第十章　深度学习

10.1　简介

10.1 简 介

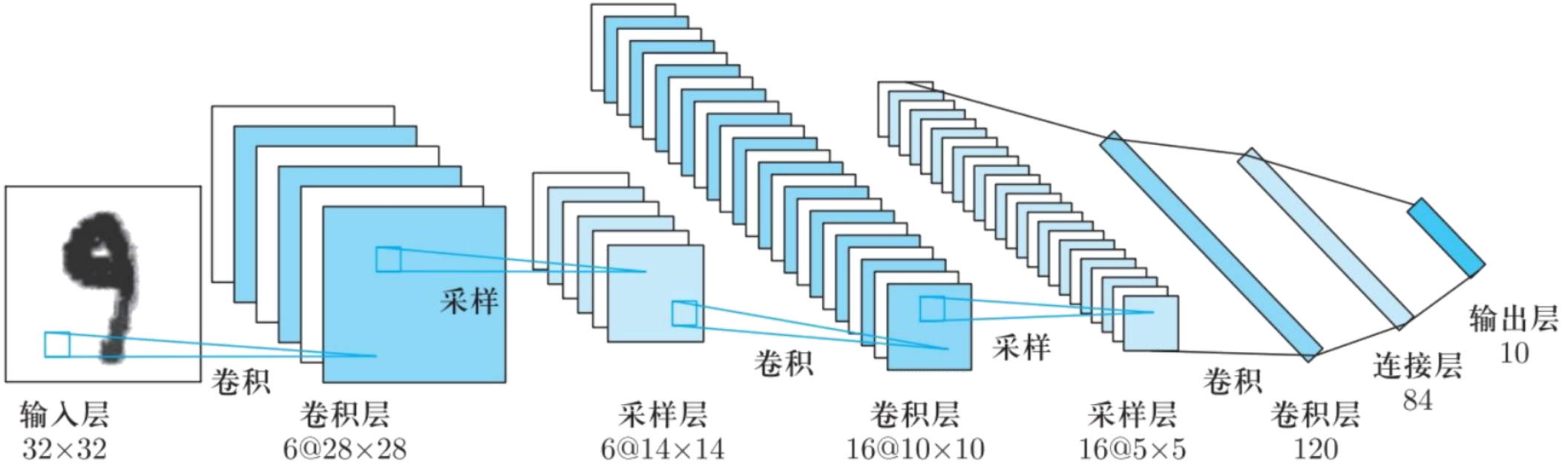
- 深度学习和人工神经网络之间存在密切的关系, 人工神经网络是深度学习的基础, 可以将人工神经网络视为深度学习的基本组成单元. 深度学习通过使用深层次的神经网络 (深度神经网络), 强调通过层次化的方式学习数据的表示, 每一层对数据进行不同层次的特征提取, 能够学习更复杂的特征.
- 理论上来说, 参数越多的模型复杂度越高、容量 (capacity) 越大, 这意味着它能完成越复杂的学习任务. 但一般情形下, 复杂模型的训练效率低, 易陷入过拟合, 因此难以受到人们青睐. 而随着云计算、大数据时代的到来, 计算能力的大幅提高可缓解训练低效性, 训练数据的大幅增加则可降低过拟合风险, 因此, 以深度学习为代表的复杂模型开始受到人们的关注.
- 典型的深度学习模型就是有许多层的神经网络. 显然, 对神经网络模型,提高容量的一个简单办法是增加隐含层的数目. 隐含层多了, 相应的神经元连接权、阈值等参数就会更多. 模型复杂度也可通过单纯增加隐含层神经元的数目来实现, 从上一章的学习中可以了解道, 单隐层的多层前馈网络已具有很强大的学习能力; 但从增加模型复杂度的角度来看, 增加隐含层的数目显然比增加隐含层神经元的数目更有效, 因为增加隐含层数不仅增加了拥有激活函数的神经元数目, 还增加了激活函数嵌套的层数.

10.1 简 介

- 然而, 多隐层神经网络难以直接用经典算法 (例如标准 BP 算法) 进行训练, 因为误差在多隐层内反向传播时, 往往会发散而不能收敛到稳定状态.
- 一种节省训练开销的策略是权共享 (weight sharing), 即让一组神经元使用相同的连接权. 这个策略在卷积神经网络 (convolutional neural network,CNN)[116] [117] 中发挥了重要作用.
- 以 CNN 进行手写数字识别任务为例[117], 如图 10.1 所示, 网络输入是一个 32×32 的手写数字图像, 输出是其识别结果, CNN 复合多个 “卷积层” 和“采样层” 对输入信号进行加工, 然后在连接层实现与输出目标之间的映射.每个卷积层都包含多个特征映射, 每个特征映射 (feature map) 是一个由多个神经元构成的 “平面”, 通过一种卷积滤波器提取输入的一种特征. 例如, 图10.1 中第一个卷积层由 6 个特征映射构成, 每个特征映射是一个 28×28 的神经元阵列, 其中每个神经元负责从 5×5 的区域通过卷积滤波器提取局部特征.采样层亦称为池化层 (pooling), 其作用是基于局部相关性原理进行亚采样, 从而在减少数据量的同时保留有用信息.例如图 10.1 中第一个采样层有6 个 14×14 的特征映射, 其中每个神经元与上一层中对应特征映射的 2×2 邻域相连, 并据此计算输出.

10.1 简 介

- 通过复合卷积层和采样层, 图 10.1 中的 CNN 将原始图像映射成 120 维特征向量, 最后通过一个由 84 个神经元构成的连接层和输出层连接完成识别任务. CNN 可用 BP 算法进行训练, 但在训练中, 无论是卷积层还是采样层, 其每一组神经元 (即图 10.1 中的每个 “平面”) 都是用相同的连接权, 从而大幅减少了需要训练的参数数目.



10.2　卷积神经网络

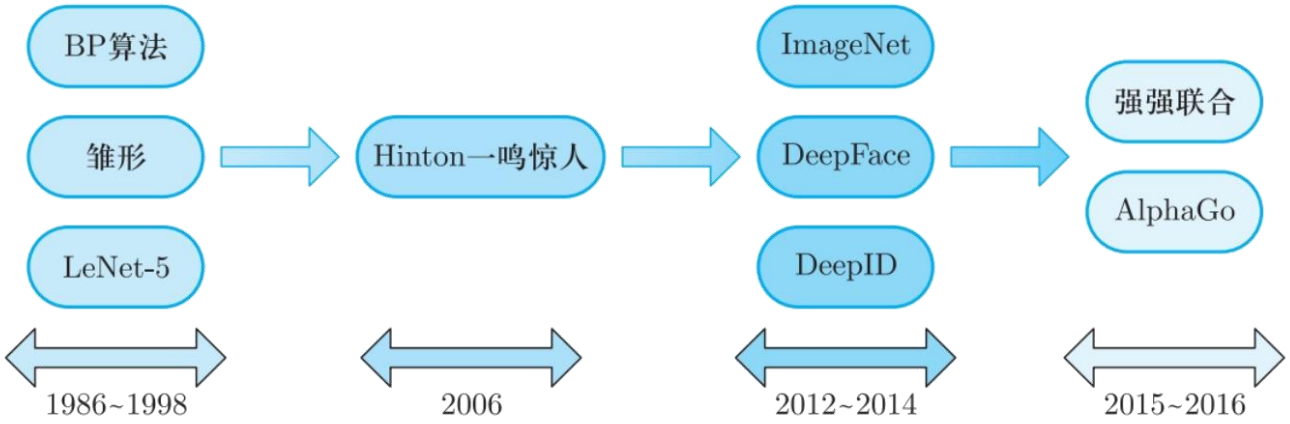
10.2 卷积神经网络

- 神经网络在得到广泛应用的同时, 参数过多、容易发生过拟合和训练时间长等缺点也暴露出来. 能否减少神经网络中参数的数目, 并进一步提升神经网络的性能? 卷积神经网络应运而生.
- 卷积神经网络 (CNN) 又称为卷积网络, 是在图像处理和计算机视觉领域应用较为广泛的一种神经网络. 与全连接神经网络同属于神经网络模型, 相对于全连接神经网络而言, 卷积神经网络的不同之处在于加入了卷积层 (con_x0002_volution) 和池化层 (pooling) 两种结构, 这两种结构是 CNN 必不可少的结构.下文将详细介绍它们的计算原理.
- CNN 最早可以追溯到 1986 年 BP 算法的提出[118], 1989 年 LeCun[119] 将BP 算法用到多层神经网络中, 1998 年 LeCun[117] 提出 LeNet-5 模型, 卷积神经网络的雏形完成. 在接下来近十年的时间里, 卷积神经网络的相关研究趋于停滞, 原因有两个: 一是研究人员意识到多层神经网络在进行 BP 训练时的计算量极其大, 当时的硬件计算能力完全不可能实现; 二是包括支持向量机在内的浅层机器学习算法也渐渐开始崭露头角.直到 2006 年, Hinton[120]在 Science 发文, 指出 “多隐层神经网络具有更为优异的特征学习能力, 并且其在训练上的复杂度可以通过逐层初始化来有效缓解”, 深度学习开始觉醒,并逐渐走入人们的视线.

10.2 卷积神经网络

- 2009 年, 李飞飞牵头建立了 ImageNet 数据集, 该数据集包含图片的种类和数量远远超过以往所有的数据集. 自 2010 年以来, 每年都举行 ImageNet 大规模视觉识别挑战赛 (ILSVRC), 研究团队在给定的数据集上评估其算法, 并在几项视觉识别任务中争夺更高的准确性. 在刚开始的两年, 冠军被支持向量机取得, 而在 2012 年, AlexNet 取得了比赛的冠军,识别的正确率得到明显提升, 这之后占据高位的一直是卷积神经网络, 包括我们现在所熟知的 VGGNet, GooLeNet, ResNet 等. 2016 年, AlphaGo 战胜了围棋世界冠军、职业九段棋手李世石, 更是将深度学习和卷积神经网络推向了高潮.

- 图 10.2 展示了近些年来 CNN 的发展历程.



10.2.1 卷积运算

- 卷积运算有连续和离散两种定义, 一维情形下有如下所示:

$$(f*g)(n)=\int_{-\infty}^{\infty} f(t)g(n-t)\mathrm{d}t,$$

$$(f*g)(n)=\sum_{-\infty}^{\infty} f(t)g(n-t).$$

  - 从公式中可以发现, 卷积运算先对函数g进行翻转, 相当于在数轴上把函数g从右边折到左边去, 也就是卷积中的 “卷”. 然后再把函数 g 平移到 n, 在这个位置对两个函数的对应点相乘, 然后相加, 这个过程是卷积中的 “积”. 因此,所谓 “卷积”, 可直观理解为 “卷” 和 “积” 两个过程.

- 在卷积神经网络中 “卷” 和 “积” 如何体现呢? 下面通过图 10.3 说明卷积神经网络的计算过程.

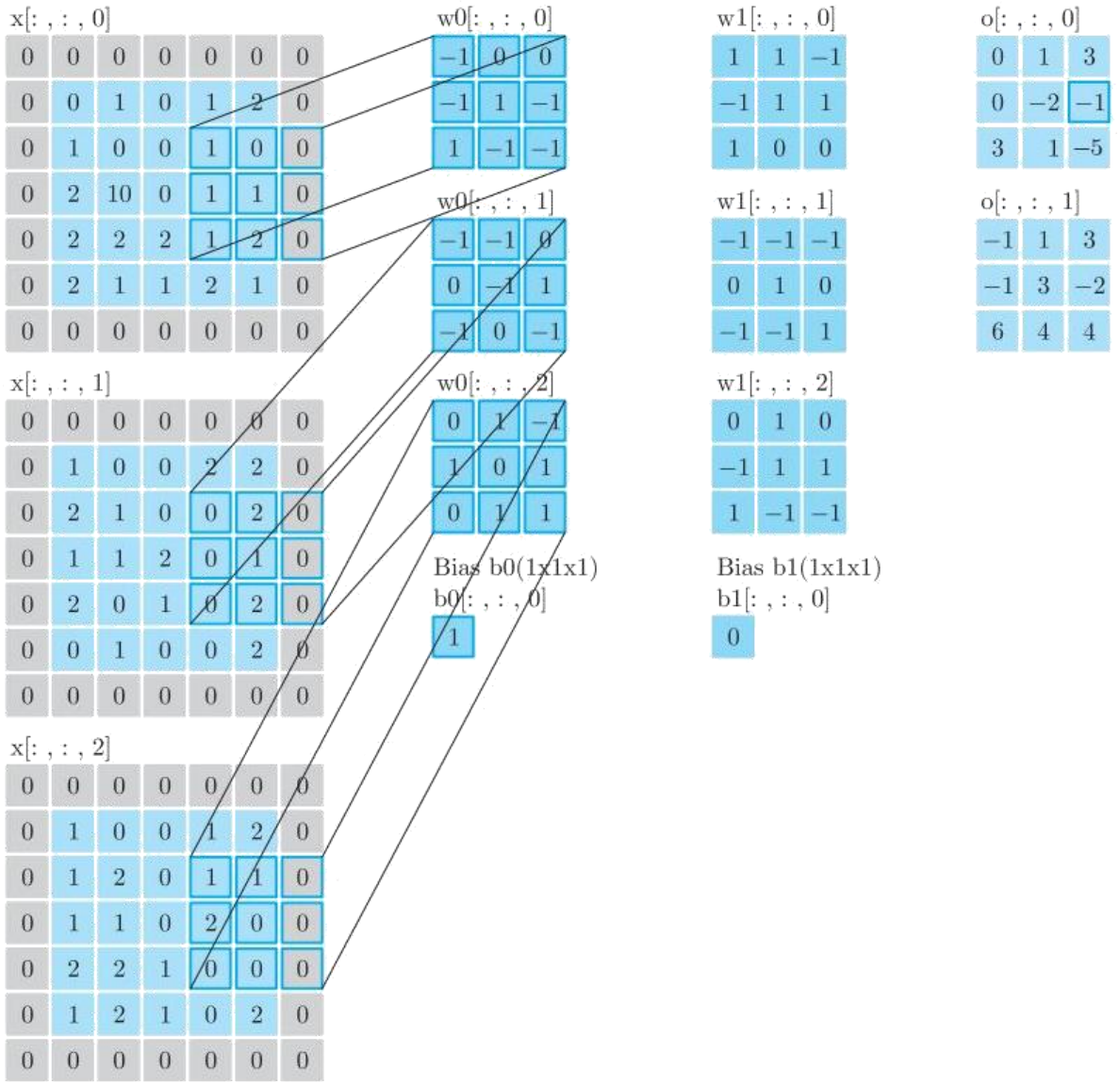
- 最左侧为输入, 中间部分表示卷积核, 最右侧为输出. 卷积核移动到输入左上角开始计算, 以此向右、向下滑动. 例如, 第一个卷积核移动到第一个输入的左上角时, 此时计算过程为:

$$0\times(-1)+0\times0+0\times0+0\times(-1)+0\times1+1\times(-1)+0\times1+1\times(-1)+0\times(-1)=-2.$$

10.2.1 卷积运算

- 类似地, 可以继续后面的运算.

- 通过计算的过程可以发现, “卷” 并没有发挥作用, 卷积神经网络仅仅应用了 “积”. 卷积神经网络具备两大特点: **局部连接和权值共享**. 局部连接指的是卷积层的节点仅仅和其前一层的部分节点相连接; 权值共享指的是同一张图使用相同的卷积核. 局部连接和权值共享减少了参数数量, 加快了神经网络的学习速率, 同时也在一定程度上减少了过拟合的可能.



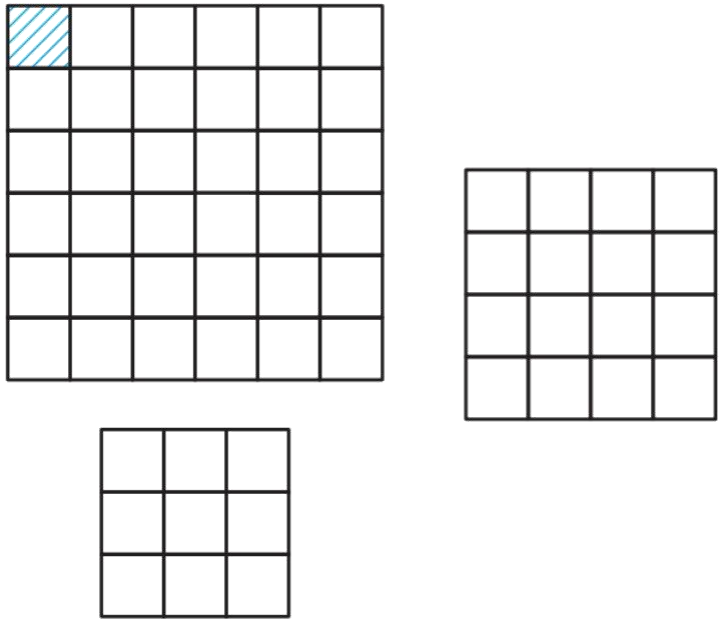
10.2.2 基本参数

- 了解了卷积神经网络的基本过程后, 接下来学习卷积神经网络中的几个基本参数

1. 填充 (padding)

- 在进行卷积操作的时候,6×6的图像经过3×3的卷积过滤器得到4×4的卷积结果, 如图10.4 所示. 用更加一般的形式表达: 如果我们有一个$n \times n$的图像, 用$f \times f$的过滤器做卷积, 那么输出的维度就是$(n-f+1) \times (n-f+1)$. 在这个例子里是 $6-3+1=4$, 因此得到了一个 4×4 的输出.

- 但是这样做存在明显缺点: 每次卷积操作后图像尺寸都会缩小, 同时角落边缘的像素点, 在卷积计算的时候只被一个输出所触碰或者使用, 但是中间的像素点会被多次卷积计算, 所以那些在角落的像素点参与卷积计算较少,意味着丢掉一些图像边缘位置的信息.



10.2.2 基本参数

- 卷积操作中的填充, 可以自行指定 p 的值. 一般有如下两种填充方式:
  - (1) 不填充: 不填充, 即 $p = 0$;
  - (2) 填充至与原矩阵相同大小: 填充后输出矩阵的大小与原矩阵保持一致, 即 $p = \frac{f-1}{2}$, 其中卷积核的大小为 $f \times f$. 通常将 f 设置为奇数, 这样方便我们对称填充, 填充数值可以为零.

2. 步幅 (stride)

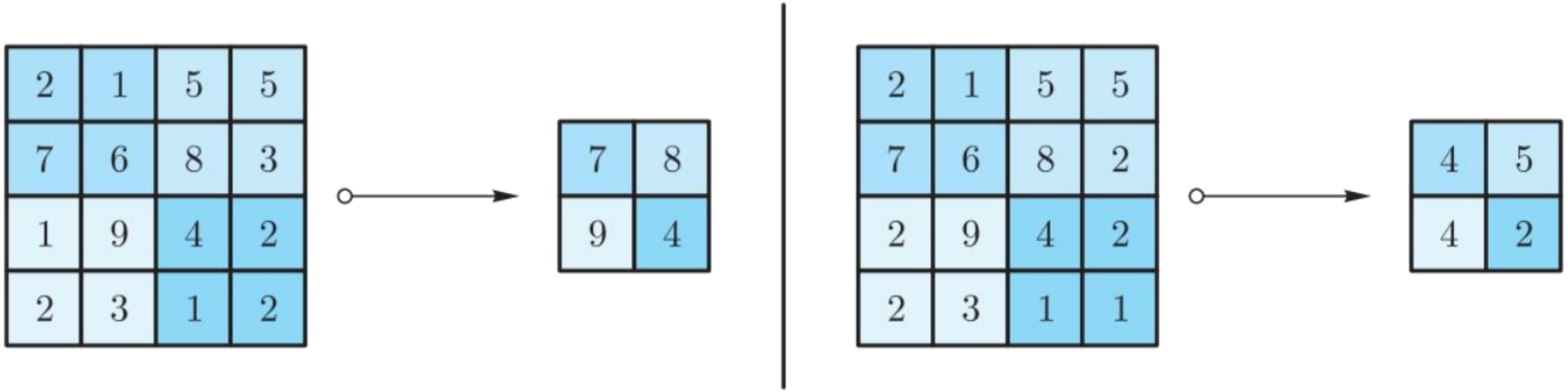
- 滑动卷积核时, 我们会先从输入的左上角开始, 每次往左滑动一列或者往下滑动一行逐一计算输出, 将每次滑动的行数和列数称为步幅.

3. 池化

- 池化是实现下采样的一种运算, 能在提取图像关键特征的基础上减小图片尺寸, 以减少训练中的参数数量, 达到减少计算量、增大感受野并防止过拟合的目的.

10.2.2 基本参数

- 如图 10.5 所示, 池化主要有最大值池化 (max-pooling) 和平均值池化(average-pooling) 两种. 最大值池化从所选区域的矩阵元素中取最大值, 而平均值池化是将所选区域的矩阵元素求和后取平均值.



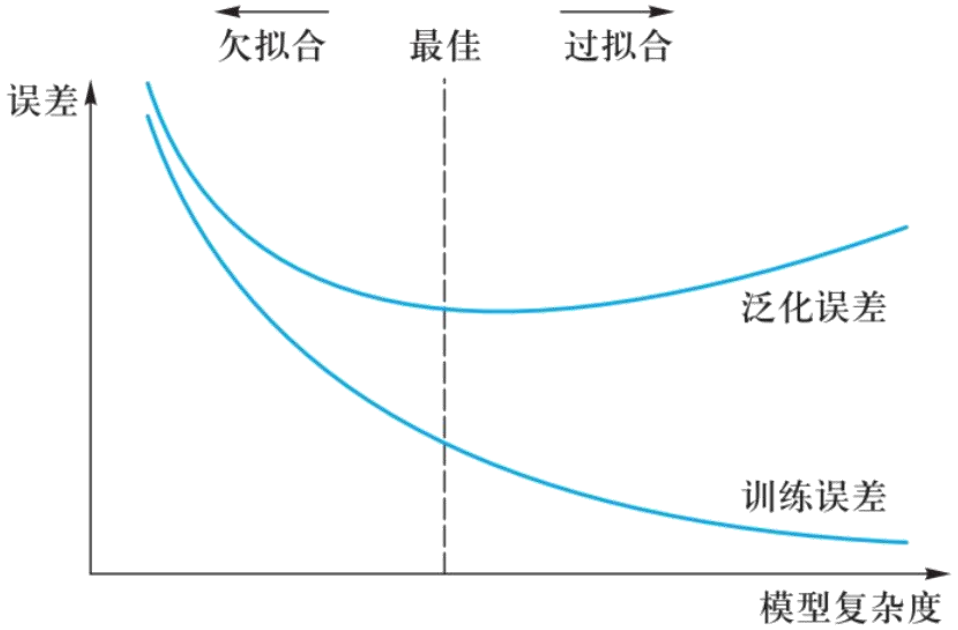
4. 欠拟合和过拟合

- (1) 欠拟合
  - 如图 10.6, 欠拟合是指模型拟合程度不高, 数据距离拟合曲线较远, 或指模型没有很好地捕捉到数据特征, 不能够很好地拟合数据. 解决办法是提高模型的复杂度, 如增加神经网络的层数、宽度, 减少正则化的参数等.

10.2.2 基本参数

- (2) 过拟合

  - 如图 10.6, 一个假设在训练数据上能够获得比其他假设更好的拟合, 但是在训练数据外的数据集上却不能很好地拟合数据, 表现为泛化能力差, 称为过拟合. 导致过拟合的原因有很多, 主要包括以下几点: 模型过于复杂、参数过多; 数据太少; 训练集和测试集的数据分布不同; 样本里的噪音数据干扰过大, 大到模型过分记住了噪音特征, 反而忽略了真实的输入输出间的关系. 解决办法包括 L1、L2 正则化, 扩增数据, Dropout; 提前终止 (earlystopping) 等.



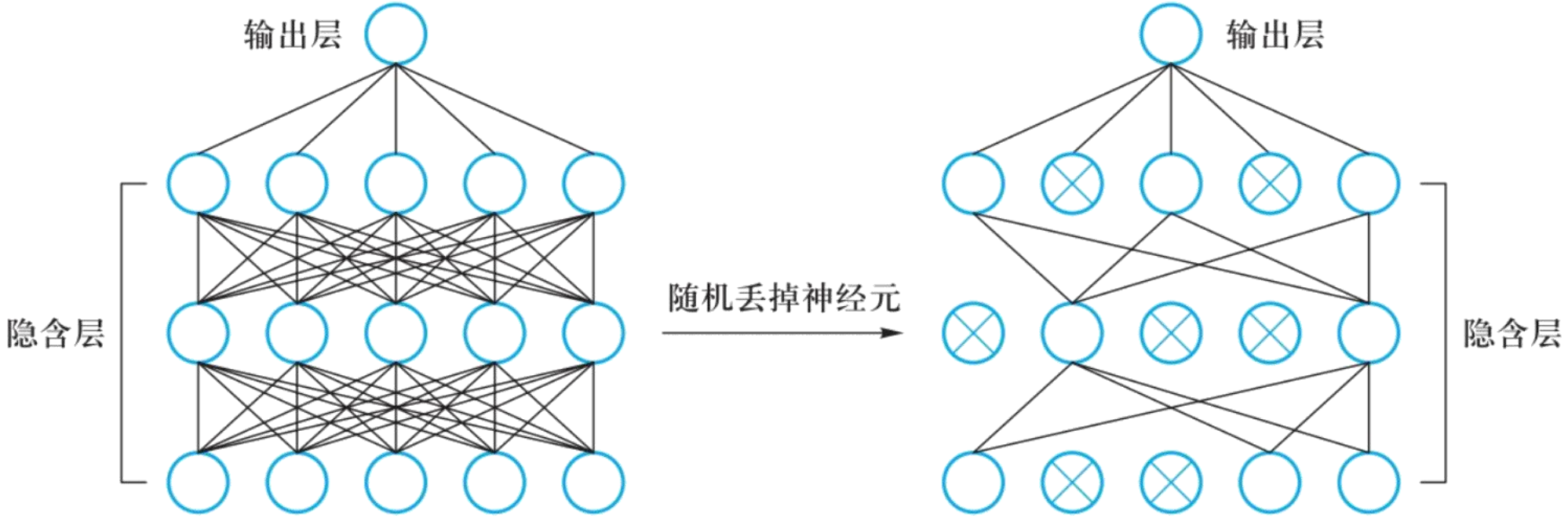
10.2.2 基本参数

- L_1、L_2 正则化
  - 在损失函数上添加正则化项, 其中 L_1 正则化为参数 ω 的绝对值 (Lasso回归)、L_2 正则化为参数 ω 的平方值 (岭回归). 通过对 ω 值的修正, 使其偏离不会太大, 从而减少过拟合的产生.
- 扩增数据
  - 更多的数据集, 能够让搭建的网络在更多的数据中不断修正调整, 进而训练出更好的模型. 然而数据量都是有限的, 所以可以从已有数据出发, 对其进行调整以得到更多的数据集. 如图 10.7 和 10.8, 可以针对已有图像应用随机图像转换, 如旋转、对称和放大等, 来增加图像数量.

10.2.2 基本参数

Dropout

- 不同于 L1、L2 正则化通过修改损失函数防止过拟合, Dropout 修改的对象是神经网络. 如图 10.9 所示, 其思想主要是随机使得神经网络隐含层中的一些神经元失活, 在简化网络结构的同时防止了过拟合, 而且随机失活迫使每一个神经元学习到有效的特征, 增强了搭建网络的泛化能力.



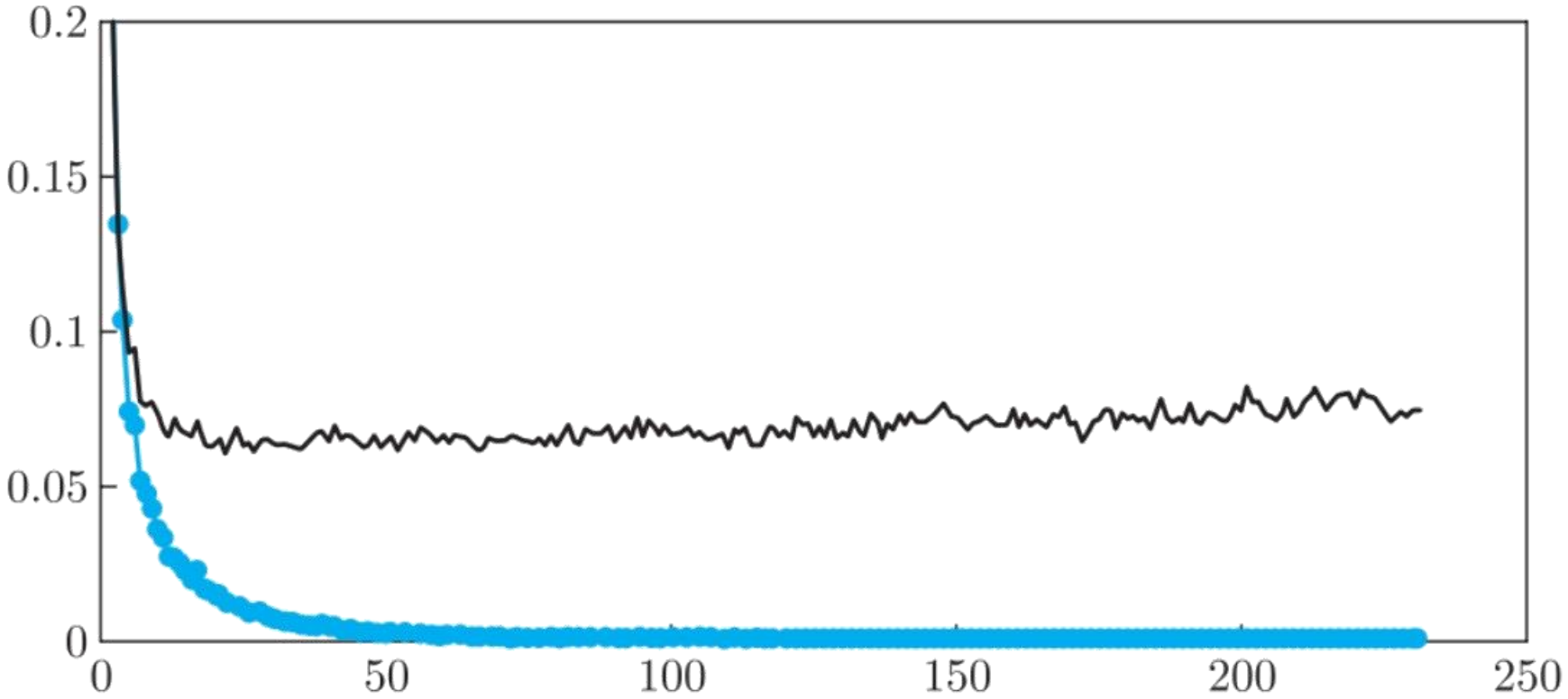
10.2.2 基本参数

- 在神经网络中使用 Dropout 时, 一般以概率 0.5 选择神经元是否有效, 即每层仅留下半数的神经元. 因此使用 Dropout 相当于训练了多个只有半数隐含层神经元的神经网络, 每一个这样的网络, 都可以给出一个分类结果, 这些结果有对有错. 随着训练的进行, 大部分网络可以给出正确的分类结果, 此时少数的错误分类结果不会对最终结果造成大的影响.

提前终止

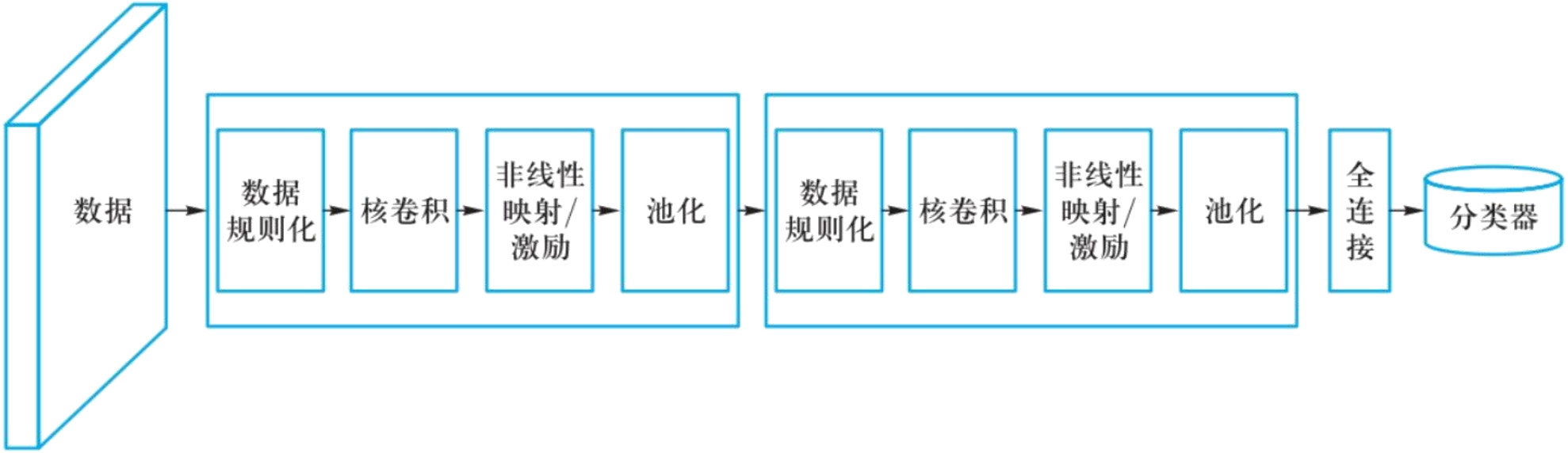
- 当训练有足够的表示能力甚至会过拟合的大模型时, 经常观察到训练误差会随着时间的推移逐渐降低但验证集的误差会再次上升. 这意味着如果返回使验证集误差最低的参数设置, 就可以获得更好的模型 (有希望获得更好的测试误差). 在每次验证集误差有所改善后, 我们存储模型参数的副本. 当训练算法终止时, 返回这些参数而不是最新的参数. 当验证集上的误差在事先指定的循环次数内没有进一步改善时, 算法就会终止. 需要注意的是, 通过提前终止自动选择超参数的显著代价是训练期间要定期评估验证集.
- 图 10.10 中横轴为迭代次数, 纵轴为损失. 黑线为训练集上的损失, 随着迭代次数的不断增加明显减少, 蓝线为验证集上的误差. 上述策略称为提前终止.

10.2.2 基本参数



10.2.3 卷积神经网络的一般结构

- 在卷积神经网络中, 输入图像通过多个卷积层和池化层进行特征提取, 逐步由低层特征变为高层特征; 高层特征再经过全连接层和输出层进行特征分类, 产生一维向量, 表示当前输入图像的类别. 因此, 根据每层的功能, 卷积神经网络可以划分为两个部分: 由输入层、卷积层和池化层构成特征提取器, 以及由全连接层和输出层构成分类器.
- 在实际操作中, 卷积神经网络的架构可以如图 10.11 所示.



10.2.3 卷积神经网络的一般结构

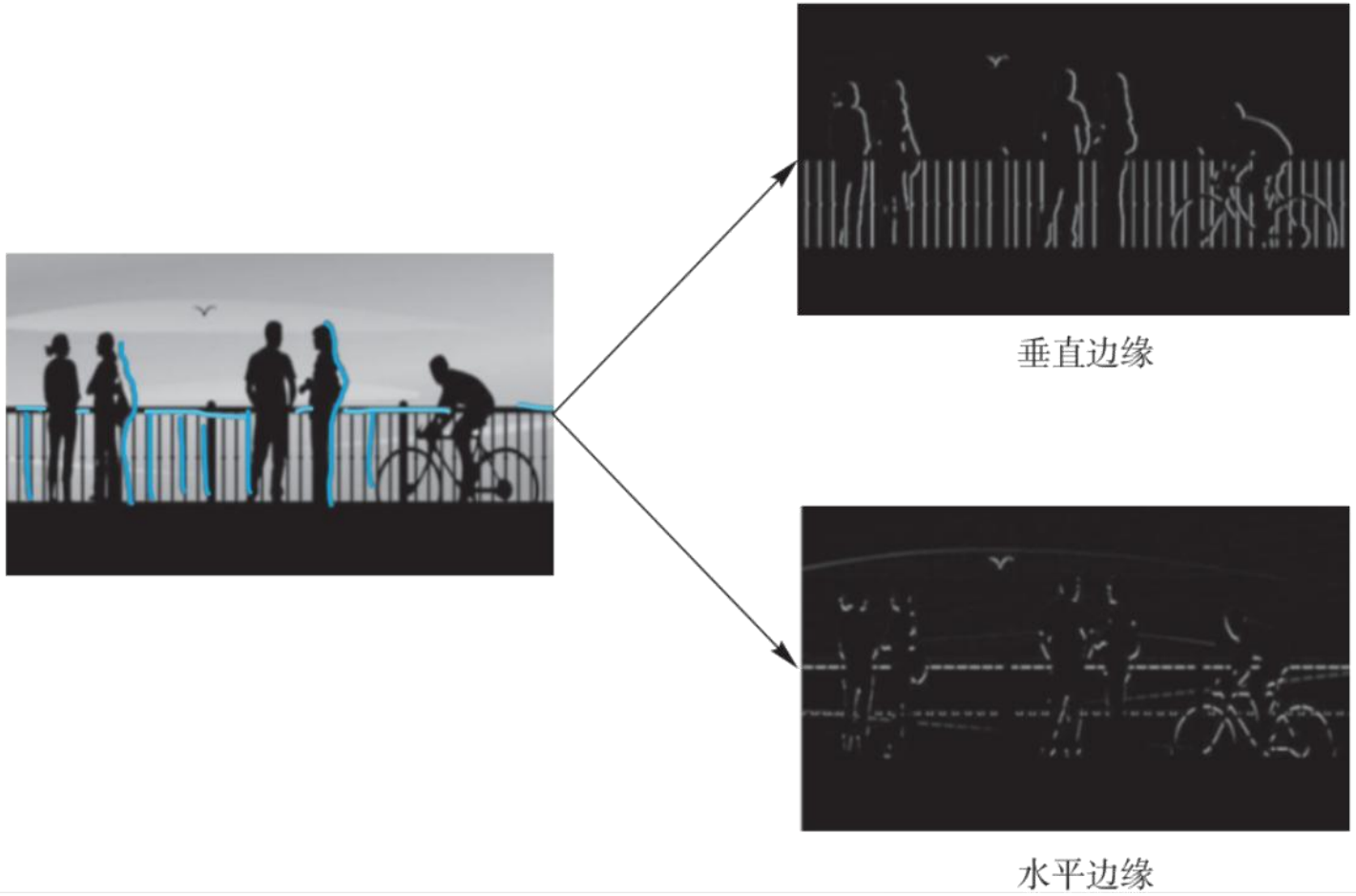
1. 数据规则化: Batch Normalization

- 随着网络的深度增加, 每层特征值分布会逐渐地向激活函数的输出区间的上下两端 (激活函数饱和区间) 靠近, 这样下去会导致梯度消失. 批量标准化 (batch normalization, BN) 通过将该层特征值分布重新拉回标准正态分布,特征值将落在激活函数对于输入较为敏感的区间, 输入的小变化可导致损失函数较大的变化, 使得梯度变大, 避免梯度消失, 同时也可加快收敛.
- BN 在实际工程中被证明了能够缓解神经网络难以训练的问题. 主要有如下优点:
  - (1) BN 使得神经网络中每层输入数据的分布相对稳定, 加速了模型学习速度;
  - (2) BN 使得模型对网络中的参数不那么敏感, 简化调参过程, 使得网络学习更加稳定;
  - (3) BN 允许网络使用饱和性激活函数 (例如 Sigmoid, tanh 等), 缓解梯度消失问题;
  - (4) BN 具有一定的正则化效果.

10.2.3 卷积神经网络的一般结构

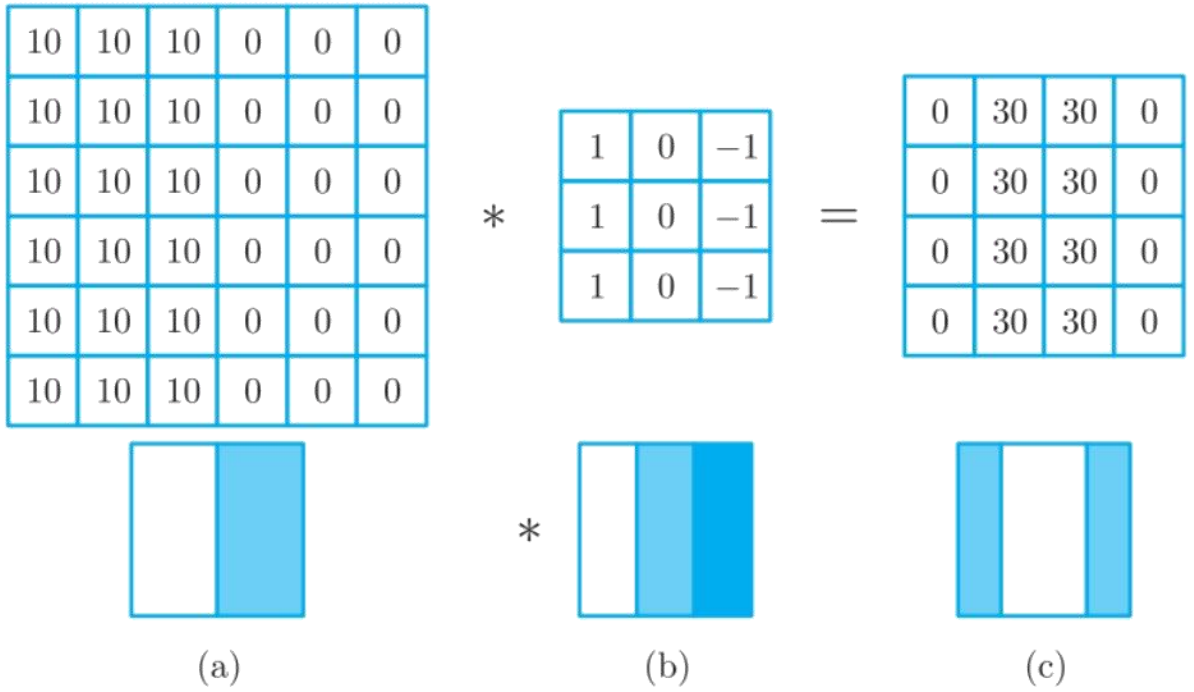
2. 卷积池化层: 特征提取

- 如图 10.12, 给定图片, 如何确定图中包含哪些物体? 可以检测图片中的边缘来达到识别的目的. 比如说, 图片中的栏杆和行人的轮廓线都可以看作是垂线, 同样, 栏杆就是很明显的水平线, 它们也能被检测到. 图片中的大部分物体都能用垂直和水平边缘线来刻画, 那么如何在图像中检测这些边缘呢?下面通过一个简单例子来说明.



10.2.3 卷积神经网络的一般结构

- 一个 6×6 的灰度图, 可以表示为左亮右暗的形式, 如图 10.13(a) 所示. 很明显, 中间存在明显的分界线, 因此我们试图将该垂直边缘识别出来. 选择卷积核为 3×3, 相应的输出为 4×4. 仍旧通过颜色的亮暗来表示卷积核和输出矩阵, 如图 10.13(b) 和图 10.13(c) 所示. 不难看出, 输出矩阵对应的图中间有段发亮的区域, 对应原图中的垂直边缘.



10.2.3 卷积神经网络的一般结构

- 显然这里检测到的边缘宽度远大于原图中的边缘宽度, 这和原图像素大小有关, 如果改用像素值为500×500 的图像或者更大, 检测效果将会有显著提升. 垂直边缘通过这样的方式可以很容易检测出, 水平边缘的检测类似. 推广到一般情况, 卷积核就是通过这样的方式, 提取到输入图像的特征.

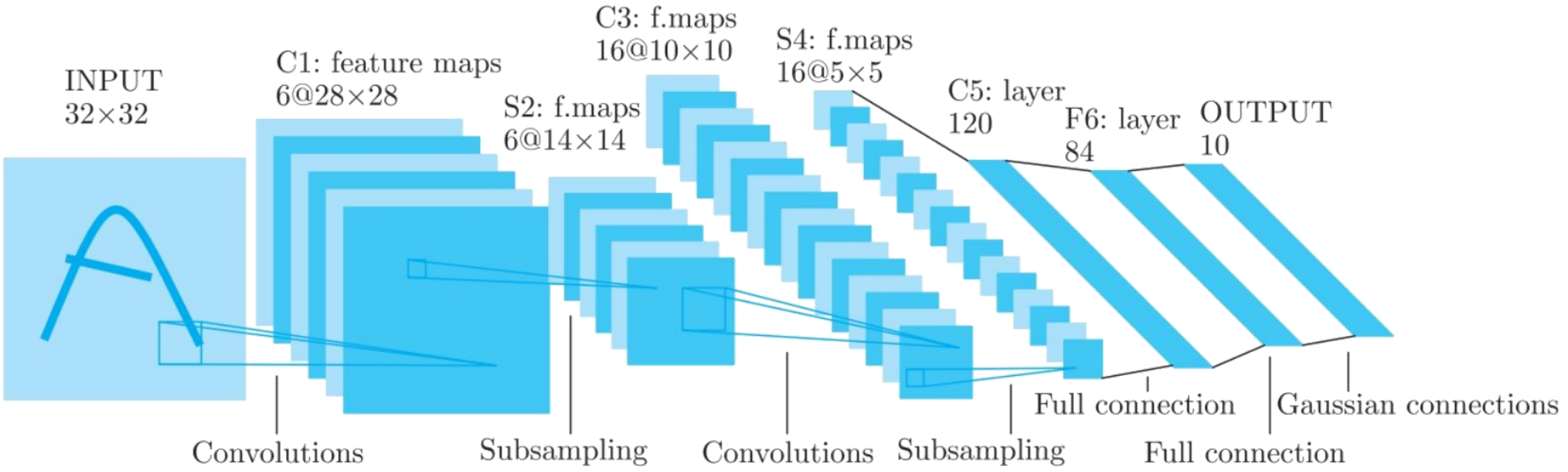
3. 全连接层

- 把分布式特征映射到样本标记空间, 即将特征整合到一起输出为一个值,减少特征位置对分类带来的影响.
- 例如: 假设你是一只小蚂蚁, 你的任务是找小面包. 你的视野比较窄, 因此只能看到很小的一片区域. 当你找到一片小面包之后, 你不确定你找到的是不是全部的小面包, 所以你和其余的蚂蚁一块儿开了个会, 把所有的小面包都拿出来以确定是否已经找到所有的小面包. 某种程度上可以认为, 全连接层就是这个蚂蚁大会. 需要注意的是, 在数据输入到全连接层前还需要进行拉直操作, 将所有数据拉成一维向量. 或者可以认为是某种卷积操作. 由于全连接层参数过多, 正逐渐被全局平均池化 (global average pooling, GAP) 的方法代替.

10.2.4 常见的卷积神经网络

1. LeNet

- LeNet 由 LeCun[117] 在 1998 年提出, 用于解决手写数字识别的视觉任务.图 10.14 中, Convolutions 表示卷积, Subsampling 表示下采样, 即池化,Full connection 表示全连接, Gaussian connection 表示高斯连接. 具体来看, LeNet 共有 5 层, 包括 2 层卷积层和 3 层全连接层.



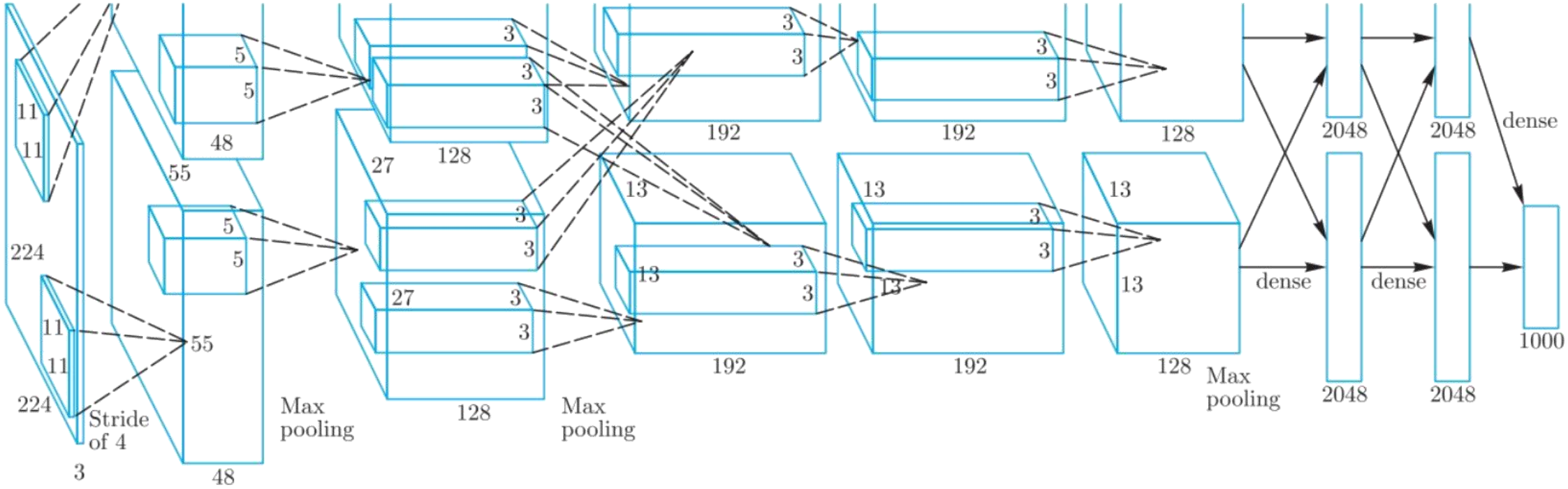
10.2.3 卷积神经网络的一般结构

- LeNet 中每个卷积层均使用尺寸为 5×5 的卷积核, 步长为 1, 并使用 Sig_x0002_moid 激活函数. 两个池化层中池化核均为 2×2, 且步长为 2, 这里选择的是平均池化. 由于池化核尺寸与步长相同, 因此池化后在输出上每次滑动所覆盖的区域互不重叠, 池化不改变通道数量. 当卷积层块的输出传入全连接层块时, 全连接层块会将小批量中每个样本变平. 全连接层块含 3 个全连接层. 它们的输出个数分别是 120、 84 和 10, 其中 10 为输出类别的个数.

2. AlexNet

- AlexNet 是 2012 年 ImageNet 竞赛的冠军, 自此越来越多越来越复杂的神经网络被提出并得到广泛应用.
- 限于单个图形处理器 (GPU) 的计算能力, AlexNet 在两个图形处理器上实现卷积神经网络的搭建和计算. 图 10.15 中 Stride 表示步幅, Max Pooling表示池化选择最大池化, Dense 为全连接. 搭建和计算的大致如下:

10.2.3 卷积神经网络的一般结构



- AlexNet 共有八层, 包括 5 层卷积层和 3 层全连接层. 输入尺寸为 227×227×3 的彩色图片, 使用 11×11 的卷积核, 步幅选择为 4, 输出图像的尺寸为55×55×96, 其中

$$\text{输出图像长（宽）度}=\frac{\text{输入图像长（宽）度}+2\times\text{Padding}-\text{卷积核长（宽）度}}{\text{步幅}}+1,$$

10.2.3 卷积神经网络的一般结构

- 因此 $\frac{227+2\times0-11}{4}+1=55$. Pooling 选择步长为 2、3×3 的池化核, 输出图像的尺寸变为 27×27×96.

- 再次使用 5×5 的卷积核, 步幅选择为 1, 并且使用 Padding=2, 输出得到

$$\frac{27+2\times2-5}{1}+1=27,$$

- 通道数为 256. Pooling 选择步幅为 2、3×3 的池化核, 输出图像的尺寸为 13×13×256.

- 第三、四、五层均选择 3× 3 的卷积核, 且步幅均为 1. Pooling 仍旧选择步幅为 2、3×3 的池化核, 输出图像的尺寸变为 6×6×256.

- AlexNet 具有如下四个特点:

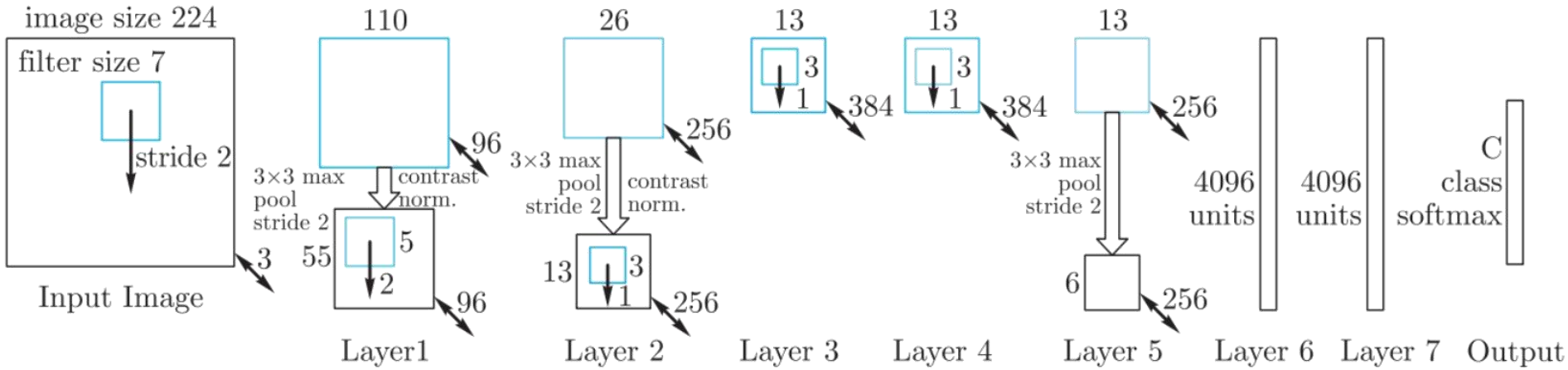
  - (1) 使用 ReLU 函数作为激活函数, 很好地增强了网络的非线性表达能力.
  - (2) 引入局部响应归一化 (local response normalization, LRN). 通过 ReLU函数得到的值域没有固定区间, 因此要对得到的结果进行归一化. 具体来说放大对分类贡献较大的神经元, 抑制对分类贡献较小的神经元.

10.2.3 卷积神经网络的一般结构

- (3) 使用重叠的最大池化. 即池化层中卷积核的尺寸大于步幅, 这样池化层的输出出现重叠和覆盖, 提升了特征的丰富性.
- (4) 通过 Dropout 和数据扩增等方式来防止神经网络出现过拟合现象.

3. ZFNet

图 10.16 中 image size 为输入图片的尺寸, 224×224, filter size 为卷积核的大小, stride 为步幅, max pool 表示池化选择最大池化, softmax 为最终用于分类的函数, Layer 标明了 ZFNet 的每一层. ZFNet 是 AlexNet 的改进版本, 共有八层, 仍旧是 5 层卷积层和 3 层全连接层. 改进主要有以下两点:



10.2.3 卷积神经网络的一般结构

- (1) 使用一块图形处理器搭建稠密连接结构;
- (2) 第一个卷积层中卷积核的尺寸从 11× 11 变为 7× 7, 同时步幅从 4 减小为 2; 为了让后续输出图像的尺寸与 AlexNet 中的输出图像尺寸保持一致,第 2 个卷积层的步幅从 1 变为 2.

- 整体来看, ZFNet 的改进似乎并不明显. 那么为什么要做这样的修改?修改的动机又是什么呢? 这是基于卷积神经网络深层特征的可视化提出的.可视化操作, 针对的是已经训练好的网络, 或者训练过程中的网络快照. 可视化操作不会改变网络的权重, 只是用于分析和理解在给定输入图像时网络观察到了什么样的特征, 以及训练过程中特征发生了什么变化.

- 可视化主要有如下三个操作:
  - (1) 修正 (rectification): 因为使用的是 ReLU 激活函数, 前向传播时只将正值原封不动输出, 负值置 0, “反激活” 过程与激活过程没什么分别, 直接将来自上层的输出再次输入到 ReLU 激活函数中即可.
  - (2) 上池化 (unpooling): 在前向传播时, 记录相应最大池化层每个最大值来自的位置, 在上池化时, 根据来自上层的 map 直接填在相应位置上, 其余位置为 0.

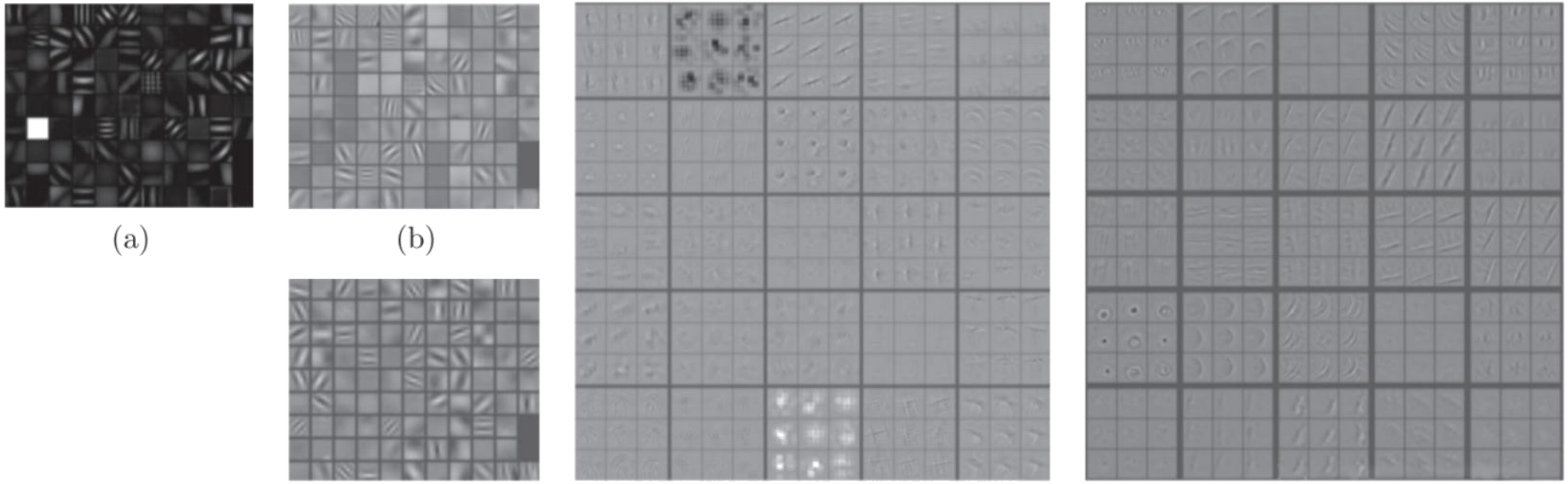
10.2.3 卷积神经网络的一般结构

- (3) 转置卷积 (transposed convolution): 卷积操作输出图像的尺寸一般小于等于输入图像的尺寸, 转置卷积可以将尺寸恢复到与输入相同, 相当于上采样过程, 该操作的做法是, 与卷积共享同样的卷积核, 但需要将其左右上下翻转 (即中心对称), 然后作用在来自上层的输出图像进行卷积, 结果继续向下传递.

- 不断进行上述操作, 可以将特征映射回输入所在的像素空间, 进而呈现出人眼可以理解的特征. 给定不同的输入图像, 看看每一层关注到最显著的特征是什么.

- 图 10.17(a) 为没有经过裁剪的图片经过第一个卷积层后的特征可视化图, 注意到有一个特征全白, (b) 为 AlexNet 中第一个卷积层特征可视化图,(c) 为 ZFNet 中第一个卷积层可视化图, 可以看到相比前面有更多的独特的特征以及更少的无意义的特征, 如第 3 列的第 3 到 6 行, (d) 为 AlexNet 中第二个卷积层特征可视化图, (e) 为 ZFNet 中的第二个卷积层特征可视化图, 可以看到 (e) 中的特征更加干净, 清晰, 保留了更多的第一层和第二层中的信息.

10.2.3 卷积神经网络的一般结构



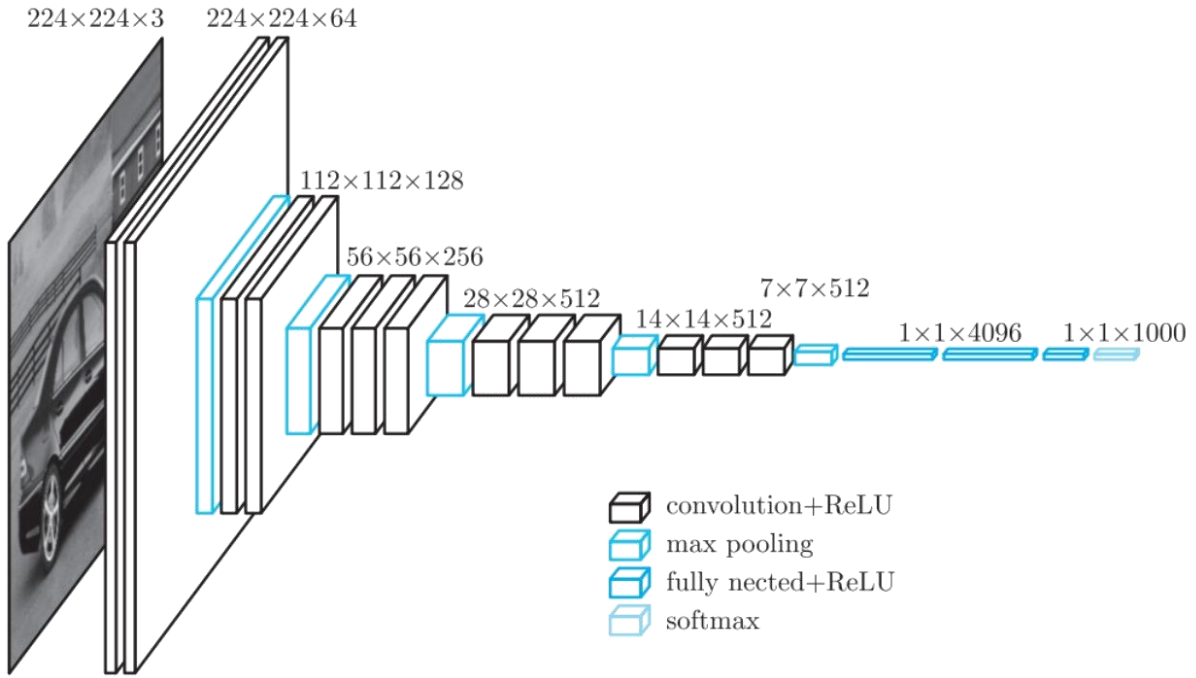
- 通过对 AlexNet 的特征进行可视化, Zeiler 等人发现 AlexNet 第一层中有大量的高频和低频信息的混合, 却几乎没有覆盖到中间的频率信息; 且第二层中由于第一层卷积用的步幅为 4 太大了, 导致了有非常多的混叠情况.为了解决这个问题, Zeiler 等人提高采样频率, 将步幅从 4 调整为 2, 与之相应地将卷积核尺寸也缩小 (可以认为步幅变小了, 卷积核没有必要看那么大范围了), 修改后第一层呈现了更多更具区分力的特征, 第二层的特征也更加清晰[121].

10.2.4 常见的卷积神经网络

4. VGGNet

- VGGNet(visual geometry group net) 是由牛津大学计算机视觉组合和Google DeepMind 公司研究员[122] 一起研发的深度卷积神经网络. 它探索了卷积神经网络的深度和其性能之间的关系, 通过反复的堆叠 3×3 的卷积核和 2×2 的最大池化层, 成功地构建了 16 到 19 层的卷积神经网络. 因此,VGGNet 是指一系列网络, 下面以 VGG-16 为例进行介绍.

- 图 10.18 中 convolution+Relu 表示卷积后选择 Relu 激活函数, max pool_x0002_ing 表示池化层选择最大池化, full nected+Relu 表示全连接层后选择 Relu 激活函数, softmax 激活函数用于最后的分类.



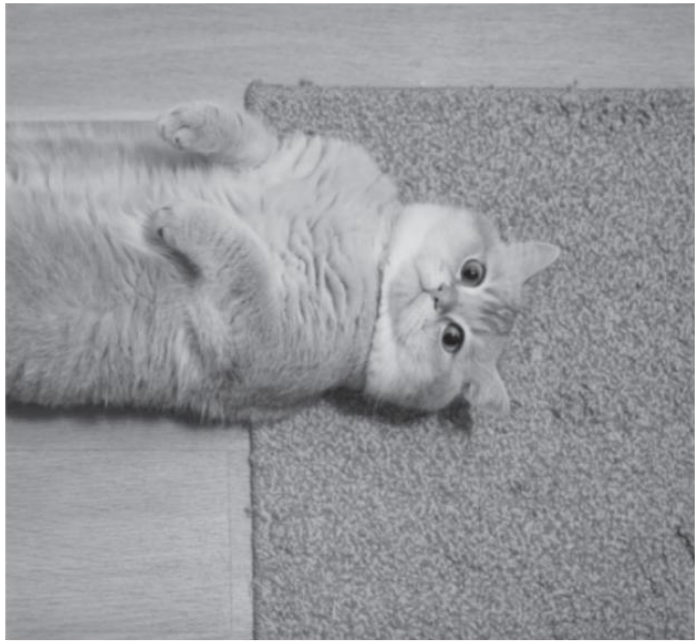
10.2.4 常见的卷积神经网络

- VGGNet 是一种专注于构建卷积层的简单网络, 相比 AlexNet 和 ZFNet参数量大大减少, 一个很重要的原因是用到了卷积核的堆叠. 例如通过 2 个3×3 的卷积核代替 1 个 5×5 的卷积核, 3 个 3×3 的卷积核代替 1 个 7×7 的卷积核. 通过这样的方式不仅节省了大量参数, 还使得网络获得了更大的感受野, 同时增强了卷积神经网络的非线性能力.

5. GoogLeNet

- 在通过卷积神经网络识别处理图像时, 经常遇到图像突出部分的大小差别很大的情况. 如图 10.19, 猫的图像可以是以下任意情况. 每张图像中猫所占区域是不同的.

(a)



(b)

10.2.4 常见的卷积神经网络

- 由于信息位置的巨大差异, 为卷积操作选择合适的卷积核大小就比较困难. 信息分布更全局性的图像偏好较大的卷积核, 信息分布比较局部的图像偏好较小的卷积核. 同时, 非常深的网络更容易过拟合. 将梯度更新传输到整个网络是很困难的. 再加上简单地堆叠较大的卷积层非常消耗计算资源, 因此考虑在同一层级上运行具备多个尺寸的卷积核呢? 网络本质上会变得稍微宽一些, 而不是更深. 2015 年, 串并联网络架构的 GoogLeNet 应运而生[123].GoogLeNet 最基本的网络块是 Inception, 它是一个并联网络块, 经过不断地迭代优化, 发展出了 Inception-v1、Inception-v2、Inception-v3、Inception-v4、Inception-ResNet 共 5 个版本. Inception 家族的迭代逻辑是通过结构优化来提升模型泛化能力、降低模型参数.

- Inception 就是把多个卷积或池化操作, 放在一起组装成一个网络模块,设计神经网络时以模块为单位去组装整个网络结构. 在未使用这种方式的网络里, 我们一层往往只使用一种操作, 比如卷积或者池化, 而且卷积操作的卷积核尺寸也是固定大小的. 但是, 在实际情况下, 在不同尺度的图片里, 需要不同大小的卷积核, 这样才能使性能最好, 或者, 对于同一张图片, 不同尺寸的卷积核的表现效果是不一样的, 因为它们的感受野不同.

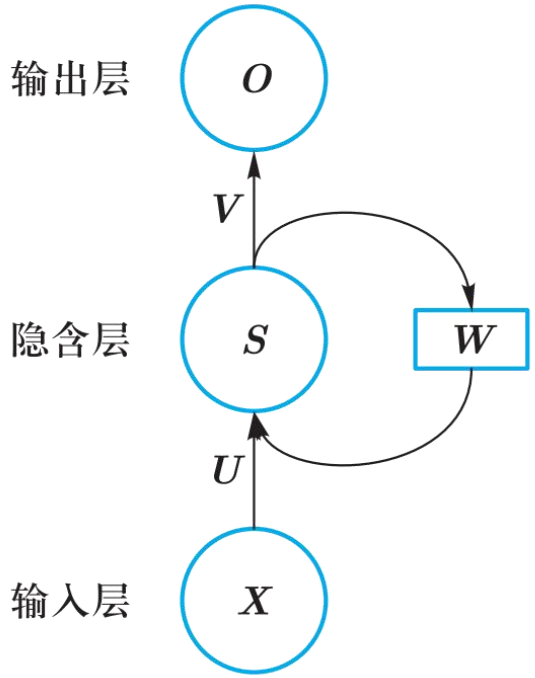
10.2.4 常见的卷积神经网络

- 所以, 我们希望让网络自己去选择, Inception 便能够满足这样的需求, 一个 Inception 模块中并列提供多种卷积核的操作, 网络在训练的过程中通过调节参数自己去选择使用, 同时, 由于网络中都需要池化操作, 所以此处也把池化层并列加入网络中.

10.3　简单循环神经网络

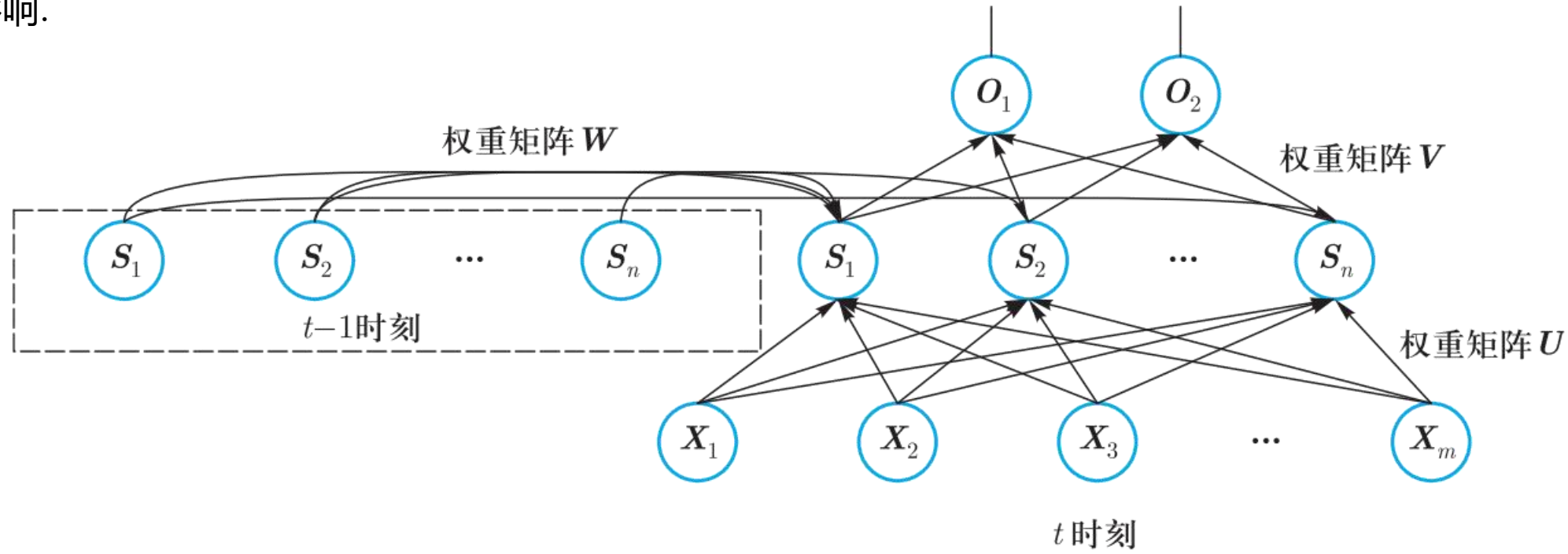
10.3 简单循环神经网络

- 循环神经网络 (recurrent neural network, RNN) 是一类以序列 (sequence)数据为输入, 在序列的演进方向进行递归 (recursion) 且所有节点 (循环单元)按链式连接的递归神经网络 (recursive neural network).
- 在日常生活和学习中, 有许多数据的样本之间是有关联的, 比如气象数据、经济数据, 而之前讲述的神经网络并不能很好地学习这一关联, 而循环神经网络就能很好解决这一问题.
- 要了解 RNN, 首先从一个简单的循环神经网络开始 (如图 10.20). 其中,X 是一个向量, 它表示输入层的值 (图中并未画出表示神经元节点的圆圈);S 是一个向量, 它表示隐含层的值; U 是输入层到隐含层的权重矩阵; O 也是一个向量, 它表示输出层的值; V 是隐含层到输出层的权重矩阵; 而权重矩阵 W 就是隐含层上一时间的值对这一时间的输入的权重, 它表示上一时间隐含层的值对这一时间的影响. 这一简单的循环神经网络与全连接神经网络的不同就在于多出了权重矩阵 W.



10.3 简单循环神经网络

- 图 10.21 展示了这个神经网络展开的样子, 可以发现权重矩阵 $\boldsymbol{W}$ 实际上就相当于 t − 1 时刻对 t 时刻的一个影响.



- 这个网络在 t 时刻接收到输入值为 $\mathbf{S}_t$, 隐含层的值是 $\mathbf{S}_t$, 输出值是 $\boldsymbol{O}_t$.关键一点是, $\mathbf{S}_t$ 的值不仅仅取决于 $\mathbf{S}_t$, 还取决于 $\mathbf{S}_{t-1}$.

10.3 简单循环神经网络

- 可以用下面的公式来表示循环神经网络的计算方法:

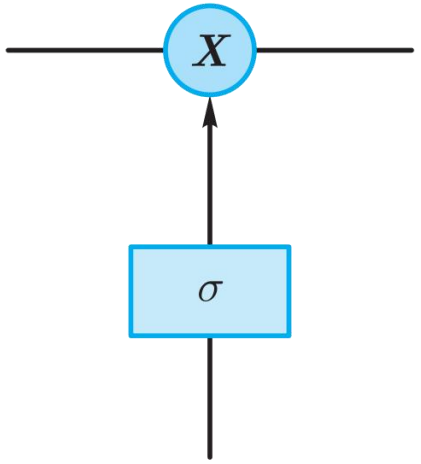
$$\boldsymbol{O}_t = g\left(\boldsymbol{V} \cdot \boldsymbol{S}_t\right),$$
$$\boldsymbol{S}_t = f\left(\boldsymbol{U} \cdot \boldsymbol{X}_t + \boldsymbol{W} \cdot \boldsymbol{S}_{t-1}\right).$$

- 循环神经网络的参数同样可以利用前文提到的反向传播算法进行求解.但同时, 需要注意的是: 反向传播算法按照时间逆序将信息一步步向前传递,如果输入序列较长, 会存在梯度爆炸与梯度消失问题, 也称为长程依赖问题,长程依赖问题的具体表现为: 很久以前的输入, 对当前时刻的网络影响较小,在反向传播时, 梯度也很难影响到很久以前的输入.

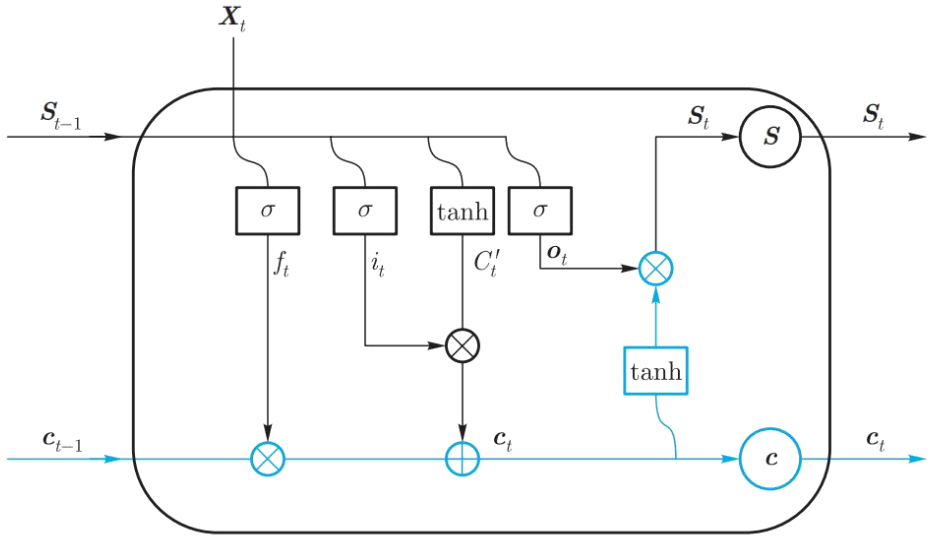
10.4　长短时记忆神经网络

10.4 长短时记忆神经网络

- 为了解决长程依赖问题, 人们将基于图 10.22 中所展示的神经网络进行了改进, 引入了门控机制 (gating mechanism). 门 (gate) 是一种让信息有选择地通过的方式, 它由一个 Sigmoid 函数和一个点乘运算组成. 利用 Sigmoid 函数返回 0 或 1 的数字的特性, 令这个数字描述每个元素有多少信息可以通过, 0 表示不通过任何信息, 1 表示全部通过.



- 门控机制控制了对于当前时刻有哪些信息将被保留, 有哪些信息将被遗忘. 也将控制有哪些新信息将被加入记忆信息, 有哪些信息又将从记忆信息中被提取出作为当前时刻的输出. 下面介绍一个加入了门控机制后的循环神经网络——长短时记忆神经网络 (long short-term memory neural network,LSTM). LSTM 的结构有很多种形式, 但是都大同小异. 这里介绍一种较为流行的结构——门控循环单元 (gated recurrent unit, GRU). 结构图 10.23 如下:



10.4 长短时记忆神经网络

- LSTM 模型的关键是引入了一组记忆单元 c_t, 以及三个控制 “门”: 输入门、输出门、控制门. 它们允许网络可以学习何时遗忘历史信息, 何时用新信息更新记忆单元. 在时刻 t 时, 记忆单元 c_t 记录了到当前时刻为止的所有历史信息, 并受三个 “门” 控制, 三个门中元素的数值取值范围是 (0, 1):

$$\text{输入门：}\ \boldsymbol{i}_t = \sigma\left(\boldsymbol{U}_i \boldsymbol{X}_t + \boldsymbol{W}_i \boldsymbol{S}_{t-1}\right),$$

$$\text{遗忘门：}\ \boldsymbol{f}_t = \sigma\left(\boldsymbol{U}_f \boldsymbol{X}_t + \boldsymbol{W}_f \boldsymbol{S}_{t-1}\right),$$

$$\text{输出门：}\ \boldsymbol{o}_t = \sigma\left(\boldsymbol{U}_o \boldsymbol{X}_t + \boldsymbol{W}_o \boldsymbol{S}_{t-1}\right).$$

- 下面具体介绍 LSTM 模型. 首先了解 LSTM 模型中的核心机制: 记忆细胞状态.
- 细胞状态就像一根传送带, 直接在整个链上运行, 运行过程中只有少量的线性交互, 以保证信息在上面流传保持不变. 这些信息将会随着网络一直传递下去, 保证了在学习当前时刻输入信息的同时, 不会遗忘掉之前学习到的信息.

10.4 长短时记忆神经网络

- 在 LSTM 模型中引入的门控机制, 目的就是为了控制记忆细胞状态, 向其中增添与删除信息. 首先是遗忘门 $\boldsymbol{f}_t$ 的更新机制:

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{U}_f \boldsymbol{X}_t + \boldsymbol{W}_f \boldsymbol{S}_{t-1}\right),$$

  - 其中, $\boldsymbol{W}_f$是关于隐藏状态 $\boldsymbol{S}_t$ 的参数矩阵, $\boldsymbol{U}_f$是关于当前输入 $\boldsymbol{X}_t$ 的参数矩阵. 由于 Sigmoid 函数的存在, $\boldsymbol{f}_t$ 的取值范围在 (0, 1) 内.

- 遗忘门 $\boldsymbol{f}_t$ 的具体功能是根据 $\boldsymbol{X}_t$ (当前输入) 和 $\boldsymbol{S}_{t-1}$ (前一个隐藏状态),为记忆状态 c_{t-1} ($t-1$ 时刻状态) 中的每个元素输出 $0 \sim 1$ 之间的数字, 以决定每一个 c_{t-1} 元素保留多少信息. 1 代表完全保留, 0 代表彻底删除.

- 下一步是确定向记忆细胞中增添多少新信息. 这一步骤分为两部分, 首先需要确定当前输入信息, 再决定输入信息的保留程度. 对于当前输入信息 $\boldsymbol{C}_t'$ 的计算方式如下:

$$\boldsymbol{C}_t' = \tanh\left(\boldsymbol{U}_c \boldsymbol{X}_t + \boldsymbol{W}_c \boldsymbol{S}_{t-1}\right).$$

10.4 长短时记忆神经网络

- 这里同样利用上一时刻隐含层向量 $\boldsymbol{S}_{t-1}$ 与当前输入向量 $\boldsymbol{X}_t$ 分别乘参数矩阵 $\boldsymbol{W}_c$ 与 $\boldsymbol{U}_c$, 进行一个线性变化. 不同的是由于 $\boldsymbol{C}_t'$ 不是一个门, 不需要其取值范围限定在 (0, 1) 内, 于是选择用 tanh() 函数代替 Sigmoid 函数进行非线性激活.
- 得到 $\boldsymbol{C}_t'$ 后, 使用输入门 $\boldsymbol{i}_t$ 来决定保留 $\boldsymbol{C}_t'$ 中的信息至记忆细胞状态中:

$$i_t = \sigma\left(\boldsymbol{U}_i \boldsymbol{X}_t + \boldsymbol{W}_i \boldsymbol{S}_{t-1}\right).$$

- 与遗忘门相同, $\boldsymbol{W}_i$ 是关于前一个隐藏状态 $\boldsymbol{S}_{t-1}$ 的参数矩阵, $\boldsymbol{U}_i$ 是关于当前输入 $\boldsymbol{x}_t$ 的参数矩阵. 同理, 输入门 $\boldsymbol{i}_t$ 的取值范围在 (0, 1) 内.
- 最后是输出门 $\boldsymbol{o}_t$: 基于当前的 $\boldsymbol{c}_t$, 输出那些作为当前时刻隐含层的信息$\boldsymbol{h}_t$. $\boldsymbol{o}_t$ 的更新方法依旧类似于之前的遗忘门与输出门:

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{U}_o \boldsymbol{h}_{t-1} + \boldsymbol{W}_o \boldsymbol{X}_t + \boldsymbol{b}_o\right),$$

10.4 长短时记忆神经网络

- 其中, U_o 是关于前一个隐藏状态 h_{t-1} 的参数矩阵, W_0 是关于当前输入 o_t的参数矩阵, b_0 是一个偏置向量. 同理, 由于 Sigmoid 函数, 输入门 o_t 的取值范围也在 (0, 1) 内. 在获得 o_t 之后, 利用之前更新好的当前时刻历史记忆信息 c_t, 进行当前时刻隐含层 h_t 信息的更新:

$$h_t = o_t\left(c_t\right)$$

- 在更新 h_t 的时候, 首先用非线性函数 tanh() 对 c_t 进行激活, 然后再与o_t 进行点乘运算, 保证向量维度保持不变. h_t 会作为下一个时刻的输入之一,再参与未来新的一系列更新.
- 有两点需要强调:
  - (1) 尽管三个门, 以及当前输入信息 C_t' 的更新都具有相同的形式, 但是每个更新所用的参数都是独立不同的, 这保证了反向传播时对三个门控单元即当前记忆信息独立地进行更新.
  - (2) 以上公式中的所有参数都是可学习的参数. 它们在最初是自动随机生成的随机数, 但是随着模型的不断训练, 这些参数会随着反向传播算法不断更新, 最终使得损失函数值尽可能小, 以提高模型的表达能力.

10.4 长短时记忆神经网络

- LSTM 结构回顾:
  - (1) LSTM 有单独的细胞状态, 该状态贯穿整个 LSTM 网络.
  - (2) 用遗忘门 C_t' 和输入门 i_t 决定历史记忆信息与新记忆信息的保留或放弃.
  - (3) 当前输入信息 C_t' 来源于 S_{t-1} 和 X_t.
  - (4) 当前记忆信息由 $f_t + i_t \times C_t'$ 计算得出.
  - (5) 输出门控制细胞状态的输出 $S_t = o_t(c_t)$.
- 以上所介绍的循环神经网络大量应用在自然语言处理场合. 自然语言处理 (NLP) 是指利用人类交流所使用的自然语言与机器进行交互通讯的技术.通过人为地对自然语言处理, 使其可读并被计算机理解. 在本章 10.7 节, 引入了一个在自然语言处理情境应用循环神经网络的代码示例, 可供参考与学习.

10.5　自编码器

10.5 自编码器

在机器学习任务中经常需要采用某种方式对数据进行有效编码, 例如对原始数据进行特征提取便是几乎所有机器学习问题均需解决的编码任务. 除此之外, 对数据进行降维处理或稀疏编码也是常见的编码任务. 通常对数据进行编码时需要按照编码要求将原始数据转化为特定形式的编码数据, 并要求编码数据尽可能多地保留原始数据信息, 对这样的编码数据进行分析处理不仅会更加方便, 而且可保证分析处理的结果较为准确. 对数据的编码过程事实上是一个数据映射过程. 若将某类原始数据 $\boldsymbol{X}$ 编码为特定形式的数据 Y, 则需找到某个合适的映射 f 使得 $Y=f(\boldsymbol{X})$. 通常称映射 f 为编码器, 并称将原始数据 $\boldsymbol{X}$ 映射为编码数据 Y 的过程为编码过程. 若编码数据 Y 保留了大部分原始数据 $\boldsymbol{X}$ 中的信息, 则从理论上说一定存在某种方式将数据 Y 映射为与 $\boldsymbol{X}$ 相近的数据 $\boldsymbol{X}'$. 假设存在映射 g 满足

$$\boldsymbol{X}'=g\left(Y\right)=g\left[f\left(\boldsymbol{X}\right)\right], \tag{10.5.1}$$

- 则认为编码器 f 可对原始数据 $\boldsymbol{X}$ 进行有效编码, 此时称映射 g 为解码器.

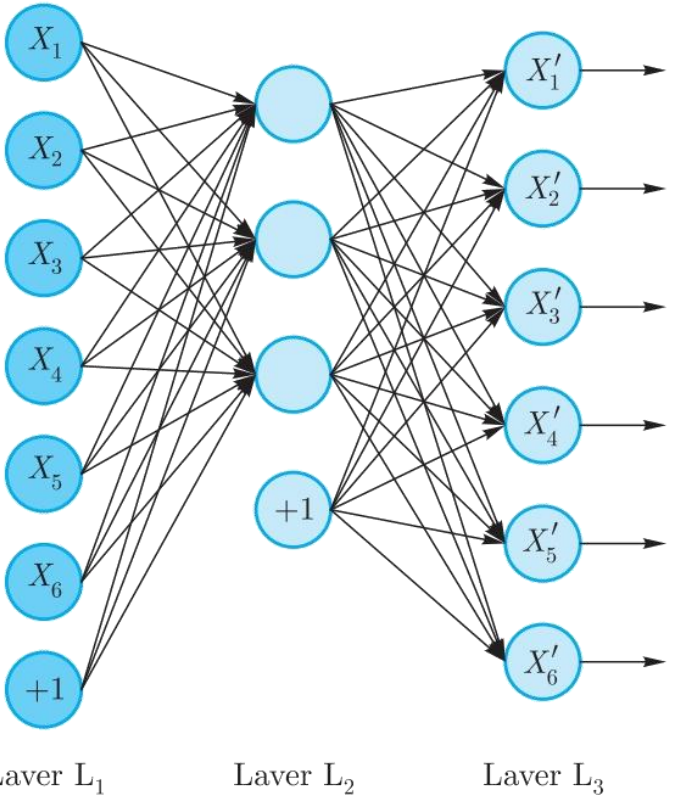
10.5 自编码器

- 通常将用于实现自编码器的神经网络模型称为自编码器. 由于神经网络输入层不具备数据处理能力, 故自编码器中除输入层之外还包括两个分别用于实现编码器 f 和解码器 g 的模块, 每个模块的神经元层数既可为单层也可为多层. 一个单隐层的自编码器网络结构如图 10.24 所示:

- 由图 10.24 可以看到, 自编码器输出层的节点与输入层相等, 训练这个网络以期望得到近似恒等函数. 该模型首先使用编码器对输入数据进行编码, 然后使用解码器尽量将编码数据还原为输入数据, 即模型输出 $\boldsymbol{X}'$ 尽量与模型输入 $\boldsymbol{X}$ 保持一致. 显然, 若编码器 f 和解码器 g 均采用恒等映射, 即 $f(a)=a, g(a)=a$, 则有如下关系:

$$Y=f(\boldsymbol{X})=\boldsymbol{X}, \boldsymbol{X}'=g(Y)=g\left[f(\boldsymbol{X})\right]=\boldsymbol{X}, \qquad (10.5.2)$$

  - 此时恒有 $\boldsymbol{X}'=\boldsymbol{X}$. 但这样的自编码器显然毫无意义, 因为自编码器的主要功能是对原始数据进行编码而非对编码数据进行还原, 即更加注重自编码器中的编码器模块.



10.5 自编码器

- 若自编码器的神经元均使用激活函数 σ, 则对于输入数据 $\boldsymbol{X}=(X_1,X_2,\cdots,X_p)^{\mathrm{T}}$, 由前向计算方法可知自编码器隐含层第 j 个神经元的输出应为

$$f_j(\boldsymbol{X})=\sigma\left(\sum_{i=1}^{p}\omega_{ij}^{(1)}X_i+b_j^{(2)}\right), \tag{10.5.3}$$

  - 由于隐含层数据处理节点个数为 s, 故自编码器的隐含层可将 p 维数据输入转化为 s 维数据 $\boldsymbol{Y}=(f_1(\boldsymbol{X}),f_2(\boldsymbol{X}),\cdots,f_s(\boldsymbol{X}))^{\mathrm{T}}$. 若 $s<p$, 则是对数据 $\boldsymbol{X}$进行降维; 若 $s>p$, 则是对数据 $\boldsymbol{X}$ 进行升维. 自编码器隐含层到输出层的数据处理过程与此类似, 最终输出 $\boldsymbol{X}'$ 为

$$\boldsymbol{X}'=\left(g_1(\boldsymbol{Y}),g_1(\boldsymbol{Y}),\cdots,g_1(\boldsymbol{Y})\right)^{\mathrm{T}}. \tag{10.5.4}$$

- 上式输出层第 j 个神经元输出 $g_j(y)$ 的具体取值为

$$g_j(\boldsymbol{Y})=\sigma\left(\sum_{i=1}^{s}\omega_{ij}^{(2)}f_i(\boldsymbol{X})+b_j^{(3)}\right). \tag{10.5.5}$$

10.5 自编码器

- 若已经完成模型训练过程, 则 X' 应与数据输入 X 差别不大. 自编码器的模型训练通常使用不带标注信息的示例样本, 故这是一种无监督学习方式. 但由于要求自编码器的输入数据与输出数据尽可能接近, 故对于训练样本 X_k, 若模型参数均已知, 则可直接通过对比模型输入 X_k 与输出 X'_k 的差异确定损失函数 $L(X_k, X'_k)$. 这相当于将自编码器的训练样本集看作监督学习的训练样本集 $\{(X_1, X'_1), (X_2, X'_2), \cdots, (X_n, X'_n)\}$, 即根据自编码器的特点将无监督学习方式转化为监督学习方式.

- 可得自编码器模型优化目标函数

$$J(W) = \frac{1}{n}\sum_{k=1}^{n} L(X_k, X'_k), \tag{10.5.6}$$

  - 其中 W 为自编码器的参数向量.

- 损失函数 L 通常采用 X_k, X'_k 之间欧氏距离的平方, 即 $L(X_k, X'_k) = \|X_k - X'_k\|_2^2$.由此可得目标函数的具体形式为

$$J(W) = \frac{1}{n}\sum_{k=1}^{n} \|X_k - X'_k\|_2^2. \tag{10.5.7}$$

10.5 自编码器

- 确定了目标函数之后, 可采用适当模型优化算法并结合反向传播算法对模型参数进行优化, 得到所求的自编码器. 具体过程与反向传播神经网络优化过程类似.
- 如上所述, 若自编码器隐含层神经元数目 s 大于输入数据维度 p, 则可实现对输入数据的升维, 此时只需确保编码后数据向量中取值为 0 的分量较多即可实现对原始数据的稀疏编码.
- 由于自编码器隐含层输出向量 $\boldsymbol{Y}=(f_1(\boldsymbol{X}), f_2(\boldsymbol{X}), \cdots, f_s(\boldsymbol{X}))^{\mathrm{T}}$ 即为编码数据, 故只需使 $\boldsymbol{Y}$ 中取值为 0 的分量较多便可实现对原始数据的稀疏编码. 令自编码器隐含层神经元采用 Sigmoid 激活函数, 则对于训练样本集D $=\{\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots, \boldsymbol{X}_n\}$ 自编码器隐含层第 j 个神经元的平均激活程度为

$$\overline{f}_j(\boldsymbol{X})=\frac{1}{n}\sum_{k=1}^{n} f_i(\boldsymbol{X}_i). \tag{10.5.8}$$

- 为将隐含层第 j 个神经元的输出值限制为 0, 可令 $\overline{f}_j(\boldsymbol{X})=\varepsilon$. 这里$\varepsilon$ 为某个接近于 0 的正数.由于 $f_j(\boldsymbol{X}_i)>0$ 且 $f_j(\boldsymbol{X}_i)$ 在数据集 D 上期望接近于 0, 故 $f_j(\boldsymbol{X}_i)$ 的取值也接近于 0.

10.5 自编码器

- 若对隐含层中大部分神经元输出均施加此约束条件, 即 $\bar{f}_j(\boldsymbol{X}) = \varepsilon, j = 1, 2, \cdots, s$, 则可保证 $\boldsymbol{Y}$ 中大部分元素取值均接近于 0, 实现对原始数据的稀疏编码. 为将该约束条件纳入训练过程, 需调整模型优化的目标函数, 即在原始目标函数 $J(\boldsymbol{W})$ 基础上添加约束 $(\boldsymbol{X}) = \varepsilon, j = 1, 2, \cdots, s$ 的惩罚项 $\lambda\left(\bar{\boldsymbol{f}}(\boldsymbol{X})\right)$, 将目标函数转化为如下形式:

$$J(\boldsymbol{W}) = J'(W) = J(W) + \lambda\left(\bar{\boldsymbol{f}}(\boldsymbol{X})\right) + \frac{1}{n}\sum_{k=1}^{n} L\left(\boldsymbol{X}_k, \boldsymbol{X}'_k\right) + \alpha\lambda\left(\bar{\boldsymbol{f}}(\boldsymbol{X})\right), \quad (10.5.9)$$

  - 其中, α 为惩罚项的权重. 当 α 取值较大时, 所求自编码器的编码数据具有较好的稀疏性, 但会舍弃较多的原始数据信息; 当 α 取值较小时, 所求自编码器的编码数据会保留较多的原始数据信息, 但稀疏性较差.

- 一般, 惩罚项 $\lambda\left(\bar{\boldsymbol{f}}(\boldsymbol{X})\right)$ 具有如下形式:

$$\lambda\left(\bar{\boldsymbol{f}}(\boldsymbol{X})\right) = \frac{1}{s}\sum_{j=1}^{s}\left(\bar{f}_j(\boldsymbol{X}) - \varepsilon\right)^2. \quad (10.5.10)$$

10.5 自编码器

- 除此之外, 还可使用 $\left(\bar{\boldsymbol{f}}(\boldsymbol{X})\right)$与 ε 之间的 KL 散度作为单个隐含层节点的惩罚项, 由此得到如下惩罚项:

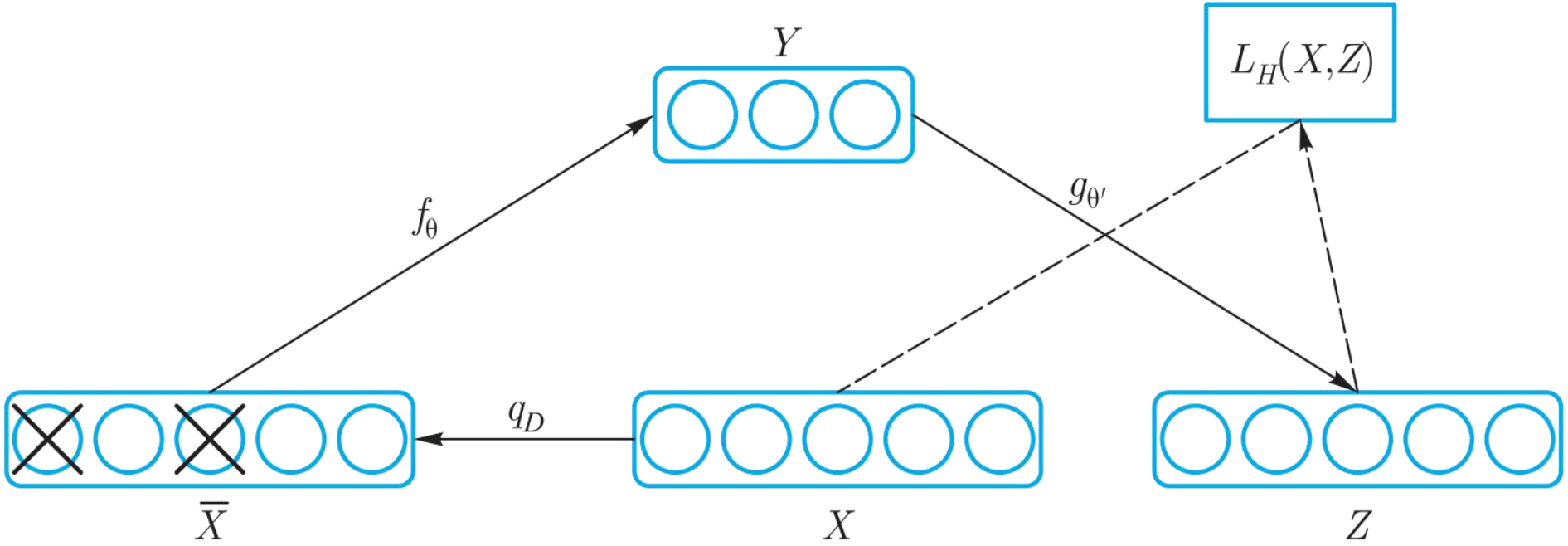
$$\begin{aligned}\lambda\left(\bar{\boldsymbol{f}}(\boldsymbol{X})\right) &= \frac{1}{s}\sum_{j=1}^{s}\mathrm{KL}\left(\varepsilon, \bar{f}_j(\boldsymbol{X})\right)^2 \\ &= \frac{1}{s}\sum_{j=1}^{s}\left[\varepsilon\log\frac{\varepsilon}{\bar{f}_j(\boldsymbol{X})} + (1-\varepsilon)\log\frac{1-\varepsilon}{1-\bar{f}_j(\boldsymbol{X})}\right],\end{aligned} \tag{10.5.11}$$

  - 其中 KL $\left(\varepsilon, \bar{\boldsymbol{f}}(\boldsymbol{X})\right)$ 为 $\bar{f}_j(\boldsymbol{X})$ 与 ε 之间的 KL 散度.

- 上述自编码器模型均要求模型的输入与输出尽可能相一致. 这种自编码器得到的编码数据有时未必是对原始数据的最优表示.

- 若某种自编码器能对被破坏的数据 $\bar{\boldsymbol{X}}$ (如图 10.25) 进行编码并将其解码为真实原始数据 $\boldsymbol{X}$, 则该自编码器的编码方式显然更为有效. 通常称这种通过引入噪声来增加编码鲁棒性的自编码器为降噪自编码器, 如图 10.25 所示.

10.5 自编码器



- 对于输入数据 $\boldsymbol{X}$, 按照 q_D 分布进行加噪 “损坏”, 由图 10.25 可以看到, 加噪过程是按照一定的概率将输入层的某些节点清零, 然后将其作为自编码器的输入进行训练, 除了输入层数据的处理不同, 其余部分都和自编码器相同.

10.6　玻尔兹曼机

10.6 玻尔兹曼机

- 玻尔兹曼机 (Boltzmann machines, BM) 是一种反馈随机神经网络, 其具有两种输出状态, 即 0 或 1. 输出状态的取值根据概率统计方法取得的结果决定, 这种概率统计方法类似于一个玻尔兹曼分布. 在神经元状态变化中引入了概率, 网络的平衡状态服从玻尔兹曼分布, 其运行机制基于模拟退火算法,可以理解为

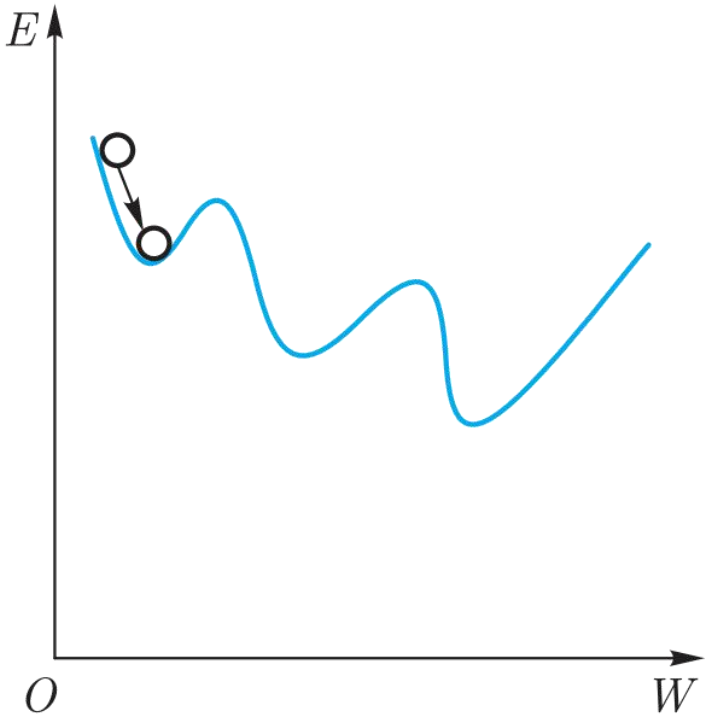
离散霍普菲尔德 (Hopfield) 神经网络 + 模拟退火 + 隐单元 =BM.

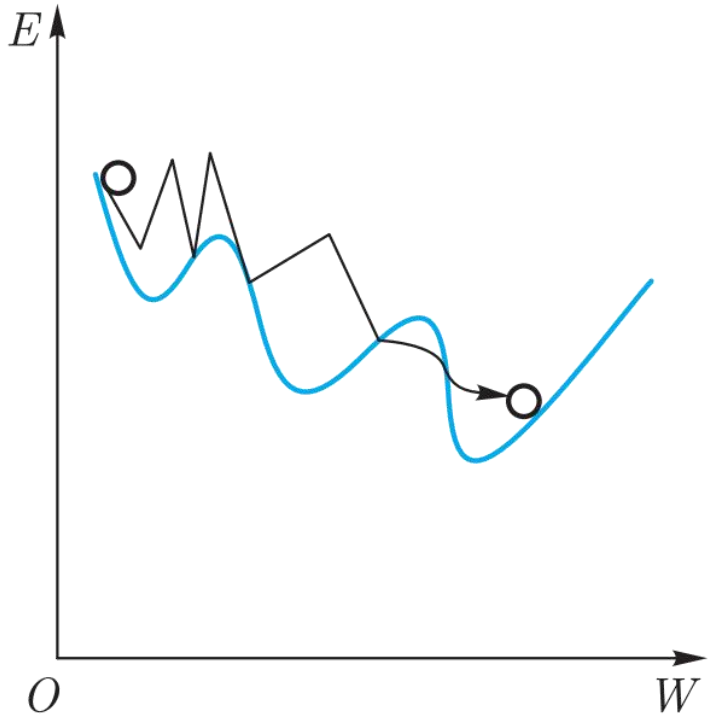
10.6.1 随机神经网络

- 若将 BP 算法中的误差函数看作一种能量函数, 则 BP 算法通过不断调整网络参数使其能量函数按梯度单调下降, 而反馈网络 (霍普菲尔德神经网络) 通过动态演变过程使网络的能量函数沿着梯度单调下降, 在这一点上两类网络的指导思想是一致的. 正因如此, 常常导致网络落入局部极小点而达不到全局最小点, 对于 BP 网络, 局部极小点意味着训练可能不收敛; 对于霍普菲尔德网络, 则得不到期望的最优解. 导致这两类网络陷入局部极小点的原因是: 网络的误差函数或能量函数是具有多个极小点的非线性空间, 而所用的算法却一味追求网络误差或能量函数的单调下降. 也就是说, 算法赋予网络的是只会 “下山” 而不具备 “爬山” 的能力. 如果为具有多个局部极小点的系统打一个形象的比喻, 设想托盘上有一个凸凹不平的多维能量曲面, 若在该曲面上放置一个小球, 它在重力作用下, 将滚入最邻近的一个低谷 (局部最小点) 而无法跳出. 但该低谷不一定就是曲面上最低的那个低谷 (全局最小点). 因此, 局部极小问题只能通过改进算法来解决. 随机神经网络可赋予网络既能 “下坡” 也能 “爬山” 的本领, 因而能有效地克服上述缺陷. 随机网络与其他神经网络相比有两个主要区别:

10.6.1 随机神经网络

- (1) 在学习阶段, 随机网络不像其他网络那样基于某种确定性算法调整权值, 而是按某种概率分布进行修改.
- (2) 在运行阶段, 随机网络不是按某种确定性的网络方程进行状态演变,而是按某种概率分布决定其状态的转移. 神经元的净输入不能决定其状态取1 还是取 0, 但能决定其状态取 1 还是取 0 的概率. 这就是随机神经网络算法的基本概念, 图 10.26 为随机网络算法与梯度下降算法区别的示意图.



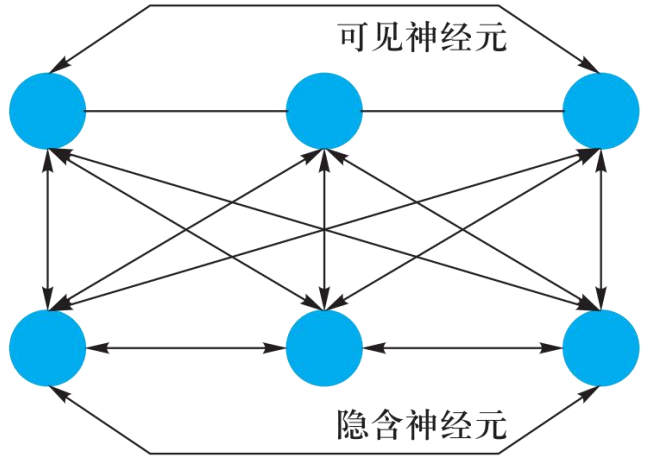


10.6.2 模拟退火算法

- 模拟退火算法是随机网络中解决能量局部极小问题的一个有效方法, 其基本思想是模拟金属退火过程, 金属退火过程大致是, 先将物体加热至高温,使其原子处于高速运动状态, 此时物体具有较高的内能; 然后, 缓慢降温, 随着温度的下降, 原子运动速度减慢, 内能下降; 最后, 整个物体达到内能最低的状态. 模拟退火过程相当于沿水平方向晃动托盘, 温度高则意味着晃动的幅度大, 小球肯定会从任何低谷中跳出, 而落入另一个低谷. 这个低谷的高度(网络能量) 可能比小球原来所在低谷的高度低 (网络能量下降), 但也可能反而比原来高 (能量上升). 后一种情况的出现, 从局部和当前来看, 这个运动方向似乎是错误的; 但从全局和发展的角度看, 正是由于给小球赋予了 “爬山”的本事, 才使它有可能跳出局部低谷而最终落入全局低谷. 当然, 晃动托盘的力度要合适, 并且还要由强至弱 (温度逐渐下降), 小球才不致因为有了 “爬山” 的本领而越爬越高.
- 在随机网络学习过程中, 先令网络权值作随机变化, 然后计算变化后的网络能量函数. 网络权值的修改应遵循以下准则: 若权值变化后能量变小, 则接受这种变化; 否则也不应完全拒绝这种变化, 而是按预先选定的概率分布接受权值的这种变化. 其目的在于赋予网络一定的 “爬山” 能力. 实现这一思想的一个有效方法就是 Metropol et al.[124] 提出的模拟退火算法.

10.6.3 BM

- 神经网络模型的训练构造通常使用目标函数最小化的优化方式实现. 前述各类前馈神经网络模型的训练构造均从误差最小化的角度设计目标函数,对此类目标函数进行优化后可保证在训练集上的整体预测误差达到最小且具备一定的泛化能力. 事实上, 还可从系统稳定性角度出发设计目标函数. 由于系统越稳定其能量越低, 故为得到一个稳定的模型输出, 可设计与网络模型相关的能量函数作为网络模型优化的目标函数, 由此实现对神经网络模型的优化求解. BM 便是此类神经网络的代表模型之一. BM 包含可视层与隐含层 (如图 10.27) 所示, 通过可视层神经元完成与外部的信息交互且可视层与隐含层的所有神经元均参与信息处理过程. 该模型的理想效果是获得训练集 D 中样本在模型稳定状态下的输出值.



- BM 中所有神经元两两之间均存在信息传递且任意两个神经元之间的连接权重均相等, 若使用 ω_{ij} 表示 BM 中的 i 个神经元到第 j 个神经元的连接权重, 则有 $\omega_{ij} = \omega_{ji}$, 当 $i=j$ 时有 $\omega_{ij} = \omega_{ji} = 0$. BM 中每个神经元的输出信号均限制为 0 或 1, 并且每个神经元的状态取值具有一定的随机性, 即以一定概率输出 0 或 1, 这个概率与该神经元的输入相关.

10.6.3 BM

- 具体地说, 对于包含 k 个神经元的 BM, 由于其第 j 个神经元的输入数据为其他所有神经元的输出信号, 故该节点的总输入为

$$I_j = \sum_{i=1}^{k} \omega_{ij} O_i + \theta_j, \tag{10.6.1}$$

  - 其中 O_i 为第 i 个神经元的输出信号, θ_j 为第 j 个神经元所对应的偏置项. 在网络中添加一个取值恒为 1 的第 0 个神经元作为偏置项, 则可将偏置项 θ_j 表示为连接权重 ω_{0j}, 则有

$$I_j = \sum_{i=1}^{k} \omega_{ij} O_i. \tag{10.6.2}$$

- 此时可将第 j 个神经元的输出信号 O_j 取值为 1 的概率定义为

$$P\left(O_j = 1\right) = \frac{1}{1 + \mathrm{e}^{-\frac{I_j}{T}}}, \tag{10.6.3}$$

10.6.3 BM

- 相应地, 该神经元输出为 0 的概率为

$$P\left(O_j=0\right)=\frac{\mathrm{e}^{-\frac{I_j}{T}}}{1+\mathrm{e}^{-\frac{I_j}{T}}}. \tag{10.6.4}$$

- 通常称参数 T 为温度.

- BM 所采用能量函数具体形式如下:

$$J\left(\omega_{ij},O_i,O_j\right)=-\frac{1}{2}\sum_{i=0}^{k}\sum_{j=0}^{k}\omega_{ij}O_iO_j. \tag{10.6.5}$$

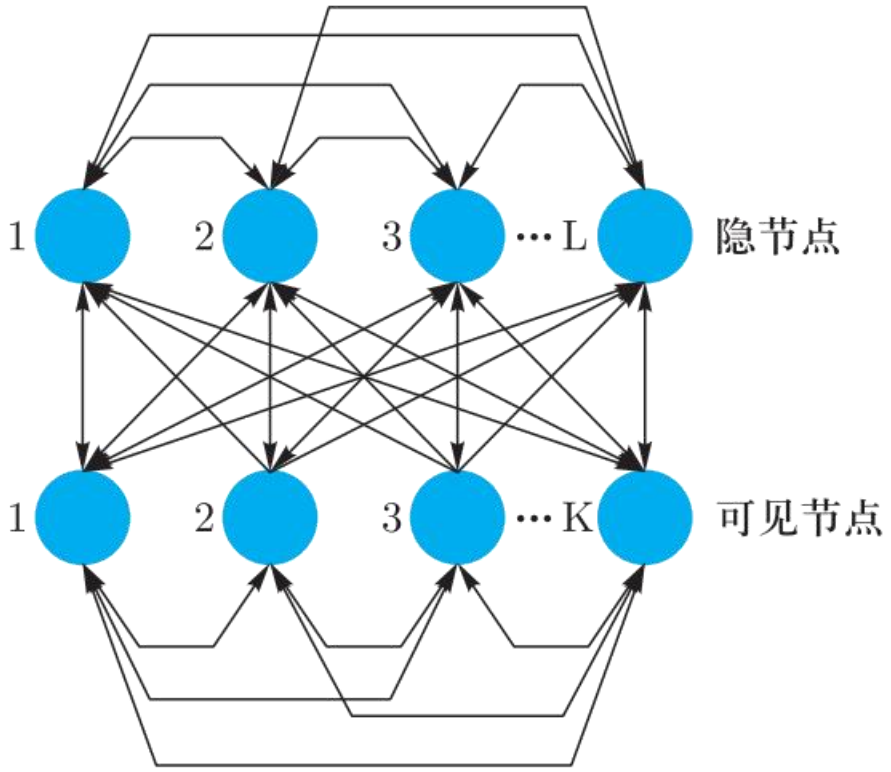
- 对于第 j 个节点的输出 O_j :

  - 若 $O_j=0$, 则对于 $i \leqslant$ j, 有 $J(\ \omega_{ij}\ , O_i, 0)=0$;
  - 若 $O_j=1$, 则对于 $i \leqslant$ j, 有 $J(\ \omega_{ij}\ , O_i, 0)=-\frac{1}{2}\sum_{i=0}^{k}\omega_{ij}O_i=-\frac{1}{2}I_j$.

10.6.3 BM

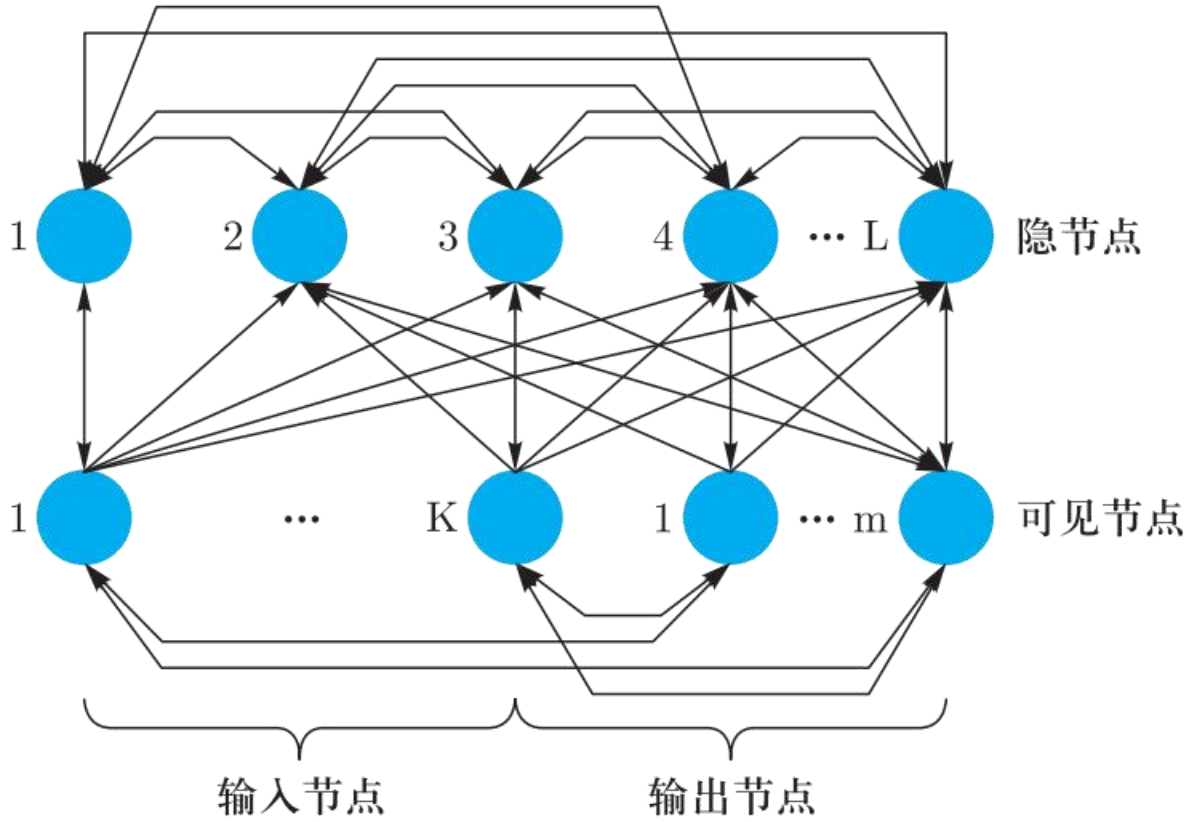
- 若满足 $J(\omega_{ij}, O_i, 1) > J(\omega_{ij}, O_i, 0) = 0$, 则说明 O_j 取 1 时的能量高于 O_j 取 0 时的能量且 $I_j < 0$, 可知 $P(O_j = 1) < 0.5 < P(O_j = 0)$, 此时第 j 个神经元以较大概率输出使得网络能量降低的取值, 但仍有可能选择使得网络能量升高的取值.
- 若满足 $J(\omega_{ij}, O_i, 1) < J(\omega_{ij}, O_i, 0) = 0$, 则说明 O_j 取 1 时的能量低于 O_j 取 0 时的能量且 $I_j > 0$, 故有 $P(O_j = 1) > 0.5 > P(O_j = 0)$, 此时第 j 个神经元倾向于选择使得网络能量更低的取值作为输出.
- 由以上分析可知, BM 的各神经元均倾向于选择使得网络能量降低的输出值, 故该网络模型的能量函数取值呈现总体下降趋势, 但亦存在能量函数取值上升的可能性. 这样可有效避免网络模型的优化计算陷入局部最优.
- 用 BM 网络进行联想时, 可通过学习用网络稳定状态的概率来模拟训练集样本的出现概率. 根据学习类型, BM 网络可分为自联想和异联想两种情况 (如图 10.28 所示). 自联想 BM 网络中的可见节点既是输入节点又是输出节点, 隐节点的数目由学习的需要而定, 最少可以为 0. 异联想 BM 网络中的可见节点需按功能分为输入节点组和输出节点组.

10.6.3 BM



既是输入节点也是输出节点

(a) 自联想BM机



(b) 异联想BM机

10.7　深度学习实践

实践代码