\boldsymbol{g}$ 为零或负时，近似的泰勒级数表明增加 ϵ 将永远使 f 下降。在实践中，泰勒级数不会在 ϵ 大的时候也保持准确，因此在这种情况下我们必须采取更启发式的选择。当 $\boldsymbol{g}^\top \boldsymbol{H}\boldsymbol{g}$ 为正时，通过计算可得，使近似泰勒级数下降最多的最优步长为

$$\epsilon^* = \frac{\boldsymbol{g}^\top \boldsymbol{g}}{\boldsymbol{g}^\top \boldsymbol{H}\boldsymbol{g}}. \tag{4.10}$$

最坏的情况下，$\boldsymbol{g}$ 与 $\boldsymbol{H}$ 最大特征值 $\lambda_{\max}$ 对应的特征向量对齐，则最优步长是 $\frac{1}{\lambda_{\max}}$。我们要最小化的函数能用二次函数很好地近似的情况下，Hessian 的特征值决定了学习率的量级。

二阶导数还可以被用于确定一个临界点是否是局部极大点、局部极小点或鞍点。回想一下，在临界点处 $f'(x) = 0$。而 $f''(x) > 0$ 意味着 $f'(x)$ 会随着我们移向右边而增加，移向左边而减小，也就是 $f'(x - \epsilon) < 0$ 和 $f'(x + \epsilon) > 0$ 对足够小的 ϵ 成立。换句话说，当我们移向右边，斜率开始指向右边的上坡，当我们移向左边，斜率开始指向左边的上坡。因此我们得出结论，当 $f'(x) = 0$ 且 $f''(x) > 0$ 时，$\boldsymbol{x}$ 是一个局部极小点。同样，当 $f'(x) = 0$ 且 $f''(x) < 0$ 时，$\boldsymbol{x}$ 是一个局部极大点。这就是所谓

的**二阶导数测试**（second derivative test）。不幸的是，当 $f''(x) = 0$ 时测试是不确定的。在这种情况下，$\boldsymbol{x}$ 可以是一个鞍点或平坦区域的一部分。

在多维情况下，我们需要检测函数的所有二阶导数。利用 Hessian 的特征值分解，我们可以将二阶导数测试扩展到多维情况。在临界点处（$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = 0$），我们通过检测 Hessian 的特征值来判断该临界点是一个局部极大点、局部极小点还是鞍点。当 Hessian 是正定的（所有特征值都是正的），则该临界点是局部极小点。因为方向二阶导数在任意方向都是正的，参考单变量的二阶导数测试就能得出此结论。同样的，当 Hessian 是负定的（所有特征值都是负的），这个点就是局部极大点。在多维情况下，实际上我们可以找到确定该点是否为鞍点的积极迹象（某些情况下）。如果 Hessian 的特征值中至少一个是正的且至少一个是负的，那么 $\boldsymbol{x}$ 是 f 某个横截面的局部极大点，却是另一个横截面的局部极小点。见图 4.5 中的例子。最后，多维二阶导数测试可能像单变量版本那样是不确定的。当所有非零特征值是同号的且至少有一个特征值是 0 时，这个检测就是不确定的。这是因为单变量的二阶导数测试在零特征值对应的横截面上是不确定的。

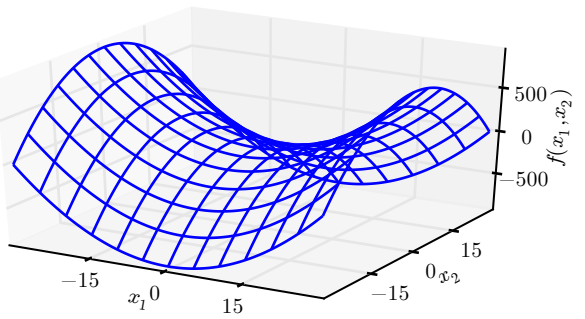

图 4.5: 既有正曲率又有负曲率的鞍点。示例中的函数是 $f(\boldsymbol{x}) = x_1^2 - x_2^2$。函数沿 x_1 轴向上弯曲。x_1 轴是 Hessian 的一个特征向量，并且具有正特征值。函数沿 x_2 轴向下弯曲。该方向对应于 Hessian 负特征值的特征向量。名称 “鞍点” 源自该处函数的鞍状形状。这是具有鞍点函数的典型示例。维度多于一个时，鞍点不一定要具有 0 特征值：仅需要同时具有正特征值和负特征值。我们可以想象这样一个鞍点（具有正负特征值）在一个横截面内是局部极大点，而在另一个横截面内是局部极小点。

多维情况下，单个点处每个方向上的二阶导数是不同。Hessian 的条件数衡量这些二阶导数的变化范围。当 Hessian 的条件数很差时，梯度下降法也会表现得很

差。这是因为一个方向上的导数增加得很快，而在另一个方向上增加得很慢。梯度下降不知道导数的这种变化，所以它不知道应该优先探索导数长期为负的方向。病态条件也导致很难选择合适的步长。步长必须足够小，以免冲过最小而向具有较强正曲率的方向上升。这通常意味着步长太小，以致于在其他较小曲率的方向上进展不明显。见图 4.6 的例子。

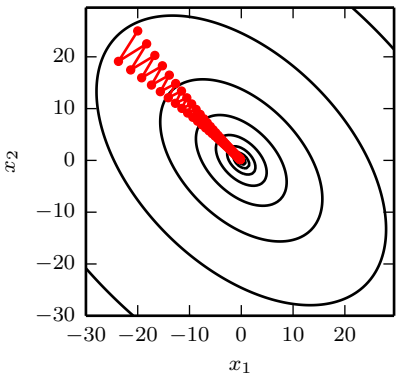

图 4.6: 梯度下降无法利用包含在 Hessian 矩阵中的曲率信息。这里我们使用梯度下降来最小化 Hessian 矩阵条件数为 5 的二次函数 $f(\boldsymbol{x})$。这意味着最大曲率方向具有比最小曲率方向多五倍的曲率。在这种情况下，最大曲率在 $[1,1]^\top$ 方向上，最小曲率在 $[1,-1]^\top$ 方向上。红线表示梯度下降的路径。这个非常细长的二次函数类似一个长峡谷。梯度下降把时间浪费于在峡谷壁反复下降，因为它们是最陡峭的特征。由于步长有点大，有超过函数底部的趋势，因此需要在下一次迭代时在对面的峡谷壁下降。与指向该方向的特征向量对应的 Hessian 的大的正特征值表示该方向上的导数快速增加，因此基于 Hessian 的优化算法可以预测，在此情况下最陡峭方向实际上不是有前途的搜索方向。

我们可以使用 Hessian 矩阵的信息来指导搜索，以解决这个问题。其中最简单的方法是 **牛顿法**（Newton's method）。牛顿法基于一个二阶泰勒展开来近似 $\boldsymbol{x}^{(0)}$ 附近的 $f(\boldsymbol{x})$：

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x}-\boldsymbol{x}^{(0)})^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)}) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}^{(0)})^\top \boldsymbol{H}(f)(\boldsymbol{x}^{(0)})(\boldsymbol{x}-\boldsymbol{x}^{(0)}). \quad (4.11)$$

接着通过计算，我们可以得到这个函数的临界点：

$$\boldsymbol{x}^* = \boldsymbol{x}^{(0)} - \boldsymbol{H}(f)(\boldsymbol{x}^{(0)})^{-1} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)}). \quad (4.12)$$

当 f 是一个正定二次函数时，牛顿法只要应用一次式 (4.12) 就能直接跳到函数的最小点。如果 f 不是一个真正二次但能在局部近似为正定二次，牛顿法则需要多次迭

代应用式 (4.12) 。迭代地更新近似函数和跳到近似函数的最小点可以比梯度下降更快地到达临界点。这在接近局部极小点时是一个特别有用的性质，但是在鞍点附近是有害的。如式 (8.2.3) 所讨论的，当附近的临界点是最小点（Hessian 的所有特征值都是正的）时牛顿法才适用，而梯度下降不会被吸引到鞍点(除非梯度指向鞍点)。

仅使用梯度信息的优化算法被称为**一阶优化算法** (first-order optimization algorithms)，如梯度下降。使用 Hessian 矩阵的优化算法被称为**二阶最优化算法** (second-order optimization algorithms)(Nocedal and Wright, 2006)，如牛顿法。

在本书大多数上下文中使用的优化算法适用于各种各样的函数，但几乎都没有保证。因为在深度学习中使用的函数族是相当复杂的，所以深度学习算法往往缺乏保证。在许多其他领域，优化的主要方法是为有限的函数族设计优化算法。

在深度学习的背景下，限制函数满足 **Lipschitz 连续**（Lipschitz continuous）或其导数Lipschitz连续可以获得一些保证。Lipschitz 连续函数的变化速度以 **Lipschitz 常数**（Lipschitz constant）$\mathcal{L}$ 为界：

$$\forall \boldsymbol{x},\ \forall \boldsymbol{y},\ |f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq \mathcal{L}\|\boldsymbol{x} - \boldsymbol{y}\|_2. \tag{4.13}$$

这个属性允许我们量化我们的假设——梯度下降等算法导致的输入的微小变化将使输出只产生微小变化，因此是很有用的。Lipschitz 连续性也是相当弱的约束，并且深度学习中很多优化问题经过相对较小的修改后就能变得 Lipschitz 连续。

最成功的特定优化领域或许是**凸优化**（Convex optimization）。凸优化通过更强的限制提供更多的保证。凸优化算法只对凸函数适用，即 Hessian 处处半正定的函数。因为这些函数没有鞍点而且其所有局部极小点必然是全局最小点，所以表现很好。然而，深度学习中的大多数问题都难以表示成凸优化的形式。凸优化仅用作一些深度学习算法的子程序。凸优化中的分析思路对证明深度学习算法的收敛性非常有用，然而一般来说，深度学习背景下凸优化的重要性大大减少。有关凸优化的详细信息，详见 Boyd and Vandenberghe (2004) 或 Rockafellar (1997)。

4.4 约束优化

有时候，在 $\boldsymbol{x}$ 的所有可能值下最大化或最小化一个函数 $f(x)$ 不是我们所希望的。相反，我们可能希望在 $\boldsymbol{x}$ 的某些集合 $\mathbb{S}$ 中找 $f(\boldsymbol{x})$ 的最大值或最小值。这被称为**约束优化**（constrained optimization）。在约束优化术语中，集合 $\mathbb{S}$ 内的点 $\boldsymbol{x}$ 被称

为 **可行**（feasible）点。

我们常常希望找到在某种意义上小的解。针对这种情况下的常见方法是强加一个范数约束，如 $\|\boldsymbol{x}\| \leq 1$。

约束优化的一个简单方法是将约束考虑在内后简单地对梯度下降进行修改。如果我们使用一个小的恒定步长 ϵ，我们可以先取梯度下降的单步结果，然后将结果投影回 $\mathbb{S}$。如果我们使用线搜索，我们只能在步长为 ϵ 范围内搜索可行的新 $\boldsymbol{x}$ 点，或者我们可以将线上的每个点投影到约束区域。如果可能的话，在梯度下降或线搜索前将梯度投影到可行域的切空间会更高效 (Rosen, 1960)。

一个更复杂的方法是设计一个不同的、无约束的优化问题，其解可以转化成原始约束优化问题的解。例如，我们要在 $\boldsymbol{x} \in \mathbb{R}^2$ 中最小化 $f(\boldsymbol{x})$，其中 $\boldsymbol{x}$ 约束为具有单位 L^2 范数。我们可以关于 θ 最小化 $g(\theta) = f([\cos\theta, \sin\theta]^\top)$，最后返回 $[\cos\theta, \sin\theta]$ 作为原问题的解。这种方法需要创造性；优化问题之间的转换必须专门根据我们遇到的每一种情况进行设计。

Karush–Kuhn–Tucker（KKT）方法[2]是针对约束优化非常通用的解决方案。为介绍KKT方法，我们引入一个称为 **广义 Lagrangian**（generalized Lagrangian）或 **广义 Lagrange 函数**（generalized Lagrange function）的新函数。

为了定义Lagrangian，我们先要通过等式和不等式的形式描述 $\mathbb{S}$。我们希望通过 m 个函数 $g^{(i)}$ 和 n 个函数 $h^{(j)}$ 描述 $\mathbb{S}$，那么 $\mathbb{S}$ 可以表示为 $\mathbb{S} = \{\boldsymbol{x} \mid \forall i, g^{(i)}(\boldsymbol{x}) = 0 \text{ and } \forall j, h^{(j)}(\boldsymbol{x}) \leq 0\}$。其中涉及 $g^{(i)}$ 的等式称为 **等式约束**（equality constraint），涉及 $h^{(j)}$ 的不等式称为 **不等式约束**（inequality constraint）。

我们为每个约束引入新的变量 λ_i 和 α_j，这些新变量被称为 KKT 乘子。广义 Lagrangian 可以如下定义：

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x}). \tag{4.14}$$

现在，我们可以通过优化无约束的广义 Lagrangian 解决约束最小化问题。只要存在至少一个可行点且 $f(\boldsymbol{x})$ 不允许取 ∞，那么

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) \tag{4.15}$$

[2]KKT 方法是 **Lagrange 乘子法**（只允许等式约束）的推广。

与如下函数有相同的最优目标函数值和最优点集 $\boldsymbol{x}$

$$\min_{\boldsymbol{x}\in\mathbb{S}} f(\boldsymbol{x}). \tag{4.16}$$

这是因为当约束满足时，

$$\max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha},\boldsymbol{\alpha}\geq 0} L(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\alpha}) = f(\boldsymbol{x}), \tag{4.17}$$

而违反任意约束时，

$$\max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha},\boldsymbol{\alpha}\geq 0} L(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\alpha}) = \infty. \tag{4.18}$$

这些性质保证不可行点不会是最佳的，并且可行点范围内的最优点不变。

要解决约束最大化问题，我们可以构造 $-f(\boldsymbol{x})$ 的广义 Lagrange 函数，从而导致以下优化问题：

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha},\boldsymbol{\alpha}\geq 0} -f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x}). \tag{4.19}$$

我们也可将其转换为在外层最大化的问题：

$$\max_{\boldsymbol{x}} \min_{\boldsymbol{\lambda}} \min_{\boldsymbol{\alpha},\boldsymbol{\alpha}\geq 0} f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) - \sum_j \alpha_j h^{(j)}(\boldsymbol{x}). \tag{4.20}$$

等式约束对应项的符号并不重要；因为优化可以自由选择每个 λ_i 的符号，我们可以随意将其定义为加法或减法。

不等式约束特别有趣。如果 $h^{(i)}(\boldsymbol{x}^*) = 0$，我们就说这个约束 $h^{(i)}(\boldsymbol{x})$ 是**活跃** (active) 的。如果约束不是活跃的，则有该约束的问题的解与去掉该约束的问题的解至少存在一个相同的局部解。一个不活跃约束有可能排除其他解。例如，整个区域（代价相等的宽平区域）都是全局最优点的凸问题可能因约束消去其中的某个子区域，或在非凸问题的情况下，收敛时不活跃的约束可能排除了较好的局部驻点。然而，无论不活跃的约束是否被包括在内，收敛时找到的点仍然是一个驻点。因为一个不活跃的约束 $h^{(i)}$ 必有负值，那么 $\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha},\boldsymbol{\alpha}\geq 0} L(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\alpha})$ 中的 $\alpha_i = 0$。因此，我们可以观察到在该解中 $\boldsymbol{\alpha} \odot \boldsymbol{h}(\boldsymbol{x}) = 0$。换句话说，对于所有的 i，$\alpha_i \geq 0$ 或 $h^{(j)}(\boldsymbol{x}) \leq 0$ 在收敛时必有一个是活跃的。为了获得关于这个想法的一些直观解释，我们可以说这个解是由不等式强加的边界，我们必须通过对应的 KKT 乘子影响 $\boldsymbol{x}$ 的解，或者不等式对解没有影响，我们则归零 KKT 乘子。

我们可以使用一组简单的性质来描述约束优化问题的最优点。这些性质称为 **Karush–Kuhn–Tucker**（KKT）条件 (Karush, 1939; Kuhn and Tucker, 1951)。这些是确定一个点是最优点的必要条件，但不一定是充分条件。这些条件是：

- 广义 Lagrangian 的梯度为零。
- 所有关于 $\boldsymbol{x}$ 和 KKT 乘子的约束都满足。
- 不等式约束显示的 “互补松弛性”：$\boldsymbol{\alpha} \odot \boldsymbol{h}(\boldsymbol{x}) = 0$。

有关 KKT 方法的详细信息，请参阅 Nocedal and Wright (2006)。

4.5 实例：线性最小二乘

假设我们希望找到最小化下式的 $\boldsymbol{x}$ 值

$$f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2. \tag{4.21}$$

存在专门的线性代数算法能够高效地解决这个问题；但是，我们也可以探索如何使用基于梯度的优化来解决这个问题，这可以作为这些技术是如何工作的一个简单例子。

首先，我们计算梯度：

$$\nabla_x f(\boldsymbol{x}) = \boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \boldsymbol{A}^\top\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top\boldsymbol{b}. \tag{4.22}$$

然后，我们可以采用小的步长，并按照这个梯度下降。见算法 4.1 中的详细信息。

算法 4.1 从任意点 $\boldsymbol{x}$ 开始，使用梯度下降关于 $\boldsymbol{x}$ 最小化 $f(\boldsymbol{x}) = \frac{1}{2}||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2^2$ 的算法。

将步长 (ϵ) 和容差 (δ) 设为小的正数。
while $||\boldsymbol{A}^\top\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top\boldsymbol{b}||_2 > \delta$ **do**
  $\boldsymbol{x} \leftarrow \boldsymbol{x} - \epsilon\left(\boldsymbol{A}^\top\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top\boldsymbol{b}\right)$
end while

我们也可以使用牛顿法解决这个问题。因为在这个情况下，真实函数是二次的，牛顿法所用的二次近似是精确的，该算法会在一步后收敛到全局最小点。