



河海大学
HOHAI UNIVERSITY

神经网络与深度学习实践

目录

Contents

01

神经网络发展简史

02

神经网络基础

03

深度卷积神经网络

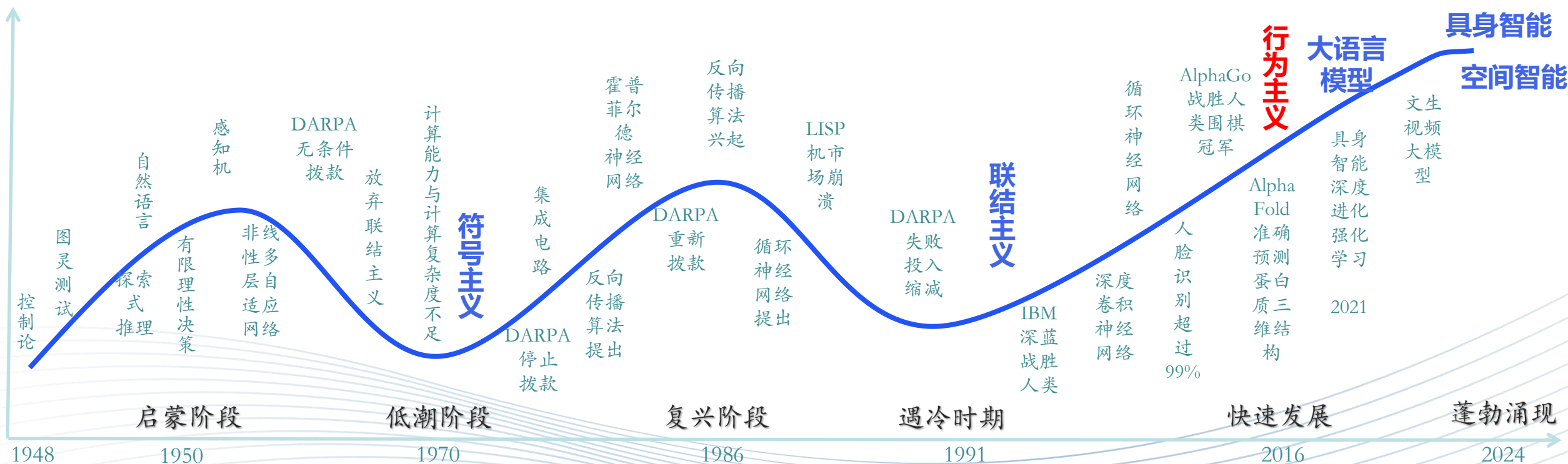
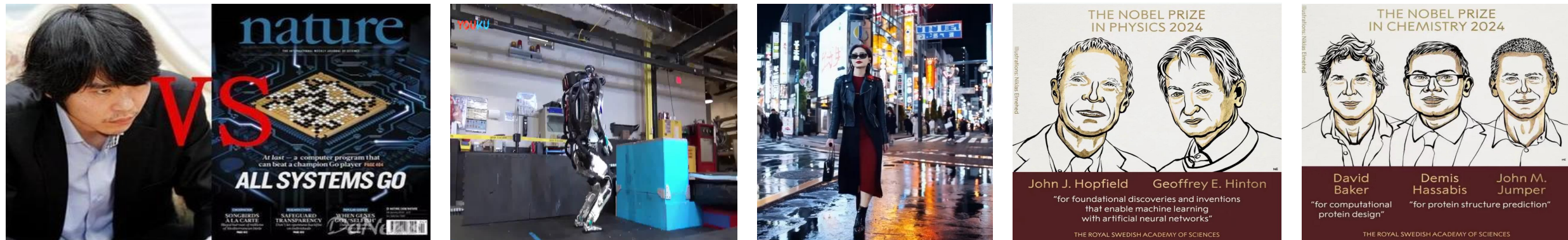
04

图像分类实践

01 / PART ONE

神经网络发展简史

神经网络发展简史



常用深度学习框架

- 简易和快速的原型设计
- 自动梯度计算
- 无缝CPU和GPU切换

 PaddlePaddle

 TensorFlow

 Chainer

 Microsoft
CNTK

 Caffe2

 Keras

 theano

 PYTORCH

 APACHE
mxnet

02 / PART TWO

神经网络基础

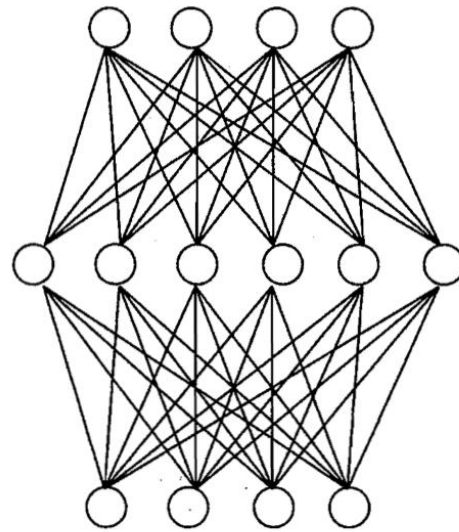


全连接前馈 神经网络

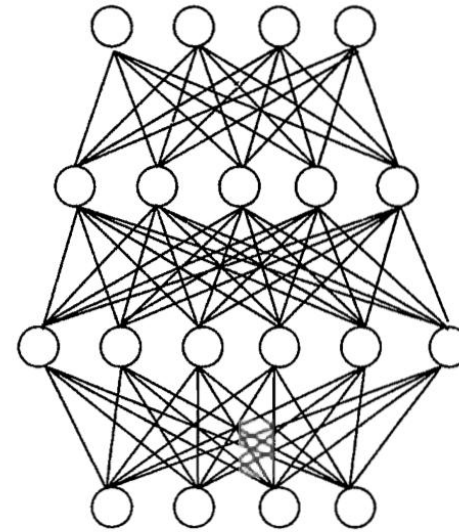


全链接前馈神经网络

- 多层神经元可以解决非线性分类问题。
- 常见的神经网络结构是如下图所示的层级结构，**每层神经元与下一层神经元全互连，神经元不存在同层连接，也不存在跨层互联**，这种结构称为**全连接前馈神经网络**。



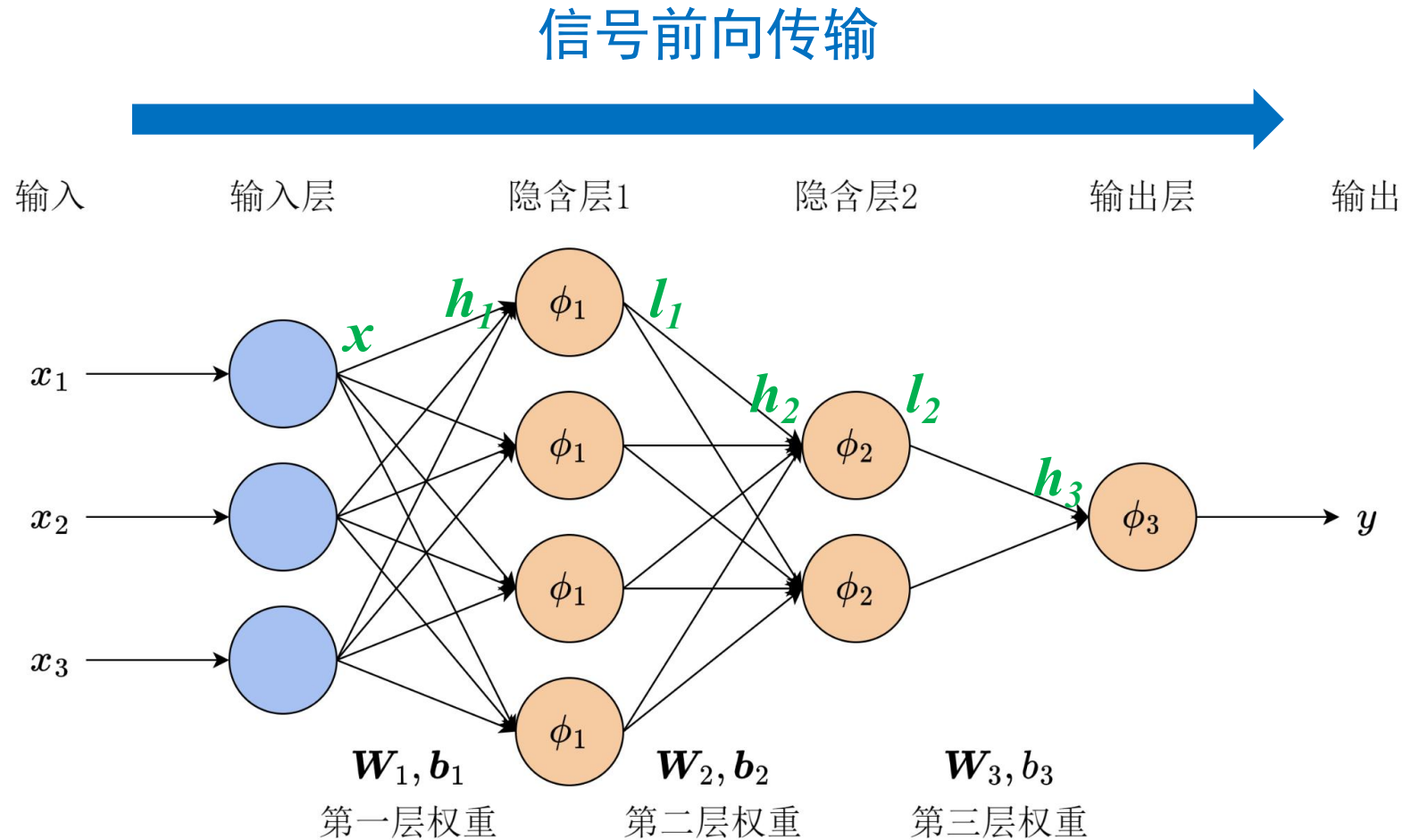
(a) 单隐层前馈网络



(b) 双隐层前馈网络

神经网络信号前向传输过程

□ 以3层神经网络为例

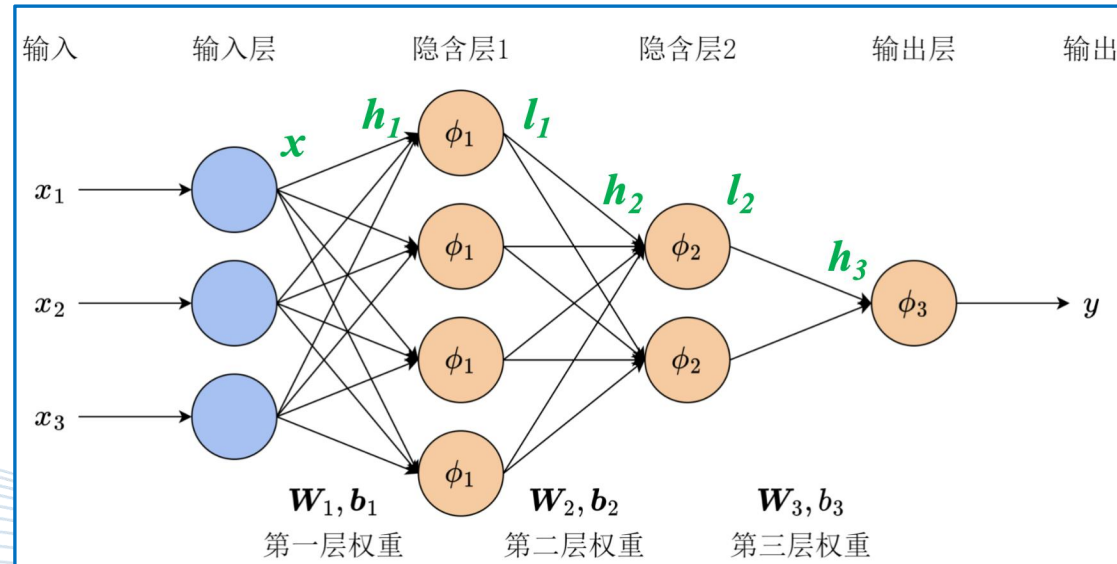


神经网络信号前向传输过程

□ 输出层的设计

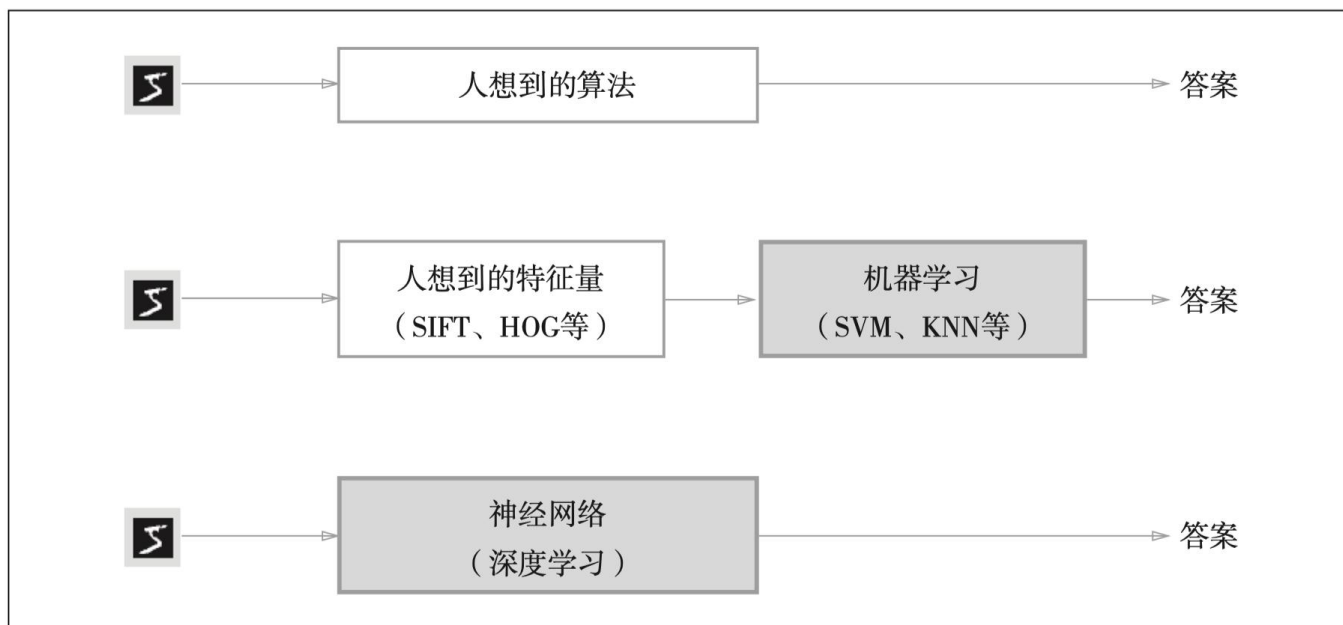
神经网络可以用在**分类**问题和**回归**问题上，不过需要根据情况改变输出层的激活函数。

一般而言，回归问题用**恒等函数**，分类问题用**softmax函数**。



神经网络的学习

- 神经网络中的“学习”是指**自动获取最优权重参数**的过程。
- 为了使神经网络能进行学习，将导入**损失函数**这一指标。而学习的目的就是
以该损失函数为基准，找出能使它的值达到最小的权重参数。
- 学习一般利用函数导数的**梯度法**。



深度学习是一种“端到端”
(end-to-end)
学习方式

神经网络学习步骤

□ 步骤1 (mini-batch)

从训练数据中随机选出一部分数据 (mini-batch)。目标是减小mini-batch的损失函数的值。

□ 步骤2 (计算梯度)

为了减小mini-batch的损失函数的值，需要求出各个权重参数的梯度。梯度表示损失函数的值减小最多的方向。

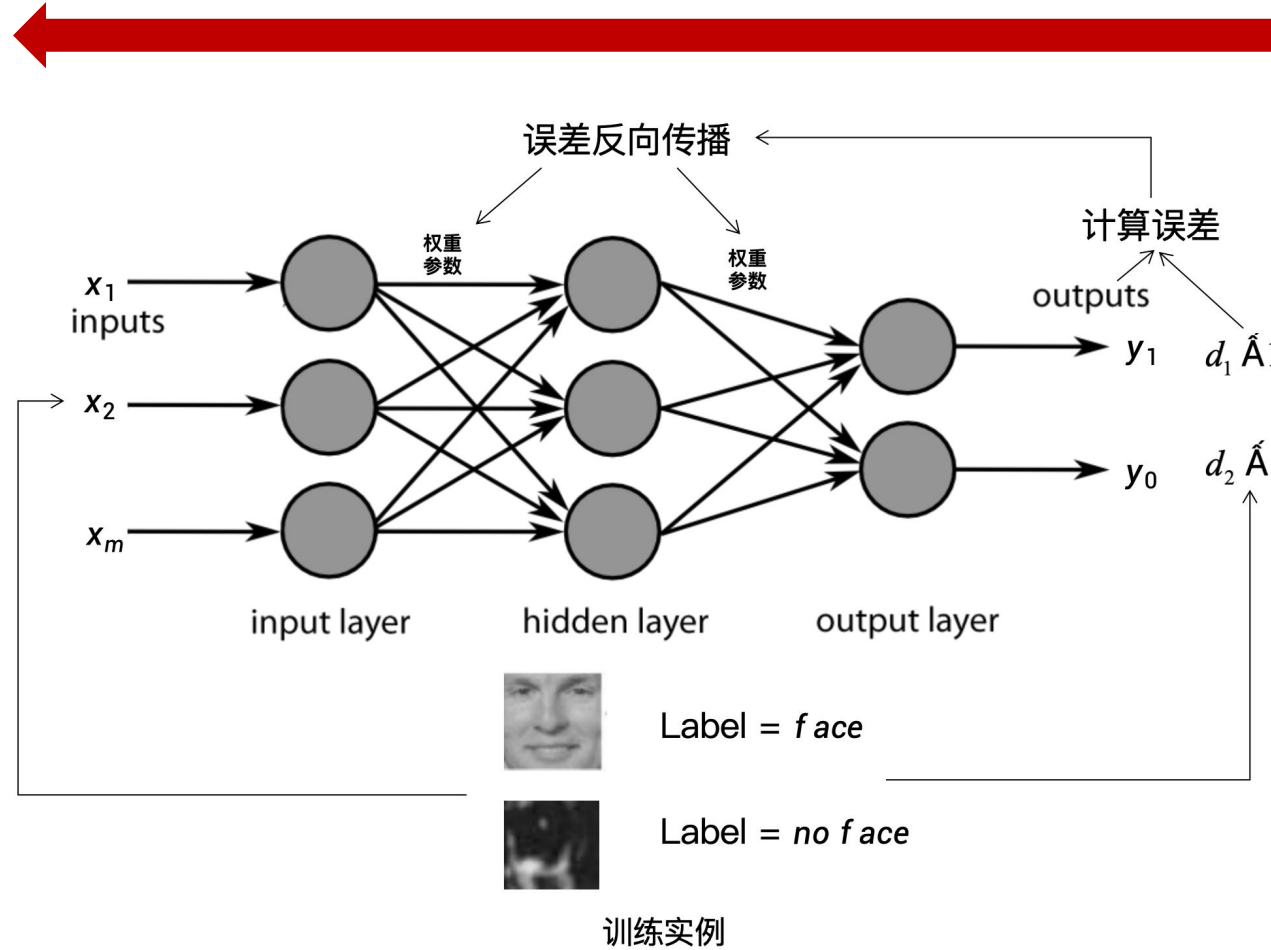
□ 步骤3 (更新参数)

将权重参数沿梯度方向进行微小更新。

□ 步骤4 (重复)

重复步骤1、步骤2、步骤3，直至满足停止条件。

误差反向传播



误差反向传播
(训练多层神经网络)

信号正向传输

如何计算神经网络的梯度？

□神经网络为一个复杂的复合函数

■链式法则

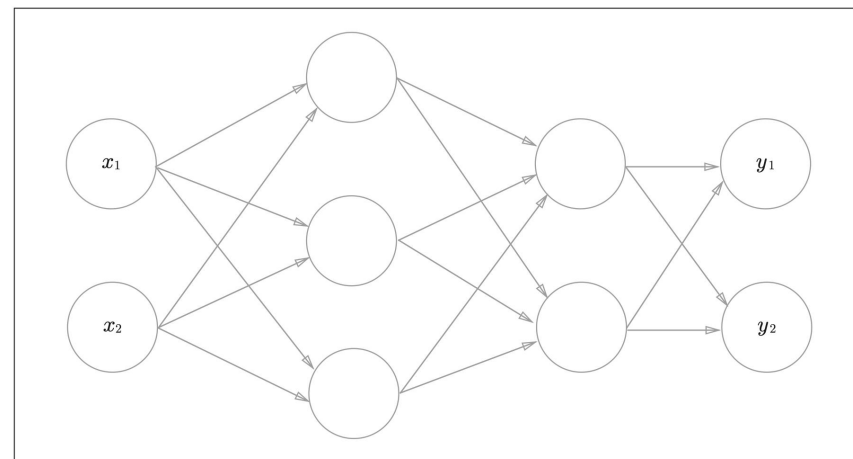
$$y = f^5(f^4(f^3(f^2(f^1(x))))) \rightarrow \frac{\partial y}{\partial x} = \frac{\partial f^1}{\partial x} \frac{\partial f^2}{\partial f^1} \frac{\partial f^3}{\partial f^2} \frac{\partial f^4}{\partial f^3} \frac{\partial f^5}{\partial f^4}$$

□反向传播算法

■根据前馈网络的特点而设计的高效方法

□一个更加通用的计算方法

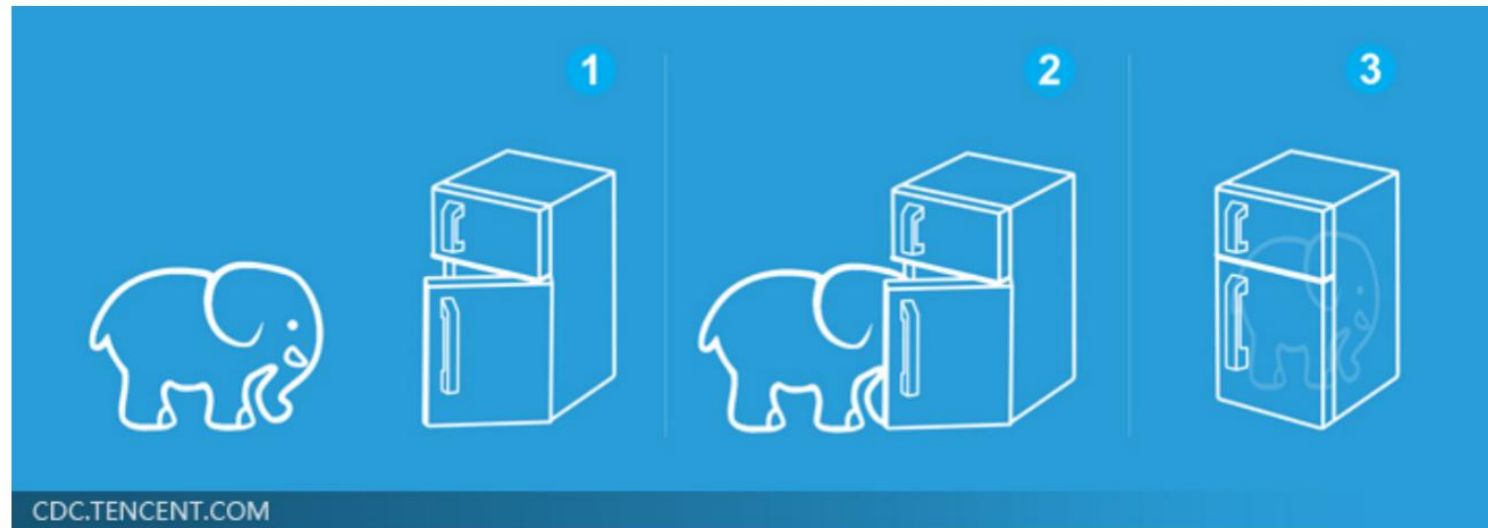
■自动微分 (Automatic Differentiation, AD)



神经网络学习的三个步骤



Deep Learning is so simple

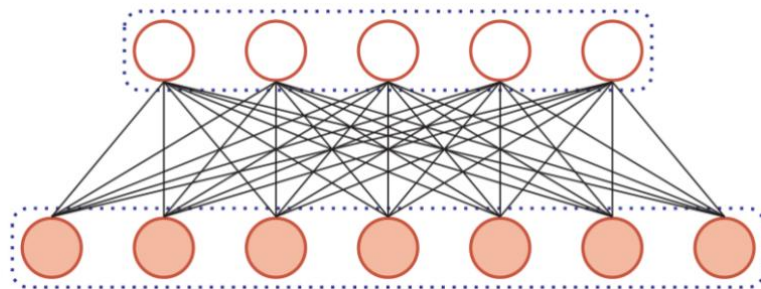


03 / PART THREE

深度卷积神经网络

全连接前馈神经网络的问题

▶ 权重矩阵的参数非常多

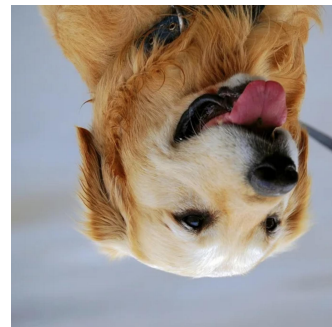
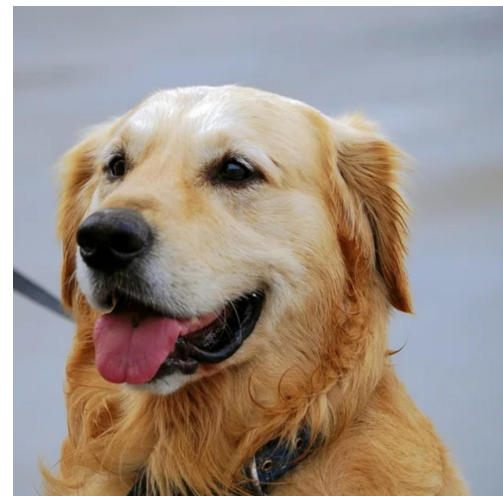
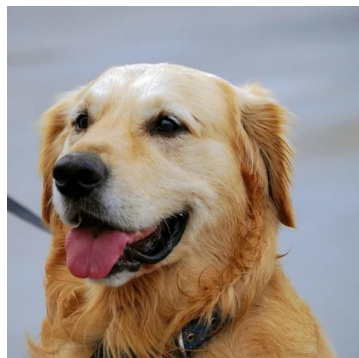


▶ 局部不变性特征

▶ 自然图像中的物体都具有局部不变性特征

▶ 尺度缩放、平移、旋转等操作不影响其语义信息。

▶ 全连接前馈网络很难提取这些局部不变特征



卷积神经网络

▶ 卷积神经网络 (Convolutional Neural Networks, CNN)

- ▶ 一种前馈神经网络

- ▶ 受生物学上感受野 (Receptive Field) 的机制而提出的

- ▶ 在视觉神经系统中，一个神经元的感受野是指视网膜上的特定区域，只有这个区域内的刺激才能够激活该神经元。

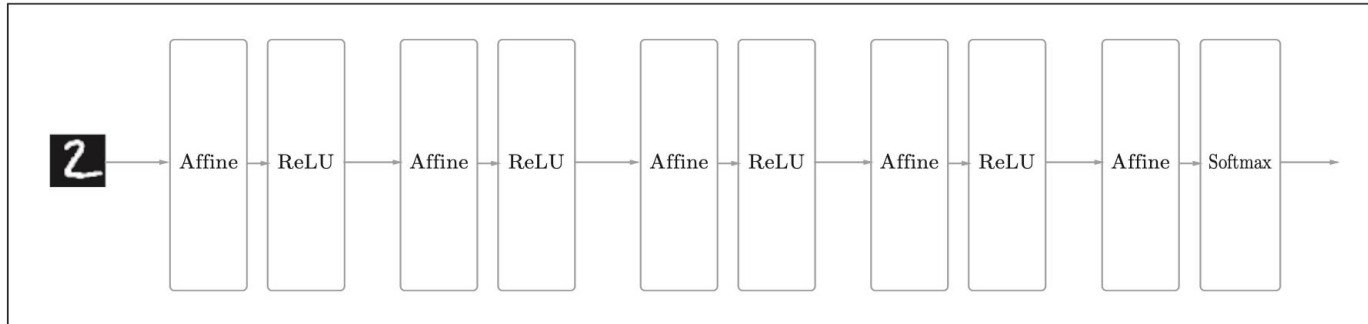
- ▶ 卷积神经网络有三个结构上的特性：

- ▶ 局部连接

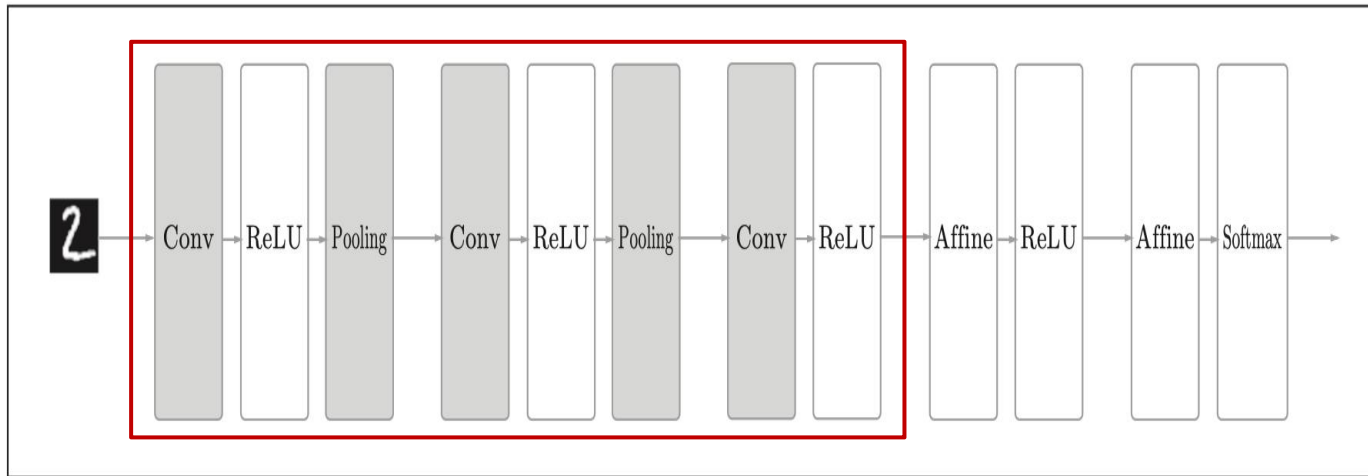
- ▶ 权重共享

- ▶ 空间或时间上的次采样

卷积神经网络整体结构



全连接神经网络结构



卷积神经网络结构

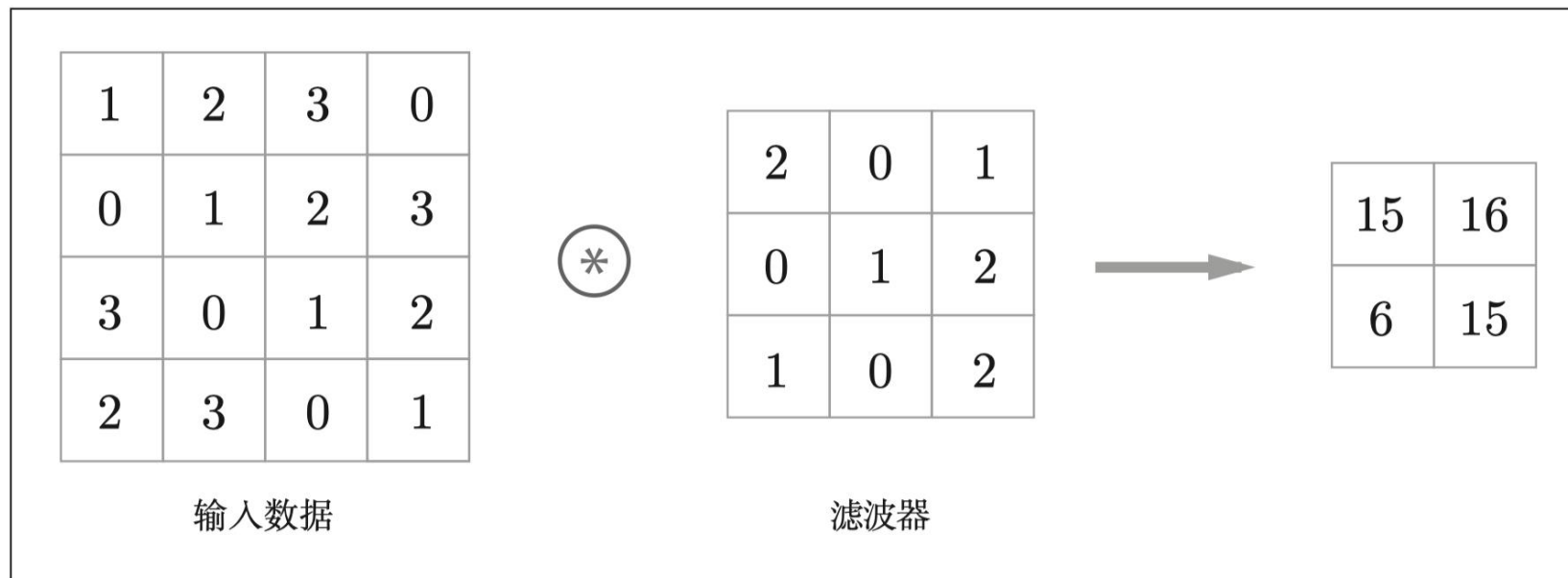
- CNN中新增了 Convolution层和 Pooling 层。CNN的层的连接顺序是 “Convolution-ReLU-(Pooling)”。
- 靠近输出的层中使用了之前的 “Affine-ReLU” 组合。
- 最后的输出层中使用了之前的 “Affine-Softmax” 组合。

卷积运算

□ 卷积层进行的处理就是卷积运算。卷积运算相当于图像处理中的“**滤波器运算**”

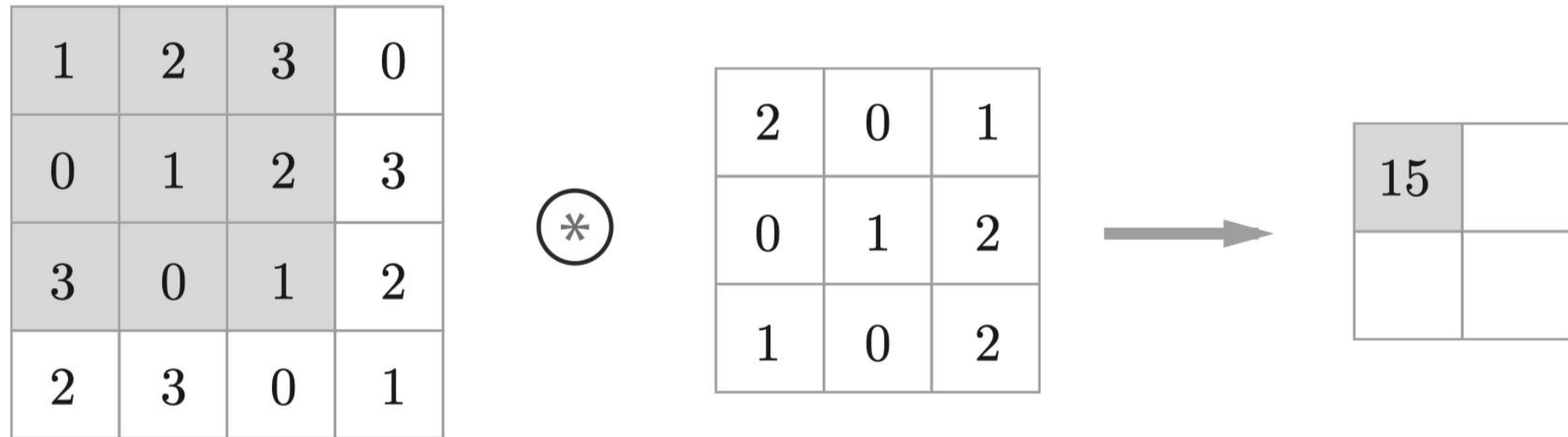
一个输入信息 $\mathbf{X}$ 和滤波器 $\mathbf{W}$ 的二维卷积定义为

$$\mathbf{Y} = \mathbf{W} * \mathbf{X}, \quad y_{ij} = \sum_{u=1}^U \sum_{v=1}^V w_{uv} x_{i-u+1, j-v+1}.$$

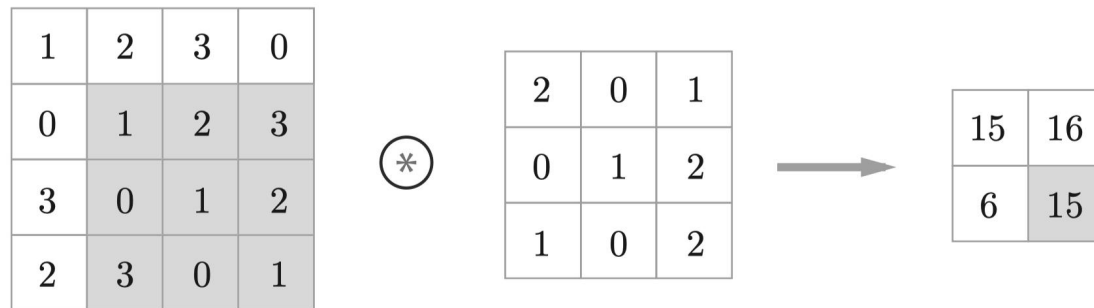
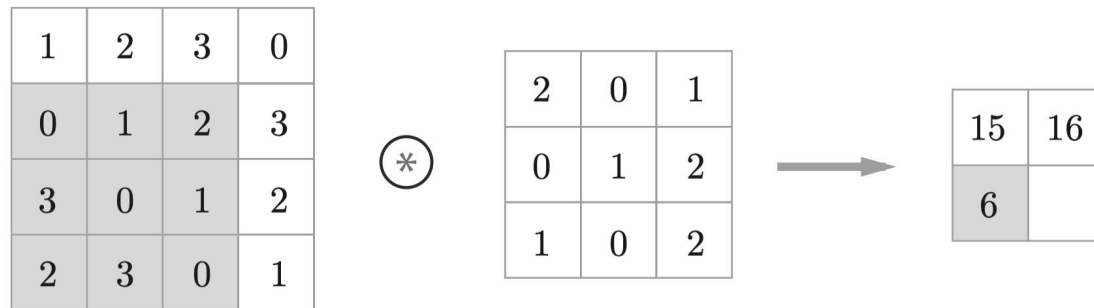
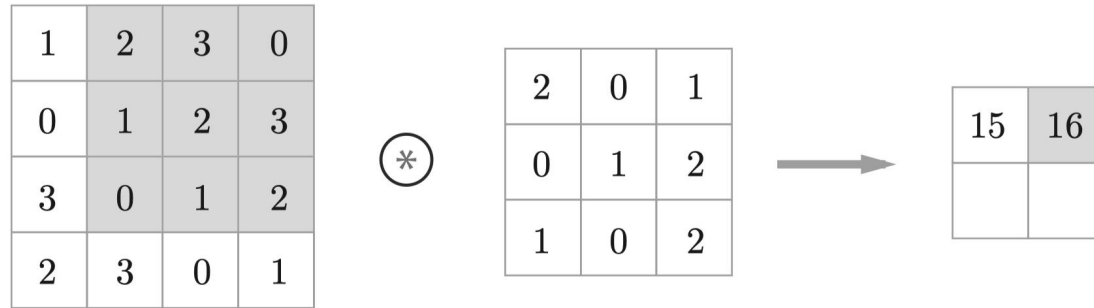


卷积运算

- ❑ 卷积计算过程就是将各个位置上滤波器（卷积核）的元素和输入的对应元素相乘，然后再求和，将这个结果保存到输出的对应位置。



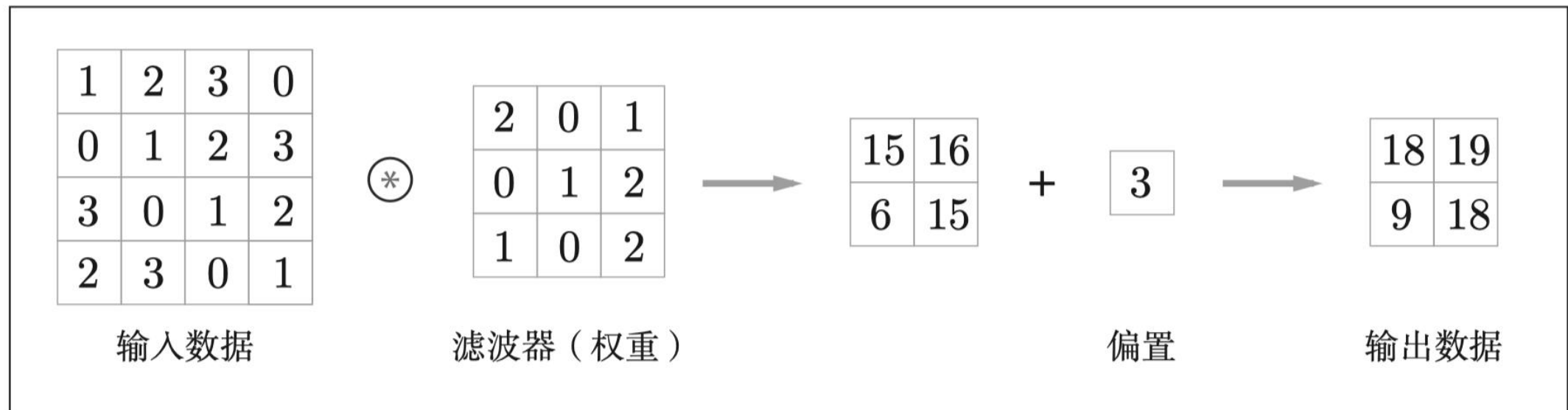
卷积运算



■ 将这个过程在所有位置都进行一遍，就可以得到卷积运算的输出。

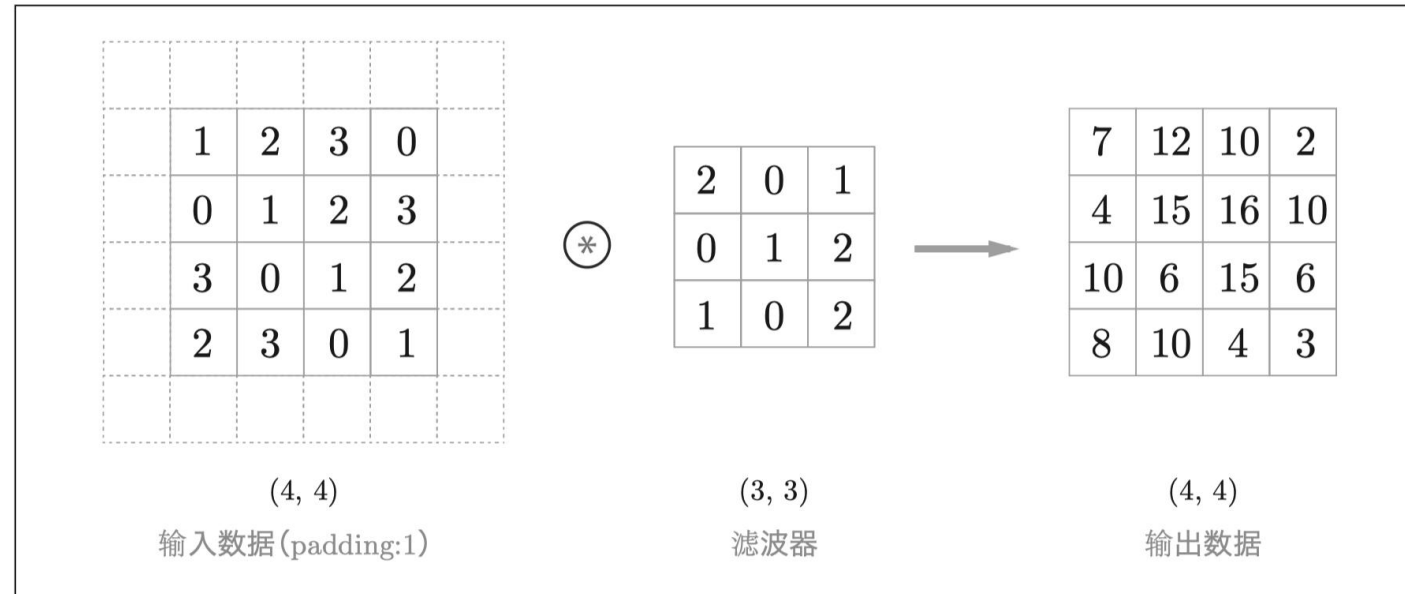
卷积运算

- ❑ CNN中，滤波器（卷积核）的参数就对应全连接神经网络中的权重。
- ❑ 在全连接的神经网络中，除了权重参数，还存在偏置。CNN中也存在偏置，偏置通常只有1个，这个值会被加到应用了滤波器的所有元素上。
- ❑ 经过卷积运算后得到的输出数据也称为**特征图**。



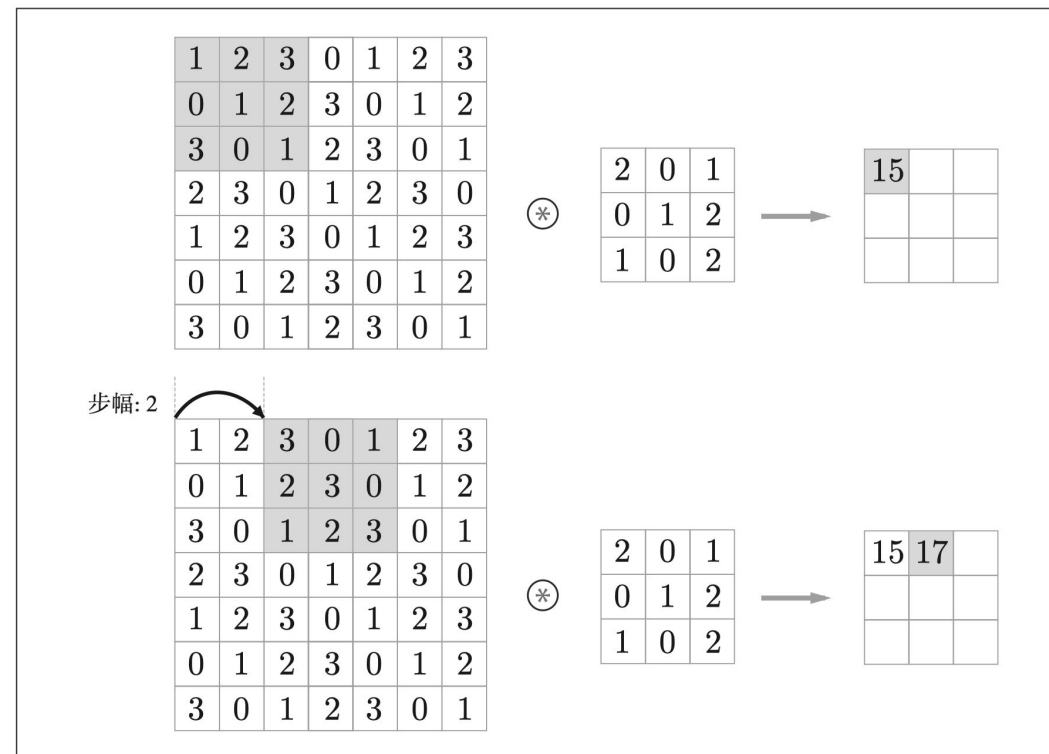
卷积层-填充

- ❑ 为了抵消卷积对图像尺寸的影响，有时要向输入数据的周围填入固定的数据，称为**填充（padding）**，是卷积运算中经常会用到的处理。
- ❑ 填充常用的方式有**全零填充**、**常数填充**、**边界扩展**填充等等。其中，全零填充和常数填充即用0或某指定的常数进行填充，边界扩展填充则是将边界处的值向外复制。



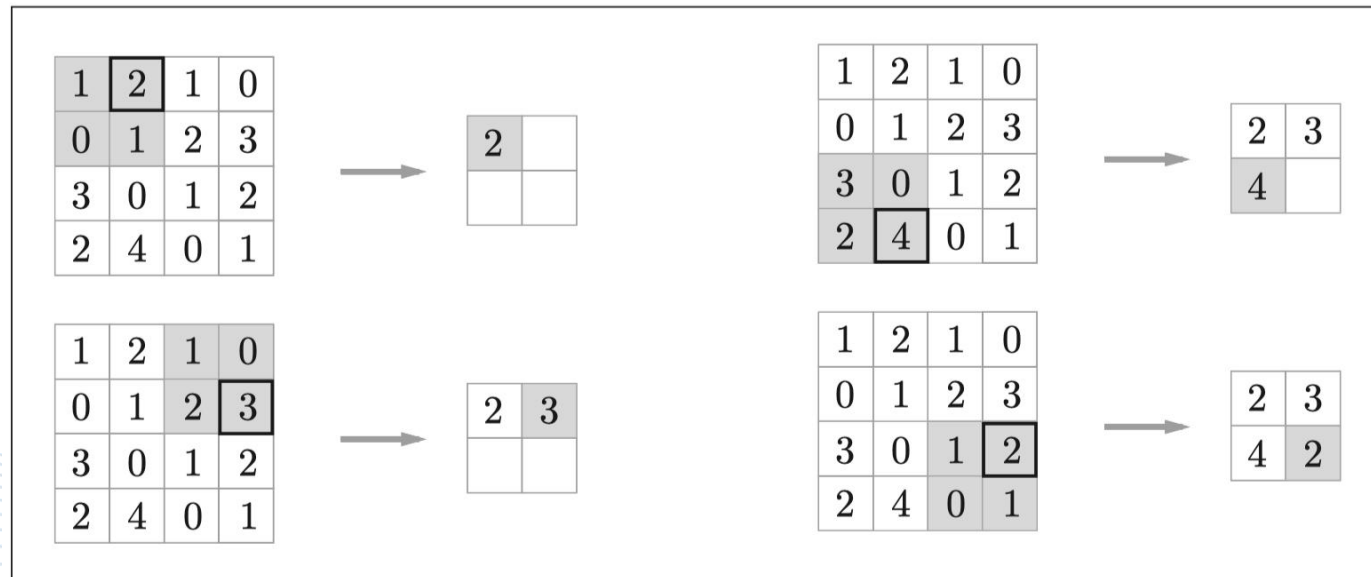
卷积层-步幅

应用滤波器的位置间隔称为**步幅**（stride）。步幅设为2，应用滤波器的窗口的间隔变为2个元素。



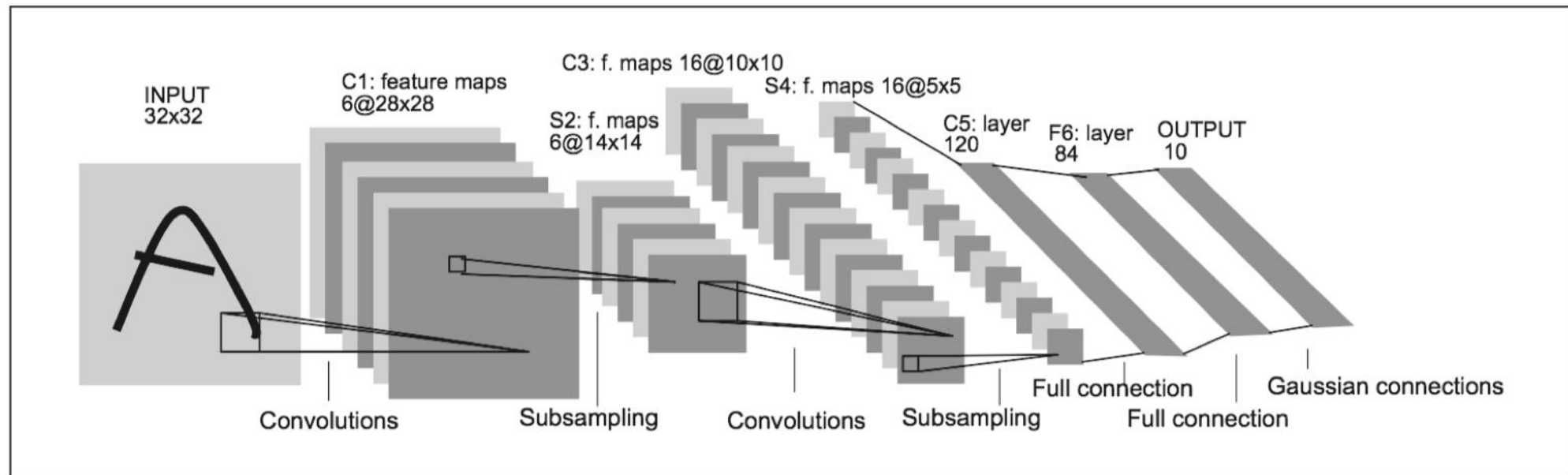
池化层 (Pooling)

- ❑ 池化是缩小高、宽方向上的空间的运算。（**压缩数据**）
- ❑ 例如，按步幅2进行 2×2 的Max池化时的处理顺序如下所示。“Max池化”是获取最大值的运算，“ 2×2 ”表示目标区域的大小。
- ❑ 一般来说，**池化的窗口大小会和步幅设定成相同的值。**



典型卷积神经网络

- ❑ LeNet最早诞生于1994年，经过多次迭代后，于1998年被命名为LeNet-5，是进行手写数字识别的网络。它有连续的卷积层和池化层，最后经全连接层输出结果。
- ❑ 和“现在的CNN”相比，LeNet有几个不同点：（1）LeNet中使用sigmoid函数，而现在的CNN中主要使用ReLU函数；（2）原始的LeNet中使用子采样(subsampling)缩小中间数据的大小，而现在的CNN中Max池化是主流。



典型卷积神经网络

LeNet-5由两个部分组成：

特征提取部分：由**两个卷积层**和**两个平均池化层**组成；

全连接层：由三个全连接层组成。

模型单元结构：

一个卷积层、一个sigmoid激活函数，一个池化层

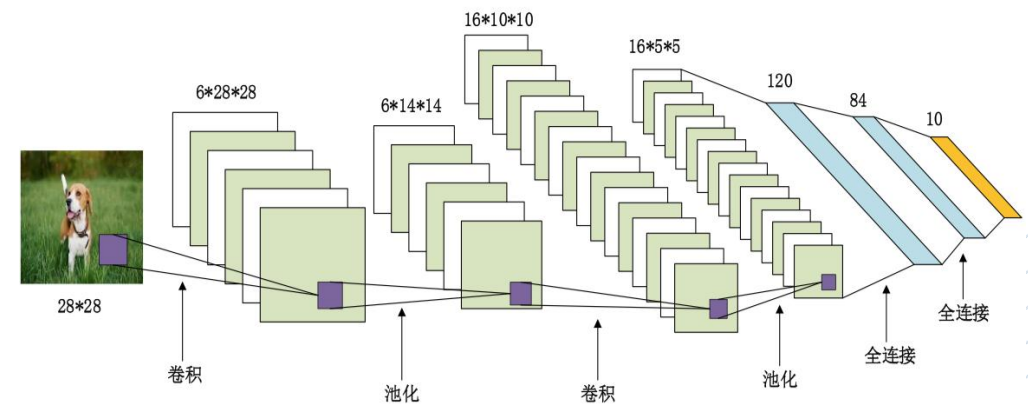
数据的传输：

卷积层输入为4维的数据 (B, C, W, H)

卷积层输出为4维的数据 (B, FN, OW, OH)

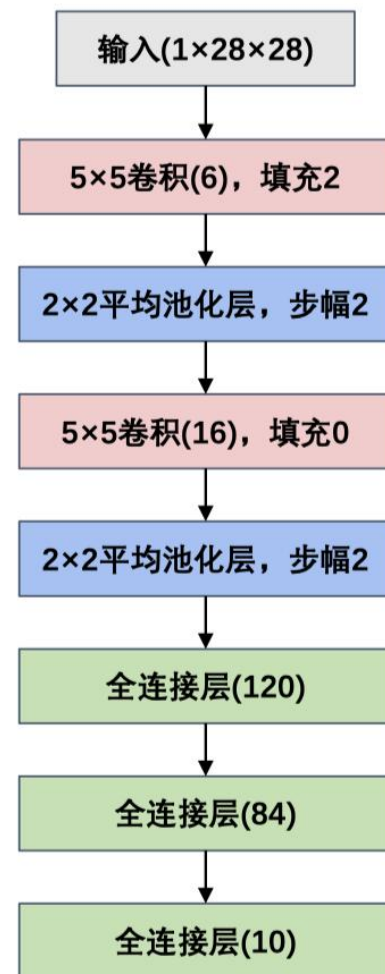
全连接层的输入为2维数据 (B, L)

全连接层的输出为2维数据 (B, FL)



LeNet-5总结

- ❑ 在卷积神经网络中，组合使用卷积层（提取特征）、非线性激活函数sigmoid（激活特征）和全连接层（组合特征）。
- ❑ 为了构造高性能的卷积神经网络，我们通常对卷积层进行排列，逐渐降低其表示的空间分辨率（**感受野随着层数的增加而变大**），同时增加通道数（**卷积核的数量**）。
- ❑ 卷积块编码得到的表征在输出之前需由一个或多个全连接层进行处理。
- ❑ LeNet是最早发布的卷积神经网络之一，它的问世有开创意义。



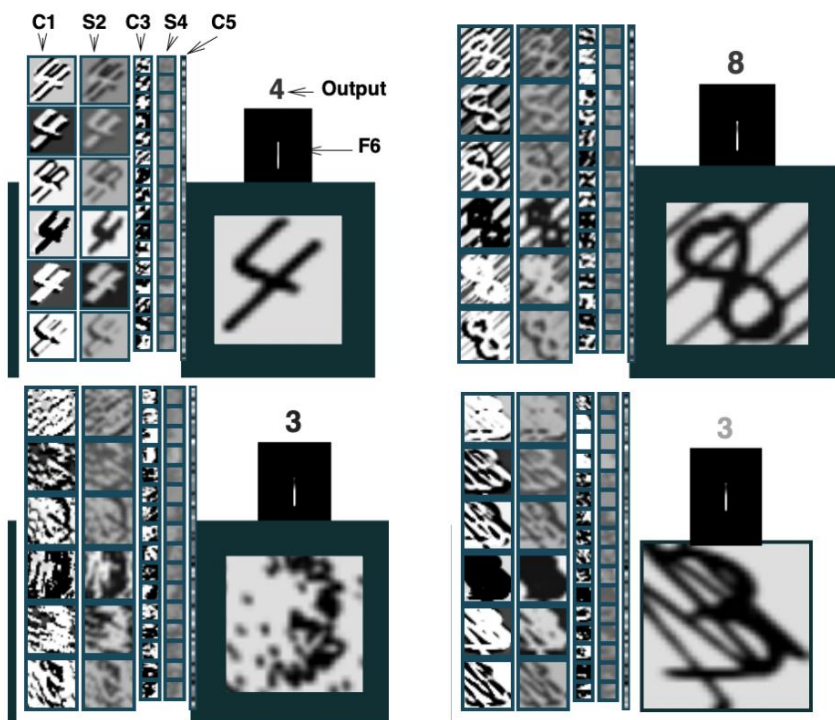
典型卷积神经网络

□ MNIST(手写数字)数据集

- <http://yann.lecun.com/exdb/mnist/>
- 60k训练和10k测试示例

用LeNet-5网络上总共只有82个错误
左边是正确答案，右边是预测的答案

□ 测试错误率0.95%



4	3	8	1	3	4	2	3	6	1	4→6	3→5	8→2	2→1	5→3	4→8	2→8	3→5	6→5	7→3
4	0	7	5	8	6	3	2	8	4	9→4	8→0	7→8	5→3	8→7	0→6	3→7	2→7	8→3	9→4
8	5	4	3	0	2	9	6	9	1	8→2	5→3	4→8	3→9	6→0	9→8	4→9	6→1	9→4	9→1
9	0	1	3	2	9	0	0	0	8	9→4	2→0	6→1	3→5	3→2	9→5	6→0	6→0	6→0	6→8
4	7	9	4	2	9	4	9	9	9	4→6	7→3	9→4	4→6	2→7	9→7	4→3	9→4	9→4	9→4
2	4	8	3	4	6	8	3	3	9	8→7	4→2	8→4	3→5	8→4	6→5	8→5	3→8	3→8	9→8
1	9	6	0	6	9	0	1	4	2	1→5	9→8	6→3	0→2	6→5	9→5	0→7	1→6	4→9	2→1
2	8	9	7	7	6	9	6	5	5	2→8	8→5	4→9	7→2	7→2	6→5	9→7	6→1	5→6	5→0
4	2									4→9	2→8								

04 / PART FOUR

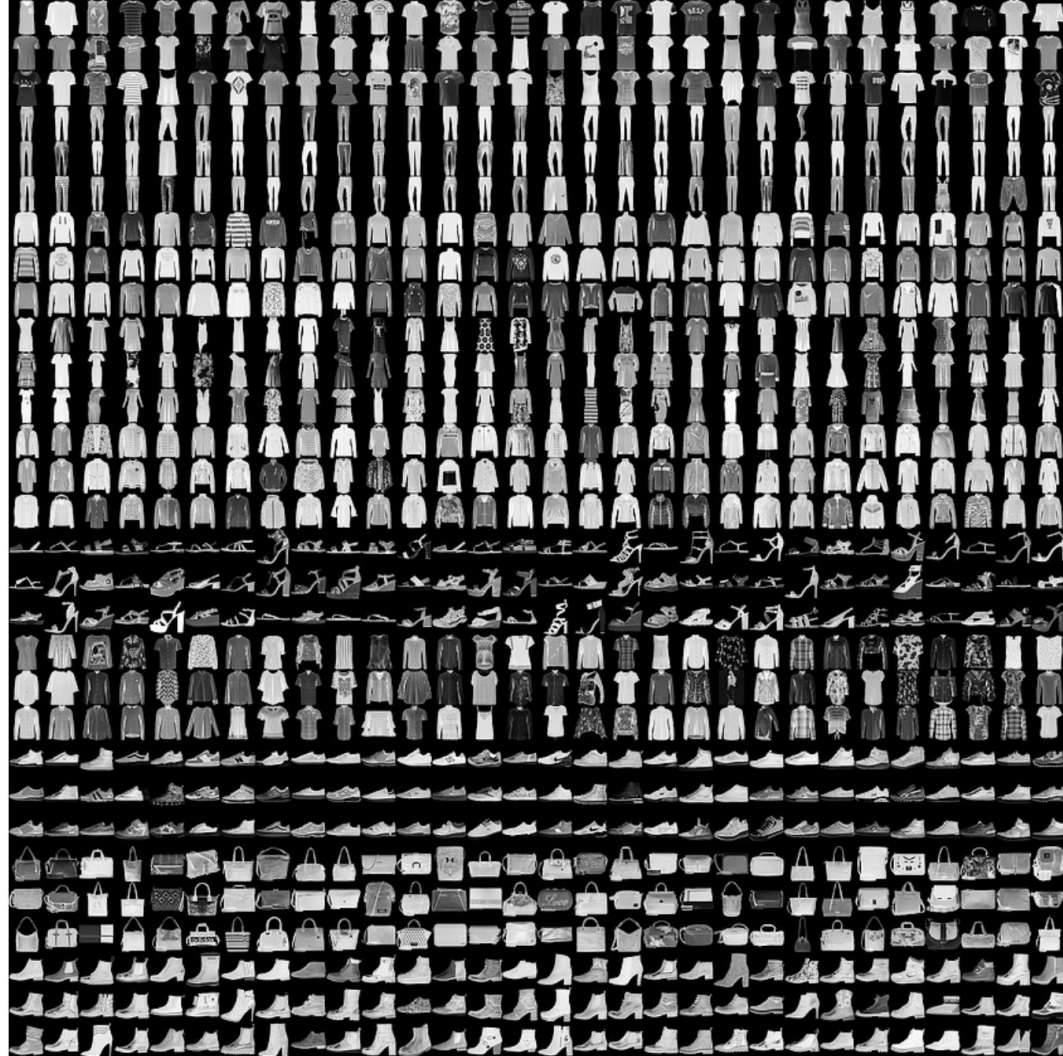
图像分类实践

Fashion-MNIST数据集

- 图像分类数据集中最常用的是手写数字识别数据集MNIST。但大部分模型在MNIST上的分类精度都超过了95%。
- Fashion-MNIST是一个替代MNIST手写数字集的图像数据集。它是由Zalando（一家德国的时尚科技公司）旗下的研究部门提供。其涵盖了来自10种类别的共7万个不同商品的正面图片。
- Fashion-MNIST的大小、格式和训练集/测试集划分与原始的MNIST完全一致。
60000/10000的训练测试数据划分，28x28的灰度图片。

Fashion-MNIST数据集

标注编号	描述
0	T-shirt/top (T恤)
1	Trouser (裤子)
2	Pullover (套衫)
3	Dress (裙子)
4	Coat (外套)
5	Sandal (凉鞋)
6	Shirt (汗衫)
7	Sneaker (运动鞋)
8	Bag (包)
9	Ankle boot (踝靴)



Fashion-MNIST数据集

- ❑ Fashion-MNIST的数据集的存储方式和命名与经典MNIST数据集完全一致。
- ❑ 很多的机器学习库已经内置了Fashion-MNIST数据或接口，方便你直接使用。

名称	描述	样本数量	文件大小	链接	MD5校验和
<code>train-images-idx3-ubyte.gz</code>	训练集图像	60,000	26 MBytes	下载	<code>8d4fb7e6c68d591d4c3dfef9ec88bf0d</code>
<code>train-labels-idx1-ubyte.gz</code>	训练集类别标签	60,000	29 KBytes	下载	<code>25c81989df183df01b3e8a0aad5dffbe</code>
<code>t10k-images-idx3-ubyte.gz</code>	测试集图像	10,000	4.3 MBytes	下载	<code>bef4ecab320f06d8554ea6380940ec79</code>
<code>t10k-labels-idx1-ubyte.gz</code>	测试集类别标签	10,000	5.1 KBytes	下载	<code>bb300cfdad3c16e7a12a480ee83cd310</code>

模型训练

① mini-batch训练

② 信号前向传输

③ 误差反向传播

④ 梯度法更新参数

```
78 # 训练模型，对每一个mini-batch训练和计算
79 for step, (b_x, b_y) in enumerate(train_dataloader):
80     # 将特征和标签放入到训练设备中
81     b_x, b_y = b_x.to(device), b_y.to(device)
82     model.train()
83
84     # 前向传播过程，输入为一个batch，输出为一个batch中对应的预测
85     output = model(b_x)
86     # 查找每一行中最大值对应的行标
87     pre_lab = torch.argmax(output, dim=1)
88     # 计算每一个batch的损失函数
89     loss = criterion(output, b_y)
90
91     # 将梯度初始化为0
92     optimizer.zero_grad()
93     # 反向传播计算
94     loss.backward()
95     # 根据网络反向传播的梯度信息来更新网络的参数，以起到降低loss函数计算值的作用
96     optimizer.step()
97     # 对损失函数进行累加
98     train_loss += loss.item() * b_x.size(0)
99     # 如果预测正确，则准确度train_corrects加1
100     train_corrects += torch.sum(pre_lab == b_y.data)
101     # 当前用于训练的样本数量
102     train_num += b_x.size(0)
```

模型评估

- ❑ 损失函数（判断是否过拟合）
- ❑ 预测正确率

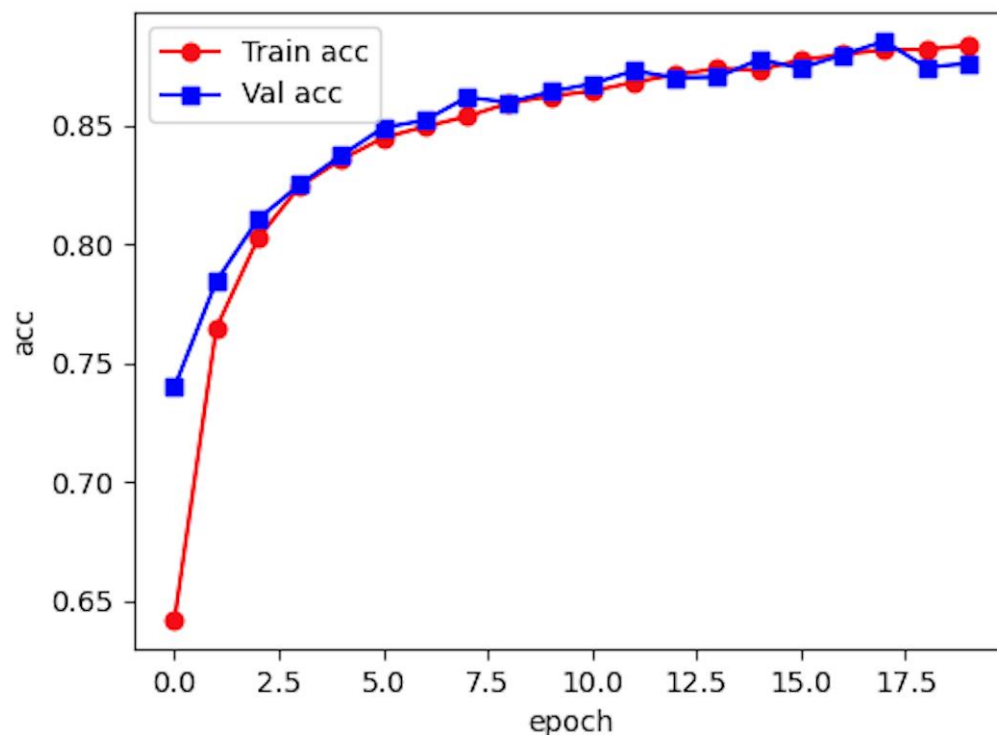
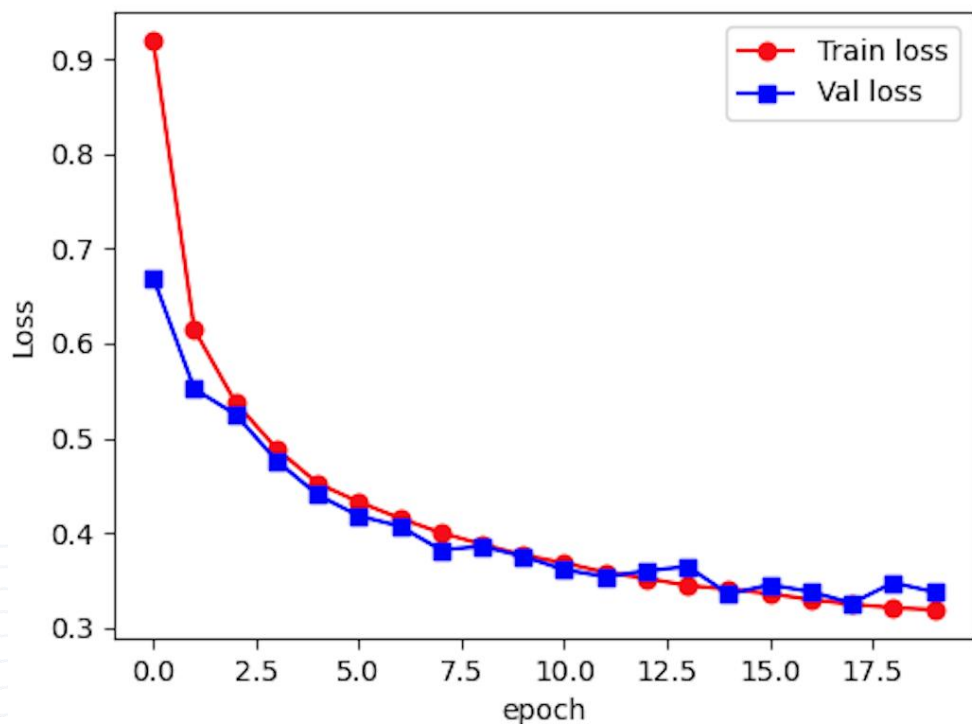
```
104 # 测试集验证模型
105 for step, (b_x, b_y) in enumerate(val_data_loader):
106     # 将特征和标签放入到验证设备中
107     b_x, b_y = b_x.to(device), b_y.to(device)
108     # 设置模型为评估模式
109     model.eval()
110     # 前向传播过程，输入为一个batch，输出为一个batch中对应的预测
111     output = model(b_x)
112     # 查找每一行中最大值对应的行标
113     pre_lab = torch.argmax(output, dim=1)
114     # 计算每一个batch的损失函数
115     loss = criterion(output, b_y)
116
117     # 对损失函数进行累加
118     val_loss += loss.item() * b_x.size(0)
119     # 如果预测正确，则准确度train_corrects加1
120     val_corrects += torch.sum(pre_lab == b_y.data)
121     # 当前用于验证的样本数量
122     val_num += b_x.size(0)
```

结果可视化 (LeNet)

19 train loss:0.3188 train acc: 0.8839

19 val loss:0.3376 val acc: 0.8763

训练和验证耗费的时间2m44s

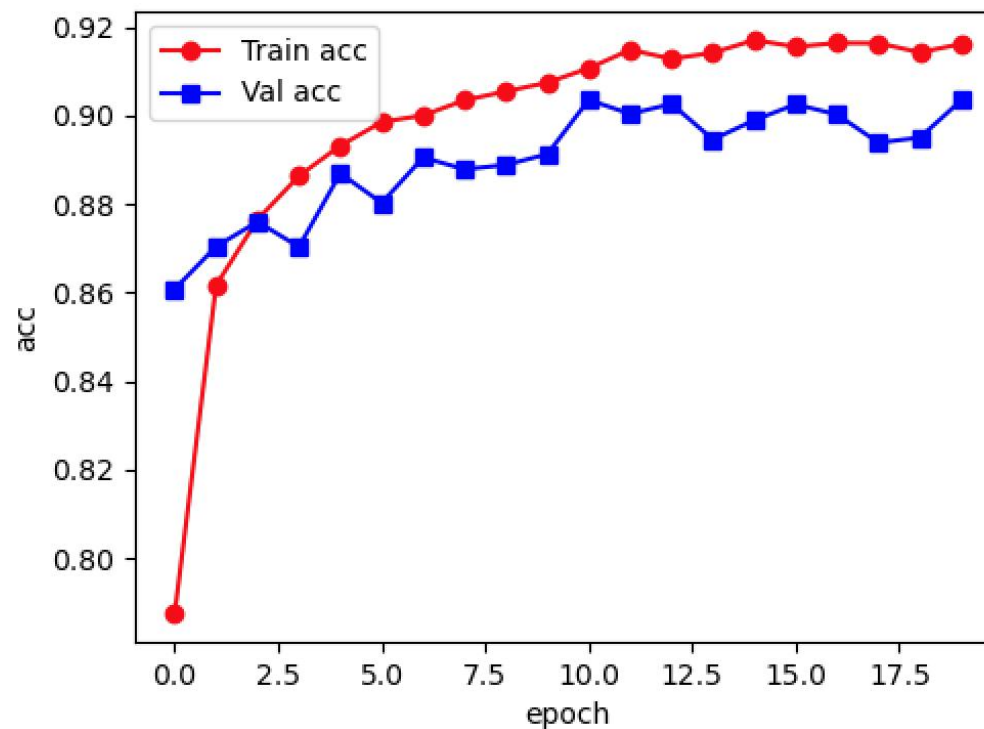
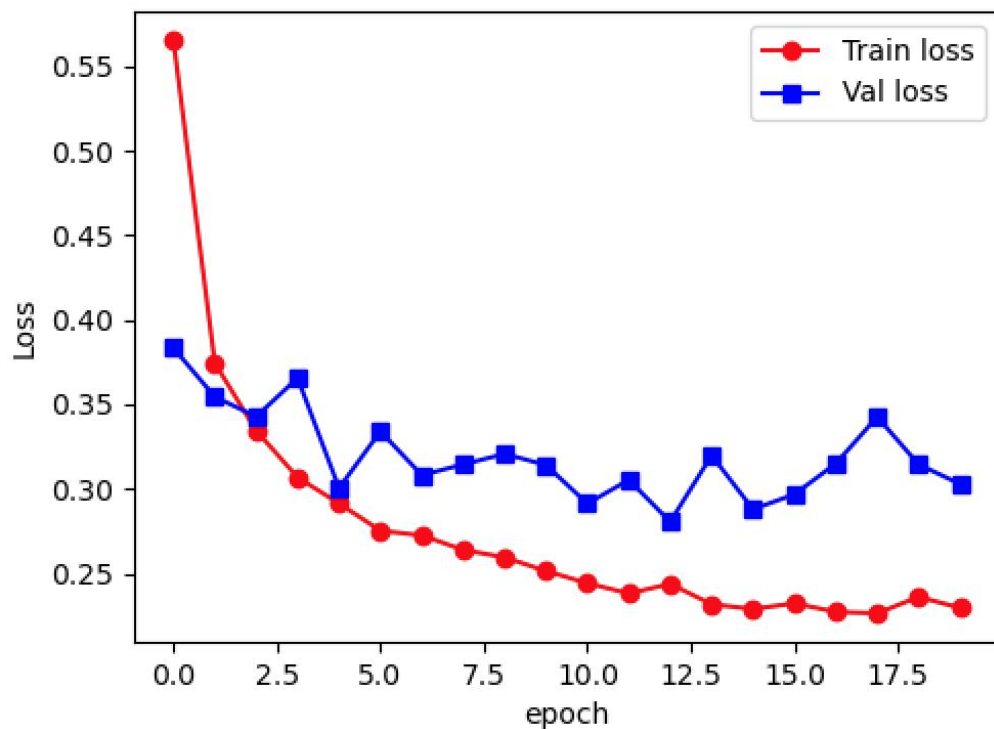


实验结果 (AlexNet)

19 train loss:0.2296 train acc: 0.9161

19 val loss:0.3029 val acc: 0.9034

训练和验证耗费的时间307m12s





谢谢!