Contents

TODO TODO 与原书类似

- a 数
- $\boldsymbol{a}$ 向量
- $\boldsymbol{A}$ 矩阵
- 需要时加 tensor $\mathbf{A}$
- a 随机数
- $\mathbf{a}$ 随机向量
- 需要时加随机矩阵 $\mathbf{A}$
- $\mathbb{A}$ 集合

Chapter 1

前言

自古以来，创造者就梦想着创造能思考的机器。这个愿望至少可以追溯到古希腊的时期。神话人物皮格马利翁 (Pygmalion)、代达罗斯 (Daedalus) 和赫淮斯托斯 (Hephaestus) 都可以被视为传说中的发明家，而加拉蒂亚 (Galatea)、塔洛斯 (Talos) 和潘多拉 (Pandora) 都可以被看作是人造生命 (**???**)。

当人类第一次构思可编程计算机时，就已经在思考计算机能否变得智能，尽管这距造出一台还有一百年多年之久 (**?**)。如今，**人工智能** (AI) 是一个具有许多实际应用和活跃研究课题的领域，并蓬勃发展着。我们指望通过智能软件自动化处理常规劳动、理解语音或图像、帮助医学诊断和支持基础科学研究。

在人工智能的早期，那些对人类智力来说是困难但对计算机来说是相对简单的问题得到迅速解决，比如那些可以通过一系列形式的数学规则来描述的问题。人工智能的真正挑战被证明是解决对人来说很容易执行，但很难形式化描述的任务，也就是我们人类能自动的靠直观解决的问题，比如识别说的话或图像中的脸。

这本书讨论这些更直观的问题一种解决方案。这种解决方案是为了让计算机从经验中学习，并通过层次化概念体系来理解世界，其中每个概念通过与较简单概念之间的联系来定义。让计算机通过经验获取知识可以避免人工形式地指定的计算机需要的所有知识。层次化的概念让计算机通过构建较简单的概念来学习复杂概念。如果绘制出这些概念如何建立在彼此之上的图，我们将得到一张“深”(层次很多) 的图。出于这个原因，我们称这种方法为AI**深度学习**(deep learning)。

许多AI的早期成功发生在相对干净且正式的环境中，计算机不需要具备很多关于世界的知识。例如，IBM 的深蓝 (Deep Blue) 国际象棋系统在 1997 年击败了世界冠军Garry Kasparov(**?**)。当然国际象棋是一个非常简单的领域，仅含有 64 个位置并只能以严格限制的方式移动 32 个棋子。设计一种成功的国际象棋策略是一个巨大的成就，但挑战并不是向计算机描述棋子和允许的移动的困难性。国际象棋完全可以由一个非常简短的、完全形式化的规则列表描述，并可以轻松

由程序员提前提供。

讽刺的是，抽象和形式的任务对人类而言是最困难的脑力任务之一，对计算机而言却属于最容易的。计算机早已能够打败即便是最好的人类棋手，但直到最近才在识别对象或语音的任务中到达匹配人类平均的能力。一个人的日常生活需要关于世界的巨量知识。很多这方面的知识是主观的、直观的，因此很难通过形式的方式表达清楚。为了表现出智能，计算机需要获取同样的知识。人工智能的一个关键挑战就是如何将这些非形式的知识传达给计算机。

一些人工智能项目都力求将关于世界的知识用形式化的语言进行硬编码。计算机可以通过这些形式化语言自动地使用逻辑推理规则来理解声明。这就是所谓的人工智能的**知识图谱**(knowledge base) 方法。这些项目都没有导致重大的成功。其中最著名的项目是的 Cyc (**?**)。Cyc 包括一个推断引擎和一个使用 CycL 语言描述的声明数据库。这些声明是由人类监督者输入的。这是一个笨拙的过程。人们设法设计出具有足够复杂性，能准确描述世界的形式规则。例如，Cyc 不能理解一个关于名为Fred的人在早上剃须的故事 (**?**)。它的推理引擎检测到故事中的不一致性：它知道人没有电气零件，但由于Fred拿着一个电动剃须刀，它认为实体 “FredWhileShaving” 含有电气部件。因此就会产生这样的疑问——Fred在刮胡子的时候是否仍然是一个人。

依靠硬编码的知识体系面对的困难表明，AI系统需要具备自己获取知识的能力，即从原始数据中提取模式的能力。这种能力被称为**机器学习**(machine learning)。引入机器学习使计算机能够解决涉及现实世界知识的问题，并能作出看似主观的决策。所谓**逻辑回归**(logistic regression) 的简单机器学习算法可以决定是否建议剖腹产 (**?**)。所谓**朴素贝叶斯**(naive Bayes) 的简单机器学习算法可以从垃圾邮件中区分合法的电子邮件。

这些简单的机器学习算法的性能在很大程度上依赖于给定数据的**表示**(representation)。例如，当逻辑回归被用于推荐剖腹产时，AI系统不直接检查患者。相反，需要医生告诉系统几条相关的信息，诸如子宫疤痕是否存在。表示患者的每条信息被称为一个特征。逻辑回归学习病人的这些特征如何与各种结果相关联。然而，它丝毫不能影响该特征定义的方式。如果将病人的 MRI 扫描作为逻辑回归的输入，而不是医生正式的报告，它将无法作出有用的预测。MRI 扫描的单一像素与分娩过程中的并发症只有微不足道的相关性。

对表示的依赖是在整个计算机科学乃至日常生活中出现的普遍现象。在计算机科学中，如果数据集合已经过明智地结构化并建立索引，数据操作的处理速度可以成倍的加快 (如搜索)。人们可以很容易地在阿拉伯数字的表示下进行算术运算，但在罗马数字的表示下运算会更耗时。毫不奇怪，表示的选择会对机器学习算法的性能产生巨大的影响。图1.1显示了一个简单的可视化例子。

许多人工智能的任务都可以通过提取一个合适的特征集，然后将这些特征提供给简单的机器学习算法来解决。例如，从声音鉴别说话者的一个有用特征是说话者声道大小的估计。这个特征为判断说话者是一个男性，女性还是儿童提供了

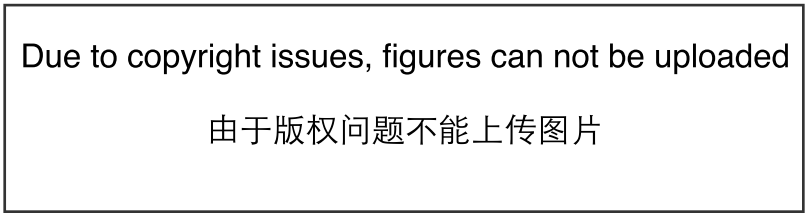


Figure 1.1: TODO

有力线索。

然而，对于许多任务来说，很难知道应该提取哪些特征。例如，假设我们想编写一个程序来检测照片中的车。我们知道，汽车有轮子，所以我们可能会想用车轮的存在作为特征。不幸的是，我们难以准确地从像素值的角度描述一个车轮看起来如何。车轮具有简单的几何形状，但它的图像可以因为环境而变得很复杂，如落在车轮上的阴影、太阳照亮的车轮的金属零件、汽车的挡泥板或者遮挡的车轮一部分的前景物体，等等。

解决这个问题一个途径的是使用机器学习来发现表示本身，而不仅仅把表示映射到输出。这种方法被称为**表示学习**(representation learning)。学习到的表示往往获得比手动设计的表示更好的性能。并且它们只需最少的人工干预，就能让AI系统迅速适应新的任务。表示学习算法只需几分钟就可以为简单的任务发现一个很好的特征集，对于复杂任务则需要几小时到几个月。手动为一个复杂的任务设计特征需要耗费大量的人工时间和精力；甚至需要花费整个社群研究人员几十年时间。

一个表示学习算法的典型例子是**自动编码器**(autoencoder)。自动编码器是组合了将输入转换到不同表示**编码器**(encoder) 函数和将新的表示转回原来形式的**解码器**(decoder) 函数。自动编码器的训练目标是，输入经过编码器和解码器之后尽可能多的保留信息，同时希望新的表示有各种好的属性。不同种类的自动编码器的目标是实现不同种类的属性。

当设计特征或学习特征的算法时，我们的目标通常是分离出能解释观察数据的**变化因素** (factors of variation)。在此背景下，“因素” 这个词仅指代的影响不同来源；因素通常不是乘性组合。这些因素通常是不能被直接观察到的量。相反，他们可能是现实世界中观察不到的物体或者不可观测的力，但影响能观测到的量。他们还存在于人类的思维构造中，为了给观察到的数据提供有用的简化解释或推断原因。它们可以被看作帮助我们了解富有变化的数据的概念或者抽象。当分析语音记录时，变化的因素包括说话者的年龄、性别、他们的口音和他们正在说的词语。当分析汽车的图像时，变化的因素包括汽车的位置、它的颜色、太阳的角度和亮度。

在许多现实世界的人工智能应用困难的一个重要原因是很多因素的变化影响

着我们能够观察到的每一个数据。在夜间，一张图片中红色汽车的单个像素可能是非常接近黑色。汽车轮廓的形状取决于视角。大多数应用程序需要我们*理清*变化的因素并丢弃我们不关心的因素。

当然，从原始数据中提取这样的高层次、抽象的特征是非常困难的。许多这样的变化因素，诸如说话者的口音，只能对数据进行复杂的、接近人类水平的理解来确定。它几乎与获得原来问题的表示一样困难，乍一看，表示学习似乎并不能帮助我们。

深度学习(deep learning) 通过其它较简单的表示来表达复杂表示，解决了表示学习中核心问题。

深度学习让计算机通过较简单概念构建复杂的概念。图1.2显示了深度学习系统通过组合较简单的概念，例如转角和轮廓，转而定义边缘来表示图像中一个人的概念。深度学习模型的典型例子是前馈深度网络或**多层感知机** (MLP)。多层感

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 1.2: TODO

知机仅仅是一个将一组输入值映射到输出的数学函数。该函数由许多较简单的函数组合而构成。我们可以认为每个应用具有不同的数学函数，并为输入提供新的表示。

学习数据的正确表示的想法是解释深度学习的一个观点。另一个观点是深度允许计算机学习一个多步骤的计算机程序。表示的每一层可以被认为是并行执行另一组指令后计算机的存储器状态。更深的网络可以按顺序执行更多的指令。顺序指令提供了极大的能力，因为后面的指令可以参考早期指令的结果。根据深度学习这个观点，一层的激活函数没有必要对解释输入变化的因素进行编码。表示还存储着协助程序执行的状态信息，使输入更加有意义。这里的状态信息是类似于传统计算机程序中的计数器或指针。它与具体的输入内容无关，但有助于模型组织其处理过程。

主要有两种测量模型深度的方式。第一观点是基于评估架构所需执行的顺序指令的数目。我们可以认为这是描述每个给定输入后，计算模型输出的流程图的最长路径。正如两个等价的计算机程序根据不同的语言将具有不同的长度，相同的函数可以被绘制为具有不同深度的流程图，这取决于我们允许使用的单一步骤的函数。图1.3说明了语言的选择怎样给相同的架构两个不同的衡量。

另一种是在深度概率模型中使用的方法，不是将计算图的深度视为模型深度，

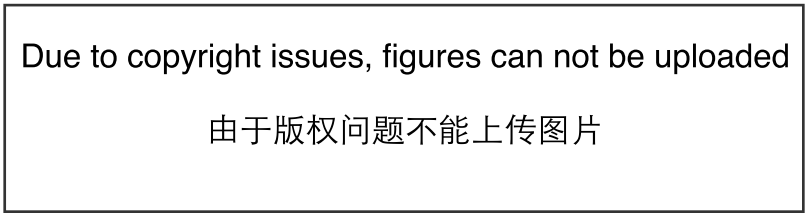


Figure 1.3: TODO

而是将描述概念如何彼此相关的图的深度视为模型深度。在这种情况下，计算每个概念表示的计算流程图的深度可能比概念本身的图更深。这是因为系统对较简单概念的理解可以在给出更复杂概念的信息后进一步细化。例如，一个AI系统观察一张其中一只眼睛在阴影中的脸部图像，最初可能只看到一只眼睛。当检测到的脸部的存在后，它可以推断的第二只眼睛也可能是存在的。在这种情况下，概念的图仅包括两层——关于眼睛的层和关于脸的层，但是，如果我们根据每个概念给出的其它 n 次估计进行改进，计算的图将包括 $2n$ 个层。

由于并不总是清楚计算图的深度或概率模型图的深度哪一个是最相关的，并且由于不同的人选择不同的最小元素集来构建相应的图，导致架构的深度不存在单一的正确值，就像计算机程序的长度不存在单一的正确值。也不存在模型多少深才能被修饰为“深”的共识。<BAD> 但是，深度学习可以可靠地被视为比传统机器学习涉及更大量的学到函数或学到概念组合的模型的研究。

总之，这本书的主题——深度学习是AI的途径之一。具体地讲，它是机器学习的一种，一种允许计算机系统能从经验和数据中得到提高的技术。

我们主张机器学习是构建能在复杂实际环境下运行的AI系统的唯一可行方法。深度学习是一种特定类型的机器学习，通过将世界表示为由较简单概念定义复杂概念，从一般抽象到高级抽象的嵌套概念体系获得极大的能力和灵活性。图1.4说明了这些不同的AI学科之间的关系。图1.5给出了每个学科如何工作的一个高层次的原理。

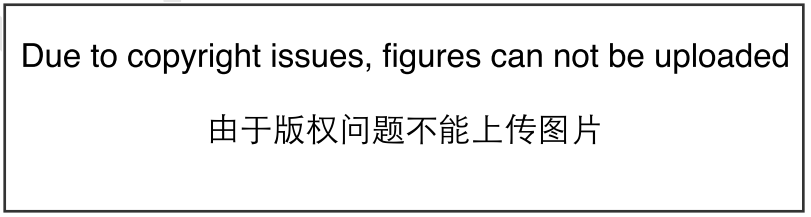


Figure 1.4: TODO

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 1.5: TODO

1.1 谁应该读这本书？

这本书对各类读者都一定用处的，但我们是基于两个主要目标受众而写的。其中一个目标受众是学习机器学习的大学生（本科或研究生），包括那些开始了职业生涯的深度学习和人工智能研究者。另一个目标群体是没有机器学习或统计背景但要迅速在他们的产品或平台开始使用深度学习的软件工程师。深度学习在许多软件领域都已被证明是有用的，包括计算机视觉、语音和音频处理、自然语言处理、机器人技术、生物信息学和化学、电子游戏、搜索引擎、网络广告和金融。

为了最好地适应各类读者，这本书被组织为三个部分。第一部分介绍了基本的数学工具和机器学习的概念。<BAD> 第二部分介绍了本质上已解决的技术、最成熟的深度学习算法。第三部分介绍了被广泛认为是深度学习未来研究重点的但更具猜测性的想法。

读者可以随意跳过不感兴趣或与自己背景不相关的部分。熟悉线性代数、概率和基本机器学习概念的读者可以跳过第一部分，例如，当读者只是想实现一个能工作的系统则不需要阅读超出第二部分的内容。为了帮助读者选择章节，图1.6展示了这本书的高层组织结构的流程图。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 1.6: TODO

我们假设所有读者都具备计算机科学背景。也假设读者熟悉编程，并且对计算的性能问题、复杂性理论、入门级微积分和一些图论的术语有基本的了解。

1.2 深度学习的历史趋势

通过历史背景了解深度学习是最简单的方式。我们指出了深度学习的几个关键趋势，而不是提供详细的历史：

- 深度学习有着悠久而丰富的历史，但伴随着很多反映不同哲学观点名称的尘封而渐渐消逝。
- 可用的训练数据的量已经增加，使得深度学习变得更加有用。
- 随着时间的推移，针对深度学习的计算机软硬件基础设施都有所改善，深度学习模型的规模也随之增长。
- 深度学习已经解决日益复杂的应用，并随着时间的推移，精度不断提高。

1.2.1 神经网络的众多名称和命运变迁

我们期待这本书的许多读者都听说过深度学习这一激动人心的新技术，并为一本书提及关于一个新兴领域的“历史”而感到惊讶。事实上，深度学习的历史可以追溯到 20 世纪 40 年代。深度学习只是*看上去像*一个新的领域，因为在目前流行的前几年它是相对冷门的，同时也因为它被赋予了许多不同的已经消逝的名称，最近才成为所谓的“深度学习”。这个邻域已经更换了很多名称，反映了不同的研究人员和不同观点的影响。

讲述整个综合性的深度学习历史超出了本书的范围。然而，一些基本的背景对理解深度学习是有用的。一般来说，目前为止已经有三次深度学习的发展浪潮：在 20 世纪 40 年代到 60 年代深度学习被称为**控制论**(cybernetics)，20 世纪 80 年代到 90 年代深度学习被誉为**连接机制**(connectionism)，并于 2006 年开始，深度学习在当前的名称下开始复苏。这在图1.7中定量给出。

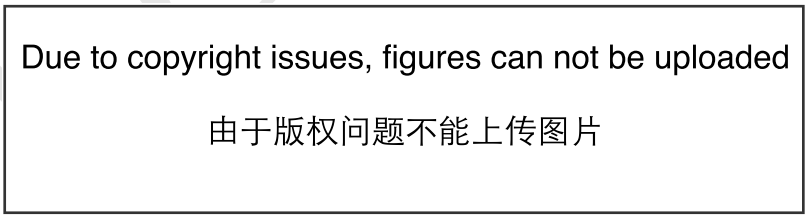


Figure 1.7: TODO

我们现在认识的一些最早的学习算法，旨在模拟生物学习的计算模型，即大脑怎样学习或为什么能学习的模型。其结果是，已经消逝的深度学习的名称之一

——**人工神经网络** (ANN)。此时深度学习模型对应的观点是他们设计的系统是受生物大脑（无论人类大脑或其他动物的大脑）所启发。尽管有些机器学习的神经网络有时被用来理解大脑功能 (**?**)，它们一般都没有被设计成生物功能的真实模型。深度学习的神经观点受两个主要思想启发的。一个想法是，大脑这个例子证明智能行为的可能性，因此建立智能概念上的直接途径是逆向大脑背后的计算原，并复制其功能。另一种看法是，理解大脑和人类智力背后的原则将是非常有趣的，因此机器学习模型除了解决工程应用的能力，如果能阐明这些基本的科学问题也将会很有用。

现代术语 “深度学习” 超越了目前机器学习模型的神经科学观点。学习的*多层次组合*这一更普遍的原则更加吸引人，这可以应用于机器学习框架且不必是受神经启发的。

现代深度学习的最早前身是从神经科学的角度出发的简单线性模型。这些模型被设计为使用一组 n 个输入 $x_1, \ldots, x_n$ 并将它们与一个输出 y 相关联。这些模型将学习一组权重 $w_1, \ldots, w_n$ 并计算它们的输出 $f(\boldsymbol{x}, \boldsymbol{w}) = x_1 w_1 + \cdots + x_n w_n$。这第一波神经网络研究的浪潮被称为控制论，如图1.7所示。

McCulloch-Pitts神经元 (**?**) 是脑功能的早期模型。该线性模型通过测试函数 $f(\boldsymbol{x}\, \boldsymbol{w})$ 的是正还是负来识别两种不同类型的输入。当然，为了使模型对应于期望的类别定义，需要正确地设置权重。这些权重可以由操作人员设定。在 20 世纪 50 年代，感知机 (**??**) 成为第一个能根据了每个类别输入的样例来学习权重的模型。约在同一时期，**自适应线性元件** (ADALINE)，简单地返回函数 $f(\boldsymbol{x})$ 本身的值来预测一个实数 (**?**)，并且还可以学习从数据预测这些数。

这些简单的学习算法大大影响了机器学习的现代景象。用于调节 ADALINE 的权重的训练算法是称为**随机梯度下降**(stochastic gradient descent) 的一种特例。稍加修改的随机梯度下降算法仍然是当今深度学习的主要训练算法。

基于感知机和 ADALINE 中使用的函数 $f(\boldsymbol{x}, \boldsymbol{w})$ 的模型被称为**线性模型**(linear model)。这些模型仍然是最广泛使用的机器学习模型，尽管在许多情况下，它们以不同于原始模型的方式进行*训练*。

线性模型有很多局限性。最著名的是，它们无法学习 XOR 函数，即 $f([0,1], \boldsymbol{w}) = 1, f([1,0], \boldsymbol{w}) = 1$，但是 $f([1,1], \boldsymbol{w}) = 0, f([0,0], \boldsymbol{w}) = 0$。在线性模型中观察到这些缺陷的评论家开始反对受生物学启发的学习 (**?**)。这是神经网络第一次热度较多的下降。

现在，神经科学被视为深度学习研究的一个重要灵感来源，但它已不再是该领域的主要导向。

如今神经科学在深度学习研究中的作用被削弱，主要的原因只是我们根本没有足够的关于大脑信息来作为指导。要获得对大脑实际使用算法的深刻理解，我们需要有能力同时监测（至少是）数千相连神经元的活动。我们不能够做到这一点，甚至连大脑的最简单、最深入研究的部分我们都还远远没有理解 (**?**)。

神经科学已经给了我们依靠单一深度学习算法解决许多不同任务的理由。神经学家们发现，如果将雪貂的大脑重新连接，使视觉信号传送到听觉区域，它们可以学会用大脑的听觉处理区域“看”(?)。这表明，多数哺乳动物大脑的可能使用单一的算法解决大部分大脑可以解决的不同任务。这个假设之前，机器学习研究更加分散，研究人员在不同的社区研究自然语言处理、计算机视觉、运动规划和语音识别。如今，这些应用的社区仍然是独立的，但是深度学习研究小组同时研究许多或甚至所有这些应用领域是很常见的。

我们能够从神经科学得到一些粗略的指南。仅通过计算单元之间的相互作用而变得智能的基本思想是受大脑启发的。新认知机 (?) 受哺乳动物视觉系统的结构启发，引入了一个处理图片的强大模型架构，后来成为了现代卷积网络的基础 (?)，我们将会在??节看到。目前大多数神经网络基于称为**修正线性单元**(rectified linear unit) 的神经单元模型。原始认知机 (?) 受我们关于大脑功能知识的启发，引入了一个更复杂的版本。简化的现代版基于许多观点进化发展，? 和? 援引神经科学作为影响，? 援引更多面向工程的影响。虽然神经科学是灵感的重要来源，它不需要被视为刚性指导。我们知道，实际的神经元与现代修正线性单元计算着非常不同函数，但更接近真实神经网络的系统并没有导致机器学习性能的提升。此外，虽然神经科学已经成功地启发了一些神经网络*架构*，但我们还没有足够的了解生物学习的神经科学，因此在训练这些架构时，不能提供给我们很多关于*学习算法*的指导。

媒体报道经常强调深度学习与大脑的相似性。虽然深度学习研究人员更可能比其他机器学习领域（如核机器或贝叶斯统计工作的研究人员）引用大脑作为影响，人们不应该认为深度学习是对模拟大脑的尝试。现代深度学习从许多领域获取灵感，特别是应用数学的基本内容如线性代数、概率论、信息论和数值优化。虽然一些深度学习的研究人员引用神经科学作为灵感的重要来源，但其他学者完全不关心神经科学。

值得注意的是，了解大脑是如何在算法层面上工作的尝试是鲜活且发展良好的。这项尝试主要是被称为“计算神经科学”，并且是独立于深度学习一个领域。研究人员两个领域之间反复研究是很常见的。深度学习领域主要是关注如何构建智能的计算机系统，用来解决需要智能才能解决的任务，而计算神经科学领域主要是关注构建大脑如何工作的更精确的模型。

在 20 世纪 80 年代，神经网络研究的第二次浪潮在很大程度上是伴随一个被称为**连接机制**(connectionism) 或**并行分布处理**运动而出现的 (??)。连接机制是在认知科学的背景下出现的。认知科学是理解心智，并结合多个不同层次分析的跨学科方法。在 20 世纪 80 年代初期，大多数认知科学家研究符号推理的模型。尽管这很流行，符号模型很难解释大脑如何真正使用神经元实现推理功能。连接主义者开始研究实际能基于神经实现的认知模型 (?)，其中很多复苏的想法可以追溯到心理学家Donald Hebb在 20 世纪 40 年代的工作 (?)。

连接机制的中心思想是，当网络将大量简单计算单元连接在一起时可以实现

智能行为。这种见解同样适用于与计算模型中隐层单元类似作用的生物神经系统中的神经元。

上世纪 80 年代的连接机制运动过程中形成的几个关键概念在今天的深度学习中仍然是非常重要的。

其中一个概念是**分布式表示**(distributed representation)(**?**)。这一想法是，系统每个的输入应该由许多特征表示的，并且每个特征应参与许多可能输入的表示。例如，假设我们有一个能够识别红色、绿色、或蓝色的汽车、卡车和鸟类的视觉系统。表示这些输入的其中一个方法是将九个可能的组合：红卡车，红汽车，红鸟，绿卡车等等使用单独的神经元或隐藏单元激活。这需要九个不同的神经元，并且每个神经必须独立地学习颜色和对象身份的概念。改善这种情况的方法之一是使用分布式表示，即用三个神经元描述颜色，三个神经元描述对象身份。这仅仅需要 6 个神经元而不是 9 个，并且描述红色的神经元能够从汽车、卡车和鸟类的图像中学习红色，而不仅仅是从一个特定类别的图像中学习。分布式表示的概念是本书的核心，我们将在**??**章中更加详细地描述。

连接机制运动的另一个重要成就是反向传播算法的成功运用（训练具有内部表示的深度神经网络）和普及 (**??**)。这个算法虽然已经黯然失色不再流行，但截至写书之时，仍是训练深度模型的主要方法。

在 20 世纪 90 年代，研究人员在使用神经网络进行序列建模的方面取得了重要进展。**?** 和**?** 指出了建模长序列的一些根本数学难题，将在**??**节中描述。**?** 引入长短期记忆(LSTM) 网络来解决这些难题。如今，LSTM在许多序列建模任务中广泛应用，包括 Google 的许多自然语言处理任务。

神经网络研究的第二次浪潮一直持续到上世纪 90 年代中期。基于神经网络等其他AI技术的企业开始在寻求投资的同时，做不切实际野心勃勃的主张。当AI研究不能实现这些不合理的期望时，投资者感到失望。同时，机器学习的其他领域取得进步。核学习机 (**???**) 和图模型 (**?**) 都在很多重要任务上实现了很好的效果。这两个因素导致了神经网络热度的第二次下降，一直持续到 2007 年。

在此期间，神经网络持续在某些任务上获得令人印象深刻的表现 (**??**)。加拿大高级研究所（CIFAR）通过其神经计算和自适应感知（NCAP）研究计划帮助维持神经网络研究。这个计划统一了由分别由Geoffrey Hinton，Yoshua Bengio和Yann LeCun引导的多伦多大学、蒙特利尔大学和纽约大学的机器学习研究小组。CIFAR NCAP 研究计划具有多学科的性质，其中还包括在人类神经科学家和计算机视觉专家。

在那个时间点上，普遍认为深度网络被难以训练的。现在我们知道，从 20 世纪 80 年代就存在的算法能工作得非常好，但是直到在 2006 年前后都没有体现出来。这个问题可能是单纯的因为计算复杂性太高，而以当时可用的硬件难以进行足够的实验。

神经网络研究的第三次浪潮开始于 2006 年的突破。Geoffrey Hinton表明名

为深度信念网络的神经网络可以使用一种称为贪婪逐层训练的策略进行有效地训练 (**?**)，我们将在**??**中更详细地描述。CIFAR 附属的其他研究小组很快表明，同样的策略可以被用来训练许多其他类型的深度网络 (**??**)，并能系统地帮助提高在测试样例上的泛化能力。神经网络研究的这一次浪潮普及了 “深度学习” 这一术语的使用，强调研究人员现在可以训练以前不可能训练的更深的神经网络，并把注意力集中于深度的理论意义 (**????**)。此时，深度神经网络已经优于与之竞争的基于其他机器学习技术以及手工设计函数的AI系统。神经网络流行的第三次浪潮在写这本书的时候还在继续，尽管深度学习的研究重点在这一段时间内发生了巨大变化。第三次浪潮开始把重点放在新的无监督学习技术和深度模型从小数据集进行推广的能力，但现在更多的注意点是在更古老的监督学习算法和深度模型充分利用大型标注数据集的能力。

1.2.2 与日俱增的数据量

人们可能想问，尽管人工神经网络的第一个实验在 20 世纪 50 年代就完成了，为什么深度学习直到最近才被认为是关键技术。自 20 世纪 90 年代以来，深度学习就已经成功用于商业应用，但通常被视为是一种艺术而不是一种技术，且只有专家可以使用的艺术，这种观点持续到最近。确实，要从一个深度学习算法获得良好的性能需要一些技巧。幸运的是，随着训练数据的增加，所需的技巧正在减少。目前在复杂的任务达到与人类表现的学习算法，与 20 世纪 80 年代努力解决的玩具问题 (toy problem) 的学习算法几乎是一样的，尽管这些算法训练的模型经历了变革，简化了极深架构的训练。最重要的新进展是现在我们有了这些算法成功训练所需的资源。图1.8展示了基准数据集的大小如何随着时间的推移显着增加。这种趋势是由社会日益数字化驱动的。由于我们的活动越来越多发生在计算机上，我们做什么也越来越多地被记录。我们的计算机越来越多地联网在一起，变得更容易集中管理这些记录，并将它们整理成适于机器学习应用的数据集。因为 “大数据” 的时代机器学习要容易得多，因为统计估计的主要负担——观察少量数据以在新数据上泛化——已经减轻。截至 2016 年，一个粗略的经验法则是，监督深度学习算法一般在每类给定约 5000 标注样本情况下可以实现可接受的性能，当至少有 1000 万标注样本的数据集用于训练时将达到或超过人类表现。<BAD> 在更小的数据集上成功工作是一个重要的研究领域，我们应特别侧重于如何通过无监督或半监督学习充分利用大量的未标注样本。

1.2.3 与日俱增的模型规模

相对 20 世纪 80 年代较少的成功，现在神经网络非常成功的另一个重要原因是现在我们拥有的计算资源可以运行更大的模型。连接机制的主要见解之一是，当动物的许多神经元一起工作时会变得聪明。单独神经元或小集合的神经元不是特别有用。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 1.8: TODO

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 1.9: TODO

生物神经元没有连接的特别密集。如图1.10所示，几十年来，我们的机器学习模型中每个神经元的连接数量甚至与哺乳动物大脑的在同一数量级。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 1.10: TODO

如图1.11所示，就神经元的总数目而言，直到最近神经网络都是惊人的少。自从引入隐藏单元以来，人工神经网络的规模大约每 2.4 年扩大一倍。这种增长是由更大内存、更快的计算机和更大的可用数据集驱动的。较大的网络能够在更复杂的任务中实现更高的精度。这种趋势看起来将持续数十年。除非有允许迅速扩展的新技术，否则至少直到 21 世纪 50 年代，人工神经网络将才能具备与人脑相同数量级的神经元。生物神经元表示的函数可能比目前的人工神经元更复杂，因此生物神经网络可能比图中描绘的更大。

现在回想起来，比一个水蛭的神经元还少的神经网络不能解决复杂的人工智能问题是不足为奇的。即使是现在从一个计算系统角度来看可能相当大的网络，实际上比相对原始的脊椎动物如青蛙的神经系统更小。

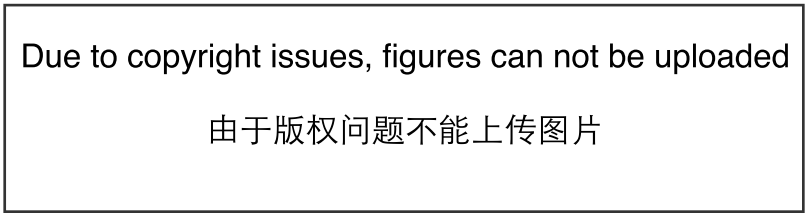


Figure 1.11: TODO

由于有更快的 CPU、通用 GPU 的到来（在5.1.2节中讨论）、更快的网络连接和更好的分布式计算的软件基础设施，模型的规模随着时间的推移不断增加是深度学习历史中最重要的趋势之一。普遍预计这种趋势将很好地持续到未来。

1.2.4 与日俱增的精度、复杂度和对现实世界的冲击

<BAD>20 世纪 80 年代以来，深度学习一直在提高提供识别精度和预测的能力。此外，深度学习持续地被成功应用到越来越广泛的应用中去。

最早的深度模型被用来识别裁剪的很合适且非常小的图像中的单个对象 (?)。自那时以来，神经网络可以处理的图像尺寸逐渐增加。现代对象识别网络能处理丰富的高分辨率照片，并且不要求在被识别的对象附近进行裁剪 (?)。类似地，最早网络只能识别两种对象（或在某些情况下，单一种类的对象的存在与否），而这些现代网络通常识别至少1000个不同类别的对象。对象识别中最大的比赛是每年举行的 ImageNet 大型视觉识别挑战（ILSVRC）。<BAD>深度学习迅速崛起的一个戏剧性的时刻是卷积网络第一次大幅赢得这一挑战，将最高水准的前 5 错误率从26.1%降到15.3%(?)，这意味着该卷积网络产生针对每个图像的可能类别的分级列表，尽管15.3%的测试样例的正确类别不会出现在此列表中的前 5。

此后，深度卷积网络一直能在这些比赛中取胜，截至写书时，深度学习的进步将这个比赛中的前 5 错误率降到3.6%，如图1.12所示。

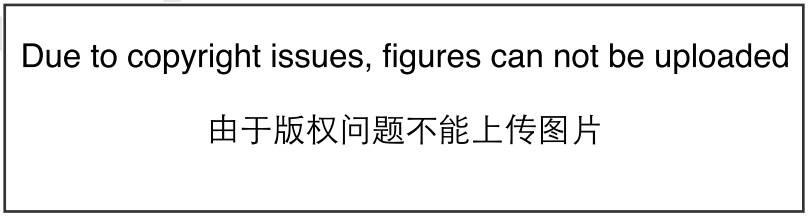


Figure 1.12: TODO

深度学习也对语音识别产生了巨大影响。语音识别在 20 世纪 90 年代提高后，

直到约 2000 年都停滞不前。深度学习的引入 (????) 导致语音识别错误率的陡然下降，有些错误率甚至降低了一半。我们将在5.3节更详细地探讨这个历史。

深度网络在行人检测和图像分割中也有引人注目的成功 (???)，在交通标志分类上取得了超越人类的表现 (?)。

在深度网络的规模和的精度有所提高的同时，它们可以解决的任务也日益复杂。? 表明，神经网络可以学习输出描述图像的整个字符序列，而不是仅仅识别单个对象。此前，人们普遍认为，这种学习需要对序列中的单个元素进行标注 (?)。循环神经网络，如之前提到的LSTM序列模型，现在用于建模序列和其他序列之间的关系，而不是仅仅固定输入之间的关系。这个序列到序列的学习似乎处于另一个应用演进的浪尖：机器翻译 (??)。

日益复杂的趋势已将其推向逻辑结论，即神经图灵机 (?) 的引入，它能学习读取存储单元和向存储单元写入任意内容。这样的神经网络可以从期望行为的样例中学习简单的程序。例如，从杂乱和排好序的样例中学习对一系列数进行排序。这种自我编程技术正处于起步阶段，但原则上未来可以适用于几乎所有的任务。

深度学习的另一个冠军成就是在**强化学习**(reinforcement learning) 领域的扩展。在强化学习的背景下，一个自主体必须通过试错来学习执行任务，而无需人类操作者的任何指导。DeepMind 表明，基于深度学习的强化学习系统能够学会玩 Atari 视频游戏，并在多种任务中可与人类匹敌 (?)。深度学习也显著改善了机器人强化学习的性能 (?)。

许多深度学习应用都是高利润的。现在深度学习被许多顶级的技术公司使用，包括 Google、Microsoft、Facebook、IBM、Baidu、Apple、Adobe、Netflix、NVIDIA 和 NEC。

深度学习的进步也严重依赖于软件基础架构的进展。软件库如 Theano(??)、PyLearn2(?)、Torch(?)、DistBelief(?)、Caffe(?)、MXNet(?) 和 TensorFlow(?) 都能支持重要的研究项目或商业产品。

深度学习也为其他科学做出了贡献。识别对象的现代卷积网络提供给神经科学家可以研究的视觉处理模型 (?)。深度学习也为处理海量数据、在科学领域作出的有效预测提供了非常有用的工具。它已成功地用于预测分子如何相互作用，这能帮助制药公司设计新的药物 (?)、搜索的亚原子粒子 (?) 和自动解析用于构建人脑三维图的显微镜图像 (?)。我们期待深度学习未来出现在越来越多的科学领域。

总之，深度学习是机器学习的一种方法，过去几十年的发展中，它深深地吸收了我们关于人脑、统计学与应用数学的知识。近年来，深度学习的普及性和实用性有了极大的发展，这在很大程度上得益于更强大的计算机、更大的数据集和能够训练更深网络的技术。未来几年充满了进一步提高深度学习并将它带到新领域挑战和机遇。

Chapter 2

数值计算

机器学习算法通常需要大量的数值计算。<BAD> 数值计算通常指的是通过迭代地更新解来解决数学问题的算法，而不是解析地提供正确解的符号表达。常见的操作包括优化（找到最小化或最大化函数值的参数）和线性方程的求解。对数字计算机来说，即使计算只涉及实数的函数也是困难的，因为实数无法在有限内存下精确表示。

2.1 上溢和下溢

在数字计算机上实现连续数学的基本困难是，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，几乎都会引入一些近似误差。在许多情况下，这仅仅是舍入误差。如果在理论上可行的算法没有被设计为最小化舍入误差的累积，可能会在实践中失效，因此舍入误差是有问题的(特别是许多操作复合时)。

一种特别的毁灭性舍入误差是**下溢**(underflow)。当接近零的数被四舍五入为零时发生下溢。许多函数会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抛出异常，有些会返回一个非数字 (not-a-number) 的占位符）或取零的对数（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。进一步的运算通常将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

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} \tag{2.1}$$

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 $\log \text{softmax}$。可以使用相同的技巧稳定 $\log \text{softmax}$ 函数。

在大多数情况下，我们没有明确地对实现本书描述的各种算法时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，有可能在实现一个新的算法时自动保持数值稳定。Theano(**??**) 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。

2.2 病态条件数

条件数表明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是可能是有问题的，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

$$\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|. \tag{2.2}$$

这是最大和最小特征值的模之比。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，病态条件数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。

2.3 基于梯度的优化方法

大多数深度学习算法涉及某种形式的优化。优化指的是改变 $\boldsymbol{x}$ 以最小化或最大化的某个函数 $f(\boldsymbol{x})$ 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。

我们希望最小化或最大化的函数叫做**目标函数**(objective function) 或**判据**(criterion)。当我们对其进行最小化时，我们也把它叫做**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。

我们经常使用一个上标 $*$ 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。<BAD> 换句话说，它表明需要如何缩放输入的小变化以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 $f(x)$。这个技术被称为**梯度下降**(gradient descent)(**?**)。图2.1展示了一个例子。

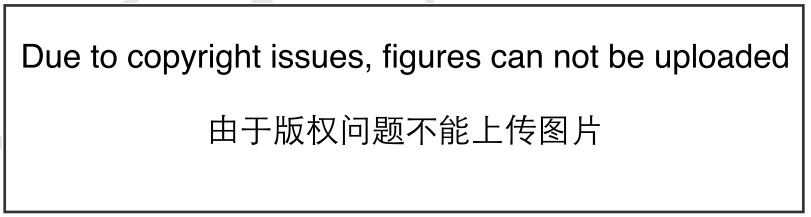


Figure 2.1: TODO

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 points) 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) 是 $f(x)$ 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点。这些点被称为**鞍点**(saddle points)。见图2.2给出的各种临界点的例子。

使 $f(x)$ 取得全局最小值的点是**全局最小点**(global minimum)。只有一个全局

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 2.2: TODO

最小点或存在多个全局最小点的函数是有可能的。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，我们优化的函数可能含有许多不是最优的局部极小点，或许多被非常平坦的区域包围的鞍点。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 f 非常小的值，但在任何形式意义下并不一定是最小。见图2.3的例子。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 2.3: TODO

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)。

我们必须利用**偏导数**(partial derivatives) 的概念针对具有多维输入的函数。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 的元素是 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：

$$\min_{\boldsymbol{u}, \boldsymbol{u}^\top \boldsymbol{u} = 1} \boldsymbol{u}^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{2.3}$$

$$= \min_{\boldsymbol{u}, \boldsymbol{u}^\top \boldsymbol{u} = 1} \|\boldsymbol{u}\|_2 \|\nabla_{\boldsymbol{x}} f(\boldsymbol{x})\|_2 \cos\theta \tag{2.4}$$

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

$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{2.5}$$

其中 ϵ 为学习速率，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）。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）的一般概念可以被推广到离散空间。递增带有离散参数的目标函数被称为**爬山**(hill climbing) 算法 (?)。

2.3.1 梯度之上：雅各比和海森矩阵

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**雅各比**(Jacobian) 矩阵。具体来说，如果我们有一个函数：$\boldsymbol{f}: \mathbb{R}^m \to \mathbb{R}^n$，$\boldsymbol{f}$ 的雅各比矩阵 $\boldsymbol{J} \in \mathbb{R}^{n \times m}$ 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 $f''(x)$。二阶导数告诉我们的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，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2.4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。

当我们的函数具有多维输入时，二阶导数也有很多。可以这些导数合并成一

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 2.4: TODO

个矩阵，称为**海森**(Hessian) 矩阵。海森矩阵 $\boldsymbol{H}(f)(\boldsymbol{x})$ 定义为

$$\boldsymbol{H}(f)(\boldsymbol{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x}). \tag{2.6}$$

海森等价于梯度的雅各比矩阵。

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：

$$\frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x}) = \frac{\partial^2}{\partial x_j \partial x_i} f(\boldsymbol{x}). \tag{2.7}$$

这意味着 $H_{i,j} = H_{j,i}$，因此海森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海森几乎处处都是对称的。因为海森矩阵是实对称的，我们可以将其分解成一组实特征值和特征向量的正交。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有更大的权重。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：

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{g} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}^{(0)}), \tag{2.8}$$

其中 $\boldsymbol{g}$ 是梯度，$\boldsymbol{H}$ 是 $\boldsymbol{x}^{(0)}$ 点的海森。如果我们使用学习速率 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

$$f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \epsilon \boldsymbol{g}^\top \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^\top \boldsymbol{H}\boldsymbol{g}. \tag{2.9}$$

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。当这最后一项太大时，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 将永远导致 f 的下降。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

$$\epsilon^* = \frac{\boldsymbol{g}^\top \boldsymbol{g}}{\boldsymbol{g}^\top \boldsymbol{H}\boldsymbol{g}} \tag{2.10}$$

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，海森的特征值决定了学习速率的量级。

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。利用海森的特征值分解，我们可以将二阶导数测试扩展到多维情况。在临界点处 ($\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = 0$)，我们通过检测海森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。当海森是正定的 (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，当海森是负定的 (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)。如果海森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2.5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。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 2.5: TODO

多维情况下，单个点处每个方向上的二阶导数是不同。海森的条件数衡量这

些二阶导数的变化范围。当海森的条件数很差时，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。病态条件数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的正曲率方向上升。这通常意味着步长太小，以致于在其它较小曲率的方向上进展不明显。见图2.6的例子。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 2.6: TODO

使用海森矩阵的信息指导搜索可以解决这个问题。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 $f(\boldsymbol{x})$:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{H}(f)(\boldsymbol{x}^{(0)})(\boldsymbol{x} - \boldsymbol{x}^{(0)}). \tag{2.11}$$

接着通过计算，我们可以得到这个函数的临界点:

$$\boldsymbol{x}^* = \boldsymbol{x}^{(0)} - \boldsymbol{H}(f)(\boldsymbol{x}^{(0)})^{-1} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)}) \tag{2.12}$$

当 f 是一个正定二次函数时，牛顿法只要应用一次式2.12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 2.12。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**??**节所讨论的，当附近的临界点是最小点（海森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。使用海森矩阵的优化算法被称为**二阶最优化算法** (second-order optimization algorithms)(**?**)，如牛顿法。

在本书的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：

$$\forall \boldsymbol{x}, \forall \boldsymbol{y}, |f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq \mathcal{L}\|\boldsymbol{x} - \boldsymbol{y}\|_2. \tag{2.13}$$

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。

最成功的特定优化邻域或许是**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——即海森处处半正定的函数。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。凸优化仅用作的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，深度学习的背景下的凸优化的重要性大大减少。有关凸优化的详细信息，见**?** 或**?**。

2.4 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 (**?**)。

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[1]是针对约束优化非常通用的解决方案。

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

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 constraints)，涉及 $h^{(j)}$ 的不等式称为**不等式约束**(inequality constraints)。

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：

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x}). \tag{2.14}$$

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

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) \tag{2.15}$$

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

$$\min_{\boldsymbol{x} \in \mathbb{S}} f(\boldsymbol{x}). \tag{2.16}$$

这是因为当约束满足时，

$$\max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\boldsymbol{x}) \tag{2.17}$$

而违反任意约束时，

$$\max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = \infty. \tag{2.18}$$

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：

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} -f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x}). \tag{2.19}$$

我们也可将其转换为在外层最大化的一个问题：

$$\max_{\boldsymbol{x}} \min_{\boldsymbol{\lambda}} \min_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) - \sum_j \alpha_j h^{(j)}(\boldsymbol{x}). \tag{2.20}$$

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。

可以使用一组简单性质描述约束优化问题的最优点。这些性质称为**Karush–Kuhn–Tucker** (KKT) 条件 (**??**)。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**?**。

2.5 实例：线性最小二乘

假设我们希望找到最小化下式的 $\boldsymbol{x}$ 的值

$$f(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 \tag{2.21}$$

专门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：

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \boldsymbol{A}^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top \boldsymbol{b}. \tag{2.22}$$

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

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正的函数是二次的，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。

现在假设我们希望最小化同样的函数，但受 $\boldsymbol{x}^\top \boldsymbol{x} \leq 1$ 的约束。要做到这一点，我们引入Lagrangian

$$L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda(\boldsymbol{x}^\top \boldsymbol{x} - 1). \tag{2.23}$$

现在，我们解决以下问题

$$\min_{\boldsymbol{x}} \max_{\lambda, \lambda \geq 0} L(\boldsymbol{x}, \lambda) \tag{2.24}$$

可以用Moore-Penrose伪逆：$\boldsymbol{x} = \boldsymbol{A}^+ \boldsymbol{b}$ 找到无约束最小二乘问题的最小范数解。如果这一点是可行，那么这也是约束问题的解。否则，我们必须找到约束是活跃的解。关于 $\boldsymbol{x}$ 对Lagrangian微分，我们得到方程

$$\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top \boldsymbol{b} + 2\lambda \boldsymbol{x} = 0. \tag{2.25}$$

这就告诉我们，该解的形式将会是

$$\boldsymbol{x} = (\boldsymbol{A}^\top \boldsymbol{A} + 2\lambda \boldsymbol{I})^{-1} \boldsymbol{A}^\top \boldsymbol{b}. \tag{2.26}$$

λ 的选择必须使结果服从约束。我们可以关于 λ 进行梯度上升找到这个值。为了做到这一点，观察

$$\frac{\partial}{\partial \lambda} L(\boldsymbol{x}, \lambda) = \boldsymbol{x}^\top \boldsymbol{x} - 1. \tag{2.27}$$

当 $\boldsymbol{x}$ 的范数超过 1 时，该导数是正的，所以为了跟随导数上坡并相对 λ 增加Lagrangian，我们需要增加 λ。因为 $\boldsymbol{x}^\top \boldsymbol{x}$ 的惩罚系数增加了，求解关于 $\boldsymbol{x}$ 的线性方程现在将得到具有较小范数的解。求解线性方程和调整 λ 的过程一直持续到 $\boldsymbol{x}$ 具有正确的范数并且关于 λ 的导数是 0。

本章总结了开发机器学习算法所需的数学基础。现在，我们已经做好了建立和分析一些成熟学习系统的准备。

Chapter 3

深度学习的正则化

机器学习中的一个核心问题是设计不只在训练数据表现好，而且能在新输入上泛化的算法。在机器学习中，许多策略被明确设计为以增大训练误差为代价来减少测试误差。这些策略统称为正则化。正如我们会看到的，深度学习工作者可以使用许多形式的正则化。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**??**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，我们会更详细地描述正则化，重点描述深度模型（或模块）的正则化策略。

本章某些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，本章的大多数内容涉及这些基本概念到特定神经网络的扩展。

在**??**节，我们将正则化定义为“旨在减少学习算法的泛化误差而不是训练误差的修改”。目前有许多正则化策略。有些向机器学习模型添加额外的约束，如增加对参数的限制。有些向目标函数增加额外项，对应于参数值的软约束。如果仔细选择，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，这些约束和惩罚设计为编码特定类型的先验知识。其他时候，这些约束和惩罚的目的是表达对简单模型的一般偏好，以便提高泛化能力。有时候，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，如集成方法，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基于对估计量进行正则化。估计量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**??**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——对应于欠

拟合和偏差引入，(2) 匹配真实数据生成过程，(3) 除了包含真实的数据生成过程，还包含了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进入到第二个情况。

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，甚至近似的都不包含。我们几乎从来无法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领域，其中真实数据的生成过程几乎可以肯定是模型族之外。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正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性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，在实际的深度学习场景中我们几乎总是发现，最好的拟合模型（最小化泛化误差的意义上）是一个适当正则化的大型模型。

现在我们回顾几种创建这样的大型深度正则化模型的策略。

3.1 参数范数惩罚

正则化在深度学习的出现前就已经应用了数十年。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 $\Omega(\boldsymbol{\theta})$，以限制模型 (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 $\tilde{J}$:

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}) \tag{3.1}$$

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 就是没有正则化。越大的 α，对应于越多的正则化惩罚。

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参数 $\boldsymbol{\theta}$ 的规模（或在某些衡量下参数子集的规模)。参数规范 Ω 的不同选择可以导致不同优先解。在本节中，我们讨论各种范数惩罚模型参数时的影响。

在探究不同范数的正则化表现之前，我们需要提一下，在神经网络中我们通常只对每一层仿射变换的权重做惩罚而不对偏置做正则惩罚。通常使用比拟合权重更少的数据, 就能精确拟合偏置。每个权重指定两个变量如何相互作用。需要在各种条件下观察这两个变量才能拟合权重。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引起太大方差。另外，正则化的偏置参数

可能会引起显著的欠拟合。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 $\boldsymbol{w}$ 和不用正则化的参数)。

在神经网络的情况下，有时希望对网络的每个层使用单独的惩罚，并分配不同的 α 系数。搜索多个正确超参数的代价很大，因此在所有层使用相同权重衰减以减少搜索空间是合理的。

3.1.1 L^2 参数正则化

在**??**节中我们已经看到最简单和最常见的参数范数惩罚是，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：

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}), \tag{3.2}$$

与之对应的梯度为

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}). \tag{3.3}$$

使用单步梯度下降更新的权重，即执行以下更新：

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon(\alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})). \tag{3.4}$$

换种写法就是：

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}). \tag{3.5}$$

我们可以看到，在加入权重衰减后会修改学习规则——在每一步执行通常的梯度更新之前对权重向量乘以一个常数因子以收缩权重向量。这是单一步骤发生的变化。但是，在训练的整个过程会发生什么？

我们进一步简化分析，令 $\boldsymbol{w}^*$ 为没有正则化的目标函数取得最小训练误差的权重向量，即 $\boldsymbol{w}^* = \text{argmin}_{\boldsymbol{w}} J(\boldsymbol{w})$，并在 $\boldsymbol{w}^*$ 的邻域作二次近似。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下)，则该近似是完美的。近似的 $\hat{J}(\boldsymbol{\theta})$ 如下

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*), \tag{3.6}$$

[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，令人惊讶的是仍有正则化效果，并且更接近真实值将获得更好的结果。当我们不知道正确的值应该是正还是负，零是有意义的默认值。由于将模型参数正则化为零的情况更常见，我们将关注这种特殊情况。

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

当 $\hat{J}$ 取最小时，其梯度

$$\nabla_{\boldsymbol{w}}\hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) \tag{3.7}$$

为 0。

为了研究权重衰减的影响，我们将权重衰减的梯度加到式3.7中。现在我们要找正则化版本 $\hat{J}$ 的最小值。我们使用变量 $\tilde{\boldsymbol{w}}$ 表示最小值的位置:

$$\alpha\tilde{\boldsymbol{w}} + \boldsymbol{H}(\tilde{\boldsymbol{w}} - \boldsymbol{w}^*) = 0 \tag{3.8}$$

$$(\boldsymbol{H} + \alpha\boldsymbol{I})\tilde{\boldsymbol{w}} = \boldsymbol{H}\boldsymbol{w}^* \tag{3.9}$$

$$\tilde{\boldsymbol{w}} = (\boldsymbol{H} + \alpha\boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{w}^* \tag{3.10}$$

当 α 趋向于 0, 正则化的解 $\tilde{\boldsymbol{w}}$ 趋向 $\boldsymbol{w}^*$。那么当 α 增加时会发生什么？因为 $\boldsymbol{H}$ 是实对称的，我们可以将其分解为一个对角矩阵 $\boldsymbol{\Lambda}$ 和一个正交特征向量 $\boldsymbol{Q}$，并且 $\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top$。将其应用于式3.10，可得:

$$\tilde{\boldsymbol{w}} = (\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top + \alpha\boldsymbol{I})^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^* \tag{3.11}$$

$$= [\boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})\boldsymbol{Q}^\top]^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^* \tag{3.12}$$

$$= \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^*. \tag{3.13}$$

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，与 $\boldsymbol{H}$ 第 i 个特征向量对齐的 $\boldsymbol{w}^*$ 的分量根据 $\frac{\lambda_i}{\lambda_i+\alpha}$ 因子缩放。 (不妨查看 |||c||| 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 $\lambda_i \ll \alpha$ 的分量将会被缩小到几乎为零。这种效应在图3.1示出。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 3.1: TODO

只有显著减小目标函数方向的参数会被保留得相对完好。无助于目标函数减小的方向，即海森的小的特征值告诉我们，在这个方向的移动不会显著增加梯度。对应于这种不重要方向的分量会随训练过程中使用的正则化而衰减掉。

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。这些影响怎么影响具体的机器学习？我们可以研究线性回归，其真实代价函数是二次的，因此适合目前为止我们使用的分析。再次应用分析，我们将能够获得相同结果的一个特例，但这次通过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：

$$(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y})^\top(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y}). \tag{3.14}$$

我们添加 L^2 正则项后，目标函数变为

$$(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y})^\top(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y})+\frac{1}{2}\alpha\boldsymbol{w}^\top\boldsymbol{w}. \tag{3.15}$$

这会将普通方程的解从

$$\boldsymbol{w}=(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} \tag{3.16}$$

变为

$$\boldsymbol{w}=(\boldsymbol{X}^\top\boldsymbol{X}+\alpha\boldsymbol{I})^{-1}\boldsymbol{X}^\top\boldsymbol{y} \tag{3.17}$$

式3.16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3.17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于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）的特征的权重将会被收缩。

3.1.2 L^1 参数正则化

L^2权重衰减是权重衰减最常见的形式，还有其他的方法来惩罚模型参数的大小。另一种选择是使用 L^1正则化。

对模型参数 $\boldsymbol{w}$ 的 L^1正则化形式定义为：

$$\Omega(\boldsymbol{\theta})=\|\boldsymbol{w}\|_1=\sum_i|w_i|, \tag{3.18}$$

即各个参数的绝对值之和[2]。现在我们将讨论 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

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

L^2正则化之间的差异。与 L^2权重衰减类似，L^1权重衰减的强度也可以通过缩放惩罚项 Ω 的正超参数 α 控制。因此，正则化的目标函数$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ 如下给出

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \|\boldsymbol{w}\|_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \tag{3.19}$$

对应的梯度 (实际上是次梯度)：

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \operatorname{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \tag{3.20}$$

其中 $\operatorname{sign}(\boldsymbol{w})$ 只是简单地取 $\boldsymbol{w}$ 各个元素的符号。

观察式3.20，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贡献不再是线性地缩放每个 w_i；而是用与 $\operatorname{sign}(w_i)$ 同号的常数因子。这种形式的梯度的一个后果是，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)。

我们可以通过泰勒级数表示代价函数是二次的简单线性模型。或者我们可以设想，这是一个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

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*), \tag{3.21}$$

同样，$\boldsymbol{H}$ 是 J 在 $\boldsymbol{w}^*$ 处计算的海森矩阵 (关于 $\boldsymbol{w}$)。

由于 L^1 惩罚项在满的、一般的海森的情况下，无法得到直接干净的代数表达式，因此我们将进一步简化假设海森是对角的，即 $\boldsymbol{H} = \operatorname{diag}([H_{1,1}, \ldots, H_{n,n}])$，其中每个 $H_{i,i} > 0$。如果用于线性回归问题的数据已被预处理 (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一个关于参数的求和：

$$\hat{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}^*; \boldsymbol{X}, \boldsymbol{y}) + \sum_i \left[\frac{1}{2} H_{i,i} (w_i - w_i^*)^2 + \alpha |w_i| \right]. \tag{3.22}$$

有解析解（对每一维 i）可以最小化这个近似代价函数，如下列形式：

$$w_i = \operatorname{sign}(w_i^*) \max\{|w_i^*| - \frac{\alpha}{H_{i,i}}, 0\} \tag{3.23}$$

考虑所有 i 且 $w_i^* > 0$ 的情形，会有两种可能输出：

1. $w_i^* \leq \frac{\alpha}{H_{i,i}}$ 的情况。正则化后的目标中 w_i 的最优值是 $w_i = 0$。这是因为在方向 i 上 $J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ 对于 $\hat{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ 的贡献受到压制，L^1正则化项将 w_i 推向 0。

2. $w_i^* > \frac{\alpha}{H_{i,i}}$ 的情况。在这种情况，正则化不会将 w_i 的最优值移向 0，而仅仅在那个方向移动 $\frac{\alpha}{H_{i,i}}$ 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(减少 $\frac{\alpha}{H_{i,i}}$) 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。在此稀疏性指的是一些参数具有 0 的最优值。L^1正则化的稀疏性是相比 L^2正则化是质的不同。式3.13给出了 L^2正则化的解 $\tilde{\boldsymbol{w}}$。如果我们使用 L^1正则化分析时海森$\boldsymbol{H}$ 为对角正定的假设，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 0，那 $\tilde{w}_i$ 也会保持非 0。这表明 L^2正则化不会导致参数变得稀疏，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作**特征选择**(feature selection) 机制。特征选择从可用的特征子集选择应该使用的子集，简化了机器学习问题。特别是著名的 LASSO(**?**) (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**??**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，用于正则化一个代价函数的惩罚项 $\alpha\Omega(\boldsymbol{w}) = \alpha\sum_i |w_i|$ 与先验是各向同性拉普拉斯分布 $|||\mathrm{c}|||(\boldsymbol{w} \in \mathbb{R}^n)$ 的MAP贝叶斯推断的最大化对数先验项是等价的:

$$\log p(\boldsymbol{w}) = \sum_i \log \text{ Laplace}(w_i; 0, \frac{1}{\alpha}) = -\alpha \left\|\boldsymbol{w}\right\|_1 + n\log\alpha - n\log 2. \tag{3.24}$$

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。

3.2 作为约束的范数惩罚

考虑通过参数范数正则化的代价函数:

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}) \tag{3.25}$$

回顾2.4节我们可以通过构造一个广义 Lagrange 函数来最小化受约束的函数，即在原始目标函数加上一系列惩罚项。每个惩罚是一个系数之间的乘积，称为**Karush－Kuhn－Tucker**(Karush－Kuhn－Tucker) 乘子，以及一个表示约束是否满足的函数。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

$$\mathcal{L}(\boldsymbol{\theta}, \alpha; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha(\Omega(\boldsymbol{\theta}) - k). \tag{3.26}$$

这个约束问题的解由下式给出

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} \max_{\alpha, \alpha > 0} \mathcal{L}(\boldsymbol{\theta}, \alpha). \tag{3.27}$$

如2.4节中描述，解决这个问题需要同时改变 $\boldsymbol{\theta}$ 和 α。2.5给出了一个带 L^2 约束的线性回归实例。许多不同的程序是可能的，有些可能会利用梯度下降而其他可能使用梯度为 0 的解析解，但在所有程序中 α 在 $\Omega(\boldsymbol{\theta}) > k$ 时必须增加，在 $\Omega(\boldsymbol{\theta}) < k$ 时必须减小。所有正的 α 鼓励 $\Omega(\boldsymbol{\theta})$ 收缩。最佳值 α^* 也将鼓励 $\Omega(\boldsymbol{\theta})$ 收缩，但不会如 $\Omega(\boldsymbol{\theta})$ 小于 k 时那么强烈。

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：

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \alpha) = \operatorname*{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha^* \Omega(\boldsymbol{\theta}) \tag{3.28}$$

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。通常我们不知道通过使用权重衰减的系数 α^* 约束的区域大小，因为 α^* 的值不直接告诉我们 k 的值。原则上可以解得 k，但 k 和 α^* 之间的关系取决于的 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，将导致一个较小的约束区域。较小的 α，将导致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2.4所描述，我们能修改下降算法（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导致非凸优化程序而陷入局部极小 (对应于小的 $\boldsymbol{\theta}$)。当训练神经网络，这通常表现为训练带有几个 “死单元” 的神经网络。这些单元不会对网络学到的函数的行为有太大贡献，因为进入或离开他们的权重都非常小。当使用权重范数的惩罚训练，这些配置可以是局部最优，即使可能通过增加权重以显著减少 J。通过重投影实现的显式约束可以在这些情况下工作得更好，因为它们不鼓励权重接近原点。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，因此是另一个好处。当使用较高的学习率时，很可能进入的正反馈，即大的权重诱导大梯度，然后诱发权重的大更新。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的权重无限制持续增加。? 建议结合使用约束和高学习速率，能更快地参数空间探索，并保持一定的稳定性。

? 尤其推荐由? 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实现。

3.3 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，都依赖于求逆矩阵 $\boldsymbol{X}^\top \boldsymbol{X}$。只要 $\boldsymbol{X}^\top \boldsymbol{X}$ 是奇异的这就是不可能的。每当数据生成分布的一些方向上真的没有差异时，或因为例子较少（即相对输入特征，$\boldsymbol{X}$ 的列来说）而在一些方向没有观察到方差，这个矩阵都是奇异的。在这种情况下，正则化的许多形式对应于求逆 $\boldsymbol{X}^\top \boldsymbol{X} + \alpha \boldsymbol{I}$。这个正则化矩阵可以保证是可逆的。

相关矩阵可逆时，这些线性问题有封闭形式的解。没有闭合形式解的问题也可能是欠定。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 $\boldsymbol{w}$ 是能够实现完美分类，那么 $2\boldsymbol{w}$ 也以较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，理论上将永远不会停止。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，权重衰减将导致梯度下降不再增加权重的大小。

使用正则化来解决欠定问题的想法超出了机器学习范畴。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??节看到，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：

$$\boldsymbol{X}^+ = \lim_{\alpha \searrow 0} (\boldsymbol{X}^\top \boldsymbol{X} + \alpha \boldsymbol{I})^{-1} \boldsymbol{X}^\top. \tag{3.29}$$

现在我们可以将式3.29看作执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3.29是式3.17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。

3.4 数据集增强

让机器学习模型泛化得更好的最好办法是用更多的数据来训练。当然，在实践中，我们拥有数据量是有限的。解决这个问题的一种方法是创建假数据并把它添加到训练集。对于一些机器学习任务，创造新的假数据是相当简单。

对分类来说这种方法是最简单的。<BAD> 分类器需要一个复杂的高维输入 $\boldsymbol{x}$，并用单一类别的身份 y 总结 $\boldsymbol{x}$。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，在一个密度估计任务中的产生新的假数据是困难的，除非我们已经解决了密度估计问题。

数据集增强对一个具体的分类问题来说是一个特别有效的方法：对象识别。图像是高维的并包括各种巨大的变化因素，其中有许多可以容易地模拟。即使模型已使用卷积和池化技术（**??**）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，不能应用改变正确类别的转换。例如，光学字符识别任务需要认识 “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° 并不是适当的数据集增强方式。

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 (**?**)。

在神经网络的输入层注入噪声 (**?**) 也可以被看作是数据增强的一种形式。对于许多分类甚至一些回归任务，即使小的随机噪声被加到输入，任务仍应该是能解决的。然而，神经网络被证明对噪声不是非常健壮 (**?**)。改善神经网络健壮性的方法之一是简单地将随机噪声施加到其输入再进行训练。输入噪声注入是一些无监督学习算法的一部分，如降噪自动编码器(**?**)。当噪声被施加到隐藏单元也是可行的，这可以被看作是在多个抽象层上进行的数据集增强。**?** 最近表明，噪声的幅度被细心调整后，该方法是非常高效的。dropout，一个将在3.12节描述的强大的正则化策略，可以被看作是通过乘噪声来构建新输入的过程。

当比较机器学习基准测试的结果时，将其采取的数据集增强的影响考虑在内是很重要的。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。当比较机器学习算法 A 和机器学习算法 B 时，应该确保这两个算法使用同一个人工设计的数据集增强方案进行评估。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。有时候，确定实验是否已

经适当控制需要主观判断。例如，噪声注入输入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，而特定于一个应用邻域（如随机地裁剪图像）的操作被认为是独立的预处理步骤。

3.5 噪声鲁棒性

5.2.2节已经将噪声作用于输入的数据集增强策略。对于某些模型，在模型的输入加上方差极小的噪音等价于对权重施加范数惩罚 (**??**)。在一般情况下，噪声注入远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。向隐藏单元添加噪音是值得单独讨论重要的话题；在3.12节所述dropout算法这种做法主要发展方向。

另一种正则化模型的噪声使用方式是将其加到的权重。这项技术主要被用于循环神经网络的情况下 (**??**)。这可以解释为一个关于权重的贝叶斯推断的随机实现。使用贝叶斯处理学习过程将权重视为不确定的，且能通过概率分布表示的这种不确定性。向权重添加噪声是反映这种不确定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正则化形式等同，鼓励要学习函数的稳定性。我们研究回归的情形，也就是训练一个将一组特征 $\boldsymbol{x}$ 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：

$$J = \mathbb{E}_{p(x,y)}[(\hat{y}(\boldsymbol{x}) - y)^2]. \tag{3.30}$$

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

现在我们假设我们在对每个输入表示上添加网络权重的随机扰动 $\epsilon_{\boldsymbol{W}} \sim \mathcal{N}(\boldsymbol{\epsilon}; 0, \eta\boldsymbol{I})$ 想象我们有一个标准的 l 层 MLP。我们将扰动模型记为 $\hat{y}_{\epsilon_{\boldsymbol{W}}}(\boldsymbol{x})$。尽管有噪声的注入，我们仍对减少网络输出误差的平方感兴趣。因此目标函数变为：

$$\tilde{J}_{\boldsymbol{W}} = \mathbb{E}_{p(\boldsymbol{x},y,\epsilon_{\boldsymbol{W}})}[(\hat{y}_{\epsilon_{\boldsymbol{W}}}(\boldsymbol{x}) - y)^2] \tag{3.31}$$

$$= \mathbb{E}_{p(\boldsymbol{x},y,\epsilon_{\boldsymbol{W}})}[\hat{y}^2_{\epsilon_{\boldsymbol{W}}}(\boldsymbol{x}) - 2y\hat{y}_{\epsilon_{\boldsymbol{W}}}(\boldsymbol{x}) + y^2] \tag{3.32}$$

对于小的 η，最小化带权重噪声（方差为 $\eta\boldsymbol{I}$）的 J 等同于最小化附加上正则化项的 J：$\eta\mathbb{E}_{p(\boldsymbol{x},y)}[\|\nabla_{\boldsymbol{W}}\hat{y}(\boldsymbol{x})\|^2]$。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最小点 (**?**)。在简化的线性回归（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贡献。

3.5.1 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。当 y 是错误的，那对最大化 $\log p(y|\boldsymbol{x})$ 会是有害的。为了防止这一点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 $1-\epsilon$，任何其他可能的标签可能是正确的。这个假设很容易就能解析地与代价函数结合，而不用显式地采噪声样本。例如，**标签平滑** (label smoothing) 基于 k 个输出的softmax函数，把明确分类 0 和 1 替换成 $\frac{\epsilon}{k-1}$ 和 $1-\epsilon$，对模型进行正则化。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防止模型追求明确概率而不妨碍正确分类。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 (**?**)。

3.6 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 $P(\mathbf{y}|\mathbf{x})$ 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$h = f(\boldsymbol{x})$。学习表示的目标是使相同类中的样例有类似的表示。无监督学习可以为如何在表示空间聚集样例提供有用的线索。在输入空间紧密聚集的样例应该被映射到类似的表示。在新空间上的线性分类器在许多情况下可以达到较好的泛化 (**??**)。这种方法的长期存在的一个变种是应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 $P(\mathbf{y}|\mathbf{x})$ 共享参数，而不用将无监督和监督部分分离。可以对监督判据$-\log P(\boldsymbol{y}|\boldsymbol{x})$ 与无监督或生成的判据（如 $-\log P(\boldsymbol{x})$ 或 $-\log P(\boldsymbol{x},\boldsymbol{y})$）进行权衡。生成模型判据表达了对监督学习问题解的特殊形式的先验知识 (**?**)，即 $P(\mathbf{x})$ 的结构通过某种共享参数的方式连接到 $P(\mathbf{y}|\mathbf{x})$。通过控制在总判据中的生成判据，我们可以找到比纯生成或纯判别训练判据更好的权衡 (**??**)。

? 描述了一种学习回归内核机中内核函数的方法，其中建模的 $P(\mathbf{x})$ 时使用的未标记的样本大大提高了 $P(\mathbf{y}|\mathbf{x})$。

更多半监督学习的信息，请参阅**?**。

3.7 多任务学习

多任务学习 (?) 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。额外的训练样本以同样的方式将模型的参数推向泛化更好的方向，当模型的一部分在任务之间共享时，模型的这一部分被更多地约束为良好的值（假设共享是合理的），往往能更好的泛化。

图3.2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个部分和相关参数：

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3.2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3.2中的下层。

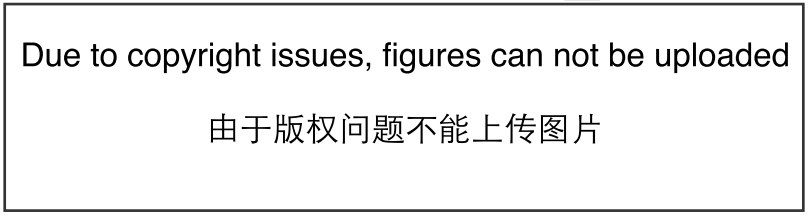


Figure 3.2: TODO

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式增加的比例），能改善泛化和泛化误差的范围 (?)。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时才会发生，也就是意味着某些参数能通过不同任务共享。

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，一些是跨两个或更多任务共享的。*

3.8 提前终止

当训练有足够的表示能力甚至会过度拟合的大模型时，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 |||c||| 是这些现象的一个例子，这种现象几乎一定会出现。

这意味着我们可以返回使验证集误差最低的参数设置来获得更好的模型（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内没有进一步改善时，算法就会终止。此过程在算法 |||c||| 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。

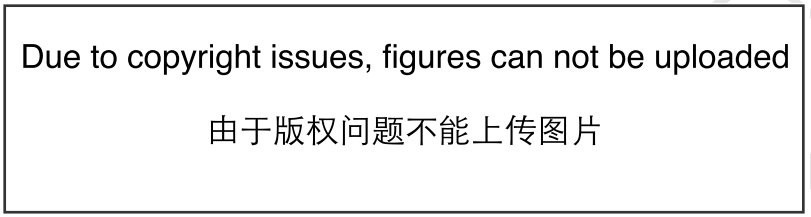


Figure 3.3: TODO

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3.3可以看到的，这个超参数在验证集上的具有 U 形性能曲线。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 |||c|||。在提前终止的情况下，我们通过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昂贵的猜测和检查过程，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” 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。理想情况下，这是可以并行在与主训练过程分离的机器，或独立的 CPU，或独离的 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，因为将它储存在较慢的较大的存储器上，(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。

提前终止是正则化的非常不显眼的形式，它几乎不需要对基本训练过程、目标函数或一组允许的参数值进行变化。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，防止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它的正则化策略相结合。即使修改目标函数以鼓励更好泛化的正则化策略，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，我们可以使用提前终止初步训练之后，进行额外的训练。

在第二轮额外的训练步骤中，所有的训练数据都被包括在内。有两个基本的策略可以用于第二轮训练过程。

一种策略（算法 |||c|||）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，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的遍数哪一个更好。由于训练集变大了，在第二轮训练时，每遍历一次数据集将会对应更多次的参数更新。

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。在这个阶段中，现在我们已经在训练多少步停止上不再有指导。相反，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停止时的目标值。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。这个过程在算法 |||c||| 中更正式地展示。

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）。

提前终止是如何充当正则化的： 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 形的来支持这种说法。提前终止正则化模型的真正机制是什么？**？** 和**？** 认为提前终止具有将优化过程的参数空间限制在初始参数值 $\boldsymbol{\theta}_0$ 相对小的邻域内的效果。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会限制从 $\boldsymbol{\theta}_0$ 到达的参数空间大小，如图3.4所示。在这个意义上，$\epsilon\tau$ 的行为就好像它是权重衰减系数的倒数。

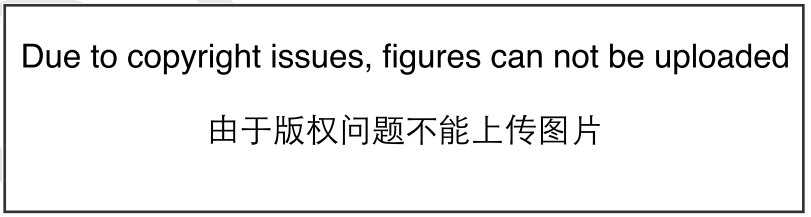


Figure 3.4: TODO

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，我们考察唯一的参数是线性权重（$\boldsymbol{\theta} = \boldsymbol{w}$）的简

单设定。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：

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*), \tag{3.33}$$

其中 $\boldsymbol{H}$ 是 J 关于 $\boldsymbol{w}$ 在 $\boldsymbol{w}^*$ 点的海森。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：

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*). \tag{3.34}$$

我们要研究训练时参数向量的轨迹。为简化起见，我们参数向量初始化为原点[3]，也就是 $\boldsymbol{w}^{(0)} = 0$。我们通过分析 $\hat{J}$ 的梯度下降研究 J 上近似的梯度下降行为：

$$\boldsymbol{w}^{(\tau)} = \boldsymbol{w}^{(\tau-1)} - \epsilon \nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}^{(\tau-1)}) \tag{3.35}$$

$$= \boldsymbol{w}^{(\tau-1)} - \epsilon \boldsymbol{H}(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*), \tag{3.36}$$

$$\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon \boldsymbol{H})(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*). \tag{3.37}$$

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。

$$\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top)(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*), \tag{3.38}$$

$$\boldsymbol{Q}^\top(\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^*) = (\boldsymbol{I} - \epsilon \boldsymbol{\Lambda})\boldsymbol{Q}^\top(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*). \tag{3.39}$$

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：

$$\boldsymbol{Q}^\top \boldsymbol{w}^{(\tau)} = [\boldsymbol{I} - (\boldsymbol{I} - \epsilon \boldsymbol{\Lambda})^\tau]\boldsymbol{Q}^\top \boldsymbol{w}^*. \tag{3.40}$$

现在, 式3.13中 $\boldsymbol{Q}^\top \tilde{\boldsymbol{w}}$ 的表达式能被重写为：

$$\boldsymbol{Q}^\top \tilde{\boldsymbol{w}} = (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^\top \boldsymbol{w}^*, \tag{3.41}$$

$$\boldsymbol{Q}^\top \tilde{\boldsymbol{w}} = [\boldsymbol{I} - (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\alpha]\boldsymbol{\Lambda}\boldsymbol{Q}^\top \boldsymbol{w}^*. \tag{3.42}$$

比较式3.40和式3.42，我们能够发现, 如果超参数 ϵ, α 和 τ 满足如下：

$$(\boldsymbol{I} - \epsilon \boldsymbol{\Lambda})^\tau = (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\alpha, \tag{3.43}$$

[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**??**节所讨论的)。然而，对于其他任何初始值 $\boldsymbol{w}_{(0)}$ 该论证都成立

那么 L^2正则化和权重衰减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

$$\tau \approx \frac{1}{\epsilon\alpha}, \tag{3.44}$$

$$\alpha \approx \frac{1}{\tau\epsilon}. \tag{3.45}$$

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。

对应显著曲率 (目标函数) 方向的参数值正则化小于小曲率方向。当然，在提前终止的情况下，这实际上意味着对应于显著曲率方向的参数比较小的曲率方向的参数学习得更早。

本节中的推导表明长度为 τ 的轨迹结束于最小的 L^2正则化 目标的最小点。当然，提前终止比仅仅的轨迹长度限制更丰富；相反，提前终止通常涉及监控验证集的错误，以便在空间特别好的点处停止轨迹。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，而权重衰减需要多个训练实验来测试其超参数的不同值。

3.9 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采取什么样的参数，但我们从领域和模型结构方面的知识知道，模型参数之间应该存在一些相关性。

我们经常想要表达的常见类型的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，这些任务是足够相似 (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，但也可以有其它选择。

这种方法由**?** 提出，正则化一个模型 (监督模式下训练的分类器) 的参数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)。这个架构的构造成使得许多分类模型中的参数能与对应的无监督模型的参数匹配。

参数范数惩罚是正则化参数彼此接近的一种方式，在更流行的方法是使用约束：强迫某些集合中的参数相等。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)。参数共享相对于正则化参数接近（通过范数惩罚）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，这可能导致内存占用的显著减少。

3.9.1 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)。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，仍保持猫的相片，。CNN通过在图像多个位置共享参数考虑这个特性。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。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CNN模型不同参数的数量，并显著提高了网络的大小而不需要增加相应的训练数据。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。

卷积神经网络将会在第**??**章更详细地讨论。

3.10 稀疏表示

权重衰减施加直接作用于模型参数的惩罚。另一种策略是将惩罚放在神经网络的激活单元，鼓励对应的激活是稀疏。这间接的对模型参数施加了复杂惩罚。

我们已经讨论过（在3.1.2）L^1 惩罚如何诱导稀疏的参数，意味着许多参数为零（或接近于零）。表示的稀疏，在另一方面描述了一种其中许多的元素是零（或

接近零）的表示。可以线性回归情况下简单说明这种区别：

$$\underbrace{\begin{bmatrix}18\\5\\15\\-9\\-3\end{bmatrix}}_{\boldsymbol{y}\in\mathbb{R}^m} = \underbrace{\begin{bmatrix}4&0&0&-2&0&0\\0&0&-1&0&3&0\\0&5&0&0&0&0\\1&0&0&-1&0&-4\\1&0&0&0&-5&0\end{bmatrix}}_{\boldsymbol{A}\in\mathbb{R}^{m\times n}} \underbrace{\begin{bmatrix}2\\3\\-2\\-5\\1\\4\end{bmatrix}}_{\boldsymbol{x}\in\mathbb{R}^n} \tag{3.46}$$

$$\underbrace{\begin{bmatrix}-14\\1\\19\\2\\23\end{bmatrix}}_{\boldsymbol{y}\in\mathbb{R}^m} = \underbrace{\begin{bmatrix}3&-1&2&-5&4&1\\4&2&-3&-1&1&3\\-1&5&4&2&-3&-2\\3&1&2&-3&0&-3\\-5&4&-2&2&-5&-1\end{bmatrix}}_{\boldsymbol{B}\in\mathbb{R}^{m\times n}} \underbrace{\begin{bmatrix}0\\2\\0\\0\\-3\\0\end{bmatrix}}_{\boldsymbol{h}\in\mathbb{R}^n} \tag{3.47}$$

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，但是用一个稀疏向量表示。

表示的正则化可以通过在参数正则化中使用的同种类型的机制来实现的。

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。记这个惩罚为 $\Omega(\boldsymbol{h})$。和以前一样，我们将正则化后的损失函数记为 $\tilde{J}$:

$$\tilde{J}(\boldsymbol{\theta};\boldsymbol{X},\boldsymbol{y}) = J(\boldsymbol{\theta};\boldsymbol{X},\boldsymbol{y}) + \alpha\Omega(\boldsymbol{h}) \tag{3.48}$$

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

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 惩罚是导致稀疏表示的选择之一。其他包括从表示上Student t 先验导出的惩罚 (**??**) 和KL 散度惩罚 (**?**) 有利于表示元素约束于单位区间上。<BAD>**?** 和**?** 都提供了基于平均几个实例激活的正则化策略的例子，$\sum_i^m \boldsymbol{h}^{(i)}$，使其接近某些目标值，如每项都是 .01 的向量。

其他的方法使用激活值的硬性约束获得表示稀疏。例如，**正交匹配追踪** (orthogonal matching pursuit)(**?**) 通过解决约束优化问题将输入值 $\boldsymbol{x}$ 编码成表示$\boldsymbol{h}$

$$\underset{\boldsymbol{h},\|\boldsymbol{h}\|_0<k}{\operatorname{argmin}} \|\boldsymbol{x}-\boldsymbol{W}\boldsymbol{h}\|^2, \tag{3.49}$$

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，这个问题可以高效地解决。这种方法通常被称为OMP-k，通过 k 指定允许非零特征的数量。**?** 证明OMP-1可以成为深度架构中非常有效的特征提取器。

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

3.11 Bagging和其他集成的方法

Bagging(bootstrap aggregating) 是通过结合几个模型降低泛化误差的技术 (**?**)。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。这在机器学习中普通策略的一个例子，被称为模型平均**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错误。

考虑 k 个回归模型的例子。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

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i \left(\epsilon_i^2 + \sum_{j \neq i} \epsilon_i \epsilon_j\right)\right], \tag{3.50}$$

$$= \frac{1}{k}v + \frac{k-1}{k}c. \tag{3.51}$$

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的大小线性地减小。换言之，集成平均上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。每个数据集与原始数据集具有相同数量的样例，但从原始数据集中有替换采样构成。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。每个数据集包含样本的差异导致训练模型之间的差异。图3.5是一个例子。

神经网络的解能达到足够多的变化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、minibatch的随机选择、超参数的差异或不同输出的非确定性的实现往往足以引起集成中不同成员的误差部分独立。

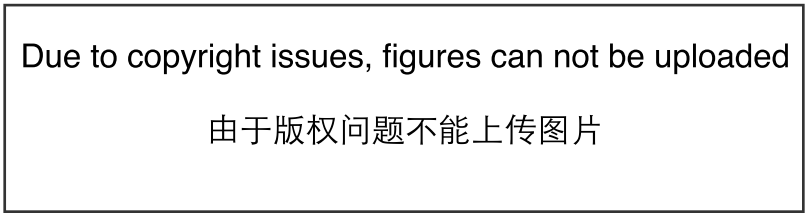


Figure 3.5: TODO

模型平均是减少泛化误差一个非常强大可靠的方法。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可从模型平均中大幅获益(以增加计算和存储的代价)。

机器学习比赛通常使用超过几十种模型平均的方法取胜。最近一个突出的例子是Netflix Grand Prize(**?**)。

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**(Boosting) 的技术 (**??**) 构建比单一模型容量更高集成模型。向集成逐步添加神经网络,Boosting已经应用于构建神经网络的集成(**?**)。Boosting也可以将单个神经网络解释为单个集成，即通过逐渐增加神经网络的隐藏单元。

3.12 dropout

dropout(dropout)(**?**) 提供了正则化一大类模型的方法，计算方便但功能强大的。第一个近似下，dropout可以被认为是集成非常多的大神经网络的实用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。当每个模型是一个大型神经网络时,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。通常集成五至十个神经网络，如**?** 用六个赢得 ILSVRC，超过这个数量就会迅速变得难处理。dropout提供了一种廉价的Bagging集成近似,能够训练和评估指数级的神经网络。

具体而言，dropout训练的集成包括所有从基本的基础网络除去非输出单元形成子网络，如在图3.6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，我们可以将一些单元的输出乘零以有效地删除一个单元。这个过程需要对模型一些修改，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。为了简单起见，我们在这里提出乘零的简单dropout算法，但是它被简单地修改后，可以与从网络中移除单元的其他操作一起工作。

回想一下使用Bagging学习，我们定义 k 个不同的模型，从训练集有替换采样构造 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

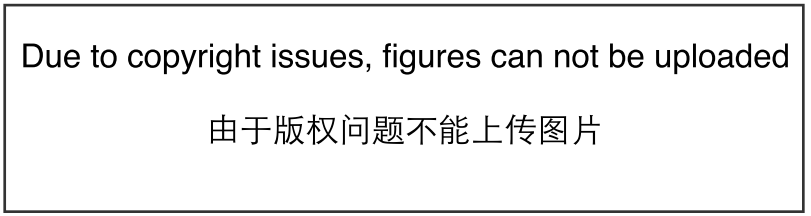


Figure 3.6: TODO

级数量的神经网络上近似这个过程。具体来说，训练中使用dropout，我们使用基于minibatch的学习算法和小的步长，如梯度下降等。我们每次在minibatch加载一个样本，我们随机抽样应用与网络中所有输入和隐藏单元的不同二值掩码。对于每个单元，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）是训练开始前固定一个超参数。它不是模型当前参数值或输入样本的函数。通常一个输入单元包括的概率为 0.8，一个隐藏单元包括的概率为 0.5。然后，我们运行之前一样的前向传播、反向传播以及学习更新。图3.7说明了在dropout下的前向传播。

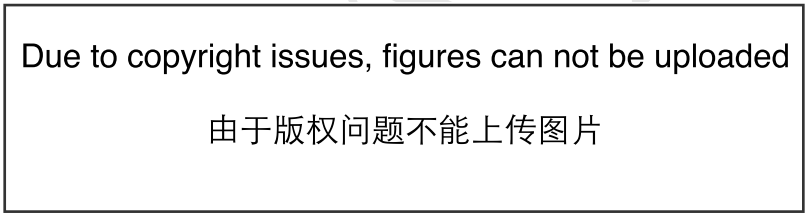


Figure 3.7: TODO

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训练在于最小化 $\mathbb{E}_{\boldsymbol{\mu}} J(\boldsymbol{\theta}, \boldsymbol{\mu})$。期望包含指数多的项，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，所有模型是独立的。在dropout的情况下，模型是共享参数的，其中每个模型继承的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代表指数数量的模型变得可能。在Bagging的情况下，每一个模型被训练在其相应训练集上收敛。在dropout的情况下，通常大部分模型都没有显式地被训练，通常该模型很大，以致到宇宙毁灭都不能采样所有可能的子网络。取而代之的是，可能的子网络的一小部分训练单个步骤，参数共享导致剩余的子网络能有好的参数设定。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替换采样的原始训练集的一个子集。

Bagging集成必须从所有成员的积累投票做一个预测。在这种背景下，我们将这个过程称为推断。目前为止，Bagging和dropout的描述中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 $p^{(i)}(y|\boldsymbol{x})$。集成的预测由这些分布的算术平均值给出，

$$\frac{1}{k}\sum_{i=1}^{k} p^{(i)}(y|\boldsymbol{x}) \tag{3.52}$$

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

$$\sum_{u} p(\boldsymbol{\mu})p(y|\boldsymbol{x},\boldsymbol{\mu}) \tag{3.53}$$

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

因为这个求和包含指数多的项，除非在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任何可行的简化。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，有一个更好的方法能得到一个不错的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，我们改用集成成员的预测分布的几何平均而不是算术平均。**?** 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

$$\tilde{p}_{\text{ensemble}}(y|\boldsymbol{x}) = \sqrt[2^d]{\prod_{\boldsymbol{\mu}} p(y|\boldsymbol{x},\boldsymbol{\mu})}, \tag{3.54}$$

其中 d 是可被丢弃的单元数。在这里，我们使用一个均匀分布的 $\boldsymbol{\mu}$ 以简化介绍，但非均匀分布也是可能的。为了作出预测，我们必须重新标准化集成：

$$p_{\text{ensemble}}(y|\boldsymbol{x}) = \frac{\tilde{p}_{\text{ensemble}}(y|\boldsymbol{x})}{\sum_{y'}\tilde{p}_{\text{ensemble}}(y'|\boldsymbol{x})}. \tag{3.55}$$

涉及dropout的一个重要观点 (**?**) 是，我们可以通过评估模型中 $p(y|\boldsymbol{x})$ 近似 $p_{\text{ensemble}}(y|\boldsymbol{x})$：该模型具有所有单元，但单元 i 输出的权重乘以包括单元 i 的概

率。这个修改的动机是捕获从该单元输出的正确期望值。我们把这种方法称为**权重比例推理规则**(weight scaling inference rule)。目前还没有在深度非线性网络上对这种近似推理规则的准确性作任何理论上的说法，但经验上表现得很好。

因为我们通常使用一个 $\frac{1}{2}$ 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。实现相同的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是与在训练时该单元的期望总输入是大致相同（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，权重比例推理规则是精确的。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：

$$P(\mathrm{y} = y \mid \mathbf{v}) = \mathrm{softmax}\big(\boldsymbol{W}^\top \mathbf{v} + \boldsymbol{b}\big)_y. \tag{3.56}$$

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：

$$P(\mathrm{y} = y \mid \mathbf{v}; \boldsymbol{d}) = \mathrm{softmax}\big(\boldsymbol{W}^\top (\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}\big)_y. \tag{3.57}$$

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：

$$P_{\mathrm{ensemble}}(\mathrm{y} = y \mid \mathbf{v}) = \frac{\tilde{P}_{\mathrm{ensemble}}(\mathrm{y} = y \mid \mathbf{v})}{\sum_{y'} \tilde{P}_{\mathrm{ensemble}}(\mathrm{y} = y' \mid \mathbf{v})}, \tag{3.58}$$

其中

$$\tilde{P}_{\mathrm{ensemble}}(\mathrm{y} = y \mid \mathbf{v}) = \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} P(\mathrm{y} = y \mid \mathbf{v}; \boldsymbol{d})}. \tag{3.59}$$

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

$$\tilde{P}_{\mathrm{ensemble}}(\mathrm{y} = y \mid \mathbf{v}) = \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} P(\mathrm{y} = y \mid \mathbf{v}; \boldsymbol{d})} \tag{3.60}$$

$$= \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} \mathrm{softmax}(\boldsymbol{W}^\top (\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b})_y} \tag{3.61}$$

$$= \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} \frac{\exp(\boldsymbol{W}_{y,:}^\top (\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}_y)}{\sum_{y'} \exp(\boldsymbol{W}_{y',;}^\top (\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}_{y'})}} \tag{3.62}$$

$$= \frac{\sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} \exp(\boldsymbol{W}_{y,:}^\top (\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}_y)}}{\sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} \sum_{y'} \exp(\boldsymbol{W}_{y',:}^\top (\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}_{y'})}} \tag{3.63}$$

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：

$$\tilde{P}_{\text{ensemble}}(\mathrm{y} = y \mid \mathbf{v}) \propto \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} \exp(\boldsymbol{W}_{y,:}^{\top}(\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}_y)} \tag{3.64}$$

$$= \exp\left(\frac{1}{2^n} \sum_{\boldsymbol{d} \in \{0,1\}^n} \boldsymbol{W}_{y,:}^{\top}(\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}_y\right) \tag{3.65}$$

$$= \exp\left(\frac{1}{2} \boldsymbol{W}_{y,:}^{\top}\mathbf{v} + \boldsymbol{b}_y\right) \tag{3.66}$$

将其代入式3.58，我们得到了一个权重为 $\frac{1}{2}\boldsymbol{W}$ 的softmax 函数分类器。

权重比例推理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，权重比例推理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。**?** 实验发现，集成预测权重比例推理规则可以比蒙特卡罗近似工作得更好（在分类精度方面）。即使允许蒙特卡罗近似采样多达 1000 子网络时也比不过。**?** 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。

? 显示，dropout比其他标准的计算开销小的正则化 项，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是与不使用dropout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，如受限玻耳兹曼机(**?**)，以及循环神经网络(**??**)。许多其他差不多强大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 (**?**)，贝叶斯神经网络 (**?**) 比dropout表现更好 (**?**)。

当有其他未分类的数据可用时，无监督特征学习比dropout更有优势。

? 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。其他线性模型有类似的结果。而对于深度模型，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。? 导出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)，由于梯度计算中的随机性减小导致更快的收敛时间。这种方法也可以在测试时应用，比权重比例推理规则更合理地（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地改善或尚未应用。

正如随机性对实现dropout的正则化效果不是必要的，这也不是充分的。为了证明这一点，? 使用一种称为**dropout boosting**(dropout boosting) 的方法设计了一个对照实验，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。在相同意义上，传统的dropout类似于Bagging，这种方式类似于Boosting。如预期一样，比较单一模型训练整个网络的情况，dropout boosting几乎没有正则化效果。这表明，dropoutBagging的解释超过dropout作为稳健性的噪音的解释。当随机抽样的集成成员相互独立地训练好后，Bagging集成的正则化效果才能达到。

dropout启发其它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 (?)。随机池化是构造卷积神经网络集成的一种随机池化的形式 (见??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。

关于dropout的一个重要见解是，通过随机行为训练网络并平均多个随机决定进行预测，通过参数共享实现了Bagging的一种形式。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，我们必须选择能让神经网络能够学习对抗的修改类型。理想情况下，我们也应该使用可以快速近似推断的模型类。我们可以认为由向量 $\boldsymbol{\mu}$ 作为任何形式修改的参数，对于 $\boldsymbol{\mu}$ 所有可能的值训练 $p(y \mid \boldsymbol{x}, \boldsymbol{\mu})$ 的集成。这里不要求 $\boldsymbol{\mu}$ 具有有限值。例如，$\boldsymbol{\mu}$ 可以是实值。? 表明，权重乘以 $\boldsymbol{\mu} \sim \mathcal{N}(1, \boldsymbol{I})$ 比基于二值掩码 dropout表现更好。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理规则。

到目前为止，我们已经介绍dropout纯粹作为一种高效近似Bagging的手段。然而，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。这意味着无论其它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。? 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，更要在许多情况下良好的特征。**?** 将dropout与大集成的训练相比并得出结论：相比独立模型的集成获得的泛化误差，dropout会提供的额外改进。

dropout的强大大部分是由于被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的、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，要么是脸部的另一特征，如嘴。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 的修正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，batch normalization，在训练时向隐藏单元引入加性和乘性噪声的方法重参数化模型。batch normalization的主要目的是改善优化，但噪音具有正则化的效果，有时使dropout变得没有必要。batch normalization将会在**??**节更详细的讨论。

3.13 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时已经达到人类表现。因此，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。为了探测网络对底层任务的理解层次，我们可以搜索这个模型错误分类的例子。**?** 发现，精度达到人类水平的神经网络在通过优化过程故意构造的点上的误差率接近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，人类观察者不知道原始样本和**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3.8的例子。

对抗样本有很多的影响，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上错误率——在对抗扰动的训练集样本上训练 (**??**)。

? 表明，这些对抗样本的主要原因之一是过度线性。神经网络主要基于线性块构建的。因此在一些实验中，他们实现的整体函数被证明是高度线性。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 $\boldsymbol{w}$ 线性函数可以改

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 3.8: TODO

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。这可以被看作是明确地向监督神经网络引入局部恒定的先验的方法。

对抗训练有助于说明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，由于他们被限制为线性而无法抵抗对抗样本。神经网络是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。

对抗样本也提供了实现半监督学习的一种手段。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，那么 $\hat{y}$ 提供真正标签的可能性很大。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称为**虚拟对抗样本**(virtual adversarial example)(**?**)。分类器可以被训练成对 $\boldsymbol{x}$ 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是鲁棒的函数。驱动这种方法的假设是，不同的类通常位于分离的流形上，并且小的扰动不能从一类的流形跳到另一个类的流形。

3.14 切面距离、正切传播和流形切分类

许多机器学习的目标旨在假设数据位于低维流形附近来克服维数灾难，如**??**描述。

一个利用流形假设的早期尝试是**切面距离**(tangent distance) 算法 (**??**)。它是一种非参数最近邻算法，其中度量使用的不是通用的欧几里德距离，而是是从邻近流形关于聚集概率的知识导出的。这假设我们正在尝试的分类样本，且同一流形上的样本共享相同的类别。由于分类器应该对变化的局部因素（对应于流形上的移动）不变，将点 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$ 分别所在的流形 M_1 和 M_2 的距离作为点 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$ 的最近邻距离是合理的。尽管这可能是在计算上是困难（它需要解决一个找到 M_1 和 M_2 最近对点的优化问题），一种廉价的替代是局部合理的，即使用 $\boldsymbol{x}_i$ 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的距离。可以通过求解

一个低维线性系统（流形的维数）实现。当然，这种算法需要制定一个的切向量。

受相关启发，**正切传播**(tangent prop) 算法 (**?**)（图3.9）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。局部不变性是通过要求 $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ 与已知流形的切向 $\boldsymbol{v}^{(i)}$ 正交实现的，或者等价地通过正则化惩罚 Ω 使 f 在 $\boldsymbol{x}$ 的 $\boldsymbol{v}^{(i)}$ 方向的导数是小的：

$$\Omega(f) = \sum_i \left((\nabla_{\boldsymbol{x}} f(\boldsymbol{x})^\top \boldsymbol{v}^{(i)}) \right)^2. \tag{3.67}$$

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，我们将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, 切向量推导先验，通常是从变换，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 (**?**)，还在强化学习 (**?**) 中有所应用。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 3.9: TODO

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不改变网络输出的转换, 编码其先验知识。不同的是在数据集增强的情况下，网络是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在对应于指定转换的方向抵抗扰动。虽然这种分析方法是聪明优雅的，它有两个主要的缺点。首先，模型的正则化只能抵抗无穷小扰动。显式的数据集增强能对抗较大扰动。二是无限小的做法对基于修正线性单元的模型是困难的。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。他们不能像sigmoid或tanh单元一样通过大的权重在高值处饱和以收缩导数。数据集增强在修正线性单元上工作得很好，因为修正单元的不同子集针对每一个原始输入不同的转换版本激活。

正切传播也涉及到双反向传播(**?**) 和对抗训练(**??**)。双反向传播正则化雅各比要小，而对抗训练找到原输入附近的点，训练模型以在这些点产生与原来输入相同的输出。正切传播和手工指定转换的数据集增强都要求模型对输入变化的某些特定的方向是不变的。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）都应当是不变的。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。

流形切面分类器 (**?**) 无需知道切线向量的先验。正如我们将在第7章看到，自动编码器可以估算流形的切向量。流形切面分类器使用这种技术来避免用户指定切向量。如图**??**所示，这些估计的切向量超出了图像（如转化、旋转和缩放）几何的经典不变，还必须掌握因为特定对象（如移动身体的部分）的因素。因此根据流形切面分类提出的算法是简单：（1）使用自动编码器通过无监督学习来学习流形的结构，以及（2）如正切传播（式3.67）一样使用这些切面正则化神经网络分类器。

在本章中，我们已经描述了大多数用于正则化神经网络的通用策略。正则化是机器学习的中心主题，因此将定期在其余各章中重新回顾。机器学习的另一个中心主题是优化，将在下章描述。

Chapter 4

序列建模：循环和递归网络

循环神经网络(recurrent neural network) 或RNN(**?**) 是一类用于处理序列数据的神经网络。就像卷积网络是专门用于处理网格化数据 $\mathbf{X}$（如一个图像）的神经网络，循环神经网络是专门用于处理序列 $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}$ 的神经网络。<BAD> 正如卷积网络可以很容易地扩展到具有很大宽度和高度的图像，以及处理大小可变的图像，循环网络可以扩展到更长的序列（比不基于序列的特化网络长得多）。大多数循环网络也能处理可变长度的序列。

从多层网络出发到循环网络，我们需要利用上世纪 80 年代机器学习和统计模型早期思想的优点：在模型的不同部分共享参数。参数共享使得模型能够扩展到不同形式的样本（如不同的长度）并进行泛化。如果我们在每个时间点都有一个单独的参数，我们不但不能泛化到训练时没有见过序列长度，也不能在时间上共享不同序列长度和不同位置的统计强度。当信息的特定部分能够在该序列内多个位置出现时，这样的共享尤为重要。例如，考虑这两句话：“I went to Nepal in 2009” 和 “In 2009, I went to Nepal.” 如果我们让一个机器学习模型读取这两个句子，并提取叙述者去Nepal的年份，无论 “2009 年” 出现在句子的第六个单词或第二个单词，我们都希望它能认出 “2009 年” 作为相关资料片段。假设我们训练一个处理固定长度句子的前馈网络。传统的全连接前馈网络会给每个输入特征一个单独的参数，所以需要分别学习句子每个位置的所有语言规则。相比之下，循环神经网络在几个时间步内共享相同的权重。

一个相关的想法是跨越 1 维时间序列的卷积。这种卷积方法是时延神经网络的基础 (**???**)。卷积操作允许网络跨越时间共享参数，但是浅层的。卷积的输出是一个序列，其中输出中的每一项是相邻几项的函数。参数共享的概念体现在每个时间步中使用的相同卷积核。循环神经网络以不同的方式共享参数。输出的每一项是输出前一项的函数。输出的每一项对先前的输出应用相同的更新规则而产生。这种循环导致参数在一个很深的计算图中共享。

为简单起见，我们说的RNN是指在序列上的操作，并且该序列在时刻 t（从

1 到 τ）包含向量 $\boldsymbol{x}^{(t)}$。在实践中，循环网络通常在序列的minibatch上操作，并且minibatch的每项具有不同序列长度 τ。我们省略了minibatch索引来简化记号。此外，时间步索引不必是字面上现实世界中流逝的时间，也可以是序列中的位置。<BAD>RNN也可以应用于跨越两个维度的空间数据（如图像），并且当应用于涉及的时间数据时，该网络可具有在时间上向后的连接，只要在将整个序列提供给网络之前，让网络观察整个序列。

本章将计算图的想法扩展到包括周期。这些周期代表变量自身的值在未来某一时间步对自身值的影响。这样的计算图允许我们定义循环神经网络。然后，我们对构建、训练和使用循环神经网络的许多不同方式进行描述。

除了本章中关于循环神经网络的介绍，我们建议读者参考**?** 的著作获取更多详细信息。

4.1 展开计算图

计算图是一种形式化一组计算结构的方式，如那些涉及将输入和参数映射到输出和损失的计算。综合的介绍请参考**??**。在本节中，我们对**展开**(unfolding) 递归或循环计算得到的重复结构进行解释，这通常对应于一个事件链。**展开**(unfolding) 这个图导致深度网络结构中的参数共享。

例如，考虑动态系统的经典形式：

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta}), \tag{4.1}$$

其中 $\boldsymbol{s}^{(t)}$ 称为系统的状态。

$\boldsymbol{s}$ 在时刻 t 的定义需要参考时刻 $t-1$ 时同样的定义，因此式(4.1)是循环的。

对一个有限时间步τ，$\tau-1$ 次应用这个定义可以展开这个图。例如，如果我们对式(4.1)关于 $\tau = 3$ 展开，可以得到：

$$\begin{aligned} \boldsymbol{s}^{(3)} &= f(\boldsymbol{s}^{(2)}; \boldsymbol{\theta}) && (4.2)\\ &= f(f(\boldsymbol{s}^{(1)}; \boldsymbol{\theta}); \boldsymbol{\theta}). && (4.3) \end{aligned}$$

以这种方式重复应用定义，展开等式，就能得到不涉及循环的表达。现在可以通过一个传统的有向无环计算图来表示这样的表达。

式(4.1)和式(4.3)的展开计算图如图4.1所示。

作为另一个例子，让我们考虑由外部信号 $\boldsymbol{x}^{(t)}$ 驱动的动态系统，

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}), \tag{4.4}$$

我们可以看到，当前状态包含了整个过去序列的信息。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 4.1: TODO

循环神经网络可以通过许多不同的方式建立。就像几乎所有函数都可以被认为是前馈网络，基本上任何涉及循环的函数可以被认为是一个循环神经网络。

很多循环神经网络使用式(4.5)或类似的公式定义隐藏单元的值。为了表明状态是网络的隐藏单元，我们使用变量 h 代表状态来重写式(4.4)：

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}), \tag{4.5}$$

如图4.2所示，典型RNN会增加额外的架构，如读取状态信息 $\boldsymbol{h}$ 进行预测的输出层。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 4.2: TODO

当循环网络被训练为根据过去预测未来，网络通常要学会使用 $\boldsymbol{h}^{(t)}$ 作为过去序列（直到 t）与任务相关方面的有损摘要。此摘要一般而言一定是有损的，因为其映射任意长度的序列 $(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$ 到一固定长度的向量 $\boldsymbol{h}^{(t)}$。根据不同的训练判据，摘要可能选择性地精确保留过去序列的某些方面。例如，如果在统计语言建模中使用的RNN，通常给定前一个词预测下一个词，可能没有必要存储 t 前所有输入序列中的信息；而仅仅存储足够预测句子的其余部分信息。最苛刻的情况是我们要求 $\boldsymbol{h}^{(t)}$ 足够丰富，并能大致恢复输入序列，如自动编码器框架（第7章）。

<BAD> 式(4.5)可以用两种不同的方式绘制。一种方法是为可能在模型的物理实现中存在的部分赋予一个节点，如生物神经网络。在这个观点下，网络定义了实时操作的回路，如图4.2的左侧，其当前状态可以影响其未来的状态。在本章中，我们使用回路图的黑色方块表明在时刻 t 的状态到时刻 $t+1$ 的状态单个时

刻延迟中的相互作用。另一个绘制RNN的方法是展开的计算图，其中每一个组件是由许多不同的变量表示，每个时间步一个变量，表示在该时间点组件的状态。每个时刻的每个变量绘制为计算图的一个单独的节点，如图4.2的右侧。我们所说的展开是左图中的回路映射为右图中包含重复组件的计算图的操作。目前，展开图的大小取决于序列长度。

我们可以用一个函数 $g^{(t)}$ 代表经 t 步展开后的循环：

$$\boldsymbol{h}^{(t)} = g^{(t)}(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}) \tag{4.6}$$

$$= f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}) \tag{4.7}$$

函数 $g^{(t)}$ 将全部的过去序列 $(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$ 作为输入来生成当前状态，但是展开的循环架构允许我们将 $g^{(t)}$ 分解为函数 f 重复的应用。因此，展开过程引入两个主要优点：

1. 无论序列的长度，学习好的模型始终具有相同的输入大小，因为它指定的是从一种状态到另一种状态的转移，而不是在可变长度的历史状态上操作。

2. 我们可以在每个时间步使用具有相同参数的*相同*转移函数 f。

这两个因素使得学习在所有时间步和所有序列长度上操作的单一模型 f 是可能的，而不需要在所有可能时间步学习独立的模型 $g^{(t)}$。学习单一的共享模型允许泛化到没有见过的序列长度（没有出现在训练集中），并且估计模型需要的训练样本远远少于不带参数共享的模型。

无论是循环图和展开图都有其用途。循环图简洁。展开图能够明确描述其中的计算流程。展开图还通过显式的信息流动路径帮助说明信息在时间上向前（计算输出和损失）和向后（计算梯度）的思想。

4.2 循环神经网络

基于4.1节中的图展开和参数共享的思想，我们可以设计各种循环神经网络。循环

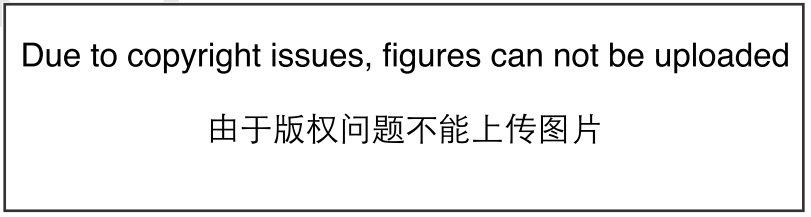


Figure 4.3: TODO

神经网络中一些重要的设计模式包括以下几种：

1. 每个时间步都有输出，并且隐藏单元之间有循环连接的循环网络，如图4.3所示。

2. 每个时间步都产生一个输出，只有当前时刻的输出到下个时刻的隐藏单元之间有循环连接的循环网络，如图4.4所示。

3. 隐藏单元之间存在循环连接，但读取整个序列后产生单个输出的循环网络，如图4.5所示。

图4.3是一个非常具有代表性的例子，我们将会在本章大部分涉及这个例子。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 4.4: TODO

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 4.5: TODO

任何图灵可计算的函数都可以通过这样一个有限维的循环网络计算，在这个意义上图4.3和式(4.8)的循环神经网络是万能的。RNN经过若干时间步后读取输出，这与由图灵机所用时间步是渐近线性的，与输入长度也是渐近线性的 (**????**)。由图灵机计算的函数是离散的，所以这些结果都是函数的具体实现，而不是近似。RNN作为图灵机使用时，需要一个二进制序列作为输入，其输出必须离散化以提供二进制输出。<BAD> 有可能利用单个有限大小的特定RNN来计算在此设置下的所有函数（**?** 用了 886 个单元）。图灵机的 “输入” 是要计算函数的详细说明 (specification)，所以模拟此图灵机的相同网络足以应付所有问题。用于证明的理论RNN可以通过激活和权重（由无限精度的有理数表示）来模拟无限堆栈。

现在我们研究图4.3中RNN的前向传播公式。这个图没有指定隐藏单元的激活函数。我们假设使用双曲正切激活函数。此外，图中没有明确指定何种形式的输

出和损失函数。我们假定输出是离散的，如用于预测词或字符的RNN。一种代表离散变量的自然方式是把输出 $\boldsymbol{o}$ 作为每个离散变量可能值的非标准化对数概率。然后，我们可以应用softmax 函数后续处理后，获得标准化后概率的输出向量 $\hat{\boldsymbol{y}}$。RNN从特定的初始状态 $\boldsymbol{h}^{(0)}$ 开始前向传播。从 $t = 1$ 到 $t = \tau$ 的每个时间步，我们应用以下更新方程：

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \tag{4.8}$$

$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)}), \tag{4.9}$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \tag{4.10}$$

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}(\boldsymbol{o}^{(t)}), \tag{4.11}$$

其中的参数的偏置向量 $\boldsymbol{b}$ 和 $\boldsymbol{c}$ 连同权重矩阵 $\boldsymbol{U}$、$\boldsymbol{V}$ 和 $\boldsymbol{W}$，分别对应于输入到隐藏、隐藏至输出和隐藏到隐藏的连接。这是一个输入序列映射到相同长度的输出序列的循环网络例子。与 $\boldsymbol{x}$ 序列配对的 $\boldsymbol{y}$ 的总损失就是所有时间步的损失之和。例如，$L^{(t)}$ 为给定的 $\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(t)}$ 后 $\boldsymbol{y}^{(t)}$ 的负对数似然，则

$$L\big(\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \dots, \boldsymbol{y}^{(\tau)}\}\big) \tag{4.12}$$

$$= \sum_t L^{(t)} \tag{4.13}$$

$$= -\sum_t \log p_{\text{model}}\big(y^{(t)} | \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(t)}\}\big), \tag{4.14}$$

其中 $p_{\text{model}}\big(y^{(t)} | \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(t)}\}\big)$ 需要读取模型输出向量 $\hat{\boldsymbol{y}}^{(t)}$ 中对应于 $y^{(t)}$ 的项。关于各个参数计算这个损失函数的梯度是昂贵的操作。梯度计算涉及执行一次前向传播（如在图4.3展开图中从左到右的传播），接着是由右到左的反向传播。运行时间是 $\mathcal{O}(\tau)$，并且不能通过并行化来降低，因为前向传播图顺序是固有的；每个时间步只能一前一后地计算。在前向传播中各个状态必须保存，直到它们被反向传播中再次使用，因此内存代价也是 $\mathcal{O}(\tau)$。应用于展开图且代价为 $\mathcal{O}(\tau)$ 的反向传播算法称为**通过时间反向传播**(back-propagation through time) 或BPTT，并会在4.2.2节中进一步讨论。因此隐藏单元之间存在循环的网络非常强大但训练代价也很大。我们是否有其他选择呢？

4.2.1 Teacher Forcing和输出循环网络

仅在一个时间步的输出和下一个时间步的隐藏单元间存在循环连接的网络（示于图4.4）确实没有那么强大（因为缺乏隐藏到隐藏的循环连接）。例如，它不能模拟通用图灵机。因为这个网络缺少隐藏到隐藏的循环，它要求输出单元捕捉将用于预测未来的所有关于过去的信息。因为输出单元明确地训练成匹配训练集的目标，它们不太能捕获关于过去输入历史的必要信息，除非用户知道如何描述系统

的全部状态，并将它作为训练目标的一部分。消除隐藏到隐藏循环的优点在于，任何基于比较时刻 t 的预测和时刻 t 的训练目标的损失函数，所有时间步都解耦了。因此训练可以并行化，在各时刻 t 分别计算梯度。因为训练集提供输出的理想值，所以没有必要先计算前一时刻的输出。

从输出导致循环连接的模型可用Teacher Forcing进行训练。Teacher Forcing是使用最大似然判据的程序，并且训练模型时在时刻 $t+1$ 接收真实值 $y^{(t)}$ 作为输入。我们可以通过检查两个时间步的序列得知这一点。条件最大似然判据是

$$\log p(\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)} \mid \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}) \tag{4.15}$$

$$= \log p(\boldsymbol{y}^{(2)} \mid \boldsymbol{y}^{(1)}, \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}) + \log p(\boldsymbol{y}^{(1)} \mid \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}). \tag{4.16}$$

在这个例子中，我们可以看到在时刻 $t=2$ 时，*同时*给定迄今为止的 $\boldsymbol{x}$ 序列和来自训练集的前一 $\boldsymbol{y}$ 值，模型被训练为最大化 $\boldsymbol{y}^{(2)}$ 的条件概率。因此最大似然在训练时指定正确反馈，而不是将自己的输出反馈到模型。如图4.6所示。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 4.6: TODO

我们使用Teacher Forcing的最初动机是为了避免缺乏隐藏到隐藏连接的通过时间反向传播。Teacher Forcing仍然可以应用到有隐藏到隐藏连接的模型，只要它们在一个时间步的输出与下一时间步计算的值存在连接。然而，只要隐藏单元成为较早时间步的函数，BPTT算法是必要的。因此训练某些模型时要同时使用Teacher Forcing和BPTT。

如果之后网络在**开环** (open-loop) 模式下使用，即网络输出（或输出分布的样本）反馈作为输入，那么Teacher Forcing的缺点就会出现。在这种情况下，训练期间该网络看到的输入与测试时看到的会有很大的不同。<BAD> 减轻此问题的一种方法是同时使用Teacher Forcing和自由运行的输入进行训练，比如在展开循环的输出到输入路径预测几个步骤的正确目标值。通过这种方式，网络可以学会考虑在训练时没有接触到的输入条件（如自由运行模式下，自身生成自身），以及将状态映射回使网络几步之后生成正确输出的状态。另外一种方式 (**?**) 是通过随意选择生成值或真实的数据值作为输入以减小训练时和测试时看到的输入之间的差别。这种方法利用了课程学习策略，逐步使用更多生成值作为输入。

4.2.2 计算循环神经网络的梯度

计算循环神经网络的梯度是容易的。可以简单地将??节中的推广反向传播算法应用于展开的计算图。不需要特殊化的算法。由反向传播计算得到的梯度，并结合任何通用的基于梯度的技术就可以训练RNN。

为了获得BPTT算法行为的一些直观理解，我们举例说明如何通过BPTT计算上述RNN公式（式(4.8)和式(4.12)）的梯度。计算图的节点包括参数 $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}, \boldsymbol{b}$ 和 $\boldsymbol{c}$，以及以 t 索引的顺序节点 $\boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t)}, \boldsymbol{o}^{(t)}$ 和 $L^{(t)}$。对于每一个节点 N，我们需要基于它后面节点的梯度，递归地计算梯度$\nabla_{\mathsf{N}} L$。我们从紧接着最终损失的节点开始递归：

$$\frac{\partial L}{\partial L^{(t)}} = 1. \tag{4.17}$$

在这个导数中，我们假设输出 $\boldsymbol{o}^{(t)}$ 是作为softmax 函数函数的参数来获得关于输出概率的向量 $\hat{\boldsymbol{y}}$。我们也假设损失是迄今为止给定了输入后的真实目标 $y^{(t)}$ 的负对数似然。对于所有 i, t，关于时间步t 输出的梯度$\nabla_{\boldsymbol{o}^{(t)}} L$ 如下：

$$(\nabla_{\boldsymbol{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}}. \tag{4.18}$$

我们从序列的末尾开始，反向进行计算。在最后的时间步τ, $\boldsymbol{h}^{(\tau)}$ 只有 $\boldsymbol{o}^{(\tau)}$ 作为后续节点，因此这个梯度很简单：

$$\nabla_{\boldsymbol{h}_{(\tau)}} L = \boldsymbol{V}^\top \nabla_{\boldsymbol{o}^{(\tau)}} L. \tag{4.19}$$

然后，我们可以从时刻 $t = \tau - 1$ 到 $t = 1$ 反向迭代，通过时间反向传播梯度，注意 $\boldsymbol{h}^{(t)} (t < \tau)$ 同时具有 $\boldsymbol{o}^{(t)}$ 和 $\boldsymbol{h}^{(t+1)}$ 两个后续节点。因此，它的梯度由下式计算

$$\nabla_{\boldsymbol{h}_{(\tau)}} L = \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{o}^{(t)}} L) \tag{4.20}$$

$$= \boldsymbol{W}^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) \mathrm{diag}\left(1 - (\boldsymbol{h}^{(t+1)})^2\right) + \boldsymbol{V}^\top \nabla_{\boldsymbol{o}^{(\tau)}} L, \tag{4.21}$$

其中 $\mathrm{diag}\left(1 - (\boldsymbol{h}^{(t+1)})^2\right)$ 表示一个包含元素 $1 - (h_i^{(t+1)})^2$ 的对角矩阵。这是关于时刻 $t+1$ 与隐藏单元i 关联的双曲正切的雅各比。

一旦获得了计算图内部节点的梯度，我们就可以得到关于参数节点的梯度。因为参数在许多时间步共享，必须在表示这些变量的微积分操作时谨慎对待。我们希望实现的等式使用??中的 `bprop` 方法计算计算图中单一边对梯度贡献。然而微积分中的 $\nabla_{\boldsymbol{W}} f$ 算子，计算 $\boldsymbol{W}$ 对于 f 的贡献时将计算图中*所有*边都考虑进去

了。为了消除这种歧义，我们定义只在 t 时刻使用的虚拟变量 $\boldsymbol{W}^{(t)}$ 作为 $\boldsymbol{W}$ 的副本。然后，我们可以使用 $\nabla_{\boldsymbol{W}^{(t)}}$ 来表示权重在时间步t 对梯度的贡献。

使用这个表示，关于剩下参数的梯度可以由下式给出：

$$\nabla_{\boldsymbol{c}} L = \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}}\right)^\top \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L, \tag{4.22}$$

$$\nabla_{\boldsymbol{b}} L = \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\right)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L, \tag{4.23}$$

$$\nabla_{\boldsymbol{V}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_{\boldsymbol{V}} o_i^{(t)} = \sum_t (\nabla_{\boldsymbol{o}^{(t)}} L) \boldsymbol{h}^{(t)\top}, \tag{4.24}$$

$$\nabla_{\boldsymbol{W}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)} \tag{4.25}$$

$$= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{h}^{(t-1)\top}, \tag{4.26}$$

$$\nabla_{\boldsymbol{U}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} \tag{4.27}$$

$$= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{x}^{(t)\top} \tag{4.28}$$

因为计算图中定义的损失的任何参数都不是训练数据 $\boldsymbol{x}^{(t)}$ 的父节点，所以我们不需要计算关于它的梯度。

4.2.3 作为有向图模型的循环网络

目前为止，我们接触的循环网络例子中损失 $L^{(t)}$ 是训练目标 $\boldsymbol{y}^{(t)}$ 和输出 $\boldsymbol{o}^{(t)}$ 之间的交叉熵。与前馈网络类似，原则上循环网络几乎可以使用任何损失。但必须根据任务来选择损失。如前馈网络，我们通常希望将RNN的输出解释为一个概率分布，并且我们通常使用与分布相关联的交叉熵来定义损失。均方误差是与单位高斯分布的输出相关联的交叉熵损失，例如前馈网络中所使用的。

当我们使用一个预测性对数似然的训练目标，如式(4.12)，我们将RNN训练为能够根据之前的输入估计下一个序列元素 $\boldsymbol{y}^{(t)}$ 的条件分布。这可能意味着，我们最大化对数似然

$$\log p(\boldsymbol{y}^{(t)} \mid \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}), \tag{4.29}$$

或者，如果模型包括来自一个时间步的输出到下一个时间步的连接，

$$\log p(\boldsymbol{y}^{(t)} \mid \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}, \boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(t-1)}). \tag{4.30}$$

将整个序列 $\boldsymbol{y}$ 的联合分布分解为一系列单步的概率预测是捕获关于整个序列完整联合分布的一种方法。当我们不把过去的 $\boldsymbol{y}$ 值反馈给下一步作为预测的条件时，那么有向图模型不包含任何从过去 $\boldsymbol{y}^{(i)}$ 到当前 $\boldsymbol{y}^{(t)}$ 的边。在这种情况下，输出 $\boldsymbol{y}$ 与给定的 $\boldsymbol{x}$ 序列是条件独立的。当我们反馈真实的 $\boldsymbol{y}$ 值（不是它们的预测值，而是真正观测到或生成的值）给网络时，那么有向图模型包含所有从过去 $\boldsymbol{y}^{(i)}$ 到当前 $\boldsymbol{y}^{(t)}$ 的边。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 4.7: TODO

举一个简单的例子，让我们考虑只有一个标量随机变量序列 $\mathbb{Y} = \{\mathrm{y}^{(1)}, \ldots, \mathrm{y}^{(\tau)}\}$ 的RNN模型，也没有额外的输入 x。在时间步t 的输入仅仅是时间步$t-1$ 的输出。该RNN则定义了关于 y 变量的有向图模型。我们使用链式法则（|||c|||）参数化这些条件联合分布：

$$P(\mathbb{Y}) = P(\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(\tau)}) = \prod_{t=1}^{\tau} P(\mathbf{y}^{(t)} \mid \mathbf{y}^{(t-1)}, \mathbf{y}^{(t-2)}, \ldots, \mathbf{y}^{(1)}), \tag{4.31}$$

其中当 $t = 1$ 时竖杠右侧显然为空。因此，根据这样一个模型，一组值 $\{y^{(1)}, \ldots, y^{(\tau)}\}$ 的负对数似然为

$$L = \sum_t L^{(t)}, \tag{4.32}$$

其中

$$L^{(t)} = -\log P(\mathrm{y}^{(t)} = y^{(t)} \mid y^{(t-1)}, y^{(t-2)}, \ldots, y^{(1)}). \tag{4.33}$$

图模型中的边表示哪些变量直接依赖于其他变量。许多图模型的目标是省略不存在强相互作用的边以实现统计和计算的效率。例如，通常可以作Markov假设，即图模型应该只包含从 $\{\mathrm{y}^{(t-k)}, \ldots, \mathrm{y}^{(t-1)}\}$ 到 $\mathrm{y}^{(t)}$ 的边，而不是包含整个过去历史的边。然而，在一些情况下，我们认为整个过去的输入会对序列的下一个元素有一定影响。当我们认为 $\mathrm{y}^{(t)}$ 的分布可能取决于遥远过去 (在某种程度) 的 $y^{(i)}$ 的值，且无法通过 $y^{(t-1)}$ 捕获 $y^{(i)}$ 的影响时，RNN将会很有用。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 4.8: TODO

解释RNN作为图模型的一种方法是将RNN视为定义一个结构为完全图，能够表示任何一对的 y 值之间直接相关性的图模型。图4.7是关于 y 值且具有完全图结构的图模型。该RNN完全图的解释基于排除并忽略模型中的隐藏单元$\boldsymbol{h}^{(t)}$。

更有趣的是，将隐藏单元$\boldsymbol{h}^{(t)}$ 视为随机变量，从而产生RNN的图模型结构[1]。在图模型中包括隐藏单元预示RNN能对观测的联合分布提供非常有效的参数化。假设我们用表格表示法来表示离散值上任意的联合分布，即对每个值可能的赋值有一个单独条目的数组，该条目表示发生该赋值的概率。如果 y 可以取 k 个不同的值，表格表示法将有 $\mathcal{O}(k^{\tau})$ 个参数。比较RNN，由于参数共享，RNN的参数数目为 $\mathcal{O}(1)$ 且是序列长度的函数。可以调节RNN的参数数量来控制模型容量，但不用被迫与序列长度成比例。式(4.5)展示了所述RNN通过循环应用相同的函数 f 以及在每个时间步的相同参数 $\boldsymbol{\theta}$，有效地参数化的变量之间的长期联系。图4.8说明了这个图模型的解释。在图模型中结合 $\boldsymbol{h}^{(t)}$ 节点可以用作过去和未来之间的中间量，从而将它们解耦。遥远过去的变量 $y^{(i)}$ 可以通过其对 $\boldsymbol{h}$ 的影响来影响变量 $y^{(t)}$。该图的结构表明可以在时间步使用相同的条件概率分布来有效地参数化模型，并且当观察到全部变量时，可以有效地评估联合地分配给所有变量的概率。

即便使用高效参数化的图模型，某些操作在计算上仍然具有挑战性。例如，难以预测序列中缺少的值。

循环网络为其减少的参数数目付出的代价是*优化*参数可能变得困难。

在循环网络中使用的参数共享依赖于相同参数可用于不同时间步的假设。等效地，假设在时刻 $t+1$ 变量的条件概率分布在时刻 t 给定的变量是**平稳的**(stationary)，这意味着之前的时间步与下个时间步之间的关系并不依赖于 t。原则上，可以使用 t 作为每个时间步的额外输入，并让学习器在发现任何时间依赖性的同时，在不同时间步之间尽可能多地共享。相比在每个 t 使用不同的条件概率分布已经好很多了，但网络将必须在面对新 t 时进行推断。

为了完整描述将RNN作为图模型的观点，我们必须描述如何从模型采样。我们需要执行的主要操作是简单地从每一时间步的条件分布采样。然而，这会导致额外的复杂性。RNN必须有某种机制来确定序列的长度。这可以通过多种方式实

[1]给定这些变量的父变量，其条件分布是确定性的。尽管设计具有这样确定性的隐藏单元的图模型是很少见的，但这是完全合理的。

现。

在当输出是从词汇表获取的符号的情况下，我们可以添加一个对应于序列末端的特殊符号 (**?**)。当产生该符号时，采样过程停止。在训练集中，我们将该符号作为序列的一个额外成员，即紧跟每个训练样本 $\boldsymbol{x}^{(\tau)}$ 之后。

另一种选择是在模型中引入一个额外的Bernoulli输出，表示在每个时间步决定是否继续产生或停止产生。这种方法比向词汇表增加一个额外符号的方法更普遍的，因为它可以适用于任何RNN，而不仅仅是输出符号序列的RNN。例如，它可以应用于一个产生实数序列的RNN。新的输出单元通常使用sigmoid单元，并通过交叉熵训练。在这种方法中，sigmoid被训练为最大化正确预测的对数似然，即在每个时间步序列是否结束或继续。

确定序列长度 τ 的另一种方法是将一个额外的输出添加到模型并预测整数 τ 本身。模型可以采出 τ 的值，然后采 τ 步有价值的数据。这种方法需要在每个时间步的循环更新中增加一个额外输入，使得循环更新知道它是否是靠近所产生序列的末尾。这种额外的输入可以是 τ 的值，也可以是 $\tau - t$ 即剩下时间步的数量。如果没有这个额外的输入，RNN可能会产生突然结束序列，如一个句子在最终完整前结束。此方法是基于分解

$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}) = P(\tau) P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)} \mid \tau) \tag{4.34}$$

直接预测 τ 的例子见**?**。

4.2.4 基于上下文的RNN序列建模

上一节描述了没有输入 $\boldsymbol{x}$ 时，关于随机变量序列 $y^{(t)}$ 的RNN如何对应于有向图模型。当然，如式(4.8)所示的RNN包含一个输入序列 $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(\tau)}$。一般情况下，RNN允许将图模型的观点扩展到不仅代表 y 变量的联合分布也能表示给定 $\boldsymbol{x}$ 后 y 条件分布。如在**??**节的前馈网络情形中所讨论的，任何代表变量 $P(\boldsymbol{y}; \boldsymbol{\theta})$ 的模型都能被解释为代表条件分布 $P(\boldsymbol{y}; \boldsymbol{\omega})$ 的模型，其中 $\boldsymbol{\omega} = \boldsymbol{\theta}$。我们能像之前一样使用 $P(\boldsymbol{y}; \boldsymbol{\omega})$ 代表分布 $P(\boldsymbol{y}; \boldsymbol{x})$ 来扩展这样的模型，但要令 $\boldsymbol{\omega}$ 是关于 $\boldsymbol{x}$ 的函数。在RNN的情况，这可以通过不同的方式来实现。此处，我们回顾最常见和最明显的选择。

之前，我们已经讨论了将 $t = 1, \ldots, \tau$ 的向量 $\boldsymbol{x}^{(t)}$ 序列作为输入的RNN。另一种选择是只使用单个向量 $\boldsymbol{x}$ 作为输入。当 $\boldsymbol{x}$ 是一个固定大小的向量，我们可以简单地将其看作产生 $\boldsymbol{y}$ 序列RNN的额外输入。将额外输入提供到RNN的一些常见方法是：

1. 在每个时刻作为一个额外输入，或
2. 作为初始状态 $\boldsymbol{h}^{(0)}$，或

3. 结合两种方式。

第一个也是最常用的方法如图4.9所示。输入 $\boldsymbol{x}$ 和每个隐藏单元向量 $\boldsymbol{h}^{(t)}$ 之间的相互作用通过新引入的权重矩阵 $\boldsymbol{R}$ 参数化的，这是只包含 y 序列的模型所没有的。同样的乘积 $\boldsymbol{x}^\top \boldsymbol{R}$ 在每个时间步作为隐藏单元的一个额外输入。我们可以认为 $\boldsymbol{x}$ 的选择（确定 $\boldsymbol{x}^\top \boldsymbol{R}$ 值），是有效地用于每个隐藏单元的一个新偏置参数。权重与输入保持独立。我们可以把这种模型视为采取非条件模型 $\boldsymbol{\theta}$ 并使之成为 $\boldsymbol{\omega}$，其中 $\boldsymbol{\omega}$ 内的偏置参数是输入的函数。

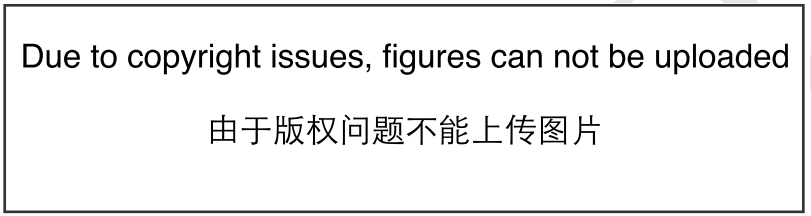


Figure 4.9: TODO

RNN可以接收向量序列 $\boldsymbol{x}^{(t)}$ 作为输入，而不是仅接收单个向量 $\boldsymbol{x}$ 作为输入。式(4.8)描述的RNN对应条件分布 $P(\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)} \mid \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)})$，并在条件独立的假设下这个分布分解为

$$\prod_t P(\boldsymbol{y}^{(t)} \mid \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}). \tag{4.35}$$

为去掉条件独立的假设，我们可以在时刻 t 的输出到时间 $t+1$隐藏单元添加连接，如图4.10所示。该模型就可以代表关于 $\boldsymbol{y}$ 序列的任意概率分布。这种给定一

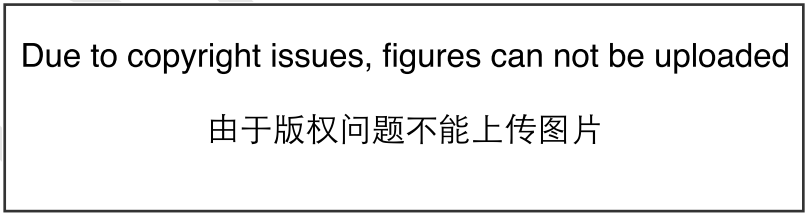


Figure 4.10: TODO

个序列表示另一个序列分布的模型的还是有一个限制，就是这两个序列的长度必须是相同的。我们将在4.4节描述如何消除这种限制。

4.3 双向RNN

目前为止我们考虑的所有循环神经网络有一个“因果”结构，意味着在时刻 t 的状态只能从过去的序列 $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t-1)}$ 以及当前的输入 $\boldsymbol{x}^{(t)}$ 捕获信息。我们还讨论了某些在 $\boldsymbol{y}$ 可用时，允许过去的 $\boldsymbol{y}$ 值信息影响当前状态的模型。

然而，在许多应用中，我们要输出的 $\boldsymbol{y}^{(t)}$ 的预测可能依赖于整个输入序列。例如，在语音识别中，由于协同发音，当前声音作为音素的正确解释可能取决于未来几个音素，甚至潜在的可能取决于未来的几个词，因为词与附近的词之间的存在语义依赖：如果当前的词有两种声学上合理的解释，我们可能要在更远的未来(和过去）来寻找信息区分它们。这在手写识别和许多其他序列到序列学习的任务中也是如此，将会在下一节中描述。

双向循环神经网络（或双向RNN）为满足这种需要而被发明 (**?**)。他们在需要双向信息的应用中非常成功 (**?**)，如手写识别 (**??**)，语音识别 (**??**) 以及生物信息学 (**?**)。

顾名思义，双向RNN结合时间上从序列起点开始移动的RNN和另一个时间上从序列末尾开始移动的RNN。图4.11展示了典型的双向RNN，其中 $\boldsymbol{h}^{(t)}$ 代表通过时间向前移动的子RNN的状态，$\boldsymbol{g}^{(t)}$ 代表通过时间向后移动的子RNN的状态。这允许输出单元 $\boldsymbol{o}^{(t)}$ 能够计算同时依赖于过去和未来且对时刻 t 的输入值最敏感的表示，而不必指定 t 周围固定大小的窗口（这是前馈网络、卷积网络或具有固定大小的先行缓存器的RNN必须要做的）。

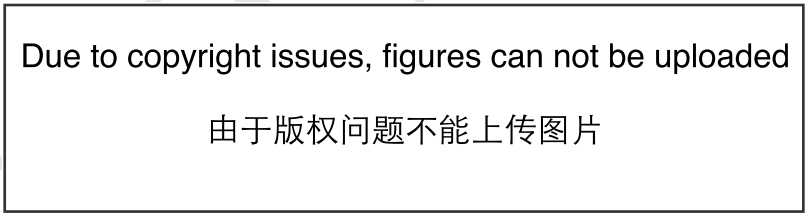


Figure 4.11: TODO

这个想法可以自然扩展到 2 维输入，如图像，由*四个*RNN组成，每一个沿着四个方向中的一个计算：上、下、左、右。如果RNN能够学习到承载长期信息，那在 2 维网格每个点 (i, j) 的输出 $O_{i,j}$ 就能计算一个能捕捉到大多局部信息并且依赖于长期输入的表示。相比卷积网络，应用于图像的RNN通常更昂贵，但允许同一特征图的特征之间存在长期横向的相互作用 (**??**)。实际上，对于这样的RNN，前向传播公式可以写成表示使用卷积的形式，计算自底向上到每一层的输入（在整合横向相互作用的特征图的循环传播之前）。

4.4 基于编码-解码的序列到序列架构

我们已经在图4.5看到RNN如何将输入序列映射成固定大小的向量，在图4.9中看到RNN如何将固定大小的向量映射成一个序列，在图4.3、4.4、4.10和4.11中看到RNN如何将一个输入序列映射到等长的输出序列。

本节我们讨论如何训练RNN，使其将输入序列映射到不一定等长的输出序列。这在许多场景中都有应用，如语音识别、机器翻译或问答，其中训练集的输入和输出序列的长度通常不相同（虽然它们的长度可能相关）。

我们经常将RNN的输入称为“上下文”。我们希望产生此上下文的表示，C。这个上下文 C 可能是一个概括输入序列 $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n_x)})$ 的向量或者向量序列。

用于映射可变长度序列到另一可变长度序列最简单的RNN架构最初由**?** 提出，之后不久由**?** 独立开发，并且第一个使用这种方法获得翻译的最好结果。前一系统是对另一个机器翻译系统产生的建议进行评分，而后者使用独立的循环网络来生成翻译。这些作者分别将该架构称为编码-解码或序列到序列架构，如图4.12所示。这个想法非常简单：(1) **编码器**(encoder) 或**读取器** (reader) 或**输入** (input)RNN处理输入序列。编码器输出上下文 C （通常是最终隐藏状态的简单函数）。(2)**解码器**(decoder) 或**写入器** (writer) 或**输出** (output)RNN则以固定长度的向量（如图4.9）为条件产生输出序列 $\boldsymbol{Y} = (\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(n_y)})$。这种架构对比本章前几节提出的架构的创新之处在于长度 n_x 和 n_y 可以彼此不同，而之前的架构约束 $n_x = n_y = \tau$。在一个序列到序列的架构中，两个RNN共同训练以最大化 $\log P(\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(n_y)} \mid \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n_x)})$(关于训练集中所有 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 对的平均)。编码器RNN的最后一个状态 $\boldsymbol{h}_{n_x}$ 通常被当作输入的表示 C 并作为解码器RNN的输入。

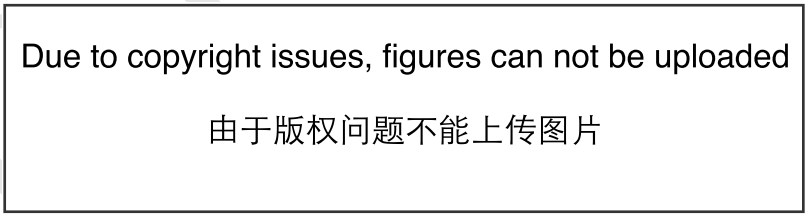


Figure 4.12: TODO

如果上下文 C 是一个向量，则编码器RNN只是在4.2.4节描述的向量到序列RNN。正如我们所见，向量到序列RNN至少有两种接受输入的方法。输入可以被提供为RNN的初始状态，或连接到每个时间步中的隐藏单元。这两种方式也可以结合。

这里并不强制要求编码器与解码器的隐藏层具有相同的大小。

此架构的一个明显限制是，编码器RNN输出的上下文 C 的维度太小而难以适当地概括一个长序列。这种现象由**?** 在机器翻译中观察到。他们提出让 C 成为可变长度的序列，而不是一个固定大小的向量。此外，他们还引入了将序列 C 的元素和输出序列的元素相关联的**注意机制**(attention mechanism)。可在**??**了解更多详情。

4.5 深度循环网络

大多数RNN中的计算可以分解成三块参数及其相关的变换：

1. 从输入到隐藏状态，
2. 从前一隐藏状态到下一隐藏状态，以及
3. 从隐藏状态到输出。

根据图4.3中的RNN架构，这三个块都与单个权重矩阵相关联。换句话说，当网络被展开时，每个块对应一个浅的变换。能通过深度MLP内单个层来表示的变换称为浅变换。通常，这是由学好的仿射变换和一个固定非线性表示的转换。

在这些操作中引入深度会有利的吗？实验证据 (**??**) 强烈暗示理应如此。实验证据与我们需要足够的深度以执行所需映射的想法一致。可以参考**??** 或**?** 了解更早的关于深度RNN的研究。

? 第一个展示了将RNN的状态分为多层的显著好处，如图4.13（左）。我们可以认为，在图4.13(a) 所示层次结构中较低的层起到了将原始输入转化为对更高层的隐藏状态更合适表示的作用。**?** 更进一步提出在上述三个块中各使用一个单独的MLP（可能是深度的），如图4.13(b) 所示。考虑表示容量，我们建议在这三个步中都分配足够的容量，但增加深度可能会因为优化困难而损害学习效果。在一般情况下，更容易优化较浅的架构，加入图4.13(b) 的额外深度导致从时间步 t 的变量到时间步 $t+1$ 的最短路径变得更长。例如，如果具有单个隐藏层的MLP被用于状态到状态的转换，那么与图4.3相比，我们就会加倍任何两个不同时间步变量之间最短路径的长度。然而**?** 认为，在隐藏到隐藏的路径中引入跳跃连接可以缓和这个问题，如图4.13(c) 所示。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 4.13: TODO

4.6 递归神经网络

递归神经网络[2]代表循环网络的另一个扩展，它被构造为深的树状结构而不是RNN的链状结构，因此是不同类型的计算图。对于递归网络的典型的计算图如图4.14所示。递归神经网络由**?** 引入，而**?** 描述了这类网络的潜在用途——学习推论。递归网络已成功地应用于输入是*数据结构*的神经网络(**??**)，如自然语言处理 (**???**) 和计算机视觉 (**?**)。

递归网络的一个明显优势是，对于相同的长度为 τ 的序列，深度（通过非线性操作的组合数量来衡量）可以急剧地从 τ 减小为 $\mathcal{O}(\log \tau)$，这可能有助于解决长期依赖。一个悬而未决的问题是如何以最佳的方式构造树。一种选择是使用不依赖于数据的树结构，如平衡二叉树。在某些应用领域，外部方法可以为选择适当的树结构提供借鉴。例如，处理自然语言的句子时，用于递归网络的树结构可以被固定为句子语法分析树的结构（可以由自然语言语法分析程序提供）(**??**)。理想的情况下，人们希望学习器自行发现和推断适合于任意给定输入的树结构，如 (**?**) 所建议。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 4.14: TODO

许多递归网络想法的变种是可能的。例如，**?** 和**?** 将数据与树结构相关联，并将输入和目标与树的单独节点相关联。由每个节点执行的计算无须是传统的人工神经计算（所有输入的仿射变换后跟一个单调非线性）。例如，**?** 提出用张量运算和双线性形式，在这之前人们已经发现当概念是由连续向量（嵌入）表示时，这

[2]我们建议不要将 “递归神经网络” 缩写为 “RNN”，以免与 “循环神经网络” 混淆。

种方式有利于建模概念之间的关系 (**??**)。

4.7 长期依赖的挑战

学习循环网络长期依赖的数学挑战在**??**中引入。根本问题是，经过许多阶段传播后的梯度倾向于消失（大部分时间）或爆炸（很少，但对优化过程影响很大)。即使我们假设循环网络是参数稳定的（可存储记忆，且梯度不爆炸)，但长期依赖的困难来自比短期相互作用指数小的权重（涉及许多雅各比相乘)。许多资料提供了更深层次的讨论 (**????**)。在这一节中，我们会更详细地描述该问题。其余几节介绍克服这个问题的方法。

循环网络涉及相同函数的多次组合，每个时间步一次。这些组合可以导致极端非线性行为，如图4.15所示。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 4.15: TODO

特别地，循环神经网络所使用的函数组合有点像矩阵乘法。我们可以认为，循环联系

$$\boldsymbol{h}^{(t)} = \boldsymbol{W}^{\top}\boldsymbol{h}^{(t-1)}, \tag{4.36}$$

是一个非常简单的、缺少非线性激活函数和输入 $\boldsymbol{x}$ 的循环神经网络。如**??**描述，这种递推关系本质上描述了幂法。它可以被简化为

$$\boldsymbol{h}^{(t)} = (\boldsymbol{W}^{t})^{\top}\boldsymbol{h}^{(0)}, \tag{4.37}$$

而当 $\boldsymbol{W}$ 符合下列形式的特征分解

$$\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\top}, \tag{4.38}$$

其中 $\boldsymbol{Q}$ 正交，循环性可进一步简化为

$$\boldsymbol{h}^{(t)} = \boldsymbol{Q}^{\top}\boldsymbol{\Lambda}^{t}\boldsymbol{Q}\boldsymbol{h}^{(0)}. \tag{4.39}$$

特征值提升到 t 次后，导致幅值不到一的特征值衰减到零，而幅值大于一的就会激增。任何不与最大特征向量对齐的 $\boldsymbol{h}^{(0)}$ 的部分将最终被丢弃。

这个问题是针对循环网络的。在标量情况下，想象多次乘一个权重 w。根据 w 的幅值，该乘积 w^t 要么消失要么激增。然而，如果每个时刻使用不同权重 $w^{(t)}$ 的非循环网络，情况就不同了。如果初始状态给定为 1，那么时刻 t 的状态可以由 $\prod_t w^{(t)}$ 给出。假设 $w^{(t)}$ 的值是随机生成的，各自独立，且有 0 均值 v 方差。乘积的方差就为 $\mathcal{O}(v^n)$。为了获得某些期望的方差 v^*，我们可以选择单个方差为 $v = \sqrt[n]{v^*}$ 权重。因此，非常深的前馈网络通过精心设计的比例可以避免梯度消失和爆炸问题，如**?** 所主张的。

RNN梯度消失和爆炸问题是由不同研究人员独立发现 (**???**)。有人可能会希望通过简单地停留在梯度不消失或爆炸的参数空间来避免这个问题。不幸的是，为了储存记忆并对小扰动的鲁棒，RNN必须进入参数空间中的梯度消失区域 (**??**)。具体来说，每当模型能够表示长期依赖，长期相互作用的梯度幅值就会变得指数小（相比短期相互作用的梯度幅值）。这并不意味着这是不可能学习的，由于长期依赖关系的信号很容易被短期相关性产生的最小波动隐藏，因而可能需要很长的时间去学习长期依赖。实践中，**?** 的实验表明，当我们增加了需要捕获的依赖关系的跨度，基于梯度的优化变得越来越困难，SGD在长度仅为 10 或 20 的序列上成功训练传统RNN的概率迅速变为 0。

将循环网络作为动力系统更深入探讨的资料见**???** 及 **?** 的回顾。本章的其余部分将讨论目前已经提出的降低学习长期依赖（在某些情况下，允许一个RNN学习横跨数百步的依赖）难度的不同方法，但学习长期依赖的问题仍是深度学习中的一个主要挑战。

4.8 回声状态网络

从 $\boldsymbol{h}^{(t-1)}$ 到 $\boldsymbol{h}^{(t)}$ 的循环权重映射以及从 $\boldsymbol{x}^{(t)}$ 到 $\boldsymbol{h}^{(t)}$ 的输入权重映射是一个循环网络中最难学习的参数。研究者 (**???**) 提出避免这种困难的方法是设定循环隐藏单元，使其能很好地捕捉过去输入历史，并且*只学习输出权重*。**回声状态网络**(echo state networks) 或ESN(**??**)，以及**流体状态机**(liquid state machines)(**?**) 分别独立地提出了这种想法。后者是类似的，只不过它使用脉冲神经元（二进制输出）而不是ESN中的连续隐藏单元。ESN和流体状态机都被称为**储层计算**(reservoir computing)(**?**), 因为隐藏单元形成了可能捕获输入历史不同方面的临时特征池。

储层计算循环网络类似于核机器, 这是思考它们的一种方式：它们将任意长度的序列（到时刻 t 的输入历史）映射为一个长度固定的向量（循环状态 $\boldsymbol{h}^{(t)}$），之后可以施加一个线性预测算子（通常是一个线性回归）以解决感兴趣的问题。训练判据就可以很容易地设计为输出权重的凸函数。例如，如果输出是从隐藏单元到输出目标的线性回归，训练判据就是均方误差，由于是凸的就可以用简单的学习算法可靠地解决 (**?**)。

因此，重要的问题是：我们如何设置输入和循环权重才能让一组丰富的历史

可以在循环神经网络的状态中表示？储层计算研究所给出的答案是将循环网络视为动态系统，并设定让动态系统接近稳定边缘的输入和循环权重。

最初的想法是使状态到状态转换函数的雅各比的特征值接近 1。如**??**解释，循环网络的一个重要特征就是雅各比的特征值谱 $\boldsymbol{J}^{(t)} = \frac{\partial s^{(t)}}{\partial s^{(t-1)}}$。特别重要的是 $\boldsymbol{J}^{(t)}$ 的**谱半径**(spectral radius)，定义为特征值的最大绝对值。

为了解谱半径的影响，可以考虑反向传播中雅各比矩阵 $\boldsymbol{J}$ 不随 t 改变的简单情况。例如当网络是纯线性时，会发生这种情况。假设 $\boldsymbol{J}$ 特征值 λ 对应的特征向量为 $\boldsymbol{v}$。考虑当我们通过时间向后传播梯度向量时会发生什么。如果我们刚开始有梯度向量 $\boldsymbol{g}$，然后经过反向传播的一个步骤后，我们将有 $\boldsymbol{J}\boldsymbol{g}$，在 n 步之后我们会得到 $\boldsymbol{J}^n\boldsymbol{g}$。现在考虑如果我们向后传播扰动版本的 $\boldsymbol{g}$ 会发生什么。如果我们刚开始是 $\boldsymbol{g} + \delta\boldsymbol{v}$，一步之后，我们会得到 $\boldsymbol{J}(\boldsymbol{g} + \delta\boldsymbol{v})$。$n$ 步之后，我们将得到 $\boldsymbol{J}^n(\boldsymbol{g} + \delta\boldsymbol{v})$。由此我们可以看出，由 $\boldsymbol{g}$ 开始的反向传播和由 $\boldsymbol{g} + \delta\boldsymbol{v}$ 开始的反向传播，n 步之后偏离 $\delta\boldsymbol{J}^n\boldsymbol{v}$。如果 $\boldsymbol{v}$ 选择为 $\boldsymbol{J}$ 特征值 λ 对应的一个单位特征向量，那么在每一步乘雅各比只是简单地缩放。反向传播的两次执行分离的距离为 $\delta|\lambda|^n$。当 $\boldsymbol{v}$ 对应于最大特征值 $|\lambda|$，初始扰动为 δ 时这个扰动达到可能的最宽分离。

当 $|\lambda| > 1$，偏差 $\delta|\lambda|^n$ 就会指数增长。当 $|\lambda| < 1$，偏差就会变得指数小。

当然，这个例子假定雅各比在每个时间步是相同的，即对应于没有非线性循环网络。当非线性存在时，非线性的导数将在许多时间步后接近零，并有助于防止因过大的谱半径而导致的爆炸。事实上，关于回声状态网络的最近工作提倡使用远大于 1 的谱半径(**??**)。

我们已经说过多次，通过反复矩阵乘法的反向传播同样适用于没有非线性的正向传播的网络，其状态为 $\boldsymbol{h}^{(t+1)} = \boldsymbol{h}^{(t)\top}\boldsymbol{W}$。

当线性映射 $\boldsymbol{W}^\top$ 在 L^2 范数的测度下总是缩小 $\boldsymbol{h}$，那么我们说这个映射是**收缩**(contractive) 的。当谱半径小于一，则从 $\boldsymbol{h}^{(t)}$ 到 $\boldsymbol{h}^{(t+1)}$ 的映射是收缩的，因此小变化在每个时间步后变得更小。当我们使用有限精度（如 32 位整数）来存储状态向量时，必然会使得网络忘掉过去的信息。

雅各比矩阵告诉我们 $\boldsymbol{h}^{(t)}$ 一个微小的变化如何向前传播，或等价的，$\boldsymbol{h}^{(t+1)}$ 的梯度如何向后传播。需要注意的是，$\boldsymbol{W}$ 和 $\boldsymbol{J}$ 都不需要是对称的（尽管它们是实方阵），因此它们可能有复的特征值和特征向量，其中虚数分量对应于潜在的振荡行为（如果迭代地应用同一雅各比）。即使 $\boldsymbol{h}^{(t)}$ 或 $\boldsymbol{h}^{(t)}$ 感兴趣的小变化在反向传播中是实值的，它们可以以复数基来表示。重要的是，当向量乘以矩阵时，这些复数基的系数幅值（复数的绝对值）会发生什么变化。幅值大于 1 的特征值对应于放大（如果反复应用则指数增长）或收缩（如果反复应用则指数减小）。

非线性映射情况时，雅各比会在每一步任意变化。<BAD> 因此，动态量变得更加复杂。然而，一个小的初始变化多步之后仍然会变成一个大的变化。纯线性和非线性情况的一个不同之处在于使用压缩非线性（如 tanh）可以使循环动态量有界。注意，即使前向传播动态量有界，反向传播的动态量仍然可能无界，例

如，当 tanh 序列都在它们状态中间的线性部分，并且由谱半径大于 1 的权重矩阵连接。然而，所有 tanh 单元同时位于它们的线性激活点是非常罕见的。

回声状态网络的策略是简单地固定权重使其具有一定的谱半径如 3，其中信息通过时间向前传播，但会由于饱和非线性单元（如 tanh）的稳定作用而不会爆炸。

最近，已经有研究表明，用于设置ESN权重的技术可以用来*初始化*完全可训练的循环网络的权重（通过时间反向传播来训练隐藏到隐藏的循环权重），帮助学习长期依赖(**??**)。在这种设定下，结合**??**中稀疏初始化的方案，设置 1.2 的初始谱半径表现不错。

4.9 渗漏单元和其他多时间尺度的策略

处理长期依赖的一种方法是设计工作在多个时间尺度的模型，使其某些部分在细粒度时间尺度上操作并能处理小细节，而其他部分在粗时间尺度上操作并能把遥远过去的信息更有效地传递过来。存在多种同时构建粗细时间尺度的策略。这些策略包括在时间轴增加跳跃连接，“渗漏单元” 使用不同时间常数整合信号，并去除一些用于建模细粒度时间尺度的连接。

4.9.1 时间维度的跳跃连接

增加从遥远过去的变量到目前变量的直接连接是得到粗时间尺度的一种方法。使用这样跳跃连接的想法可以追溯到**?**，紧接是向前馈网络引入延迟的想法 (**?**)。在普通的循环网络中，循环从时刻 t 的单元连接到时刻 $t+1$ 单元。可以构造较长的延迟循环网络(**?**)。

正如我们在**??**看到，梯度可能*关于时间步数数消失或成倍爆炸*。(**?**) 引入了 d 延时的循环连接以减轻这个问题。现在导数指数减小的速度与 $\frac{\tau}{d}$ 相关而不是 τ。既然同时存在延迟和单步连接，梯度仍可能成 t 指数爆炸。这允许学习算法捕获更长的依赖性，但不是所有的长期依赖都能在这种方式下良好地表示。

4.9.2 渗漏单元和一系列不同时间尺度

获得导数乘积接近 1 的另一方式是设置*线性*自连接单元，并且这些连接的权重接近 1。

我们对某些 v 值应用更新 $\mu^{(t)} \leftarrow \alpha\mu^{(t-1)} + (1-\alpha)v^{(t)}$ 累积一个滑动平均值 $\mu^{(t)}$，其中 α 是一个从 $\mu^{(t-1)}$ 到 $\mu^{(t)}$ 线性自连接的例子。当 α 接近 1 时，滑动平均值能记住过去很长一段时间的信息，而当 α 接近 0，关于过去的信息被迅速丢弃。线性自连接的隐藏单元可以模拟滑动平均的行为。这种隐藏单元称为**渗漏单元**(leaky unit)。

d时间步的跳跃连接是确保单元总能被之前 d 个时间步的值影响的一个方法。使用权重接近 1 的线性自连接是确保该单元可以访问过去值的不同方式。线性自连接通过调节实值 α 更平滑灵活地调整这种效果，而不是调整整数值的跳跃长度。

这个想法由**?** 和**?** 提出。在回声状态网络中，渗漏单元也被发现很有用 (**?**)。

两个基本策略可以设置渗漏单元使用的时间常数。一种策略是手工将其固定为常数，例如在初始化时从某些分布进行采样它们的值。另一种策略是使时间常数成为自由变量，并学习出来。在不同时间尺度使用这样的渗漏单元似乎能帮助学习长期依赖(**??**)。

4.9.3 删除连接

处理长期依赖另一种方法是在多个时间尺度组织RNN状态的想法 (**?**)，信息在较慢的时间尺度上更容易长距离流动。

这个想法与之前讨论的时间维度上的跳跃连接不同，因为它涉及主动*删除*长度为一的连接并用更长的连接替换它们。以这种方式修改的单元被迫在长时间尺度上运作。通过时间跳跃连接是*添加*边。收到这种新连接的单元，可以学习在长时间尺度上运作，但也可以选择专注于自己其他的短期连接。

强制一组循环单元在不同时间尺度上运作有不同的方式。一种选择是使循环单元变成渗漏单元，但不同的单元组关联不同的固定时间尺度。这由**?** 提出，并被成功应用于**?**。另一种选择是使显式且离散的更新发生在不同的时间，不同的单元组有不同的频率。这是**?** 和**?** 的方法。它在一些基准数据集上表现不错。

4.10 长短期记忆和其他门限 RNN

本文撰写之时，实际应用中最有效的序列模型称为**门限 RNN**(gated RNN)。包括基于**长短期记忆**(long short-term memory) 和基于**门限循环单元**(gated recurrent unit) 的网络。

像渗漏单元一样，门限 RNN想法也是基于生成通过时间的路径，其中导数既不消失也不发生爆炸。渗漏单元通过手动选择常量的连接权重或参数化的连接权重来达到这一目的。门限 RNN将其推广为在每个时间步都可能改变的连接权重。

渗漏单元允许网络在较长持续时间内*积累*信息（诸如用于特定特征或类的线索）。然而，一旦该信息被使用，让神经网络*遗忘*旧的状态可能是有用的。例如，如果一个序列是由子序列组成，我们希望渗漏单元能在各子序列内积累线索，我们需要将状态设置为 0 以忘记旧状态的的机制。我们希望神经网络学会决定何时清除状态，而不是手动决定。这就是门限 RNN要做的事。

4.10.1 LSTM

引入自循环的巧妙构思，以产生梯度长时间持续流动的路径是初始**长短期记忆** (long short-term memory, LSTM) 的核心贡献 (**?**)。其中一个关键扩展是使自循环的权重视上下文而定，而不是固定的 (**?**)。门限此自循环（由另一个隐藏单元控制）的权重，累积的时间尺度可以动态地改变。在这种情况下，即使是具有固定参数的LSTM，累积的时间尺度也可以因输入序列而改变，因为时间常数是模型本身的输出。LSTM已经在许多应用中取得重大成功，如无约束手写识别 (**?**)、语音识别 (**??**)、手写识别 (**?**)、机器翻译 (**?**)，为图像生成标题 (**???**) 和解析 (**?**)。

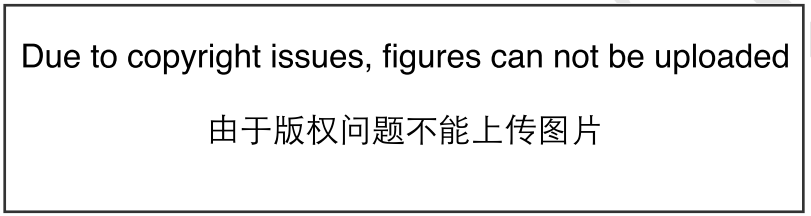


Figure 4.16: TODO

LSTM块如图4.16所示。在浅循环网络的架构下，相应的前向传播公式如下。更深的架构也成功地应用于 (**??**)。<BAD>LSTM循环网络除了外部RNN的循环外，还具有内部的循环（自环）的 “LSTM细胞”，因此不是简单地向输入和循环单元的仿射变换之后施加一个逐元素的非线性。与普通的循环网络类似，每个单元有相同的输入和输出，但也有更多的参数和控制信息流动的门限单元系统。最重要的组成部分是状态单元 $s_i^{(t)}$，与前一节讨论的渗漏单元有类似的线性自环。然而，此处自环的权重（或相关联的时间常数）由**遗忘门**(forget gate)$f_i^{(t)}$ 控制(时刻 t 和细胞 i)，由sigmoid单元将权重设置为 0 和 1 之间的值：

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right), \tag{4.40}$$

其中 $\boldsymbol{x}^{(t)}$ 是当前输入向量，$\boldsymbol{h}^t$ 是当前隐藏层向量，$\boldsymbol{h}^t$ 包含所有LSTM细胞的输出。$\boldsymbol{b}^f$, $\boldsymbol{U}^f$, $\boldsymbol{W}^f$ 分别是偏置、输入权重和遗忘门的循环权重。因此LSTM细胞内部状态以如下方式更新，其中有一个条件的自环权重 $f_i^{(t)}$：

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right), \tag{4.41}$$

其中 $\boldsymbol{b}$, $\boldsymbol{U}$, $\boldsymbol{W}$ 分别是 LSTM 细胞中的偏置、输入权重和遗忘门的循环权重。**外部输入门** (external input gate) 单元 $g_i^{(t)}$ 以类似遗忘门（使用一个sigmoid获得一个

0 和 1 之间的值）的方式更新，但有自身的参数：

$$g_i^{(t)} = \sigma\Big(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\Big). \tag{4.42}$$

LSTM细胞的输出 $h_i^{(t)}$ 也可以由**输出门** (output gate) $q_i^{(t)}$ 关闭（使用sigmoid单元作为门限）：

$$h_i^{(t)} = \tanh\big(s_i^{(t)}\big) q_i^{(t)}, \tag{4.43}$$

$$q_i^{(t)} = \sigma\Big(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\Big), \tag{4.44}$$

其中 $\boldsymbol{b}^o$, $\boldsymbol{U}^o$, $\boldsymbol{W}^o$ 分别是其偏置、输入权重和遗忘门的循环权重。在这些变体中，可以选择使用细胞状态 $s_i^{(t)}$ 作为额外的输入（及其权重），输入到第 i 个单元的三个门，如图4.16所示。这将需要三个额外的参数。

LSTM网络比简单的循环架构更易于学习长期依赖，先是用于测试长期依赖学习能力的人工数据集 (**???**)，然后是在具有挑战性的序列处理任务上获得最先进的表现（(**???**)）。LSTM的变体和替代也已经被研究和使用，会在下文进行讨论。

4.10.2 其他门限 RNN

LSTM架构中哪些部分是真正必须的？还可以设计哪些其他成功架构允许网络动态地控制的时间尺度和不同单元的遗忘行为？

最近关于门限 RNN的工作给出了这些问题的某些答案，其单元也被称为门限循环单元或 GRU(**?????**)。与LSTM的主要区别是，单个门限单元同时控制遗忘因子和更新状态单元的决定。更新公式如下：

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + (1 - u_i^{(t-1)}) \sigma\Big(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)}\Big), \tag{4.45}$$

其中 $\boldsymbol{u}$ 代表 “更新” 门，$\boldsymbol{r}$ 表示 “复位” 门。它们的值就如通常所定义的：

$$u_i^{(t)} = \sigma\Big(b_i^u + \sum_j U_{i,j}^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)}\Big), \tag{4.46}$$

和

$$r_i^{(t)} = \sigma\Big(b_i^r + \sum_j U_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t)}\Big). \tag{4.47}$$

复位和更新门能独立地“忽略”的状态向量的一部分。更新门像条件渗漏累积器一样可以线性门限任意维度，从而选择将它复制（在sigmoid的一个极端）或完全由新的“目标状态”值（朝向渗漏累积器的收敛方向）替换并完全忽略它（在另一个极端)。复位门控制状态用于计算下一个目标状态的某部分，引入了过去状态和未来状态之间的关系附加非线性效应。

围绕这一主题可以设计更多的变种。例如复位门（或遗忘门）的输出可以在多个隐藏单元间共享。或者，全局门的乘积（覆盖一整组的单元，例如整一层）和一个局部门（每单元）可用于结合全局控制和局部控制。然而，一些调查发现没有LSTM和 GRU 架构的变种能明显在广泛的任务中同时击败这两个 (**??**)。**?** 发现其中的关键因素是 遗忘门，而**?** 发现向LSTM遗忘门加入 1 的偏置(由**?** 提倡)能让LSTM变得与已探索的最佳变种一样健壮。

4.11 优化长期依赖

??和4.7节已经描述过在许多时间步上优化RNN时发生的梯度消失和爆炸的问题。

由**?** 提出了一个有趣的想法是，二阶导数可能在一阶导数消失的同时消失。二阶优化算法可以大致被理解为将一阶导数除以二阶导数（在更高维数，由梯度乘以海森的逆)。如果二阶导数与一阶导数以类似的速率收缩，那么一阶和二阶导数的比率可保持相对恒定。不幸的是，二阶方法有许多缺点，包括高的计算成本、需要一个大的minibatch、并且倾向于被吸引到鞍点。**?** 发现，采用二阶方法有前途的结果。之后，**?** 发现使用较简单的方法可以达到类似的结果，例如细心初始化的Nesterov动量法。更详细的内容参考**?**。应用于LSTM时，这两种方法在很大程度上被单纯的SGD（甚至没有动量）取代。这是机器学习中一个延续的主题，设计一个易于优化模型通常比设计出更加强大的优化算法更容易。

4.11.1 截断梯度

如**??**节讨论，强非线性函数如由许多时间步计算的循环网络往往倾向于非常大或非常小幅度的梯度。如图 |||c||| 和图4.17所示，我们可以看到，目标函数（作为参数的函数）有一个“地形”伴随一个“悬崖”：宽且相当平坦区域被目标函数变化快的小区域隔开，形成了一种悬崖。

这导致的困难是，当参数梯度非常大时，梯度下降的参数更新可以将参数抛出很远，进入目标函数较大的区域，到达当前解所作的努力变成了无用功。梯度告诉我们，围绕当前参数的无穷小区域内最速下降的方向。这个无穷小区域之外，代价函数可能开始沿曲线背面而上。更新必须被选择为足够小，以避免过分穿越向上的曲面。我们通常使用衰减速度足够慢的学习率，使连续的步骤具有大致相同的学习率。适合于一个相对线性的地形部分的步长经常在下一步进入地形中更加弯曲的部分时变得不适合，会导致上坡运动。

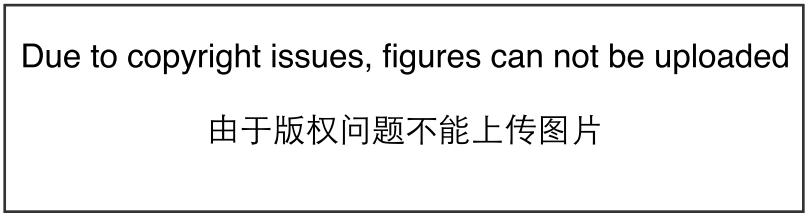


Figure 4.17: TODO

一个简单的解决方案已被从业者使用多年：**截断梯度**(clipping the gradient)。此想法有不同实例 (**??**)。一种选择是在参数更新之前，*逐元素*地截断minibatch产生的参数梯度(**?**)。另一种是在参数更新之前*截断梯度* $\boldsymbol{g}$ *的范数* $\|\boldsymbol{g}\|$(**?**)：

$$\text{if } \|\boldsymbol{g}\| > v \tag{4.48}$$

$$\boldsymbol{g} \leftarrow \frac{\boldsymbol{g}v}{\|g\|}, \tag{4.49}$$

其中 v 是范数上界，$\boldsymbol{g}$ 用来更新参数。因为所有参数（包括不同的参数组，如权重和偏置）的梯度被单个缩放因子联合重整化，所以后一方法具有的优点是保证了每个步骤仍然是在梯度方向上的，但实验表明两种形式类似。虽然参数更新与真实梯度具有相同的方向梯度，经过梯度范数截断，参数更新的向量范数现在变得有界。这种有界梯度能避免执行梯度爆炸时的有害一步。事实上，当梯度大小高于阈值时，即使是采取简单的*随机步骤*往往工作得几乎一样好。如果爆炸非常严重，梯度数值上为 Inf 或 Nan（无穷大或不是一个数字），则可以采取大小为 v 的随机一步，通常会离开数值不稳定的状态。截断每minibatch梯度范数不会改变单个minibatch的梯度方向。然而，许多minibatch使用范数截断梯度后的平均值不等同于截断真实梯度（使用所有的实例所形成的梯度）的范数。大导数范数的样本，和像这样的出现在同一minibatch的样本，其对最终方向的贡献将消失。不像传统minibatch梯度下降，其中真实梯度的方向是等于所有minibatch梯度的平均。换句话说，传统的随机梯度下降使用梯度的无偏估计，而与使用范数截断的梯度下降引入了经验上是有用的启发式偏置。通过逐元素截断，更新的方向与真实梯度或minibatch的梯度不再对齐，但是它仍然是一个下降方向。还有提出(**?**)（相对于隐藏单元）截断反向传播梯度，但与这些变种之间没有发布比较; 我们推测，所有这些方法表现类似。

4.11.2 引导信息流的正则化

梯度截断有助于处理爆炸的梯度，但它无助于消失的梯度。<BAD> 为了解决消失的梯度和更好地捕获长期依赖，我们讨论了在展开循环架构的计算图，沿着与弧度相关联的梯度乘积接近 1 的部分创建路径的想法。在上节4.10节中已经讨论

过，实现这一点的一种方法是使用LSTM以及其他自循环和门限机制。另一个想法是正则化或约束参数，以引导“信息流”。特别是，我们希望梯度向量 $\nabla_{\boldsymbol{h}^{(t)}} L$ 在反向传播时能维持其幅度，即使损失函数只对序列未部的输出作惩罚。形式的，我们要使

$$(\nabla_{\boldsymbol{h}^{(t)}} L)\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \tag{4.50}$$

与

$$\nabla_{\boldsymbol{h}^{(t)}} L \tag{4.51}$$

一样大。在这个目标下，(?) 提出以下正则项：

$$\Omega = \sum_t \left(\frac{\left\| (\nabla_{\boldsymbol{h}^{(t)}} L)\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \right\|}{\left\| \nabla_{\boldsymbol{h}^{(t)}} L \right\|} - 1 \right)^2 . \tag{4.52}$$

计算这一梯度的正则项可能会出现困难，但 (?) 提出可以将后向传播向量 $\nabla_{\boldsymbol{h}^{(t)}} L$ 考虑为近乎恒定的近似（为了计算正则化的目的，没有必要通过它们向后传播）。使用该正则项的实验表明，如果与标准的启发式截断（处理梯度爆炸）相结合，该正则项可以显著地增加RNN可以学习的依赖的跨度。梯度截断特别重要，因为它保持了爆炸梯度边缘的RNN动态。如果没有梯度截断，梯度爆炸将阻碍成功学习。

这种方法的一个主要弱点是，在处理数据冗余的任务时如语言模型时，它并不像LSTM一样有效。

4.12 外显记忆

智能需要知识并且可以通过学习获取知识，这已促使大型深度架构的发展。然而，存在不同种类的知识。有些知识是隐含的、潜意识的并且难以用语言表达——比如怎么行走或狗与猫的样子有什么不同。其他知识可以是明确的、可陈述的以及可以相对简单地使用词语表达——每天常识性的知识，如“猫是一种动物”，或者需要知道的实现自己目前目标非常具体的事实，如“与销售团队会议在 141 室于下午 3:00 开始“。

神经网络擅长存储隐性知识，但是他们很难记住事实。被存储在神经网络参数中之前，随机梯度下降需要多次提供相同的输入，即使如此，该输入也不会被特别精确地存储。(?) 推测这是因为神经网络缺乏**工作存储** (working memory) 系统，即类似人类为实现一些目标而明确保存和操作相关信息片段的系统。这种外显记忆组件将使我们的系统不仅能够快速“故意” 地存储和检索具体的事实，也

能利用他们循序推论。神经网络处理序列信息的需要，改变了每个步骤向网络注入输入的方式，长期以来推理能力被认为是重要的，而不是对输入做出自动的、直观的反应 (**?**) 。

为了解决这一难题，(**?**) 引入了**记忆网络**(memory network)，其中包括一组可以通过寻址机制来访问的记忆单元。记忆网络原本需要监督信号指示他们如何使用自己的记忆单元。(**?**) 引入的**神经网络图灵机**(neural Turing machine)，不需要明确的监督指示采取哪些行动而能学习从记忆单元读写任意内容，并通过使用基于内容的软注意机制 (见 (**?**) 和**??**节)，允许端到端的训练。这种软寻址机制已成为其他允许基于梯度优化的模拟算法机制的相关架构的标准 (**?????**)。

每个记忆单元可以被认为是LSTM和 GRU 中记忆单元的扩展。不同的是，网络输出一个内部状态来选择从哪个单元读取或写入，正如数字计算机读或写入到特定地址的内存访问。

产生确切整数地址的函数很难优化。为了缓解这一问题，NTM实际同时从多个记忆单元写入或读取。读取时，它们采取许多单元的加权平均值。写入时，他们对多个单元修改不同的数值。用于这些操作的系数被选择为集中在一个小数目的单元，如通过softmax函数产生它们。使用这些具有非零导数的权重允许函数控制访问存储器，从而能使用梯度下降法优化。关于这些系数的梯度指示着其中每个参数是应该增加还是减少，但梯度通常只在接收大系数的存储器地址上变大。

这些记忆单元通常扩充为包含向量，而不是由LSTM或 GRU 存储单元所存储的单个标量。增加记忆单元大小的原因有两个。原因之一是，我们已经增加了访问记忆单元的成本。我们为产生用于许多单元的系数付出计算成本，但我们预期这些系数聚集在周围小数目的单元。通过读取向量值，而不是一个标量，我们可以抵消部分成本。使用向量值的记忆单元的另一个原因是，它们允许**基于内容的寻址** (content-based addressing)，其中从一个单元读或写的权重是该单元的函数。如果我们能够生产符合某些但并非所有元素的模式，向量值单元允许我们检索一个完整向量值的记忆。这类似于人们能够通过几个歌词回忆起一首歌曲的方式。我们可以认为基于内容的读取指令是说，“检索一首副歌歌词中带有’ 我们都住在黄色潜水艇’ 的歌”。当我们要检索的对象很大时，基于内容的寻址更为有用——如果歌曲的每一个字母被存储在单独的记忆单元中，我们将无法通过这种方式找到他们。通过比较，**基于位置的寻址** (location-based addressing) 不允许引用存储器的内容。我们可以认为基于位置的读取指令是说 “检索 347 档的歌的歌词”。即使当存储单元是很小时，基于位置的寻址通常是完全合理的机制。

如果一个存储单元的内容在大多数时间步上会被复制 (不被忘记)，则它包含的信息可以在时间上向前传播，随时间向后传播的梯度也不会消失或爆炸。

外显记忆的方法在图4.18说明，其中我们看到一个与存储器耦接的 “任务神经网络”。虽然这一任务神经网络可以是前馈或循环的，但整个系统是一个循环网络。任务网络可以选择读取或写入的特定内存地址。外显记忆似乎允许模型学习普通RNN或LSTM RNN不能学习的任务。这种优点的一个原因可能是因为信息

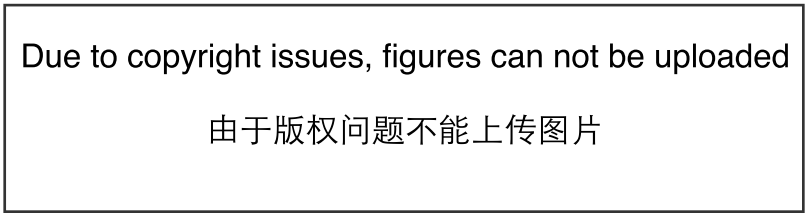


Figure 4.18: TODO

和梯度可以在非常长的持续时间内传播（分别在时间上向前或向后）。

作为存储器单元的加权平均值反向传播的替代，我们可以将存储器寻址系数解释为概率，并随机从一个单元读取 (**?**)。优化离散决策的模型需要专门的优化算法，在**??**中描述。目前为止，训练这些做离散决策的随机架构，仍比训练进行软判决的确定性算法更难。

无论是软（允许反向传播）或随机硬性的，用于选择一个地址的机制与先前在机器翻译的背景下引入的注意机制形式相同 (**?**)，这在**??**中也有讨论。甚至更早之前，注意机制的想法就被引入了神经网络，在手写生成的情况下 (**?**)，有一个被约束为通过序列只向前移动的注意机制。在机器翻译和记忆网络的情况下，每个步骤中关注的焦点可以移动到一个完全不同的地方 (相比之前的步骤)。

循环神经网络提供了将深度学习扩展到序列数据的一种方法。他们是我们的深度学习工具箱中最后一个主要的工具。现在我们的讨论将转移到如何选择和使用这些工具，以及如何将它们应用到真实世界的任务。

Chapter 5

应用

在本章中，我们介绍了如何使用深度学习来解决计算机视觉，语音识别，自然语言处理以及其它商业领域中的应用。首先我们讨论了在大多数人工智能应用中大规模神经网络的实现。接着，我们介绍了深度学习已经成功应用的几个特定的领域。尽管深度学习的一个目标是设计能够处理多种任务的算法，然而截止目前应用深度学习仍然需要一些特定的条件。比如说，计算机视觉中的任务对每一个样本都需要大量的输入特征（像素）。自然语言处理中对每一个输入特征都需要对大量的可能值（词汇）建模。

5.1 大规模深度学习

深度学习的基本思想是建立在连接机制上的：尽管机器学习模型中的单个生物性的神经元或者说是单个特征不是智能的，但是大量的神经元或者特征作用在一起往往能够表现出智能。我们必须着重强调神经元数量必须很大这个事实。相比于 20 世纪 80 年代，今日的神经网络的精度以及处理任务的复杂度都有了巨大的提升，神经元的数量是一个关键的因素。正如我们在1.2.3节中看到的一样，在过去的三四十年中，网络的规模是以指数级的速度递增的，尽管如今的人工神经网络的规模也仅仅和昆虫的神经系统差不多。

由于大规模神经网络的必要性，所以深度学习需要高性能的硬件设施和软件实现。

5.1.1 快速的 CPU 实现

传统做法中，神经网络是用单台机器的 CPU 来训练的。在今天，这种做法通常被视为是不可取的。如今，我们通常使用GPU或者许多台机器的 CPU 连接在一

起进行计算。在使用这种昂贵而又复杂的配置之前，研究者们付出了很大的努力来论证 CPU 无法承担神经网络所需要的巨大的计算量。

描述如何快速的用 CPU 实现以及超出了本书的讨论范围，但是我们在这里还是要强调通过设计一些特定的 CPU 上的操作可以大大加快效率。举个例子，在 2011 年，最好的 CPU 在训练神经网络的时候使用固定点运算比浮点数运算。通过调整固定点运算的实现方式，? 获得了相对于一个很强的浮点数运算系统 3 倍的加速。各个 CPU 都有各自不同的特性，所以有时候采用浮点数运算来实现会更快。一条重要的准则就是通过特殊设计的数值运算可以获得巨大的回报。其它的策略，除了选择固定点运算或者浮点数运算以外，包括了通过优化数据结构来避免高速缓存缺失，使用向量指令。除了当模型的规模限制了模型的表现时，机器学习的研究者们一般忽略这些实现的细节。

5.1.2 GPU 实现

许多现代的神经网络的实现是基于**图形处理器** (Graphics Processing Units, GPU)。GPU是一种特殊设计的硬件，设计的原始目的是为了处理图形应用。它的为视频游戏所设计的特性也可以使神经网络的计算受益。

视频游戏要求许多操作能够快速的并行的实现。环境和角色的模型是通过一系列的 3D 坐标结点来定位的。为了将大量的 3D 坐标转化为 2D 的显示器上的坐标，显卡必须快速的实现矩阵的乘法或者除法。之后，显卡必须并行的实现对每个像素的计算来确定每个像素点的颜色。在这两种情况下，计算是非常简单的，并且不包括 CPU 通常遇到的复杂的分支运算。比如说，同一个刚体里的每一个结点都会乘上一个相同的矩阵；也就是说，不需要进行一个 if 的判断来确定需要乘上哪一个矩阵。计算过程也是完全相互独立的，因此也能够并行操作。计算过程还包括了处理大量的缓冲和内存以及描述每一个物体的纹理（颜色）如何渲染的位图信息。这使得显卡能够拥有并行特性以及很高的内存带宽，同时也付出了一些代价，如相比于传统的 CPU 更慢的时钟速度以及更弱的处理分支运算的能力。

与上面描述的实时的图形算法相比，神经网络算法需要的是相同的硬件特性。神经网络算法涉及大量的参数，激活值，梯度值通过缓冲区，其中在每一次训练迭代中都要被完全的更新。对于传统的桌面计算机来说，缓冲区是足够大的，所以内存的带宽成为了主要的瓶颈。相比于 CPU，GPU的极高的内存带宽成为了一个显著的优势。神经网络的训练算法通常并不包含分支运算和复杂的控制指令，所以更适用于在 GPU 上训练。由于同一层神经网络往往能够被分为多个独立处理的神经元，所以神经网络能够从 GPU 的并行特性中受益匪浅。

GPU最初是专门为了图形任务而设计的。随着时间的推移，GPU也变得越发的灵活，能够处理转化结点的坐标或者计算像素颜色的任务。通过将缓冲区中的像素值作为计算的输出值，GPU也可以被用在科学计算中。? 在 GPU 上实现了

一个有两层的全连接的神经网络，然后获得了相对于基于 CPU 的基线方法的三倍的加速。不久以后，**?** 也论证了相同的技术可以被用来加速训练监督的卷积神经网络。

使用显卡来训练神经网络的热度在**通用 GPU** (general purpose GPU,) 的发布以后开始爆炸性增长。这种通用 GPU可以执行任意的代码，而并非渲染任务。NVIDIA 的 CUDA 编程语言使得我们可以用一种像 C 一样的语言实现任意的代码。由于相对简便的编程语言，强大的并行能力以及巨大的内存带宽，通用GPU是我们神经网络训练的理想平台。在它发布以后不久，这个平台就迅速被深度学习的研究者们接纳了 (**??**)。

如何在通用 GPU上写高效的代码依然是一个难题。在GPU上获得很好表现所需要的技术与 CPU 上的技术有所不同。比如说，一个好的基于 CPU 的代码通常被设计为尽可能从 cache 中读取更多的信息。然而在GPU中，许多信息并不在 cache 中，所以往往重复计算某个值会比计算它一边在从内存中读取更快。GPU代码是多线程的，各个线程之间必须协调好。比如说，如果能够把数据**级联**(coalesced) 起来，那么涉及内存的操作一般会更快。当几个线程同时需要读/写一个值的时候，像这样的级联作为一次内存操作出现。不同的GPU采用了不同的级联读/写数据的方式。通常来说，如果在 n 个线程中，线程 i 对应的是第 $i+j$ 处的内存，其中 j 是 2 的某个幂的乘积。具体的设定在不同的GPU中有所区别。GPU的另一个常见的设定是一个组中的所有线程都同时执行同一指令。这意味着在GPU上分支操作是很困难的。线程被分为一个个称作**warp**(warp) 的小组。在一个warp中的每一个线程在每一个循环中只执行同一指令，所以当同一个warp中的不同线程需要执行不同的指令的时候，需要使用串行而非并行的方式。

由于实现高效GPU代码的困难性，研究者们应该避免对每一个新的模型都实现新的GPU代码。通常来讲，人们会选择建立一个包含了高效的卷积矩阵乘法的软件库来解决这个问题，在确定模型时再从库中调用所需要的操作。比如说，机器学习的库 Pylearn2 (**?**) 包含了许多机器学习的算法通过调用 Theano (**??**) 和 cuda-convnet (**?**) 来提供了高性能运算操作。这种分解的方法简化了硬件上的编程。比如说，一个 Theano 程序可以在 CPU 或者GPU上跑，而不需要改变调用 Theano 的方式。其它的库比如说 Tensorflow(**?**)，Torch(**?**) 也提供了类似的功能。

5.1.3 大规模的分布式实现

在许多情况下，单个机器的计算资源往往是有限的。因此我们希望把训练或者推断的任务分摊到多个机器上进行。

分布式的推断是容易实现的，因为每一个输入的样本都可以在单个机器上运行。这也被成为是**数据并行**(data parallelism)。

同样的，**模型并行**(model parallelism) 也是可行的，其中多个机器共同运行一个数据点，每一个机器负责模型的一个部分。对于推断和学习，这都是可行的。

训练中，数据并行在某种程度上说更难。对于随机梯度下降的一步来说，我们可以增加minibatch的数量，但是从优化性能角度说，我们无法得到一个线性的反馈。使用多个机器并行的计算多个梯度下降是一个更好的选择。不幸的是，梯度下降的标准定义完全是一个串行的过程。在第 t 步的梯度是一个第 t-1 步所获得的参数的函数。

这个问题可以通过**异步随机梯度下降**(Asynchoronous Stochastic Gradient Descent)(**??**) 来解决。在这个方法中，几个处理器的核共用了存有参数的内存。每一个核在无锁的情况下读取了这个参数，然后计算其对应的梯度，然后在无锁的状态下更新了这个参数。这种方法减少了每一个梯度下降所获得的平均收益，因为一些核把其它的核所更新的参数（写）覆盖了。但是从总体上说还是加快了学习的过程。**?** 率先提出多机器无锁的梯度下降方法，其中参数是通过**参数服务器**(parameter server) 而非存储在共用的内存中。分布式的异步梯度下降方法依然是训练深度神经网络的基本方法，在工业界被很多的机器学习组使用 (**??**)。学术圈的深度学习研究者们通常无法负担那么大的规模的学习系统，但是一些研究者关注于如何在校园环境的相对较廉价的硬件系统中构造分布式的网络 (**?**)。

5.1.4 模型压缩

在许多商业应用的机器学习模型中，一个时间和内存开销较小的推断算法比一个时间和内存开销较小的训练算法要更为重要。对于那些不需要个性化设计的应用来说，我们只需要一次性的训练模型，然后它就可以被成千上万的用户使用。在许多情况下，相比于开发者，终端的用户可用的资源往往是很有限的。比如说，研究者们可以用巨大的计算机集群来训练一个语音识别的网络，然后发布到移动手机上。

减少推断所需要的开销中的一个关键的策略是**模型压缩**(model compression)(**?**)。模型压缩的基本思想是用一个更小的需要更少的内存和时间的模型取代替原始的耗时的模型。

为了防止过拟合，原始模型的规模很大的时候，模型压缩可以起到作用。在许多情况下，拥有最小的泛化误差的模型往往是多个独立训练而成的模型的集合。评估所有的 n 个集合的成员是困难的。有时候，当单个模型很大（比如，如果它使用dropout）时，其泛化能力也会很好。

这些巨大的模型学习某个函数 $f(\boldsymbol{x})$ 的时候，选用了超过其需要的数量的参数。训练样本的个数是有限的，所以网络的规模也是有限的。只要我们拟合了这个函数 $f(\boldsymbol{x})$，那么我们可以生成一个拥有了无穷多的训练样本的训练集，只需要作用 f 于任意生成的 $\boldsymbol{x}$。然后，我们使用这些样本来训练一个新的更小的模型。为了更加充分的利用了这个新的小的模型的表达能力，最好能够从一个类似于真实的测试数据（后面会用到）的分布中采样 $\boldsymbol{x}$。这个过程可以通过给训练样本扰动或者从由训练数据训练成的生成模型中采样来完成。

此外，人们还可以在原始训练数据上训练一个更小的模型，但是只是为了复制模型的其它特征，比如在不正确的类上的后验分布 (**??**)。

5.1.5 动态结构

一般来说，加速数据处理系统的一种策略是构造一个系统，这个系统用**动态结构**(dynamic structure) 来描述处理输入的过程。在给定一个输入的情况中，数据处理系统可以动态的决定运行神经网络系统的哪一部分。单个神经网络内部同样也存在了动态结构，来决定特征（隐含层）的哪一部分用于计算。这种神经网络中的动态结构被叫做是**条件计算**(conditional computation)(**??**)。由于模型结构许多部分可能只是跟输入的一小部分有关，只计算那些需要的特征可以起到加速的目的。

动态结构的计算是一种基础的计算机科学方法，广泛应用于软件工程项目。神经网络中动态结构的最简单的应用是决定一组神经网络（或者其它机器学习模型）模型中的哪些神经网络能够运行相应的输入。

用于在分类器中加速推断的可行策略是使用**级联**(cascade) 的分类器。当目标是检测稀有对象（或事件）的存在时，可以应用级联策略。要确定对象是否存在，我们必须使用具有高容量，运行昂贵的复杂分类器。然而，因为对象是罕见的，我们通常可以使用更少的计算来拒绝不包含对象的输入。在这些情况下，我们可以训练一系列分类器。序列中的第一分类器具有低容量，高召回率。换句话说，他们被训练确保当对象存在时，我们不会错误地拒绝输入。最终的分类器被训练成具有高精度。在测试时，我们按照顺序运行分类器来进行推理，一旦级联中的任何一个元素拒绝它，就选择放弃。总的来说，这允许我们使用高容量模型以高置信度验证对象的存在，而不是强制我们为每个样本付出完全推断的成本。有两种不同的方式可以使得级联实现高容量。一种方法是使级联中较后的成员单独具有高容量。在这种情况下，系统作为一个整体显然具有高容量，因为它的一些个体成员。还可以进行级联，其中每个单独的模型具有低容量，但是由于许多小型模型的组合，整个系统具有高容量。**?** 使用级联的增强决策树来实现适合在手持数字相机中使用的快速和鲁棒的面部检测器。它们的分类器使用滑动窗口方法来定位面部，许多窗口被检查，如果它们不包含面部则被拒绝。级联的另一个版本使用早期的模型来实现一种强制注意机制：级联的早期成员本地化对象，并且级联的后续成员在给定对象的位置的情况下执行进一步处理。例如，Google 使用两步级联从街景视图图像中转换地址编号，首先使用一个机器学习模型查找地址编号，然后使用另一个机器学习模型将其转录 (**?**)。

决策树本身是动态结构的一个例子，因为树中的每个节点决定应该为输入评估哪个子树。一个完成深度学习和动态结构联合的简单方法是训练一个决策树，其中每个节点使用神经网络做出决策 (**?**)，虽然这种方法没有实现加速推理计算的目标。

类似的，我们可以使用称为**选通器**(gater) 的神经网络来选择在给定当前输入的情况下将使用几个**专家网络**(expert network) 中的哪一个来计算输出。这个想法的第一个版本被称为**专家混合体**(mixture of experts)(**??**)，其中选通器为每个专家输出一个概率或权重（通过非线性的softmax 函数获得），并且最终输出由各个专家输出的加权组合获得。在这种情况下，使用选通器不会减少计算成本，但如果每个例子的选通器选择单个专家，我们获得一个特殊的**硬专家混合体**(hard mixture of experts) (**??**)，这可以加速推断和训练的时间。当选通器决策的数量很小，由于它不是组合的，这个策略效果会很好。但是当我们想要选择不同的单元或参数子集时，不可能使用 “软开关”，因为它需要枚举（和计算输出）所有的选通器配置。为了解决这个问题，许多工作探索了几种方法来训练组合选通器。**?** 提出使用选通器概率梯度的若干估计器，而**??** 使用强化学习技术（**策略梯度**(policy gradient)）来学习一种形式的隐藏单元的条件dropout，减少了实际的计算成本，而不会对近似的质量产生负面影响。

另一种动态结构是开关，其中隐藏单元可以根据上下文从不同单元接收输入。这种动态路由方法可以理解为**注意机制**(attention mechanism) (**?**)。到目前为止，硬开关的使用在大规模应用中还没有被证明是有效的。较为先进的方法一般采用对许多可能的输入使用加权平均，因此不能收获动态结构的所有可能的计算益处。先进的注意机制在**??**中描述。

使用动态结构化系统的一个主要障碍是由于系统针对不同输入的不同代码分支导致的并行度降低。这意味着网络中的很少的操作可以被描述为对样本的minibatch的矩阵乘法或批量卷积。我们可以写更多的专用子例程，来用不同的核对样本做卷积，或者通过不同的权重列来乘以设计矩阵的每一行。不幸的是，这些专业的子程序难以高效地实现。CPU 实现将是缓慢的，由于缺乏高速缓存的一致性。GPU的实现也将是缓慢的，因为缺乏级联的内存操作以及warp成员使用不同分支时需要串行化操作。在一些情况下，可以通过将样本分成组，这些组都采用相同的分支并且同时处理这些样本组来缓解这些问题。在离线环境中，这是最小化处理固定量的样本所需的时间的一项可接受的策略。在实时系统中，样本必须连续处理，对工作负载进行分区可能会导致负载平衡问题。例如，如果我们分配一台机器来处理级联中的第一步，另一台机器来处理级联中的最后一步，那么第一个机器将倾向于过载，最后一个机器倾向于欠载。如果每个机器被分配以实现神经决策树的不同节点，也会出现类似的问题。

5.1.6 深度网络的专用硬件实现

自从早期的神经网络研究以来，硬件设计者已经致力于可以加速神经网络算法的训练和/或推断的专用硬件实现。读者可以查看早期和更近的专业硬件深层网络的评论 (**???**)。

不同形式的专用硬件 (**??????**) 的研究已经持续了好几十年，比如**专用集成电**

路 (application-specific integrated circuit, ASIC) 的数字（基于数字的二进制表示），模拟 (**??**)（基于作为电压或电流的连续值的物理实现）和混合实现（组合数字和模拟组件）近年来更灵活的**可编程门阵列** (field programmable gated array, FPGA) 实现（其中的细节电路可以在芯片上建立后写入）也得到了长足发展。

虽然 CPU 和GPU上的软件实现通常使用 32 或 64 位的精度来表示浮点数，但是长期以来使用较低的精度在更短的时间内完成推断也是可行的 (**??????**)。这已成为近年来更迫切的问题，因为深度学习在工业产品中越来越受欢迎，并且由于更快的硬件产生的巨大影响已经通过GPU的使用得到了证明。激励当前对深度网络的专用硬件的研究的另一个因素是单个 CPU 或GPU核心的进展速度已经减慢，并且最近计算速度的改进来自于核心的并行化（无论 CPU 还是GPU)。这与 20 世纪 90 年代的情况（前面的神经网络时代）非常不同，其中神经网络的硬件实现（从开始到芯片的可用性可能需要两年）不能跟上快速进展和价格低廉的通用 CPU 的脚步。因此，在针对诸如电话的低功率设备开发新的硬件设计的时候，建立专用硬件是一种进一步推动发展的方式，旨在用于深度学习的一般公共应用 (例如，具有语音，计算机视觉或自然语言功能的设施)。

最近对基于反向传播的神经网络的低精度实现的工作 (**??**) 表明，8 和 16 位之间的精度可以足以使用或训练具有反向传播的深度神经网络。显而易见的是，在训练期间需要比在推理时更精确，并且数字的一些形式的动态定点表示可以用于减少每个数需要的存储空间。传统的固定点数限制为固定范围（其对应于浮点表示中的给定指数)。动态固定点表示在一组数字（例如一个层中的所有权重）之间共享该范围。使用固定点而不是浮点表示和每个数目使用较少的比特减少了执行乘法所需的硬件表面积，功率需求和计算时间，并且乘法是使用或训练反向传播的现代深度网络中要求最高的操作。

5.2 计算机视觉

一直以来计算机视觉就是深度学习应用的几个最活跃的研究方向。因为视觉是一个对人类以及许多动物毫不费力，但对计算机却充满挑战的任务 (**?**)。深度学习中的许多流行的基准任务有关物体识别以及字符识别。

计算机视觉是一个广阔的发展领域，其中包含了多种多样的处理图片的方式以及应用方向。计算机视觉的应用既包含复现人类视觉能力，比如识别人脸，也囊括了可视化一个新类别的物体这样的任务。举个例子，近期一个新的计算机视觉应用是从视频中可视物体振动中识别相应的声波 (**?**)。大多数计算机视觉领域的深度学习研究未曾关注过这样一个奇异的应用，它扩展了图像的范围，而不是仅仅关注于人工智能中较小的核心目标，即复制人类的能力。无论是报告图像中存在哪个物体，还是用每个对象周围的边界框注释图像，从图像转录符号序列，或标记图像中的每个像素具有它所属的对象的标识，计算机视觉的深度学习用于某种形式的对象识别或检测。因为生成模型已经是深度学习研究的指导原则，还

有大量的使用深度模型的图像合成工作。尽管图像合成通常不包括在计算机视觉内，但是能够进行图像合成的模型通常用于图像恢复，修复图像中的缺陷或从图像中移除对象这样的计算机视觉任务。

5.2.1 预处理

许多应用领域需要复杂的预处理，因为原始输入以许多深度学习架构难以表示的形式出现。计算机视觉通常只需要相对少的这种预处理。图像应该被标准化，从而使得它们的像素都在相同并且合理的范围内，比如 $[0, 1]$ 或者 $[-1, 1]$。将 [0,1] 中的图像与 [0,255] 中的图像混合通常会导致失败。将图像格式化为具有相同的比例严格上说是唯一一种必要的预处理。许多计算机视觉架构需要标准尺寸的图像，因此必须裁剪或缩放图像以适应该尺寸。然而，即使是这种重新调整并不总是必要的。一些卷积模型接受可变大小的输入并动态地调整它们的池区域的大小以保持输出大小恒定 (**?**)。其他卷积模型具有可变大小的输出，其尺寸随输入自动缩放，例如对图像中的每个像素进行去噪或标注的模型 (**?**)。

数据集增强可以被看作是预处理训练集的方式。数据集增强是减少大多数计算机视觉模型的泛化误差的一种极好的方法。在测试时间可用的一个相关想法是生成模型的相同输入的许多不同版本（例如，在稍微不同的位置处裁剪的相同图像），并且在模型的不同实例上决定模型的输出。后一个想法可以被理解为集成方法，并且有助于减少泛化误差。

其他种类的预处理被应用于训练和测试集合，目的是将每个示例置于更规范的形式，以便减少模型需要考虑的变化量。减少数据中的变化量可以减少泛化误差并减小拟合训练集所需的模型的大小。更简单的任务可以通过更小的模型来解决，而更简单的解决方案泛化能力一般更好。这种类型的预处理通常被设计为去除输入数据中的某种可变性，这对于人工设计者来说是容易描述的，并且人工设计者能够保证不受到任务影响。当使用大型数据集和大型模型训练时，这种预处理通常是不必要的，并且最好只是让模型学习哪些变异性应该保留。例如，用于分类 ImageNet 的 AlexNet 系统仅具有一个预处理步骤：减去每个像素的训练样本的平均值 (**?**)。

5.2.1.1 对比度归一化

可以为许多任务安全地移除的最明显的变化源之一是图像中的对比度量。对比度仅指图像中亮像素和暗像素之间的差异的大小。存在量化图像的对比度的许多方式。在深度学习中，对比度通常指的是图像或图像区域中的像素的标准差。假设我们有一个张量表示的图像 $\mathbf{X} \in \mathbb{R}^{r \times c \times 3}$，其中 $\mathrm{X}_{i,j,1}$ 表示第 i 行第 j 列的红色的强度，$\mathrm{X}_{i,j,2}$ 对应的是绿色，$\mathrm{X}_{i,j,3}$ 对应的是蓝色。然后整个图像的对比度可以表

示如下：

$$\sqrt{\frac{1}{3rc}\sum_{i=1}^{r}\sum_{j=1}^{c}\sum_{k=1}^{3}(\mathsf{X}_{i,j,k}-\bar{\mathbf{X}})^2} \tag{5.1}$$

其中 $\bar{\mathbf{X}}$ 是图片的平均强度，满足

$$\bar{\mathbf{X}} = \frac{1}{3rc}\sum_{i=1}^{r}\sum_{j=1}^{c}\sum_{k=1}^{3}\mathsf{X}_{i,j,k} \tag{5.2}$$

全局对比度归一化 (Global contrast normalization, GCN) 旨在通过从每个图像中减去平均值，然后重新缩放其使得其像素上的标准差等于某个常数 s 来防止图像具有变化的对比度。这种方法非常复杂，没有缩放因子可以改变零对比度图像 (所有像素都具有相等强度的图像) 的对比度。具有非常低但非零对比度的图像通常几乎没有信息内容。在这种情况下除以真实标准差通常仅能实现放大传感器噪声或压缩伪像。这种现象启发我们引入小的正的正则化参数 λ 来平衡估计的标准差。或者，我们至少可以约束分母。给定输入图像 $\mathbf{X}$，全局对比度归一化产生输出图像 $\mathbf{X}'$，定义为

$$\mathsf{X}'_{i,j,k} = s\frac{\mathsf{X}_{i,j,k}-\bar{\mathsf{X}}}{\max\{\epsilon, \sqrt{\lambda+\frac{1}{3rc}\sum_{i=1}^{r}\sum_{j=1}^{c}\sum_{k=1}^{3}(\mathsf{X}_{i,j,k}-\bar{\mathsf{X}})^2}\}} \tag{5.3}$$

由剪切到有趣对象的大图像组成的数据集不可能包含具有几乎恒定强度的任何图像。在这些情况下，通过设置 $\lambda = 0$ 来忽略小分母问题是安全的，并且在极少的情况下为了避免除以 0，通过将极小值设置为 10^{-8}。这也是**?** 在 CIFAR-10 数据集上所使用的方法。随机剪裁的小图像更可能具有几乎恒定的强度，使得正则化更有用。在处理从 CIFAR-10 数据中随机选择的补丁时，**?** 使用 $\epsilon = 0, \lambda = 10$。

尺度参数 s 通常可以设置为 1，如**?**，或选择使所有样本上每个像素的标准差接近 1，如**?**。

公式 (5.3)中的标准差仅仅是对 L^2 范数的重新缩放 (假设图像的平均值已经被移除)。我们更偏向于根据标准差而不是 L^2 范数来定义全局对比度归一化，因为标准差包括除以像素数量，因此基于标准差的全局对比度归一化允许使用与图像大小无关的相同的 s。然而，观察到 L^2 范数与标准差成比例，这符合我们的直觉。我们可以把全局对比度归一化理解成到球壳的一种映射。图5.1给了一个说明。这可能是一个有用的属性，因为神经网络往往更好地响应空间方向，而不是精确的位置。响应相同方向上的多个距离需要具有共线的权重向量但具有不同偏置的隐藏单元。这对于学习算法来说可能是困难的。<bad> 此外，许多浅层的图

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 5.1: TODO

模型往往会把多个分离的峰值表示在一条线上。全局对比度归一化采用一个样本一个方向[1]而不是不同的方向和距离来避免这些问题。

与直觉相反的是，存在被称为**sphering**(sphering) 的预处理操作，并且它不同与全局对比度归一化。sphering并不会使数据位于球形壳上，而是将主要分量重新缩放以具有相等方差，使得主成分分析使用的多变量正态分布具有球形等高线。sphering通常被称为**whitening**(whitening)。

全局对比度归一化常常不能突出我们想要突出的图像特征，例如边缘和角。如果我们有一个场景，有一个大的黑暗区域和一个大的明亮的区域（例如一个城市广场有一半的区域处于建筑物的阴影），则全局对比度归一化将确保暗区域的亮度与亮区域的亮度之间存在大的差异。然而，它不能确保暗区内的边缘突出。

这催生了**局部对比度归一化** (local contrast normalization, LCN) 。局部对比度归一化确保对比度在每个小窗口上被归一化，而不是作为整体在图像上被归一化。关于局部对比度归一化和全局对比度归一化的比较可以参考图5.2。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 5.2: TODO

局部对比度归一化的各种定义都是可行的。在所有情况下，我们可以通过减去邻近像素的平均值并除以邻近像素的标准差来修改每个像素。在一些情况下，要计算修改的像素为中心的矩形窗口中的所有像素的平均值和标准差(**?**)。在其他情况下，这是使用以要修改的像素为中心的高斯权重的加权平均和加权标准差。在彩色图像的情况下，一些策略分别处理不同的颜色通道，而其他策略组合来自不同通道的信息以使每个像素标准化 (**?**)。

[1]译者：所有样本相似的距离

局部对比度归一化通常可以通过使用可分离卷积（参见??节）来计算特征映射所需要的局部平均值和局部标准差，然后使用元素层面的减法和元素层面的除法实现不同的特征映射。

局部对比度归一化是可微分的操作，并且还可以用作应用于网络的隐藏层的非线性作用，以及应用于输入的预处理操作。

与全局对比度归一化一样，我们通常需要正则化局部对比度归一化以避免除以零。事实上，因为局部对比度归一化通常作用于较小的窗口，所以正则化更加重要。较小的窗口更可能包含彼此几乎相同的值，因此更可能具有零标准差。

5.2.2 数据集增强

如??节中所讲到的，通过增加训练集的额外副本来增加训练集的大小，从而改进分类器的泛化能力。其中训练集的额外副本并不改变其标签。对象识别是特别适合于这种形式的数据集增强的分类任务，因为标签对于许多变换是不变的，可以对输入使用许多几何变换。如前所述，分类器可以受益于随机转换旋转，以及在某些情况下，输入的翻转以增强数据集。在专门的计算机视觉应用中，更高级的变换通常用于数据集增强。这些方案包括图像中颜色的随机扰动 (?)，以及对输入的非线性几何变形 (?)。

5.3 语音识别

语音识别任务是将一段包括了自然语言发音的声音信号投影到对应的说话人的词序列上。令 $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(T)})$ 表示语音的输入向量（传统做法是通过以20ms 为一桢分割信号）。许多语音识别的系统通过特殊的手工设计的方法预处理输入信号，从而提取特征，但是某些深度学习系统 (?) 直接从原始输入中学习特征。令 $\boldsymbol{y} = (y_1, y_2, \ldots, y_N)$ 表示目标的输出序列（通常是一个词或者字符的序列）。**自动语音识别** (Automatic Speech Recognition, ASR) 任务指的是构造一个函数 f^*_{ASR}，使得它能够在给定语音序列 $\boldsymbol{X}$ 的情况下计算最有可能的 $\boldsymbol{y}$ 序列：

$$f^*_{\text{ASR}}(\boldsymbol{X}) = \arg\max_{\boldsymbol{y}} P^*(\boldsymbol{y}|\mathbf{X} = \boldsymbol{X}) \tag{5.4}$$

其中 P^* 是给定输入值 $\boldsymbol{X}$ 时对应目标 $\boldsymbol{y}$ 的条件分布。

从 20 世纪 80 年代直到 2009-2012 年，最先进的语音识别系统是**隐马尔可夫模型** (Hidden Markov Models, HMMs) 和**高斯混合模型** (Gaussian Mixture Models, GMMs) 的结合。GMMs对语音特征和**音位**(phoneme) 之间的关系建模 (?)，HMMs对音位序列建模。GMMs-HMMs模型将语音信号视作由如下过程生成：首先，一个HMMs生成了一个音位的序列以及离散的音位子状态（比如每一个音位的开始，中间，结尾），然后GMMs把每一个离散的状态转化为一个简短的

声音信号。尽管直到最近GMMs-HMMs一直在自动语音识别中占据主导地位，语音识别仍然是神经网络所成功应用的第一个领域。从 20 世纪 80 年代末期到 90 年代初期，许多的语音识别系统使用了神经网络(??????)。在那段时间，基于神经网络的自动语音识别的表现和GMMs-HMMs系统的表现差不多。比如说，? 在 TIMIT 数据集 (?)（有 39 个区分的音位）上达到了 26% 的音位错误率，这个结果优于或者是可比于基于HMMs的结果。从那时起，TIMIT 成为了音位识别的一个基准数据集，在语音识别中的作用就和 MNIST 在图像中的作用差不多。然而，由于语音识别软件系统中的复杂的工程因素以及在基于GMMs-HMMs的系统中所已经付出的巨大努力，工业界并没有转向神经网络。结果，直到 21 世纪 00 年代末期，学术界和工业界的研究者们更多的是用神经网络为GMMs-HMMs系统学习一些额外的特征。

之后，随着更大的更深的模型，更大的数据集的出现，通过使用神经网络代替GMMs来实现将语音的特征转化为音位（或者音位的子状态）可以大大的提高识别的精度。从 2009 年开始，语音识别的研究者们将一种无监督学习的深度学习方法应用于语音识别。这种深度学习方法基于训练一个被称作是受限玻耳兹曼机的无向概率模型，从而对输入数据建模。受限玻耳兹曼机将会在第三部分中被描述。为了完成语音识别任务，无监督的提前训练被用来构造一个深度前馈网路，这个神经网络是通过训练受限玻耳兹曼机来初始化的。这些网络的输入是从一个固定规格的滑动窗（以当前桢为中心）的谱特征中抽取，预测了当前桢的条件概率。训练一个这样的神经网络能够可以显著提高在 TIMIT 数据集上的识别率 (??)，并将音位级别的错误率从 26% 降到了 20.7%。关于这个模型成功原因的详细分析可以参考?。对于基础语音识别的扩展包括了添加自适应的说话人相关的特征 (?) 的方法，可以进一步的降低错误率。紧接着的工作是将结构从音位识别（TIMIT 所主要关注的）转向了大规模词汇语音识别 (?)，这不仅包含了识别音位，还包括了识别大规模词汇的序列。语音识别上的深度神经网络从最初的使用受限玻耳兹曼机进行提前训练发展到了使用修正线性单元和dropout等较为先进的技术 (??)。从那时开始，工业界的几个语音研究组开始寻求与学术圈的研究者之间的合作。? 描述了这些合作所带来的突破性进展，这些技术现在在移动手机端广泛应用。

随后，当他们使用了越来越大的带标签的数据集，加入了各种初始化，训练方法以及调试深度神经网络的结构之后，他们发现这种无监督的提前训练方式是没有必要的，或者说不能带来任何显著的改进。

用语音识别中词错误率来衡量，在语音识别性能上的这些突破是史无前例的(大约 30% 的提高)，在这之前的长达十年左右的时间基于GMMs-HMMs的系统的传统技术已经停滞不前了，尽管数据集的规模是随时间增长的（见? 的图 2.4)。这也导致了语音识别领域快速的转向深度学习的研究。在大约的两年时间内，工业界的大多数的语音识别产品都包含了深度学习，这种成功也激发了自动语音识别领域对深度学习算法和结构的一波新的研究浪潮，并且影响至今。

其中的一个创新点是卷积神经网络的应用 (**?**)。卷积神经网络 在时间和频率维度复用了权重，改进了之前的仅对时间使用重复权值的时延神经网络。这种新的二维的卷积模型并不是将输入的频谱当作一个长的向量，而是当成是一个图像，其中一个轴对应着时间，另一个轴对应的是谱分量的频率。

另一个重要的至今仍然活跃的推动，是完全抛弃了HMMs的**端到端的**(end-to-end)深度学习语音识别系统。这个领域的第一个主要的突破是 (**?**)，其中训练了一个深度的长短期记忆的循环神经网络 (见4.10节)，使用了桢－音位排列的最大后验估计推断，比如**?**，以及 CTC 框架 (**??**)。一个深度循环神经网络 (**?**) 在每一步都有状态变量，has state variables from several layers at each time step, giving the unfolded graph two kinds of depth: ordinary depth due to a stack of layers, and depth due to time unfolding 这个工作把 TIMIT 数据集上音位的错误率降到了记录的新低 17.7%。关于应用于其它领域的深度循环神经网络的变种可以参考**??**。

另一个端到端的深度学习语音识别的最新方法是让系统学习如何利用**语音**(phonetic) 层级的信息排列**声学**(acoustic) 层级的信息 (**??**)。

5.4 自然语言处理

自然语言处理(Natural Language Processing) 让计算机使用人类语言，例如英语或法语。计算机程序通常读取和发出专门的语言，让简单的程序能够高效和明确地解析。而自然的语言通常是模糊的，并且会违背形式的描述。自然语言处理 中的应用如机器翻译，学习者必须用一种人类语言读取句子并用另一种人类语言发出等同的句子。许多NLP应用程序基于语言模型，语言模型定义一个关于自然语言中的字，字符或字节序列的概率分布。

与本章讨论的其他应用一样，非常通用的神经网络技术可以成功地应用于自然语言处理。然而，为了实现卓越的性能并扩展到大型应用程序，一些领域特定的策略也很重要。为了构建自然语言的有效模型，通常必须使用专门处理序列数据的技术。在很多情况下，我们将自然语言视为一系列词，而不是单个字符或字节序列。因为可能的词总数非常大，基于词的语言模型必须在极高维度和稀疏的离散空间上操作。为使这种空间上的模型在计算和统计意义上都高效，研究者已经开发了几种策略。

5.4.1 n-gram

语言模型(language model) 定义了自然语言中的标记序列的概率分布。根据模型的设计，标记可以是词、字符、甚至是字节。标记总是离散的实体。最早成功的语言模型基于固定长度序列的标记模型，称为n-gram。一个n-gram是一个包含 n 个标记的序列。

基于n-gram的模型定义给定前 $n-1$ 个标记后的第 n 个标记的条件概率。该模型使用这些条件分布的乘积来定义较长序列的概率分布：

$$P(x_1, \dots, x_\tau) = P(x_1, \dots, x_{n-1}) \prod_{t=n}^{\tau} P(x_t \mid x_{t-n+1}, \dots, x_{t-1}). \tag{5.5}$$

这个分解可以由概率的链式法证明。初始序列 $P(x_1, \dots, x_{n-1})$ 的概率分布可以通过带有较小 n 值的不同模型来建模。

训练n-gram模型是简单的，因为最大似然估计可以简单地计算每个可能的n-gram在训练集中出现的次数来计算。几十年来，基于n-gram的模型都是统计语言模型的核心模块 (**???**)。

对于小的 n 值，模型有特定的名称：$n=1$ 称为**一元语法**(unigram)，$n=2$ 称为**二元语法**(bigram) 及 $n=3$ 称为**三元语法**(trigram)。这些名称源于相应数字的拉丁前缀和希腊后缀 “-gram”，表示所写的东西。

通常我们同时训练n-gram模型和 $n-1$ gram 模型。这使得它很容易计算概率：

$$P(x_t \mid x_{t-n+1}, \dots, x_{t-1}) = \frac{P_n(x_{t-n+1}, \dots, x_t)}{P_{n-1}(x_{t-n+1}, \dots, x_{t-1})} \tag{5.6}$$

简单地查找两个存储的概率就能计算。为了在 P_n 中精确地再现推断，我们训练 P_{n-1} 时必须省略每个序列最后的字符。

举个例子，我们演示三元模型如何计算句子 “THE DOG RAN AWAY.” 的概率。句子的第一个词不能通过上述条件概率的公式计算，因为句子的开头没有上下文。取而代之，在句子的开头我们必须使用词的边缘概率。因此我们计算 $P_3(\texttt{THE DOG RAN})$。最后，可以使用条件分布 $P(\texttt{AWAY} \mid \texttt{DOG RAN})$（典型情况）来预测最后一个词。将这与式(5.6)放在一起，我们得到：

$$P(\texttt{THE DOG RAN AWAY}) = P_3(\texttt{THE DOG RAN}) P_3(\texttt{DOG RAN AWAY}) / P_2(\texttt{DOG RAN}). \tag{5.7}$$

n-gram模型最大似然的基本限制是，在许多情况下从训练集计数估计的 P_n 很可能为零，即使元组 $(x_{t-n+1}, \dots, x_t)$ 可能出现在测试集中。这可能会导致两种不同的灾难性后果。当 P_{n-1} 为零时，该比率是未定义的，因此模型甚至不能产生意义的输出。当 P_{n-1} 非零而 P_n 为零时，测试样本的对数似然为 $-\infty$。为避免这种灾难性的后果，大多数n-gram模型采用某种形式的**平滑**(smoothing)。平滑技术将概率质量从观察到的元组转移到类似的未观察到的元组。见**?** 的综述和实验比较。其中一种基本技术基于向所有可能的下一个符号值添加非零概率质量。这个方法可以被证明是，计数参数具有均匀或Dirichlet先验的贝叶斯推断。另一个非常流行的想法是包含高阶和低阶n-gram模型的混合模型，其中高阶模型提供更多的容量，而低阶模型尽可能地避免零计数。如果上下文 $x_{t-n+k}, \dots, x_{t-1}$ 的频率太小而不能使用高阶模型，**回退方法** (back-off methods) 就查找低阶n-gram 。

更正式地说，它们通过使用上下文 $x_{t-n+k}, \ldots, x_{t-1}$ 来估计 x_t 上的分布，并增加 k 直到找到足够可靠的估计。有 $|\mathbb{V}|^n$ 可能的n-gram，而且 $|\mathbb{V}|$ 通常很大。

经典的n-gram模型特别容易引起维数灾难。即使有大量训练数据和适当的 n，大多数n-gram也不会出现在训练集中。经典n-gram模型的一种观点是执行最近邻查询。换句话说，它可以被视为局部非参数预测器，类似于 k-最近邻。这些极端局部预测器面临的统计问题在**??**节中描述。语言模型的问题甚至比普通模型更严重，因为任何两个不同的词在one-hot向量空间中具有彼此相同的距离。因此，难以大量利用来自任意 "邻居" 的信息——只有重复相同上下文的训练样本对局部泛化有用。为了克服这些问题，语言模型必须能够在一个词和其他语义相似的词之间共享知识。为了提高n-gram模型的统计效率，**基于类的语言模型** (class-based language model) (**???**) 引入词类别的概念，然后属于同一类别的词共享词之间的统计强度。这个想法使用聚类算法，基于它们与其他词同时出现的频率，将该组词分成集群或类。随后，模型可以在条件竖杠的右侧使用词类 ID 而不是单个词 ID。混合（或回退）词模型和类模型的复合模型也是可能的。尽管词类提供了在序列之间泛化的方式，但其中一些词被相同类的另一个替换，导致该表示丢失了很多信息。

5.4.2 神经语言模型

神经语言模型 (Neural Language Model, NLM) 是一类设计用来克服维数灾难的语言模型，它使用词的分布式表示对自然语言序列建模 (**?**)。不同于基于类n-gram模型，神经语言模型在识别两个相似的词的基础上，而不丧失将每个词编码为彼此不同的能力。神经语言模型共享一个词（及其上下文）和其他类似词(和上下文之间）的统计强度。模型为每个词学习的分布式表示，允许模型处理具有类似共同特征的词来实现这种共享。例如，如果词 `dog` 和词 `cat` 映射到具有许多属性的表示，则包含词 `cat` 的句子可以告知模型对包含词 `dog` 的句子做出预测，反之亦然。因为这样的属性很多，所以存在许多泛化的方式，可以将信息从每个训练语句传递到指数数量的语义相关语句。维数灾难需要模型泛化到相对句子长度是指数多的句子。该模型通过将每个训练句子与指数数量的类似句子相关联来克服这个问题。

我们有时将这些词表示称为**词嵌入**(word embedding)。在这个解释下，我们将原始符号视为维度等于词表大小的空间中的点。词表示将这些点嵌入到较低维的特征空间中。在原始空间中，每个词由一个one-hot向量表示，因此每对词彼此之间的欧氏距离都是 $\sqrt{2}$。在嵌入空间中，经常出现在类似上下文（或共享由模型学习的一些 "特征" 的任何词对）中的词彼此接近。这通常导致具有相似含义的词变得邻近。图5.3放大了学到的词嵌入空间的特定区域，我们可以看到语义上相似的词如何映射到彼此接近的表示。

也有其他领域的神经网络定义嵌入。例如，卷积网络的隐藏层提供 "图像嵌

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 5.3: TODO

入”。通常NLP从业者对嵌入的这个想法更感兴趣，因为自然语言最初不在于实值向量空间。隐藏层在表示数据的方式上提供了更质变的戏剧性变化。

使用分布式表示来改进自然语言处理模型的基本思想不用局限于神经网络。它还可以用于图模型，其中分布式表示基于多个隐变量形式。

5.4.3 高维输出

在许多自然语言应用中，我们经常希望我们的模型产生词（而不是字符）作为输出的基本单位。对于大词汇表，由于词汇量很大，在词的选择上表示输出分布计算上可能是昂贵的。在许多应用中，$\mathbb{V}$ 包含数十万词。表示这种分布的朴素方法是应用一个仿射变换，将隐藏表示转换到输出空间，然后应用softmax函数。假设我们的词汇表 $\mathbb{V}$ 大小为 $|\mathbb{V}|$。因为其输出维数为 $|\mathbb{V}|$，描述该仿射变换线性分量的权重矩阵非常大。这增加了表示该矩阵的高存储成本，以及乘以它的高计算成本。因为softmax在所有 $|\mathbb{V}|$ 输出之间归一化，所以在训练时以及测试时执行全矩阵乘法是必要的——我们不能仅计算与正确输出的权重向量的点积。因此，输出层的高计算成本在训练期间（计算似然性及其梯度）和测试期间（计算所有或所选词的概率）同时出现。对于专门的损失函数，可以有效地计算梯度 (**?**)，但是应用于传统softmax输出层的标准交叉熵损失时会出现了许多困难。

假设 $\boldsymbol{h}$ 是用于预测输出概率 $\hat{\boldsymbol{y}}$ 的顶部隐藏层。如果我们用学到的权重 $\boldsymbol{W}$ 和学到的偏置$\boldsymbol{b}$ 来参数化从 $\boldsymbol{h}$ 到 $\hat{\boldsymbol{y}}$ 的变换，则仿射softmax输出层执行以下计算：

$$a_i = b_i + \sum_j W_{ij} h_j \quad \forall i \in \{1, \ldots, |\mathbb{V}|\}, \tag{5.8}$$

$$\hat{y}_i = \frac{e^{a_i}}{\sum_{i'=1}^{|\mathbb{V}|} e^{a_{i'}}}. \tag{5.9}$$

如果 $\boldsymbol{h}$ 包含 n_h 个元素，则上述操作复杂度是 $O(|\mathbb{V}|n_h)$。n_h 为数千和 $|\mathbb{V}|$ 数十万的情况下，这个操作占据了神经语言模型的大多数计算。

5.4.3.1 使用短列表

第一个神经语言模型(**??**) 通过将词汇量限制为 10,000 或 20,000 来减轻大词汇表上softmax的高成本。**?** 和 **?** 在这种方法的基础上建立新的方式，将词汇表 $\mathbb{V}$ 分为最常见词汇（由神经网络处理）的**短列表**(shortlist) $\mathbb{L}$ 和较稀有词汇的尾列表 $\mathbb{T} = \mathbb{V} \backslash \mathbb{L}$（由$n$-gram模型处理)。为了组合这两个预测，神经网络还必须预测在上下文 C 之后出现的词位于尾部列表的概率。可以添加额外的sigmoid输出单元估计 $P(i \in \mathbb{T} \mid C)$ 实现这个预测。额外输出则可以用来估计 $\mathbb{V}$ 中所有词的概率分布，如下：

$$\begin{aligned} P(y = i \mid C) =& 1_{i \in \mathbb{L}} P(y = i \mid C, i \in \mathbb{L})(1 - P(i \in \mathbb{T} \mid C)) \\ &+ 1_{i \in \mathbb{T}} P(y = i \mid C, i \in \mathbb{T}) P(i \in \mathbb{T} \mid C) \end{aligned} \tag{5.10}$$

其中 $P(y = i \mid C, i \in \mathbb{L})$ 由神经语言模型提供 $P(y = i \mid C, i \in \mathbb{T})$ 由n-gram模型提供。稍作修改，这种方法也可以在神经语言模型模型的softmax层中使用额外的输出值，而不是单独的sigmoid单元。

短列表方法的一个明显的缺点是，神经语言模型模型的潜在泛化优势仅限于最常用的词，这大概是最没用的。这个缺点激发了处理高维输出替代方法的探索，如下所述。

5.4.3.2 分层 Softmax

减少大词汇表 $\mathbb{V}$ 上高维输出层计算负担的经典方法 (**?**) 是分层地分解概率。无需与 $|\mathbb{V}|$ 成比例数量（并且也与隐藏单元数量 n_h 成比例）的计算，$|\mathbb{V}|$ 因子可以降低到 $\log |\mathbb{V}|$ 一样低。**?** 和**?** 将这种因子分解方法引入神经语言模型中。

我们可以认为这种层次结构是先建立词的类别，然后是词类别的类别，然后是词类别的类别的类别，等等这些嵌套类别构成一棵树，其叶上是词。在平衡树中，树的深度为 $\log |\mathbb{V}|$。选择一个词的概率是由路径（从树根到包含该词叶子的路径）上的每个节点通向该词分支概率的乘积给出。图**??**是一个简单的例子。**?** 也描述了使用多个路径来识别单个词的方法，以便更好地建模具有多个含义的词。计算词的概率则涉及在导向该词所有路径上的求和。

为了预测树的每个节点所需的条件概率，我们通常在树的每个节点处使用逻辑回归模型，并且为所有这些模型提供与输入相同的上下文 C。因为正确的输出被编码在训练集中，我们可以使用监督学习训练逻辑回归模型。通常使用标准交叉熵损失，对应于最大化正确判断序列的对数似然。

因为可以高效地计算输出对数似然（低至 $\log |\mathbb{V}|$ 而不是 $|\mathbb{V}|$），所以也可以高效地计算梯度。不仅包括关于输出参数的梯度，而且还包括关于隐藏层激活的梯度。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 5.4: TODO

优化树结构最小化期望的计算数量是可能的，但通常不实际。给定词的相对频率，信息理论的工具可以指定如何选择最佳二进制代码。为此，我们可以构造树，使得与词相关联的位数量近似等于该词频率的对数。然而在实践中，节省计算通常不值得努力，因为输出概率的计算仅是神经语言模型中总计算的一部分。例如，假设有 l 个全连接的宽度为 n_h 的隐藏层。令 n_b 是识别一个词所需比特数的加权平均值，其加权由这些词的频率给出。在这个例子中，计算隐藏激活所需的操作数增长为 $O(ln_h^2)$，而输出计算增长为 $O(n_h n_b)$。只要 $n_b \leq ln_h$，我们可以通过收缩 n_h 比收缩 n_b 减少更多的计算量。事实上，n_b 通常很小。因为词汇表的大小很少超过一百万而 $\log_2(10^6) \approx 20$，所以可以将 n_b 减小到大约 20，但 n_h 通常大得多，大约为 10^3 或更大。我们可以定义深度为 2 和分支因子为 $\sqrt{|\mathbb{T}|}$ 的树，而不用仔细优化分支因子为 2 的树。这样的树对应于简单定义一组互斥的词类。基于深度为 2 的树的简单方法可以获得层级策略大部分的计算益处。

一个仍然有点开放的问题是如何最好地定义这些词类，或者如何定义一般的词层次结构。早期工作使用现有的层次结构 (?) ，但也可以理想地与神经语言模型联合学习层次结构。学习层次结构很困难。对数似然的精确优化似乎难以解决，因为词层次的选择是离散的，不适于基于梯度的优化。然而，可以使用离散优化来近似地最优化词类的分割。

分层softmax的一个重要优点是，它在训练期间和测试期间（如果在测试时我们想计算特定词的概率）都带来了计算的好处。

当然即使使用分层softmax，计算所有 $|\mathbb{T}|$ 词的概率仍将是昂贵的。另一个重要的操作是在给定上下文中选择最可能的词。不幸的是，树结构不能为这个问题提供有效和精确的解决方案。

缺点是在实践中，分层softmax倾向于更差的测试结果（相对基于采样的方法），我们将在下面描述。这可能是词类选择得不好。

5.4.3.3 重要采样

加速神经语言模型训练的一种方式是避免明确计算所有词（未出现在下一位置）对梯度的贡献。每个不正确的词在此模型下应该具有低概率。枚举所有这些单词

计算成本可能会很高。相反，可以仅采样词的子集。使用式(5.8)中引入的符号，梯度可以写成如下形式：

$$\frac{\partial \log P(y \mid C)}{\partial \theta} = \frac{\partial \log \text{softmax}_y(\boldsymbol{a})}{\partial \theta} \tag{5.11}$$

$$= \frac{\partial}{\partial \theta} \log \frac{e^{a_y}}{\sum_i e^{a_i}} \tag{5.12}$$

$$= \frac{\partial}{\partial \theta}(a_y - \log \sum_i e^{a_i}) \tag{5.13}$$

$$= \frac{\partial a_y}{\partial \theta} - \sum_i P(y = i \mid C) \frac{\partial a_i}{\partial \theta} \tag{5.14}$$

其中 $\boldsymbol{a}$ 是presoftmax激活（或分数）向量，每个词对应一个元素。第一项是**正相** (positive phase) 项推动 a_y 向上，而第二项是**负相** (negative phase) 项，对于所有 i 以权重 $P(i \mid C)$ 推动 a_i 向下。由于负相项是期望值，我们可以用蒙特卡罗采样来估计。然而，这将需要从模型本身采样。从模型中抽样需要对词汇表中所有的 i 计算 $P(i \mid C)$，这正是我们试图避免的。

我们可以从另一个分布中抽样，而不是从模型中抽样，称为**提议分布**(proposal distribution)（记为 q），并通过适当的权重来校正从错误分布抽样引入的偏差(**??**)。这是一种称为**重要采样**(Importance Sampling) 的更通用技术的应用，我们将在9.2节中更详细地描述。不幸的是，即使精确重要采样也不一定有效，因为需要计算权重 p_i/q_i，其中的 $p_i = P(i \mid C)$ 只能在计算所有得分 a_i 后才能计算。这个应用采取的解决方案称为有偏重要采样，其中重要性权重被归一化加和为 1。当对负词 n_i 进行采样时，相关联的梯度被加权为：

$$w_i = \frac{p_{n_i}/q_{n_i}}{\sum_{j=1}^{N} p_{n_j}/q_{n_j}}. \tag{5.15}$$

这些权重用于对来自 q 的 m 个负样本给出适当的重要性，以形成负相估计对梯度的贡献：

$$\sum_{i=1}^{|\mathbb{V}|} P(i \mid C) \frac{\partial a_i}{\partial \theta} \approx \frac{1}{m} \sum_{i=1}^{m} w_i \frac{\partial a_{n_i}}{\partial \theta}. \tag{5.16}$$

一元语法或二元语法分布与提议分布q 工作得一样好。很容易从数据估计这种分布的参数。在估计参数之后，也可以非常高效地从这样的分布采样。

重要采样(Importance Sampling) 不仅可以加速具有较大softmax输出的模型。更一般地，它可以加速具有大稀疏输出层的训练，其中输出是稀疏向量而不是 n 选 1。其中一个例子是词袋。词袋具有稀疏向量 $\boldsymbol{v}$，其中 v_i 表示词汇表中的词 i

存不存在文档中。或者，v_i 可以指示词 i 出现的次数。由于各种原因，训练产生这种稀疏向量的机器学习模型可能是昂贵的。在学习的早期，模型可能不会真的使输出真正稀疏。此外，将输出的每个元素与目标的每个元素进行比较，可能是描述用于训练的损失函数可能最自然的方式。这意味着稀疏输出并不一定能带来计算上的好处，因为模型可以选择使大多数输出非零，并且所有这些非零值需要与相应的训练目标进行比较（即使训练目标是零）。**?** 证明可以使用重要采样加速这种模型。高效算法最小化“正词”（在目标中非零的那些词）和相等数量的“负词”的重构损失。随机选择负词，如使用启发式抽样更可能被误解的词。该启发式过采样引入的偏差则可以使用重要性权重校正。

在所有这些情况下，输出层梯度估计的计算复杂度被减少为与负样本数量成比例，而不是与输出向量的大小成比例。

5.4.3.4 噪声对比估计和排名损失

为减少训练大词汇表的神经语言模型计算成本，研究者也提出了其他基于抽样的方法。早期的例子是 **?** 提出的排名损失，将神经语言模型每个词的输出视为一个分数，并试图使正确词的分数 a_y 比其他词 a_i 排名更高。提出的排名损失则是

$$L = \sum_i \max(0, 1 - a_y + a_i). \tag{5.17}$$

如果观察到词的分数 a_y 远超过负词的分数 a_i（相差大于 1），则第 i 项梯度为零。这个判据的一个问题是它不提供估计的条件概率，条件概率在很多应用中是有用的，包括语音识别和文本生成（包括诸如翻译的条件文本生成任务）。

最近用于神经语言模型的训练目标是噪声对比估计，将在**??**节中介绍。这种方法已成功应用于神经语言模型(**??**)。

5.4.4 结合n-gram和神经语言模型

n-gram模型相对神经网络的主要优点是n-gram模型具有更高的模型容量（通过存储非常多的元组的频率），并且处理样本只需非常少的计算量（通过查找只匹配当前上下文的几个元组）。如果我们使用哈希表或树来访问计数，那么用于n-gram的计算量几乎与容量无关。相比之下，将神经网络的参数数目加倍通常也大致加倍计算时间。<BAD> 例外包括避免每次计算时使用所有参数的模型。嵌入层每次只索引单个嵌入，所以我们可以增加词汇量，而不增加每个样本的计算时间。一些其他模型，例如平铺卷积网络，可以在减少参数共享程度的同时添加参数以保持相同的计算量。然而，基于矩阵乘法的典型神经网络层需要与参数数量成比例的计算量。

因此，增加容量的一种简单方法是将两种方法结合，由神经语言模型和n-gram语言模型组成集成 (**??**)。

对于任何集成，如果集成成员产生独立的错误，这种技术可以减少测试误差。集成的领域提供了许多方法来组合集成成员的预测，包括统一加权和在验证集上选择权重。**?** 扩展了集成，不仅包括两个模型，而包括大量模型。也可以将神经网络与最大熵模型配对并联合训练 (**?**)。该方法可以被视为训练具有一组额外输入的神经网络，额外输入直接连接到输出并且不连接到模型的任何其他部分。额外输入是输入上下文中特定n-gram是否存在的指示器，因此这些变量是非常高维且非常稀疏的。

模型容量的增加是巨大的（架构的新部分包含高达 $|sV|^n$ 个参数），但是处理输入所需的额外计算量是很小的（因为额外输入非常稀疏）。

5.4.5 神经机器翻译

机器翻译以一种自然语言读取句子并产生等同含义的另一种语言的句子。机器翻译系统通常涉及许多组件。在高层次，一个组件通常会提出许多候选翻译。由于语言之间的差异，这些翻译中的许多翻译将是不符合语法的。例如，许多语言在名词后放置形容词，因此直接翻译成英语时，它们会产生诸如 “apple red’’ 的短语。提议机制提出建议翻译的许多变体，理想地应包括 “red apple’’。翻译系统的第二个组成部分，语言模型，评估提议的翻译，并可以评分 “red apple’’ 比 “apple red’’ 更好。

<BAD> 最早的机器翻译神经网络探索中已经纳入了编码器和解码器的想法 (Allen 1987; Chrisman 1991; Forcada and Ñeco 1997)，而翻译中神经网络的第一个大规模有竞争力的用途是通过神经语言模型升级翻译系统的语言模型 (**??**)。之前，大多数机器翻译系统在该组件使用n-gram模型。机器翻译中基于n-gram的模型不仅包括传统的回退n-gram模型，而且包括**最大熵语言模型** (maximum entropy language models)，其中给定上下文中常见的词 affine-softmax 层预测下一个词。

传统语言模型仅仅报告自然语言句子的概率。因为机器翻译涉及给定输入句子产生输出句子，所以将自然语言模型扩展为有条件的是有意义的。如**??**节所述可以直接地扩展一个模型，该模型定义某些变量的边缘分布，以便在给定上下文 C（C 可以是单个变量或变量列表）的情况下定义该变量的条件分布。**?** 在一些统计机器翻译的基准中击败了最先进的技术，他给定源语言中短语 $\mathrm{s}_1, \mathrm{s}_2, \ldots, \mathrm{s}_k$ 后使用MLP对目标语言的短语 $\mathrm{t}_1, \mathrm{t}_2, \ldots, \mathrm{t}_k$ 进行评分。这个MLP估计 $P(\mathrm{t}_1, \mathrm{t}_2, \ldots, \mathrm{t}_k \mid \mathrm{s}_1, \mathrm{s}_2, \ldots, \mathrm{s}_k)$。这个MLP的估计替代了条件$n$-gram模型提供的估计。

基于MLP方法的缺点是需要将序列预处理为固定长度。为了使翻译更加灵活，我们希望使用允许输入长度可变和输出长度可变的模型。RNN具备这种能力。

4.2.4节描述了给定某些输入后，关于序列条件分布的RNN的几种构造方法，并且4.4节描述了当输入是序列时如何实现这种条件分布。在所有情况下，一个模型首先读取输入序列并产生概括输入序列的数据结构。我们称这个概括为“上下文”C。上下文 C 可以是向量列表，或者向量或张量。读取输入以产生 C 的模型可以是RNN(**???**) 或卷积网络(**?**)。第二模型（通常是 RNN），则读取上下文 C 并且生成目标语言的句子。在图5.5中示出了这种用于机器翻译的编码器 - 解码器框架的总体思想。

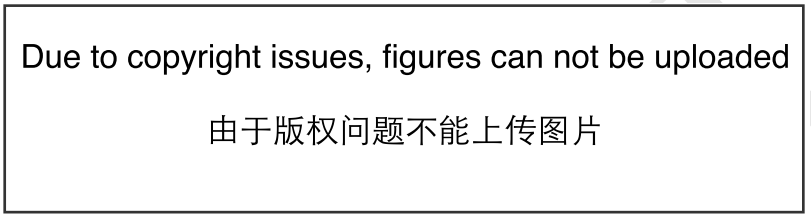


Figure 5.5: TODO

为生成以源句为条件的整句，模型必须具有表示整个源句的方式。早期模型只能表示单个词或短语。从表示学习的观点来看，具有相同含义的句子具有类似表示是有用的，无论它们是以源语言还是以目标语言书写。首先使用卷积和RNN的组合来探索该策略 (**?**)。后来的工作介绍了使用RNN对所提议的翻译进行打分 (**?**) 或生成翻译句子 (**?**)。**?** 将这些模型扩展到更大的词汇表。

5.4.5.1 使用注意机制并对齐数据片段

使用固定大小的表示来概括非常长的句子（例如 60 个词）的所有语义细节是非常困难的。这可以使用足够大的 RNN，并且用足够长时间训练得很好，才能实现，如 **?** 和**?** 所表明的。然而，更高效的方法是先读取整个句子或段落（以获得正在表达的上下文和焦点），然后一次翻译一个词，每次聚焦于输入句子的不同部分来收集产生下一个输出词所需的语义细节。这正是 **?** 第一次引入的想法。图5.6中展示了注意机制，其中每个时间步关注输入序列的特定部分。

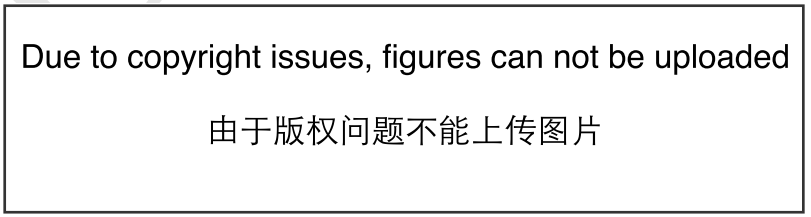


Figure 5.6: TODO

我们可以认为基于注意机制的系统有三个组件：

- 读取器 *读取*原始数据（例如源语句中的源词）并将其转换为分布式表示，其中一个特征向量与每个词的位置相关联。
- 存储器存储读取器输出的特征向量列表。这可以被理解为包含事实序列的*存储器*，而之后不必以相同的顺序从中检索，也不必访问全部。
- 最后一个程序*利用*存储器的内容来顺序地执行任务，每个时间步能聚焦于某个存储器元素的内容（或几个，具有不同权重）。

第三组件生成翻译语句。

当用一种语言书写的句子中的词与另一种语言的翻译语句中的相应词对齐时，可以使对应的词嵌入相关联。早期的工作表明，我们可以学习将一种语言中的词嵌入与另一种语言中的词嵌入相关联的翻译矩阵 (?)，与传统的基于短语表中频率计数的方法相比，可以产生较低的对齐错误率。甚至更早的工作 (?) 研究跨语言词向量。这种方法的存在很多扩展。例如，允许在更大数据集训练的更高效的跨语言对齐 (?) 。

5.4.6 历史观点

<BAD> 在反向传播的第一次探索中，? 等人提出了分布式表示符号的思想，其中符号对应于族成员的身份，而神经网络捕获族成员之间的关系，训练样本形成三元组如（Colin，Mother，Victoria）。神经网络的第一层学习每个族成员的表示。例如，Colin 的特征可能代表 Colin 所在的族树，他所在树的分支，他来自哪一代等等。我们可以将神经网络认为是将这些属性关联在一起的计算学习规则，可以获得期望预测。模型则可以进行预测，例如推断谁是 Colin 的母亲。

? 将符号嵌入的想法扩展对词的嵌入。这些嵌入使用 SVD 学习。之后，嵌入将通过神经网络学习。

自然语言处理的历史是由流行表示（对模型输入不同方式的表示）的变化标志的。在早期对符号和词建模的工作之后，神经网络在NLP上一些最早的应用 (??) 将输入表示为字符序列。

? 将焦点重新引到建模词并引入神经语言模型，能产生可解释的词嵌入。这些神经模型已经从在一小组符号上的定义表示（20 世纪 80 年代）扩展到现代应用中的数百万字（包括专有名词和拼写错误）。这种计算扩展的努力导致了5.4.3节中描述的技术发明。

最初，使用词作为语言模型的基本单元可以改进语言建模的性能 (?)。而今，新技术不断推动基于字符 (?)）和基于词的模型向前发展，最近的工作 (?) 甚至建模 Unicode 字符的单个字节。

神经语言模型背后的思想已经扩展到多个自然语言处理应用，如解析 (**???**)，词性标注，语义角色标注，分块等，有时使用共享词嵌入的单一多任务学习架构 (**??**)。

<BAD> 随着 t-SNE降维算法的发展 (**?**)，用于分析语言模型嵌入的二维可视化成为一种流行的工具，以及 Joseph Turian 在 2009 年引入的专用于可视化词嵌入的应用。

5.5 其他应用

在本节中，我们介绍深度学习一些其他类型的应用，它们与上面讨论的标准对象识别、语音识别和自然语言处理任务不同。本书的第三部分将扩大这个范围，甚至进一步扩展到仍是目前主要研究领域的任务。

5.5.1 推荐系统

信息技术部门中机器学习的主要应用之一是向潜在用户或客户推荐项目。可以分为两种主要的应用：在线广告和项目建议（通常这些建议的目的仍然是为了销售产品)。两者都依赖于预测用户和项目之间的关联，如果展示了广告或向该用户推荐了该产品，推荐系统要么预测一些行为的概率（用户购买产品或该行为的一些代替）或预期增益（其可取决于产品的价值)。目前，互联网的资金主要来自于各种形式的在线广告。经济的主要部分依靠网上购物。包括 Amazon 和 eBay 在内的公司使用机器学习（包括深度学习）推荐他们的产品。有时，项目不是实际出售的产品。如选择在社交网络新闻信息流上显示的帖子、推荐观看的电影、推荐笑话、推荐专家建议、匹配视频游戏的玩家或匹配约会的人。

通常，这种关联问题像监督学习问题一样处理：给出一些关于项目和关于用户的信息，预测感兴趣的行为（用户点击广告、输入评级、点击“喜欢”按钮、购买产品，在产品上花钱、花时间访问产品页面等)。通常这最终会归结到回归问题（预测一些条件期望值）或概率分类问题（预测一些离散事件的条件概率)。

早期推荐系统的工作依赖于作为这些预测输入的最小信息：用户 ID 和项目 ID。在这种情况下，唯一的推广方式是依赖于不同用户或不同项目的目标变量值之间的模式相似性。假设用户 1 和用户 2 都喜欢项目 A，B 和 C. 由此，我们可以推断出用户 1 和用户 2 具有类似的口味。如果用户 1 喜欢项目 D，那么这可以强烈提示用户 2 也喜欢 D。基于此原理的算法称为**协同过滤**(collaborative filtering)。非参数方法（例如基于估计偏好模式之间相似性的最近邻方法）和参数方法都可能用来解决这个问题。参数方法通常依赖于为每个用户和每个项目学习分布式表示（也称为嵌入)。目标变量的双线性预测（例如评级）是一种简单的参数方法，这种方法非常成功，通常被认为是最先进系统的组成部分。通过用户嵌入和项目嵌入之间的点积（可能需要通过仅依赖于用户 ID 或项目 ID 的常数

来校正）获得预测。令 $\hat{\boldsymbol{R}}$ 是包含我们预测的矩阵，$\boldsymbol{A}$ 矩阵行中是用户嵌入，$\boldsymbol{B}$ 矩阵列中具有项目嵌入。令 $\boldsymbol{b}$ 和 $\boldsymbol{c}$ 是分别包含针对每个用户（表示用户平常坏脾气或积极的程度）以及每个项目（表示其大体受欢迎程度）的偏置向量。因此，双线性预测如下获得：

$$\hat{R}_{u,i} = b_u + c_i + \sum_j A_{u,j} B_{j,i}. \tag{5.18}$$

通常，人们希望最小化预测评级 $\hat{R}_{u,i}$ 和实际评级 $\hat{R}_{u,i}$ 之间的平方误差。当用户嵌入和项目嵌入首次缩小到低维度（两个或三个）时，它们就可以方便地可视化，或者可以用于将用户或项目彼此进行比较（就像词嵌入）。获得这些嵌入的一种方式是对实际目标（例如评级）的矩阵 $\boldsymbol{R}$ 进行奇异值分解。这对应于将 $\boldsymbol{R} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}'$（或归一化的变体）分解为两个因子的乘积，低秩矩阵 $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}$ and $\boldsymbol{B} = \boldsymbol{V}'$。SVD的一个问题是它以任意方式处理丢失的条目，如同它们对应于目标值 0。相反，我们希望避免为缺失条目做出的预测付出任何代价。幸运的是，观察到的评级的平方误差总和也可以通过基于梯度的优化最小化。SVD和式 (5.18)中的双线性预测在Netflix奖（目的是仅基于大量匿名用户的之前评级来预测电影的评级）的竞争中表现得非常好 (**?**)。许多机器学习专家参加了 2006 年和 2009 年之间的这场比赛。它提高了使用先进机器学习的推荐系统的研究水平，并改进了推荐系统。即使简单的双线性预测或SVD本身并没有赢得比赛，但它是大多数竞争对手提出的整体模型中一个组成部分，包括胜者 (**??**)。

除了这些具有分布式表示的双线性模型之外，第一次用于协同过滤的神经网络之一是基于RBM的无向概率模型 (**?**)。RBM是赢得Netflix比赛方法的一个重要组成部分 (**??**)。神经网络社区中也已经探索了对评级矩阵进行因子分解的更高级变体 (**?**)。

然而，协同过滤系统有一个基本限制：当引入新项目或新用户时，缺乏评级历史意味着无法评估其与其他项目或用户的相似性，或者说无法评估新的用户和现有项目的联系。这被称为冷启动推荐问题。解决冷启动推荐问题的一般方式是引入单个用户和项目的额外信息。例如，该额外信息可以是用户简要信息或每个项目的特征。使用这种信息的系统被称为**基于内容的推荐系统** (content-based recommender system)。从丰富的用户特征或项目特征集到嵌入的映射可以通过深度学习架构来学习 (**??**)。

专用深度学习架构，如卷积网络已经应用于从丰富内容中提取特征，如从用于音乐推荐的音乐音轨 (**?**)。在该工作中，卷积网络将声学特征作为输入并计算相关歌曲的嵌入。该歌曲嵌入和用户嵌入之间的点积则可以预测用户是否将收听该歌曲。

5.5.1.1 探索与开发

当向用户推荐时，会产生超出普通监督学习范围的问题，并进入强化学习的领域。理论上，许多推荐问题最准确的描述是contextual bandit(**??**)。问题是，当我们使用推荐系统收集数据时，我们得到一个有偏的和不完整的用户偏好观：我们只能看到用户对推荐给他们项目的反应，而不是其他项目。此外，在某些情况下，我们可能无法获得未向其进行推荐的用户的任何信息（例如，在广告竞价中，可能是广告的建议价格低于最低价格阈值，或者没有赢得竞价，因此广告不会显示）。更重要的是，我们不知道推荐任何其他项目会产生什么结果。这就像训练一个分类器，为每个训练样本 $\boldsymbol{x}$ 挑选一个类别 $\hat{y}$（通常是基于模型最高概率的类别），无论这是否是正确的类别都只能获得反馈。显然，每个样本传达的信息少于监督的情况（其中真实标签 y 是可直接访问的）因此需要更多的样本。更糟糕的是，如果我们不够小心，即使收集越来越多的数据，我们得到的系统可能会继续挑选错误的决定，因为正确的决定最初只有很低的概率：直到学习者选择正确的决定之前都无法学习正确的决定。这类似于强化学习的情况，其中仅观察到所选动作的奖励。一般来说，强化学习会涉及许多动作和许多奖励的序列。bandit情景是强化学习的特殊情况，其中学习者仅采取单一动作并接收单个奖励。bandit问题在学习者知道哪个奖励与哪个动作相关联的时更容易。在一般的强化学习场景中，高奖励或低奖励可能是由最近的行动或很久以前的行动引起的。术语**contextual bandit**(contextual bandit) 指的是在一些输入变量可以通知决定的上下文中采取动作的情况。例如，我们至少知道用户身份，并且我们要选择一个项目。从上下文到操作的映射也称为**策略**(policy)。学习者和数据分布（现在取决于学习者的行动）之间的反馈循环是强化学习和bandit研究的中心问题。

强化学习需要权衡**探索**(exploration) 与**开发**(exploitation)。开发指的是从目前学到的最好策略采取动作，也就是我们所知的将获得高奖励的动作。**探索**(exploration) 是指采取行动以获得更多的训练数据。如果我们知道给定上下文 $\boldsymbol{x}$，动作 a 给予我们 1 的奖励，但我们不知道这是否是最好的奖励。我们可能想利用我们目前的策略，并继续采取行动 a 相对肯定地获得 1 的奖励。然而，我们也可能想通过尝试动作 a' 来探索。我们不知道尝试动作 a' 会发生什么。我们希望得到 2 的奖励，但有获得 0 奖励的风险。无论如何，我们至少获得了一些知识。

探索(exploration) 可以以许多方式实现，从覆盖可能动作的整个空间的随机动作到基于模型的方法（基于预期回报和模型对该回报不确定性的量来计算动作的选择）。

许多因素决定了我们喜欢探索或开发的程度。最突出的因素之一是我们感兴趣的时间尺度。如果代理只有短暂的时间积累奖励，那么我们喜欢更多的开发。如果代理有很长时间积累奖励，那么我们开始更多的探索，以便使用更多的知识来更有效地规划未来的动作。

监督学习在探索或开发之间没有权衡，因为监督信号总是指定哪个输出对于

每个输入是正确的。我们总是知道标签是最好的输出，没有必要尝试不同的输出来确定是否优于模型当前的输出。

除了权衡探索和开发之外，强化学习背景下出现的另一个困难是难以评估和比较不同的策略。强化学习包括学习者和环境之间的相互作用。这个反馈回路意味着使用固定的测试集输入来评估学习者的表现不是直接的。策略本身确定将看到哪些输入。**?** 提出了评估contextual bandit的技术。

5.5.2 知识表示、推理和回答

深度学习方法在语言模型、机器翻译和自然语言处理方面非常成功，因为使用符号 (**?**) 和词嵌入 (**??**)。这些嵌入表示关于单个词或概念的语义知识。研究前沿是为短语或词和事实之间的关系开发嵌入。搜索引擎已经使用机器学习来实现这一目的，但是要改进这些更高级的表示还有许多工作要做。

5.5.2.1 知识、联系和回答

一个有趣的研究方向是确定如何训练分布式表示才能捕获两个实体之间的**关系**(relation)。

数学中，二元关系是一组有序的对象对。集合中的对具有这种关系，而那些不在集合中的对没有。例如，我们可以在实体集 $\{1, 2, 3\}$ 上定义关系 "小于" 来定义有序对的集合 $\mathbb{S} = \{(1, 2), (1, 3), (2, 3)\}$。一旦这个关系被定义，我们可以像动词一样使用它。因为 $(1, 2) \in \mathbb{S}$，我们说 1 小于 2。因为 $(2, 1) \notin \mathbb{S}$，我们不能说 2 小于 1。当然，彼此相关的实体不必是数字。我们可以定义关系 `is_a_type_of` 包含如（狗，哺乳动物）的元组。

在AI的背景下，我们将关系看作句法上简单且高度结构化的语言。关系起到动词的作用，而关系的两个参数发挥着主体和客体的作用。这些句子是一个三元组标记的形式：

$$(\text{subject}, \text{verb}, \text{object}) \tag{5.19}$$

其值是

$$(\text{entity}_i, \text{relation}_j, \text{entity}_k). \tag{5.20}$$

我们还可以定义**属性**(attribute)，类似于关系的概念，但只需要一个参数：

$$(\text{entity}_i, \text{attribute}_j). \tag{5.21}$$

例如，我们可以定义 `has_fur` 属性，并将其应用于像狗这样的实体。

许多应用中需要表示关系和推理。我们应该如何在神经网络中做到这一点？

机器学习模型当然需要训练数据。我们可以推断非结构化自然语言组成的训练数据集中实体之间的关系。也有明确定义关系的结构化数据库。这些数据库的共同结构是关系型数据库，它存储这种相同类型的信息，虽然没有格式化为三元标记的句子。当数据库旨在将日常生活中常识或关于应用领域的专业知识传达给人工智能系统时，我们将这种数据库称为知识图谱。知识图谱包括一般的像 Freebase、OpenCyc、WordNet、Wikibase,[2]等等，和专业的知识库，如 GeneOntology[3]。实体和关系的表示可以将知识图谱中的每个三元组作为训练样本来学习，并且最大化捕获他们联合分布的训练目标 (?)。

除了训练数据，我们还需定义训练的模型族。一种常见的方法是将神经语言模型扩展到模型实体和关系。神经语言模型学习提供每个词分布式表示的向量。他们还通过学习这些向量的函数来学习词之间的相互作用，例如哪些词可能出现在词序列之后。我们可以学习每个关系的嵌入向量将这种方法扩展到实体和关系。事实上，建模语言和通过关系编码的建模知识的联系非常接近，研究人员可以*同时*使用知识图谱*和*自然语言句子训练这样的实体表示 (???)，或组合来自多个关系型数据库的数据 (?)。有许多可能与这种模型相关联的特定参数化。早期关于学习实体间关系的工作 (?) 假定高度受限的参数形式（“线性关系嵌入”），通常对关系使用与实体形式不同的表示。例如，? 和? 用向量表示实体而矩阵表示关系，其思想是关系在实体上像运算符。或者，关系可以被认为是任何其他实体 (?)，允许我们关于关系作声明，但是更灵活的是将它们结合在一起并建模联合分布的机制。

这种模型的实际短期应用是**链接预测**(link prediction)：预测知识图谱中缺失的弧。这是基于旧事实推广新事实的一种形式。目前存在的大多数知识图谱都是通过人力劳动构建的，这往往使知识图谱缺失许多并且可能是大多数真正的关系。请查看?、? 和? 中这样应用的例子。

很难评估链接预测任务上模型的性能，因为我们的数据集只有正样本（已知是真实的事实）。如果模型提出了不在数据集中的事实，我们不确定模型是犯了错误还是发现了一个新的以前未知的事实。度量基于测试模型如何将已知真实事实的留存集合与不太可能为真的其他事实相比较，因此有些不精确。构造感兴趣的负样本（可能为假的事实）的常见方式是从真实事实开始，并创建该事实的损坏版本，例如用随机选择的不同实体替换关系中的一个实体。<BAD> 通用的测试精度（10% 度量）计算模型在该事实的所有损坏版本的前 10% 中选择“正确”事实的次数。

知识图谱和分布式表示的另一个应用是**词义消歧**(word-sense disambiguation) (??)，这个任务决定在某些语境中哪个词的意义是恰当。

[2]分别可以在如下网址获取：freebase.com，cyc.com/opencyc，wordnet.princeton.edu，wikiba.se

[3]geneontology.org

最后，知识的关系结合一个推理过程和对自然语言的理解可以让我们建立一个一般的问答系统。一般的问答系统必须能处理输入信息并记住重要的事实，并以之后能检索和推理的方式组织。这仍然是一个困难的开放性问题，只能在受限的“玩具”环境下解决。目前，记住和检索特定声明性事实的最佳方法是使用显式记忆机制，如4.12节所述。记忆网络首先被提出来解决一个玩具问答任务 (**?**)。**?** 提出了一种扩展，使用 GRU循环网络将输入读入存储器并且在给定存储器的内容后产生回答。

深度学习已经应用于其它许多应用（除了这里描述的应用以外），并且肯定会在此之后应用于更多的场景。不可能一下子描述全面覆盖主题的所有略微相似的应用。在本文写作之时，这项调查尽可能提供了有代表性的样本。

第二部分介绍了涉及深度学习的现代实践，包括所有非常成功的方法。一般而言，这些方法使用代价函数的梯度寻找模型（近似于某些所期望的函数）的参数。当具有足够的训练数据时，这种方法是非常强大的。我们现在转到第三部分，开始进入研究领域，旨在使用较少的训练数据或执行更多样的任务，其中的挑战更困难也远远没有解决（相比目前为止所描述的情况）。

Chapter 6

线性因子模型

许多深度学习的研究前沿涉及到了建立输入的概率模型 $p_{\text{model}}(\boldsymbol{x})$。原则上说，给定任何其他变量的情况下，这样的模型可以使用概率推理来预测其环境中的任何变量。这些模型中的许多还具有隐变量 $\boldsymbol{h}$，其中 $p_{\text{model}}(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{h}} p_{\text{model}}(\boldsymbol{x}|\boldsymbol{h})$。这些隐变量提供了表示数据的另一种方式。基于隐变量的分布式表示可以有很多优点，这些我们在深度前馈网路和循环神经网络中已经发现。

在本章中，我们描述了一些带有隐变量的最简单的概率模型：**线性因子模型**(linear factor model)。这些模型有时被用来构建混合块模型 (**???**) 或者更大的深度概率模型 (**?**)。他们还展示了构建生成模型所需的许多基本方法，更先进的深层模型也将在此基础上进一步扩展。

线性因子模型通过使用随机线性解码器函数来定义，该函数通过对 $\boldsymbol{h}$ 的线性变换以及添加噪声来生成 $\boldsymbol{x}$。

这些模型很有趣，因为它们使得我们能够发现一些拥有简单联合分布的解释性因子。线性解码器的简单性使得这些模型（含有隐含变量的模型）能够被广泛研究。

线性因子模型描述如下的数据生成过程。首先，我们从一个分布中抽取解释性因子

$$\boldsymbol{h} \sim p(\boldsymbol{h}) \tag{6.1}$$

其中 $p(\boldsymbol{h})$ 是一个可分解的分布，满足 $p(\boldsymbol{h}) = \prod_i p(h_i)$，所以很容易从中采样。接下来，在给定因子的情况下，我们对实值的可观察变量进行抽样

$$\boldsymbol{x} = \boldsymbol{W}\boldsymbol{h} + \boldsymbol{b} + \text{noise} \tag{6.2}$$

其中噪声通常是对角化的（在维度上是独立的）且服从高斯分布。在图6.1有具体说明。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 6.1: TODO

6.1 概率 PCA和因子分析

概率 PCA(probabilistic PCA)，因子分析和其他线性因子模型是上述等式((6.1),(6.2)) 的特殊情况，并且仅在对观测到 $\boldsymbol{x}$ 之前的噪声分布和隐变量 $\boldsymbol{h}$ 的先验的选择上不同。

因子分析(factor analysis)(**??**) 中，隐变量的先验是一个方差为单位矩阵的高斯分布

$$\boldsymbol{h} \sim \mathcal{N}(\boldsymbol{h}; \boldsymbol{0}, \boldsymbol{I}) \tag{6.3}$$

同时，假定观察值 x_i 在给定 $\boldsymbol{h}$ 的条件下是条件独立的。具体的说，噪声可以被假设为是从对角的协方差矩阵的高斯分布中抽出的，协方差矩阵为 $\boldsymbol{\psi} = \text{diag}(\boldsymbol{\sigma}^2)$，其中 $\boldsymbol{\sigma}^2 = [\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2]^\top$ 表示一个向量。

因此，隐变量的作用是捕获不同观测变量 x_i 之间的依赖关系。实际上，可以容易地看出 $\boldsymbol{x}$ 是多变量正态分布，并满足

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}; \boldsymbol{b}, \boldsymbol{W}\boldsymbol{W}^\top + \boldsymbol{\psi}) \tag{6.4}$$

为了将PCA引入到概率框架中，我们可以对因子分析模型进行轻微修改，使条件方差 σ_i^2 等于同一个值。在这种情况下，$\boldsymbol{x}$ 的协方差是 $\boldsymbol{W}\boldsymbol{W}^\top + \sigma^2\boldsymbol{I}$, 这里的 σ^2 是一个标量。由此可以得到条件分布，如下:

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}; \boldsymbol{b}, \boldsymbol{W}\boldsymbol{W}^\top + \sigma^2\boldsymbol{I}) \tag{6.5}$$

或者等价于

$$\boldsymbol{x} = \boldsymbol{W}\boldsymbol{h} + \boldsymbol{b} + \sigma\boldsymbol{z} \tag{6.6}$$

其中 $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{z}; \boldsymbol{0}, \boldsymbol{I})$ 是高斯噪音。之后 (**?**) 提出了一种迭代的EM算法来估计参数 $\boldsymbol{W}$ 和 σ^2。

概率 PCA模型利用了这样一种观察的现象：除了一些小的重构误差 σ^2，数据中的大多数变化可以由隐变量 $\boldsymbol{h}$ 描述。通过**?** 的研究可以发现，当 $\sigma \to 0$ 的时

候，概率 PCA 等价于PCA。在这种情况下，给定 $\boldsymbol{x}$ 情况下的 $\boldsymbol{h}$ 的条件期望等于将 $\boldsymbol{x}-\boldsymbol{b}$ 投影到 $\boldsymbol{W}$ 的 d 列的生成空间，与PCA一样。

当 $\sigma \to 0$ 的时候，概率 PCA 所定义的密度函数在 $\boldsymbol{W}$ 的 d 维列生成空间的坐标周围非常尖锐。如果数据实际上没有集中在超平面附近，这会导致模型为数据分配非常低的可能性。

6.2 独立分量分析

独立分量分析 (independent component analysis, ICA) 是最古老的表示学习之一算法 (**???????**)。它是一种建模线性因子的方法，旨在分离观察到的信号，并转换为许多基础信号的叠加。这些信号是完全独立的，而不是仅仅彼此不相关[1]。

许多不同的具体方法被称为独立分量分析。与我们本书中描述的其他的生成模型最相似的独立分量分析变种是训练完全参数化的生成模型(**?**)。隐含因子 $\boldsymbol{h}$ 的先验 $p(\boldsymbol{h})$，必须由用户给出并固定。接着模型确定性的生成 $\boldsymbol{x}=\boldsymbol{W}\boldsymbol{h}$。我们可以通过非线性变化（使用公式(**??**)）来确定 $p(\boldsymbol{x})$。然后通过一般的方法比如最大似然估计进行学习。

这种方法的动机是，通过选择一个独立的 $p(\boldsymbol{h})$，我们可以尽可能恢复接近独立的隐含因子。这是一种常用的方法，它并不是用来捕捉高级别的抽象的因果因子，而是恢复已经混合在一起的低级别信号。在该设置中，每个训练样本对应一个时刻，每个 x_i 是一个传感器的对混合信号的观察值，并且每个 h_i 是单个原始信号的一个估计。例如，我们可能有 n 个人同时说话。如果我们具有放置在不同位置的 n 个不同的麦克风，则独立分量分析可以检测每个麦克风的音量变化，并且分离信号，使得每个 h_i 仅包含一个人清楚地说话。这通常用于脑电图的神经科学，一种用于记录源自大脑的电信号的技术。放置在对象的头部上的许多电极传感器用于测量来自身体的许多电信号。实验者通常仅对来自大脑的信号感兴趣，但是来自受试者的心脏和眼睛的信号强到足以混淆在受试者的头皮处进行的测量。信号到达电极，并且混合在一起，因此独立分量分析是必要的，以分离源于心脏与源于大脑的信号，并且将不同脑区域中的信号彼此分离。

如前所述，独立分量分析存在许多变种。一些版本在 $\boldsymbol{x}$ 的生成中添加一些噪声，而不是使用确定性的解码器。大多数不使用最大似然估计准则，而是旨在使 $\boldsymbol{h}=\boldsymbol{W}^{-1}\boldsymbol{x}$ 的元素彼此独立。许多准则能够达成这个目标。方程(**??**)需要用到 $\boldsymbol{W}$ 的行列式，这可能是昂贵且数值不稳定的操作。独立分量分析的一些变种通过将 $\boldsymbol{W}$ 约束为正交来避免这个有问题的操作。

独立分量分析的所有变种要求 $p(\boldsymbol{h})$ 是非高斯的。这是因为如果 $p(\boldsymbol{h})$ 是具有高斯分量的独立先验，则 $\boldsymbol{W}$ 是不可识别的。对于许多 $\boldsymbol{W}$ 值，我们可以在 $p(\boldsymbol{x})$ 上获得相同的分布。这与其他线性因子模型有很大的区别，例如概率 PCA和因子

[1] **??**节讨论了不相关变量和独立变量之间的差异。

分析，通常要求 $p(\boldsymbol{h})$ 是高斯的，以便使模型上的许多操作具有闭式解。在用户明确指定分布的最大似然估计方法中，一个典型的选择是使用 $p(h_i) = \frac{d}{dh_i}\sigma(h_i)$。这些非高斯分布的典型选择在 0 附近具有比高斯分布更高的峰值，因此我们也可以看到独立分量分析经常在学习稀疏特征时使用。

按照我们的说法独立分量分析的许多变种不是生成模型。在本书中，生成模型可以直接表示 $p(\boldsymbol{x})$，也可以认为是从 $p(\boldsymbol{x})$ 中抽取样本。独立分量分析的许多变种仅知道如何在 $\boldsymbol{x}$ 和 $\boldsymbol{h}$ 之间变换，但没有任何表示 $p(\boldsymbol{h})$ 的方式，因此也无法确定 $p(\boldsymbol{x})$。例如，许多独立分量分析变量旨在增加 $\boldsymbol{h} = \boldsymbol{W}^{-1}\boldsymbol{x}$ 的样本峰度，因为高峰度使得 $p(\boldsymbol{h})$ 是非高斯的，但这是在没有显式表示 $p(\boldsymbol{h})$ 的情况下完成的。这是为什么独立分量分析被常用作分离信号的分析工具，而不是用于生成数据或估计其密度。

正如PCA可以推广到第7章中描述的非线性自动编码器，独立分量分析可以推广到非线性生成模型，其中我们使用非线性函数 f 来生成观测数据。关于非线性独立分量分析最初的工作可以参考**?**，它和集成学习的成功结合可以参见**??**。独立分量分析的另一个非线性扩展是**非线性独立分量估计** (nonlinear independent components estimation, NICE) 方法 (**?**)，这个方法堆叠了一系列可逆变换（编码器），从而能够高效的计算每个变换的雅可比行列式。这使得我们能够精确地计算似然，并且像独立分量分析一样，NICE尝试将数据变换到具有可分解的边缘分布的空间。由于非线性编码器的使用相比于NICE，这种方法更可能成功。因为编码器和一个于其（编码器）完美逆作用的解码器相关联，所以可以直接从模型生成样本（通过首先从 $p(\boldsymbol{h})$ 采样，然后应用解码器）。

独立分量分析的另一个应用是通过在组内鼓励统计依赖关系在组之间抑制依赖关系来学习一组特征。当相关单元的组不重叠时，这被称为**独立子空间分析**(independent subspace analysis)。还可以向每个隐藏单元分配空间坐标，并且空间上相邻的单元形成一定程度的重叠。这能够鼓励相邻的单元学习类似的特征。当应用于自然图像时，这种拓扑独立分量分析方法学习 Gabor 滤波器，从而使得相邻特征具有相似的定向，位置或频率。在每个区域内出现类似 Gabor 函数的许多不同相位偏移，使得在小区域上的合并产生了平移不变性。

6.3 慢特征分析

慢特征分析(slow feature analysis) 是使用来自时间信号的信息来学习不变特征的线性因子模型(**?**)。

SFA的想法源于所谓的**慢原则**(slowness principle)。基本思想是，与场景中的描述作用的物体相比，场景的重要特性通常变化得非常缓慢。例如，在计算机视觉中，单个像素值可以非常快速地改变。如果斑马从左到右移动穿过图像并且它的条纹穿过对应的像素时，该像素将迅速从黑色变为白色，并再次恢复。通过比

较，指示斑马是否在图像中的特征将根本不改变，并且描述斑马的位置的特征将缓慢地改变。因此，我们可能希望规范我们的模型，从而能够学习到随时间变化缓慢的特征。

慢原则早于SFA，并已被应用于各种模型 (**????**)。一般来说，我们可以将慢原则应用于可以使用梯度下降训练的任何可微分模型。为了引入慢原则，我们可以通过向代价函数添加以下项

$$\lambda \sum_t L(f(\boldsymbol{x}^{(t+1)}), f(\boldsymbol{x}^{(t)})) \tag{6.7}$$

其中 λ 是确定慢度正则化的强度的超参数项，t 是实例的时间序列的索引，f 是特征提取器，L 是测量 $f(\boldsymbol{x}^{(t)})$ 和 $f(\boldsymbol{x}^{(t+1)})$ 之间的距离的损失函数。L 的一个常见选择是平均误差平方。

SFA是慢原则中特别有效的应用。由于它被应用于线性特征提取器，并且可以以通过闭式解训练，所以他是高效的。像ICA的一些变体一样，SFA本身不是生成模型，只是在输入空间和特征空间之间定义了线性映射，但是没有定义特征空间的先验，因此输入空间中不存在 $p(\boldsymbol{x})$ 分布。

SFA算法 (**?**) 包括将 $f(\boldsymbol{x}; \theta)$ 定义为线性变换，并求解满足如下约束

$$\mathbb{E}_t f(\boldsymbol{x}^{(t)})_i = 0 \tag{6.8}$$

以及

$$\mathbb{E}_t[f(\boldsymbol{x}^{(t)})_i^2] = 1 \tag{6.9}$$

的优化问题

$$\min_{\boldsymbol{\theta}} \mathbb{E}_t (f(\boldsymbol{x}^{(t+1)})_i - f(\boldsymbol{x}^{(t)})_i)^2 \tag{6.10}$$

学习特征具有零均值的约束对于使问题具有唯一解是必要的; 否则我们可以向所有特征值添加一个常数，并获得具有慢度目标的相等值的不同解。特征具有单位方差的约束对于防止所有特征趋近于 0 的病态问题是必要的。与PCA类似，SFA特征是有序的，其中学习第一特征是最慢的。要学习多个特征，我们还必须添加约束

$$\forall i < j, \quad \mathbb{E}_t[f(\boldsymbol{x}^{(t)})_i f(\boldsymbol{x}^{(t)})_j] = 0. \tag{6.11}$$

这要求学习的特征必须彼此线性去相关。没有这个约束，所有学习的特征将简单地捕获一个最慢的信号。可以想象使用其他机制，如最小化重建误差，迫使特征多样化。但是由于SFA特征的线性，这种去相关机制只能得到一种简单的解。SFA问题可以通过线性代数软件获得闭式解。

在运行SFA之前，SFA通常通过对 $\boldsymbol{x}$ 使用非线性的基扩充来学习非线性特征。例如，通常用 $\boldsymbol{x}$ 的二次基扩充来代替原来的 $\boldsymbol{x}$，得到一个包含所有 $x_i x_j$ 的向量。然后可以通过重复学习线性SFA特征提取器，对其输出应用非线性基扩展，然后在该扩展之上学习另一个线性 SFA 特征提取器，来组合线性SFA模块以学习深非线性慢特征提取器。Linear SFA modules may then be composed to learn deep nonlinear slow feature extractors by repeatedly learning a linear SFA feature extractor, applying a nonlinear basis expansion to its output, and then learning another linear SFA feature extractor on top of that expansion.

当训练在自然场景的视频的小空间补丁的时候，使用二次基扩展的SFA能够学习到与 V1 皮层中那些复杂细胞类似的许多特征 (**?**)。当训练在 3-D 计算机呈现环境内的随机运动的视频时，深度SFA模型能够学习到与大鼠脑中用于导航的神经元学到的类似的特征 (**?**)。因此从生物学角度上说SFA是一个合理的有依据的模型。

SFA的一个主要优点是，即使在深度非线性条件下，它依然能够在理论上预测SFA能够学习哪些特征。为了做出这样的理论预测，必须知道关于配置空间的环境的动态 (例如，在 3D 渲染环境中的随机运动的情况下，理论分析出位置，相机的速度的概率分布)。已知潜在因子如何改变的情况下，我们能够理论分析解决表达这些因子的最佳函数。在实践中，基于模拟数据的实验上，使用深度SFA似乎能够恢复了理论预测的函数。相比之下其他学习算法中的成本函数高度依赖于特定像素值，使得更难以确定模型将学习什么特征。

深度SFA也已经被用于学习用在对象识别和姿态估计的特征 (**?**)。到目前为止，慢度原理尚未成为任何最先进的技术应用的基础。什么因素限制了其性能也有待研究。我们推测，或许慢度先验是太过强势，并且，最好添加这样一个先验使得当前步骤到下一步的预测更加容易，而不是加一个先验使得特征应该近似为一个常数。对象的位置是一个有用的特征，无论对象的速度是高还是低。但慢度原则鼓励模型忽略具有高速度的对象的位置。

6.4 稀疏编码

稀疏编码(sparse coding)(**?**) 是一个线性因子模型，已作为无监督特征学习和特征提取机制进行了大量研究。严格地说，术语“稀疏编码”是指在该模型中推断 $\boldsymbol{h}$ 的值的过程，而“稀疏建模”是指设计和学习模型的过程，但是通常这两个概念都可以用术语“稀疏编码”描述。

像其它的线性因子模型一样，它使用了线性的解码器加上噪音的方式获得一个 $\boldsymbol{x}$ 的重构，就像公式 (6.2)描述的一样。更具体的说，稀疏编码模型通常假设线

性因子有一个各向同性的精度为 β 的高斯噪音：

$$p(\boldsymbol{x}|\boldsymbol{h}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{W}\boldsymbol{h} + \boldsymbol{b}, \frac{1}{\beta}\boldsymbol{I}) \tag{6.12}$$

关于 $p(\boldsymbol{h})$ 分布通常选择一个峰值很尖锐且接近 0 的分布 (**?**)。常见的选择包括了可分解的 Laplace，Cauchy 或者可分解的 Student-t 分布。例如，以稀疏惩罚系数 λ 为参数的 Laplace 先验可以表示为

$$p(h_i) = \text{Laplace}(h_i; 0, \frac{2}{\lambda}) = \frac{\lambda}{4}\text{e}^{-\frac{1}{2}|h_i|} \tag{6.13}$$

相应的，Student-t 分布可以表示为

$$p(h_i) \propto \frac{1}{(1 + \frac{h_i^2}{\nu})^{\frac{\nu+1}{2}}} \tag{6.14}$$

使用最大似然估计的方法来训练稀疏编码模型是不可行的。相反，训练在编码数据和训练解码器之间交替，以在给定编码的情况下更好地重建数据。稍后在10.3节中，这种方法将被进一步证明为解决似然最大化问题的一种通用的近似方法。

对于诸如PCA的模型，我们已经看到使用了预测 $\boldsymbol{h}$ 的参数化的编码器函数，并且该函数仅包括乘以权重矩阵。在稀疏编码中的编码器不是参数化的。相反，编码器是一个优化算法，在这个优化问题中，我们寻找单个最可能的编码值：

$$\boldsymbol{h}^* = f(\boldsymbol{x}) = \arg\max_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{x}) \tag{6.15}$$

结合方程 (6.13)和 (6.12)，我们得到如下的优化问题：

$$\arg\max_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{x}) \tag{6.16}$$

$$= \arg\max_{\boldsymbol{h}} \log p(\boldsymbol{h}|\boldsymbol{x}) \tag{6.17}$$

$$= \arg\min_{\boldsymbol{h}} \lambda\|\boldsymbol{h}\|_1 + \beta\|\boldsymbol{x} - \boldsymbol{W}\boldsymbol{h}\|_2^2 \tag{6.18}$$

其中，我们扔掉了与 $\boldsymbol{h}$ 无关的项，除以一个正的伸缩因子来简化表达。

由于在 $\boldsymbol{h}$ 上施加 L^1 范数，这个过程将产生稀疏的 $\boldsymbol{h}^*$（见3.1.2节）。

为了训练模型而不仅仅是进行推理，我们交替迭代关于 $\boldsymbol{h}$ 和 $\boldsymbol{W}$ 的最小化过程。在本文中，我们将 β 视为超参数。通常将其设置为 1，因为其在此优化问题中的作用与 λ 类似，没有必要使用两个超参数。原则上，我们还可以将 β 作为模

型的参数，并学习它。我们在这里已经放弃了一些不依赖于 $\boldsymbol{h}$ 但依赖于 β 的项。要学习 β，必须包含这些项，否则 β 将退化为 0。

不是所有的稀疏编码方法都显式地构建了 $p(\boldsymbol{h})$ 和 $p(\boldsymbol{x}|\boldsymbol{h})$。通常我们只是对学习一个带有激活值的特征的字典感兴趣，当使用这个推断过程时，这个激活值通常为 0。

如果我们从 Laplace 先验中采样 $\boldsymbol{h}$，$\boldsymbol{h}$ 的元素实际上为零是一个零概率事件。生成模型本身并不稀疏，只有特征提取器是。**?** 描述了不同模型族中的近似推理，和 spike and slab稀疏编码模型，其中先验的样本通常包含许多 0。

与非参数化的编码器结合的稀疏编码方法原则上可以比任何特定的参数化的编码器更好地最小化重构误差和对数先验的组合。另一个优点是编码器没有泛化误差。参数化的编码器必须泛化地学习如何将 $\boldsymbol{x}$ 映射到 $\boldsymbol{h}$。对于与训练数据差异很大的异常的 $\boldsymbol{x}$，所学习的参数化的编码器可能无法找到对应精确重建的 $\boldsymbol{h}$ 或稀疏的编码。对于稀疏编码模型的绝大多数形式，推理问题是凸的，优化过程将总是找到最优值（除非出现简并的情况，例如重复的权重向量）。显然，稀疏和重构成本仍然可以在不熟悉的点上升，但这是归因于解码器权重中的泛化误差，而不是编码器中的泛化误差。当稀疏编码用作分类器的特征提取器时，而不是使用参数化的函数来预测时，基于优化的稀疏编码模型的编码过程中泛化误差的减小可导致更好的泛化能力。**?** 证明了在对象识别任务中稀疏编码特征比基于参数化的编码器（如线性 sigmoid 自动编码器）的特征拥有更好的泛化能力。受他们的工作启发，**?** 表明稀疏编码的变体在其中极少标签（每类 20 个或更少标签）的情况中比其他特征提取器拥有更好的泛化能力。

非参数编码器的主要缺点是在给定 $\boldsymbol{x}$ 的情况下需要大量的时间来计算 $\boldsymbol{h}$，因为非参数方法需要运行迭代算法。在第7章中讲到的参数化的自动编码器方法仅使用固定数量的层，通常只有一层。另一个缺点是它不直接通过非参数编码器进行反向传播，这使得我们很难采用先使用无监督方式预训练稀疏编码模型然后使用有监督方式对其进行微调的方法。允许近似导数的稀疏编码模型的修改版本确实存在但未被广泛使用 (**?**)。

像其他线性因子模型一样，稀疏编码经常产生糟糕的样本，如图6.2所示。即使当模型能够很好地重构数据并为分类器提供有用的特征时，也会发生这种情况。这种现象原因是每个单独的特征可以很好地被学习到，但是隐含结点可分解的先验会导致模型包括每个生成的样本中的所有特征的随机子集。the factorial prior on the hidden code results in the model including random subsets of all of the features in each generated sample. 这促使人们在深度模型中的最深层以及一些复杂成熟的浅层模型上施加一个不可分解的分布。

这促进了更深层模型的发展，可以在最深层上施加 non-factorial 分布，以及开发更复杂的浅层模型。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 6.2: TODO

6.5 PCA的流形解释

线性因子模型，包括了PCA和因子分析，可以被解释为学习一个流形(**?**)。我们可以将概率 PCA定义为高概率的薄饼状区域，即高斯分布，沿着某些轴非常窄，就像薄饼沿着其垂直轴非常平坦，但沿着其他轴是细长的，正如薄饼在其水平轴方向是很宽的一样。图 |||c||| 解释了这种现象。PCA可以理解为将该薄饼与更高维空间中的线性流形对准。这种解释不仅适用于传统PCA，而且适用于学习矩阵 $\boldsymbol{W}$ 和 $\boldsymbol{V}$ 的任何线性自动编码器，其目的是使重构的 $\boldsymbol{x}$ 尽可能接近于原始的 $\boldsymbol{x}$。

编码器表示为

$$\boldsymbol{h} = f(\boldsymbol{x}) = \boldsymbol{W}^{\top}(\boldsymbol{x} - \boldsymbol{\mu}) \tag{6.19}$$

编码器计算 h 的低维表示。从自动编码器的角度来看，解码器负责重构计算

$$\hat{\boldsymbol{x}} = g(\boldsymbol{h}) = \boldsymbol{b} + \boldsymbol{V}\boldsymbol{h} \tag{6.20}$$

选择适当的线性编码器和解码器来最小化重建误差

$$\mathbb{E}[\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2] \tag{6.21}$$

其中 $\boldsymbol{V} = \boldsymbol{W}$，$\boldsymbol{\mu} = \boldsymbol{b} = \mathbb{E}[\boldsymbol{x}]$，$\boldsymbol{W}$ 的列形成一组正交基，这组基生成的子空间于主特征向量对应的协方差矩阵相同。

$$\boldsymbol{C} = \mathbb{E}[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^{\top}] \tag{6.22}$$

在PCA中，$\boldsymbol{W}$ 的列是按照对应的特征值（其全部是实数和非负数）的大小排序所对应的特征向量。

我们还可以发现 $\boldsymbol{C}$ 的特征值 λ_i 对应了 $\boldsymbol{x}$ 在特征向量 $\boldsymbol{v}^{(i)}$ 方向上的方差。如果 $\boldsymbol{x} \in \mathbb{R}^D$，$\boldsymbol{h} \in \mathbb{R}^d$ 并且满足 $d < D$，则（给定上述的 $\boldsymbol{\mu}, \boldsymbol{b}, \boldsymbol{V}, \boldsymbol{W}$ 的情况下）最佳的重构误差是

$$\min \mathbb{E}[\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2] = \sum_{i=d+1}^{D} \lambda_i \tag{6.23}$$

因此，如果协方差矩阵的秩为 d，则特征值 λ_{d+1} 到 λ_D 都为 0，并且重构误差为 0。

此外，还可以证明上述解可以通过在正交矩阵 $\boldsymbol{W}$ 下最大化 $\boldsymbol{h}$ 元素的方差而不是最小化重建误差来获得。

某种程度上说，线性因子模型是最简单的生成模型和学习数据表示的最简单模型。许多模型比如线性分类器和线性回归模型可以扩展到深度前馈网路，这些线性因子模型可以扩展到执行的是相同任务但具有更强大和更灵活的模型族，比如自动编码器网络和深概率模型。

Chapter 7

自动编码器

自动编码器(autoencoder) 是神经网络的一种，经过训练后能尝试将输入复制到输出。**自动编码器**(autoencoder) 内部有一个隐含层 $\boldsymbol{h}$，可以产生**编码**(code) 来表示输入。该网络可以看作由两部分组成：一个编码器函数 $\boldsymbol{h} = f(\boldsymbol{x})$ 和一个生成重构的解码器$\boldsymbol{r} = g(\boldsymbol{h})$。图7.1展示了这种架构。如果一个自动编码器学会简单地设置 $g(f(\boldsymbol{x})) = \boldsymbol{x}$, 那么这个自动编码器不会很有用。相反，自动编码器应该被设计成不能学会完美地复制。这通常需要强加一些约束，使自动编码器只能近似地复制，并只能复制类似训练数据的输入。这些约束强制模型划定输入数据不同方面的主次顺序，因此它往往能学习到数据的有用特性。

现代自动编码器将编码器和解码器的思想推广，将其中的确定函数推广为随机映射 $p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$ 和 $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$。

数十年间，自动编码器的想法一直是神经网络历史景象的一部分 (**???**)。传统上，自动编码器被用于降维或特征学习。近年来，自动编码器与隐变量模型理论的联系将自动编码器带到了生成建模的前沿，我们将在**??**章看到更多细节。自动编码器可以被看作是前馈网络的一种特殊情况，并且可以使用完全相同的技术进行训练，通常使用minibatch梯度下降法（基于反向传播计算的梯度）。不像一般的前馈网络，自动编码器也可以使用**再循环**(recirculation) 训练 (**?**)，这是一种基于比较原始输入和重构输入激活的学习算法。相比反向传播算法，再循环算法从生物学上看似更有道理，但很少用于机器学习。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 7.1: TODO

7.1 欠完备自动编码器

将输入复制到输出听起来没什么用，但我们通常不关心解码器的输出。相反，我们希望通过训练自动编码器对输入进行复制的任务使 $\boldsymbol{h}$ 获得有用的特性。

从自动编码器获得有用特征的一种方法是限制 $\boldsymbol{h}$ 的维度比 $\boldsymbol{x}$ 小，这种编码维度小于输入维度的自动编码器称为**欠完备**(undercomplete)自动编码器。学习欠完备的表示将强制自动编码器捕捉训练数据中最显著的特征。

学习过程可以简单地描述为最小化一个损失函数

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))), \tag{7.1}$$

其中 L 是一个损失函数，衡量 $g(f(\boldsymbol{x}))$ 与 $\boldsymbol{x}$ 的不相似性，如均方误差。

当解码器是线性的且 L 是均方误差，欠完备的自动编码器会学习出与PCA相同生成子空间。在这种情况下，自动编码器学到了训练数据的主元子空间（执行复制任务的副效用）。

因此拥有非线性编码函数 f 和非线性解码器函数 g 的自动编码器能够学习出更强大的PCA非线性推广。不幸的是，如果编码器和解码器被赋予太大的容量，自动编码器会执行复制任务而捕捉不到有关数据分布的有用信息。从理论上说，我们可以想象只有一维编码的自动编码器，但具有一个非常强大的非线性编码器，能够将每个训练数据 $\boldsymbol{x}^{(i)}$ 表示为编码 i。解码器可以学习将这些整数索引映射回特定训练样本的值。这种特定情形不会在实践中发生，但它清楚地说明，如果自动编码器的容量太大，那训练来执行复制任务的自动编码器可能无法学习到数据集的任何有用信息。

7.2 正则自动编码器

编码维数小于输入维数的欠完备自动编码器可以学习数据分布最显著的特征。我们已经看到，如果这类自动编码器被赋予过大的容量，它就不能学到任何有用的信息。

如果隐藏层编码的维数允许与输入相等，或隐藏层编码维数大于输入的**过完备**(overcomplete) 情况下，会发生类似的问题。在这些情况下，即使是线性编码器和线性解码器也可以学会将输入复制到输出，而学不到任何有关数据分布的有用信息。

理想情况下，根据要建模的数据分布的复杂性，选择合适的编码维数和编码器、解码器 容量，就可以成功训练任意架构的自动编码器。正则自动编码器提供这样做的可能。正则自动编码器使用的损失函数可以鼓励模型学习其它特性（除了将输入复制到输出），而不用限制使用浅层的编码器和解码器以及小的编码维数来限制模型的容量。这些特性包括稀疏表示、表示的小导数、以及对噪声或输入缺失的鲁棒性。即使模型容量大到足够学习一个简单的复制功能，非线性且过完备的正则自动编码器仍然能学到一些与数据分布相关的有用信息。

除了这里所描述的方法（正则化自动编码器最自然的解释），几乎任何带有隐变量并配有一个推断过程（计算给定输入的隐含表示）的生成模型，都可以看作是自动编码器的一种特殊形式。强调与自动编码器联系的两个生成建模方法是Helmholtz 机 (**?**) 的衍生模型，如变分自编码 (**??**节) 和生成随机网络 (**??**节)。这些模型能自然地学习大容量、对输入过完备的有用编码，而不需要正则化。这些编码显然是有用的，因为这些模型被训练为近似训练数据的最大概率而不是将输入复制到输出。

7.2.1 稀疏自动编码器

稀疏自动编码器简单地在训练时结合编码层的稀疏惩罚 $\Omega(\boldsymbol{h})$ 和重构误差：

$$L(\boldsymbol{x}, g(f(\boldsymbol{x})) + \Omega(\boldsymbol{h}), \tag{7.2}$$

其中 $g(\boldsymbol{h})$ 是解码器的输出，通常 $\boldsymbol{h}$ 是编码器的输出，即 $\boldsymbol{h} = f(\boldsymbol{x})$。

稀疏自动编码器通常用于学习特征，以便用于其他任务如分类。稀疏正则化的自动编码器必须反映训练数据集的独特统计特征，而不是简单地充当恒等函数。以这种方式训练，执行附带稀疏罚的复制任务可以得到能学习有用特征的模型。

我们可以简单地将惩罚项 $\Omega(\boldsymbol{h})$ 视为加到前馈网络的正则项，这个前馈网络的主要任务是将输入复制到输出（无监督学习的目标），并尽可能地根据这些稀疏特征执行一些监督学习任务（根据监督学习的目标）。不像其它正则项如权重衰减，这个正则化没有直观的贝叶斯解释。如**??**节描述，权重衰减和其他正则惩罚可以被解释为一个MAP近似贝叶斯推断，正则化的惩罚对应于模型参数的先验概率分布。这种观点认为，正则化的最大似然对应最大化 $p(\boldsymbol{\theta}|\boldsymbol{x})$，相当于最大化 $\log p(\boldsymbol{x} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$。$\log p(\boldsymbol{x}|\boldsymbol{\theta})$ 即通常的数据似然项，参数的对数先验项 $\log p(\boldsymbol{\theta})$ 则包含了对 $\boldsymbol{\theta}$ 特定值的偏好。这种观点在**??**节有所描述。正则自动编码器不适用这样的解释是因为正则项取决于数据，因此根据定义上（从文字的正式

意义）来说，它不是一个先验。我们仍可以认为这些正则项隐式地表达了对函数的偏好。

我们可以认为整个稀疏自动编码器框架是对带有隐变量的生成模型的近似最大似然训练，而不将稀疏惩罚视为复制任务的正则化。假如我们有一个带有可见变量 $\boldsymbol{x}$ 和隐变量 $\boldsymbol{h}$ 的模型，且具有明确的联合分布 $p_{\text{model}}(\boldsymbol{x}, \boldsymbol{h}) = p_{\text{model}}(\boldsymbol{h}) p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{h})$。<BAD> 我们将 $p_{\text{model}}(\boldsymbol{h})$ 视为模型关于隐变量的先验分布，表示模型看到 $\boldsymbol{x}$ 的信念先验。这与我们之前使用“先验”的方式不同，之前指分布 $p(\boldsymbol{\theta})$ 在我们看到数据前就对模型参数的先验进行编码。对数似然函数可分解为

$$\log p_{\text{model}}(\boldsymbol{x}) = \log \sum_{\boldsymbol{h}} p_{\text{model}}(\boldsymbol{h}, \boldsymbol{x}). \tag{7.3}$$

我们可以认为自动编码器使用一个高似然值 $\boldsymbol{h}$ 的点估计来近似这个总和。这类似于稀疏编码生成模型（6.4节），但 $\boldsymbol{h}$ 是参数编码器的输出，而不是从优化结果推断出的最可能的 $\boldsymbol{h}$。从这个角度看，我们根据这个选择的 $\boldsymbol{h}$，最大化如下

$$\log p_{\text{model}}(\boldsymbol{h}, \boldsymbol{x}) = \log p_{\text{model}}(\boldsymbol{h}) + \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{h}). \tag{7.4}$$

$\log p_{\text{model}}(\boldsymbol{h})$ 项能被稀疏诱导。如Laplace先验，

$$p_{\text{model}}(h_i) = \frac{\lambda}{2} e^{-\lambda |h_i|}, \tag{7.5}$$

对应于绝对值稀疏惩罚。将对数先验表示为绝对值惩罚，我们得到

$$\Omega(\boldsymbol{h}) = \lambda \sum_i |h_i|, \tag{7.6}$$

$$-\log p_{\text{model}}(\boldsymbol{h}) = \sum_i (\lambda |h_i| - \log \frac{\lambda}{2}) = \Omega(\boldsymbol{h}) + \text{const}, \tag{7.7}$$

这里的常数项只跟 λ 有关。通常我们将 λ 视为超参数，因此可以丢弃不影响参数学习的常数项。其它如Student-t先验也能诱导稀疏性。从稀疏性导致 $p_{\text{model}}(\boldsymbol{h})$ 学习成近似最大似然的结果看，稀疏惩罚完全不是一个正则项。这仅仅影响模型关于隐变量的分布。这个观点提供了训练自动编码器的另一个动机：这是近似训练生成模型的一种途径。这也给出了为什么自动编码器学到的特征是有用的另一个解释：它们描述的隐变量可以解释输入。

稀疏自动编码器的早期工作 (**??**) 探讨了各种形式的稀疏性，并提出了稀疏惩罚和 $\log Z$ 项（将最大似然应用到无向概率模型 $p(\boldsymbol{x}) = \frac{1}{Z}\tilde{p}(\boldsymbol{x})$ 时产生）之间的联系。这个想法是最小化 $\log Z$ 防止概率模型处处具有高概率，同理强制稀疏可以防止自动编码器处处具有低的重构误差 。这种情况下，这种联系是对通用机制

的直观理解而不是数学上的对应。在数学上更容易解释稀疏惩罚对应于有向模型 $p_{\text{model}}(\boldsymbol{h})p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{h})$ 中的 $\log p_{\text{model}}(\boldsymbol{h})$。

? 提出了一种在稀疏（和去噪）自动编码器的 $\boldsymbol{h}$ 中实现*真正为零*的方式。该想法是使用修正线性单元来产生编码层。基于将表示真正推向零（如绝对值惩罚）的先验，可以间接控制表示中零的平均数量。

7.2.2 去噪自动编码器

除了向代价函数增加一个惩罚项，我们也可以改变重构误差项得到一个能学到有用信息的自动编码器。

传统的自动编码器最小化以下目标

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))), \tag{7.8}$$

其中 L 是一个损失函数，衡量 $g(f(\boldsymbol{x}))$ 与 $\boldsymbol{x}$ 的不相似性，如它们不相似度的 L^2 范数。如果模型被赋予足够的容量，L 仅仅鼓励 $g \circ f$ 学成一个恒等函数。

相反，**去噪自动编码器** (denoising autoencoder, DAE) 最小化

$$L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}}))), \tag{7.9}$$

其中 $\tilde{\boldsymbol{x}}$ 是被某种噪声损坏的 $\boldsymbol{x}$ 的副本。因此去噪自动编码器必须撤消这些损坏，而不是简单地复制输入。

? 和? 指出去噪训练过程强制 f 和 g 隐式地学习 $p_{\text{data}}(\boldsymbol{x})$ 的结构。因此去噪自动编码器也是一个通过最小化重构误差来获取有用特性的例子。这也是将过完备、高容量的模型用作自动编码器的一个例子——只要小心防止这些模型仅仅学习一个恒等函数。去噪自动编码器将在7.5节给出更多细节。

7.2.3 惩罚导数作为正则

另一正则化自动编码器的策略是使用一个类似稀疏自动编码器中的惩罚项 Ω,

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h}, \boldsymbol{x}), \tag{7.10}$$

但 Ω 的形式不同：

$$\Omega(\boldsymbol{h}, \boldsymbol{x}) = \lambda \sum_i \|\nabla_{\boldsymbol{x}} h_i\|^2. \tag{7.11}$$

这迫使模型学习一个在 $\boldsymbol{x}$ 变化小时目标也没有太大变化的函数。因为这个惩罚只对训练数据适用，它迫使自动编码器学习可以反映训练数据分布信息的特征。

这样正则化的自动编码器被称为**收缩自动编码器** (contractive autoencoder, CAE)。这种方法与去噪自动编码器、流形学习和概率模型存在一定理论联系。收缩自动编码器在7.7节有更详细的描述。

7.3 表示能力、层的大小和深度

自动编码器通常只有单层的编码器和解码器，但这不是必然的。实际上深度编码器和解码器能提供更多优势。

回忆**??**节，其中提到加深前馈网络有很多优势。这些优势也同样适用于自动编码器，因为它也属于前馈网络。此外，编码器和解码器自身都是一个前馈网络，因此这两个部分也能各自从深度中获得好处。

普遍逼近定理保证至少有一层隐含层且隐藏单元足够多的前馈神经网络能以任意精度近似任意函数（在很大范围里），这是非平凡深度的一个主要优点。这意味着单层隐藏层的自动编码器在数据范围能表示任意接近数据的恒等函数。但是，从输入到编码的映射是浅层的。这意味这我们不能任意添加约束，比如约束编码稀疏。编码器至少包含一层额外隐藏层的深度自动编码器能够在给定足够多隐藏单元的情况，以任意精度近似任何从输入到编码的映射。

深度可以指数减少表示某些函数的计算成本。深度也能指数减少学习一些函数所需的训练数据量。可以参考**??**节巩固深度在前馈网络中的优势。

实验中，深度自动编码器能比相应的浅层或线性自动编码器产生更好的压缩效率 (**?**)。

训练深度自动编码器的普遍的策略是训练一堆浅层的自动编码器来贪心地预训练相应的深度架构。所以我们经常会遇到浅层自动编码器，即使最终目标是训练深度自动编码器。

7.4 随机编码器和解码器

自动编码器仅仅是一个前馈网络，可以使用与传统的前馈网络同样的损失函数和输出单元。

如**??**节中描述，设计前馈网络的输出单元和损失函数普遍策略是定义一个输出分布 $p(\boldsymbol{y} \mid \boldsymbol{x})$ 并最小化负的对数似然 $-\log p(\boldsymbol{y} \mid \boldsymbol{x})$。在这种情况下，$\boldsymbol{y}$ 是关于目标的向量（如类标）。

在自动编码器中，$\boldsymbol{x}$ 既是输入也是目标。然而，我们仍然可以使用与之前相同的架构。给定一个隐藏层编码$\boldsymbol{h}$，我们可以认为解码器提供了一个条件分布 $p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{h})$. 接着我们根据最小化 $-\log p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h})$ 来训练自动编码器。损失函数的具体形式视 p_{decoder} 的形式而定。就传统的前馈网络来说，我们通常使用线性输出单元来参数化高斯分布的均值（如果 $\boldsymbol{x}$ 是实的）。在这种情况下，负对数似然就对应均方误差判据。类似地，二值 $\boldsymbol{x}$ 对应参数由sigmoid单元确定的Bernoulli分布，离散的 $\boldsymbol{x}$ 对应softmax分布等等。为了便于计算概率分布，通常认为输出变量与给定 $\boldsymbol{h}$ 是条件独立的，但一些技术（如混合密度输出）可以解决输出相关

的建模。

为了更彻底地区别之前看到的前馈网络，我们也可以将**编码函数** (encoding function) $f(\boldsymbol{x})$ 的概念推广为**编码分布** (encoding distribution) $p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$，如图7.2中所示。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 7.2: TODO

任何隐变量模型 $p_{\text{model}}(\boldsymbol{h}, \boldsymbol{x})$ 定义了一个随机编码器

$$p_{\text{encoder}}(\boldsymbol{h} \mid \boldsymbol{x}) = p_{\text{model}}(\boldsymbol{h} \mid \boldsymbol{x}) \tag{7.12}$$

以及一个随机解码器

$$p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h}) = p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{h}). \tag{7.13}$$

一般情况下，编码器和解码器的分布没有必要与一个唯一的联合分布 $p_{\text{model}}(\boldsymbol{x}, \boldsymbol{h})$ 的条件分布相容。**?** 指出将编码器和解码器作为去噪自动编码器来训练能使它们渐近地相容（有足够的容量和样本）。

7.5 去噪自动编码器

去噪自动编码器 (denoising autoencoder, DAE) 是一类接受损坏数据作为输入，并训练来预测原始未被损坏数据作为输出的自动编码器。

DAE的训练过程如图7.3中所示。我们引入一个损坏过程 $C(\tilde{\mathbf{x}} \mid \mathbf{x})$，这个条件分布代表给定数据样本 $\mathbf{x}$ 产生损坏样本 $\tilde{\mathbf{x}}$ 的概率。自动编码器则根据以下过程，从训练数据对 $(\boldsymbol{x}, \tilde{\boldsymbol{x}})$ 中学习**重构分布** (reconstruction distribution) $p_{\text{reconstruct}}(\mathbf{x} \mid \tilde{\mathbf{x}})$：

1. 从训练数据中采一个训练样本 $\boldsymbol{x}$。
2. 从 $C(\tilde{\mathbf{x}} \mid \mathbf{x} = \boldsymbol{x})$ 采一个损坏样本 $\tilde{\boldsymbol{x}}$。
3. 将 $(\boldsymbol{x}, \tilde{\boldsymbol{x}})$ 作为训练样本来估计自动编码器的重构分布 $p_{\text{reconstruct}}(\boldsymbol{x} \mid \tilde{\boldsymbol{x}}) = p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h})$，其中 $\boldsymbol{h}$ 是编码器 $f(\tilde{\boldsymbol{x}})$ 的输出，p_{decoder} 根据解码函数 $g(\boldsymbol{h})$ 定义。

通常我们可以简单地对负对数似然 $-\log p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$ 进行基于梯度法(如minibatch梯度下降)的近似最小化。只要编码器是确定性的，去噪自动编码器就是一个前馈网络，并且可以使用与其他前馈网络完全相同的方式进行训练。

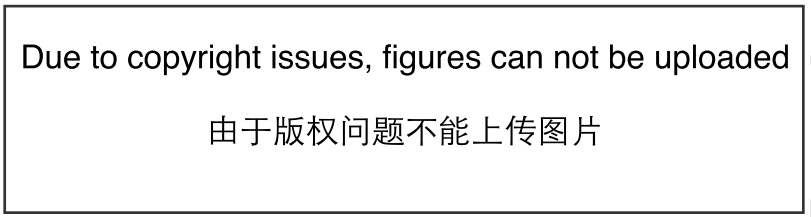


Figure 7.3: TODO

因此我们可以认为DAE是在以下期望下进行随机梯度下降：

$$-\mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}(\mathbf{x})}\mathbb{E}_{\tilde{\mathbf{x}}\sim C(\tilde{\mathbf{x}}|\boldsymbol{x})}\log p_{\text{decoder}}(\boldsymbol{x}\mid\boldsymbol{h}=f(\tilde{\boldsymbol{x}})), \tag{7.14}$$

其中 $\hat{p}_{\text{data}}(\boldsymbol{x})$ 是训练数据的分布。

7.5.1 分数估计

分数匹配(?)是最大似然的代替。它提供了概率分布的一致估计，鼓励模型在各个数据点 $\boldsymbol{x}$ 上获得与数据分布相同的**分数**(score)。在这种情况下，分数是一个特定的梯度场：

$$\nabla_{\boldsymbol{x}}\log p(\boldsymbol{x}). \tag{7.15}$$

将在??中更详细的讨论分数匹配。对于现在讨论的自动编码器，理解学习 $\log p_{\text{data}}$ 的梯度场是学习 p_{data} 结构的一种方式就足够了。

DAE的训练判据(条件高斯 $p(\boldsymbol{x}\mid\boldsymbol{h})$)能让自动编码器学到能估计数据分布得分的向量场 $(g(f(\boldsymbol{x}))-\boldsymbol{x})$，这是DAE的一个重要特性。具体如图7.4所示。

去噪地训练一类采用高斯噪声和均方误差作为重构误差的特定去噪自动编码器(sigmoid隐藏单元，线性重构单元)，与训练一类特定的被称为RBM的无向概率模型是等价的(?)。这类模型将在??给出更详细的介绍；对于现在的讨论，我们只需知道这个模型能显式的给出 $p_{\text{model}}(\boldsymbol{x};\boldsymbol{\theta})$。当RBM使用**去噪得分匹配**(denoising score matching)(?)训练时，它的学习算法与训练对应的去噪自动编码器是等价的。在一个确定的噪声水平下，正则化的分数匹配不是一致估计量；相反它会恢复分布的一个模糊版本。然而，当噪声水平趋向于 0 且训练样本数趋向与无穷时，一致性就会恢复。将会在??更详细地讨论去噪得分匹配。

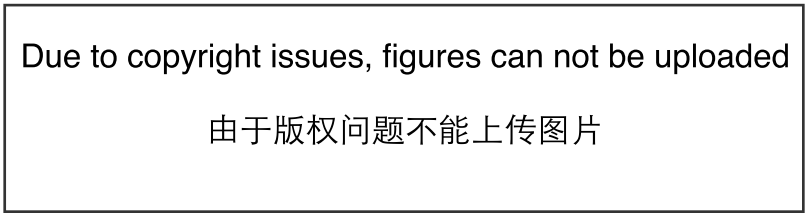


Figure 7.4: TODO

自动编码器和RBM还存在其它联系。分数匹配应用于RBM后，其代价函数将等价于重构误差结合类似CAE惩罚的正则项 (**?**)。**?** 指出自动编码器的梯度是对RBM对比散度训练的近似。

对于连续的 $\boldsymbol{x}$，高斯损坏和重构分布的去噪判据得到的分数估计适用于一般编码器和解码器的参数化 (**?**)。这意味着一个使用平方误差判据

$$\|g(f(\tilde{\boldsymbol{x}})) - \boldsymbol{x}\|^2 \tag{7.16}$$

和噪声方差为 σ^2 的损坏

$$C(\tilde{x} = \tilde{\boldsymbol{x}}|\boldsymbol{x}) = N(\tilde{\boldsymbol{x}}; \mu = \boldsymbol{x}, \Sigma = \sigma^2 I) \tag{7.17}$$

的通用编码器-解码器架构可以用来训练估计分数。图7.5展示其中的工作原理。

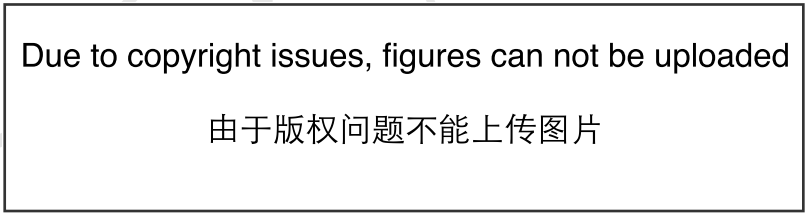


Figure 7.5: TODO

一般情况下，不能保证重构函数 $g(f(\boldsymbol{x}))$ 减去输入 $\boldsymbol{x}$ 后对应于某个函数的梯度，更不用说分数 。这是早期工作 (**?**) 专用于特定参数化的原因 (其中 $g(f(\boldsymbol{x})) - \boldsymbol{x}$ 能通过另一个函数的导数获得)。**?** 通过标识一类特殊的浅层自动编码器家族，使 $g(f(\boldsymbol{x})) - \boldsymbol{x}$ 对应于这个家族所有成员的一个分数，以此推广**?** 的结果。

目前为止我们所讨论的仅限于去噪自动编码器如何学习表示一个概率分布。更一般的，我们可能希望使用自动编码器作为生成模型，并从该分布中进行采样。这将在**??**中讨论。

7.5.2 历史观点

采用MLP去噪的想法可以追溯到? 和? 的工作。? 也曾使用循环网络对图像去噪。在某种意义上，去噪自动编码器仅仅是被训练去噪的MLP。然而，“去噪自动编码器” 的命名指的不仅仅是学习去噪，而且可以学到一个好的内部表示（作为学习去噪的副效用)。这个想法提出较晚 (??)。学习到的表示可以被用来预训练更深的无监督网络或监督网络。与稀疏自动编码器、稀疏编码、收缩自动编码器等正则化的自动编码器类似，DAE的动机是允许使用一个容量非常大的编码器，同时防止在编码器和解码器学习一个毫无用处的恒等函数。

在引入现代DAE之前，? 探讨了与一些相同的方法和相同的目标。他们的做法在有监督目标的情况下最小化重构误差 ，并在监督MLP的隐藏层注入噪声，通过引入重构误差和注入噪声提升泛化能力。然而，他们的方法基于线性编码器，因此无法学习到现代DAE能学习的强大函数族。

7.6 使用自动编码器学习流形

如??描述，自动编码器跟其它很多机器学习算法一样，也应用了将数据集中在一个低维流形或者一小组这样的流形的思想。其中一些机器学习算法仅能学习到在流形上表现良好但给定不在流形上的输入会导致异常的函数。自动编码器进一步借此想法，旨在学习流形的结构。

要了解自动编码器如何做到这一点，我们必须介绍流形的一些重要特性。

流形的一个重要特征是**切平面**(tangent plane) 的集合。d 维流形上的一点 $\boldsymbol{x}$，切平面由能张成流形上允许变动的局部方向的 d 维基向量给出。如图7.6所示，这些局部方向说明了我们能如何微小地改变 $\boldsymbol{x}$ 但一直处于流形上。

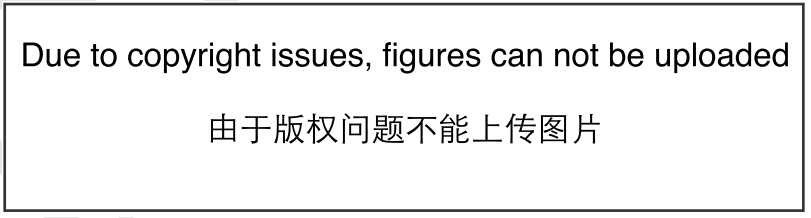


Figure 7.6: TODO

所有自动编码器的训练过程涉及两种推动力的折衷：

1. 学习训练样本 $\boldsymbol{x}$ 的表示$\boldsymbol{h}$ 使得 $\boldsymbol{x}$ 能通过解码器近似地从 $\boldsymbol{h}$ 中恢复。$\boldsymbol{x}$ 是从训练数据挑出的事实是关键的，因为这意味着在自动编码器不需要成功重构不属于数据生成分布下的输入。

2. 满足约束或正则惩罚。这可以是限制自动编码器容量的架构约束，也可以是加入到重构代价的一个正则项。这些技术一般倾向那些对输入较不敏感的解。

显然，单一的推动力是无用的——从它本身将输入复制到输出是无用的，同样忽略输入也是没用的。相反，两种推动力结合是有用的，因为它们迫使隐藏的表示能捕获有关数据分布结构的信息。重要的原则是，自动编码器必须有能力表示*重构训练实例所需的变化*。如果该数据生成分布集中靠近一个低维流形，自动编码器能隐式产生捕捉这个流形局部坐标系的表示：仅在 $\boldsymbol{x}$ 周围关于流形的相切变化需要对应于 $\boldsymbol{h} = f(\boldsymbol{x})$ 中的变化。因此，编码器学习从输入空间 $\boldsymbol{x}$ 到表示空间的映射，映射仅对沿着流形方向的变化敏感，并且对流形正交方向的变化不敏感。

图7.7中一维的例子，说明使重构函数对数据点周围的扰动输入不敏感，我们可以让自动编码器恢复流形的结构。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 7.7: TODO

对比其他的方法是有用且受启发的，可以了解自动编码器为什么对流形学习是有用的。学习表征流形最常见的是流形上（或附近）数据点的表示。对于特定的实例，这样的表示也被称为嵌入。它通常由一个低维向量给出，具有比这个流形的“外围”空间更少的维数。有些算法（下面讨论的非参数流形学习算法）直接学习每个训练样例的嵌入，而其他算法学习一个更一般的映射（有时被称为编码器或表示函数），将周围空间（输入空间）的任意点映射到它的嵌入。

流形学习大多专注于试图捕捉到这些流形的无监督学习过程。最初始的学习非线性流形的机器学习研究专注基于**最近邻图**(nearest neighbor graph) 的**非参数**(non-parametric) 方法。该图中每个训练样例对应一个节点，它的边连接近邻点对。如图7.8所示，这些方法 (**??????????**) 将每个节点与张成实例和近邻之间的差向量变化方向的切平面相关联。

全局坐标系就可以通过优化或求解线性系统获得。图7.9展示了如何通过大量局部线性的类高斯样平铺（或“薄煎饼”，因为高斯块在切平面方向是扁平的）得到一个流形。

然而，**?** 指出了这些局部非参数方法应用于流形学习的根本困难：如果流形不是很光滑（它们有许多波峰、波谷和弯曲），我们可能需要非常多的训练样本以覆

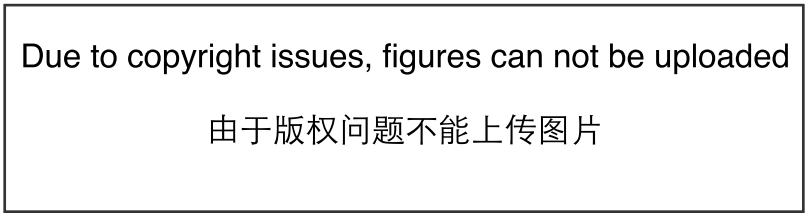


Figure 7.8: TODO

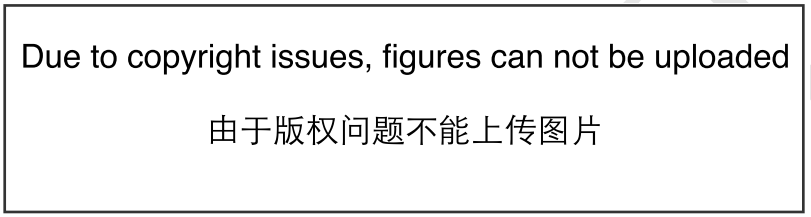


Figure 7.9: TODO

盖其中的每一个变化，导致没有能力泛化到没见过的变化。实际上，这些方法只能通过内插，概括相邻实例之间流形的形状。不幸的是，AI问题中涉及的流形可能具有非常复杂的结构，难以仅从局部插值捕获特征。考虑图7.6转换所得的流形样例。如果我们只观察输入向量内的一个坐标 $\boldsymbol{x}_i$，当平移图像，我们可以观察到当这个坐标遇到波峰或波谷时，图像的亮度也会经历一个波峰或波谷。换句话说，底层图像模板亮度的模式复杂性决定执行简单的图像变换所产生的流形的复杂性。这是采用分布式表示和深度学习来捕获流形结构的动机。

7.7 收缩自动编码器

收缩自动编码器(**??**) 在编码$\boldsymbol{h} = f(\boldsymbol{x})$ 的基础上添加了显式的正则项，鼓励 f 的导数尽可能小：

$$\Omega(\boldsymbol{h}) = \lambda \left\| \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right\|_F^2 . \tag{7.18}$$

惩罚项 $\Omega(\boldsymbol{h})$ 为平方Frobenius范数（元素平方之和），作用于与编码器的函数相关偏导数的雅各比矩阵。

去噪自动编码器和收缩自动编码器之间存在一定联系：**?** 指出在小高斯噪声的限制下，当重构函数将 $\boldsymbol{x}$ 映射到 $\boldsymbol{r} = g(f(\boldsymbol{x}))$ 时，去噪重构误差与收缩惩罚项是

等价的。换句话说，去噪自动编码器能抵抗小且有限的输入扰动，而收缩自动编码器使特征提取函数能抵抗极小的输入扰动。

分类任务中，基于雅各比的收缩惩罚预训练特征函数 $f(\boldsymbol{x})$，将收缩惩罚应用在 $f(\boldsymbol{x})$ 而不是 $g(f(\boldsymbol{x}))$ 可以产生最好的分类精度。如7.5.1节所讨论，应用于 $f(\boldsymbol{x})$ 的收缩惩罚与分数匹配也有紧密的联系。

收缩(contractive) 源于CAE弯曲空间的方式。具体来说，由于CAE是训练成能抵抗输入扰动，鼓励将输入点邻域映射到输出点一更小的邻域。我们能认为这是将输入的邻域收缩到更小的输出邻域。

说得更清楚一点，CAE只在局部收缩——一个训练样本 $\boldsymbol{x}$ 的所有扰动都映射到 $f(\boldsymbol{x})$ 的附近。全局来看，两个不同的点 $\boldsymbol{x}$ 和 $\boldsymbol{x}'$ 会分别被映射到远离原点的两个点 $f(\boldsymbol{x})$ 和 $f(\boldsymbol{x}')$。f 扩展到数据流形的中间或远处是合理的（见图7.7中小例子的情况）。当 $\Omega(\boldsymbol{h})$ 惩罚应用于sigmoid单元时，收缩雅各比的简单方式是令sigmoid趋向饱和的 0 或 1。这鼓励CAE使用sigmoid的极值编码输入点，或许可以解释为二进制编码。它也保证了CAE可以穿过大部分sigmoid隐藏单元能张成的超立方体，进而扩散其编码值。

我们可以认为点 $\boldsymbol{x}$ 处的雅各比矩阵 $\boldsymbol{J}$ 能将非线性编码器近似为线性算子。这允许我们更形式地使用 “收缩” 这个词。在线性理论中，当 $\boldsymbol{Jx}$ 的范数对于所有单位 $\boldsymbol{x}$ 都小于等于 1 时，$\boldsymbol{J}$ 被称为收缩的。换句话说，如果 $\boldsymbol{J}$ 收缩了单位球，他就是收缩的。我们可以认为CAE为了鼓励每个局部线性算子具有收缩性，而在每个训练数据点上将Frobenius范数作为 $f(\boldsymbol{x})$ 的局部线性近似的惩罚。

如7.6中描述，正则自动编码器基于两种相反的推动力学习流形。在CAE的情况下，这两种推动力是重构误差和收缩惩罚 $\Omega(\boldsymbol{h})$。单独的重构误差鼓励CAE学习一个恒等函数。单独的收缩惩罚将鼓励CAE学习关于 $\boldsymbol{x}$ 是恒定的特征。这两种推动力的的折衷产生一个导数 $\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}$ 大多是微小的自动编码器。只有少数隐藏单元，对应于一小部分输入数据的方向，可能有显著的导数。

CAE的目标是学习数据的流形结构。大的 $\boldsymbol{Jx}$ 快速改变 $\boldsymbol{h}$ 的方向 $\boldsymbol{x}$ 很可能是近似流形切平面的方向。**??** 的实验显示训练CAE会导致 $\boldsymbol{J}$ 中大部分奇异值（幅值）比 1 小，因此是收缩的。然而，有些奇异值仍然比 1 大，因为重构误差的惩罚鼓励CAE对最大局部变化的方向进行编码。对应于最大奇异值的方向被解释为收缩自动编码器学到的切方向。理想情况下，这些切方向应对应于数据的真实变化。比如，一个应用于图像的CAE应该能学到显示图像改变的切向量，如7.6图中物体渐渐改变状态。如图7.10所示，实验获得的奇异向量的可视化似乎真的对应于输入图象有意义的变换。

收缩自动编码器正则化判据的一个实际问题是，尽管它在单一隐含层的自动编码器情况下是容易计算的，但在更深的自动编码器情况下会变的难以计算。根据**?** 的策略，分别训练一系列单层的自动编码器，并且每个被训练为重构前一个自动编码器的隐含层。这些自动编码器的组合就组成了一个深度自动编码器。

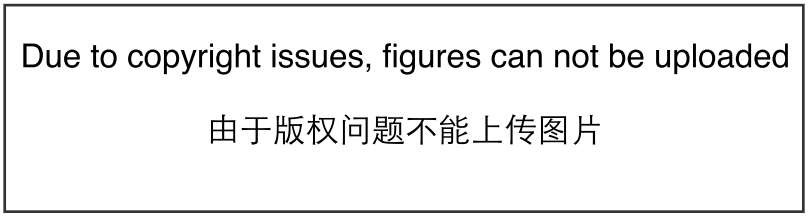


Figure 7.10: TODO

因为每个层分别训练成局部收缩，深度自动编码器自然也是收缩的。这个结果与联合训练深度模型完整架构（带有关于雅各比的惩罚项）获得的结果是不同的，但它抓住了许多理想的定性特征。

另一个实际问题是，如果我们不对解码器强加一些约束，收缩惩罚可能导致无用的结果。例如，编码器将输入乘一个小常数 ϵ，解码器将编码除以一个小常数 ϵ。随着 ϵ 趋向于 0，编码器会使收缩惩罚项 $\Omega(\boldsymbol{h})$ 趋向于 0 而学不到任何关于分布的信息。同时，解码器保持完美的重构。**?** 通过绑定 f 和 g 的权重来防止这种情况。f 和 g 都是由线性仿射变换后进行逐元素非线性变换的标准神经网络层组成，因此将 g 的权重矩阵设成 f 权重矩阵的转置是很直观的。

7.8 预测稀疏分解

预测稀疏分解 (predictive sparse decomposition, PSD) 是稀疏编码和参数化自动编码器(**?**) 的混合模型。参数化编码器被训练为能预测迭代推断的输出。PSD被应用与图片和视频中对象识别的无监督特征学习 (**????**)，在音频中也有所应用 (**?**)。这个模型由一个编码器$f(\boldsymbol{x})$ 和一个解码器$g(\boldsymbol{h})$ 组成，并且都是参数化的。在训练过程中，$\boldsymbol{h}$ 由优化算法控制。优化过程是最小化

$$\|\boldsymbol{x} - g(\boldsymbol{h})\|^2 + \lambda|\boldsymbol{h}|_1 + \gamma\|\boldsymbol{h} - f(\boldsymbol{x})\|^2. \tag{7.19}$$

就像稀疏编码，训练算法交替地相对 $\boldsymbol{h}$ 和模型的参数最小化上述目标。相对 $\boldsymbol{h}$ 最小化较快，因为 $f(\boldsymbol{x})$ 提供 $\boldsymbol{h}$ 的良好初始值以及损失函数将 $\boldsymbol{h}$ 约束在 $f(\boldsymbol{x})$ 附近。简单的梯度下降算法只需 10 步左右就能获得理想的 $\boldsymbol{h}$。

PSD所使用的训练程序不是先训练稀疏编码模型，然后训练 $f(\boldsymbol{x})$ 来预测稀疏编码的特征。PSD训练过程正则化解码器，使用 $f(\boldsymbol{x})$ 可以推断出良好编码的参数。

预测稀疏分解是**学习近似推断**(learned approximate inference) 的一个例子。在10.5节中，这个话题将会进一步展开。10章中展示的工具能让我们了解到，PSD能够被解释为最大化模型的对数似然下界来训练有向稀疏编码的概率模型。

在PSD的实际应用中，迭代优化仅在训练过程中使用。模型被部署后，参数编码器f 用于计算学习好的特征。相比于通过梯度下降来推断 $\boldsymbol{h}$，计算 f 是很容易的。因为 f 是一个可微带参函数，PSD模型可堆叠，并用于初始化其他训练判据的深度网络。

7.9 自动编码器的应用

自动编码器已成功应用于降维和信息检索任务。降维是表示学习和深度学习的第一批应用之一。它是研究自动编码器早期动机之一。例如，**?** 训练了一个堆叠RBM，然后利用它们的权重初始化一个深度自动编码器并逐渐变小隐藏层，在30 个单元的瓶颈处达到极值。生成的编码比 30 维的PCA产生更少的重构误差，所学的表示更容易定性解释，并能联系基础类别，这些类别表现为分离良好的集群。

低维表示可以提高许多任务的性能，例如分类。小空间的模型消耗更少的内存和运行时间。据**?** 和**?** 观察，降维的许多形式是跟彼此邻近的例子语义相关的。映射到低维空间能帮助泛化提示了这个想法。

从降维中比普通任务受益更多的是信息检索，即在数据库中查询类似条目的任务。此任务从降维获得类似其他任务的一般益处，同时在某些种低维空间中的搜索变得极为高效。特别的，如果我们训练降维算法来生成一个低维且二值的编码，那么我们就可以将所有数据库条目在哈希表映射为二进制编码向量。这个哈希表允许我们返回具有相同二进制代码的数据库条目作为查询结果进行信息检索。我们也可以非常高效地搜索稍有不同条目，只需反转查询编码的各个位。这种通过降维和二值化的信息检索方法被称为**语义哈希**(semantic hashing)(**??**)，已经被用于文本输入 (**??**) 和图像 (**???**)。

通常在最终层上使用sigmoid的编码函数来产生语义哈希的二值编码。sigmoid单元必须被训练为到达饱和，对所有输入值都接近 0 或接近 1。能做到这一点的窍门就是训练时在sigmoid非线性单元前简单地注入加性噪声。噪声的大小应该随时间增加。要对抗这种噪音并且保存尽可能多的信息，网络必须加大输入到sigmoid函数的幅度，直到饱和。

<BAD> 学习哈希函数的思想也在其他几个方向进一步探讨，包括改变损失训练表示的想法，这个表示与哈希表中查找附近样本的任务有更直接的联系 (**?**)。

Chapter 8

深度学习中的结构化概率模型

深度学习为研究者们提供了许多的指导性的建模和设计算法的思路。其中一种形式是**结构化概率模型**(structured probabilistic models)。我们曾经在**??**中简要讨论过。那个简单的介绍已经足够使我们充分了解如何使用结构化概率模型来描述第二部分中的某些算法。现在，在第三部分，结构化概率模型是许多深度学习中重要的研究方向的重要组成部分。作为讨论这些研究方向的预备知识，本章详细的描述了结构化概率模型。本章中我们努力做到内容的自包含性。在阅读本章之前读者不需要回顾之前的介绍。

结构化概率模型通过使用图描述概率分布中的随机变量之间的直接相互作用来描述一个概率分布。在这里我们使用来图论（一系列的结点通过一系列的边来连接）中的图的概念，由于模型的结构是由图来定义的，所以这些模型也通常被叫做**图模型**(graphical models)。

图模型的研究领域是巨大的，曾提出过大量的模型，训练算法和推断算法。在本章中，我们介绍了图模型中几个核心方法的基本背景，并且强调了在深度学习领域中图模型已经被公认为是有效的。如果你已经对图模型已经了解很多，那么你可以跳过本章的绝大部分。然而，我们相信即使是资深的图模型方向的研究者也会从本章的最后一节中获益匪浅，详见8.7节，其中我们强调了在深度学习算法中一些图模型特有的算法。相比于普通的图模型研究者，深度学习的研究者们通常会使用完全不同的模型结构，学习算法和推断过程。在本章中，我们讨论了这种区别并且解释了其中的原因。

在本章中，我们首先介绍了建立大尺度概率模型中面临的挑战。之后，我们介绍了如何使用一个图来描述概率分布的结构。尽管这个方法能够帮助我们解决许多的挑战和问题，它本身也有很多缺陷。图模型中的一个主要难点就是判断哪些变量之间存在直接的相互作用关系，也就是对于给定的问题哪一种图结构是最适合的。在8.5节中，通过了解**依赖性**(dependency)，我们简要概括了解决这个难点的两种基本方法。最后，在8.7节中，我们讨论并强调了图模型在深度学习中的一

些独特之处和一些特有的方法，作为本章的收尾。

8.1 非结构化建模的挑战

深度学习的目标是使得机器学习能够解决许多人工智能中需要解决的挑战。这也意味着能够理解具有丰富结构的高维数据。举个例子，我们希望人工智能的算法能够理解自然图片[1]，包含语音的声音信号和拥有许多词和标点的文档。

分类问题可以把这样一个来自高维分布的数据作为输入，然后用一个标签来概括它———这个标签可以是照片中是什么物品，一段语音中说的是哪个单词，也可以是一段文档描述的是哪个话题。分类的这个过程丢弃了输入数据中的大部分信息，然后给出了一个单个值的输出（或者是一个输出值的概率分布）。这个分类器通常可以忽略输入数据的很多部分。举个例子，当我们识别一个照片中是哪一个物品的时候，我们通常可以忽略图片的背景。

我们也可以使用概率模型来完成许多其它的任务。这些任务通常比分类更加昂贵。其中的一些任务需要产生多个输出。大部分任务需要对输入数据的整个结构的完整理解，所以并不能舍弃数据的一部分。这些任务包括了以下几个：

- 估计密度函数：给定一个输入 $\boldsymbol{x}$，机器学习系统返回了一个对数据生成分布的真实密度函数 $p(\boldsymbol{x})$ 的估计。这只需要一个输出，但它需要完全理解整个输入。即使向量中只有一个元素不太正常，系统也会给它赋予很低的概率。

- 去噪音：给定一个受损的或者观察有误的输入数据 $\tilde{\boldsymbol{x}}$，机器学习系统返回了一个对于原始的真实的 $\boldsymbol{x}$ 的估计。举个例子，有时候机器学习系统需要从一张老相片中去除污渍或者抓痕。这个系统会产生多个输出（对应着估计的干净的样本 $\boldsymbol{x}$ 的每一个元素），并且需要我们有一个对输入的整体理解（因为即使一个严重损害的区域也需要在最后的输出中恢复）。

- 缺失值的插入：给定 $\boldsymbol{x}$ 的某些元素作为观察值，模型被要求返回一个 $\boldsymbol{x}$ 的一些或者全部未观察值的概率分布。这个模型返回的也是多个输出。由于这个模型需要恢复 $\boldsymbol{x}$ 的每一个元素，所以它必须理解整个输入。

- 采样：模型从分布 $p(\boldsymbol{x})$ 中抽取新的样本。应用包含了语音合成，即产生一个听起来很像人说话的声音。这个模型也需要多个输出以及对整体输入的良好建模。即使样本只有一个从错误分布中产生的元素，那么采样的过程肯定是错误的。

图8.1中描述了一个使用较小的自然图片的采样任务。

[1] 自然图片指的是能够在正常的环境下被照相机拍摄的图片，以区别于合成的图片，或者一个网页的截图。

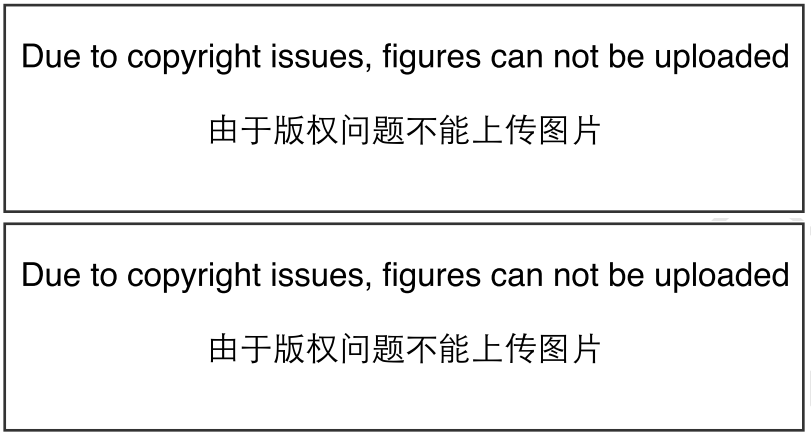


Figure 8.1: TODO

对上千甚至是上百万的随机变量的分布建模，无论从计算上还是从统计意义上说，都是一个具有挑战性的任务。假设我们只想对二值的随机变量建模。这是一个最简单的例子，但是仍然无能为力。对一个小的 32×32 像素的彩色（RGB）图片来说，存在 2^{3072} 种可能的二值图片。这个数量已经超过了 10^{800}，比宇宙中的原子总数还要多。

通常意义上讲，如果我们希望对一个包含 n 个离散变量并且每个变量都能取 k 个值的 $\boldsymbol{x}$ 的分布建模，那么最简单的表示 $P(\boldsymbol{x})$ 的方法需要存储一个可以查询的表格。这个表格记录了每一种可能的值的概率，需要记录 k^n 个参数。

这种方式是不可行的，基于下述几个原因：

- 内存：存储开销。对于除了极小的 n 和 k 以外的值来说，表示用表格的形式来这样一个分布需要太多的存储空间。
- 统计的高效性：当模型中的参数个数增加的时候，使用随机估计时训练数据的量也需要相应的增加。因为基于查表的模型拥有天文数字级别的参数，为了准确的拟合，相应的训练集的大小也是相同级别的。任何这样的模型都会导致严重的过拟合，除非我们添加一些额外的假设来联系表格中的不同元素（正如第5.4.1中所举的 back-off 或者平滑过的n-gram模型）。
- 运行时间：推断的开销。假设我们需要完成一个推断的任务，其中我们通过对联合分布 $p(\boldsymbol{x})$ 来计算某些其它的分布，比如说边缘分布 $p(x_1)$ 或者是条件分布 $p(x_2|x_1)$。计算这样的分布需要对整个表格的某些项进行求和操作，因此这样的操作的运行时间和上述的高昂的内存开销是一个级别的。
- 运行时间：采样的开销。类似的，假设我们想要从这样的模型中采样。最简

单的方法就是从均匀分布中采样，$u \sim \mathrm{U}(0,1)$，然后把表格中的元素累加起来，直到和大于 u，然后返回最后一个加上的元素。最差情况下，这个操作需要读取整个表格，所以相比于其它操作，它需要指数性的时间。

基于表格操作的方法的主要问题是我们显示的表示出了每一种可能的变量的取值可能以及每一种相互作用方式。在实际问题中我们遇到的概率分布远比这个简单。通常，许多变量只是间接的相互作用。

比如说，我们想要对接力跑步比赛中一个队伍完成比赛的时间进行建模。假设这个队伍有三名成员：Alice, Bob 和 Carol。在比赛开始的时候，Alice 拿着接力棒，开始跑第一段距离。在跑完她的路程以后，她把棒递给了 Bob。然后 Bob 开始跑，再把棒给 Carol，Carol 跑最后一棒。我们可以用连续变量来建模他们每个人完成的时间。因为 Alice 第一个跑，所以她的完成时间并不依赖于其它的人。Bob 的完成时间依赖于 Alice 的完成时间，因为 Bob 只能在 Alice 跑完以后才能开始跑。如果 Alice 跑的更快，那么 Bob 也会完成的更快。所有的其它关系都可以被类似的推出。最后，Carol 的完成时间依赖于她的两个队友。如果 Alice 跑的很慢，那么 Bob 也会完成的更慢。结果，Carol 将会更晚开始跑步，因此她的完成时间也更有可能更晚。然而，在给定 Bob 的完成时间的时候 Carol 的完成时间只是间接的依赖于 Alice 的完成时间。如果我们已经知道了 Bob 的完成时间，知道 Alice 的完成时间对估计 Carol 的完成时间并无任何帮助。这意味着可以通过仅仅两个相互作用来建模这个接力赛。这两个相互作用分别是 Alice 的完成时间对 Bob 的完成时间的影响和 Bob 的完成时间对 Carol 的完成时间的影响。在这个模型中，我们可以忽略第三种，间接的相互作用，即 Alice 的完成时间对 Carol 的完成时间的影响。

结构化概率模型为随机变量之间的直接作用提供了一个正式的建模框架。这种方式大大减少了模型的参数个数以致于模型只需要更少的数据来进行有效的估计。这些更轻便的模型在模型存储，模型推断以及采样的时候有着更小的计算开销。

8.2 使用图来描述模型结构

结构化概率模型使用图（在图论中结点是通过边来连接的）来表示随机变量之间的相互作用。每一个结点代表一个随机变量。每一条边代表一个直接相互作用。这些直接相互作用隐含着其它的间接相互作用，但是只有直接的相互作用是被显式的建模的。

使用图来描述概率分布中的相互作用的方法不止一种。在下文中我们会介绍几种最为流行和有用的方法。图模型可以被大致的分为两类：基于模型的有向无环图，和基于模型的无向模型。

8.2.1 有向模型

有一种结构化概率模型是**有向图模型**(directed graphical model)，也被叫做**信念网络**(belief network) 或者**贝叶斯网络**(Bayesian network)[2] (?)。

之所以命名为有向图模型是因为所有的边都是有方向的，即从一个结点指向另一个结点。这个方向可以通过画一个箭头来表示。箭头所指的方向表示了这个随机变量的概率分布依赖于其它的变量。画一个从结点 a 到结点 b 的箭头表示了我们用一个条件分布来定义 b，而 a 是作为这个条件分布的符号右边的一个变量。换句话说，b 的概率分布依赖于 a 的取值。

我们继续8.1节所讲的接力赛的例子，我们假设 Alice 的完成时间为 t_0，Bob 的完成时间为 t_1，Carol 的完成时间为 t_2。就像我们之前看到的一样，t_1 的估计是依赖于 t_0 的，t_2 的估计是依赖于 t_1 的，但是仅仅间接的依赖于 t_0。我们用一个有向图模型来建模这种关系，就如在图8.2中看到的一样。

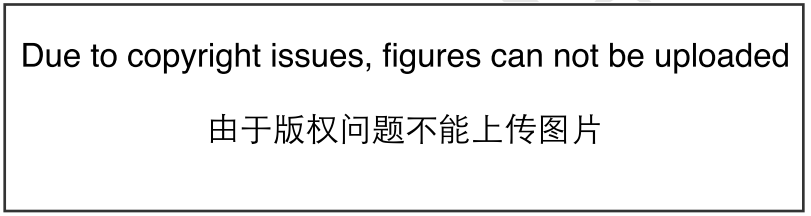


Figure 8.2: TODO

正式的讲，变量 $\boldsymbol{x}$ 的有向概率模型是通过有向无环图 $\mathcal{G}$ 和一系列**局部条件概率分布**(local conditional probability distribution)$p(x_i|P_{a\mathcal{G}}(x_i))$ 来定义的，其中 $P_{a\mathcal{G}}(x_i)$ 表示结点 x_i 的所有父结点。$\boldsymbol{x}$ 的概率分布可以表示为

$$p(\boldsymbol{x}) = \prod_i p(x_i|P_{a\mathcal{G}}(x_i)) \tag{8.1}$$

在之前所述的接力赛的例子中，这意味着概率分布可以被表示为

$$p(t_0, t_1, t_2) = p(t_0)p(t_1|t_0)p(t_2|t_1) \tag{8.2}$$

这是我们看到的第一个结构化概率模型的实际例子。我们能够检查这样建模的计算开销，为了验证相比于非结构化建模，结构化建模为什么有那么多的优势。

假设我们采用从第 0 分钟到第 10 分钟每 6 秒一块的方式离散化的表示时间。这使得 t_0，t_1 和 t_2 都是一个有 100 个取值可能的离散变量。如果我们尝试着用

[2]当我们希望强调从网络中计算出的值的推理本质，尤其是强调这些值代表的是置信的程度大小时，Judea Pearl 建议使用贝叶斯网络这个术语。

一个表来表示 (t_0, t_1, t_2)，那么我们需要存储 $999,999$ 个值（t_0 的取值 $100 \times t_1$ 的取值 $100 \times t_2$ 的取值 100 减去 1，所有的概率的和为 1，所以其中有 1 个值的存储是多余的）。反之，如果我们用一个表来记录条件概率分布，那么记录 t_0 的分布需要存储 99 个值，给定 t_0 情况下 t_1 的分布需要存储 9900 个值，给定 t_1 情况下 t_2 的分布也需要存储 9900 个值。加起来总共需要存储 $19,899$ 个值。这意味着使用有向图模型将需要存储的参数的个数减少了超过 50 倍！

通常意义上说，对每个变量都能取 k 个值的 n 个变量建模，基于建表的方法需要的复杂度是 $O(k^n)$，就像我们之前讨论过的一样。现在假设我们用一个有向图模型来对这些变量建模。如果 m 代表图模型的单个条件概率分布中最大的变量数目（在条件符号的左右皆可），那么对这个有向模型建表的复杂度大致为 $O(k^m)$。只要我们在设计模型时使其满足 $m \ll n$，那么复杂度就会被大大的减小。

换一句话说，只要每个变量只有少量的父结点，那么这个分布就可以用较少的参数来表示。图结构上的一些限制条件，比如说要求这个图为一棵树，也可以保证一些操作（比如说求一小部分变量的边缘或者条件分布）更加的高效。

决定那些信息需要被包含在图中而哪些不需要是很重要的。如果许多变量可以被假设为是条件独立的，那么这个图模型可以被大大简化。当然也存在其它的简化图模型的假设。比如说，我们可以假设无论 Alice 的表现如何，Bob 总是跑的一样快（实际上，Alice 的表现很大概率的会影响 Bob 的表现，这取决于 Bob 的性格，如果在之前的比赛中 Alice 跑的特别快，这有可能鼓励 Bob 更加努力，当然这也有可能使得 Bob 懒惰）。那么 Alice 对 Bob 的唯一影响就是在计算 Bob 的完成时间时需要加上 Alice 的时间。这个假设使得我们所需要的参数量从 $O(k^2)$ 降到了 $O(k)$。然而，值得注意的是在这个假设下 t_0 和 t_1 仍然是直接相关的，因为 t_1 表示的是 Bob 完成的时间，并不是他跑的时间。这也意味着图模型中会有一个从 t_0 指向 t_1 的箭头。“Bob 的个人跑步时间相对于其它因素是独立的'' 这个假设无法在 t_0, t_1, t_2 的图模型中被表示出来。我们只能将这个关系表示在条件分布中。这个条件分布不再是一个大小为 $k \times k - 1$ 的分别对应着 t_0, t_1 的表格，而是一个包含了 $k - 1$ 个参数的略复杂的公式。有向图模型并不能对我们如何定义条件分布做出任何限制。它只能定义哪些变量之间存在着依赖性关系。

8.2.2 无向模型

有向图模型为我们提供了一个描述结构化概率模型的语言。而另一种常见的语言则是**无向模型**(undirected Model)，也叫做**马尔可夫随机场**(Markov random fields)或者是**马尔可夫网络**(Markov network) (**?**)。就像它们的名字所说的那样，无向模型中所有的边都是没有方向的。

有向模型适用于这种情况，当存在一个很明显的理由来描述每一个箭头的时候。有向模型中，经常存在我们理解的具有因果关系以及因果关系有明确的方向的情况下。接力赛的例子就是一个这样的例子。之前的运动员的表现影响了后面

的运动员，而后面的运动员却不会影响前面的运动员。

然而并不是所有的情况的相互影响中都有一个明确的方向关系。当相互的作用并没有本质的方向，或者是明确的有相互影响的时候，使用无向模型更加合适。

作为一个这种情况的例子，假设我们希望对三个二值随机变量建模：你是否生病，你的同事是否生病以及你的室友是否生病。就像在接力赛的例子中所作的简化的假设一样，我们可以在这里做一些简化的假设。假设你的室友和同事并不认识，所以他们不太可能直接相互传染一些疾病，比如说是感冒。这个事件太过稀有，所以我们不对此事件建模。然而，如果他们之一感染了感冒，很有可能通过你来传染给了另一个人。我们通过对你的同事传染给你以及你传染给你的室友建模来对这种间接的从你的同事到你的室友的感冒传染建模。

在这种情况下，你传染给你的室友和你的室友传染给你都是非常容易的，所以不存在一个明确的单向的模型。这启发我们使用无向模型，其中随机变量对应着图中的相互作用的结点。不像是有向模型中，在无向模型中的边是没有方向的，并不与一个条件分布相关联。

我们把对应你的健康的随机变量记住 h_y，对应你的室友健康状况的随机变量记住 h_r，你的同事健康的变量记住 h_c。图8.3表示来这种关系。

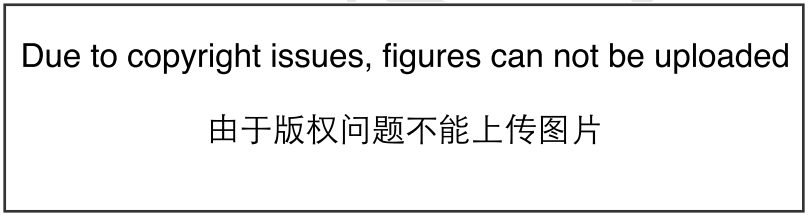


Figure 8.3: TODO

正式的说，一个无向模型是一个定义在无向模型$\mathcal{G}$ 上的一个结构化概率模型。对图中的每一个**团**(clique)[3]$\mathcal{C}$，一个**因子**(factor)$\phi(\mathcal{C})$(也叫做**团势能**(clique potential))，衡量了团中的变量的不同状态所对应的密切程度。这些因子都被限制为是非负的。合在一起他们定义了**未归一化概率函数**(unnormalized probability function)：

$$\tilde{p}(\boldsymbol{x}) = \prod_{\mathcal{C} \in \mathcal{G}} \phi(\mathcal{C}) \tag{8.3}$$

只要所有的团中的结点数都不大，那么我们就能够高效的处理这些未归一化概率函数。它包含了这样的思想，越高密切度的状态有越大的概率。然而，不像贝叶斯网络，几乎不存在团的定义的结构，所以不能保证把它们乘在一起能够得到一个有效的概率分布。图8.4展示了一个从无向模型中读取分解信息的例子。

[3]图的一个团是图中结点的一个子集，并且其中的点是全连接的

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 8.4: TODO

在你，你的室友和同事之间的感冒传染的例子中包含了两个团。一个团包含了 h_y 和 h_c。这个团的因子可以通过一个表来定义，可能取到下面的值：

	$h_y = 0$	$h_y = 1$
$h_c = 0$	2	1
$h_c = 1$	1	10

状态为 1 代表了健康的状态，相对的状态为 0 则表示不好的健康状态（即被感冒传染)。你们两个通常都是健康的，所以对应的状态拥有最高的密切程度。两个人中只有一个人是生病的密切程度是最低的，因为这是一个很少见的状态。两个人都生病的状态（通过一个人来传染给了另一个人）有一个稍高的密切程度，尽管仍然不及两个人都健康的密切程度。

为了完整的定义这个模型，我们需要对包含 h_y 和 h_r 的团定义类似的因子。

8.2.3 分割函数

尽管这个未归一化概率函数处处不为零，我们仍然无法保证它的概率之和或者积分为 1。为了得到一个有效的概率分布，我们需要使用一个归一化的概率分布[4]：

$$p(\boldsymbol{x}) = \frac{1}{Z}\tilde{p}(\boldsymbol{x}) \tag{8.4}$$

其中，Z 是使得所有的概率之和或者积分为 1 的常数，并且满足：

$$Z = \int \tilde{p}(\boldsymbol{x})d\boldsymbol{x} \tag{8.5}$$

当函数 ϕ 固定的时候，我们可以把 Z 当成是一个常数。值得注意的是如果函数 ϕ 带有参数时，那么 Z 是这些参数的一个函数。忽略控制 Z 的变量而直接写 Z 是一个常用的方式。归一化常数 Z 被称作是分割函数，一个从统计物理学中借鉴的术语。

[4]一个通过归一化团势能的乘积的分布通常被称作是吉布斯分布

由于 Z 通常是由对所有的可能的 $\boldsymbol{x}$ 的状态的联合分布空间求和或者求积分得到的，它通常是很难计算的。为了获得一个无向模型的归一化的概率分布，模型的结构和函数 ϕ 的定义通常需要特殊的设计从而使得能够高效的计算 Z。在深度学习中，Z 通常是难以处理的。由于 Z 难以精确的计算出，我们只能使用一些近似的方法。这样的近似方法是第**??**章的主要内容。

在设计无向模型时我们必须牢记在心的一个要点是设置一些因子使得 Z 不存在这样的方法也是有可能的。当模型中的一些变量是连续的，且对 $\tilde{p}$ 上的积分无法收敛的时候这种情况就会发生。比如说，当我们需要对一个单独的标量变量 $x \in \mathbb{R}$ 建模，并且这个包含一个点的团势能定义为 $\phi(x) = x^2$ 时。在这种情况下，

$$Z = \int x^2 dx \tag{8.6}$$

由于这个积分是发散的，所以不存在一个对应着这个势能函数的概率分布。有时候 ϕ 函数的某些参数的选择可以决定相应的概率分布能否存在。比如说，对 ϕ 函数 $\phi(x;\beta) = \exp(-\beta x^2)$ 来说，参数 β 决定了归一化常数 Z 是否存在。一个正的 β 使得 ϕ 函数是一个关于 x 的高斯分布，但是一个非正的参数 β 则使得 ϕ 不可能归一化。

有向模型和无向模型之间的一个重要的区别就是有向模型是通过从起始点的概率分布直接定义的，反之无向模型的定义显得更加宽松，通过 ϕ 函数转化为概率分布而定义。这和我们处理这些建模问题的直觉相反。当我们处理无向模型时需要牢记一点，每一个变量的定义域对于 ϕ 函数所对应的概率分布有着重要的影响。举个例子，我们考虑一个 n 维的向量 $\boldsymbol{x}$ 以及一个由偏置向量 $\boldsymbol{b}$ 参数化的无向模型。假设 $\boldsymbol{x}$ 的每一个元素对应着一个团，并且满足 $\phi^{(i)}(x_i) = \exp(b_i x_i)$。在这种情况下概率分布是怎么样的呢？答案是我们无法确定，因为我们并没有指定 $\boldsymbol{x}$ 的定义域。如果 $\boldsymbol{x}$ 满足 $\boldsymbol{x} \in \mathbb{R}^n$，那么对于归一化常数 Z 的积分是发散的，这导致了对应的概率分布是不存在的。如果 $\boldsymbol{x} \in \{0,1\}^n$，那么 $p(\boldsymbol{x})$ 可以被分解成 n 个独立的分布，并且满足 $p(x_i = 1) = \text{sigmoid}(b_i)$。如果 $\boldsymbol{x}$ 的定义域是基本单位向量的集合 $(\{[1,0,\dots,0],[0,1,\dots,0],\dots,[0,0,\dots,1]\})$，那么 $p(x) = \text{softmax}(\boldsymbol{b})$，因此一个较大的 b_i 的值会降低所有的 $p(x_j = 1)$ 的概率（其中 $j \neq i$）。通常情况下，通过特殊设计变量的定义域，能够使得一个相对简单的 ϕ 函数可以获得一个相对复杂的表达。我们会在**??**节中讨论这个想法的实际应用。

8.2.4 基于能量的模型

无向模型的许多有趣的理论结果依赖于 $\forall \boldsymbol{x},\ \tilde{p}(\boldsymbol{x}) > 0$ 这个假设。使这个条件满足的一种简单的方式是使用**基于能量的模型**(Energy-based model)，其中

$$\tilde{p}(\boldsymbol{x}) = \exp(-E(\boldsymbol{x})) \tag{8.7}$$

$E(\boldsymbol{x})$ 被称作是**能量函数**(energy function)。对所有的 z $\exp(z)$ 都是正的，这保证了没有一个能量函数会使得某一个状态 $\boldsymbol{x}$ 的概率为 0。我们可以很自由的选择那些能够简化学习过程的能量函数。如果我们直接学习各个团势能，我们需要利用带约束的优化方法来指定一些特定的最小概率值。学习能量函数的过程中，我们可以采用无约束的优化方法[5]。基于能量的模型中的概率可以无限趋近于 0 但是永远达不到 0。

服从方程 (8.7)形式的任意分布都是**玻耳兹曼分布**(Boltzmann distribution)的一个实例。正是基于这个原因，我们把许多的基于能量的模型叫做**玻耳兹曼机**(Boltzmann Machine)(**????**)。关于什么时候叫基于能量的模型，什么时候叫玻耳兹曼机不存在一个公认的判别标准。一开始玻耳兹曼机这个术语是用来描述一个只有二进制变量的模型，但是如今许多模型，比如均值-方差受限玻耳兹曼机，也涉及到了实值变量。虽然玻耳兹曼机最初的定义既可以包含隐变量也可以不包含隐变量，但是时至今日玻耳兹曼机这个术语通常用于指拥有隐含变量的模型，而没有隐含变量的玻耳兹曼机则经常被称为马尔可夫随机场或对数线性模型。

无向模型中的团对应于未归一化概率函数的因子。通过 $\exp(a + b) = \exp(a)\exp(b)$，我们发现无向模型中的不同团对应于能量函数的不同项。换句话说，基于能量的模型只是一种特殊的马尔可夫网络：求幂使能量函数中的每个项对应于不同团的因子。关于如何从无向模型结构获得能量函数的形式的示例参见图8.5。人们可以将能量函数带有多个项的基于能量的模型视作是**专家之积**(product of expert)(**?**)。能量函数中的每一项对应的是概率分布中的一个因子。能量函数中的每一项都可以看作决定一个软约束是否能够满足的“专家”。每个专家可以仅执行仅涉及随机变量的低维投影的一个约束，但是结合概率的乘法时，专家们合理构造了复杂的高维约束。

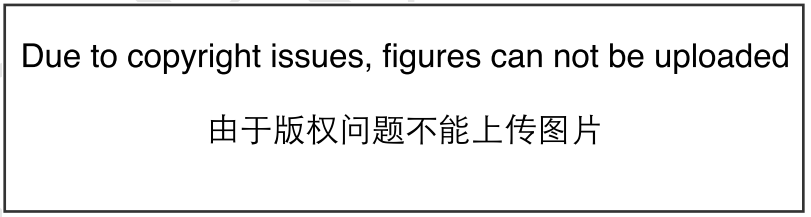


Figure 8.5: TODO

基于能量的模型的定义的一部分无法用机器学习观点来解释：即等式 (8.7)中的负号。这个负号的存在主要是为了保持机器学习文献和物理学文献之间的兼容性。概率建模的许多研究最初由统计物理学家做出的，其中 E 是指实际的，物理概念的能量，没有任意的符号。诸如“能量”和“分割函数”这类术语仍然与这些技术相关联，尽管它们的数学适用性比在物理中更宽。一些机器学习研究者

[5]对于某些模型，我们可以仍然使用带约束的优化方法来确保 Z 存在。

(例如，(**?**) 将负能量称为**harmony**(harmony)) 发出了不同的声音，但这些都不是标准惯例。

许多对概率模型进行操作的算法不需要计算 $p_{\text{model}}(\boldsymbol{x})$，而只需要计算 $\log p_{\text{model}}(\boldsymbol{x})$。对于具有隐含变量 $\boldsymbol{h}$ 的基于能量的模型，这些算法有时会将该量的负数称为**自由能**(free energy)：

$$\mathcal{F}(\boldsymbol{x}) = -\log \sum_{\boldsymbol{h}} \exp(-E(\boldsymbol{x}, \boldsymbol{h})) \tag{8.8}$$

在本书中，我们更倾向于更为通用的基于 $\log \tilde{p}_{\text{model}}(\boldsymbol{x})$ 的定义。

8.2.5 separation和 d-separation

图模型中的边告诉我们哪些变量直接相互作用。我们经常需要知道哪些变量间接相互作用。某些间接相互作用可以通过观察其他变量来启用或禁用。更正式地，我们想知道在给定其他变量子集的值的情况下，哪些变量子集彼此条件独立。

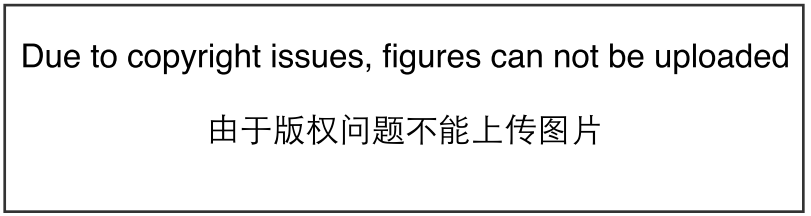
在无向模型的情况下，识别图中的条件独立性是非常简单的。在这种情况下，图中隐含的条件独立性称为**separation**(separation)。如果图结构显示给定变量 $\mathbb{S}$ 的情况下变量 $\mathbb{A}$ 与变量 $\mathbb{B}$ 无关，那么我们声称给定变量 $\mathbb{S}$ 时，变量 $\mathbb{A}$ 与另一组变量 $\mathbb{B}$separation。如果两个变量 a 和 b 通过涉及未观察变量的路径连接，那么这些变量不是separation的。如果它们之间没有路径，或者所有路径都包含可观测的变量，那么它们是separation的。我们认为仅涉及到未观察到的变量的路径是“活跃”的，将包括观察到的变量的路径称为“非活跃”的。

当我们画图时，我们可以通过加阴影来表示观察到的变量。图8.6用于描述当以这种方式绘制时无向模型中的活跃和非活跃路径的样子。图8.7描述了一个从一个无向模型中读取separation的例子。

类似的概念适用于有向模型，除了在有向模型中，这些概念被称为 d-separation。“d”代表“依赖性”程度。有向图的 d-separation的定义与无向模型的相同：我们认为如果图结构中给定 $\mathbb{S}$ 时 $\mathbb{A}$ 与变量集 $\mathbb{B}$ 无关，那么给定变量集 $\mathbb{S}$ 时，变量集 $\mathbb{A}$d-separation于变量集 $\mathbb{B}$。

与无向模型一样，我们可以通过查看图中存在的活跃路径来检查图中隐含的独立性。如前所述，如果两个变量之间存在活跃路径，则两个变量是相关的，如果没有活跃路径，则为 d-separation。在有向网络中，确定路径是否活跃有点复杂。关于在有向模型中识别活跃路径的方法可以参见图8.8。图8.9是从一个图中读取一些属性的例子。

重要的是要记住separation和 d-separation只能告诉我们图中隐含的条件独立性。不需要表示存在的所有独立性的图。进一步的，使用完全图（具有所有可能的边的图）来表示任何分布总是合法的。事实上，一些分布包含不可能用现有图



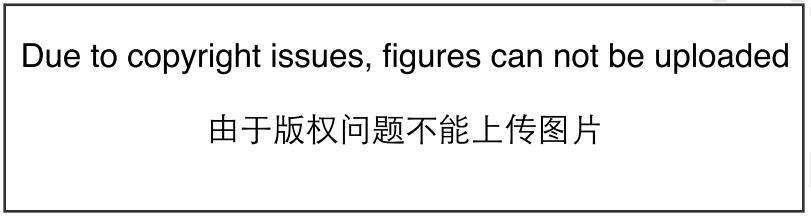


Figure 8.6: TODO

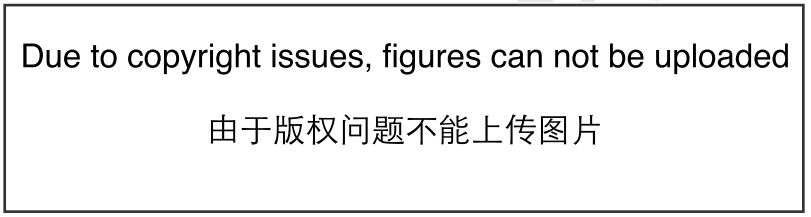


Figure 8.7: TODO

形符号表示的独立性。特定环境下的独立（context-specific independences）指的是取决于网络中一些变量的值的独立性。例如，考虑一个三个二进制变量的模型：a, b 和 c。假设当 a 是 0 时，b 和 c 是独立的，但是当 a 是 1 时，b 确定地等于 c。当 a = 1 时图模型需要连接 b 和 c 的边。当 a = 0 时 b 和 c 不是独立的。

一般来说，当独立性不存在的时候，图不会显示独立性。然而，图可能无法显示存在的独立性。

8.2.6 在有向模型和无向模型中转换

我们经常将特定的机器学习模型称为无向模型或有向模型。例如，我们通常将受限玻耳兹曼机称为无向模型，而稀疏编码则被称为有向模型。这种措辞的选择可能有点误导，因为没有概率模型是有向或无向的。但是，一些模型很适合使用有向图描述，而另一些模型很适用于使用无向模型描述。

有向模型和无向模型都有其优点和缺点。这两种方法都不是明显优越和普遍优选的。相反，我们根据具体的每个任务来决定使用哪一种模型。这个选择将部

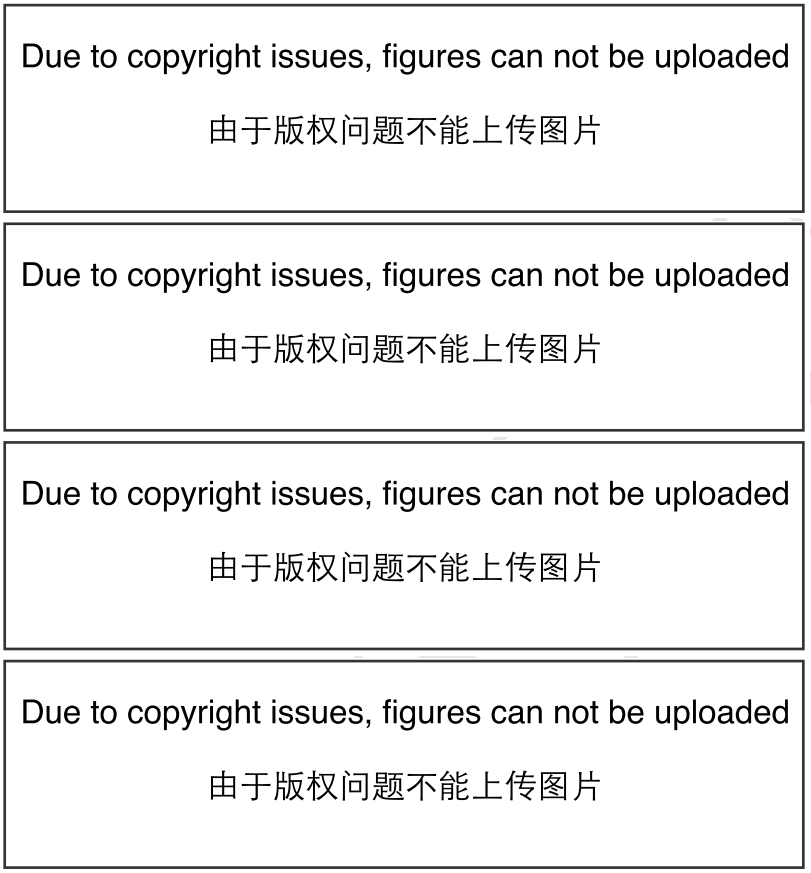


Figure 8.8: TODO

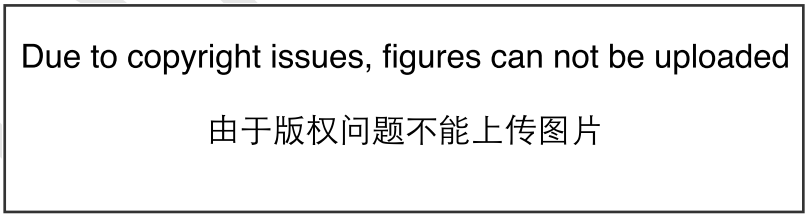


Figure 8.9: TODO

分取决于我们希望描述的概率分布。根据哪种方法可以最大程度的捕捉到概率分布中的独立性，或者哪种方法使用最少的边来描述分布，我们可以决定使用有向建模还是无向建模。还有其他因素可以影响我们决定使用哪种建模方式。即使

在使用单个概率分布时，我们有时可以在不同的建模方式之间切换。有时，如果我们观察到变量的某个子集，或者如果我们希望执行不同的计算任务，换一种建模方式可能更合适。例如，有向模型通常提供了一种高效的从模型中抽取样本(在8.3节中描述）的直接的方法。而无向模型公式通常用于近似推理过程（我们将在第10章中看到，等式(10.56)强调了无向模型的作用)。

每个概率分布可以由有向模型或由无向模型表示。在最坏的情况下，可以使用“完全图”来表示任何分布。在有向模型的情况下，完全图是任何有向无环图，其中我们对随机变量排序，并且每个变量在排序中位于其之前的所有其他变量作为其图中的祖先。对于无向模型，完全图只是一个包含所有变量的团。图8.10给出了一个实例。

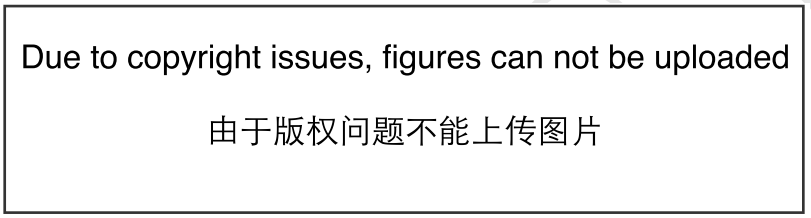


Figure 8.10: TODO

当然，图模型的主要意味着一些变量不直接交互。完全图并不是很有用，因为它并不包含任何独立性。

当我们用图表示概率分布时，我们想要选择一个包含尽可能多的独立性的图，但是并不会假设任何实际上不存在的独立性。

从这个角度来看，一些分布可以使用有向模型更高效地表示，而其他分布可以使用无向模型更高效地表示。换句话说，有向模型可以编码一些无向模型所不能编码的独立性，反之亦然。

有向模型能够使用一种无向模型不能完美地表示的特定类型的子结构。这个子结构被称为**不道德**(immorality)。这种结构出现在当两个随机变量 a 和 b 都是第三个随机变量 c 的父结点，并且不存在任一方向上直接连接 a 和 b 的边的时候。（“不道德”的名字可能看起来很奇怪; 它在图模型文献中被创造为一个关于未婚父母的笑话。）为了将有向模型图 $\mathcal{D}$ 转换为无向模型，我们需要创建一个新图 $\mathcal{U}$。对于每对变量 x 和 y，如果存在连接 x 中的 x 和 y 的有向边（在任一方向上），或者如果 x 和 y 都是图 $\mathcal{D}$ 中另一个变量 z 的父节点，则添加将 x 和 y 连接到 $\mathcal{U}$ 的无向边。得到的图 $\mathcal{U}$ 被称为是**道德图**(moralized graph)。关于一个将有向图模型转化为无向模型的例子可以参见图8.11。

同样的，无向模型可以包括有向模型不能完美表示的子结构。具体来说，如果 $\mathcal{U}$ 包含长度大于 3 的循环，则有向图 $\mathcal{D}$ 不能捕获无向模型$\mathcal{U}$ 所包含的所有条件独立性，除非该循环还包含弦。**环**(loop) 指的是由无向边连接的变量的序列，并且

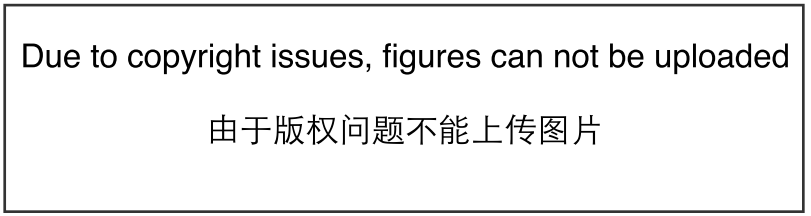


Figure 8.11: TODO

满足序列中的最后一个变量连接回序列中的第一个变量。**弦**(chord) 是定义环的序列中任意两个非连续变量之间的连接。如果 $\mathcal{U}$ 具有长度为 4 或更大的环，并且这些环没有弦，我们必须在将它们转换为有向模型之前添加弦。添加这些弦会丢弃了在 $\mathcal{U}$ 中编码的一些独立信息。通过将弦添加到 $\mathcal{U}$ 形成的图被称为**弦图**(chordal graph) 或者**三角形化图**(triangulated graph)，现在可以用更小的三角形环来描述所有的环。要从弦图构建有向图 $\mathcal{D}$，我们还需要为边指定方向。当这样做时，我们不能在 $\mathcal{D}$ 中创建有向循环，否则将无法定义有效的有向概率模型。为 $\mathcal{D}$ 中的边分配方向的一种方法是对随机变量排序，然后将每个边从排序较早的节点指向稍后排序的节点。一个简单的实例参见图8.12。

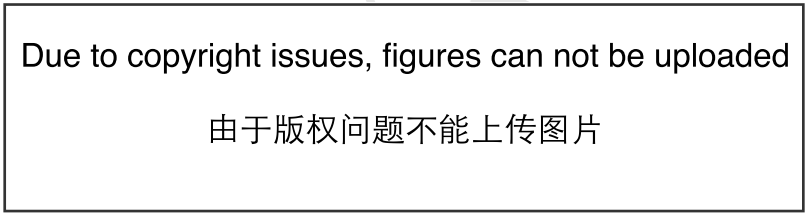


Figure 8.12: TODO

8.2.7 因子图

因子图(factor graph) 是从无向模型中抽样的另一种方法，可以解决无向模型图中表达的模糊性。在无向模型中，每个 ϕ 函数的范围必须是图中某个团的子集。我们无法确定每一个团是否含有一个作用域包含整个团的因子———比如说一个包含三个结点的团可能对应的是一个有三个结点的因子，也可能对应的是三个因子并且每个因子包含了一对结点，这通常会导致模糊性。通过显式地表示每一个 ϕ 函数的作用领域，因子图解决了这种模糊性。具体来说，因子图是由包含无向二分图的无向模型的图形表示。一些节点被绘制为圆形。这些节点对应于随机变量，如在标准无向模型中。其余节点绘制为正方形。这些节点对应于未归一化概率函数的因子ϕ。变量和因子通过无向边连接。当且仅当变量包含在未归一化概率

函数的因子中时，变量和因子在图中连接。没有因子可以连接到图中的另一个因子，也不能将变量连接到变量。图**??**给出了一个例子来说明因子图如何解决无向网络中的模糊性。

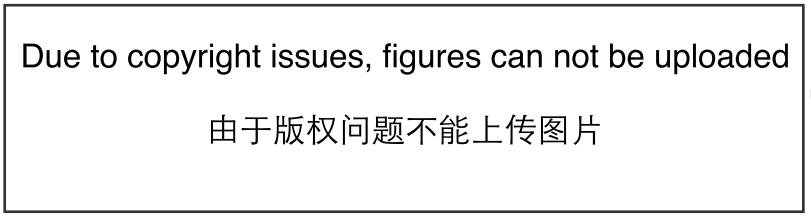


Figure 8.13: TODO

8.3 从图模型中采样

图模型大大简化了从模型中的采样过程。

有向图模型的一个优点是，可以通过一个简单高效的被称作是原始采样的过程从由模型表示的联合分布中抽取样本。

其基本思想是将图中的变量 x_i 使用拓扑排序，使得对于所有 i 和 j，如果 x_i 是 x_j 的父亲结点，则 j 大于 i。然后可以按此顺序对变量进行采样。换句话说，我们可以首先采 $x_1 \sim P(x_1)$，然后采 $x_2 \sim P(x_2|Pa_{\mathcal{G}}(x_2))$，以此类推，直到最后我们采 $x_n \sim P(x_n|Pa_{\mathcal{G}}(x_n))$。只要每个条件分布 $x_i \sim P(x_i|Pa_{\mathcal{G}}(x_i))$ 都很容易从中采样，那么很容易从整个模型中抽样。拓扑排序操作保证我们可以按照方程 (8.1)中的条件分布的顺序依次采样。如果没有拓扑排序，我们可能会尝试在其父节点可用之前对该变量进行抽样。

对于一些图，可能有多个拓扑排序。原始采样可以使用这些拓扑排序中的任何一个。

原始采样通常非常快（假设从每个条件的采样容易）并且方便。

原始采样的一个缺点是其仅适用于有向图模型。另一个缺点是它每次采样并不都是条件采样操作。当我们希望从有向图模型中的变量的子集中抽样时，给定一些其他变量，我们经常要求所有的给定的条件变量比要抽样的变量的顺序要早。在这种情况下，我们可以从模型分布指定的局部条件概率分布进行抽样。否则，我们需要采样的条件分布是给定观测变量的后验分布。这些后验分布在模型中通常没有明确指定和参数化。推断这些后验分布可能是昂贵的。在这种情况下的模型中，原始采样不再有效。

不幸的是，原始采样仅适用于有向模型。我们可以通过将无向模型转换为有向模型来实现从无向模型中抽样，但是这通常需要解决棘手的推理问题（以确定新

有向图的根节点上的边缘分布)，或者需要引入许多边，从而会使得到的有向模型变得难以处理。从无向模型抽样，而不首先将其转换为有向模型的做法似乎需要解决循环依赖的问题。每个变量与每个其他变量相互作用，因此对于抽样过程没有明确的起点。不幸的是，从无向模型模型中抽取样本是一个昂贵的过程。理论上最简单的方法是吉布斯采样。假设我们在随机变量 $\boldsymbol{x}$（一个 n 维向量）上有一个图模型。我们迭代地访问每个变量 x_i，在给定其它变量的条件下从 $p(x_i|x_{-i})$ 中抽样。由于图模型的separation性质，抽取 x_i 的时候我们可以仅对 x_i 的邻居条件化。不幸的是，在我们遍历图模型一次并采样所有 n 个变量之后，我们仍然无法得到一个来自 $p(\boldsymbol{x})$ 的客观样本。相反，我们必须重复该过程并使用它们的邻居的更新的值对所有 n 个变量重新取样。在多次重复之后，该过程渐近地收敛到目标分布。很难确定样本何时达到期望分布的足够精确的近似。无向模型的抽样技术是一个值得深入的研究方向，第9章将对此进行更详细的讨论。

8.4 结构化建模的优势

使用结构化概率模型的主要优点是它们能够显著降低表示概率分布以及学习和推断的成本。有向模型可以加速采样的过程，但是对于无向模型情况则较为复杂。允许所有这些操作使用较少的运行时间和内存的主要机制是选择不对某些变量的相互作用进行建模。图模型构造边来传达信息。在没有边的情况下，模型假设不对变量间的相互作用直接建模。

使用结构化概率模型的一个较小的益处是它们允许我们明确地将给定的现有的知识与知识的学习或者推断分开。这使我们的模型更容易开发和调试。我们可以设计，分析和评估适用于更广范围的学习算法和推理算法。并且我们可以设计能够捕捉到我们认为重要的关系的模型。然后，我们可以组合这些不同的算法和结构，并获得不同可能性的笛卡尔乘积。为每种可能的情况设计端到端算法是困难的。

8.5 学习依赖性关系

良好的生成模型需要准确地捕获所观察到的或“可见”变量 $\boldsymbol{v}$ 上的分布。通常 $\boldsymbol{v}$ 的不同元素彼此高度依赖。在深度学习中，最常用于建模这些依赖性关系的方法是引入几个潜在的或“隐藏”变量 $\boldsymbol{h}$。然后，该模型可以捕获任何对之间的依赖性关系（变量 v_i 和 v_j 间接依赖，v_i 和 $\boldsymbol{h}$ 之间直接依赖，$\boldsymbol{v}$ 和 h_i 直接依赖)。

一个好的不包含任何潜在变量的 $\boldsymbol{v}$ 的模型需要在贝叶斯网络中的每个节点具有非常大量父节点或在马尔可夫网络中具有非常大的团。但是代表这些高阶的交互是昂贵的，首先从计算角度上，存储在存储器中的参数的数量是团中的成员的数量的指数级别，接着在统计学意义上，因为这个指数数量的参数需要大量的数

据来准确估计。

当模型旨在描述具有直接连接的可见变量之间的依赖性关系时，通常不可能连接所有变量，因此设计图模型时需要连接紧密耦合的那些变量，并忽略其他变量之间的作用。机器学习中有一个称为**结构学习**(structure learning) 的领域来专门讨论这个问题。**?** 是一个结构学习的好的参考资料。大多数结构学习技术是基于一种贪婪搜索的形式。它们提出了一种结构，对具有该结构的模型进行训练，然后给出分数。该分数奖励训练集上的高精度并惩罚复杂的模型。然后提出添加或移除少量边的候选结构作为搜索的下一步。搜索向一个预计会增加分数的方向发展。

使用隐变量而不是自适应结构避免了离散搜索和多轮训练的需要。可见变量和隐变量之间的固定结构可以使用可见单元和隐藏单元之间的直接交互，从而使得可见单元之间间接交互。使用简单的参数学习技术，我们可以学习到一个具有固定结构的模型，在边缘分布 $p(\boldsymbol{v})$ 上输入正确的结构。

隐含变量还有一个额外的优势，即能够高效的描述 $p(\boldsymbol{v})$。新变量 $\boldsymbol{h}$ 还提供了 $\boldsymbol{v}$ 的替代表示。例如，如**??**节所示，高斯混合模型学习了一个隐含变量，这个隐含变量对应于特征是从哪一个混合体中抽出。这意味着高斯混合模型中的隐变量可以用于做分类。在7章中，我们看到了简单的概率模型如稀疏编码是如何学习可以用作分类器的输入特征或者作为流形上坐标的隐含变量的。其他模型也可以使用相同的方式，其中具有多种相互作用方式的模型和深层模型可以获得更丰富的输入描述。许多方法通过学习隐含变量来完成特征学习。通常，给定 $\boldsymbol{v}$ 和 $\boldsymbol{h}$，实验观察显示 $\mathbb{E}[\boldsymbol{h}|\boldsymbol{v}]$ 或 $\arg\max_{\boldsymbol{h}} p(\boldsymbol{h}, \boldsymbol{v})$ 都是 $\boldsymbol{v}$ 的良好特征映射。

8.6 推断和近似推断

我们可以使用概率模型的主要方法之一是提出关于变量如何相互关联的问题。给定一组医学测试，我们可以询问患者可能患有什么疾病。在隐含变量模型中，我们可能需要提取能够描述 $\boldsymbol{v}$ 的特征 $\mathbb{E}[\boldsymbol{h}|\boldsymbol{v}]$。有时我们需要解决这些问题来执行其他任务。我们经常使用最大似然估计来训练我们的模型。由于

$$\log p(\boldsymbol{v}) = \mathbb{E}_{\boldsymbol{h}\sim p(\boldsymbol{h}|\boldsymbol{v})}[\log p(\boldsymbol{h}, \boldsymbol{v}) - \log p(\boldsymbol{h}|\boldsymbol{v})] \tag{8.9}$$

学习过程中，我们经常想要计算 $p(\boldsymbol{h}|\boldsymbol{v})$。所有这些都是推理问题的例子，其中我们必须预测给定其他变量的情况下一些变量的值，或者在给定其他变量的值的情况下预测一些变量的概率分布。

不幸的是，对于大多数有趣的深层模型，这些推理问题都是难以处理的，即使我们使用结构化的图模型来简化它们。图结构允许我们用合理数量的参数来表示复杂的高维分布，但是用于深度学习的图模型并不满足这样的条件，从而难以实现高效的推理。

可以直接看出，计算一般图模型的边缘概率是 # P-hard 的。复杂性类别 # P 是复杂性类别 NP 的泛化。NP 中的问题仅仅需要确定问题是否有解决方案，并找到解决方案（如果存在）。# P 中的问题需要计算解决方案的数量。要构建最坏情况的图模型，想象一下我们在 3-SAT 问题中定义了二进制变量的图模型。我们可以对这些变量施加均匀分布。然后我们可以为每个子句添加一个二进制隐变量，来表示每个子句是否得到满足。然后，我们可以添加另一个隐含变量，来表示是否满足所有子句。这可以通过构造一个隐含变量的缩减树来完成，树中的每个结点表示其它两个变量是否满足，从而不需要构造一个大的团。该树的叶是每个子句的变量。树的根报告整个问题是否满足。由于 literal 上的均匀分布，缩减树的跟结点的边缘分布表示多少比例的分配能够使得该问题成立。虽然这是一个设计的最坏情况的例子，NP-hard 图通常出现在实际现实世界的场景。

这促使我们使用近似推理。在深度学习中，这通常涉及变分推理，其中通过寻求尽可能接近真实分布的近似分布 $q(\boldsymbol{h}|\boldsymbol{v})$ 来逼近真实分布 $p(\boldsymbol{h}|\boldsymbol{v})$。这个技术在第10章中有深入的描述。

8.7 结构化概率模型的深度学习方法

深度学习实践者通常使用与从事结构化概率模型研究的机器学习研究者相同的基本计算工具。然而，在深度学习中，我们通常对如何组合这些工具做出不同的设计，导致总体算法和模型与更传统的图模型具有非常不同的风格。

深度学习并不总是涉及特别深的图模型。在图模型中，我们可以根据图模型的图而不是图计算来定义模型的深度。我们可以认为潜在变量 h_j 处于深度 j，如果从 h_i 到观察到的最短路径变量是 j 步。我们通常将模型的深度描述为任何这样的 h_j 的最大深度。这种深度不同于由图计算定义的深度。用于深度学习的许多生成模型没有隐含变量或只有一层隐含变量，但使用深度计算图来定义模型中的条件分布。

深度学习基本上总是利用分布式表示的思想。即使是用于深度学习目的的浅层模型（例如预训练浅层模型，稍后将形成深层模型），也几乎总是具有单个大的潜在变量层。深度学习模型通常具有比观察到的变量更多的潜变量。复杂的变量之间的非线性交互通过多个隐含变量的间接连接来实现。

相比之下，传统的图模型通常包含偶尔观察到的变量，即使一些训练样本中的许多变量随机丢失。传统模型大多使用高阶项和结构学习来捕获变量之间复杂的非线性相互作用。如果有潜在变量，它们通常数量很少。

隐变量的设计方式在深度学习中也有所不同。深度学习实践者通常不希望隐含变量提前采用任何特定的含义，从而训练算法可以自由地开发它需要建模的适用于特定的数据集的概念。在事后解释潜在变量通常是很困难的，但是可视化技术可以允许它们表示的一些粗略表征。当隐含变量在传统图模型中使用时，它们

通常被赋予具有一些特定含义，比如文档的主题，学生的智力，导致患者症状的疾病等。这些模型通常由研究者解释，并且通常具有更多的理论保证，但是不能扩展到复杂的问题，并且不能在与深度模型一样多的不同背景中重复使用。

另一个明显的区别是深度学习方法中经常使用的连接类型。深度图模型通常具有大的与其它单元组全连接的单元组，使得两个组之间的交互可以由单个矩阵描述。传统的图模型具有非常少的连接，并且每个变量的连接的选择可以单独设计。模型结构的设计与推理算法的选择紧密相关。图模型的传统方法通常旨在保持精确推断的可追踪性。当这个约束太强的时候，我们可以采用一种流行的被称为是**环状信念传播**(loopy belief propagation) 的近似推理算法。这两种方法通常在连接非常稀疏的图上有很好的效果。相比之下，在深度学习中使用的模型倾向于将每个可见单元 v_i 连接到非常多的隐藏单元 h_j 上，从而使得 $\boldsymbol{h}$ 可以获得一个 v_i 的分布式表示 (也可能是其他几个可观察变量)。分布式表示具有许多优点，但是从图模型和计算复杂性的观点来看，分布式表示有一个缺点就是对于精确推断和循环信任传播等传统技术来说不能产生足够稀疏的图。结果，图模型和深度图模型的最大的区别之一就是深度学习中从来不会使用环状信念传播。相反的，许多深度学习模型可以用来加速吉布斯采样或者变分推断。此外，深度学习模型包含了大量的隐含变量，使得高效的数值计算代码显得格外重要。除了选择高级推理算法之外，这提供了另外的动机，用于将结点分组成层，相邻两层之间用一个矩阵来描述相互作用。这要求实现算法的各个步骤具有高效的矩阵乘积运算，或者专门适用于稀疏连接的操作，例如块对角矩阵乘积或卷积。

最后，图模型的深度学习方法的一个主要特征在于对未知量的较高容忍度。与简化模型直到它的每一个量都可以被精确计算不同的是，我们让模型保持了较高的自由度，以增强模型的威力。我们一般使用边缘分布不能计算但是可以简单的从中采样的模型。我们经常训练具有难以处理的目标函数的模型，我们甚至不能在合理的时间内近似，但是如果我们能够高效地获得这样的函数的梯度的估计，我们仍然能够近似训练模型。深度学习方法通常是找出我们绝对需要的最小量的信息，然后找出如何尽快得到该信息的合理近似。

8.7.1 实例：受限玻耳兹曼机

受限玻耳兹曼机(Restricted Boltzmann Machine)(**?**) 或者 **harmonium**(harmonium) 是图模型如何用于深度学习的典型例子。RBM本身不是一个深层模型。相反，它有一层隐含变量，可用于学习输入的表示。在第**??**章中，我们将看到RBM如何被用来构建许多的深层模型。在这里，我们举例展示了RBM在许多深度图模型中使用的许多实践：它的结点被分成层，层之间的连接由矩阵描述，连通性相对密集。该模型能够进行高效的吉布斯采样，并且模型设计的重点在于以很高的自由度来学习隐含变量，而不是设计师指定。之后在**??**节，我们将更详细地再次讨论RBM。

规范的RBM是具有二进制的可见和隐藏单元的基于能量的模型。其能量函数为

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\boldsymbol{b}^\top \boldsymbol{v} - \boldsymbol{c}^\top \boldsymbol{h} - \boldsymbol{v}^\top \boldsymbol{W}\boldsymbol{h} \tag{8.10}$$

其中 $\boldsymbol{b}, \boldsymbol{c}$ 和 $\boldsymbol{W}$ 都是无限制的实值的可学习参数。我们可以看到，模型被分成两组单元：$\boldsymbol{v}$ 和 $\boldsymbol{h}$，它们之间的相互作用由矩阵 $\boldsymbol{W}$ 来描述。该模型在图8.14中图示。如该图所示，该模型的一个重要方面是在任何两个可见单元之间或任何两个隐藏单元之间没有直接的相互作用（因此称为“受限”，一般的玻耳兹曼机可以具有任意连接）。

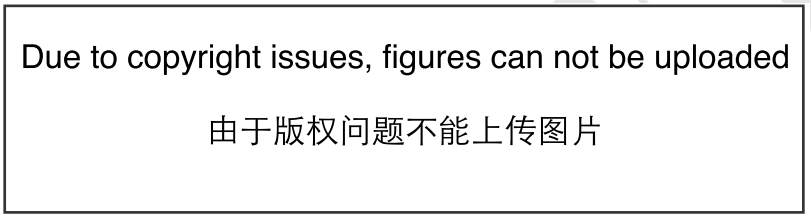


Figure 8.14: TODO

对RBM结构的限制产生了好的属性

$$p(\boldsymbol{h}|\boldsymbol{v}) = \prod_i p(h_i|\boldsymbol{v}) \tag{8.11}$$

以及

$$p(\boldsymbol{v}|\boldsymbol{h}) = \prod_i p(v_i|\boldsymbol{h}) \tag{8.12}$$

独立的条件分布很容易计算。对于二元的受限玻耳兹曼机，我们可以得到：

$$\begin{aligned} p(h_i = 1|\boldsymbol{v}) &= \sigma\left(\boldsymbol{v}^\top \boldsymbol{W}_{:,i} + b_i\right) \\ p(h_i = 0|\boldsymbol{v}) &= 1 - \sigma\left(\boldsymbol{v}^\top \boldsymbol{W}_{:,i} + b_i\right) \end{aligned} \tag{8.13}$$

结合这些属性可以得到有效的块吉布斯采样，它在同时采样所有 $\boldsymbol{h}$ 和同时采样所有 $\boldsymbol{v}$ 之间交替。RBM模型通过吉布斯采样产生的样本在图8.15中。

由于能量函数本身只是参数的线性函数，很容易获取能量函数的导数。例如，

$$\frac{\partial}{\partial W_{i,j}} E(\boldsymbol{v}, \boldsymbol{h}) = -v_i h_j \tag{8.14}$$

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 8.15: TODO

这两个属性，高效的吉布斯采样和导数计算，使训练过程非常方便。在第??章中，我们将看到，可以通过计算应用于来自模型的样本的这种导数来训练无向模型。

训练模型可以得到数据 $\boldsymbol{v}$ 的表示 $\boldsymbol{h}$。我们可以经常使用 $\mathbb{E}_{\boldsymbol{h} \sim p(\boldsymbol{h}|\boldsymbol{v})}[\boldsymbol{h}]$ 作为一组描述 $\boldsymbol{v}$ 的特征。

总的来说，RBM展示了典型的图模型的深度学习方法：结合由矩阵参数化的层之间的高效相互作用通过隐含变量层完成表示学习。

图模型的语言为描述概率模型提供了一种优雅，灵活和清晰的语言。在前面的章节中，我们使用这种语言，以及其他视角来描述各种各样的深概率模型。

Chapter 9

蒙特卡罗方法

随机算法可以粗略的分为两类：Las Vegas算法和蒙特卡罗算法。Las Vegas算法通常精确地返回一个正确答案（或者发布一个失败报告）。这类方法通常需要占用随机量的计算资源（通常包括内存和运行时间）。与此相对的，蒙特卡罗方法返回一个伴随着随机量错误的答案。花费更多的计算资源（通常包括内存和运行时间）可以减少这种随机量的错误。在任意的固定的计算资源下，蒙特卡罗算法可以得到一个近似解。

对于机器学习中的许多问题来说，我们很难得到精确的答案。这类问题很难用精确的确定性的算法如Las Vegas算法解决。取而代之的是确定性的近似算法或蒙特卡罗方法。这两种方法在机器学习中都非常普遍。本章主要关注蒙特卡罗方法。

9.1 采样和蒙特卡罗方法

机器学习中的许多工具是基于从某种分布中采样以及用这些样本对目标量做一个蒙特卡罗估计。

9.1.1 为什么需要采样？

我们希望从某个分布中采样存在许多理由。当我们需要以较小的代价近似许多项的和或某个积分时采样是一种很灵活的选择。有时候，我们使用它加速一些很费时却易于处理的和的估计，就像我们使用minibatch对整个训练成本进行subsample 的情况。在其他情况下，我们需要近似一个难以处理的和或积分，例如估计一个无向模型的分割函数的对数的梯度。在许多其他情况下，抽样实际上是我们的目标，就像我们想训练一个可以从训练分布采样的模型。

9.1.2 蒙特卡罗采样的基础

当无法精确计算和或积分（例如，和具有指数数量个项，且无法被精确简化）时，通常可以使用蒙特卡罗采样。这种想法把和或者积分视作某分布下的期望，然后通过估计对应的平均值来近似这个期望。令

$$s = \sum_{\boldsymbol{x}} p(\boldsymbol{x}) f(\boldsymbol{x}) = E_p[f(\boldsymbol{x})] \tag{9.1}$$

或者

$$s = \int p(\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x} = E_p[f(\boldsymbol{x})] \tag{9.2}$$

为我们所需要估计的和或者积分，写成期望的形式，p 是一个关于随机变量 $\boldsymbol{x}$ 的概率分布（求和时）或者概率密度函数（求积分时）。

我们可以通过从 p 中采集 n 个样本 $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}$ 来近似 s 并得到一个经验平均值

$$\hat{s}_n = \frac{1}{n} \sum_{i=1}^{n} f(\boldsymbol{x}^{(i)}). \tag{9.3}$$

这种近似可以被证明拥有如下几个性质。首先很容易观察到 $\hat{s}$ 这个估计是无偏的，由于

$$\mathbb{E}[\hat{s}_n] = \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}[f(\boldsymbol{x}^{(i)})] = \frac{1}{n} \sum_{i=1}^{n} s = s \tag{9.4}$$

此外，根据**大数定理**(Law of large number)，如果样本 $\boldsymbol{x}^{(i)}$ 独立且服从同一分布，那么其平均值几乎必然收敛到期望值，即

$$\lim_{n \to \infty} \hat{s}_n = s \tag{9.5}$$

只需要满足各个单项的方差，即 $\mathrm{Var}[f(\boldsymbol{x}^{(i)})]$ 有界。详细地说，我们考虑当 n 增大时 $\hat{s}_n$ 的方差。只要 $\mathrm{Var}[f(\boldsymbol{x}^{(i)})] < \infty$，方差 $\mathrm{Var}[f(\boldsymbol{x}^{(i)})]$ 减小并收敛到 0：

$$\mathrm{Var}[\hat{s}_n] = \frac{1}{n^2} \sum_{i=1}^{n} \mathrm{Var}[f(\boldsymbol{x})] \tag{9.6}$$

$$= \frac{\mathrm{Var}[f(\boldsymbol{x})]}{n}. \tag{9.7}$$

这个简单的结果启迪我们如果估计蒙特卡罗均值中的不确定性或者说是蒙特卡罗估计的期望误差。我们计算了 $f(\boldsymbol{x}^{(i)})$ 的经验均值和方差，[1] 然后将估计的方差除以样本数 n 来得到 $\mathrm{Var}[\hat{s}_n]$ 的估计。**中心极限定理**(central limit theorem) 告诉我们 $\hat{s}_n$ 的分布收敛到以 s 为均值以 $\frac{\mathrm{Var}[f(\boldsymbol{x})]}{n}$ 为方差的正态分布。这使得我们可以利用正态分布的累积密度函数来估计 $\hat{s}_n$ 的置信区间。

以上的所有结论都依赖于我们可以从基准分布 $p(\boldsymbol{x})$ 中轻易的采样，但是这个假设并不是一直成立的。当我们无法从 p 中采样时，一个备选方案是用重要采样，在 Sec 9.2会讲到。一种更加通用的方式是采样一系列的趋近于目标分布的估计。这就是马尔可夫链蒙特卡罗方法（见 Sec 9.3）。

9.2 重要采样

如方程 (9.2)所示，在蒙特卡罗方法求积分（或者求和）的过程中，确定积分的哪一部分作为概率分布 $p(\boldsymbol{x})$，哪一部分作为被积的函数 $f(\boldsymbol{x})$（我们感兴趣的是估计 $f(\boldsymbol{x})$ 在概率分布 $p(\boldsymbol{x})$ 下的期望）是很关键的一步。$p(\boldsymbol{x})f(\boldsymbol{x})$ 存在不唯一的分解因为它通常可以被写成

$$p(\boldsymbol{x})f(\boldsymbol{x}) = q(\boldsymbol{x})\frac{p(\boldsymbol{x})f(\boldsymbol{x})}{q(\boldsymbol{x})}, \tag{9.8}$$

在这里，我们从 q 分布中采样，然后估计 $\frac{pf}{q}$ 在此分布下的均值。许多情况中，给定 p 和 f 的情况下我们希望计算某个期望，这个问题既然是求期望，那么 p 和 f 最好是很常见的分解。然而，从衡量一定采样数所达到的精度的角度说，原始定义的问题通常不是最优的选择。幸运的是，最优的选择 q^* 通常可以简单推出。这种最优的采样函数 q^* 对应的是最优重要采样。

从方程 9.8中所示的关系中可以发现，任意蒙特卡罗估计

$$\hat{s}_p = \frac{1}{n}\sum_{i=1,\boldsymbol{x}^{(i)}\sim p}^{n} f(\boldsymbol{x}^{(i)}) \tag{9.9}$$

可以被转化为一个重要采样的估计

$$\hat{s}_q = \frac{1}{n}\sum_{i=1,\boldsymbol{x}^{(i)}\sim q}^{n} \frac{p(\boldsymbol{x}^{(i)})f(\boldsymbol{x}^{(i)})}{q(\boldsymbol{x}^{(i)})} \tag{9.10}$$

我们可以容易的发现估计值的期望与 q 分布无关

$$\mathbb{E}_q[\hat{s}_q] = \mathbb{E}_p[\hat{s}_p] = s \tag{9.11}$$

[1] 计算无偏估计的方差时，更倾向于用计算偏差平方和除以 $n-1$ 而非 n。

然而，重要采样的方差却对不同的 q 的选取非常敏感。这个方差可以表示为

$$\mathrm{Var}[\hat{s}_q] = \mathrm{Var}[\frac{p(\boldsymbol{x})f(\boldsymbol{x})}{q(\boldsymbol{x})}]/n \tag{9.12}$$

当 q 取到

$$q^*(\boldsymbol{x}) = \frac{p(\boldsymbol{x})|f(\boldsymbol{x})|}{Z} \tag{9.13}$$

时，方差达到最小值。

在这里 Z 表示归一化常数，Z 的选择使得 $q*(\boldsymbol{x})$ 之和或者积分为 1。一个好的重要采样分布会把更多的权重放在被积函数较大的地方。事实上，当 $f(\boldsymbol{x})$ 为一个常数的时候，$\mathrm{Var}[\hat{s}_{q^*}] = 0$，这意味着当使用最优的 q 分布的时候，只需要采一个样本就够了。当然，这仅仅是因为计算 q^* 的时候已经解决了所有的问题。所以这种只需要采一个样本的方法往往是实践中无法实现的。

对于重要采样来说任意 q 分布都是可行的（从得到一个期望上正确的值的角度来说），q^* 指的是最优的 q 分布。从 q^* 中采样往往是不可行的，所以总是采样其它的选择来降低方差。

另一种方法是采用**有偏重要采样**(biased importance sampling)，这种方法有一个优势并不需要归一化的 p 或 q 分布。在处理离散变量的时候，有偏重要采样估计可以表示为

$$\hat{s}_{\mathrm{BIS}} = \frac{\sum_{i=1}^{n} \frac{p(\boldsymbol{x}^{(i)})}{q(\boldsymbol{x}^{(i)})} f(\boldsymbol{x}^{(i)})}{\sum_{i=1}^{n} \frac{p(\boldsymbol{x}^{(i)})}{q(\boldsymbol{x}^{(i)})}} \tag{9.14}$$

$$= \frac{\sum_{i=1}^{n} \frac{p(\boldsymbol{x}^{(i)})}{\tilde{q}(\boldsymbol{x}^{(i)})} f(\boldsymbol{x}^{(i)})}{\sum_{i=1}^{n} \frac{p(\boldsymbol{x}^{(i)})}{\tilde{q}(\boldsymbol{x}^{(i)})}} \tag{9.15}$$

$$= \frac{\sum_{i=1}^{n} \frac{\tilde{p}(\boldsymbol{x}^{(i)})}{\tilde{q}(\boldsymbol{x}^{(i)})} f(\boldsymbol{x}^{(i)})}{\sum_{i=1}^{n} \frac{\tilde{p}(\boldsymbol{x}^{(i)})}{\tilde{q}(\boldsymbol{x}^{(i)})}} \tag{9.16}$$

其中 $\tilde{p}$ 和 $\tilde{q}$ 分别是分布 p 和 q 的未经归一化的形式，$x^{(i)}$ 是从分布 q 中采集的样本。这种估计是有偏的因为 $\mathbb{E}[\hat{s}_{\mathrm{BIS}}] \neq s$，除了当 n 渐进性地趋近于 ∞ 时，方程 (9.14)的分母收敛到 1。所以这种估计也被叫做的渐进性无偏的。

尽管一个好的 q 分布的选择可以显著的提高蒙特卡罗估计的效率，反之一个糟糕的 q 分布的选择则会使效率更糟糕。我们回过头来看看方程 (9.12)会发现，如果存在一个 q 使得 $\frac{p(\boldsymbol{x})f(\boldsymbol{x})}{q(\boldsymbol{x})}$ 很大，那么这个估计的方差也会很大。当 $q(\boldsymbol{x})$ 很小，

而 $f(\boldsymbol{x})$ 和 $p(\boldsymbol{x})$ 都较大并且无法抵消 q 时，这种情况会非常明显。q 分布经常会取一些简单常用的分布使得我们能够从 q 分布中容易的采样。当 $\boldsymbol{x}$ 是高维数据的时候，q 分布的简单性使得它很难于 p 或者 $p|f|$ 相匹配。当 $q(x^{(i)}) \gg p(x^{(i)})|f(x^{(i)})|$ 的时候，重要采样采到了很多无用的样本（权值之和很小或趋于零）。另一方面，当 $q(x^{(i)}) \ll p(x^{(i)})|f(x^{(i)})|$ 的时候，样本会很少被采到，其对应的权值却会非常大。正因为后一个事件是很少发生的，这些样本很难被采到，通常使得对 s 的估计出现了估计不足，于之前过度估计相对。这样的不均匀情况在高维数据屡见不鲜，因为在高维度分布中联合分布的动态域通常非常大。

尽管存在上述的风险，但是重要采样及其变种在机器学习中的应用中仍然扮演着重要的角色，包括深度学习问题。比方说，重要采样被应用于加速训练大规模词汇神经网络的过程中（见**??**章）或者其它有着大量输出结点的神经网络中。此外，还可以看到重要采样应用于估计分割函数（一个概率分布的归一化常数）的过程中（详见第**??**章）以及在深度有向图模型比如变分自编码中估计似然函数的对数（详见第**??**章）。采用随机梯度下降训练模型参数的时候重要采样可以用来改进对代价函数的梯度的估计，尤其是针对于分类器模型的训练中一小部分错误分类样本产生的代价函数。在这种情况下更加频繁的采集这些困难的样本可以降低梯度的估计的方差 (**?**)。

9.3 马尔可夫链蒙特卡罗方法

在许多实例中，我们希望采用蒙特卡罗方法，然而往往又不存在一种直接从目标分布 $p_{\text{model}}(\boldsymbol{x})$ 中精确采样或者一个好的（方差较小的）重要采样分布 $q(\boldsymbol{x})$。在深度学习的例子中，分布 $p_{\text{model}}(\boldsymbol{x})$ 往往表达成一个无向的模型。在这种情况下，为了从分布 $p_{\text{model}}(\boldsymbol{x})$ 中近似采样，我们引入了一种数学工具，它被命名为**马尔可夫链**(Markov Chain)。利用马尔可夫链来进行蒙特卡罗估计的这一类算法被称为**马尔可夫链蒙特卡罗**(Markov Chain Monte Carlo) 算法。马尔可夫链蒙特卡罗算法在机器学习中的应用在 (**?**) 中花了大量篇幅描述。马尔可夫链蒙特卡罗算法的最标准的要求是只适用模型分布处处不为 0 的情况。因此，最方便的目标分布的表达是从**基于能量的模型**(Energy-based model) 即 $p(\boldsymbol{x}) \propto \exp(-E(\boldsymbol{x}))$ 中采样，参见8.2.4节。在基于能量的模型中每一个状态所对应的概率都不为零。事实上马尔可夫链蒙特卡罗方法可以被广泛的应用在了许多包含概率为 0 的状态的概率分布中。然而，在这种情况下，关于马尔可夫链蒙特卡罗方法性能的理论保证只能依据具体情况具体分析。在深度学习中，应用于基于能量的模型的通用的理论分析是很常见的。

为了解释从基于能量的模型中采样的困难性，我们考虑一个包含两个变量的基于能量的模型的例子，记作 $p(a, b)$。为了采 a，我们必须先从 $p(a|b)$ 中采样，为了采 b，我们又必须从 $p(b|a)$ 中采样。这似乎成了难以解释的先有鸡还是先有蛋的问题。有向模型避免了这一问题因为它的图是有向无环的。为了以拓扑顺序

完成对包含各个变量的一个样本的**原始采样**(Ancestral Sampling)，给定每个变量的所有父结点的条件下，这个变量是确定能够被采样的（详见8.3)。原始采样定义了一种高效的，单路径的方法来采集一个样本。

在基于能量的模型中，我们通过使用马尔可夫链来采样，从而避免了先有鸡还是先有蛋的问题。马尔可夫链的核心思想是以一个任意状态的点 $\boldsymbol{x}$ 作为起始点。随着时间的推移，我们随机的反复的更新状态 $\boldsymbol{x}$。最终 $\boldsymbol{x}$ 成为了一个从 $p(\boldsymbol{x})$ 中抽出的（接近）比较公正的样本。在正式的定义中，马尔可夫链由一个随机状态 x 和一个转移分布 $T(\boldsymbol{x}'|\boldsymbol{x})$ 定义而成，$T(\boldsymbol{x}'|\boldsymbol{x})$ 是一个概率分布，说明了给定状态 $\boldsymbol{x}$ 的情况下随机的转移到 $\boldsymbol{x}'$ 的概率。运行一个马尔可夫链意味着根据转移分布 $T(\boldsymbol{x}'|\boldsymbol{x})$ 反复的用状态 $\boldsymbol{x}'$ 来更新状态 $\boldsymbol{x}$。

为了给出马尔可夫链蒙特卡罗方法的一些理论解释，重定义这个问题是很有用的。首先我们关注一些简单的情况，其中随机变量 $\boldsymbol{x}$ 有可数个状态。我们将这种状态记作正整数 x。不同的整数 x 的大小对应着原始问题中 $\boldsymbol{x}$ 的不同状态。

接下来我们考虑如果并行的运行无穷多个马尔可夫链会发生什么。不同马尔可夫链的所有的状态都会被某一个分布 $q^{(t)}(x)$ 采到，在这里 t 表示消耗的时间数。开始时，对每个马尔可夫链，我们采用一个分布 q^0 来任意的初始化 x。之后，$q^{(t)}$ 与所有之前跑过的马尔可夫链有关。我们的目标就是 $q^{(t)}(x)$ 收敛到 $p(x)$。

因为我们已经用正整数 x 重定义了这个问题，我们可以用一个向量 $\boldsymbol{v}$ 来描述这个概率分布 q，并且满足

$$q(x = i) = v_i \tag{9.17}$$

然后我们考虑从状态 x 到状态 x' 更新单一的马尔可夫链。单一状态转移到 x' 的概率可以表示为

$$q^{(t+1)}(x') = \sum_x q^{(t)}(x) T(x'|x) \tag{9.18}$$

根据状态为整数的设定，我们可以将转移算子 T 表示成一个矩阵 $\boldsymbol{A}$。矩阵 $\boldsymbol{A}$ 的定义如下：

$$\boldsymbol{A}_{i,j} = T(\boldsymbol{x}' = i|\boldsymbol{x} = j) \tag{9.19}$$

使用这一定义，我们可以重新写成方程9.18。与之前使用 q 和 T 来理解单个状态的更新相对的是，我们现在可以使用 $\boldsymbol{v}$ 和 $\boldsymbol{A}$ 来描述当我们更新时（并行运行的）不同个马尔可夫链上的整个分布是如何变化的：

$$\boldsymbol{v}^{(t)} = \boldsymbol{A}\boldsymbol{v}^{(t-1)} \tag{9.20}$$

重复的使用马尔可夫链来更新就相当于重复的乘上矩阵 $\boldsymbol{A}$。换一句话说，我们可以认为这一过程就是 $\boldsymbol{A}$ 的指数变化：

$$\boldsymbol{v}^{(t)} = \boldsymbol{A}^t \boldsymbol{v}^{(0)} \tag{9.21}$$

矩阵 $\boldsymbol{A}$ 有一种特殊的结构，因为它的每一列都代表了一个概率分布。这样的矩阵被称作是**随机矩阵**(Stochastic Matrix)。对于任意状态 x 到任意其它状态 x' 存在一个 t 使得转移概率不为 0，那么 Perron-Frobenius 定理 (**??**) 可以保证这个矩阵的最大特征值是实数且大小为 1。我们可以看到所有的特征值随着时间呈现指数变化：

$$\boldsymbol{v}^{(t)} = (\boldsymbol{V}\text{diag}(\lambda)\boldsymbol{V}^{-1})^t\boldsymbol{v}^{(0)} = \boldsymbol{V}\text{diag}(\lambda)^t\boldsymbol{V}^{-1}\boldsymbol{v}^{(0)} \tag{9.22}$$

这个过程导致了所有的不等于 1 的特征值都衰减到 0。在一些较为宽松的假设下，我们可以保证矩阵 $\boldsymbol{A}$ 只有一个特征值为 1。所以这个过程收敛到**静态分布**(Stationary Distribution)，有时也叫做**平衡分布**(Equilibrium Distribution)。收敛时，我们得到

$$\boldsymbol{v}' = \boldsymbol{A}\boldsymbol{v} = \boldsymbol{v} \tag{9.23}$$

这个条件也适用于收敛之后的每一步。这就是特征向量方程。作为收敛的静止点，$\boldsymbol{v}$ 一定是特征值为 1 所对应的特征向量。这个条件保证了收敛到了静态分布以后，之后的采样过程不会改变不同的马尔可夫链的状态的分布 (尽管转移算子不会改变每个单独的状态)。

如果我们正确的选择了转移算子 T，那么最终的静态分布q 将会等于我们所希望采样的分布 p。我们会简要的介绍如何选择 T，详见9.4。

可数状态马尔可夫链的大多数的性质可以被推广到连续状态的马尔可夫链中。在这种情况下，一些研究者把马尔可夫链叫做**哈里斯链**(Harris Chain)，但是这两种情况我们都用马尔可夫链来表示。通常情况下，在一些宽松的条件下，一个带有转移算子 T 的马尔可夫链都会收敛到一个固定点，这个固定点可以写成如下形式：

$$q'(\boldsymbol{x}') = \mathbb{E}_{\boldsymbol{x}\sim q}T(\boldsymbol{x}'|\boldsymbol{x}) \tag{9.24}$$

这个方程的离散版本就是方程 (9.23)。当 $\boldsymbol{x}$ 离散时，这个期望对应着求和，而当 $\boldsymbol{x}$ 连续时，这个期望对应的是积分。

无论状态是连续还是离散，所有的马尔可夫链方法都包括了重复，随机的更新直到最终所有的状态开始从平衡分布采样。运行马尔可夫链直到它达到平衡分布的过程通常被叫做马尔可夫链的**预烧**(Burning-in) 过程。在马尔可夫链达到平衡分布之后，我们可以从平衡分布中采集一个无限多数量的样本序列。这些样本服从同一分布，但是两个连续的样本之间存在强烈的相关性。所以一个有限的序列无法完全表达平衡分布。一种解决这个问题的方法是每隔 n 个样本返回一个样本，从而使得我们对于平衡分布的统计量的估计不会被马尔可夫链蒙特卡罗方法的样本之间的相关性所干扰。所以马尔可夫链在计算上是非常昂贵的，主要源于需要预烧的时间以及在达到平衡分布之后从一个样本转移到另一个完全无关

的样本所需要的时间。如果我们想要得到完全独立的样本，那么我们需要同时并行的运行多个马尔可夫链。这种方法使用了额外的并行计算来消除潜在因素的干扰。使用一条马尔可夫链来生成所有样本的策略和每条马尔可夫链只产生一个样本的策略是两种极端。深度学习的研究者们通常选取的马尔可夫链的数目和minibatch中的样本数相近，然后从这些马尔可夫链的集合中采集所需要的样本。常用的马尔可夫链的数目通常选为 100。

另一个难点是我们无法预先知道马尔可夫链需要运行多少步才能到达平衡分布。这段时间通常被称为**混合时间**(Mixing Time)。检测一个马尔可夫链是否达到平衡是很困难的。我们并没有足够完善的理论来解决这个问题。理论只能保证马尔可夫链会最终收敛，但是无法保证其它。如果我们从矩阵 $\boldsymbol{A}$ 作用在概率向量 $\boldsymbol{v}$ 的角度来分析马尔可夫链，那么我们可以发现当 $\boldsymbol{A}^t$ 除了 1 以外的特征值都趋于 0 的时候，马尔可夫链收敛到了平衡分布。这也意味着矩阵 $\boldsymbol{A}$ 的第二大的特征值决定了马尔可夫链的混合时间。然而，在实践中，我们通常不能将马尔可夫链表示成为矩阵的形式。我们的概率模型所能够达到的状态是变量数的指数级别，所以表达 $\boldsymbol{v}$，$\boldsymbol{A}$ 或者 $\boldsymbol{A}$ 的特征值是不现实的。由于这些阻碍，我们通常无法知道马尔可夫链是否以及收敛。作为替代，我们只能运行足够长时间马尔可夫链直到我们粗略估计是足够的，然后使用启发式的方法来决定马尔可夫链是否收敛。这些启发性的算法包括了手动检查样本或者衡量连续样本之间的相关性。

9.4 吉布斯采样

截至目前我们已经了解了如何通过反复的更新 $\boldsymbol{x} \leftarrow \boldsymbol{x}' \sim T(\boldsymbol{x}'|\boldsymbol{x})$ 从一个分布 $q(\boldsymbol{x})$ 中采样。然而我们还没有提到过如何确定一个有效的 $q(\boldsymbol{x})$ 分布。本书中描述了两种基本的方法。第一种方法是从已经学习到的分布 p_{model} 中推导出 T，就像下文描述的从基于能量的模型中采样的案例一样。第二种方法是直接用参数描述 T，然后学习这些参数，其静态分布隐式地定义了我们所感兴趣的模型 p_{model}。我们将在第**??**和第**??**中讨论第二种方法的例子。

在深度学习中，我们通常使用马尔可夫链从定义为基于能量的模型的分布 $p_{\text{model}}(\boldsymbol{x})$ 中采样。在这种情况下，我们希望马尔可夫链的 $q(\boldsymbol{x})$ 分布就是 $p_{\text{model}}(\boldsymbol{x})$。为了得到所期望的 $q(\boldsymbol{x})$ 分布，我们必须选取合适的 $T(\boldsymbol{x}'|\boldsymbol{x})$。

吉布斯采样(Gibbs Sampling) 是一种概念简单而又有效的方法来构造一个从 $p_{\text{model}}(\boldsymbol{x})$ 中采样的马尔可夫链，其中在基于能量的模型中从 $T(\boldsymbol{x}'|\boldsymbol{x})$ 采样是通过选择一个变量 x_i 然后从 p_{model} 中的该点在无向图 G 中邻接点的条件分布中抽样。给定他们的邻居结点只要一些变量是条件独立的，那么这些变量可以被同时采样。正如在8.7.1中看到的受限玻耳兹曼机的例子一样，受限玻耳兹曼机所有的隐层结点可以被同时采样，因为在给定可见层结点的条件下他们相互条件独立。同样的，所有的可见层结点也可以被同时采样因为在给定隐藏层结点的情况下他们相互条件独立。像这样的同时更新许多变量的吉布斯采样通常被叫做**块吉布斯**

采样(block Gibbs Sampling)。

设计从 p_{model} 中采样的马尔可夫链还存在另外的备选方法。比如说，Metropolis-Hastings 算法在其它情景下被广泛使用。在深度学习的无向模型中，除了吉布斯采样很少使用其它的方法。改进采样技巧也是一个潜在的研究热点。

9.5 不同的峰值之间的混合挑战

使用马尔可夫链蒙特卡罗方法的主要难点在于他们经常混合的很慢。理想情况下，从马尔可夫链中采出的连续的样本之间是完全独立的，而且在 $\boldsymbol{x}$ 空间中，马尔可夫链以正比于不同区域对应的概率的概率访问不同区域。然而，在高维的情况下，马尔可夫链蒙特卡罗方法采出的样本可能会具有很强的相关性。我们把这种现象称为慢混合甚至混合失败。<bad> 具有缓慢混合的马尔可夫链蒙特卡罗方法可以被视为对能量函数无意地执行类似于带噪声的梯度下降的事物，或者相对于链的状态（随机变量被采样）对概率进行等效的噪声爬坡。从 $\boldsymbol{x}^{(t-1)}$ 到 $\boldsymbol{x}^{(t)}$ 该链倾向于选取了很小的步长（在马尔可夫链的状态空间中），其中能量 $E(\boldsymbol{x}^{(t)})$ 通常低于或者近似等于能量 $E(\boldsymbol{x}^{(t-1)})$，倾向于产生较低能量的移动。当从可能性较小的状态（比来自 $p(\boldsymbol{x})$ 的典型样本拥有更高的能量）开始时，链趋向于逐渐减少状态的能量，并且仅仅偶尔移动到另一个峰值。一旦该链已经找到低能量的区域（例如，如果变量是图像中的像素，则低能量的区域可以是同一对象的图像的相连的流形），我们称之为峰值，链将倾向于围绕着这个峰值游走（以某一种形式的随机游走）。有时候，它会远离该峰值，通常返回该峰值或者（如果找到一条离开的路线）移向另一个峰值。问题是对于很多有趣的分布来说成功的离开路线很少，所以马尔可夫链将在一个峰值附近抽取远超过需求的样本。

当考虑到吉布斯采样算法（见9.4节）时，这种现象格外明显。在这种情况下，我们考虑在一定步数内从一个峰值走向移动到一个临近的峰值的概率。决定这个概率的是两个峰值之间的“能量障碍”的形状。隔着一个巨大的“能量障碍”（低概率的区域）的两个峰值之间的转移的概率是（随着能量障碍的高度）指数下降的，如在图9.1中展示的一样。当目标分布有很多峰值并且以很高的概率被低概率区域所分割，尤其当吉布斯采样的每一步都只是更新变量的一小部分而这一小部分又严重依赖其它的变量时，这会导致严重的问题。

举一个简单的例子，考虑两个变量 a, b 的基于能量的模型，这两个变量都是二元的，取值 $+1$ 或者 -1。如果对某个较大的正数 w, $E(a,b) = -wab$，那么这个模型传达了一个强烈的信息，a 和 b 有相同的符号。当 $a = 1$ 时用吉布斯采样更新 b。给定 b 时的条件分布满足 $p(b = 1|a = 1) = \sigma(w)$。如果 w 的值很大，sigmoid 趋近于饱和，那么 b 取到 1 的概率趋近于 1。相同的道理，如果 $a = -1$，那么 b 取到 -1 的概率也趋于 1。根据模型 $p_{\text{model}}(a,b)$，两个变量取一样的符号的概率几乎相等。根据 $p_{\text{model}}(a|b)$，两个变量应该有相同的符号。这也意味着吉

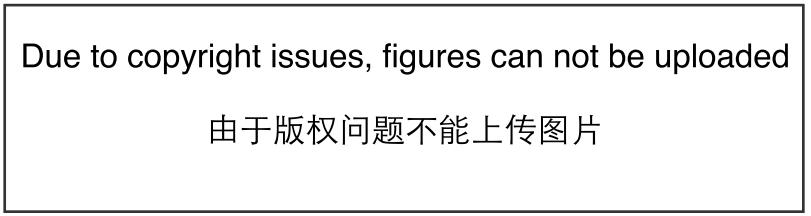


Figure 9.1: Tmp

布斯采样很难会改变这些变量的符号。

在实际问题中，这种挑战更加的艰巨因为在实际问题中我们不能仅仅关注在两个峰值之间的转移而是在多个峰值之间。如果几个这样的转移是很艰难的，那么很难得到一些覆盖大部分峰值的样本集合，同时马尔可夫链收敛到它的静态分布的过程也会非常缓慢。

通过寻找一些相互独立的组以及分块同时更新各组的变量，这个问题有时候可以被解决的。然而不幸的是，当依赖关系很复杂的时候，从这些组中采样的过程从计算角度上说过于复杂。归根结底，马尔可夫链最初就是被提出来解决这个问题，即从大组的变量中采样的问题。

在含有隐含变量的模型中，其中定义来一个联合分布 $p_{\text{model}}(\boldsymbol{x}, \boldsymbol{h})$，我们经常通过交替的从 $p_{\text{model}}(\boldsymbol{x}|\boldsymbol{h})$ 和 $p_{\text{model}}(\boldsymbol{h}|\boldsymbol{x})$ 中采样来达到抽 $\boldsymbol{x}$ 的目的。从快速混合的角度上说，我们更希望 $p_{\text{model}}(\boldsymbol{h}|\boldsymbol{x})$ 有很大的熵。然而，从学习一个 $\boldsymbol{h}$ 的有用表示的角度上考虑，我们还是希望 $\boldsymbol{h}$ 能够包含 $\boldsymbol{x}$ 的足够的信息从而能够较完整的重构它，这意味 $\boldsymbol{h}$ 和 $\boldsymbol{x}$ 有着非常高的互信息。这两个目标是相互矛盾的。我们经常学习到一些有很强编码能力的生成模型，但是无法很好的混合。这种情况在玻耳兹曼机中经常出现，玻耳兹曼机学到到分布越尖锐，该分布的马尔可夫链的采样越难混合的好。这个问题在图9.2中有所描述。

当感兴趣的分布具有对于每个类具有单独的流行结构时，所有这些问题可以使 MCMC 方法不那么有用：分布集中在许多峰值周围，并且这些模式由大量高能量区域分割。我们在许多分类问题中遇到的是这种类型的分布，它将使得 MCMC 方法非常缓慢地收敛，因为峰值之间的混合缓慢。

9.5.1 不同峰值之间通过回火来混合

当一个分布有一些陡峭的峰值并且它们的周围伴随着概率较低的区域时，很难在不同的峰值之间混合。一些加速混合的方法是基于构造一个不同的概率分布，这个概率分布的峰值没有那么高，峰值周围的低谷也没有那么低。基于能量的模型为这个想法提供一种简单的做法。截止目前，我们已经描述了一个基于能量的

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 9.2: Tmp

模型的概率分布的定义：

$$p(\boldsymbol{x}) \propto \exp(-E(\boldsymbol{x})). \tag{9.25}$$

基于能量的模型可以通过添加一个额外的控制峰值尖锐程度的参数 β 来加强：

$$p_\beta(\boldsymbol{x}) \propto \exp(-\beta E(\boldsymbol{x})). \tag{9.26}$$

β 参数可以被理解为**温度**(temperature) 的倒数，在统计物理中反映了基于能量的模型的本质。当温度趋近于 0 时，β 趋近于无穷大，此时的基于能量的模型是确定性的。当温度趋近于无穷大时，β 趋近于零，基于能量的模型 (对离散的 $\boldsymbol{x}$) 成了均匀分布。

通常情况下，在 $\beta = 1$ 的情况下训练一个模型。然而，我们利用了其它温度，尤其是 $\beta < 1$ 的情况。**回火**(tempering) 是一种通用的策略，通过从 $\beta < 1$ 模型中采样来在 p_1 的不同峰值之间快速混合。

基于回火转移的马尔可夫链(**?**) 初始从高温度的分布中采样使得在不同的峰值之间混合，然后从温度为 1 的分布中采样。这些技巧被应用在受限玻耳兹曼机中(**?**)。另一种方法是利用并行回火(**?**)。其中马尔可夫链并行的模拟许多不同温度的不同的状态。最高温度的状态混合较慢，相比之下最低温度的状态，即温度为 1 时，采出了精确的样本。转移算子包括了两个温度之间随机的跳转，所以一个高温度状态分布的样本有足够大的概率跳转到低温度的分布中。这个方法也被应用到了受限玻耳兹曼机中 (**??**)。尽管回火这种方法前景可期，现今它仍然无法让我们在采样复杂的基于能量的模型中更进一步。一个可能的原因是在**临界温度**(critical temperatures) 时温度转移算子必须设置的非常小 (因为温度需要逐渐的下降) 来确保回火的有效性。

9.5.2 深度也许会有助于混合

当我们从隐变量模型 $p(\boldsymbol{h}, \boldsymbol{x})$ 中采样的时候，我们可以发现如果 $p(\boldsymbol{h}|\boldsymbol{x})$ 将 $\boldsymbol{x}$ 编码的非常好，那么从 $p(\boldsymbol{x}|\boldsymbol{h})$ 中采样的时候，并不会太大的改变 $\boldsymbol{x}$，那么混合结果会很糟糕。解决这个问题的一种方法是使得 $\boldsymbol{h}$ 成为一种将 $\boldsymbol{x}$ 编码为 $\boldsymbol{h}$ 的深度表达，从而使得马尔可夫链在 $\boldsymbol{h}$ 空间中更快的混合。在许多表达学习的算法诸如自编码器（autoencoder）和受限玻耳兹曼机中，$\boldsymbol{h}$ 的边缘分布并不像 $\boldsymbol{x}$ 的原始数据分布，反而通常表现为均匀的单一峰值的分布。<bad> 值得指出的是，这些方法往往利用所有可用的表达空间并尽量减小重构误差，因为当训练集上的不同样本能够被非常容易的在 $\boldsymbol{h}$ 空间区分的时候，我们也会很容易的最小化重构误差。<bad> (**?**) 观察到这样的现象，带有正则项的堆叠越深的自编码器或者受限玻耳兹曼机，顶端 $\boldsymbol{h}$ 空间的边缘分布越趋向于均匀分布，而且不同峰值（比如说实验中的类别）所对应的区域之间的间距也会越模糊。在高层次的空间训练受限玻耳兹曼机使得吉布斯采样混合的更快。如何利用这种观察到的现象来辅助训练深度生成模型或者从中采样仍然有待探索。

尽管存在混合的难点，蒙特卡罗技巧仍然是一个有用的也是可用的最好的工具。事实上，在遇到难以处理的无向模型的分割函数的时候，蒙特卡罗方法仍然是最基础的工具，这将在下一章详细阐述。

Chapter 10

近似推断

许多概率模型是很难训练的，其根本原因是很难进行推断。在深度学习中，我们通常有一系列的可见变量 $\boldsymbol{v}$ 和一系列的隐含变量 $\boldsymbol{h}$。推断的挑战往往在于计算 $P(\boldsymbol{h}|\boldsymbol{v})$ 或者计算在 $P(\boldsymbol{h}|\boldsymbol{v})$ 下的期望的困难性。这样的操作在一些任务比如最大似然估计中往往又是必须的。

许多诸如受限玻耳兹曼机和概率 PCA这样的仅仅含有一层隐层的简单的图模型的定义，往往使得推断操作如计算 $P(\boldsymbol{h}|\boldsymbol{v})$ 或者计算 $P(\boldsymbol{h}|\boldsymbol{v})$ 下的期望是非常容易的。不幸的是，大多数的具有多层隐藏变量的图模型的后验分布都很难处理。精确的推断算法需要一个指数量级的运行时间。即使一些只有单层的模型，如稀疏编码，也存在着这样的问题。

在本章中，我们介绍了几个基本的技巧，用来解决难以处理的推断问题。稍后，在第**??**章中，我们还将描述如何将这些技巧应用到训练其他方法难以奏效的概率模型中，如深度信念网络，深度玻耳兹曼机。

在深度学习中难以处理的推断问题通常源于结构化概率模型中的隐含变量之间的相互作用。详见图10.1的几个例子。这些相互作用可能是无向模型的直接作用，也可能是有向模型中的一个可见变量的共同祖先之间的explaining away作用。

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 10.1: TODO

10.1 推断是一个优化问题

许多难以利用观察值进行精确推断的问题往往可以描述为是一个优化问题。通过近似这样一个潜在的优化问题，我们往往可以推导出近似推断算法。

为了构造这样一个优化问题，假设我们有一个包含可见变量 $\boldsymbol{v}$ 和隐含变量 $\boldsymbol{h}$ 的概率模型。我们希望计算观察数据的概率的对数 $\log p(\boldsymbol{v};\boldsymbol{\theta})$。有时候如果消去 $\boldsymbol{h}$ 来计算 $\boldsymbol{x}$ 的边缘分布很费时的话，我们通常很难计算 $\log p(\boldsymbol{v};\boldsymbol{\theta})$。作为替代，我们可以计算一个 $\log p(\boldsymbol{v};\boldsymbol{\theta})$ 的下界。这个下界叫做**证据下界** (evidence lower bound, ELBO)。这个下界的另一个常用的名字是负的**变分自由能**(variational free energy)。这个ELBO是这样定义的：

$$\mathcal{L}(\boldsymbol{v},\boldsymbol{\theta},q) = \log p(\boldsymbol{v};\boldsymbol{\theta}) - D_{\mathrm{KL}}(q(\boldsymbol{h}|\boldsymbol{v})\|p(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta})) \tag{10.1}$$

在这里 q 是关于 $\boldsymbol{h}$ 的一个任意的概率分布。

因为 $\log p(\boldsymbol{v})$ 和 $\mathcal{L}(\boldsymbol{v},\boldsymbol{\theta},q)$ 之间的距离是由KL 散度来衡量的。因为KL 散度总是非负的，我们可以发现 $\mathcal{L}$ 小于等于所求的概率的对数。当且仅当 q 完全相等于 $p(\boldsymbol{h}|\boldsymbol{v})$ 时取到等号。

令人吃惊的是，对某些分布 q，$\mathcal{L}$ 可以被化的更简单。通过简单的代数运算我们可以把 $\mathcal{L}$ 写成更加简单的形式：

$$\begin{aligned}
\mathcal{L}(\boldsymbol{v},\boldsymbol{\theta},q) &= \log p(\boldsymbol{v};\boldsymbol{\theta}) - D_{\mathrm{KL}}(q(\boldsymbol{h}|\boldsymbol{v})\|p(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta})) && (10.2)\\
&= \log p(\boldsymbol{v};\boldsymbol{\theta}) - \mathbb{E}_{\boldsymbol{h}\sim q}\log\frac{q(\boldsymbol{h}|\boldsymbol{v})}{p(\boldsymbol{h}|\boldsymbol{v})} && (10.3)\\
&= \log p(\boldsymbol{v};\boldsymbol{\theta}) - \mathbb{E}_{\boldsymbol{h}\sim q}\log\frac{q(\boldsymbol{h}|\boldsymbol{v})}{\frac{p(\boldsymbol{h},\boldsymbol{v};\boldsymbol{\theta})}{p(\boldsymbol{v};\boldsymbol{\theta})}} && (10.4)\\
&= \log p(\boldsymbol{v};\boldsymbol{\theta}) - \mathbb{E}_{\boldsymbol{h}\sim q}[\log q(\boldsymbol{h}|\boldsymbol{v}) - \log p(\boldsymbol{h},\boldsymbol{v};\boldsymbol{\theta}) + \log p(\boldsymbol{v};\boldsymbol{\theta})] && (10.5)\\
&= -\mathbb{E}_{\boldsymbol{h}\sim q}[\log q(\boldsymbol{h}|\boldsymbol{v}) - \log p(\boldsymbol{h},\boldsymbol{v};\boldsymbol{\theta})] && (10.6)
\end{aligned}$$

这也给出了ELBO的标准的定义：

$$\mathcal{L}(\boldsymbol{v},\boldsymbol{\theta},q) = \mathbb{E}_{\boldsymbol{h}\sim q}[\log q(\boldsymbol{h},\boldsymbol{v})] + H(q) \tag{10.7}$$

对于一个较好的选择 q 来说，$\mathcal{L}$ 是可以计算的。对任意选择 q 来说，$\mathcal{L}$ 提供了一个似然函数的下界。越好的近似 q 的分布 $q(\boldsymbol{h}|\boldsymbol{v})$ 得到的下界就越紧，即与 $\log p(\boldsymbol{v})$ 更加接近。当 $q(\boldsymbol{h}|\boldsymbol{v}) = p(\boldsymbol{h}|\boldsymbol{v})$ 时，这个近似是完美的，也意味着 $\mathcal{L}(\boldsymbol{v},\boldsymbol{\theta},q) = \log p(\boldsymbol{v};\boldsymbol{\theta})$。

我们可以将推断问题看做是找一个分布 q 使得 $\mathcal{L}$ 最大的过程。精确的推断能够通过找一组包含分布 $p(\boldsymbol{h}|\boldsymbol{v})$ 的函数，完美的最大化 $\mathcal{L}$。在本章中，我们将会讲到如何通过近似优化来找 q 的方法来推导出不同形式的的近似推断。我们可以通过限定分布 q 的形式或者使用并不彻底的优化方法来使得优化的过程更加高效，但是优化的结果是不完美的，因为只能显著的提升 $\mathcal{L}$ 而无法彻底的最大化 $\mathcal{L}$。

无论什么样的 q 的选择，$\mathcal{L}$ 是一个下界。我们可以通过选择一个更简单抑或更复杂的计算过程来得到对应的更加松的或者更紧的下界。通过一个不彻底的优化过程或者将 q 分布做很强的限定（使用一个彻底的优化过程）我们可以获得一个很差的 q，尽管计算开销是极小的。

10.2 期望最大化

我们介绍的第一个最大化下界 $\mathcal{L}$ 的算法是**期望最大化** (expectation maximization, EM)。在隐变量模型中，这是一个非常热门的训练算法。在这里我们描述**?** 所提出的EM算法。不像大多数我们在本章中介绍的其他算法一样，EM并不是一个近似推断算法，但是是一种能够学到近似后验的算法。

EM算法包含了交替的两步运算直到收敛的过程：

- **E 步**(expectation step)：令 $\boldsymbol{\theta}^{(0)}$ 表示在这一步开始时参数的初始值。令 $q(\boldsymbol{h}^{(i)}|\boldsymbol{v}) = p(\boldsymbol{h}^{(i)}|\boldsymbol{v}^{(i)};\boldsymbol{\theta}^{(0)})$ 对任何我们想要训练的 (对所有的或者小批数据均成立) 索引为 i 的训练样本 $\boldsymbol{v}^{(i)}$。通过这个定义，我们认为 q 在当前的参数 $\boldsymbol{\theta}^{(0)}$ 下定义。如果我们改变 $\boldsymbol{\theta}$，那么 $p(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta})$ 将会相应的变化，但是 $q(\boldsymbol{h}|\boldsymbol{v})$ 还是不变并且等于 $p(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta}^{(0)})$。
- **M 步**(maximization step)：使用选择的优化算法完全地或者部分的最大化关于 $\boldsymbol{\theta}$ 的

$$\sum_i \mathcal{L}(\boldsymbol{v}^{(i)}, \boldsymbol{\theta}, q) \tag{10.8}$$

这可以被看做是一个坐标上升算法来最大化 $\mathcal{L}$。在第一步中，我们更新 q 来最大化 $\mathcal{L}$，而另一步中，我们更新 $\boldsymbol{\theta}$ 来最大化 $\mathcal{L}$。

基于隐变量模型的随机梯度上升可以被看做是一个EM算法的特例，其中M步包括了单次梯度的操作。EM算法的其他变种可以实现多次梯度操作。对一些模型，M 步可以通过推出理论解直接完成，不同于其他方法，在给定当前 q 的情况下直接求出最优解。

即使E 步采用的是精确推断，我们仍然可以将EM算法视作是某种程度上的近似推断。具体地说，M 步假设了一个 q 分布可以被所有的 $\boldsymbol{\theta}$ 分享。当M 步越来

越远离E 步中的 $\boldsymbol{\theta}^{(0)}$ 的时候，这将会导致 $\mathcal{L}$ 和真实的 $\log p(\boldsymbol{v})$ 的差距。此外，当下一个循环的时候，E 步把这种差距又降到了 0。

EM算法包含了一些不同的解释。首先，学习过程中的基本结构中，我们通过更新模型参数来提高整个数据集的似然，其中缺失的变量的值是通过后验分布来估计的。这种解释并不仅仅适用于EM算法。比如说，使用梯度下降来最大化似然函数的对数这种方法也利用了相同的性质。计算对数似然函数的梯度需要对隐含节点的后验分布来求期望。EM算法的另一个关键的性质是当我们移动到另一个 $\boldsymbol{\theta}$ 时候，我们仍然可以使用旧的 q。在传统机器学习中，这种特有的性质在推导 M 步更新时候被广泛的应用。在深度学习中，大多数模型太过于复杂以致于在 M 步中很难得到一个最优解。所以EM算法的第二个特质较少的被使用。

10.3 最大后验推断和稀疏编码

我们通常使用**推断**(inference) 这个术语来指代给定一定条件下计算一系列变量的概率分布的过程。当训练带有隐含变量的概率模型的时候，我们通常关注于计算 $p(\boldsymbol{h}|\boldsymbol{v})$。在推断中另一个选择是计算一个最有可能的隐含变量的值来代替在其完整分布上的抽样。在隐含变量模型中，这意味着计算

$$\boldsymbol{h}^* = \arg\max_{\boldsymbol{h}} \; p(\boldsymbol{h}|\boldsymbol{v}) \tag{10.9}$$

这被称作是**最大后验** (Maximum A Posteriori, MAP) 推断。

最大后验推断并不是一种近似推断，它计算了最有可能的一个 $\boldsymbol{h}^*$。然而，如果我们希望能够最大化 $\mathcal{L}(\boldsymbol{v}, \boldsymbol{h}, q)$，那么我们可以把最大后验推断看成是输出一个 q 的学习过程。在这种情况下，我们可以将最大后验推断看成是近似推断，因为它并不能提供一个最优的 q。

我们回过头来看看10.1节中所描述的精确推断，它指的是关于一个不受限的 q 分布使用精确的优化算法来最大化

$$\mathcal{L}(\boldsymbol{v}, \boldsymbol{\theta}, q) = \mathbb{E}_{\boldsymbol{h}\sim q}[\log q(\boldsymbol{h}, \boldsymbol{v})] + H(q). \tag{10.10}$$

我们通过限定 q 分布属于某个分布族，能够使得最大后验推断成为一种形式的近似推断。具体的说，我们令 q 分布满足一个Dirac 分布：

$$q(\boldsymbol{h}|\boldsymbol{v}) = \delta(\boldsymbol{h} - \boldsymbol{\mu}) \tag{10.11}$$

这也意味着我们可以通过 $\boldsymbol{\mu}$ 来完全控制 q。通过将 $\mathcal{L}$ 中不随 $\boldsymbol{\mu}$ 变化的项丢弃，剩下的我们遇到的是一个优化问题：

$$\boldsymbol{\mu}^* = \arg\max_{\boldsymbol{\mu}} \; \log p(\boldsymbol{h} = \boldsymbol{\mu}, \boldsymbol{v}) \tag{10.12}$$

这等价于最大后验推断问题

$$\boldsymbol{h}^* = \arg\max_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) \tag{10.13}$$

因此我们能够解释一种类似于EM算法的学习算法，其中我们轮流迭代两步，一步是用最大后验推断估计出 $\boldsymbol{h}^*$，另一步是更新 $\boldsymbol{\theta}$ 来增大 $\log p(\boldsymbol{h}^*, \boldsymbol{v})$。从EM算法角度看，这也是一种形式的对 $\mathcal{L}$ 的坐标上升，EM算法的坐标上升中，交替迭代的时候通过推断来优化 $\mathcal{L}$ 关于 q 以及通过参数更新来优化 $\mathcal{L}$ 关于 $\boldsymbol{\theta}$。整体上说，这个算法的正确性可以被保证因为 $\mathcal{L}$ 是 $\log p(\boldsymbol{v})$ 的下界。在最大后验推断中，这个保证是无效的，因为这个界会无限的松，由于Dirac 分布的熵的微分趋近于负无穷。然而，人为加入一些 $\boldsymbol{\mu}$ 的噪声会使得这个界又有了意义。

最大后验推断作为特征提取器以及一种学习机制被广泛的应用在了深度学习中。在稀疏编码模型中，它起到了关键作用。

我们回过头来看6.4节中的稀疏编码，稀疏编码是一种在隐含节点上加上了鼓励稀疏的先验知识的线性因子模型。一个常用的选择是可分解的拉普拉斯先验，表示为

$$p(h_i) = \frac{\lambda}{2}\exp(-\lambda|h_i|) \tag{10.14}$$

可见的节点是由一个线性变化加上噪音生成的[1]:

$$p(\boldsymbol{x}|\boldsymbol{h}) = \mathcal{N}(\boldsymbol{v}; \boldsymbol{W}\boldsymbol{h} + \boldsymbol{b}, \beta^{-1}\boldsymbol{I}) \tag{10.15}$$

计算或者表达 $p(\boldsymbol{h}|\boldsymbol{v})$ 太过困难。每一对 h_i，h_j 变量都是 $\boldsymbol{v}$ 的母节点。这也意味着当 $\boldsymbol{v}$ 是可观察时，图模型包含了连接 h_i 和 h_j 的路径。因此在 $p(\boldsymbol{h}|\boldsymbol{v})$ 中所有的隐含节点都包含在了一个巨大的团中。如果模型是高斯的，那么这些关系可以通过协方差矩阵来高效的建模。然而稀疏型先验使得模型并不是高斯。

$p(\boldsymbol{x}|\boldsymbol{h})$ 的复杂性导致了似然函数的对数及其梯度也很难得到。因此我们不能使用精确的最大似然估计来进行学习。取而代之的是，我们通过MAP推断以及最大化由以 $\boldsymbol{h}$ 为中心的Dirac 分布所定义而成的ELBO来学习模型参数。

如果我们将训练集中的所有的 $\boldsymbol{h}$ 向量拼在一起并且记为 $\boldsymbol{H}$，并将所有的 $\boldsymbol{v}$ 向量拼起来组成矩阵 $\boldsymbol{V}$，那么稀疏编码问题意味着最小化

$$J(\boldsymbol{H}, \boldsymbol{W}) = \sum_{i,j}|H_{i,j}| + \sum_{i,j}\left(\boldsymbol{V} - \boldsymbol{H}\boldsymbol{W}^\top\right)^2_{i,j} \tag{10.16}$$

为了避免如极端小的 $\boldsymbol{H}$ 和极端大的 $\boldsymbol{W}$ 这样的病态的解，许多稀疏编码的应用包含了权值衰减或者 $\boldsymbol{H}$ 的列的范数的限制。

[1]此处似乎有笔误，$\boldsymbol{x}$ 应为 $\boldsymbol{v}$

我们可以通过交替迭代最小化 J 分别关于 $\boldsymbol{H}$ 和 $\boldsymbol{W}$ 的方式来最小化 J。两个子问题都是凸的。事实上，关于 $\boldsymbol{W}$ 的最小化问题就是一个线性回归问题。然而关于这两个变量同时最小化 J 的问题并不是凸的。

关于 $\boldsymbol{H}$ 的最小化问题需要某些特别设计的算法诸如特征符号搜索方法 (**?**)。

10.4 变分推断和学习

我们已经说明过了为什么ELBO是 $\log p(\boldsymbol{v};\boldsymbol{\theta})$ 的一个下界，如何将推断看做是关于 q 分布最大化 $\mathcal{L}$ 的过程以及如何将学习看做是关于参数 $\boldsymbol{\theta}$ 最大化 $\mathcal{L}$ 的过程。我们也讲到了EM算法在给定了 q 分布的条件下进行学习，而MAP推断的算法则是学习一个 $p(\boldsymbol{h}|\boldsymbol{v})$ 的点估计而非推断整个完整的分布。在这里我们介绍一些变分学习中更加通用的算法。

变分学习的核心思想就是我们通过选择给定的分布族中的一个 q 分布来最大化 $\mathcal{L}$。选择这个分布族的时候应该考虑到计算 $\mathbb{E}_q \log p(\boldsymbol{h}, \boldsymbol{v})$ 的简单性。一个典型的方法就是添加一些假设诸如 q 可以分解。

一种常用的变分学习的方法是加入一些限制使得 q 是一个可以分解的分布：

$$q(\boldsymbol{h}|\boldsymbol{v}) = \prod_i q(h_i|\boldsymbol{v}) \tag{10.17}$$

这被叫做是**均匀场**(mean-field) 方法。一般来说，我们可以通过选择 q 分布的形式来选择任何图模型的结构，通过选择变量之间的相互作用来决定近似程度的大小。这种完全通用的图模型方法叫做**结构化变分推断**(structured variational inference) (**?**)。

变分方法的优点是我们不需要为分布 q 设定一个特定的参数化的形式。我们设定它如何分解，而优化问题中决定了在这些分解限制下的最优的概率分布。对离散型的隐含变量来说，这意味着我们使用了传统的优化技巧来优化描述 q 分布的有限个数的变量。对连续性的变量来说，这意味着我们使用了一个叫做**变分法**(calculus of variations) 的数学分支来解决对一个空间的函数的优化问题。然后决定哪一个函数来表示 q。变分法是“变分学习”或者“变分推断”这些名字的来历，尽管当隐含变量是离散的时候变分法并没有用武之地。当遇到连续的隐含变量的时候，变分法是一种很有用的工具，只需要设定分布 q 如何分解，而不需要过多的人工选择模型，比如尝试着设计一个特定的能够精确的近似原后验分布的 q。

因为 $\mathcal{L}(\boldsymbol{v}, \boldsymbol{\theta}, q)$ 定义成 $\log p(\boldsymbol{v};\boldsymbol{\theta}) - D_{\mathrm{KL}}(q(\boldsymbol{h}|\boldsymbol{v})\|p(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta}))$，我们可以认为关于 q 最大化 $\mathcal{L}$ 的问题等价于最小化 $D_{\mathrm{KL}}(q(\boldsymbol{h}|\boldsymbol{v})\|p(\boldsymbol{h}|\boldsymbol{v}))$。在这种情况下，我们要用 q 来拟合 p。然而，我们并不是直接拟合一个近似，而是处理一个KL 散度的问题。当我们使用最大似然估计来将数据拟合到模型的时候，我们最小化

$D_{\mathrm{KL}}(p_{\mathrm{data}}\|p_{\mathrm{model}})$。如同图**??**中所示，这意味着最大似然估计促进模型在每一个数据达到更高概率的地方达到更高的概率，而基于优化的推断则促进了 q 在每一个真实后验分布概率较低的地方概率较小。这两种方法都有各自的优点与缺点。选择哪一种方法取决于在具体应用中哪一种性质更受偏好。在基于优化的推断问题中，从计算角度考虑，我们选择使用 $D_{\mathrm{KL}}(q(\boldsymbol{h}|\boldsymbol{v})\|p(\boldsymbol{h}|\boldsymbol{v}))$。具体的说，计算 $D_{\mathrm{KL}}(q(\boldsymbol{h}|\boldsymbol{v})\|p(\boldsymbol{h}|\boldsymbol{v}))$ 涉及到了计算 q 分布下的期望，所以通过将分布 q 设计的较为简单，我们可以简化求所需要的期望的计算过程。另一个KL 散度的方向需要计算真实后验分布下的期望。因为真实后验分布的形式是由模型的选择决定的，我们不能设计出一种能够精确计算 $D_{\mathrm{KL}}(p(\boldsymbol{h}|\boldsymbol{v})\|q(\boldsymbol{h}|\boldsymbol{v}))$ 的开销较小的方法。

10.4.1 离散隐含变量

关于离散型的隐含变量的变分推断相对来说更为直接。我们定义一个分布 q，通常 q 的每个因子都由一些离散状态的表格定义。在最简单的情况中，$\boldsymbol{h}$ 是二元的并且我们做了均匀场的假定，q 可以根据每一个 h_i 分解。在这种情况下，我们可以用一个向量 $\hat{\boldsymbol{h}}$ 来参数化 q 分布，$\hat{\boldsymbol{h}}$ 的每一个元素都代表一个概率，即 $q(h_i = 1|\boldsymbol{v}) = \hat{h}_i$。

在确定了如何表示 q 以后，我们简单的优化了它的参数。在离散型的隐含变量模型中，这是一个标准的优化问题。基本上 q 的选择可以通过任何的优化算法解决，比如说梯度下降。

这个优化问题是很高效的因为它在许多学习算法的内循环中出现。为了追求速度，我们通常使用特殊设计的优化算法。这些算法通常能够在极少的循环内解决一些小而简单的问题。一个常见的选择是使用固定点方程，换句话说，就是对 $\hat{h}_i$ 解决

$$\frac{\partial}{\partial \hat{h}_i}\mathcal{L} = 0 \tag{10.18}$$

我们反复的更新 $\hat{\boldsymbol{h}}$ 不同的元素直到收敛条件满足。

为了具体化这些描述，我们接下来会讲如何将变分推断应用到**二值稀疏编码**(binary sparse coding) 模型（这里我们所描述的模型是**?** 提出的，但是我们采用了传统的通用的均匀场方法，而原文作者采用了一种特殊设计的算法）中。推导过程在数学上非常详细，为希望完全了解变分推断细节的读者所准备。而对于并不计划推导或者实现变分学习算法的读者来说，可以完全跳过，直接阅读下一节，这并不会导致重要概念的遗漏。那些从事过二值稀疏编码研究的读者可以重新看一下**??**节中的一些经常在概率模型中出现的有用的函数性质。我们在推导过程中随意的使用了这些性质，并没有特别强调它们。

在二值稀疏编码模型中，输入 $\boldsymbol{v} \in \mathbb{R}^n$，是由模型通过添加高斯噪音到 m 个或

有或无的成分。每一个成分可以是开或者关的通过对应的隐藏层节点 $\boldsymbol{h} \in \{0,1\}^m$:

$$p(h_i = 1) = \sigma(b_i) \tag{10.19}$$

$$p(\boldsymbol{v}|\boldsymbol{h}) = \mathcal{N}(\boldsymbol{v}; \boldsymbol{W}\boldsymbol{h}, \boldsymbol{\beta}^{-1}) \tag{10.20}$$

其中 $\boldsymbol{b}$ 是一个可以学习的偏置向量[2]，$\boldsymbol{W}$ 是一个可以学习的权值矩阵，$\boldsymbol{\beta}$ 是一个可以学习的对角的精度矩阵。

使用最大似然估计来训练这样一个模型需要对参数进行求导。我们考虑对其中一个偏置进行求导的过程:

$$\frac{\partial}{\partial b_i} \log p(\boldsymbol{v}) \tag{10.21}$$

$$= \frac{\frac{\partial}{\partial b_i} p(\boldsymbol{v})}{p(\boldsymbol{v})} \tag{10.22}$$

$$= \frac{\frac{\partial}{\partial b_i} \sum_{\boldsymbol{h}} p(\boldsymbol{h}, \boldsymbol{v})}{p(\boldsymbol{v})} \tag{10.23}$$

$$= \frac{\frac{\partial}{\partial b_i} \sum_{\boldsymbol{h}} p(\boldsymbol{h}) p(\boldsymbol{v}|\boldsymbol{h})}{p(\boldsymbol{v})} \tag{10.24}$$

$$= \frac{\sum_{\boldsymbol{h}} p(\boldsymbol{v}|\boldsymbol{h}) \frac{\partial}{\partial b_i} p(\boldsymbol{h})}{p(\boldsymbol{v})} \tag{10.25}$$

$$= \sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) \frac{\frac{\partial}{\partial b_i} p(\boldsymbol{h})}{p(\boldsymbol{h})} \tag{10.26}$$

$$= \mathbb{E}_{\boldsymbol{h} \sim p(\boldsymbol{h}|\boldsymbol{v})} \frac{\partial}{\partial b_i} \log p(\boldsymbol{h}) \tag{10.27}$$

这需要计算 $p(\boldsymbol{h}|\boldsymbol{v})$ 下的期望。不幸的是，$p(\boldsymbol{h}|\boldsymbol{v})$ 是一个很复杂的分布。$p(\boldsymbol{h}|\boldsymbol{v})$ 和 $p(\boldsymbol{h}, \boldsymbol{v})$ 的图结构参见10.2。隐含节点的后验分布对应的是完全图，所以相对于暴力算法，消元算法并不能有助于提高计算所需要的期望的效率。

取而代之的是，我们可以应用变分推断和变分学习来解决这个难点。

我们可以做一个均匀场的假设:

$$q(\boldsymbol{h}|\boldsymbol{v}) = \prod_i q(h_i|\boldsymbol{v}) \tag{10.28}$$

[2]此处似乎有笔误，公式里面漏了 $\boldsymbol{b}$?

Due to copyright issues, figures can not be uploaded

由于版权问题不能上传图片

Figure 10.2: TODO

二值稀疏编码中的隐含变量是二值的，所以为了表示可分解的 q 我们假设模型为 m 个Bernoulli 分布$q(h_i|\boldsymbol{v})$。表示Bernoulli 分布的一种常见的方法是一个概率向量 $\hat{\boldsymbol{h}}$，满足 $q(h_i|\boldsymbol{v}) = \hat{h}_i$。为了避免计算中的误差，比如说计算 $\log \hat{h}_i$ 的时候，我们对 $\hat{h}_i$ 添加一个约束，即 $\hat{h}_i$ 不等于 0 或者 1。

我们将会看到变分推断方程理论上永远不会赋予 $\hat{h}_i$ 0 或者 1。然而在软件实现过程中，机器的舍入误差会导致 0 或者 1 的值。在二值稀疏编码的实现中，我们希望使用一个没有限制的变分参数向量 $\boldsymbol{z}$ 以及通过关系 $\hat{\boldsymbol{h}} = \sigma(\boldsymbol{z})$ 来获得 $\boldsymbol{h}$。因此我们可以放心的在计算机上计算 $\log \hat{h}_i$ 通过使用关系式 $\log \sigma(z_i) = -\zeta(-z_i)$ 来建立 sigmoid 和 softplus 的关系。

在开始二值稀疏编码模型的推导时，我们首先说明了均匀场的近似可以使得学习的过程更加简单。

ELBO可以表示为

$$\mathcal{L}(\boldsymbol{v}, \boldsymbol{\theta}, q) \tag{10.29}$$

$$= \mathbb{E}_{\boldsymbol{h}\sim q}[\log p(\boldsymbol{h}, \boldsymbol{v})] + H(q) \tag{10.30}$$

$$= \mathbb{E}_{\boldsymbol{h}\sim q}[\log p(\boldsymbol{h}) + \log p(\boldsymbol{v}|\boldsymbol{h}) - \log q(\boldsymbol{h}|\boldsymbol{v})] \tag{10.31}$$

$$= \mathbb{E}_{\boldsymbol{h}\sim q}\left[\sum_{i=1}^{m} \log p(h_i) + \sum_{i=1}^{n} \log p(v_i|\boldsymbol{h}) - \sum_{i=1}^{m} \log q(h_i|\boldsymbol{v})\right] \tag{10.32}$$

$$= \sum_{i=1}^{m}\left[\hat{h}_i(\log \sigma(b_i) - \log \hat{h}_i) + (1-\hat{h}_i)(\log \sigma(-b_i) - \log(1-\hat{h}_i))\right] \tag{10.33}$$

$$+ \mathbb{E}_{\boldsymbol{h}\sim q}\left[\sum_{i=1}^{n} \log \sqrt{\frac{\beta_i}{2\pi}} \exp(-\frac{\beta_i}{2}(v_i - \boldsymbol{W}_{i,:}\boldsymbol{h})^2)\right] \tag{10.34}$$

$$= \sum_{i=1}^{m}\left[\hat{h}_i(\log \sigma(b_i) - \log \hat{h}_i) + (1-\hat{h}_i)(\log \sigma(-b_i) - \log(1-\hat{h}_i))\right] \tag{10.35}$$

$$+ \frac{1}{2}\sum_{i=1}^{n}\left[\log \frac{\beta_i}{2\pi} - \beta_i\left(v_i^2 - 2v_i \boldsymbol{W}_{i,:}\hat{\boldsymbol{h}} + \sum_j \left[W_{i,j}^2 \hat{h}_j + \sum_{k\neq j} W_{i,j}W_{i,k}\hat{h}_j\hat{h}_k\right]\right)\right] \tag{10.36}$$

尽管从美学观点这些方程有些不尽如人意。他们展示了 $\mathcal{L}$ 可以被表示为少数的简单的代数运算。因此ELBO是易于处理的。我们可以把 $\mathcal{L}$ 看作是难以处理的似然函数的对数。

原则上说，我们可以使用关于 $\boldsymbol{v}$ 和 $\boldsymbol{h}$ 的梯度下降。这会成为一个完美的组合的推断学习算法。但是，由于两个原因，我们往往不这么做。第一点，对每一个 $\boldsymbol{v}$ 我们需要存储 $\hat{\boldsymbol{h}}$。我们通常更加偏向于那些不需要为每一个样本都准备内存的算法。如果我们需要为每一个样本都存储一个动态更新的向量，很难使得算法能够处理好几亿的样本。第二个原因就是为了能够识别 $\boldsymbol{v}$ 的内容，我们希望能够有能力快速提取特征 $\hat{\boldsymbol{h}}$。在实际应用下，我们需要在有限时间内计算出 $\hat{\boldsymbol{h}}$。

由于以上两个原因，我们通常不会采用梯度下降来计算均匀场的参数 $\hat{\boldsymbol{h}}$。取而代之的是，我们使用固定点方程来快速估计他们。

固定点方程的核心思想是我们寻找一个关于 $\boldsymbol{h}$ 的局部最大值，满足 $\nabla_h \mathcal{L}(\boldsymbol{v}, \boldsymbol{\theta}, \hat{\boldsymbol{h}}) = 0$。我们无法同时高效的计算所有的 $\hat{\boldsymbol{h}}$ 的元素。然而，我们可以解决单个变量的问题：

$$\frac{\partial}{\partial \hat{h}_i}\mathcal{L}(\boldsymbol{v}, \boldsymbol{\theta}, \hat{\boldsymbol{h}}) = 0 \tag{10.37}$$

我们可以迭代的将这个解应用到 $i = 1, \ldots, m$，然后重复这个循环直到我们满

足了收敛标准。常见的收敛标准包含了当整个循环所改进的 $\mathcal{L}$ 不超过预设的容忍的量时停止，或者是循环中改变的 $\hat{\boldsymbol{h}}$ 不超过某个值的时候停止。

在很多不同的模型中，迭代的均匀场固定点方程是一种能够提供快速变分推断的通用的算法。为了使他更加具体化，我们详细的讲一下如何推导出二值稀疏编码模型的更新过程。

首先，我们给出了对 $\hat{h}_i$ 的导数的表达式。为了得到这个表达式，我们将方程 (10.29)代入到方程 (10.37)的左边：

$$\frac{\partial}{\partial \hat{h}_i}\mathcal{L}(\boldsymbol{v}, \boldsymbol{\theta}, \hat{\boldsymbol{h}}) \tag{10.38}$$

$$= \frac{\partial}{\partial \hat{h}_i}\left[\sum_{j=1}^{m}\left[\hat{h}_j(\log\sigma(b_j) - \log\hat{h}_j) + (1-\hat{h}_j)(\log\sigma(-b_j) - \log(1-\hat{h}_j))\right]\right. \tag{10.39}$$

$$\left. + \frac{1}{2}\sum_{j=1}^{n}\left[\log\frac{\beta_j}{2\pi} - \beta_j\left(v_j^2 - 2v_j\,\boldsymbol{W}_{j,:}\hat{\boldsymbol{h}} + \sum_k\left[W_{j,k}^2\hat{h}_k + \sum_{l\neq k}W_{j,k}W_{j,l}\hat{h}_k\hat{h}_l\right]\right)\right]\right] \tag{10.40}$$

$$= \log\sigma(b_i) - \log\hat{h}_i - 1 + \log(1-\hat{h}_i) + 1 - \log\sigma(-b_i) \tag{10.41}$$

$$+ \sum_{j=1}^{n}\left[\beta_j\left(v_jW_{j,i} - \frac{1}{2}W_{j,i}^2 - \sum_{k\neq i}\boldsymbol{W}_{j,k}\,\boldsymbol{W}_{j,i}\hat{h}_k\right)\right] \tag{10.42}$$

$$= b_i - \log\hat{h}_i + \log(1-\hat{h}_i) + \boldsymbol{v}^\top\boldsymbol{\beta}\,\boldsymbol{W}_{:,i} - \frac{1}{2}\boldsymbol{W}_{:,i}^\top\boldsymbol{\beta}\,\boldsymbol{W}_{:,i} - \sum_{j\neq i}\boldsymbol{W}_{:,j}^\top\boldsymbol{\beta}\,\boldsymbol{W}_{:,i}\hat{h}_j \tag{10.43}$$

为了应用固定点更新的推理规则，我们通过令方程 (10.38)等于 0 来解 $\hat{h}_i$：

$$\hat{h}_i = \sigma\left(b_i + \boldsymbol{v}^\top\boldsymbol{\beta}\,\boldsymbol{W}_{:,i} - \frac{1}{2}\boldsymbol{W}_{:,i}^\top\boldsymbol{\beta}\,\boldsymbol{W}_{:,i} - \sum_{j\neq i}\boldsymbol{W}_{:,j}^\top\boldsymbol{\beta}\,\boldsymbol{W}_{:,i}\hat{h}_j\right) \tag{10.44}$$

此时，我们可以发现图模型中循环神经网络和推断之间存在着紧密的联系。具体的说，均匀场固定点方程定义了一个循环神经网络。这个神经网络的任务就是完成推断。我们已经从模型描述角度介绍了如何推导这个网络，但是直接训练这个推断网络也是可行的。有关这种思路的一些想法在第**??**章中有所描述。

<bad> 在二值稀疏编码模型中，我们可以发现方程 (10.44)中描述的循环网络包含了根据相邻的变化着的隐藏层结点值来反复更新当前隐藏层结点的操作。输入层通常给隐藏层结点发送一个固定的信息 $\boldsymbol{v}^\top\beta\,\boldsymbol{W}$，然而隐藏层结点不断地更新互相传送的信息。具体的说，当 $\hat{h}_i$ 和 $\hat{h}_j$ 两个单元的权重向量对准时，他

们会产生相互抑制。这也是一种形式的竞争———两个共同理解输入的隐层结点之间，只有一个解释的更好的才能继续保持活跃。在二值稀疏编码的后杨分布中，均匀场近似为了捕获到更多的explaining away作用，产生了这种竞争。事实上，explaining away效应会导致一个多峰值的后验分布，所以我们如果从后验分布中采样，一些样本只有一个结点是活跃的，其它的样本在其它的结点活跃，只有很少的样本能够所有的结点都处于活跃状态。不幸的是，explaining away作用无法通过均匀场中可分解的 q 分布来建模，因此建模时均匀场近似只能选择一个峰值。这个现象的一个例子可以参考图 **??**。

我们将方程 (10.44)重写成等价的形式来揭示一些深层的含义：

$$\hat{h}_i = \sigma\left(b_i + \left(\boldsymbol{v} - \sum_{j \neq i} \boldsymbol{W}_{:,j}\hat{h}_j\right)^\top \boldsymbol{\beta}\boldsymbol{W}_{:,i} - \frac{1}{2}\boldsymbol{W}_{:,i}^\top \boldsymbol{\beta}\boldsymbol{W}_{:,i}\right) \tag{10.45}$$

在这种新的形式中，我们可以将每一步的输入看成 $\boldsymbol{v} - \sum_{j \neq i} \boldsymbol{W}_{:,j}\hat{h}_j$ 而不是 $\boldsymbol{v}$。因此，我们可以把第 i 个单元视作给定其它单元的编码时编码 $\boldsymbol{v}$ 中的剩余误差。由此我们可以将稀疏编码视作是一个迭代的自动编码器，反复的将输入的编码解码，试图逐渐减小重构误差。

在这个例子中，我们已经推导出了每一次更新单个结点的更新规则。接下来我们考虑如何同时更新许多结点。某些图模型，比如DBM，我们可以同时更新 $\hat{\boldsymbol{h}}$ 中的许多元素。不幸的是，二值稀疏编码并不适用这种块更新。取而代之的是，我们使用一种称为是**衰减**(damping) 的启发式的技巧来实现块更新。在衰减方法中，对 $\hat{\boldsymbol{h}}$ 中的每一个元素我们都可以解出最优值，然后对于所有的值都在这个方向上移动一小步。这个方法不能保证每一步都能增加 $\mathcal{L}$，但是对于许多模型来说却很有效。关于在信息传输算法中如何选择同步程度以及使用衰减策略可以参考**?** 。

10.4.2 变分法

在继续描述变分学习之前，我们有必要简单的介绍一种变分学习中的重要的数学工具：**变分法**(calculus of variations)。

许多机器学习的技巧是基于寻找一个输入向量 $\boldsymbol{\theta} \in \mathbb{R}^n$ 来最小化函数 $J(\boldsymbol{\theta})$，使得它取到最小值。这个步骤可以利用多元微积分以及线性代数的知识找到满足 $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = 0$ 的临界点来完成。在某些情况下，我们希望能够找出一个最优的函数 $f(\boldsymbol{x})$，比如当我们希望找到一些随机变量的概率密度函数的时候。变分法能够让我们完成这个目标。

f 函数的函数被称为是**泛函**(functional)$J[f]$。正如我们许多情况下对一个函数求关于以向量为变量的偏导数一样，我们可以使用**泛函导数**(functional derivative)，即对任意一个给定的 $\boldsymbol{x}$，对一个泛函$J[f]$ 求关于函数 $f(\boldsymbol{x})$ 的导数，这也被称为**变分微分**(variational derivative)。泛函J 的关于函数 f 在点 $\boldsymbol{x}$ 处

的泛函导数被记作 $\frac{\delta}{\delta f(x)}J$。

完整正式的泛函导数的推导不在本书的范围之内。为了满足我们的目标，讲述可微分函数 $f(\boldsymbol{x})$ 以及带有连续导数的可微分函数 $g(y, \boldsymbol{x})$ 就足够了：

$$\frac{\delta}{\delta f(\boldsymbol{x})} \int g(f(\boldsymbol{x}), \boldsymbol{x}) d\boldsymbol{x} = \frac{\partial}{\partial y} g(f(\boldsymbol{x}), \boldsymbol{x}) \tag{10.46}$$

为了使上述的关系式更加的形象，我们可以把 $f(\boldsymbol{x})$ 看做是一个有着无穷多元素的向量。在这里（看做是一个不完全的介绍），这种关系式中描述的泛函导数和向量 $\boldsymbol{\theta} \in \mathbb{R}^n$ 的导数相同：

$$\frac{\partial}{\partial \theta_i} \sum_j g(\theta_j, j) = \frac{\partial}{\partial \theta_i} g(\theta_i, i) \tag{10.47}$$

在其他机器学习的文献中的许多结果是利用了更为通用的**欧拉拉格朗日方程**(Euler-Lagrange Equation)，它能够使得 g 不仅依赖于 f 的导数而且也依赖于 f 的值。但是本书中我们不需要把完整的讲述这个结果。

为了优化某个函数关于一个向量，我们求出了这个函数关于这个向量的梯度，然后找一个梯度的每一个元素都为 0 的点。类似的，我们可以通过寻找一个函数使得泛函导数的每个点都等于 0 从而来优化一个泛函。

下面讲一个这个过程是如何工作的，我们考虑寻找一个定义在 $x \in \mathbb{R}$ 的有最大微分熵的概率密度函数。我们回过头来看一下一个概率分布 $p(x)$ 的熵，定义如下：

$$H[p] = -\mathbb{E}_x \log p(x) \tag{10.48}$$

对于连续的值，这个期望可以看成是一个积分：

$$H[p] = -\int p(x) \log p(x) dx \tag{10.49}$$

我们不能简单的关于函数 $p(x)$ 最大化 $H[p]$，因为那样的化结果可能不是一个概率分布。为了解决这个问题，我们需要使用一个拉格朗日乘子来添加一个 $p(x)$ 积分值为 1 的约束。此外，当方差增大的时候，熵也会无限制的增加。因此，寻找哪一个分布有最大熵这个问题是没有意义的。但是，在给定固定的方差 σ^2 的时候，我们可以寻找一个最大熵的分布。最后，这个问题还是无法确定因为在不改变熵的条件下一个分布可以被随意的改变。为了获得一个唯一的解，我们再加

一个约束：分布的均值必须为 μ。那么这个问题的拉格朗日泛函可以被写成

$$\mathcal{L}[p] = \lambda_1\Big(\int p(x)dx - 1\Big) + \lambda_2(\mathbb{E}[x] - \mu) + \lambda_3(\mathbb{E}[(x-\mu)^2] - \sigma^2) + H[p] \tag{10.50}$$

$$= \int \Big(\lambda_1 p(x) + \lambda_2 p(x)x + \lambda_3 p(x)(x-\mu)^2 - p(x)\log p(x)\Big)dx - \lambda_1 - \mu\lambda_2 - \sigma^2\lambda_3 \tag{10.51}$$

为了关于 p 最小化拉格朗日乘子，我们令泛函导数等于 0：

$$\forall x, \quad \frac{\delta}{\delta p(x)}\mathcal{L} = \lambda_1 + \lambda_2 x + \lambda_3(x-\mu)^2 - 1 - \log p(x) = 0 \tag{10.52}$$

这个条件告诉我们 $p(x)$ 的泛函形式。通过代数运算重组上述方程，我们可以得到

$$p(x) = \exp\left(\lambda_1 + \lambda_2 x + \lambda_3(x-\mu)^2 - 1\right) \tag{10.53}$$

我们并没有直接假设 $p(x)$ 取这种形式，而是通过最小化这个泛函从理论上得到了这个 $p(x)$ 的表达式。为了完成这个优化问题，我们需要选择 λ 的值来确保所有的约束都能够满足。我们有很大的选择 λ 的自由。因为只要约束满足，拉格朗日关于 λ 的梯度为 0。为了满足所有的约束，我们可以令 $\lambda_1 = 1 - \log\sigma\sqrt{2\pi}$,$\lambda_2 = 0$, $\lambda_3 = -\frac{1}{2\sigma^2}$，从而取到

$$p(x) = \mathcal{N}(x;\mu,\sigma^2) \tag{10.54}$$

这也是当我们不知道真实的分布的时候总是使用正态分布的原因。因为正态分布拥有最大的熵，<bad> 我们通过这个假定来保证了最小可能量的结构。

当寻找熵的拉格朗日泛函的临界点并且给定一个固定的方差的时候，我们只能找到一个对应最大熵的临界点。那是否存在一个最小化熵的概率分布函数呢？为什么我们无法发现第二个最小值的临界点呢？原因是没有一个特定的函数能够达到最小的熵值。当函数把越多的概率密度加到 $x = \mu + \sigma$ 和 $x = \mu - \sigma$ 两个点上和越少的概率密度到其他点上时，他们的熵值会减少，而方差却不变。然而任何把所有的权重都放在这两点的函数的积分并不为 1。所以不存在一个最小熵的概率分布函数，就像不存在一个最小的正实数一样。然而，我们发现存在一个概率分布的收敛的序列，直到把权重都放在两个点上。这种情况能够退化为混合Dirac 分布。因为Dirac 分布并不是一个单独的概率分布，所以Dirac 分布或者混合Dirac 分布并不能对应函数空间的一个点。所以对我们来说，当寻找一个泛函导数为 0 的函数空间的点的时候，这些分布是不可见的。这就是这种方法的局限之处。像Dirac 分布这样的分布可以通过其他方法被找到，比如可以被猜到，然后证明它是满足条件的。

10.4.3 连续型隐含变量

当我们的图模型包含了连续型隐含变量的时候，我们仍然可以通过最大化 $\mathcal{L}$ 进行变分推断和学习。然而，我们需要使用变分法来实现关于 $q(\boldsymbol{h}|\boldsymbol{v})$ 最大化 $\mathcal{L}$。

在大多数情况下，研究者并不需要解决任何的变分法的问题。取而代之的是，均匀场固定点迭代有一种通用的方程。如果我们做了均匀场的近似：

$$q(\boldsymbol{h}|\boldsymbol{v}) = \prod_i q(h_i|\boldsymbol{v}) \tag{10.55}$$

并且对任何的 $j \neq i$ 固定了 $q(h_j|\boldsymbol{v})$，那么只需要满足 p 中任何联合分布中的变量不为 0, 我们就可以通过归一化下面这个未归一的分布

$$\tilde{q}(h_i|\boldsymbol{v}) = \exp(\mathbb{E}_{h_{-i}\sim q(h_{-i}|\boldsymbol{v})} \log \tilde{p}(\boldsymbol{v}, \boldsymbol{h})) \tag{10.56}$$

来得到最优的 $q(h_i|\boldsymbol{v})$。在这个方程中计算期望就能得到一个 $q(h_i|\boldsymbol{v})$ 的正确表达式。我们只有在希望提出一种新形式的变分学习算法时才需要直接推导 q 的函数形式。方程 (10.56)给出了适用于任何概率模型的均匀场近似。

方程 (10.56)是一个固定点方程，对每一个 i 它都被迭代的反复使用直到收敛。然而，它还包含着更多的信息。我们发现这种泛函定义的问题的最优解是存在的，无论我们是否能够通过固定点方程来解出它。这意味着我们可以把一些值当成参数，然后通过优化算法来解决这个问题。

我们拿一个简单的概率模型作为一个例子，其中隐含变量满足 $\boldsymbol{h} \in \mathbb{R}^2$，可视的变量只有一个 v。假设 $p(\boldsymbol{h}) = \mathcal{N}(\boldsymbol{h}; 0, \boldsymbol{I})$ 以及 $p(v|\boldsymbol{h}) = \mathcal{N}(v; \boldsymbol{w}^\top \boldsymbol{h}; 1)$，我们可以通过把 $\boldsymbol{h}$ 积掉来简化这个模型，结果是关于 v 的高斯分布。这个模型本身并不有趣。为了说明变分法如何应用在概率建模，我们构造了这个模型。

忽略归一化常数的时候，真实的后验分布可以给出：

$$p(\boldsymbol{h}|\boldsymbol{v}) \tag{10.57}$$
$$\propto p(\boldsymbol{h}, \boldsymbol{v}) \tag{10.58}$$
$$= p(h_1)p(h_2)p(\boldsymbol{v}|\boldsymbol{h}) \tag{10.59}$$
$$\propto \exp(-\frac{1}{2}[h_1^2 + h_2^2 + (v - h_1 w_1 - h_2 w_2)^2]) \tag{10.60}$$
$$= \exp(-\frac{1}{2}[h_1^2 + h_2^2 + v^2 + h_1^2 w_1^2 + h_2^2 w_2^2 - 2v h_2 w_2 - 2v h_2 w_2 + 2h_1 w_1 h_2 w_2]) \tag{10.61}$$

在上式中，我们发现由于带有 h_1, h_2 乘积的项的存在，真实的后验并不能将 h_1, h_2 分开。

展开方程 (10.56)，我们可以得到

$$\tilde{q}(h_1|\boldsymbol{v}) \tag{10.62}$$
$$= \exp(\mathbb{E}_{h_2\sim q(h_2|\boldsymbol{v})} \log \tilde{p}(\boldsymbol{v}, \boldsymbol{h})) \tag{10.63}$$
$$= \exp\Big(-\frac{1}{2}\mathbb{E}_{h_2\sim q(h_2|\boldsymbol{v})}[h_1^2 + h_2^2 + v^2 + h_1^2 w_1^2 + h_2^2 w_2^2 \tag{10.64}$$
$$- 2vh_1w_1 - 2vh_2w_2 + 2h_1w_1h_2w_2]\Big) \tag{10.65}$$

从这里，我们可以发现其中我们只需要从 $q(h_2|\boldsymbol{v})$ 中获得两个值：$\mathbb{E}_{h_2\sim q(h|\boldsymbol{v})}[h_2]$ 和 $\mathbb{E}_{h_2\sim q(h|\boldsymbol{v})}[h_2^2]$。把这两项记作 $\langle h_2\rangle$ 和 $\langle h_2^2\rangle$，我们可以得到：

$$\tilde{q}(h_1|\boldsymbol{v}) = \exp(-\frac{1}{2}[h_1^2 + \langle h_2^2\rangle + v^2 + h_1^2 w_1^2 + \langle h_2^2\rangle w_2^2 \tag{10.66}$$
$$- 2vh_1w_1 - 2v\langle h_2\rangle w_2 + 2h_1w_1\langle h_2\rangle w_2]) \tag{10.67}$$

从这里，我们可以发现 $\tilde{q}$ 形式满足高斯分布。因此，我们可以得到 $q(\boldsymbol{h}|\boldsymbol{v}) = \mathcal{N}(\boldsymbol{h}; \boldsymbol{\mu}, \boldsymbol{\beta}^{-1})$，其中 $\boldsymbol{\mu}$ 和对角的 $\boldsymbol{\beta}$ 是变分参数，我们可以使用任何方法来优化它。有必要说明一下，我们并没有假设 q 是一个高斯分布，这个高斯是使用变分法来最大化 q 关于 $\mathcal{L}$[3]推导出的。在不同的模型上应用相同的方法可能会得到不同形式的 q 分布。

当然，上述模型只是为了说明情况的一个简单例子。深度学习中关于变分学习中连续变量的实际应用可以参考**?**。

10.4.4 学习和推断之间的相互作用

在学习算法中使用近似推理会影响学习的过程，反过来这也会影响推理算法的准确性。

具体来说，训练算法倾向于以使得近似推理算法中的近似假设变得更加真实的方向来适应模型。<bad> 当训练参数时，变分学习增加

$$\mathbb{E}_{\boldsymbol{h}\sim q} \log p(\boldsymbol{v}, \boldsymbol{h}) \tag{10.68}$$

对于一个特定的 $\boldsymbol{v}$，对于 $q(\boldsymbol{h}|\boldsymbol{v})$ 中概率很大的 $\boldsymbol{h}$ 它增加了 $p(\boldsymbol{h}|\boldsymbol{v})$，对于 $q(\boldsymbol{h}|\boldsymbol{v})$ 中概率很小的 $\boldsymbol{h}$ 它减小了 $p(\boldsymbol{h}|\boldsymbol{v})$。

这种行为使得我们做的近似假设变得合理。<bad> 如果我们用单峰值的模型近似后验分布，我们将获得一个真实后验的模型，该模型比我们通过使用精确推理训练模型获得的模型更接近单峰。

[3]此处似乎有笔误。

因此，估计由于变分近似所产生的对模型的坏处是很困难的。存在几种估计 $\log p(\boldsymbol{v})$ 的方式。通常我们在训练模型之后估计 $\log p(\boldsymbol{v};\boldsymbol{\theta})$，然后发现它和 $\mathcal{L}(\boldsymbol{v},\boldsymbol{\theta},q)$ 的差距是很小的。从这里我们可以发现，对于特定的从学习过程中获得的 $\boldsymbol{\theta}$ 来说，变分近似普遍是很准确的。然而我们无法直接得到变分近似普遍很准确或者变分近似不会对学习过程产生任何负面影响这样的结论。为了准确衡量变分近似带来的危害，我们需要知道 $\boldsymbol{\theta}^* = \max_{\boldsymbol{\theta}} \log p(\boldsymbol{v};\boldsymbol{\theta})$。$\mathcal{L}(\boldsymbol{v},\boldsymbol{\theta},q) \approx \log p(\boldsymbol{v};\boldsymbol{\theta})$ 和 $\log p(\boldsymbol{v};\boldsymbol{\theta}) \ll \log p(\boldsymbol{v};\boldsymbol{\theta}^*)$ 同时成立是有可能的。如果存在 $\max_q \mathcal{L}(\boldsymbol{v},\boldsymbol{\theta}^*,q) \ll \log p(\boldsymbol{v};\boldsymbol{\theta}^*)$，即在 $\boldsymbol{\theta}^*$ 点处后验分布太过复杂使得 q 分布族无法准确描述，则我们无法学习到一个 $\boldsymbol{\theta}$。这样的一类问题是很难发现的，因为只有在我们有一个能够找到 $\boldsymbol{\theta}^*$ 的超级学习算法时，才能进行上述的比较。

10.5 learned近似推断

我们已经看到了推断可以被视作是一个增加函数 $\mathcal{L}$ 的值的一个优化过程。显式的通过迭代方法比如固定点方程或者基于梯度的优化算法来执行优化的过程通常是代价昂贵且耗时巨大的。<bad> 具体的说，我们可以将优化过程视作一个将输入 $\boldsymbol{v}$ 投影到一个近似分布 $q^* = \arg\max_q \mathcal{L}(\boldsymbol{v},q)$ 的函数 f。<bad> 一旦我们将多步的迭代优化过程看作是一个函数，我们可以用一个近似 $\hat{f}(\boldsymbol{v};\boldsymbol{\theta})$ 的神经网络来近似。

10.5.1 wake sleep

训练一个可以用 $\boldsymbol{v}$ 来推断 $\boldsymbol{h}$ 的模型的一个主要的难点在于我们没有一个有监督的训练集来训练模型。给定一个 $\boldsymbol{v}$，我们无法获知一个合适的 $\boldsymbol{h}$。从 $\boldsymbol{v}$ 到 $\boldsymbol{h}$ 的映射依赖于模型类型的选择，并且在学习过程中随着 $\boldsymbol{\theta}$ 的改变而变化。**wake sleep**(wake sleep) 算法 (**??**) 通过从模型分布中抽取 $\boldsymbol{v}$ 和 $\boldsymbol{h}$ 样本来解决这个问题。例如，在有向模型中，这可以通过执行从 $\boldsymbol{h}$ 开始并在 $\boldsymbol{v}$ 结束的原始采样来高效地完成。然后推断网络可以被训练来执行反向的映射：预测哪一个 $\boldsymbol{h}$ 产生了当前的 $\boldsymbol{v}$。<bad> 这种方法的主要缺点是我们将只能够训练推理网络在模型下具有高概率的 v 值。在学习早期，模型分布将不像数据分布，因此推理网络将不具有学习类似数据的样本的机会。

<bad> 在**??**节中，我们看到睡眠在人类和动物中的作用的一个可能的解释是，梦想可以提供蒙特卡罗训练算法用于近似无向模型的对数分割函数的负梯度的负相位样本。生物作梦的另一个可能的解释是它提供来自 $p(\boldsymbol{h},\boldsymbol{v})$ 的样本，这可以用于训练推理网络在给定 $\boldsymbol{v}$ 的情况下预测 $\boldsymbol{h}$。在某些意义上，这种解释比分割函数的解释更令人满意。如果蒙特卡罗算法仅使用梯度的正相位进行几个步骤，然后仅对梯级的负相位进行几个步骤，那么他们的结果不会很好。人类和动物通常醒来连续几个小时，然后睡着连续几个小时。这个时间表如何支持无向模型的蒙

特卡罗训练尚不清楚。然而，基于最大化 $\mathcal{L}$ 的学习算法可以通过长时间调整改进 q 和长期调整 $\boldsymbol{\theta}$ 来实现。如果生物作梦的作用是训练网络来预测 q，那么这解释了动物如何能够保持清醒几个小时（它们清醒的时间越长，$\mathcal{L}$ 和 $\log p(\boldsymbol{v})$ 之间的差距越大，但是 $\mathcal{L}$ 将保持下限）并且睡眠几个小时（生成模型本身在睡眠期间不被修改），而不损害它们的内部模型。当然，这些想法纯粹是猜测性的，没有任何坚实的证据表明梦想实现了这些目标之一。梦想也可以服务强化学习而不是概率建模，通过从动物的过渡模型采样合成经验，从来训练动物的策略。也许睡眠可以服务于一些机器学习社区尚未发现的其他目的。

10.5.2 learned推断的其它形式

这种learned近似推断策略已经被应用到了其它模型中。(**?**) 证明了在learned推断网络中的单一路径相比于在深度玻耳兹曼机中迭代均匀场固定点方程能够得到更快的推断。训练过程基于运行推理网络，然后应用均匀场的一步来改进其估计，并训练推理网络来输出这个精细的估计而不是其原始估计。

我们已经在7.8节中已经看到，预测性的稀疏分解模型训练浅层的编码器网络以预测输入的稀疏编码。这可以被看作是自动编码器和稀疏编码之间的混合。为模型设计概率语义是可能的，其中编码器可以被视为执行learned近似最大后验推断。由于其浅层的编码器，PSD 不能实现我们在均匀场推理中看到的单元之间的那种竞争。然而，该问题可以通过训练深度编码器来执行learned近似推理来补救，如 ISTA 技术 (**?**)。

近来learned近似推理已经成为了变分自编码形式的生成模型中的主要方法之一 (**??**)。在这种优美的方法中，不需要为推断网络构造显式的目标。反之，推断网络被用来定义 $\mathcal{L}$，然后调整推断网络的参数来增大 $\mathcal{L}$。这种模型在**??**节中详细描述。

我们可以使用近似推理来训练和使用大量的模型。许多模型将在下一章中被描述。

Bibliography

术语

声学 acoustic 98

对抗样本 adversarial example 53, 54

对抗训练 adversarial training 53–55

原始采样 Ancestral Sampling 155, 167, 190

专用集成电路 application-specific integrated circuit 91, 92

人工智能 artificial intelligence 1–3, 5, 6, 10–12, 86, 92, 112, 113, 136, 141

人工神经网络 artificial neural networks 8, 86

异步随机梯度下降 Asynchoronous Stochastic Gradient Descent 89

注意机制 attention mechanism 72, 85, 90, 91, 107, 108

属性 attribute 112

自动编码器 autoencoder 3, 36, 56, 59, 118, 122–139, 185, 191

自动语音识别 Automatic Speech Recognition 96, 97

反向传播 back propagation 108, 125

反向传播 backprop 92

通过时间反向传播 back-propagation through time 62–64

反向传播 backward propagation 62, 64, 76, 77, 82, 83, 85

词袋 bag of words 104

Bagging bootstrap aggregating 2, 46–49, 52

bandit bandit 111

batch normalization batch normalization 53

贝叶斯网络 Bayesian network 144, 146, 156

信念网络 belief network 144

Bernoulli 分布 Bernoulli distribution 130, 182

偏置 bias in affine function 28, 29, 31, 62, 69, 79–82, 101, 110

偏差 bias in statistics 27, 28, 104

有偏重要采样 biased importance sampling 104, 165

二元语法 bigram 99, 104

二元关系 binary relation 112

二值稀疏编码 binary sparse coding 180, 182, 184, 185

块吉布斯采样 block Gibbs Sampling 160, 169, 170

玻耳兹曼分布 Boltzmann distribution 149

玻耳兹曼机 Boltzmann Machine 149, 160, 171

Boosting Boosting 47, 52

预烧 Burning-in 168

变分法 calculus of variations 4, 179, 185, 188, 189

容量 capacity 90, 99, 105, 106, 126, 127, 129, 131, 134, 135

级联 cascade 90, 91

中心极限定理 central limit theorem 164

弦 chord 153, 154

弦图 chordal graph 154

截断梯度 clipping the gradient 2, 81, 82

团 clique 146–149, 153, 154, 156, 158, 178

团势能 clique potential 146–149

级联 coalesced 88, 91

编码 code 125–127, 129, 130, 136–139

协同过滤 collaborative filtering 109, 110

计算图 computational graph 2, 57, 58, 60, 64, 65, 73, 82

计算机视觉 Computer Vision 3, 44, 86, 92, 93, 96

条件计算 conditional computation 90

条件独立 conditionally independent 116

连接机制 connectionism 7, 9–11

约束优化 constrained optimization 1, 23, 25, 45