基 础 篇

第 1 章

深度学习简介

1.1 计算机视觉

1.1.1 定义

计算机视觉是使用计算机及相关设备对生物视觉的一种模拟。它的主要任务是通过对采集的图片或视频进行处理以获得相应场景的三维信息。计算机视觉是一门关于如何运用照相机和计算机获取人们所需的、被拍摄对象的数据与信息的学问。形象地说，就是给计算机安装上眼睛（照相机）和大脑（算法），让计算机能够感知环境。

1.1.2 基本任务

计算机视觉的基本任务包括图像处理、模式识别或图像识别、景物分析、图像理解等。除了图像处理和模式识别之外，它还包括空间形状的描述、几何建模以及认识过程。实现图像理解是计算机视觉的终极目标。下面举例说明图像处理、模式识别和图像理解。

图像处理技术可以把输入图像转换成具有所希望特性的另一幅图像。例如，可通过处理使输出图像有较高的信噪比，或通过增强处理突出图像的细节，以便于操作员的检验。在计算机视觉研究中经常利用图像处理技术进行预处理和特征抽取。

模式识别技术根据从图像抽取的统计特性或结构信息，把图像分成预定的类别。例如，文字识别或指纹识别。在计算机视觉中，模式识别技术经常用于对图像中的某些部分（例如分割区域）的识别和分类。

图像理解技术是对图像内容信息的理解。给定一幅图像，图像理解程序不仅描述图像本身，而且描述和解释图像所代表的景物，以便对图像代表的内容做出决定。在人工智能研究的初期经常使用景物分析这个术语，以强调二维图像与三维景物之间的区别。图

像理解除了需要复杂的图像处理以外，还需要具有关于景物成像的物理规律的知识以及与景物内容有关的知识。

1.1.3 传统方法

在深度学习算法出现之前，对于视觉算法来说，大致可以分为以下 5 个步骤：特征感知，图像预处理，特征提取，特征筛选，推理预测与识别。早期的机器学习，占优势的统计机器学习群体中，对特征的重视是不够的。

何为图片特征？用通俗的语言来说，即是最能表现图像特点的一组参数，常用到的特征类型有颜色特征、纹理特征、形状特征和空间关系特征。为了让机器尽可能完整且准确地理解图片，需要将包含庞杂信息的图像简化抽象为若干个特征量，以便于后续计算。在深度学习技术没有出现的时候，图像特征需要研究人员手工提取，这是一个繁杂且冗长的工作，因为很多时候研究人员并不能确定什么样的特征组合是有效的，而且常常需要研究人员去手工设计新的特征。在深度学习技术出现后，问题简化了许多，各种各样的特征提取器以人脑视觉系统为理论基础，尝试直接从大量数据中提取出图像特征。我们知道，图像是由多个像素拼接组成的，每个像素在计算机中存储的信息是其对应的 RGB 数值，一幅图片包含的数据量大小可想而知。

过去的算法主要依赖于特征算子，比如最著名的 SIFT 算子，即所谓的对尺度旋转保持不变的算子。它被广泛地应用于图像比对，特别是所谓的 structure from motion 这些应用中，有一些成功的应用例子。另一个是 HoG 算子，它可以提取比较健壮的物体边缘，在物体检测中扮演着重要的角色。

这些算子还包括 Textons、Spin image、RIFT 和 GLOH，都是在深度学习诞生之前或者深度学习真正流行起来之前，占领视觉算法的主流。

这些特征和一些特定的分类器组合得到了一些成功或半成功的例子，基本达到了商业化的要求，但还没有完全商业化。一是指纹识别算法，它已经非常成熟，一般是在指纹的图案上面去寻找一些关键点，寻找具有特殊几何特征的点，然后把两个指纹的关键点进行比对，判断是否匹配。然后是 2001 年基于 Haar 的人脸检测算法，在当时的硬件条件下已经能够达到实时人脸检测，我们现在所有手机相机里的人脸检测，都是基于它的变种。第三个是基于 HoG 特征的物体检测，它和所对应的 SVM 分类器组合起来就是著名的 DPM 算法。DPM 算法在物体检测上超过了所有的算法，取得了比较不错的成绩。但这种成功例子太少了，因为手工设计特征需要丰富的经验，需要研究人员对这个领域和数据特别了解，再设计出来特征还需要大量的调试工作。另一个难点在于，研究人员不只需要手工设计特征，还要在此基础上有一个比较合适的分类器算法。同时，设计特征然后选择一个分类器，这两者合并达到最优的效果，几乎是不可能完成的任务。

1.1.4 仿生学与深度学习

如果不手工设计特征，不挑选分类器，有没有别的方案呢？能不能同时学习特征和分类器？即输入某一个模型的时候，输入只是图片，输出就是它自己的标签。比如输入一个明星的头像，如图 1.1 所示神经网络示例，模型输出的标签就是一个 50 维的向量(如果要

在 50 个人中识别),其中对应明星的向量是 1,其他的位置是 0。

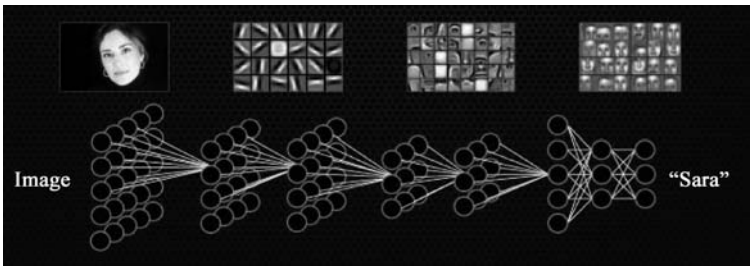

图 1.1　神经网络示例

这种设定符合人类脑科学的研究成果。1981 年,诺贝尔医学和生理学奖颁发给了神经生物学家 David Hubel。他的主要研究成果是发现了视觉系统信息处理机制,证明大脑的可视皮层是分级的。他的贡献主要有两个,一是他认为人的视觉功能一个是抽象,一个是迭代。抽象就是把非常具体、形象的元素,即原始的光线像素等信息,抽象出来形成有意义的概念。这些有意义的概念又会往上迭代,变成更加抽象、人可以感知到的抽象概念。

像素是没有抽象意义的,但人脑可以把这些像素连接成边缘,边缘相对像素来说就变成了比较抽象的概念;边缘进而形成球形,球形然后形成气球,又是一个抽象的过程,大脑最终就知道看到的是一个气球。

模拟人脑识别人脸,如图 1.2 所示,也是抽象迭代的过程,从最开始的像素到第二层的边缘,再到人脸的部分,然后到整张人脸,是一个抽象迭代的过程。

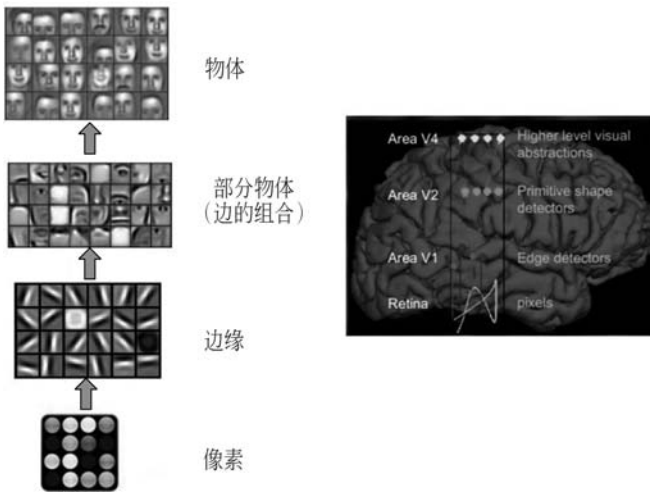

图 1.2　人脑与神经网络

再比如认识到图片中的物体是摩托车的这个过程,人脑可能只需要几秒就可以处理完毕,但这个过程中经过了大量的神经元抽象迭代。对计算机来说,最开始看到的根本也不是摩托车,而是 RGB 图像三个通道上不同的数字。

所谓的特征或者视觉特征，就是把这些数值综合起来用统计或非统计的方法，把摩托车的部件或者整辆摩托车表现出来。深度学习流行之前，大部分的设计图像特征就是基于此，即把一个区域内的像素级别的信息综合表现出来，以利于后面的分类学习。

如果要完全模拟人脑，也要模拟抽象和递归迭代的过程，把信息从最细琐的像素级别，抽象到“种类”的概念，让人能够接受。

1.1.5 现代深度学习

计算机视觉里经常使用的卷积神经网络，即 CNN，是一种对人脑比较精准的模拟。人脑在识别图片的过程中，并不是对整幅图同时进行识别，而是感知图片中的局部特征，之后再将局部特征综合起来得到整幅图的全局信息。卷积神经网络模拟了这一过程，其卷积层通常是堆叠的，低层的卷积层可以提取到图片的局部特征，例如角、边缘、线条等，高层的卷积层能够从低层的卷积层中学到更复杂的特征，从而实现对图片的分类和识别。

卷积就是两个函数之间的相互关系。在计算机视觉里面，可以把卷积当作一个抽象的过程，就是把小区域内的信息统计抽象出来。例如，对于一张爱因斯坦的照片，可以学习 n 个不同的卷积和函数，然后对这个区域进行统计。可以用不同的方法统计，比如可以着重统计中央，也可以着重统计周围，这就导致统计的函数的种类多种多样，以达到可以同时学习多个统计的累积和。

图 1.3 演示了如何从输入图像得到最后的卷积，生成相应的图。首先用学习好的卷积和对图像进行扫描，然后每个卷积和会生成一个扫描的响应图，称为响应图或者称为特征图(feature map)。如果有多个卷积和，就有多个特征图。也就是说，从一个最开始的输入图像(RGB 三个通道)可以得到 256 个通道的 feature map，因为有 256 个卷积和，每个卷积和代表一种统计抽象的方式。

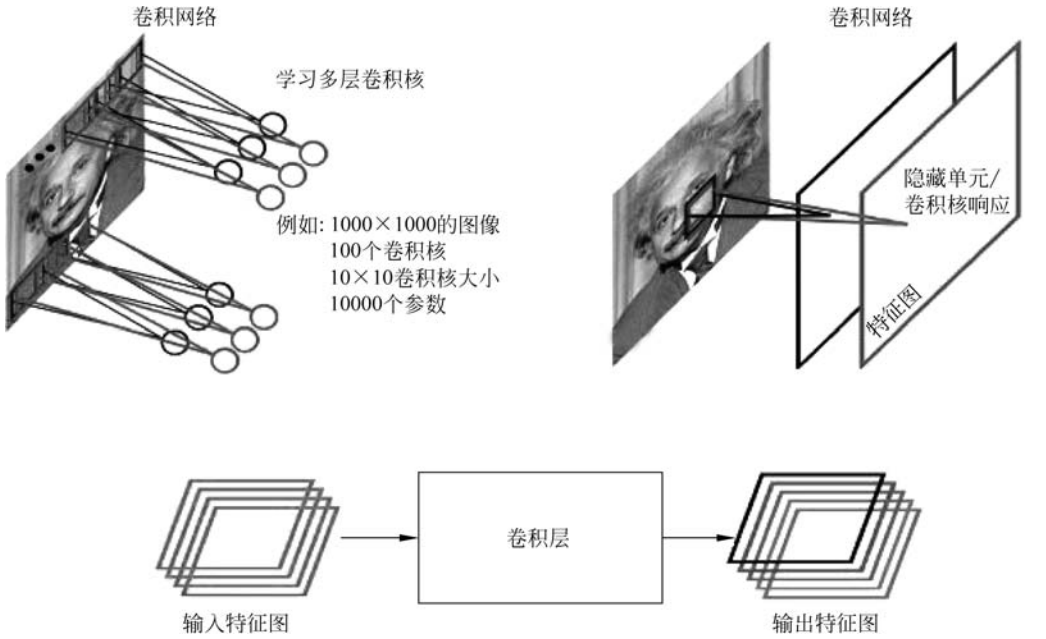

图 1.3 卷积

在卷积神经网络中，除了卷积层，还有一种叫池化的操作。池化操作在统计上的概念更明确，就是一种对一个小区域内求平均值或者求最大值的统计操作。

带来的结果是，池化操作会将输入的 feature map 的尺寸减小，让后面的卷积操作能够获得更大的视野，也降低了运算量，具有加速的作用。

在如图 1.4 所示这个例子里，池化层对每个大小为 2×2px 的区域求最大值，然后把最大值赋给生成的 feature map 的对应位置。如果输入图像的大小是 100×100px，那输出图像的大小就会变成 50×50px，feature map 变成了原来的 1/4。同时保留的信息是原来 2×2 区域里面最大的信息。

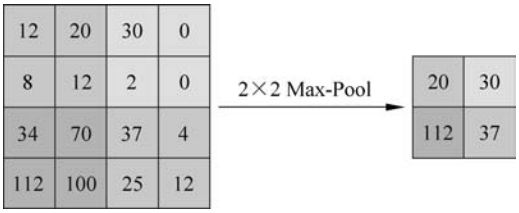

图 1.4　池化

LeNet 网络如图 1.5 所示。Le 是人工智能领域先驱 Lecun 名字的简写。LeNet 是许多深度学习网络的原型和基础。在 LeNet 之前，人工神经网络层数都相对较少，而 LeNet 5 层网络突破了这一限制。LeNet 在 1998 年即被提出，Lecun 用这一网络进行字母识别，达到了非常好的效果。

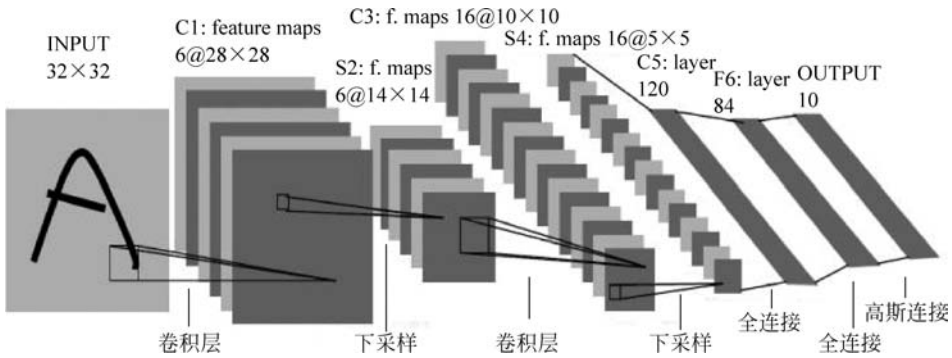

图 1.5　LeNet

LeNet 网络输入图像是大小为 32×32px 的灰度图，第一层经过了一组卷积和，生成了 6 个 28×28px 的 feature map，然后经过一个池化层，得到 6 个 14×14px 的 feature map，然后再经过一个卷积层，生成了 16 个 10×10px 的卷积层，再经过池化层生成 16 个 5×5px 的 feature map。

这 16 个大小为 5×5px 的 feature map 再经过 3 个全连接层，即可得到最后的输出结果。输出就是标签空间的输出。由于设计的是只对 0～9 进行识别，所以输出空间是 10，如果要对 10 个数字再加上 52 个大、小写字母进行识别的话，输出空间就是 62。向量各维度的值代表“图像中元素等于该维度对应标签的概率”，即若该向量第一维度输出为

0.6，即表示图像中元素是“0”的概率是0.6。那么该62维向量中值最大的那个维度对应的标签即为最后的预测结果。62维向量里，如果某一个维度上的值最大，它对应的那个字母和数字就是预测结果。

从1998年开始的15年间，深度学习领域在众多专家学者的带领下不断发展壮大。遗憾的是，在此过程中，深度学习领域没有产生足以轰动世人的成果，导致深度学习的研究一度被边缘化。直到2012年，深度学习算法在部分领域取得不错的成绩，而压在骆驼背上的最后一根稻草就是AlexNet。

AlexNet由多伦多大学提出，在ImageNet比赛中取得了非常好的效果。AlexNet识别效果超过了当时所有浅层的方法。经此一役，AlexNet在此后被不断地改进、应用。同时，学术界和工业界认识到了深度学习的无限可能。

AlexNet是基于LeNet的改进，它可以被看作LeNet的放大版，如图1.6所示。AlexNet的输入是一个大小为224×224px的图片，输入图像在经过若干个卷积层和若干个池化层后，最后经过两个全连接层泛化特征，得到最后的预测结果。

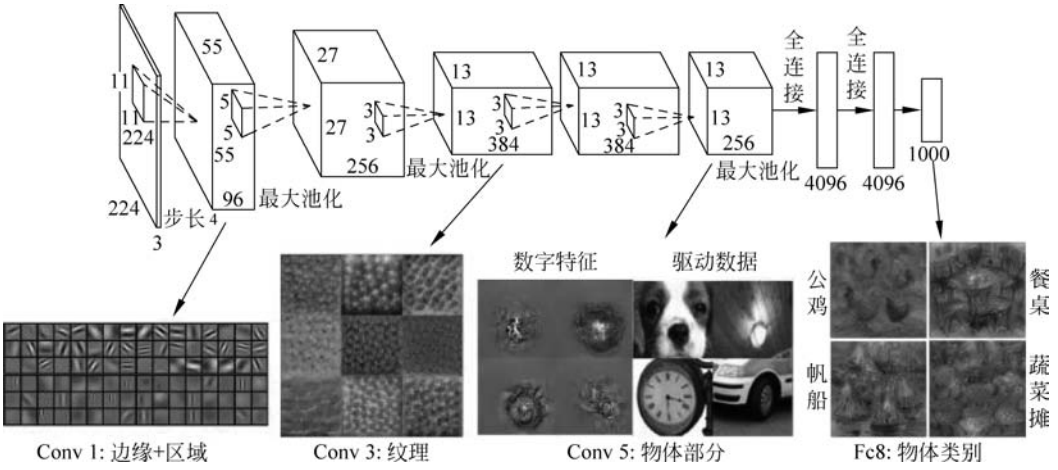

图1.6 AlexNet

2015年，特征可视化工具开始盛行。那么，AlexNet学习出的特征是什么样子的？在第一层，都是一些填充的块状物和边界等特征；中间层开始学习一些纹理特征；而在接近分类器的高层，则可以明显看到物体形状的特征；最后一层即分类层，不同物体的主要特征已经被完全提取出来。

无论对什么物体进行识别，特征提取器提取特征的过程都是渐进的。特征提取器最开始提取到的是物体的边缘特征，继而是物体的各部分信息，然后在更高层级上才能抽象到物体的整体特征。整个卷积神经网络实际上是在模拟人的抽象和迭代的过程。

1.1.6 小结

卷积神经网络的设计思路非常简洁，且很早就被提出。那为什么时隔二十多年，卷积神经网络才开始成为主流？这一问题与卷积神经网络本身的技术关系不太大，而与其他一些客观因素有关。

首先，如果卷积神经网络的深度太浅，其识别能力往往不如一般的浅层模型，比如SVM或者boosting。但如果神经网络深度过大，就需要大量数据进行训练来避免过拟合。而2006年及2007年开始，恰好是互联网开始产生大量图片数据的时期。

另外一个条件是运算能力。卷积神经网络对计算机的运算能力要求比较高，需要大量重复、可并行化的计算。在1998年CPU只有单核且运算能力比较低的情况下，不可能进行很深的卷积神经网络的训练。随着GPU计算能力的增长，卷积神经网络结合大数据的训练才成为可能。

总而言之，卷积神经网络的兴起与近些年来技术的发展是密切相关的，而这一领域的革新则不断推动了计算机视觉的发展与应用。

1.2　自然语言处理

自然语言区别于计算机所使用的机器语言和程序语言，是指人类用于日常交流的语言。而自然语言处理的目的是要让计算机来理解和处理人类的语言。

让计算机来理解和处理人类的语言也不是一件容易的事情，因为语言对于感知的抽象很多时候并不是直观的、完整的。我们的视觉感知到一个物体，就是实实在在地接收到了代表这个物体的所有像素。但是，自然语言的一个句子背后往往包含着不直接表述出来的常识和逻辑，这使得计算机在试图处理自然语言的时候不能从字面上获取所有的信息。因此自然语言处理的难度更大，它的发展与应用相比于计算机视觉也往往呈现出滞后的情况。

深度学习在自然语言处理上的应用也是如此。为了将深度学习引入这个领域，研究者尝试了许多方法来表示和处理自然语言的表层信息（如词向量、更高层次、带上下文信息的特征表示等），也尝试过许多方法来结合常识与直接感知（如知识图谱、多模态信息等）。这些研究都富有成果，其中的许多都已应用于现实中，甚至用于社会管理、商业、军事的目的。

1.2.1　自然语言处理的基本问题

自然语言处理主要研究能实现人与计算机之间用自然语言进行有效通信的各种理论和方法，其主要任务如下。

（1）**语言建模**。语言建模即计算一个句子在一种语言中出现的概率。这是一个高度抽象的问题，在第8章有详细介绍。它的一种常见形式是：给出句子的前几个词，预测下一个词是什么。

（2）**词性标注**。句子都是由单独的词汇构成的，自然语言处理有时需要标注出句子中每个词的词性。需要注意的是，句子中的词汇并不是独立的，在研究过程中，通常需要考虑词汇的上下文。

（3）**中文分词**。中文的最小自然单位是字，但单个字的意义往往不明确或者含义较多，并且在多语言的任务中与其他以词为基本单位的语言不对等。因此不论是从语言学特性还是从模型设计的角度来说，都需要将中文句子恰当地切分为单个的词。

(4) **句法分析**。由于人类表达的时候只能逐词地按顺序说,因此自然语言的句子也是扁平的序列。但这并不代表着一个句子中不相邻的词之间就没有关系,也不代表着整个句子中的词只有前后关系。它们之间的关系是复杂的,需要用树状结构或图才能表示清楚。句法分析中,人们希望通过明确句子内两个或多个词的关系来了解整个句子的结构。句法分析的最终结果是一棵句法树。

(5) **情感分类**。给出一个句子,我们希望知道这个句子表达了什么情感:有时候是正面/负面的二元分类,有时候是更细粒度的分类;有时候是仅给出一个句子,有时候是指定对于特定对象的态度/情感。

(6) **机器翻译**。最常见的是把源语言的一个句子翻译成目标语言的一个句子。与语言建模相似,给定目标语言一个句子的前几个词,预测下一个词是什么,但最终预测出来的整个目标语言句子必须与给定的源语言句子具有完全相同的含义。

(7) **阅读理解**。有许多形式。有时候是输入一个段落或一个问题,生成一个回答(类似问答),或者在原文中标定一个范围作为回答(类似从原文中找对应句子),有时候是输出一个分类(类似选择题)。

1.2.2 传统方法与神经网络方法的比较

1. 人工参与程度

传统的自然语言处理方法中,人参与得非常多。比如基于规则的方法就是由人完全控制,人用自己的专业知识完成了对一个具体任务的抽象和建立模型,对模型中一切可能出现的案例提出解决方案,定义和设计了整个系统的所有行为。这种人过度参与的现象到基于传统统计学方法出现以后略有改善,人们开始让步对系统行为的控制;被显式构建的是对任务的建模和对特征的定义,然后系统的行为就由概率模型来决定了,而概率模型中的参数估计则依赖于所使用的数据和特征工程中所设计的输入特征。到了深度学习的时代,特征工程也不需要了,人只需要构建一个合理的概率模型,特征抽取就由精心设计的神经网络架构来完成;甚至于当前人们已经在探索神经网络架构搜索的方法,这意味着人们对于概率模型的设计也部分地交给了深度学习代劳。

总而言之,人的参与程度越来越低,但系统的效果越来越好。这是合乎直觉的,因为人对于世界的认识和建模总是片面的、有局限性的。如果可以将自然语言处理系统的构建自动化,将其基于对世界的观测点(即数据集),所建立的模型和方法一定会比人类的认知更加符合真实的世界。

2. 数据量

随着自然语言处理系统中人工参与的程度越来越低,系统的细节就需要更多的信息来决定,这些信息只能来自更多的数据。今天当我们提到神经网络方法时,都喜欢把它描述成为“数据驱动的方法”。

从人们使用传统的统计学方法开始,如何取得大量的标注数据就已经是一个难题。随着神经网络架构日益复杂,网络中的参数也呈现爆炸式的增长。特别是近年来深度学习加速硬件的算力突飞猛进,人们对于使用巨量的参数更加肆无忌惮,这就显得数据量日

益捉襟见肘。特别是一些低资源的语言和领域中，数据短缺问题更加严重。

这种数据的短缺，迫使人们研究各种方法来提高数据利用效率(data efficiency)。于是 zero-shot learning、domain adaptation 等半监督乃至非监督的方法应运而生。

3. 可解释性

人工参与程度的降低带来的另一个问题是模型的可解释性越来越低。在理想状况下，如果系统非常有效，人们根本不需要关心黑盒系统的内部构造。但事实是自然语言处理系统的状态离完美还有相当大的差距，因此当模型出现问题的时候，人们总是希望知道问题的原因，并且找到相应的办法来避免或修补。

一个模型能允许人们检查它的运行机制和问题成因，允许人们干预和修补问题，要做到这一点是非常重要的，尤其是对于一些商用生产的系统来说。传统基于规则的方法中，一切规则都是由人手动规定的，要更改系统的行为非常容易；而在传统的统计学方法中，许多参数和特征都有明确的语言学含义，要想定位或者修复问题通常也可以做到。

然而现在主流的神经网络模型都不具备这种能力，它们就像黑箱子，你可以知道它有问题，或者有时候可以通过改变它的设定来大致猜测问题的可能原因；但要想控制和修复问题则往往无法在模型中直接完成，而要在后处理(post-processing)的阶段重新拾起旧武器——基于规则的方法。

这种隐忧使得人们开始探索如何提高模型的可解释性这一领域。主要的做法包括试图解释现有的模型和试图建立透明度较高的新模型。然而要做到完全理解一个神经网络的行为并控制它，还有很长的路要走。

1.2.3 发展趋势

从传统方法和神经网络方法的对比中，可以看出自然语言处理的模型和系统构建是向着越来越自动化、模型越来越通用的趋势发展的。

一开始，人们试图减少和去除人类专家知识的参与。因此就有了大量的网络参数、复杂的架构设计，这些都是通过在概率模型中提供潜在变量(latent variable)，使得模型具有捕捉和表达复杂规则的能力。这一阶段，人们渐渐地摆脱了人工制定的规则和特征工程，同一种网络架构可以被许多自然语言任务通用。

之后，人们觉得每一次为新的自然语言处理任务设计一个新的模型架构并从头训练的过程过于烦琐，于是试图开发利用这些任务底层所共享的语言特征。在这一背景下，迁移学习逐渐发展，从前神经网络时代的 LDA、Brown Clusters，到早期深度学习中的预训练词向量 word2vec、Glove 等，再到今天家喻户晓的预训练语言模型 ELMo、BERT。这使得不仅是模型架构可以通用，连训练好的模型参数也可以通用了。

现在人们希望神经网络的架构都可以不需要设计，而是根据具体的任务和数据来搜索得到。这一新兴领域方兴未艾，可以预见，随着研究的深入，自然语言处理的自动化程度一定会得到极大的提高。

1.3 强化学习

1.3.1 什么是强化学习

强化学习是机器学习的一个重要分支，它与非监督学习、监督学习并列为机器学习的三类主要学习方法，三者之间的关系如图 1.7 所示。强化学习强调如何基于环境行动，以取得最大化的预期利益，所以强化学习可以被理解为决策问题。它是多学科、多领域交叉的产物，其灵感来自心理学的行为主义理论，即有机体如何在环境给予的奖励或惩罚的刺激下，逐步形成对刺激的预期，产生能获得最大利益的习惯性行为。强化学习的应用范围非常广泛，各领域对它的研究重点各有不同，在本书中，不对这些分支展开讨论，而专注于强化学习的通用概念。

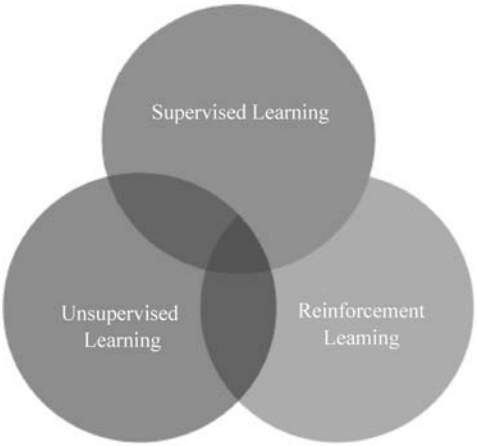

图 1.7 强化学习、监督学习、非监督学习关系示意图

在实际应用中，人们常常会把强化学习、监督学习和非监督学习这三者混淆，为了更深刻地理解强化学习和它们之间的区别，首先介绍监督学习和非监督学习的概念。

监督学习是通过带有标签或对应结果的样本训练得到一个最优模型，再利用这个模型将所有的输入映射为相应的输出，以实现分类。

非监督学习即在样本的标签未知的情况下，根据样本间的相似性对样本集进行聚类，使类内差距最小化，学习出分类器。

上述两种学习方法都会学习到输入到输出的一个映射，它们学习到的是输入和输出之间的关系，可以告诉算法什么样的输入对应着什么样的输出，而强化学习得到的是反馈，它是在没有任何标签的情况下，通过先尝试做出一些行为、得到一个结果，通过这个结果是对还是错的反馈，调整之前的行为。在不断的尝试和调整中，算法学习到在什么样的情况下选择什么样的行为可以得到最好的结果。此外，监督学习的反馈是即时的，而强化学习的结果反馈有延时，很可能需要走了很多步以后才知道之前某一步的选择是好还是坏。

1. 强化学习的 4 个元素

强化学习主要包含 4 个元素：智能体(agent)、环境状态(state)、行动(action)、反馈(reward)，它们之间的关系如图 1.8 所示，详细定义如下。

agent：智能体是执行任务的客体，只能通过与环境互动来提升策略。

state：在每个时间节点，agent 所处的环境的表示即为环境状态。

action：在每个环境状态中，agent 可以采取的动作即为行动。

reward：每到一个环境状态，agent 就有可能会收到一个反馈。

2. 强化学习算法的目标

强化学习算法的目标就是获得最多的累计奖励(正反馈)。以“幼童学习走路”为例，幼童需要自主学习走路，没有人指导他应该如何完成“走路”，他需要通过不断的尝试和获取外界对他的反馈来学习走路。

在此例中，如图 1.8 所示，幼童即为 agent，“走路”这个任务实际上包含以下几个阶段：站起来，保持平衡，迈出左腿，迈出右腿……幼童采取行动做出尝试，当他成功完成了某个子任务时(如站起来等)，他就会获得一个巧克力(正反馈)；当他做出了错误的动作时，他会被轻轻拍打一下(负反馈)。幼童通过不断地尝试和调整，找出了一套最佳的策略，这套策略能使他获得最多的巧克力。显然，他学习到的这套策略能使他顺利完成“走路”这个任务。

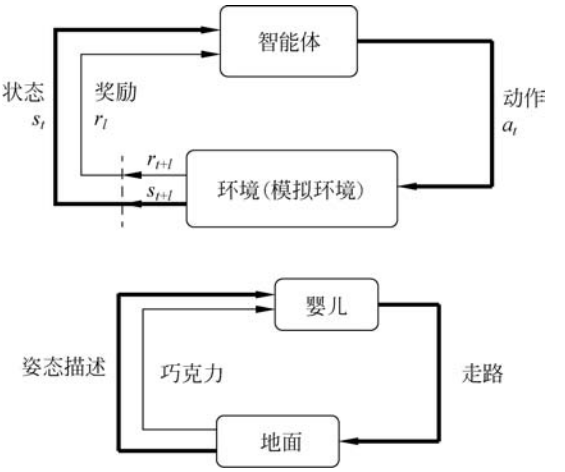

图 1.8　强化学习的 4 个元素

3. 特征

(1) 没有监督者，只有一个反馈信号。

(2) 反馈是延迟的，不是立即生成的。

(3) 强化学习是序列学习，时间在强化学习中具有重要的意义。

(4) agent 的行为会影响以后所有的决策。

1.3.2 强化学习算法简介

强化学习主要可以分为 Model-Free(无模型的)和 Model-Based(有模型的)两大类。Model-Free 算法又分成基于概率的和基于价值的。

1. Model-Free 和 Model-Based

如果 agent 不需要去理解或计算出环境模型,算法就是 Model-Free 的;相应地,如果需要计算出环境模型,那么算法就是 Model-Based 的。实际应用中,研究者通常用如下方法进行判断:在 agent 执行它的动作之前,它是否能对下一步的状态和反馈做出预测?如果可以,那么就是 Model-Based 方法;如果不能,即为 Model-Free 方法。

两种方法各有优劣。Model-Based 方法中,agent 可以根据模型预测下一步的结果,并提前规划行动路径。但真实模型和学习到的模型是有误差的,这种误差会导致 agent 虽然在模型中表现很好,但是在真实环境中可能达不到预期结果。Model-Free 的算法看似随意,但这恰好更易于研究者们去实现和调整。

2. 基于概率的算法和基于价值的算法

基于概率的算法是指直接输出下一步要采取的各种动作的概率,然后根据概率采取行动。每种动作都有可能被选中,只是可能性不同。基于概率的算法的代表算法为 policy-gradient,而基于价值的算法输出的则是所有动作的价值,然后根据最高价值来选择动作。相比基于概率的方法,基于价值的决策部分更为死板——只选价值最高的,而基于概率的,即使某个动作的概率最高,但是还是不一定会选到它。基于价值的算法的代表算法为 Q-Learning。

1.3.3 强化学习的应用

1. 交互性检索

交互性检索是在检索用户不能构建良好的检索式(关键词)的情况下,通过与检索平台交流互动并不断修改检索式,从而获得较准确检索结果的过程。

当用户想要搜索一个竞选演讲(Wu & Lee,INTERSPEECH 16)时,他不能提供直接的关键词,其交互性搜索过程如图 1.9 所示。在交互性检索中,机器作为 agent,在不断的尝试中(提供给用户可能的问题答案)接受来自用户的反馈(对答案的判断),最终找到符合要求的结果。

2. 新闻推荐

新闻推荐,如图 1.10 所示。一次完整的推荐过程包含以下过程:一个用户单击 App 底部刷新或者下拉,后台获取到用户请求,并根据用户的标签召回候选新闻,推荐引擎则对候选新闻进行排序,最终给用户推出 10 条新闻。如此往复,直到用户关闭 App,停止

浏览新闻。将用户持续浏览新闻的推荐过程看成一个决策过程，就可以通过强化学习学习每一次推荐的最佳策略，从而使得用户从开始打开 App 到关闭 App 这段时间内的点击量最高。

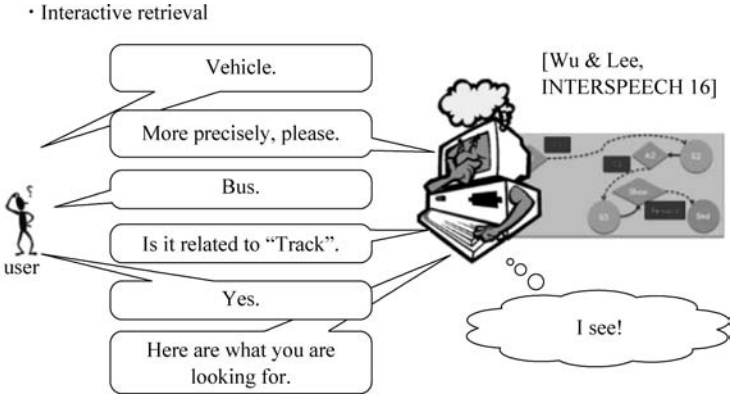

图 1.9　交互性检索

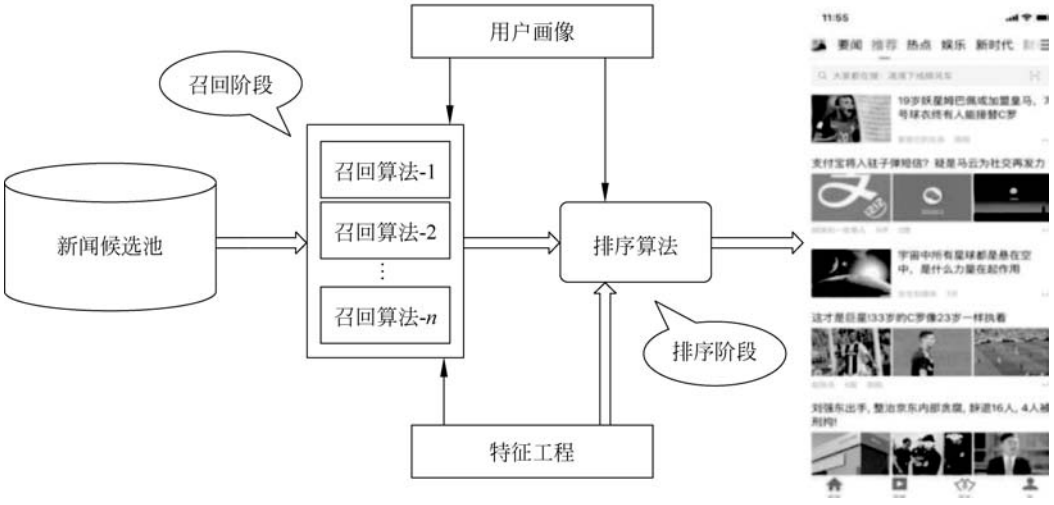

图 1.10　新闻推荐

在此例中，推荐引擎作为 agent，通过连续的行动即推送 10 篇新闻，获取来自用户的反馈，即单击：如果用户浏览了新闻，则为正反馈，否则为负反馈，从中学习出奖励最高（点击量最高）的策略。

第 2 章

深度学习框架

2.1 Caffe

2.1.1 Caffe 简介

Caffe 全称为 Convolutional Architecture for Fast Feature Embedding，是一种常用的深度学习框架，是一个清晰的、可读性高的、快速的深度学习框架，主要应用在视频、图像处理方面。Caffe 的官网是 http://caffe.berkeleyvision.org/。

Caffe 是第一个主流的工业级深度学习工具，专精于图像处理。它有很多扩展，但是由于一些遗留的架构问题，不够灵活，且对递归网络和语言建模的支持很差。在基于层的网络结构方面，Caffe 的扩展性不好。若用户如果想要增加层，则需要自己实现网络层的前向、后向和梯度更新。

2.1.2 Caffe 的特点

Caffe 的基本工作流程是设计建立在神经网络中的一个简单假设，所有的计算都是以层的形式表示的，网络层所做的事情就是输入数据，然后输出计算结果。比如卷积就是输入一幅图像，然后和这一层的参数(filter)做卷积，最终输出卷积结果。每层需要两种函数计算，一种是 forward，从输入计算到输出；另一种是 backward，从上层给的 gradient 来计算相对于输入层的 gradient。这两个函数实现之后，就可以把许多层连接成一个网络，这个网络输入数据(图像，语音或其他原始数据)，然后计算需要的输出(比如识别的标签)。在训练的时候，可以根据已有的标签计算 loss 和 gradient，然后用 gradient 来更新网络中的参数。

Caffe 是一个清晰而高效的深度学习框架，它基于纯粹的 C++/CUDA 架构，支持命令行、Python 和 MATLAB 接口，可以在 CPU 和 GPU 之间无缝切换。它的模型与优化都是通过配置文件来设置的，无需代码。Caffe 设计之初就做到了尽可能的模块化，允许

对数据格式、网络层和损失函数进行扩展。Caffe 的模型定义是用 Protocol Buffer(协议缓冲区)语言写进配置文件的,以任意有向无环图的形式。Caffe 会根据网络需要正确占用内存,通过一个函数调用实现 CPU 和 GPU 之间的切换。Caffe 每个单一的模块都对应一个测试,使得测试的覆盖非常方便,同时提供 Python 和 MATLAB 接口,用这两种语言进行调用都是可行的。

2.1.3　Caffe 概述

Caffe 是一种对新手非常友好的深度学习框架模型,它的相应优化都是以文本形式而非代码形式给出。Caffe 中的网络都是有向无环图的集合,可以直接定义,如图 2.1 所示。

数据及其导数以 blobs 的形式在层间流动,Caffe 层的定义由两部分组成:层属性与层参数,如图 2.2 所示。

```
name: "dummy-net"
layers {name: "data" …}
layers {name: "conv" …}
layers {name: "pool" …}
layers {name: "loss" …}
```

图 2.1　Caffe 网络定义

```
name:"conv1"
type:CONVOLUTION
bottom:"data"
top:"conv1"
convolution_param{
    num_output:20
    kernel_size:5
    stride:1
    weight_filler{
        type:"xavier"
    }
}
```

图 2.2　Caffe 层定义

这段配置文件的前 4 行是层属性,定义了层名称、层类型以及层连接结构(输入 blob 和输出 blob);而后半部分是各种层参数。blob 是用以存储数据的 4 维数组,例如对于数据:Number×Channel×Height×Width;对于卷积权重:Output×Input×Height×Width;对于卷积偏置:Output×1×1×1。

在 Caffe 模型中,网络参数的定义也非常方便,可以随意像图 2.3 中那样设置相应参数。感觉上更像是配置服务器参数而不像是代码。

```
# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 100
# Carry out testing every 500 training iterations.
test_interval: 500
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 10000
# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "lenet"
# solver mode: CPU or GPU
solver mode: GPU
```

图 2.3　Caffe 参数配置

2.2 TensorFlow

2.2.1 TensorFlow 简介

TensorFlow 是一个采用数据流图(data flow graph)用于数值计算的开源软件库。节点(node)在图中表示数学操作,图中的线(edge)则表示在节点间相互联系的多维数据数组,即张量(tensor)。它灵活的架构让用户可以在多种平台上展开计算,例如,台式计算机中的一个或多个 CPU(或 GPU)、服务器、移动设备等。TensorFlow 最初由 Google 大脑小组(隶属于 Google 机器智能研究机构)的研究员和工程师们开发出来,用于机器学习和深度神经网络方面的研究,但这个系统的通用性使其也可广泛用于其他计算领域。

2.2.2 数据流图

如图 2.4 所示,数据流图用“节点”(node)和“线”(edge)的有向图来描述数学计算。“节点” 一般用来表示施加的数学操作,但也可以表示数据输入(feed in)的起点/输出(push out)的终点,或者是读取/写入持久变量(persistent variable)的终点。“线”表示“节点”之间的输入/输出关系。这些数据“线”可以输运“size 可动态调整”的多维数据数组,即“张量”(tensor)。张量从图中流过的直观图像是这个工具取名为 TensorFlow 的原因。一旦输入端的所有张量准备好,节点将被分配到各种计算设备完成异步并行的运算。

2.2.3 TensorFlow 的特点

TensorFlow 不是一个严格的“神经网络”库。只要用户可以将计算表示为一个数据流图就可以使用 TensorFlow。用户负责构建图,描写驱动计算的内部循环。TensorFlow 提供有用的工具来帮助用户组装“子图”,当然用户也可以自己在 TensorFlow 基础上写自己的“上层库”。定义新复合操作和写一个 Python 函数一样容易。TensorFlow 的可扩展性相当强,如果用户找不到想要的底层数据操作,也可以自己写一些 C++代码来丰富底层的操作。

TensorFlow 在 CPU 和 GPU 上运行,比如可以运行在台式计算机、服务器、手机移动设备上等。TensorFlow 支持将训练模型自动在多个 CPU 上规模化运算,以及将模型迁移到移动端后台。

基于梯度的机器学习算法会受益于 TensorFlow 自动求微分的能力。作为 TensorFlow 用户,只需要定义预测模型的结构,将这个结构和目标函数(objective function)结合在一起,并添加数据,TensorFlow 将自动为用户计算相关的微分导数。计算某个变量相对于其他变量的导数仅仅是通过扩展用户的图来完成的,所以用户能一直清楚看到究竟在发生什么。

TensorFlow 还有一个合理的 C++使用界面,也有一个易用的 Python 使用界面来构建和执行用户的图。用户可以直接写 Python/C++程序,也可以通过交互式的 IPython 界面使用 TensorFlow 尝试实现一些想法,它可以帮用户将笔记、代码、可视化内容等有条理地归置好。

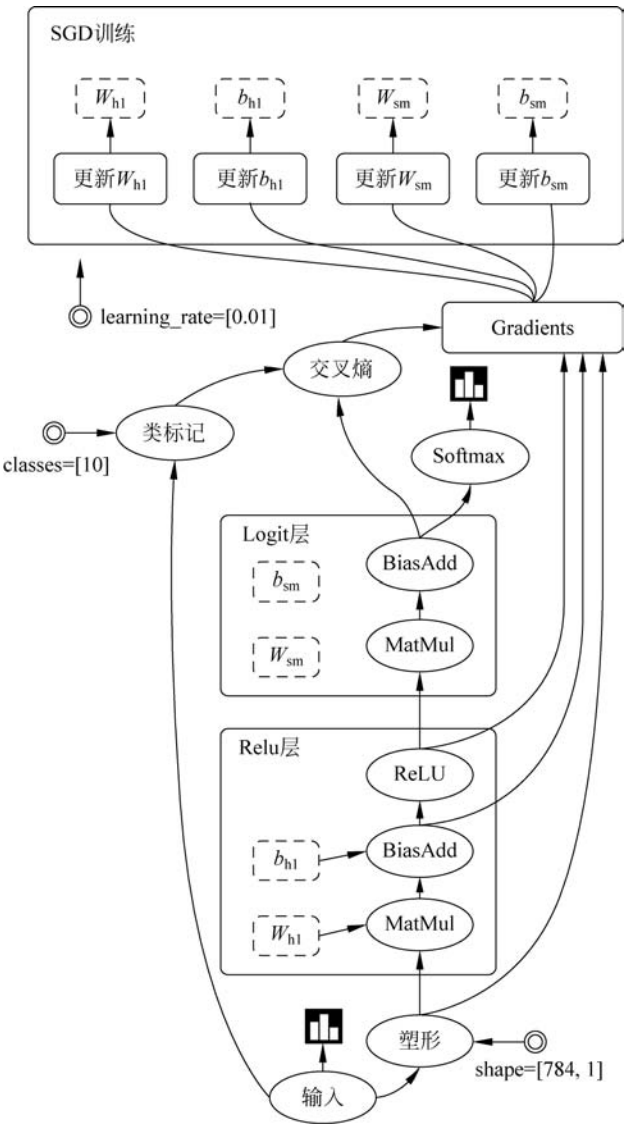

图 2.4　数据流图

2.2.4　TensorFlow 概述

TensorFlow 中的 Flow，也就是流，是其完成运算的基本方式。流是指一个计算图或简单的一个图，图不能形成环路，图中的每个节点代表一个操作，如加法、减法等。每个操作都会导致新的张量形成。

图 2.5 展示了一个简单的计算图，所对应的表达式为：$e=(a+b)(b+1)$。计算图具有以下属性：叶子节点或起始节点始终是张量。意即，操作永远不会发生在图的开头，由此可以推断，图中的每个操作都应该接受一个张量并产生一个新的张量。同样，张量不能作为非叶子节点出现，这意味着它们应始终作为输入提供给操作/节点。计算图总是以层

次顺序表达复杂的操作。通过将 $a+b$ 表示为 c，将 $b+1$ 表示为 d，可以分层次组织上述表达式。因此，可以将 e 写为：$e=c\times d$，这里 $c=a+b$ 且 $d=b+1$。以反序遍历图形而形成子表达式，这些子表达式组合起来形成最终表达式。正向遍历时，遇到的顶点总是成为下一个顶点的依赖关系，例如，没有 a 和 b 就无法获得 c，同样地，如果不解决 c 和 d 则无法获得 e。同级节点的操作彼此独立，这是计算图的重要属性之一。当按照图 2.5 所示的方式构造一个图时，很自然的是，在同一级中的节点，例如 c 和 d，彼此独立，这意味着没有必要在计算 d 之前计算 c，因此它们可以并行执行。

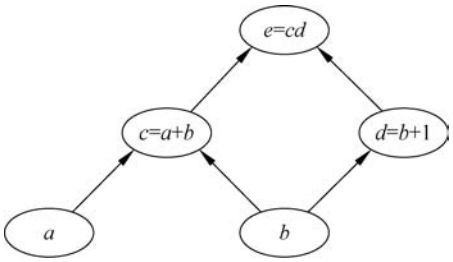

图 2.5　计算图

上文提到的最后一个属性，计算图的并行当然是最重要的属性之一。它清楚地表明，同级的节点是独立的，这意味着在 c 被计算之前不需空闲，可以在计算 c 的同时并行计算 d。TensorFlow 充分利用了这个属性。

TensorFlow 允许用户使用并行计算设备更快地执行操作。计算的节点或操作自动调度进行并行计算。这一切都发生在内部，例如在图 2.5 中，可以在 CPU 上调度操作 c，在 GPU 上调度操作 d。图 2.6 展示了两种分布式执行的过程。

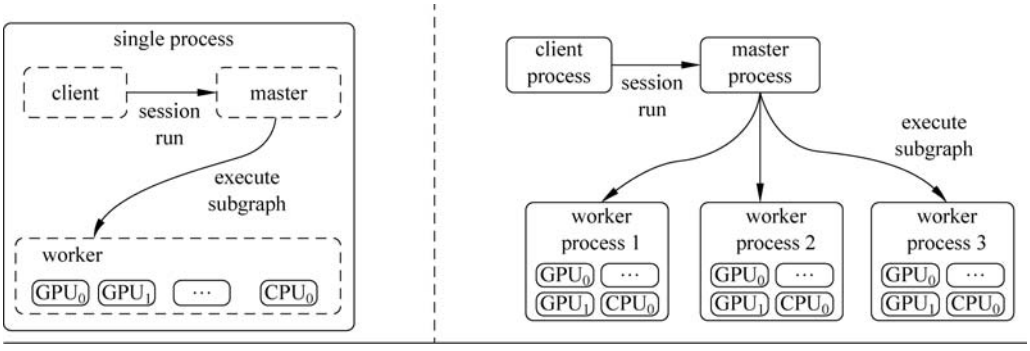

图 2.6　TensorFlow 的并行

如图 2.6 所示，第一种是单个系统分布式执行，其中单个 TensorFlow 会话(将在稍后解释)创建单个 worker，并且该 worker 负责在各设备上调度任务。在第二种系统下有多个 worker，他们可以在同一台机器上或不同的机器上，每个 worker 都在自己的上下文中运行。在图 2.6 中，worker 进程 1 运行在独立的机器上，并调度所有可用设备进行计算。

计算子图是主图的一部分，其本身就是计算图。例如，在图 2.5 中，可以获得许多子图，其中之一如图 2.7 所示。

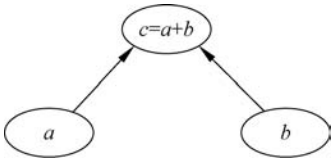

图 2.7　计算子图

图 2.7 是主图的一部分，从属性 2 可以说子图总是表示一个子表达式，因为 c 是 e 的子表达式。子图也满足最后一个属性。同一级别的子图也相互独立，可以并行