第十一章　实践方法论

要成功地使用深度学习技术，仅仅知道存在哪些算法和解释他们为何有效的原理是不够的。一个优秀的机器学习实践者还需要知道如何针对具体应用挑选一个合适的算法以及如何监控，并根据实验反馈改进机器学习系统。在机器学习系统的日常开发中，实践者需要决定是否收集更多的数据、增加或减少模型容量、添加或删除正则化项、改进模型的优化、改进模型的近似推断或调试模型的软件实现。尝试这些操作都需要大量时间，因此确定正确做法，而不盲目猜测尤为重要的。

本书的大部分内容都是关于不同的机器学习模型、训练算法和目标函数。这可能给人一种印象——成为机器学习专家的最重要因素是了解各种各样的机器学习技术，并熟悉各种不同的数学。在实践中，正确使用一个普通算法通常比草率地使用一个不清楚的算法效果更好。正确应用一个算法需要掌握一些相当简单的方法论。本章的许多建议都来自 Ng (2015)。

我们建议参考以下几个实践设计流程：

- 确定目标——使用什么样的误差度量，并为此误差度量指定目标值。这些目标和误差度量取决于该应用旨在解决的问题。
- 尽快建立一个端到端的工作流程，包括估计合适的性能度量。
- 搭建系统，并确定性能瓶颈。检查哪个部分的性能差于预期，以及是否是因为过拟合、欠拟合，或者数据或软件缺陷造成的。
- 根据具体观察反复地进行增量式的改动，如收集新数据、调整超参数或改进算法。

我们将使用街景地址号码转录系统 (Goodfellow *et al.*, 2014d) 作为一个运行示例。该应用的目标是将建筑物添加到谷歌地图。街景车拍摄建筑物，并记录与每张

建筑照片相关的 GPS 坐标。卷积网络识别每张照片上的地址号码，由谷歌地图数据库在正确的位置添加该地址。这个商业应用是一个很好的示例，它的开发流程遵循我们倡导的设计方法。

我们现在描述这个过程中的每一个步骤。

11.1 性能度量

确定目标，即使用什么误差度量，是必要的第一步，因为误差度量将指导接下来的所有工作。同时我们也应该了解大概能得到什么级别的目标性能。

值得注意的是对于大多数应用而言，不可能实现绝对零误差。即使你有无限的训练数据，并且恢复了真正的概率分布，贝叶斯误差仍定义了能达到的最小错误率。这是因为输入特征可能无法包含输出变量的完整信息，或是因为系统可能本质上是随机的。当然我们还会受限于有限的训练数据。

训练数据的数量会因为各种原因受到限制。当目标是打造现实世界中最好的产品或服务时，我们通常需要收集更多的数据，但必须确定进一步减少误差的价值，并与收集更多数据的成本做权衡。数据收集会耗费时间、金钱，或带来人体痛苦（例如，收集人体医疗测试数据）。科研中，目标通常是在某个确定基准下探讨哪个算法更好，一般会固定训练集，不允许收集更多的数据。

如何确定合理的性能期望？在学术界，通常我们可以根据先前公布的基准结果来估计预期错误率。在现实世界中，一个应用的错误率有必要是安全的、具有成本效益的或吸引消费者的。一旦你确定了想要达到的错误率，那么你的设计将由如何达到这个错误率来指导。

除了需要考虑性能度量之外，另一个需要考虑的是度量的选择。我们有几种不同的性能度量，可以用来度量一个含有机器学习组件的完整应用的有效性。这些性能度量通常不同于训练模型的代价函数。如第 5.1.2 节所述，我们通常会度量一个系统的准确率，或等价地，错误率。

然而，许多应用需要更高级的度量。

有时，一种错误可能会比另一种错误更严重。例如，垃圾邮件检测系统会有两种错误：将正常邮件错误地归为垃圾邮件，将垃圾邮件错误地归为正常邮件。阻止正常消息比允许可疑消息通过糟糕得多。我们希望度量某种形式的总代价，其中拦

截正常邮件比允许垃圾邮件通过的代价更高，而不是度量垃圾邮件分类的错误率。

有时，我们需要训练检测某些罕见事件的二元分类器。例如，我们可能会为一种罕见疾病设计医疗测试。假设每一百万人中只有一人患病。我们只需要让分类器一直报告没有患者，就能轻易地在检测任务上实现 99.9999% 的正确率。显然，正确率很难描述这种系统的性能。解决这个问题的方法是度量 **精度**（ precision ）和 **召回率**（ recall ）。精度是模型报告的检测是正确的比率，而召回率则是真实事件被检测到的比率。检测器永远报告没有患者，会得到一个完美的精度，但召回率为零。而报告每个人都是患者的检测器会得到一个完美的召回率，但是精度会等于人群中患有该病的比例（在我们的例子是 0.0001%，每一百万人只有一人患病）。当使用精度和召回率时，我们通常会画 **PR 曲线**（ PR curve ），y 轴表示精度，x 轴表示召回率。如果检测到的事件发生了，那么分类器会返回一个较高的得分。例如，我们将前馈网络设计为检测一种疾病，估计一个医疗结果由特征 $\boldsymbol{x}$ 表示的人患病的概率为 $\hat{y} = P(y = 1 \mid \boldsymbol{x})$。每当这个得分超过某个阈值时，我们报告检测结果。通过调整阈值，我们能权衡精度和召回率。在很多情况下，我们希望用一个数而不是曲线来概括分类器的性能。要做到这一点，我们可以将精度 p 和召回率 r 转换为 **F 分数**（ F-score ）

$$F = \frac{2pr}{p + r}. \tag{11.1}$$

另一种方法是报告 PR 曲线下方的总面积。

在一些应用中，机器学习系统可能会拒绝做出判断。如果机器学习算法能够估计所作判断的置信度，这将会非常有用，特别是在错误判断会导致严重危害，而人工操作员能够偶尔接管的情况下。街景转录系统可以作为这种情况的一个示例。这个任务是识别照片上的地址号码，将照片拍摄地点对应到地图上的地址。如果地图是不精确的，那么地图的价值会严重下降。因此只在转录正确的情况下添加地址十分重要。如果机器学习系统认为它不太能像人一样正确地转录，那么最好办法当然是让人来转录照片。当然，只有当机器学习系统能够大量降低需要人工操作处理的图片时，它才是有用的。在这种情况下，一种自然的性能度量是 **覆盖**（ coverage ）。覆盖是机器学习系统能够产生响应的样本所占的比率。我们权衡覆盖和精度。一个系统可以通过拒绝处理任意样本的方式来达到 100% 的精度，但是覆盖降到了 0%。对于街景任务，该项目的目标是达到人类级别的转录精度，同时保持 95% 的覆盖。在这项任务中，人类级别的性能是 98% 的精度。

还有许多其他的性能度量。例如，我们可以度量点击率、收集用户满意度调查

等等。许多专业的应用领域也有特定的标准。

最重要的是首先要确定改进哪个性能度量，然后专心提高性能度量。如果没有明确的目标，那么我们很难判断机器学习系统上的改动是否有所改进。

11.2 默认的基准模型

确定性能度量和目标后，任何实际应用的下一步是尽快建立一个合理的端到端的系统。本节给出了一些关于在不同情况下使用哪种算法作为第一个基准方法推荐。在本节中，我们提供了关于不同情况下使用哪种算法作为第一基准方法的推荐。值得注意的是，深度学习研究进展迅速，所以本书出版后很快可能会有更好的默认算法。

根据问题的复杂性，项目开始时可能无需使用深度学习。如果只需正确地选择几个线性权重就可能解决问题，那么项目可以开始于一个简单的统计模型，如逻辑回归。

如果问题属于 “AI-完全’’ 类的，如对象识别、语音识别、机器翻译等等，那么项目开始于一个合适的深度学习模型，效果会比较好。

首先，根据数据的结构选择一类合适的模型。如果项目是以固定大小的向量作为输入的监督学习，那么可以使用全连接的前馈网络。如果输入有已知的拓扑结构（例如，输入是图像），那么可以使用卷积网络。在这些情况下，刚开始可以使用某些分段线性单元（ReLU 或者其扩展，如 Leaky ReLU、PReLU 和 maxout）。如果输入或输出是一个序列，可以使用门控循环网络（LSTM 或 GRU）。

具有衰减学习率以及动量的 SGD 是优化算法一个合理的选择（流行的衰减方法有，衰减到固定最低学习率的线性衰减、指数衰减，或每次发生验证错误停滞时将学习率降低 2 – 10 倍，这些衰减方法在不同问题上好坏不一）。另一个非常合理的选择是 Adam 算法。批标准化对优化性能有着显著的影响，特别是对卷积网络和具有 sigmoid 非线性函数的网络而言。虽然在最初的基准中忽略批标准化是合理的，然而当优化似乎出现问题时，应该立刻使用批标准化。

除非训练集包含数千万以及更多的样本，否则项目应该在一开始就包含一些温和的正则化。提前终止也被普遍采用。Dropout 也是一个很容易实现，且兼容很多模型和训练算法的出色正则化项。批标准化有时也能降低泛化误差，此时可以省略 Dropout 步骤，因为用于标准化变量的统计量估计本身就存在噪声。

如果我们的任务和另一个被广泛研究的任务相似，那么通过复制先前研究中已知性能良好的模型和算法，可能会得到很好的效果。甚至可以从该任务中复制一个训练好的模型。例如，通常会使用在 ImageNet 上训练好的卷积网络的特征来解决其他计算机视觉任务 (Girshick *et al.*, 2015)。

一个常见问题是项目开始时是否使用无监督学习，我们将在第三部分进一步探讨这个问题。这个问题和特定领域有关。在某些领域，比如自然语言处理，能够大大受益于无监督学习技术，如学习无监督词嵌入。在其他领域，如计算机视觉，除非是在半监督的设定下（标注样本数量很少）(Kingma *et al.*, 2014; Rasmus *et al.*, 2015)，目前无监督学习并没有带来益处。如果应用所在环境中，无监督学习被认为是很重要的，那么将其包含在第一个端到端的基准中。否则，只有在解决无监督问题时，才会第一次尝试时使用无监督学习。在发现初始基准过拟合的时候，我们可以尝试加入无监督学习。

11.3 决定是否收集更多数据

在建立第一个端到端的系统后，就可以度量算法性能并决定如何改进算法。许多机器学习新手都忍不住尝试很多不同的算法来进行改进。然而，收集更多的数据往往比改进学习算法要有用得多。

怎样判断是否要收集更多的数据？首先，确定训练集上的性能是否可接受。如果模型在训练集上的性能就很差，学习算法都不能在训练集上学习出良好的模型，那么就没必要收集更多的数据。反之，可以尝试增加更多的网络层或每层增加更多的隐藏单元，以增加模型的规模。此外，也可以尝试调整学习率等超参数的措施来改进学习算法。如果更大的模型和仔细调试的优化算法效果不佳，那么问题可能源自训练数据的质量。数据可能含太多噪声，或是可能不包含预测输出所需的正确输入。这意味着我们需要重新开始，收集更干净的数据或是收集特征更丰富的数据集。

如果训练集上的性能是可接受的，那么我们开始度量测试集上的性能。如果测试集上的性能也是可以接受的，那么就顺利完成了。如果测试集上的性能比训练集的要差得多，那么收集更多的数据是最有效的解决方案之一。这时主要的考虑是收集更多数据的代价和可行性，其他方法降低测试误差的代价和可行性，和增加数据数量能否显著提升测试集性能。在拥有百万甚至上亿用户的大型网络公司，收集大型数据集是可行的，并且这样做的成本可能比其他方法要少很多，所以答案几乎总是收

集更多的训练数据。例如，收集大型标注数据集是解决对象识别问题的主要因素之一。在其他情况下，如医疗应用，收集更多的数据可能代价很高或者不可行。一个可以替代的简单方法是降低模型大小或是改进正则化（调整超参数，如权重衰减系数，或是加入正则化策略，如 Dropout）。如果调整正则化超参数后，训练集性能和测试集性能之间的差距还是不可接受，那么收集更多的数据是可取的。

在决定是否收集更多的数据时，也需要确定收集多少数据。如图 5.4 所示，绘制曲线显示训练集规模和泛化误差之间的关系是很有帮助的。根据走势延伸曲线，可以预测还需要多少训练数据来达到一定的性能。通常，加入总数目一小部分的样本不会对泛化误差产生显著的影响。因此，建议在对数尺度上考虑训练集的大小，例如在后续的实验中倍增样本数目。

如果收集更多的数据是不可行的，那么改进泛化误差的唯一方法是改进学习算法本身。这属于研究领域，并非对应用实践者的建议。

11.4 选择超参数

大部分深度学习算法都有许多超参数来控制不同方面的算法表现。有些超参数会影响算法运行的时间和存储成本。有些超参数会影响学习到的模型质量，以及在新输入上推断正确结果的能力。

有两种选择超参数的基本方法：手动选择和自动选择。手动选择超参数需要了解超参数做了些什么，以及机器学习模型如何才能取得良好的泛化。自动选择超参数算法大大减少了解这些想法的需要，但它们往往需要更高的计算成本。

11.4.1 手动调整超参数

手动设置超参数，我们必须了解超参数、训练误差、泛化误差和计算资源（内存和运行时间）之间的关系。这需要切实了解一个学习算法有效容量的基础概念，如第五章所描述的。

手动搜索超参数的目标通常是最小化受限于运行时间和内存预算的泛化误差。我们不去探讨如何确定各种超参数对运行时间和内存的影响，因为这高度依赖于平台。

手动搜索超参数的主要目标是调整模型的有效容量以匹配任务的复杂性。有

效容量受限于三个因素：模型的表示容量、学习算法成功最小化训练模型代价函数的能力以及代价函数和训练过程正则化模型的程度。具有更多网络层，每层有更多隐藏单元的模型具有较高的表示能力——能够表示更复杂的函数。然而，如果训练算法不能找到某个合适的函数来最小化训练代价，或是正则化项（如权重衰减）排除了这些合适的函数，那么即使模型的表达能力较高，也不能学习出合适的函数。

当泛化误差以某个超参数为变量，作为函数绘制出来时，通常会表现为 U 形曲线，如图 5.3 所示。在某个极端情况下，超参数对应着低容量，并且泛化误差由于训练误差较大而很高。这便是欠拟合的情况。另一种极端情况，超参数对应着高容量，并且泛化误差由于训练误差和测试误差之间的差距较大而很高。最优的模型容量位于曲线中间的某个位置，能够达到最低可能的泛化误差，由某个中等的泛化误差和某个中等的训练误差相加构成。

对于某些超参数，当超参数数值太大时，会发生过拟合。例如中间层隐藏单元的数量，增加数量能提高模型的容量，容易发生过拟合。对于某些超参数，当超参数数值太小时，也会发生过拟合。例如，最小的权重衰减系数允许为零，此时学习算法具有最大的有效容量，反而容易过拟合。

并非每个超参数都能对应着完整的 U 形曲线。很多超参数是离散的，如中间层单元数目或是 maxout 单元中线性元件的数目，这种情况只能沿曲线探索一些点。有些超参数是二值的。通常这些超参数用来指定是否使用学习算法中的一些可选部分，如预处理步骤减去均值并除以标准差来标准化输入特征。这些超参数只能探索曲线上的两点。其他一些超参数可能会有最小值或最大值，限制其探索曲线的某些部分。例如，权重衰减系数最小是零。这意味着，如果权重衰减系数为零时模型欠拟合，那么我们将无法通过修改权重衰减系数探索过拟合区域。换言之，有些超参数只能减少模型容量。

学习率可能是最重要的超参数。如果你只有时间调整一个超参数，那就调整学习率。相比其他超参数，它以一种更复杂的方式控制模型的有效容量——当学习率适合优化问题时，模型的有效容量最高，此时学习率是正确的，既不是特别大也不是特别小。学习率关于训练误差具有 U 形曲线，如图 11.1 所示。当学习率过大时，梯度下降可能会不经意地增加而非减少训练误差。在理想化的二次情况下，如果学习率是最佳值的两倍大时，会发生这种情况 (LeCun *et al.*, 1998b)。当学习率太小，训练不仅慢，还有可能永久停留在一个很高的训练误差。关于这种效应，我们知之甚少（不会发生于一个凸损失函数中）。

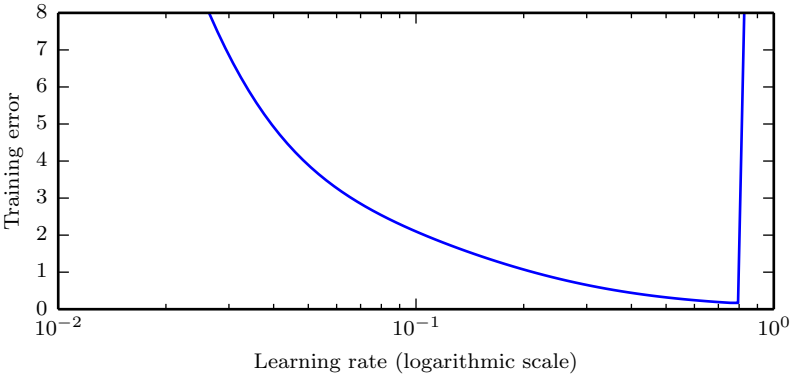

图 11.1: 训练误差和学习率之间的典型关系。注意当学习率大于最优值时误差会有显著的提升。此图针对固定的训练时间，越小的学习率有时候可以以一个正比于学习率减小量的因素来减慢训练过程。泛化误差也会得到类似的曲线，由于正则项作用在学习率过大或过小处比较复杂。由于一个糟糕的优化从某种程度上说可以避免过拟合，即使是训练误差相同的点也会拥有完全不同的泛化误差。

调整学习率外的其他参数时，需要同时监测训练误差和测试误差，以判断模型是否过拟合或欠拟合，然后适当调整其容量。

如果训练集错误率大于目标错误率，那么只能增加模型容量以改进模型。如果没有使用正则化，并且确信优化算法正确运行，那么有必要添加更多的网络层或隐藏单元。然而，令人遗憾的是，这增加了模型的计算代价。

如果测试集错误率大于目标错误率，那么可以采取两个方法。测试误差是训练误差和测试误差之间差距与训练误差的总和。寻找最佳的测试误差需要权衡这些数值。当训练误差较小（因此容量较大），测试误差主要取决于训练误差和测试误差之间的差距时，通常神经网络效果最好。此时目标是缩小这一差距，使训练误差的增长速率不快于差距减小的速率。要减少这个差距，我们可以改变正则化超参数，以减少有效的模型容量，如添加 Dropout 或权重衰减策略。通常，最佳性能来自正则化得很好的大规模模型，比如使用 Dropout 的神经网络。

大部分超参数可以通过推理其是否增加或减少模型容量来设置。部分示例如表11.1所示。

手动调整超参数时，不要忘记最终目标：提升测试集性能。加入正则化只是实现这个目标的一种方法。只要训练误差低，随时都可以通过收集更多的训练数据来

超参数	容量何时增加	原因	注意事项
隐藏单元数量	增加	增加隐藏单元数量会增加模型的表示能力。	几乎模型每个操作所需的时间和内存代价都会随隐藏单元数量的增加而增加。
学习率	调至最优	不正确的学习速率，不管是太高还是太低都会由于优化失败而导致低有效容量的模型。	
卷积核宽度	增加	增加卷积核宽度会增加模型的参数数量。	较宽的卷积核导致较窄的输出尺寸，除非使用隐式零填充减少此影响，否则会降低模型容量。较宽的卷积核需要更多的内存存储参数，并会增加运行时间，但较窄的输出会降低内存代价。
隐式零填充	增加	在卷积之前隐式添加零能保持较大尺寸的表示。	大多数操作的时间和内存代价会增加。
权重衰减系数	降低	降低权重衰减系数使得模型参数可以自由地变大。	
Dropout 比率	降低	较少地丢弃单元可以更多地让单元彼此“协力”来适应训练集。	

表 11.1: 各种超参数对模型容量的影响。

减少泛化误差。实践中能够确保学习有效的暴力方法就是不断提高模型容量和训练集的大小，直到解决问题。这种做法增加了训练和推断的计算代价，所以只有在拥有足够资源时才是可行的。原则上，这种做法可能会因为优化难度提高而失败，但对于许多问题而言，优化似乎并没有成为一个显著的障碍，当然，前提是选择了合适的模型。

11.4.2 自动超参数优化算法

理想的学习算法应该是只需要输入一个数据集，就可以输出学习的函数，而不需要手动调整超参数。一些流行的学习算法，如逻辑回归和支持向量机，流行的部

分原因是这类算法只有一到两个超参数需要调整，它们也能表现出不错的性能。有些情况下，所需调整的超参数数量较少时，神经网络可以表现出不错的性能；但超参数数量有几十甚至更多时，效果会提升得更加明显。当使用者有一个很好的初始值，例如由在相同类型的应用和架构上具有经验的人确定初始值，或者使用者在相似问题上具有几个月甚至几年的神经网络超参数调整经验，那么手动调整超参数能有很好的效果。然而，对于很多应用而言，这些起点都不可用。在这些情况下，自动算法可以找到合适的超参数。

如果我们仔细想想使用者搜索学习算法合适超参数的方式，我们会意识到这其实是一种优化：我们在试图寻找超参数来优化目标函数，例如验证误差，有时还会有一些约束（如训练时间，内存或识别时间的预算）。因此，原则上有可能开发出封装学习算法的**超参数优化**（hyperparameter optimization）算法，并选择其超参数，从而使用者不需要指定学习算法的超参数。令人遗憾的是，超参数优化算法往往有自己的超参数，如学习算法的每个超参数应该被探索的值的范围。然而，这些次级超参数通常很容易选择，这是说，相同的次级超参数能够很多不同的问题上具有良好的性能。

11.4.3 网格搜索

当有三个或更少的超参数时，常见的超参数搜索方法是**网格搜索**（grid search）。对于每个超参数，使用者选择一个较小的有限值集去探索。然后，这些超参数笛卡尔乘积得到一组组超参数，网格搜索使用每组超参数训练模型。挑选验证集误差最小的超参数作为最好的超参数。如图 11.2 所示超参数值的网络。

应该如何选择搜索集合的范围呢？在超参数是数值（有序）的情况下，每个列表的最小和最大的元素可以基于先前相似实验的经验保守地挑选出来，以确保最优解非常可能在所选范围内。通常，网格搜索大约会在**对数尺度**（logarithmic scale）下挑选合适的值，例如，一个学习率的取值集合是 $\{0.1, 0.01, 10^{-3}, 10^{-4}, 10^{-5}\}$，或者隐藏单元数目的取值集合 $\{50, 100, 200, 500, 1000, 2000\}$。

通常重复进行网格搜索时，效果会最好。例如，假设我们在集合 $\{-1, 0, 1\}$ 上网格搜索超参数 α。如果找到的最佳值是 1，那么说明我们低估了最优值 α 所在的范围，应该改变搜索格点，例如在集合 $\{1, 2, 3\}$ 中搜索。如果最佳值是 0，那么我们不妨通过细化搜索范围以改进估计，在集合 $\{-0.1, 0, 0.1\}$ 上进行网格搜索。

网格搜索带来的一个明显问题是，计算代价会随着超参数数量呈指数级增长。如

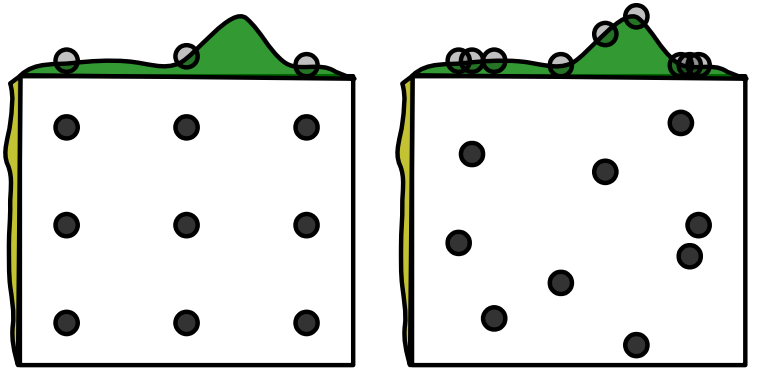

图 11.2: 网格搜索和随机搜索的比较。为了方便地说明，我们只展示两个超参数的例子，但是我们关注的问题中超参数个数通常会更多。*(左)* 为了实现网格搜索，我们为每个超参数提供了一个值的集合。搜索算法对每一种在这些集合的交叉积中的超参数组合进行训练。*(右)* 为了实现随机搜索，我们给联合超参数赋予了一个概率分布。通常超参数之间是相互独立的。常见的这种分布的选择是均匀分布或者是对数均匀（从对数均匀分布中抽样，就是对从均匀分布中抽取的样本进行指数运算）的。然后这些搜索算法从联合的超参数空间中采样，然后运行每一个样本。网格搜索和随机搜索都运行了验证集上的误差并返回了最优的解。这个图说明了通常只有一个超参数对结果有着重要的影响。在这个例子中，只有水平轴上的超参数对结果有重要的作用。网格搜索将大量的计算浪费在了指数量级的对结果无影响的超参数中，相比之下随机搜索几乎每次测试都测试了对结果有影响的每个超参数的独一无二的值。此图经 Bergstra and Bengio (2011) 允许转载。

果有 m 个超参数，每个最多取 n 个值，那么训练和估计所需的试验数将是 $O(n^m)$。我们可以并行地进行实验，并且并行要求十分宽松（进行不同搜索的机器之间几乎没有必要进行通信）。令人遗憾的是，由于网格搜索指数级增长计算代价，即使是并行，我们也无法提供令人满意的搜索规模。

11.4.4 随机搜索

幸运的是，有一个替代网格搜索的方法，并且编程简单，使用更方便，能更快地收敛到超参数的良好取值：随机搜索 (Bergstra and Bengio, 2012)。

随机搜索过程如下。首先，我们为每个超参数定义一个边缘分布，例如，Bernoulli 分布或范畴分布（分别对应着二元超参数或离散超参数），或者对数尺度上的均匀分

布（对应着正实值超参数）。例如，

$$\texttt{log\_learning\_rate} \sim u(-1, -5), \tag{11.2}$$

$$\texttt{learning\_rate} = 10^{\texttt{log\_learning\_rate}}, \tag{11.3}$$

其中，$u(a, b)$ 表示区间 (a, b) 上均匀采样的样本。类似地，`log_number_of_hidden_units` 可以从 $u(\log(50), \log(2000))$ 上采样。

与网格搜索不同，我们不需要离散化超参数的值。这允许我们在一个更大的集合上进行搜索，而不产生额外的计算代价。实际上，如图 11.2 所示，当有几个超参数对性能度量没有显著影响时，随机搜索相比于网格搜索指数级地高效。Bergstra and Bengio (2012) 进行了详细的研究并发现相比于网格搜索，随机搜索能够更快地减小验证集误差（就每个模型运行的试验数而言）。

与网格搜索一样，我们通常会重复运行不同版本的随机搜索，以基于前一次运行的结果改进下一次搜索。

随机搜索能比网格搜索更快地找到良好超参数的原因是，没有浪费的实验，不像网格搜索有时会对一个超参数的两个不同值（给定其他超参数值不变）给出相同结果。在网格搜索中，其他超参数将在这两次实验中拥有相同的值，而在随机搜索中，它们通常会具有不同的值。因此，如果这两个值的变化所对应的验证集误差没有明显区别的话，网格搜索没有必要重复两个等价的实验，而随机搜索仍然会对其他超参数进行两次独立地探索。

11.4.5 基于模型的超参数优化

超参数搜索问题可以转化为一个优化问题。决策变量是超参数。优化的代价是超参数训练出来的模型在验证集上的误差。在简化的设定下，可以计算验证集上可导误差函数关于超参数的梯度，然后我们遵循这个梯度更新 (Bengio *et al.*, 1999; Bengio, 2000; Maclaurin *et al.*, 2015)。令人遗憾的是，在大多数实际设定中，这个梯度是不可用的。这可能是因为其高额的计算代价和存储成本，也可能是因为验证集误差在超参数上本质上不可导，例如超参数是离散值的情况。

为了弥补梯度的缺失，我们可以对验证集误差建模，然后通过优化该模型来提出新的超参数猜想。大部分基于模型的超参数搜索算法，都是使用贝叶斯回归模型来估计每个超参数的验证集误差期望和该期望的不确定性。因此，优化涉及到探索（探索高度不确定的超参数，可能带来显著的效果提升，也可能效果很差）和

使用（使用已经确信效果不错的超参数——通常是先前见过的非常熟悉的超参数）之间的权衡。关于超参数优化的最前沿方法还包括 Spearmint (Snoek *et al.*, 2012)，TPE (Bergstra *et al.*, 2011) 和 SMAC (Hutter *et al.*, 2011)。

目前，我们无法明确确定，贝叶斯超参数优化是否是一个能够实现更好深度学习结果或是能够事半功倍的成熟工具。贝叶斯超参数优化有时表现得像人类专家，能够在有些问题上取得很好的效果，但有时又会在某些问题上发生灾难性的失误。看看它是否适用于一个特定的问题是值得尝试的，但目前该方法还不够成熟或可靠。就像所说的那样，超参数优化是一个重要的研究领域，通常主要受深度学习所需驱动，但是它不仅能贡献于整个机器学习领域，还能贡献于一般的工程学。

大部分超参数优化算法比随机搜索更复杂，并且具有一个共同的缺点，在它们能够从实验中提取任何信息之前，它们需要运行完整的训练实验。相比于人类实践者手动搜索，考虑实验早期可以收集的信息量，这种方法是相当低效的，因为手动搜索通常可以很早判断出某组超参数是否是完全病态的。Swersky *et al.* (2014) 提出了一个可以维护多个实验的早期版本算法。在不同的时间点，超参数优化算法可以选择开启一个新实验，“冻结” 正在运行但希望不大的实验，或是 “解冻” 并恢复早期被冻结的，但现在根据更多信息后又有希望的实验。

11.5 调试策略

当一个机器学习系统效果不好时，通常很难判断效果不好的原因是算法本身，还是算法实现错误。由于各种原因，机器学习系统很难调试。

在大多数情况下，我们不能提前知道算法的行为。事实上，使用机器学习的整个出发点是，它会发现一些我们自己无法发现的有用行为。如果我们在一个新的分类任务上训练一个神经网络，它达到 5% 的测试误差，我们没法直接知道这是期望的结果，还是次优的结果。

另一个难点是，大部分机器学习模型有多个自适应的部分。如果一个部分失效了，其他部分仍然可以自适应，并获得大致可接受的性能。例如，假设我们正在训练多层神经网络，其中参数为权重 $\boldsymbol{W}$ 和偏置 $\boldsymbol{b}$。进一步假设，我们单独手动实现了每个参数的梯度下降规则。而我们在偏置更新时犯了一个错误：

$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha, \tag{11.4}$$

其中 α 是学习率。这个错误更新没有使用梯度。它会导致偏置在整个学习中不断变为负值，对于一个学习算法来说这显然是错误的。然而只是检查模型输出的话，该错误可能并不是显而易见的。根据输入的分布，权重可能可以自适应地补偿负的偏置。

大部分神经网络的调试策略都是解决这两个难题的一个或两个。我们可以设计一种足够简单的情况，能够提前得到正确结果，判断模型预测是否与之相符；我们也可以设计一个测试，独立检查神经网络实现的各个部分。

一些重要的调试检测如下所列。

可视化计算中模型的行为：当训练模型检测图像中的对象时，查看一些模型检测到部分重叠的图像。在训练语音生成模型时，试听一些生成的语音样本。这似乎是显而易见的，但在实际中很容易只注意量化性能度量，如准确率或对数似然。直接观察机器学习模型运行其任务，有助于确定其达到的量化性能数据是否看上去合理。错误评估模型性能可能是最具破坏性的错误之一，因为它们会使你在系统出问题时误以为系统运行良好。

可视化最严重的错误：大多数模型能够输出运行任务时的某种置信度量。例如，基于softmax 函数输出层的分类器给每个类分配一个概率。因此，分配给最有可能的类的概率给出了模型在其分类决定上的置信估计值。通常，相比于正确预测的概率最大似然训练会略有高估。但是由于实际上模型的较小概率不太可能对应着正确的标签，因此它们在一定意义上还是有些用的。通过查看训练集中很难正确建模的样本，通常可以发现该数据预处理或者标记方式的问题。例如，街景转录系统原本有个问题是，地址号码检测系统会将图像裁剪得过于紧密，而省略掉了一些数字。然后转录网络会给这些图像的正确答案分配非常低的概率。将图像排序，确定置信度最高的错误，显示系统的裁剪有问题。修改检测系统裁剪更宽的图像，从而使整个系统获得更好的性能，但是转录网络需要能够处理地址号码中位置和范围更大变化的情况。

根据训练和测试误差检测软件：我们往往很难确定底层软件是否是正确实现。训练和测试误差能够提供一些线索。如果训练误差较低，但是测试误差较高，那么很有可能训练过程是在正常运行，但模型由于算法原因过拟合了。另一种可能是，测试误差没有被正确地度量，可能是由于训练后保存模型再重载去度量测试集时出现问题，或者是因为测试数据和训练数据预处理的方式不同。如果训练和测试误差都很高，那么很难确定是软件错误，还是由于算法原因模型欠拟合。这种情况需要进一步的测试，如下面所述。

拟合极小的数据集：当训练集上有很大的误差时，我们需要确定问题是真正的欠拟合，还是软件错误。通常，即使是小模型也可以保证很好地拟合一个足够小的数据集。例如，只有一个样本的分类数据可以通过正确设置输出层的偏置来拟合。通常，如果不能训练一个分类器来正确标注一个单独的样本，或不能训练一个自编码器来成功地精准再现一个单独的样本，或不能训练一个生成模型来一致地生成一个单独的样本，那么很有可能是由于软件错误阻止训练集上的成功优化。此测试可以扩展到只有少量样本的小数据集上。

比较反向传播导数和数值导数：如果读者正在使用一个需要实现梯度计算的软件框架，或者在添加一个新操作到求导库中，必须定义它的 `bprop` 方法，那么常见的错误原因是没能正确地实现梯度表达。验证这些求导正确性的一种方法是比较实现的自动求导和通过 **有限差分**（finite difference）计算的导数。因为

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}, \tag{11.5}$$

我们可以使用小的、有限的 ϵ 近似导数：

$$f'(x) \approx \frac{f(x+\epsilon) - f(x)}{\epsilon}. \tag{11.6}$$

我们可以使用 **中心差分**（centered difference）提高近似的准确率：

$$f'(x) \approx \frac{f(x+\frac{1}{2}\epsilon) - f(x-\frac{1}{2}\epsilon)}{\epsilon}. \tag{11.7}$$

扰动大小 ϵ 必须足够大，以确保该扰动不会由于数值计算的有限精度问题产生舍入误差。

通常，我们会测试向量值函数 $g : \mathbb{R}^m \to \mathbb{R}^n$ 的梯度或Jacobian矩阵。令人遗憾的是，有限差分只允许我们每次计算一个导数。我们可以使用有限差分 mn 次评估 g 的所有偏导数，也可以将该测试应用于一个新函数（在函数 g 的输入输出都加上随机投影）。例如，我们可以将导数实现的测试用于函数 $f(x) = \boldsymbol{u}^T g(\boldsymbol{v}x)$，其中 $\boldsymbol{u}$ 和 $\boldsymbol{v}$ 是随机向量。正确计算 $f'(x)$ 要求能够正确地通过 g 反向传播，但是使用有限差分能够高效地计算，因为 f 只有一个输入和一个输出。通常，一个好的方法是在多个 $\boldsymbol{u}$ 值和 $\boldsymbol{v}$ 值上重复这个测试，可以减少测试忽略了垂直于随机投影的错误的几率。

如果我们可以在复数上进行数值计算，那么使用复数作为函数的输入会有非常高效的数值方法估算梯度 (Squire and Trapp, 1998)。该方法基于如下观察

$$f(x+i\epsilon) = f(x) + i\epsilon f'(x) + O(\epsilon^2), \tag{11.8}$$

$$\text{real}(f(x+i\epsilon)) = f(x) + O(\epsilon^2), \quad \text{image}(\frac{f(x+i\epsilon)}{\epsilon}) = f'(x) + O(\epsilon^2), \tag{11.9}$$

其中 $i = \sqrt{-1}$。和上面的实值情况不同，这里不存在消除影响，因为我们对 f 在不同点上计算差分。因此我们可以使用很小的 ϵ，比如 $\epsilon = 10^{-150}$，其中误差 $O(\epsilon^2)$ 对所有实用目标都是微不足道的。

监控激活函数值和梯度的直方图：可视化神经网络在大量训练迭代后（也许是一个轮）收集到的激活函数值和梯度的统计量往往是有用的。隐藏单元的预激活值可以告诉我们该单元是否饱和，或者它们饱和的频率如何。例如，对于整流器，它们多久关一次？是否有单元一直关闭？对于双曲正切单元而言，预激活绝对值的平均值可以告诉我们该单元的饱和程度。在深度网络中，传播梯度的快速增长或快速消失，可能会阻碍优化过程。最后，比较参数梯度和参数的量级也是有帮助的。正如 (Bottou, 2015) 所建议的，我们希望参数在一个小批量更新中变化的幅度是参数量值 1% 这样的级别，而不是 50% 或者 0.001%（这会导致参数移动得太慢）。也有可能是某些参数以良好的步长移动，而另一些停滞。如果数据是稀疏的（比如自然语言），有些参数可能很少更新，检测它们变化时应该记住这一点。

最后，许多深度学习算法为每一步产生的结果提供了某种保证。例如，在第三部分，我们将看到一些使用代数解决优化问题的近似推断算法。通常，这些可以通过测试它们的每个保证来调试。某些优化算法提供的保证包括，目标函数值在算法的迭代步中不会增加，某些变量的导数在算法的每一步中都是零，所有变量的梯度在收敛时会变为零。通常，由于舍入误差，这些条件不会在数字计算机上完全成立，因此调试测试应该包含一些容差参数。

11.6 示例：多位数字识别

为了端到端的说明如何在实践中应用我们的设计方法论，我们从设计深度学习组件出发，简单地介绍下街景转录系统。显然，整个系统的许多其他组件，如街景车、数据库设施等等，也是极其重要的。

从机器学习任务的视角出发，首先这个过程要采集数据。街景车收集原始数据，然后操作员手动提供标签。转录任务开始前有大量的数据处理工作，包括在转录前使用其他机器学习技术探测房屋号码。

转录项目开始于性能度量的选择和对这些度量的期望值。一个重要的总原则是度量的选择要符合项目的业务目标。因为地图只有是高准确率时才有用，所以为这个项目设置高准确率的要求非常重要。具体地，目标是达到人类水平，98% 的准确

率。这种程度的准确率并不是总能达到。为了达到这个级别的准确率，街景转录系统牺牲了覆盖。因此在保持准确率 98% 的情况下，覆盖成了这个项目优化的主要性能度量。随着卷积网络的改进，我们能够降低网络拒绝转录输入的置信度阈值，最终超出了覆盖 95% 的目标。

在选择量化目标后，我们推荐方法的下一步是要快速建立一个合理的基准系统。对于视觉任务而言，基准系统是带有整流线性单元的卷积网络。转录项目开始于一个这样的模型。当时，使用卷积网络输出预测序列并不常见。开始时，我们使用一个尽可能简单的基准模型，该模型输出层的第一个实现包含 n 个不同的 softmax 单元来预测 n 个字符的序列。我们使用与训练分类任务相同的方式来训练这些 softmax 单元，独立地训练每个 softmax 单元。

我们建议反复细化这些基准，并测试每个变化是否都有改进。街景转录系统的第一个变化受激励于覆盖指标的理论理解和数据结构。具体地，当输出序列的概率低于某个值 t 即 $p(\boldsymbol{y} \mid \boldsymbol{x}) < t$ 时，网络拒绝为输入 $\boldsymbol{x}$ 分类。最初，$p(\boldsymbol{y} \mid \boldsymbol{x})$ 的定义是临时的，简单地将所有 softmax 函数输出乘在一起。这促使我们发展能够真正计算出合理对数似然的特定输出层和代价函数。这种方法使得样本拒绝机制更有效。

此时，覆盖仍低于 90%，但该方法没有明显的理论问题了。因此，我们的方法论建议综合训练集和测试集性能，以确定问题是否是欠拟合或过拟合。在这种情况下，训练和测试集误差几乎是一样的。事实上，这个项目进行得如此顺利的主要原因是有数以千万计的标注样本数据集可用。因为训练和测试集的误差是如此相似，这表明要么是这个问题欠拟合，要么是训练数据的问题。我们推荐的调试策略之一是可视化模型最糟糕的错误。在这种情况下，这意味着可视化不正确而模型给了最高置信度的训练集转录结果。结果显示，主要是输入图像裁剪得太紧，有些和地址相关的数字被裁剪操作除去了。例如，地址 “1849” 的图片可能裁切得太紧，只剩下 “849” 是可见的。如果我们花费几周时间改进确定裁剪区域的地址号码检测系统的准确率，或许也可以解决这个问题。与之不同，项目团队采取了更实际的办法，简单地系统性扩大裁剪区域的宽度，使其大于地址号码检测系统预测的区域宽度。这种单一改变将转录系统的覆盖提高了 10 个百分点。

最后，性能提升的最后几个百分点来自调整超参数。这主要包括在保持一些计算代价限制的同时加大模型的规模。因为训练误差和测试误差保持几乎相等，所以明确表明性能不足是由欠拟合造成的，数据集本身也存在一些问题。

总体来说，转录项目是非常成功的，可以比人工速度更快、代价更低地转录数