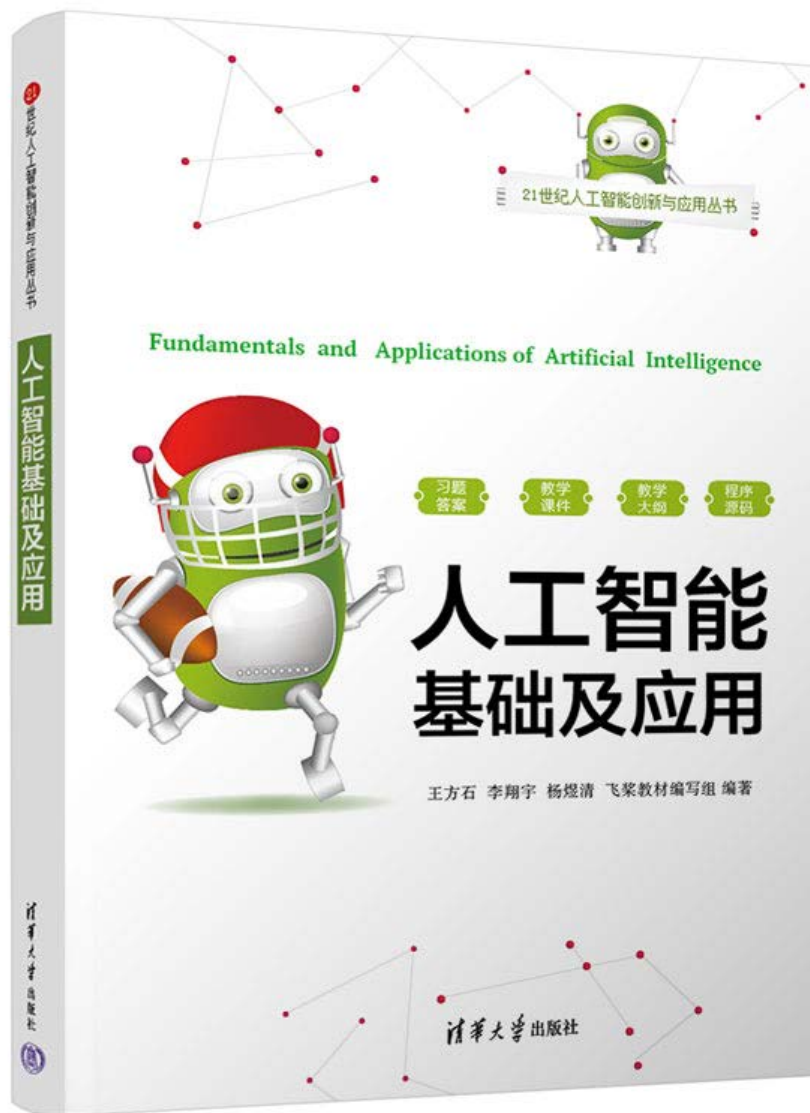


人工智能基础



北京交通大学 软件学院
王方石

Email: fshwang@bjtu.edu.cn

第5章 人工神经网络

5.1 人工神经网络的发展历程

5.2 感知机与神经网络

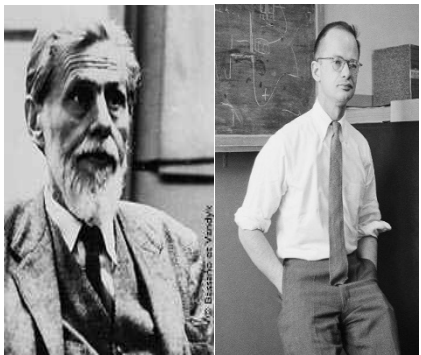
5.3 BP神经网络及其学习算法

5.4 卷积神经网络

本章学习目标

- ◆ 了解人工神经网络的发展历程。
- ◆ 理解感知机的工作原理和局限性。
- ◆ 掌握BP神经网络的结构和BP学习算法。
- ◆ 掌握卷积神经网络的结构和运算过程。

5.1 人工神经网络的发展历史



Warren McCulloch
Walter Pitts (MP模型)
神经网络开山之作

1957



Marvin Lee Minsky
Seymour Papert
指出“感知机”缺陷



Rumelhart & Hinton & Williams
反向传播



Geoffery Hinton
深度学习



吴恩达
谷歌大脑

2009

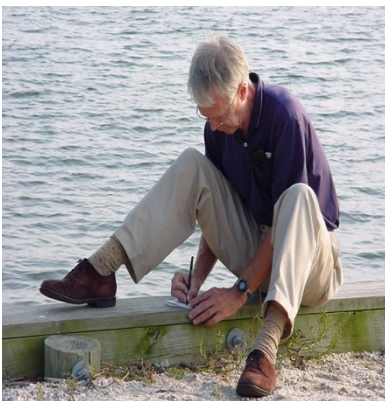
1943

Frank Rosenblatt
“感知机”模型



1969

1982
John Hopfield
霍普菲尔德网络



1986

2006

李飞飞
ImageNet



2012年

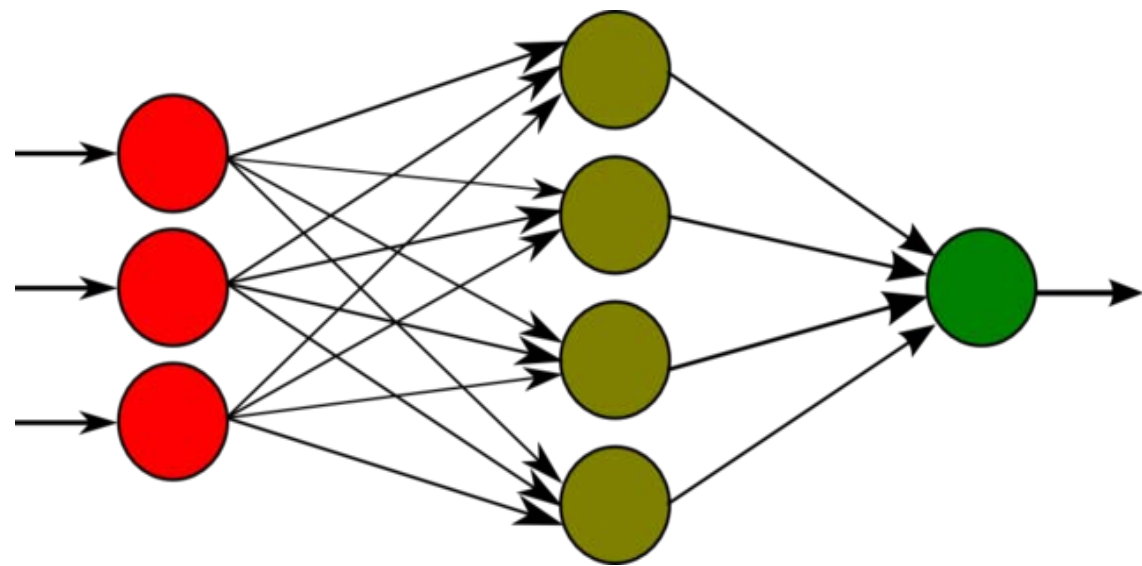
5.2 人工神经网络

Artificial Neural Networks (ANN)

- ◆ 人工神经网络是受构成**动物大脑的生物神经网络**的启发而建立的计算系统。
- ◆ 人工神经网络，简称神经网络或类神经网络，是**模拟人脑或生物神经网络的学习机制**而建立起来的一种运算模型，它由大量简单的**信息处理单元**按照一定拓扑结构相互连接而组成人工网络。



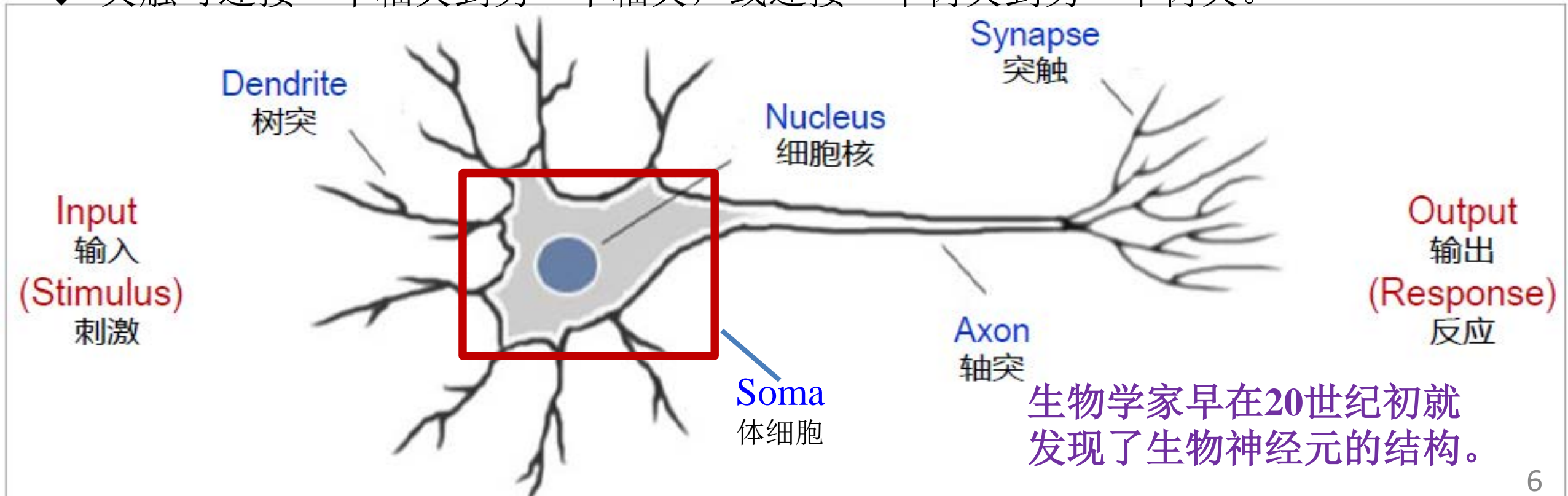
Animal brains



Artificial neural networks

生物神经元 (Biological Neuron)

- ◆ 神经元也称为神经细胞，通过**突触**（专门用于连接的组织）与其他细胞通信。
- ◆ 一个神经元由一个**细胞体**（体细胞）、多个**树突**和一条长的**轴突**组成。
- ◆ 神经元通过**树突**和**体细胞**接收信号，并沿**轴突**发出信号。
- ◆ 在大多数**突触**处，信号从一个神经元的**轴突**传递到另一个神经元的**树突**。
- ◆ 突触可连接一个轴突到另一个轴突，或连接一个树突到另一个树突。



生物神经元的两种状态

- ◆ **兴奋状态**：当传入的神经冲动使细胞膜电位超过**动作电位**的阈值，则细胞进入兴奋状态，产生神经冲动，并由轴突输出；
- ◆ **抑制状态**：当传入的神经冲动使细胞膜电位下降, 低于动作电位的阈值，则细胞进入抑制状态，没有神经冲动输出。
- ◆ **学习与遗忘**：由于神经元结构的可塑性，突触的传递作用可增强和减弱。

5.2.2 神经元数学模型—MP模型

◆1943年, **McCulloch** (麦克洛奇) 与 **Pitts** (皮兹) 提出了**M-P** 模型.

Neurologist & anatomist

mathematician

◆MP模型可以接收多路输入信号。假设一个神经元同时接收的 n 个输入信号用向量

$\mathbf{X} = (x_1, x_2, x_3, \dots, x_n)$ 表示, 则所有输入信号的线性组合, 称为该神经元的净输入

(Net Input), 记为 $u \in \mathcal{R}$, 计算公式如

◆其中, w_1, \dots, w_n 为该神经元各个输入信号的权值, $\theta \in \mathcal{R}$ 为偏置。

◆净输入 u 会被函数 $f(x)$ 转换为输出值 y , 该函数称为激活函数 (Activation Function)。

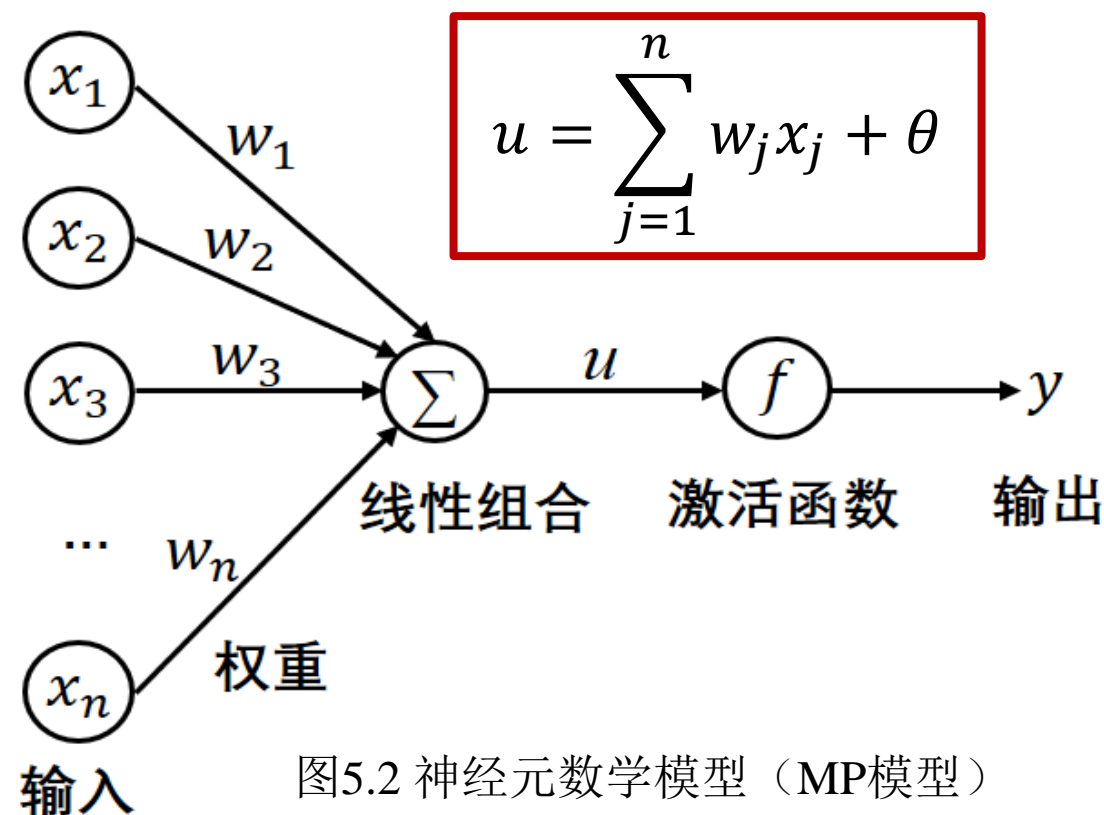
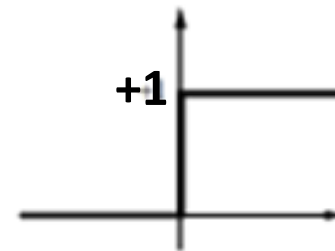


图5.2 神经元数学模型 (MP模型)

5.2.2 神经元数学模型—MP模型

◆ 在MP模型中，激活函数采用**非线性的阶跃函数** $f(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$



- ◆ **阶跃函数的缺陷**：不连续、不平滑，因为它在0点处的导数是无穷大，除了0点处之外，导数都是0，这意味着：**若学习算法采用基于梯度的优化方法，是不可行的。**
- ◆ 在MP模型中**引入激活函数的目的**是：用于模拟生物神经元的工作机制，当电位高于一个设定的阈值时，则进入兴奋状态，输出信号；否则进入抑制状态，不输出信号。

5.2.2 神经元数学模型—MP模型

- ◆MP模型中的输入和输出数据只能是二值化数据0或1，而且网络中的**权重、阈值等参数都需要人为设置，无法从数据中学习得到。**
- ◆MP模型的激活函数只是一个简单的阶跃函数。
- ◆MP模型只能处理一些**简单的分类任务**，例如线性二分类问题，但**无法解决线性不可分问题。**
- ◆虽然MP模型没有学习机制，但它开创了用电子装置模仿人脑结构和功能的新途径。

5.2.3 感知机

- ◆ 由一个神经元构成的神经网络称为**感知机**，也称为**单层神经网络**。
- ◆ 1957年提出的感知机是**第一个工程实现**的人工神经网络，可以运行感知机学习算法来训练模型。
- ◆ 感知机是一种简单的**非线性神经网络**，是人工神经网络的基础。
- ◆ 感知机只包含输入层和输出层，其输入层可以包含多个单元，而输出层只有一个单元。
- ◆ 感知机通过采用**有监督学习**来逐步增强模式分类的能力，达到学习的目的。

感知机与MP模型的异同点

- ◆ 从本质上说，感知机与MP模型没有太大的区别，两者的结构相同，计算过程也相同，都能完成线性可分的二分类任务，也都无法解决线性不可分问题。
- ◆ 但MP模型与感知机的不同之处在于：
 - ① MP模型没有“学习”的机制，其权值 w 和偏置 θ 都是人为设定的；而感知机引入了“学习”的概念，权值 w 和偏置 θ 是通过学习得到的，并非人为设置的，在一定程度上模拟了人脑的“学习”功能，这也是两者最大的区别。
 - ② 两者采用的激活函数不同，MP模型采用阶跃函数作为激活函数，而感知机通常采用sigmoid函数作为激活函数。

1. sigmoid函数

◆ sigmoid函数，现在专指logistic函数。

◆ **特点**：具有平滑性、连续性、单调性和渐近性，且连续可导。

◆ 2012年前，sigmoid是最常用的非线性激活函数，其输出值为(0,1)，表示概率或输入的归一化。

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

◆ sigmoid 函数的**求导公式**如下：

$$\begin{aligned}\frac{\partial \sigma(x)}{\partial x} &= -\frac{1}{(1+e^{-x})^2} \cdot e^{-x} \cdot (-1) = \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right) = \sigma(x)(1 - \sigma(x))\end{aligned}$$

◆ sigmoid函数的**优点**：

- 平滑、易于求导，其导数可直接用函数的输出计算，简单高效。
- sigmoid函数很好地解释了神经元在受到刺激的情况下是否被激活和向后传递的情景。

当取值接近0时，几乎没有被激活；当取值接近1时，几乎完全被激活。

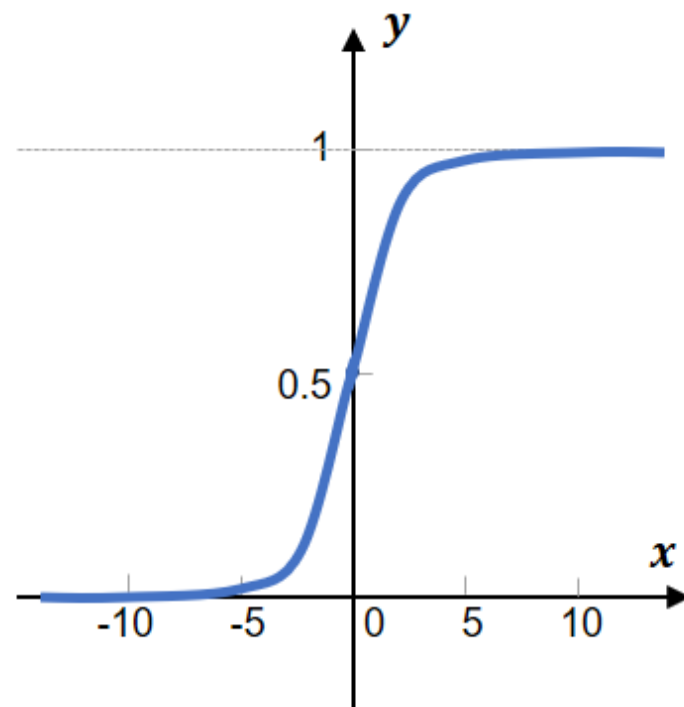


图5.3 sigmoid函数图像

Sigmoid函数的导数公式

$$\frac{\partial \sigma(x)}{\partial x} = -\frac{1}{(1+e^{-x})^2} \cdot e^{-x} \cdot (-1) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right) = \sigma(x)(1 - \sigma(x))$$

1. sigmoid函数

◆ sigmoid函数的缺点：

- (1) 当输入的绝对值大于某个阈值时，会快速进入饱和状态（即函数值趋于1或-1，不再有显著的变化，梯度趋于0），会出现梯度消失的情况，权重无法再更新，会导致算法收敛缓慢，甚至无法完成深层网络的训练。因此在一些现代的神经网络中，sigmoid函数逐渐被ReLU激活函数取代。
- (2) sigmoid函数公式中有幂函数，计算耗时长，在反向传播误差梯度时，求导运算涉及除法。
- (3) sigmoid函数的输出恒大于0，非零中心化，在多层神经网络中，可能会造成后面层神经元的输入发生偏置偏移，导致梯度下降变慢。

Softmax Function

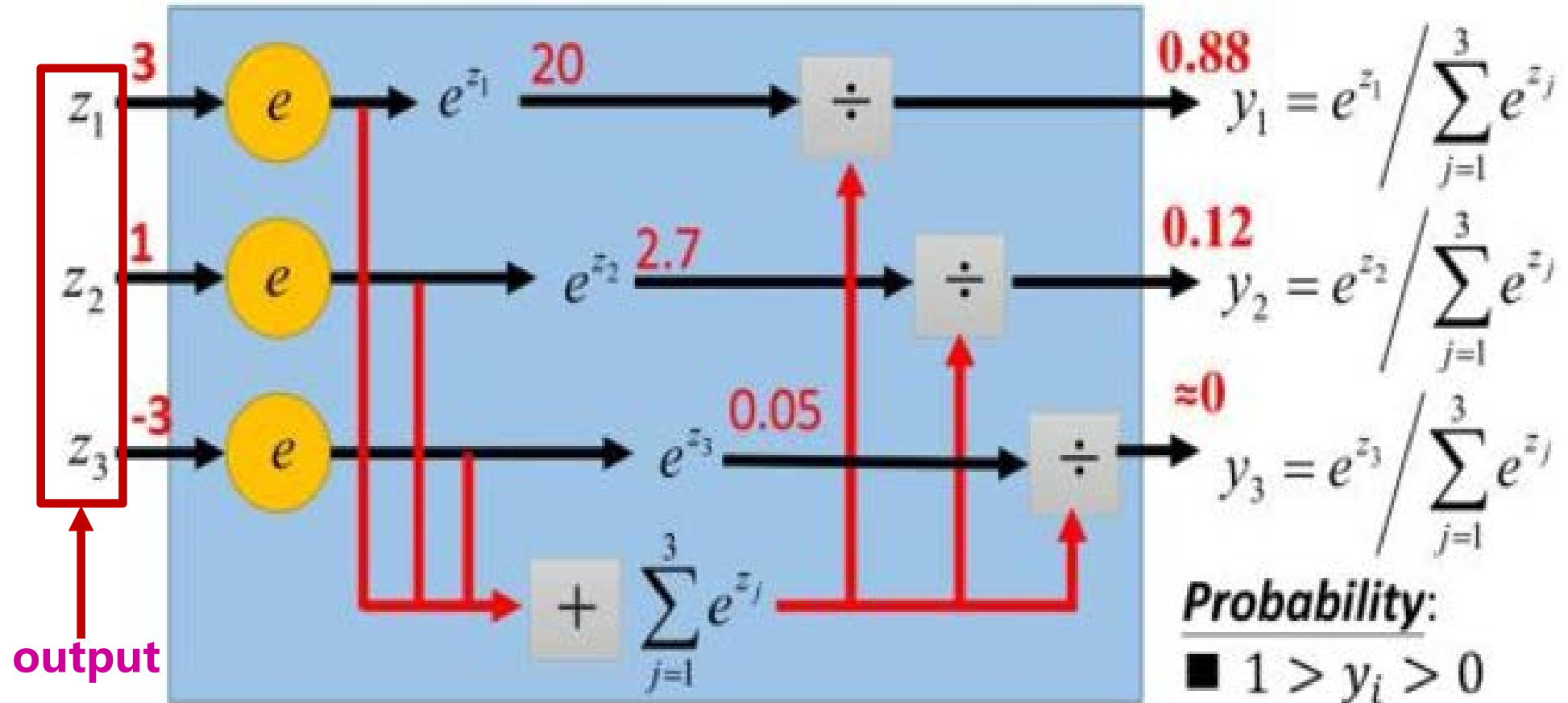
$$f(x_i) = \frac{e^{x_i}}{\sum_{k=1}^M e^{x_k}} \quad i=1, \dots, M \quad ; \quad M \text{ 为类别数}$$

- ◆ **sigmoid** 是 softmax 的特例。当类别数 $M=2$ 时, softmax 就是 sigmoid 。
- ◆ sigmoid 用于解决**二分类**问题, 而 softmax 用于解决**多分类**问题。
- ◆ 当分类数为2时, 全连接且不含隐藏层的神经网络, 就变为**logistic回归**。
- ◆ **softmax 的多类别间是互斥**的, 即一个输入只能被归为一类;
- ◆ 多个**logistic回归**也可实现多分类, 但输出的**类别并不互斥**, 如“苹果”这个词语既属于“水果”类, 也属于“食品”类别。

Softmax Function

Softmax layer as the output layer

$$20/(20+2.7+0.05)=0.879$$



2. tanh函数

- ◆ tanh函数是sigmoid函数的一个变形，称为**双曲正切函数**。
- ◆ tanh函数的值域为 $(-1,1)$ ，改进了sigmoid变化过于平缓的问题，且其输出是零中心化的，解决了sigmoid函数的偏置偏移问题。
- ◆ tanh函数的数学表达式： $y = f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$
- ◆ tanh可看作是在纵轴方向上放大到2倍并向下平移的sigmoid函数。
- ◆ tanh函数的导数公式：

$$\frac{\partial y}{\partial x} = - \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} = 1 - y^2$$

- ◆ tanh函数的**优点**：tanh 在线性区的梯度更大，能加快神经网络的收敛。
- ◆ tanh函数的**缺点**：其两端的梯度也趋于零，依旧存在梯度消失的问题，同时，幂运算也会导致计算耗时长。

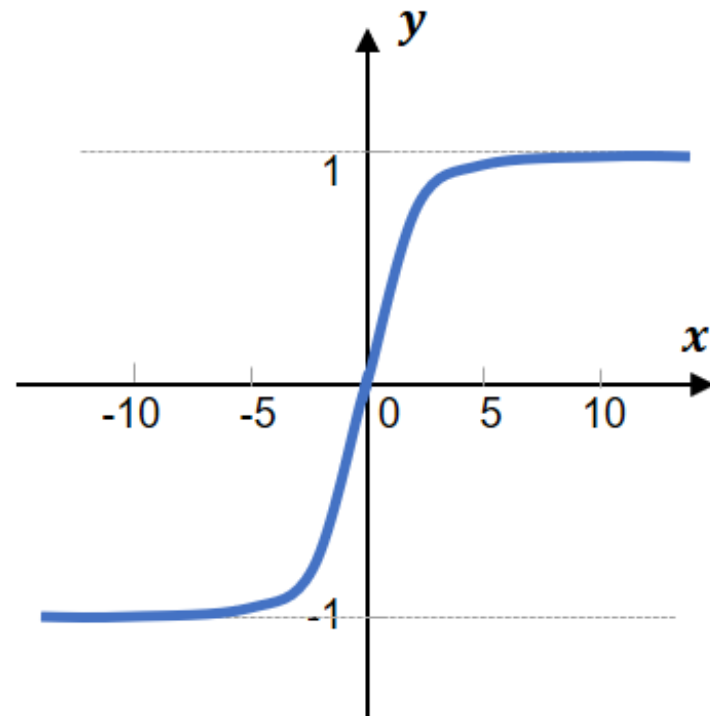
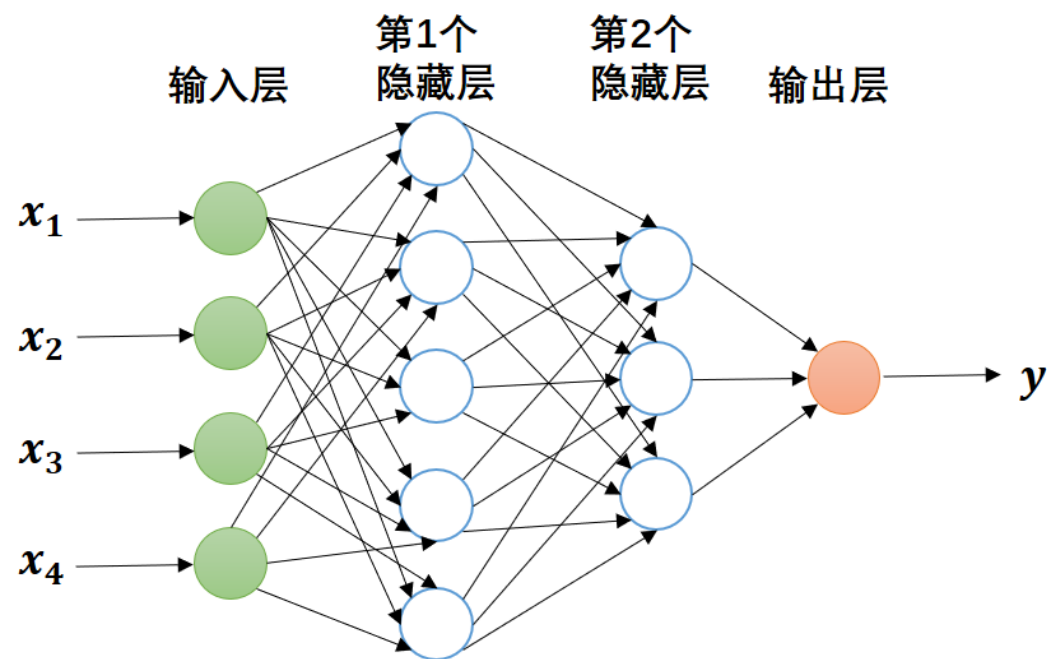


图5.4 tanh 函数图像

5.2.4 多层神经网络结构

- ◆ 单个人工神经元的结构简单，功能有限。
- ◆ 若想完成复杂的功能，就需要将许多人工神经元按照一定的拓扑结构相互连接在一起，相互传递信息，协调合作。
- ◆ 组成神经网络的所有神经元是分层排列的，一个神经网络包括输入层、**隐藏层**（也称为**隐层**、**隐含层**）和输出层。
- ◆ 每个网络只能有一个**输入层**和一个**输出层**，却可以有**0个或多个隐藏层**，每个隐藏层上可以有若干个神经元。
- ◆ 只有输入层与输出层的神经元可以与外界相连，外界无法直接接触及隐藏层，故而得名“**隐藏层**”。



5.2.4 多层神经网络结构

- ◆ 包含至少一个隐藏层的神经网络称为**多层神经网络**。
- ◆ 在包含神经元个数最多的一层，神经元的个数称为该**神经网络的宽度**。
- ◆ 除输入层外，其他层的层数称为**神经网络的深度**，即等于隐藏层个数加1（输出层）。
- ◆ **感知机**没有隐藏层，只有输入层和输出层，因此感知机的层数为1，称为**单层神经网络**。
- ◆ 人工神经网络的行为并非各个神经元行为的简单相加，而是具有学习能力的、行为复杂的非线性系统，既可以提取和表达样本的高维特征，又可以完成复杂的预测任务。

5.2.4 多层神经网络结构

- ◆ 多层神经网络的每个隐藏层后面都有一个非线性的激活函数。
- ◆ 这里激活函数的作用比感知机中作为激活函数的阶跃函数的作用要大得多，因为激活函数是对所有输入信号的线性组合结果进行非线性变换，而且多层神经网络就有多个激活函数。
- ◆ **激活函数最主要的作用**是向模型中加入非线性元素，用以解决非线性问题。
- ◆ 一般在同一个网络中使用同一种激活函数。

5.2.4 多层神经网络结构

- ◆ 根据**神经元之间的连接范围**，可以将多层人工神经网络分为**全连接神经网络**和**部分连接神经网络**。
- ◆ 若每个神经元与其相邻层的所有神经元都相连，这种结构的网络称为**全连接神经网络**。
- ◆ 若每个神经元只与相邻层上的部分神经元相连，则是**部分连接神经网络**。

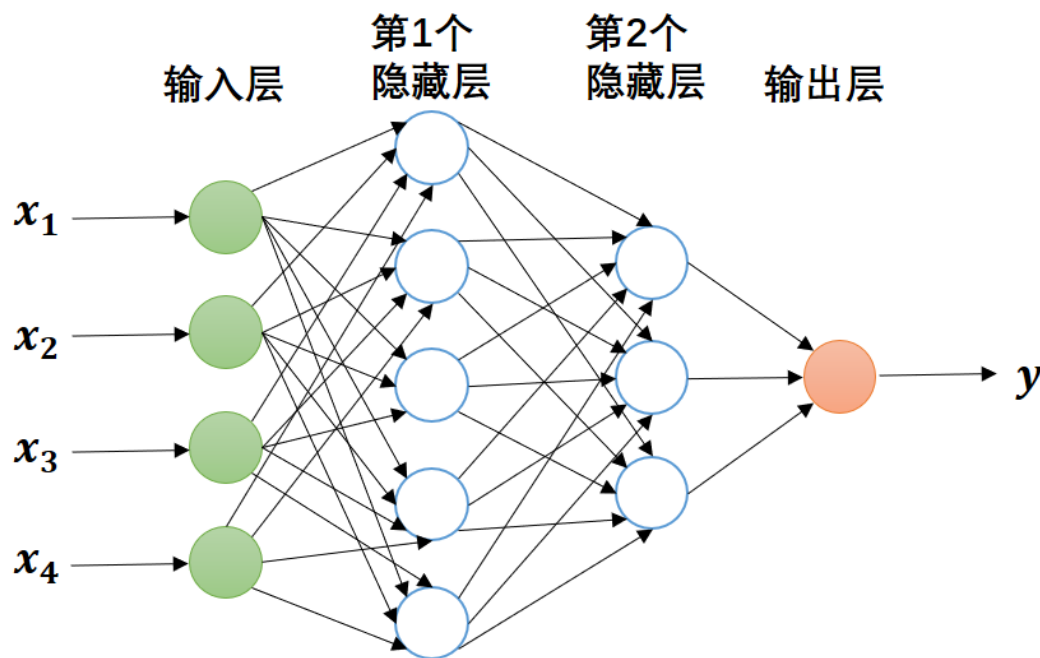


图5.5 多层前馈**全连接神经网络**

5.2.4 多层神经网络结构

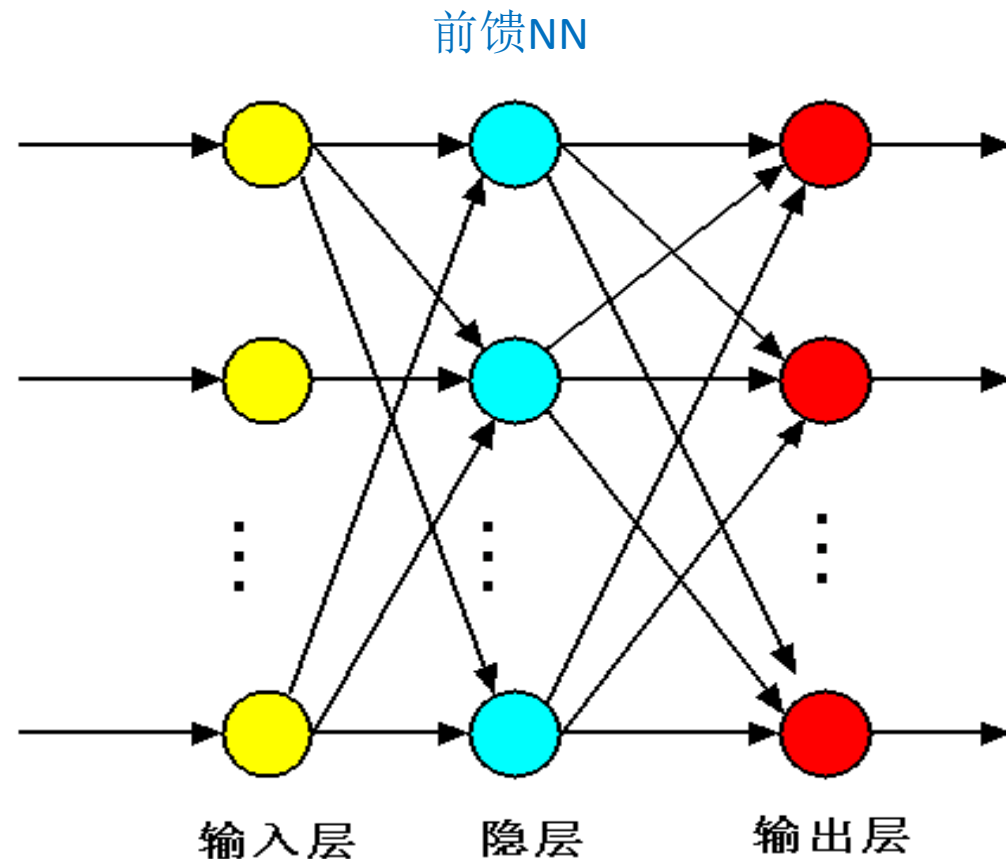
根据网络层之间的连接方式，又可以将多层人工神经网络分为前馈神经网络和反馈神经网络。

1. 前馈神经网络

- ◆ 前馈神经网络（Feedforward Neural Network）是一种多层神经网络，其中每个神经元只与其相邻层上的神经元相连，接收前一层的输出，并输出给下一层，即第 i 层神经元以第 $i-1$ 层神经元的输出作为输入，第 i 层神经元的输出作为第 $i+1$ 层神经元的输入。
- ◆ 同层的神经元之间没有连接。
- ◆ 整个网络中的信息是单向传递的，即只能按一个方向从输入层到输出层的方向传播，没有反向的信息传播，可以用一个有向无环图表示。

5.2.4 多层神经网络结构

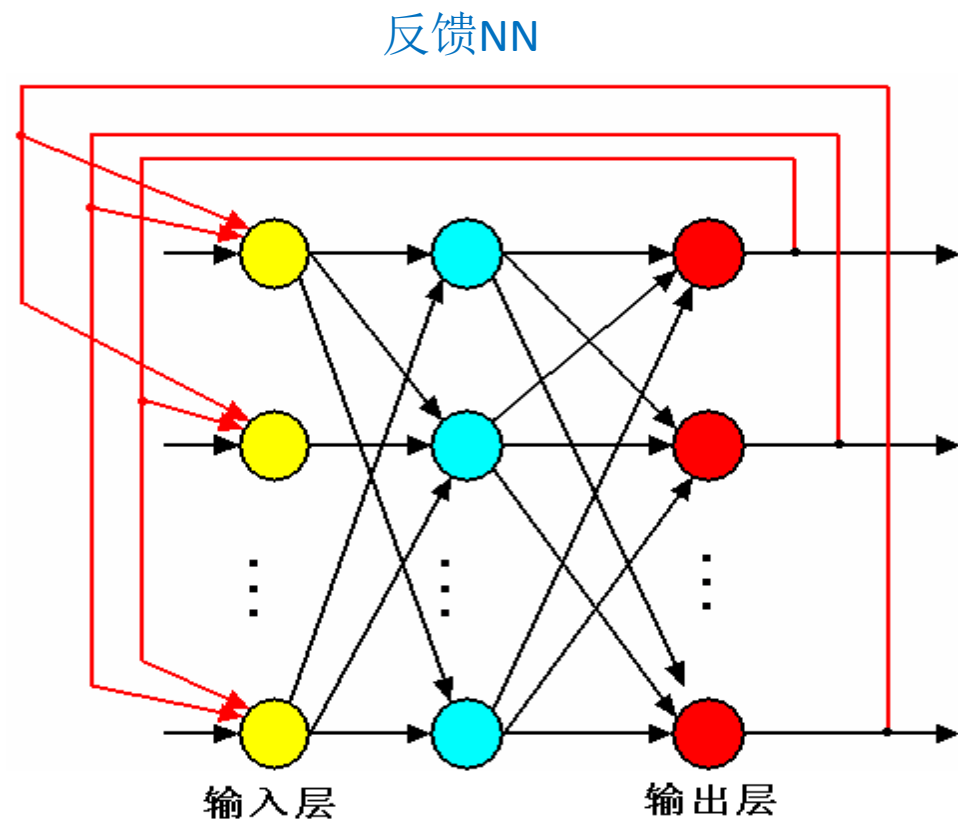
- ◆ 若前馈神经网络采用全连接方式，则称为**前馈全连接神经网络**。
- ◆ 前馈神经网络的**优点**：网络结构简单，易于实现。
- ◆ 前馈全连接神经网络的**缺点**是：当网络很深时，参数量巨大，计算量大，训练耗时。
- ◆ 前馈神经网络包括：**BP神经网络**和**卷积神经网络**。



5.2.4 多层神经网络结构

2. 反馈神经网络

- ◆ 反馈神经网络（Feedback Neural Network）是一种反馈动力学系统。
- ◆ 在这种网络中，有些神经元不但可以接收其前一层上神经元的信息，还可以接收来自于其后面层上神经元的信息。
- ◆ 神经元的连接可以形成有向循环。
- ◆ 反馈神经网络中的信息既可以单向传递，也可以双向传递，且神经元具有记忆功能，在不同时刻具有不同的状态，能建立网络的内部状态，可展现动态的时间特性。

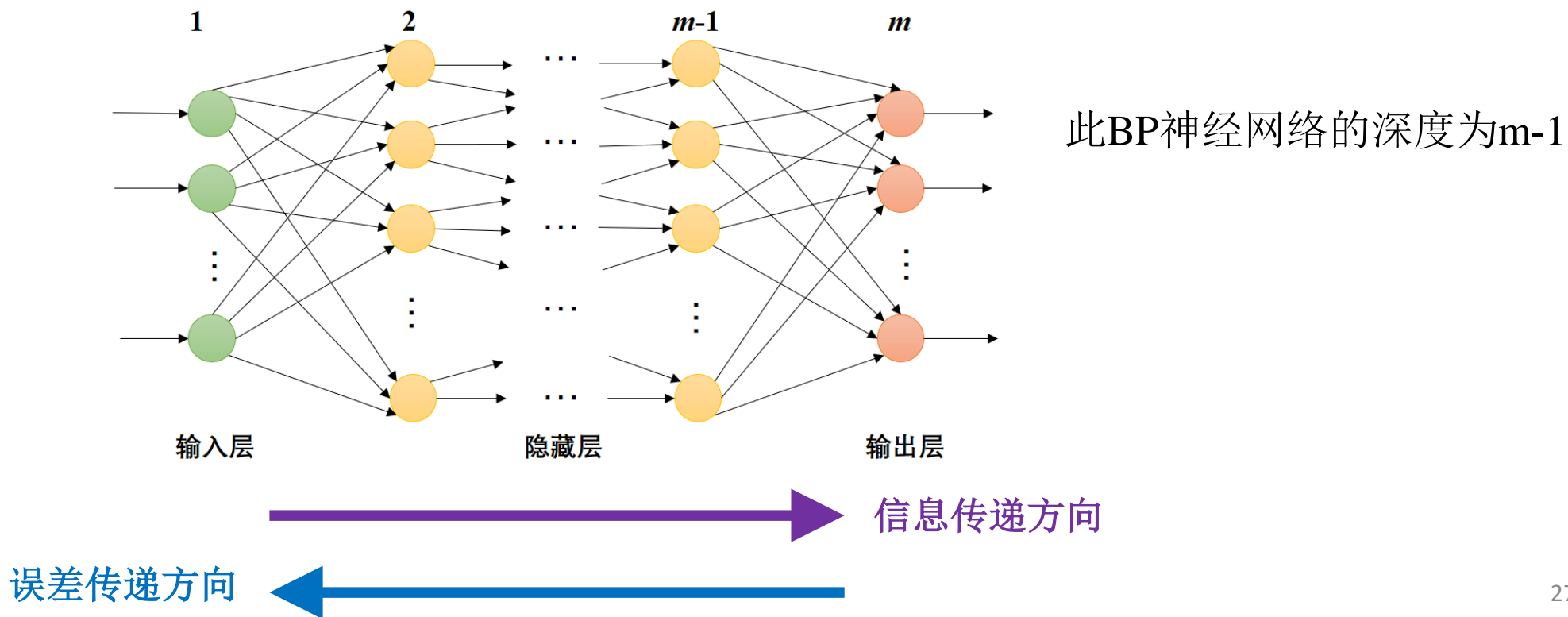


5.3 BP神经网络及其学习算法

- ◆ 多层前馈神经网络的表达能力比单层感知机要强得多。
- ◆ 要训练多层前馈神经网络，**单层感知机的学习算法是远远不够的**，需要更强大的学习算法。
- ◆ 1974年Werbos提出了BP算法；1986年辛顿等人又重新独立地提出了BP算法。
- ◆ 迄今为止，最成功的多层神经网络学习算法就是**反向传播（Back-Propagation, BP）算法**。
- ◆ BP算法能解决对参数逐一求偏导、计算效率低下的问题。
- ◆ 目前，学术界和产业界依然在用BP算法训练神经网络。

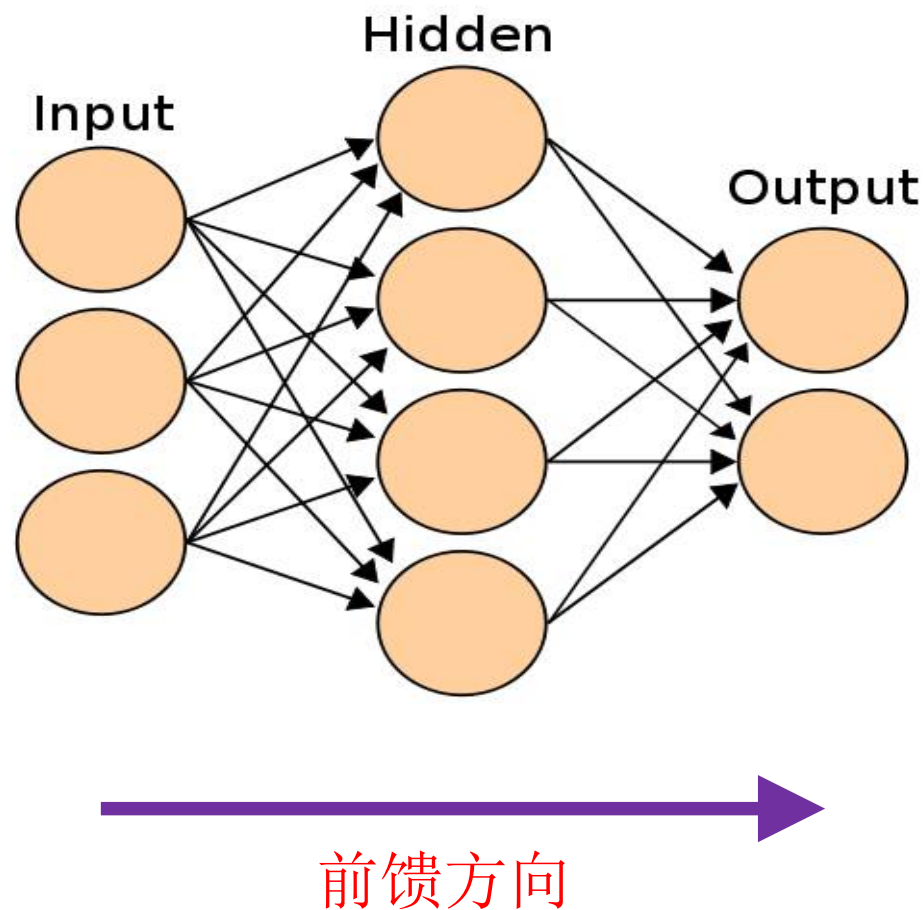
5.3.1 BP神经网络的结构

- ◆ 由于前馈神经网络**大多采用反向传播学习算法**来进行训练模型，故被称为**BP神经网络**。
- ◆ BP神经网络是一种**多层前馈神经网络**。
- ◆ 每个节点就是一个神经元，神经元之间带有箭头的连线表示信息传递的方向。



多层前馈神经网络

- ◆ 多层前馈神经网络又称为**多层前馈全连接网**。
- ◆ **多层**是指：除了输入层和输出层以外，还存在一个或者多个隐含层。
- ◆ **前馈**是指：外界信号从输入层，经由隐含层到达输出层，不存在信号的逆向传播。
- ◆ **全连接**是指：每层神经元与下一层神经元全部互相连接，**同层神经元之间不存在连接，也不存在跨层连接。**



5.3.1 BP神经网络的结构

假设神经网络中第 $k-1$ 层中有 p_{k-1} 个神经元，令

- ◆ y_j^{k-1} 表示第 $k-1$ 层中第 j 个神经元的输出，
- ◆ w_{ji}^k 表示第 $k-1$ 层的第 j 个神经元与第 k 层的第 i 个神经元之间的连接权值，
- ◆ θ^k 表示第 k 层的偏置，
- ◆ u_i^k 表示第 k 层的第 i 个神经元的净输入，
- ◆ 则 u_i^k 的计算表达式为： $u_i^k = \sum_{j=1}^{p_{k-1}} w_{ji}^k y_j^{k-1} + \theta^k$ ， $k = 2, \dots, m$
- ◆ 为使公式整齐，令 $w_{0i}^k = \theta^k$ ， $y_0^{k-1} = 1$ ，则有： $u_i^k = \sum_{j=0}^{p_{k-1}} w_{ji}^k y_j^{k-1}$ ， $k = 2, \dots, m$
- ◆ 令 $f(\bullet)$ 表示激活函数， y_i^k 表示第 k 层的第 i 个神经元的输出，其值为： $y_i^k = f(u_i^k)$

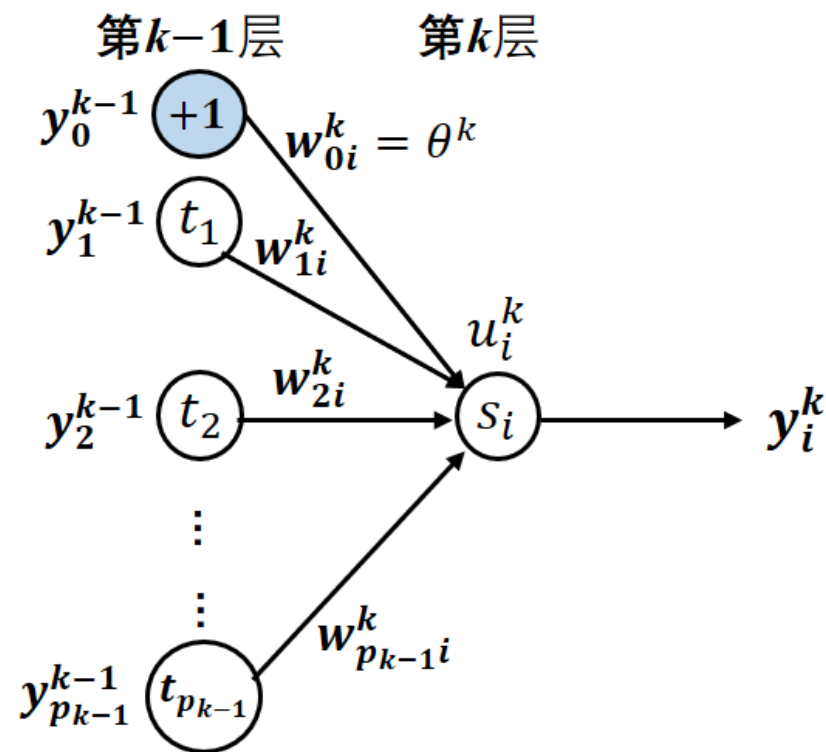


图5.7 BP网络中相邻两层的各变量之间关系

5.3.2 BP学习算法

- ◆ **神经网络学习**，也称为**神经网络训练**，是指利用训练数据集不断地修改神经网络的所有连接权值和偏置值，使神经网络的实际输出尽可能地逼近真实值。
- ◆ BP学习算法是一种有监督学习。
- ◆ 权重 w 和偏置 θ 是未知的，需要采用损失函数来指导学习算法来更新这些参数。
- ◆ 假设有 $m-1$ 层BP神经网络 F ，其输入数据 $\mathbf{X} = [x_1, x_2, \dots, x_{p_1}]^T$ (p_1 为输入层神经元的个数)，输出数据 $\mathbf{Y} = [y_1^m, y_2^m, \dots, y_{p_m}^m]^T$ (p_m 为输出层神经元的个数)。
- ◆ BP神经网络看成是一个从输入到输出的非线性映射 $\mathbf{Y}=\mathbf{F}(\mathbf{X})$ 。
- ◆ **要学习的网络参数**:
 - 由第 $k-1$ 层的第 j 个神经元到第 k 层的第 i 个神经元的**连接权值**
 $\{w_{ji}^k, k = 2, \dots, m; i = 1, \dots, p_k; j = 1, \dots, p_{k-1}\}$ (p_k 表示第 k 层中神经元的个数)
 - 第 k 层的**偏置** $\{\theta^k, k = 2, \dots, m\}$

5.3.2 BP学习算法

◆ 假设选择神经网络的实际输出与期望输出之间的**误差平方和**作为**损失函数**，其

数学表达式为：
$$E = \frac{1}{2} \sum_{j=1}^{p_m} (y_j - y_j^m)^2 \quad (\text{公式5.10})$$

- m 是输出层的层号，即输出层是第 m 层；
- p_m 是输出层中神经元的个数；
- y_j^m 是输出层第 j 个神经元的实际输出；
- y_j 是输出层第 j 个神经元的期望输出。

◆ **BP学习算法的目的**：使得目标函数（即损失函数） E 达到极小值。

◆ 损失函数的值越小，表示神经网的预测结果越接近真实值。

5.3.2 BP学习算法

BP算法的学习过程由**正向传播**和**反向传播**两个阶段组成，算法的实现过程如下。

第1步：用随机值初始化神经网络 F 的权重 \mathbf{W} 和偏置 $\boldsymbol{\theta}$ ；令 $l=1$ 。

第2步：若 $l \leq N$ （ N 为训练样本总数），向 F 输入第 l 个样本的 p_1 维向量 $\mathbf{X}_l = (x_{l1}, \dots, x_{lp_1})$ ，经过一次**正向传播**，得到预测结果 $\mathbf{Y}_l^m = F(\mathbf{X}_l)$ ， \mathbf{Y}_l^m 为一个 p_m 维的向量 $(y_{l1}^m, \dots, y_{lp_m}^m)$ ；否则，算法停止。

第3步：已知第 l 个样本的期望输出为 $\mathbf{Y}_l = (y_{l1}, \dots, y_{lp_m})$ ，**计算输出层的损失函数** E 的值。

$$E = \frac{1}{2} \sum_{j=1}^{p_m} (y_j - y_j^m)^2$$

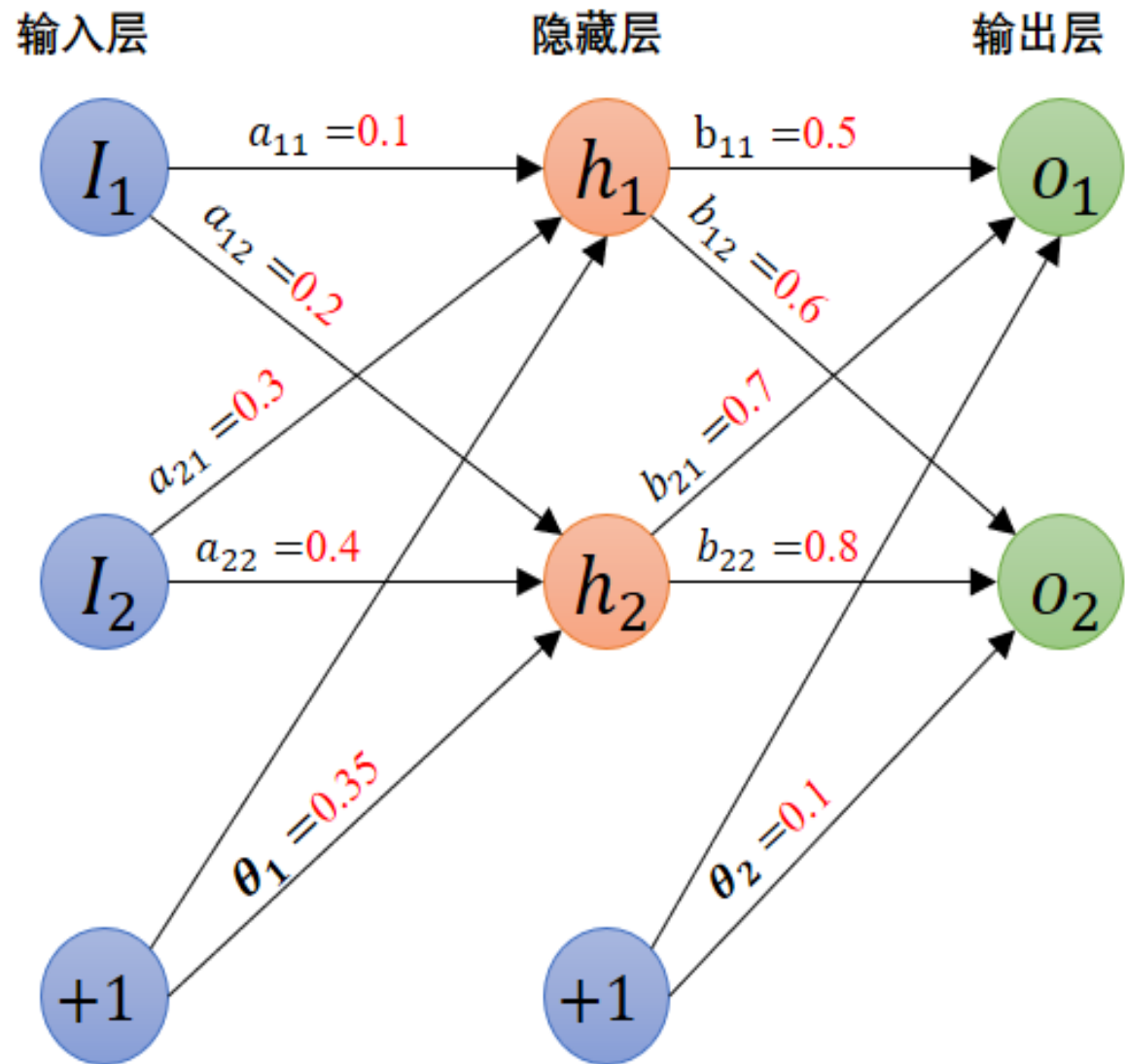
第4步：若 E 值小于设定的阈值，则 $l=l+1$ ，转向第2步；否则，进行**反向传播**，采用梯度下降法，依次计算每个权重和偏置的修正值，公式为：

$$\Delta w_{ji}^k = -\eta \frac{\partial E}{\partial w_{ji}^k}, \quad \Delta \theta^k = -\eta \frac{\partial E}{\partial \theta^k} \quad (\eta > 0) \quad (5.11), \quad \text{其中} \eta \text{是学习率, 一般小于} 0.5.$$

第5步：采用公式（5.11）计算出的修正值，依次**更新所有权重和偏置**；

第6步：转向第2步。

正向传播 (1) 假设输入样本为($x_1=0.05$, $x_2=0.1$), 期望输出值为($y_1=0.03$, $y_2=0.05$)
 令 net_j 表示编号为 j 的神经元的净输入, out_j 表示编号为 j 的神经元的输出,
 则 $out_{I_1} = x_1 = 0.05$, $out_{I_2} = x_2 = 0.1$ 。



(1) 针对**隐藏层中第一个神经元 h_1** :

$$net_{h_1} = out_{I_1} * a_{11} + out_{I_2} * a_{21} + \theta_1$$

$$= 0.05 * 0.1 + 0.1 * 0.3 + 0.35 = 0.385$$

$$out_{h_1} = \sigma(net_{h_1}) = \frac{1}{1 + e^{-net_{h_1}}} = 0.595$$

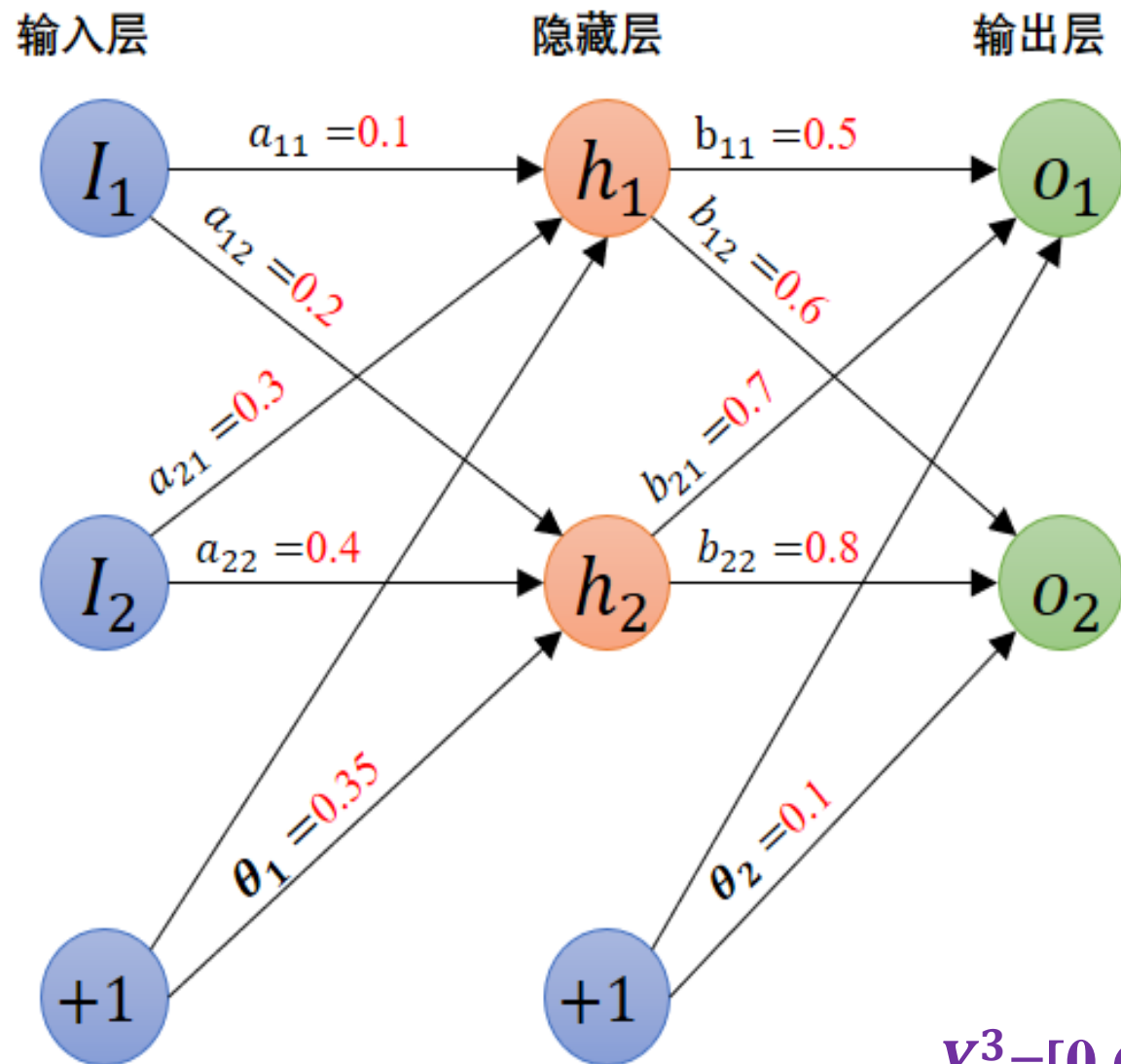
(2) 针对**隐藏层中第二个神经元 h_2** :

$$net_{h_2} = out_{I_1} * a_{12} + out_{I_2} * a_{22} + \theta_1$$

$$= 0.05 * 0.2 + 0.1 * 0.4 + 0.35 = 0.4$$

$$out_{h_2} = \sigma(net_{h_2}) = \frac{1}{1 + e^{-net_{h_2}}} = 0.599$$

正向传播 (2)



(3) 输出层中两个神经元 O_1 和 O_2 的净输入值分别如下:

$$\begin{aligned} net_{O_1} &= out_{h_1} * b_{11} + out_{h_2} * b_{21} + \theta_2 \\ &= 0.595 * 0.5 + 0.599 * 0.7 + 0.1 = 0.817 \end{aligned}$$

$$\begin{aligned} net_{O_2} &= out_{h_1} * b_{12} + out_{h_2} * b_{22} + \theta_2 \\ &= 0.595 * 0.6 + 0.599 * 0.8 + 0.1 = 0.936 \end{aligned}$$

(4) O_1 和 O_2 的输出分别为:

$$y_1^3 = out_{O_1} = \frac{1}{1 + e^{-net_{O_1}}} = 0.694$$

$$y_2^3 = out_{O_2} = \frac{1}{1 + e^{-net_{O_2}}} = 0.718$$

$Y^3 = [0.694, 0.718]$, 与期望输出 $Y = [0.03, 0.05]$ 相差较大。

反向传播 (1)

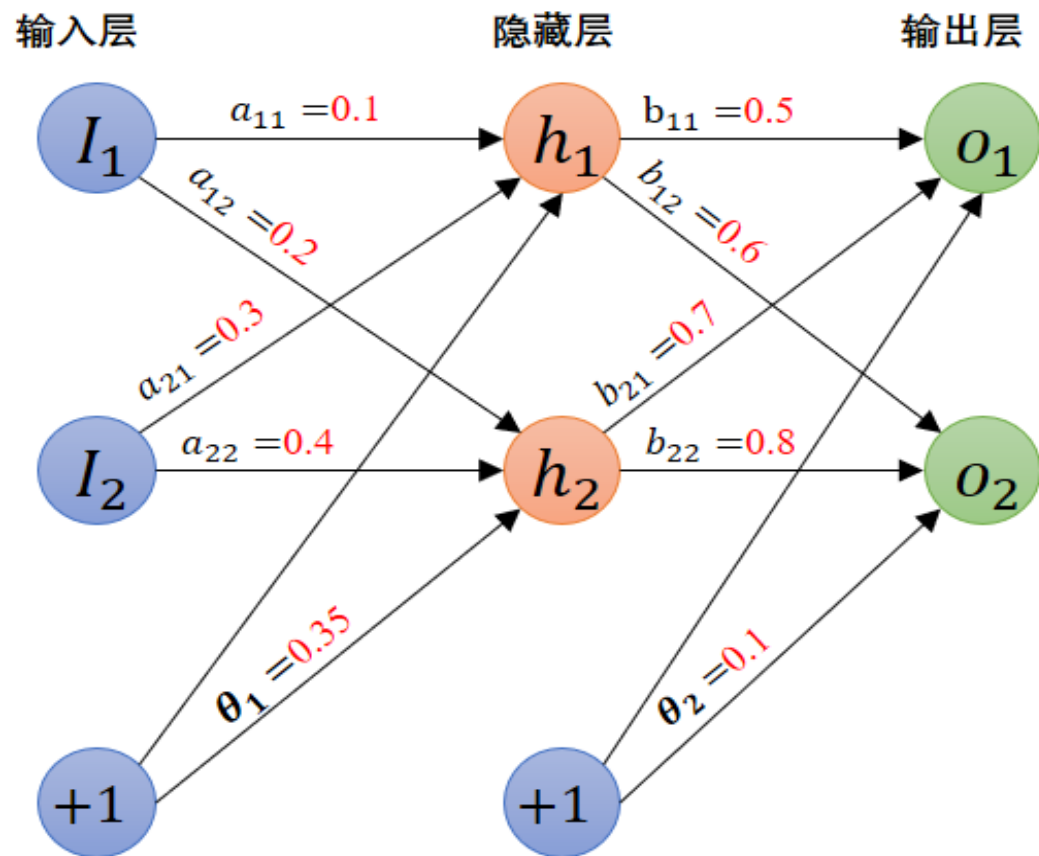
误差平方和作为损失函数:

$$E = \frac{1}{2} \sum_{j=1}^{p_m} (y_j - y_j^m)^2$$

$$\Delta w_{ji}^k = -\eta \frac{\partial E}{\partial w_{ji}^k} = -\eta d_i^{k+1} y_j^k \quad (\eta > 0) \quad (\text{公式1})$$

$$d_i^m = y_i^m (1 - y_i^m) (y_i^m - y_i) \quad (\text{公式2})$$

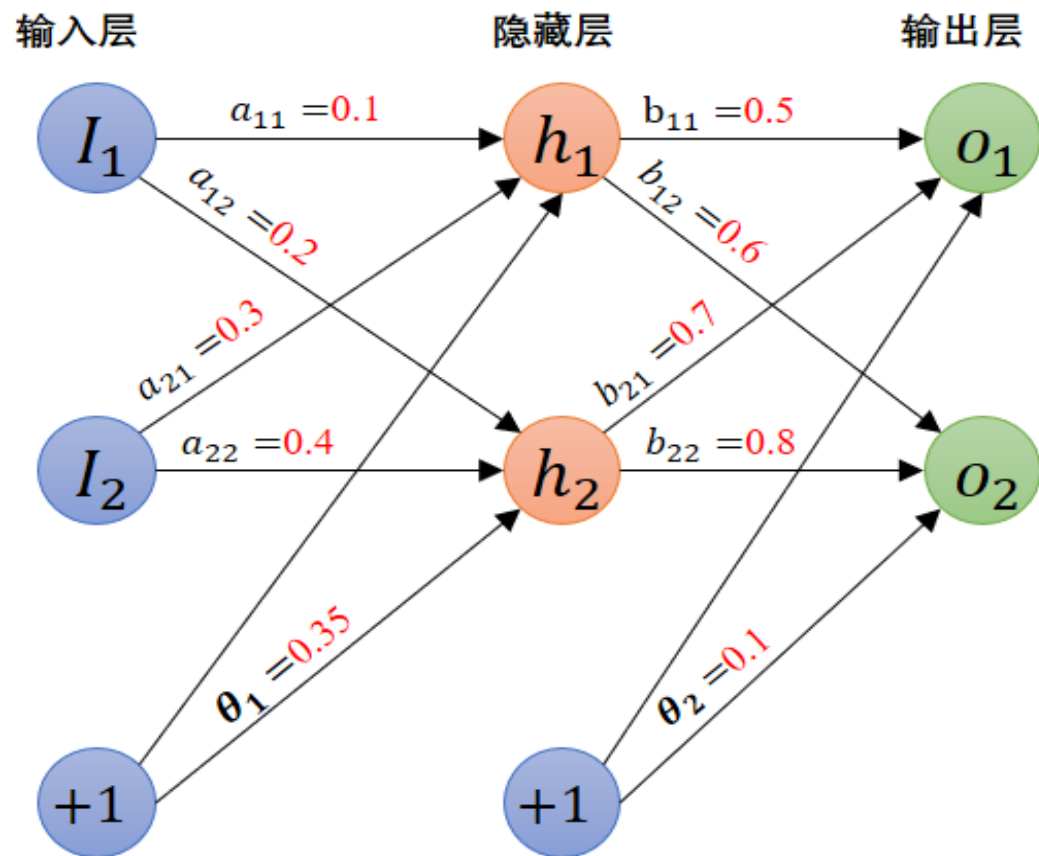
$$d_i^k = y_i^k (1 - y_i^k) \sum_{j=1} d_j^{k+1} w_{ji}^{k+1} \quad (k = m-1, \dots, 2,) \quad (\text{公式3})$$



其中, k 表示网络层的层号; d_i^{k+1} 表示第 $k+1$ 层的误差信号, y_i^k 表示第 k 层第 i 个神经元的输出。

注意: d_i^{k+1} 不是误差 Δw_{ji}^k , 两者差了一个第 k 层第 j 个神经元的输出项 y_j^k , 见公式1。

反向传播 (2)



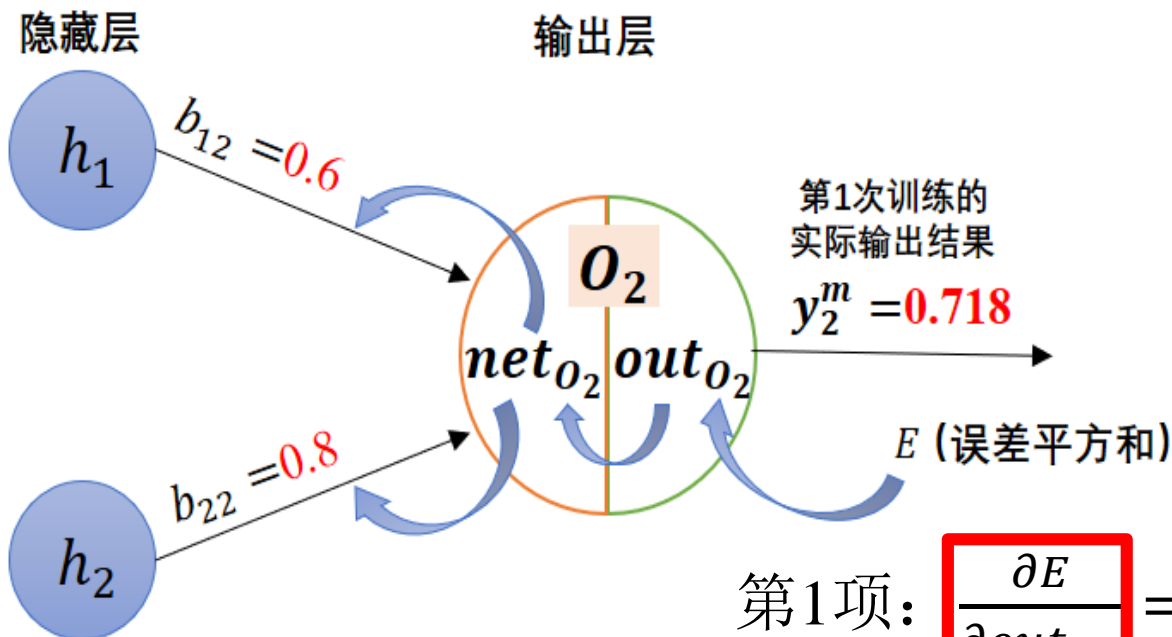
损失函数:

$$\begin{aligned} E &= \frac{1}{2} \sum_{j=1}^{p_m} (y_j - y_j^m)^2 \\ &= \frac{1}{2} [(y_1 - y_1^3)^2 + (y_2 - y_2^3)^2] \\ &= \frac{1}{2} [(0.03 - 0.694)^2 + (0.05 - 0.718)^2] = 0.444 \end{aligned}$$

以 b_{22} 为例, 计算权重修正量 Δb_{22} 。按照链式法则, 计算总误差 E 对 b_{22} 的偏导:

$$\Delta b_{22} = \frac{\partial E}{\partial b_{22}} = \frac{\partial E}{\partial out_{o_2}} * \frac{\partial out_{o_2}}{\partial net_{o_2}} * \frac{\partial net_{o_2}}{\partial b_{22}}$$

反向传播 (3)



$$E = \frac{1}{2} \sum_{j=1}^2 (y_j - y_j^3)^2 = 0.444$$

$$\Delta b_{22} = \frac{\partial E}{\partial b_{22}} = \frac{\partial E}{\partial out_{o_2}} * \frac{\partial out_{o_2}}{\partial net_{o_2}} * \frac{\partial net_{o_2}}{\partial b_{22}}$$

设定 $\eta=0.5$, 则有:

$$b_{22}^{new} = b_{22} - \eta * \Delta b_{22} = 0.8 - 0.5 * 0.081 = 0.76$$

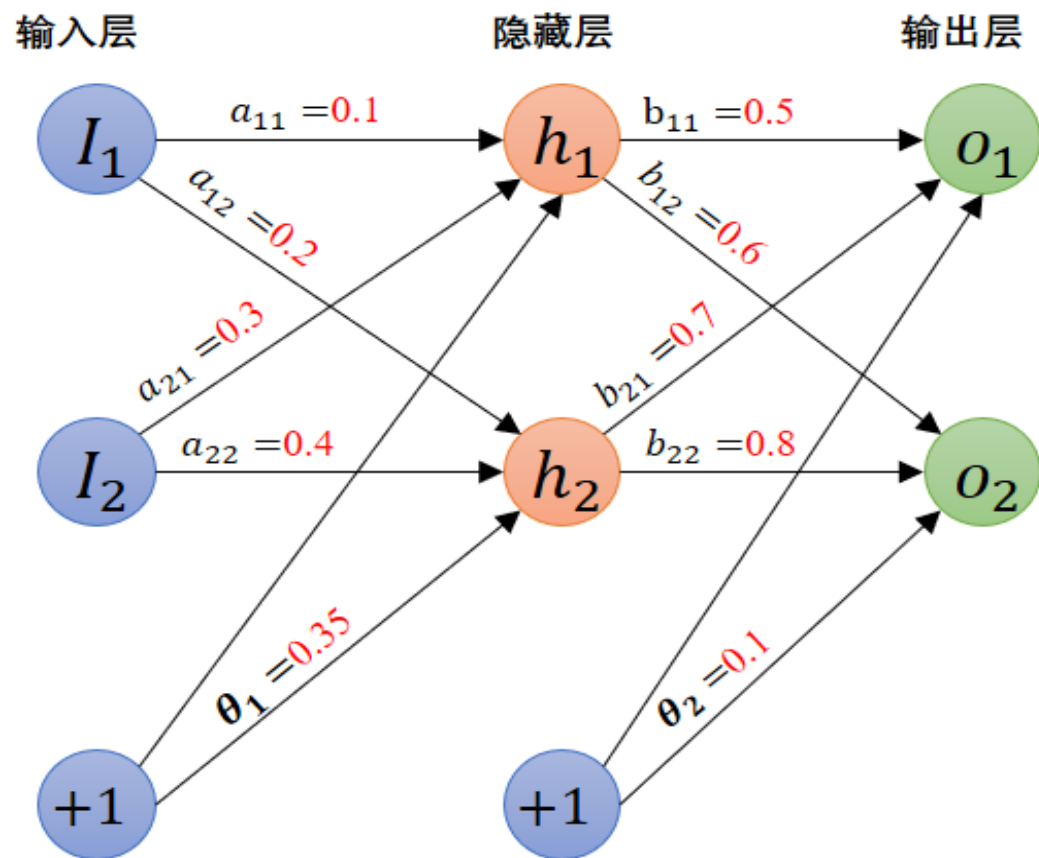
第1项: $\frac{\partial E}{\partial out_{o_2}} = 2 * \frac{1}{2} * (y_2 - y_2^3) * (-1) = 0.718 - 0.05 = 0.668$

第2项: $\frac{\partial out_{o_2}}{\partial net_{o_2}} = \frac{\partial}{\partial net_{o_2}} \left(\frac{1}{1+e^{-net_{o_2}}} \right) = \sigma(net_{o_2})(1 - \sigma(net_{o_2}))$
 $= out_{o_2} * (1 - out_{o_2}) = 0.718 * (1 - 0.718) = 0.202$

第3项: $\frac{\partial net_{o_2}}{\partial b_{22}} = \frac{\partial}{\partial b_{22}} (out_{h_1} * b_{12} + out_{h_2} * b_{22} + \theta_2) = out_{h_2} = 0.599$

则有: $\Delta b_{22} = \frac{\partial E}{\partial b_{22}} = 0.668 * 0.202 * 0.599 = 0.081$

反向传播 (4) 同理可得: $b_{11}^{new} = 0.458$, $b_{12}^{new} = 0.560$, $b_{21}^{new} = 0.658$



计算 θ_2 的修正量 $\Delta\theta_2$ 。由于每一层的偏置项对于该层所有神经元的误差都有影响，所以先用总误差对该层的每个神经元求偏导后，再求和，得到 $\Delta\theta_2$:

$$\Delta\theta_2 = \frac{\partial E}{\partial \theta_2} = \sum_{i=1}^{p_m} \left(\frac{\partial E}{\partial out_{O_i}} * \frac{\partial out_{O_i}}{\partial net_{O_i}} * \frac{\partial net_{O_i}}{\partial \theta_2} \right)$$

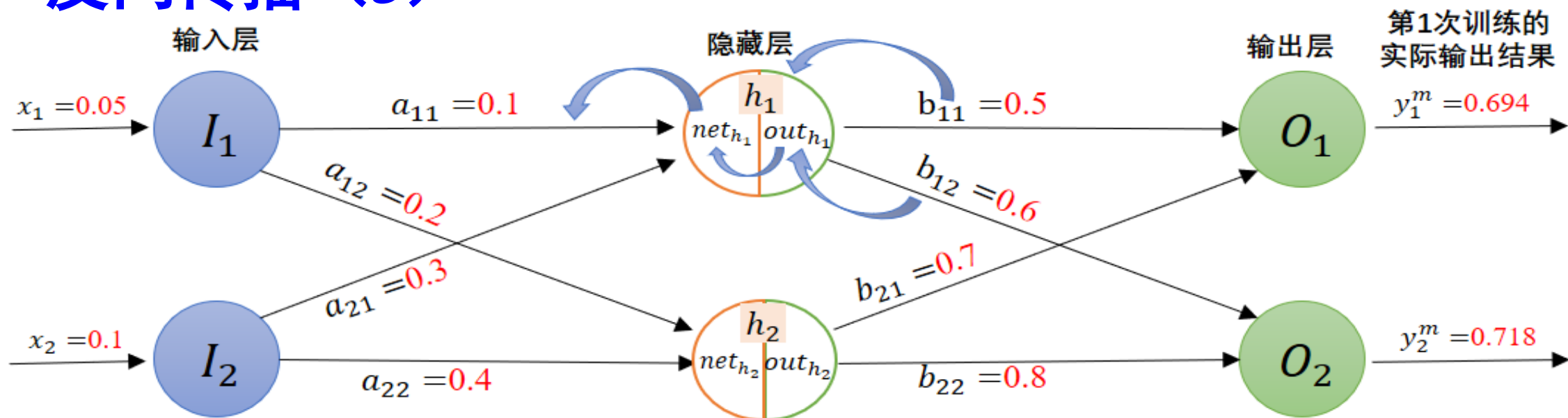
$=1$

由于最后一项求导后的值为1，故有:

$$\begin{aligned} \Delta\theta_2 &= \frac{\partial E}{\partial \theta_2} = \sum_{i=1}^{p_m} \left(\frac{\partial E}{\partial out_{O_i}} * \frac{\partial out_{O_i}}{\partial net_{O_i}} \right) = 0.174 \\ &= (0.694 - 0.03) * 0.212 + (0.718 - 0.05) * 0.202 \end{aligned}$$

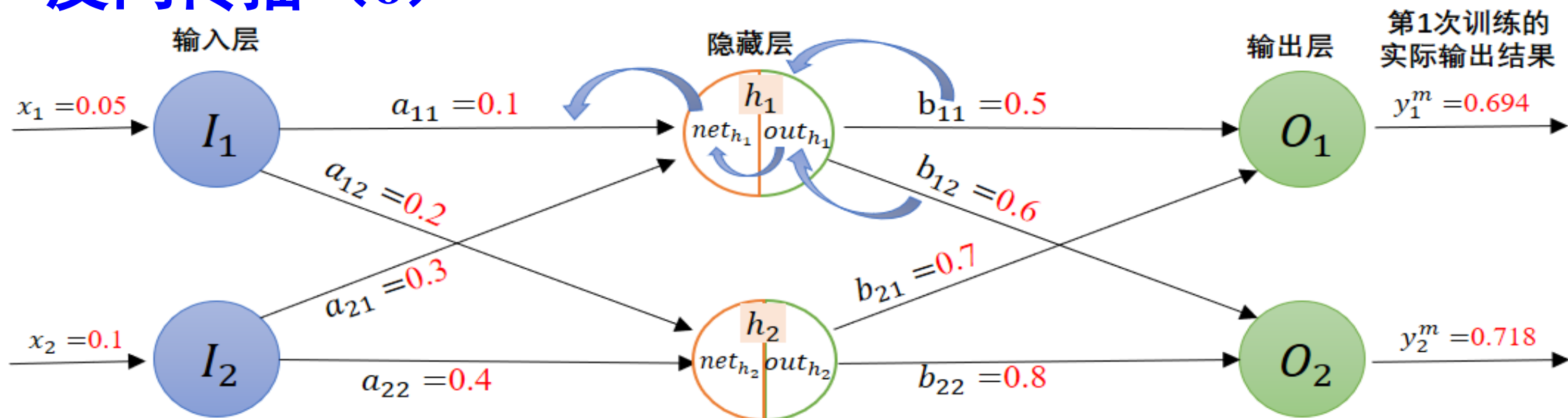
更新后的偏置项如下: $\theta_2^{new} = \theta_2 - \eta * \Delta\theta_2 = 0.1 - 0.5 * 0.174 = 0.913$

反向传播 (5)



- ◆ 从隐藏层到输入层，共有5个参数需要更新，分别为 a_{11} , a_{12} , a_{21} , a_{22} , θ_1
- ◆ 这些学习参数的更新方法与“从输出层向隐藏层反向更新学习参数”类似，
- ◆ 但有区别：在计算 Δb_{22} 时，需要考虑所有能影响 b_{22} 值的神经元，显然 b_{22} 的值只受到来自神经元 O_2 的误差影响，因为 O_2 是输出层神经元，后面没有其他网络层了；
- ◆ 但在计算 Δa_{11} 时， a_{11} 的值不仅要受到神经元 h_1 反向传过来的误差的影响，还会受到输出层神经元 O_1 和 O_2 传过来的误差的影响。

反向传播 (6)



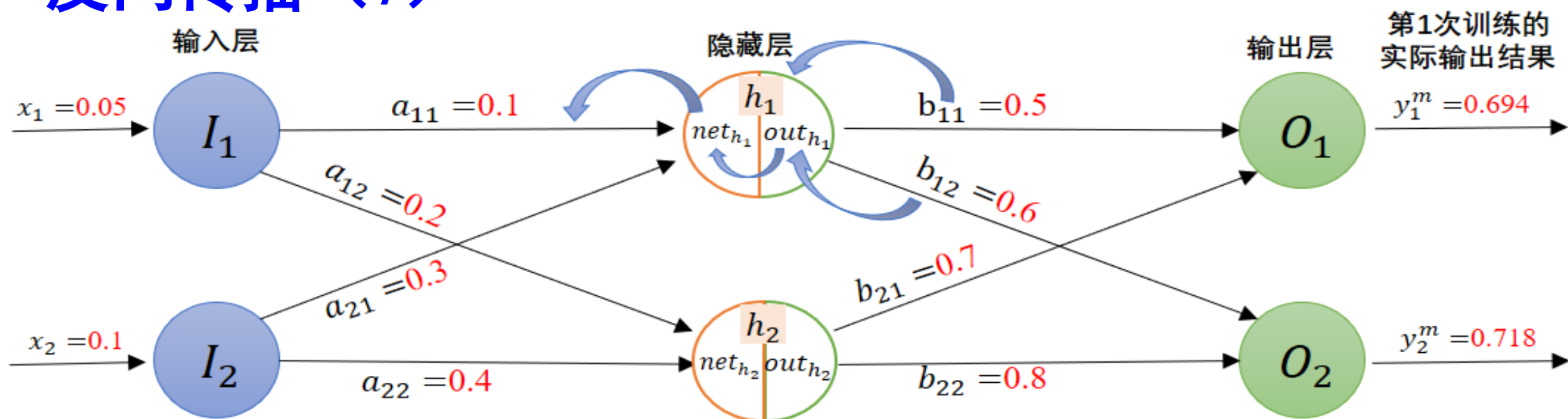
计算修正量 Δa_{11} :

$$\Delta a_{11} = \frac{\partial E}{\partial a_{11}} = \frac{\partial E}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial a_{11}} \quad (\text{公式4})$$

公式4中第1项: $\frac{\partial E}{\partial out_{h_1}} = \frac{\partial E}{\partial out_{O_1}} * \frac{\partial out_{O_1}}{\partial net_{O_1}} * \frac{\partial net_{O_1}}{\partial out_{h_1}} + \frac{\partial E}{\partial out_{O_2}} * \frac{\partial out_{O_2}}{\partial net_{O_2}} * \frac{\partial net_{O_2}}{\partial out_{h_1}}$

其余4项的值在“从输出层向隐藏层反向更新学习参数”的权值更新中均已有计算值。

反向传播 (7)



已知: $net_{O_2} = out_{h_1} * b_{12} + out_{h_2} * b_{22} + \theta_2$, 有 $\frac{\partial net_{O_2}}{\partial out_{h_1}} = b_{12} = 0.6$;

已知: $net_{O_1} = out_{h_1} * b_{11} + out_{h_2} * b_{21} + \theta_2$, 有 $\frac{\partial net_{O_1}}{\partial out_{h_1}} = b_{11} = 0.5$;

反向传播 (8)

所以公式4中第1项为:

$$\begin{aligned}\frac{\partial E}{\partial out_{h_1}} &= [(y_1 - y_1^3) * (-1)] * [y_1^3(1 - y_1^3)] * b_{11} + (y_2 - y_2^3) * (-1) * [y_2^3(1 - y_2^3)] * b_{12} \\ &= -(0.03 - 0.694) * [0.694 * (1 - 0.694)] * 0.5 \\ &\quad -(0.05 - 0.718) * [0.718 * (1 - 0.718)] * 0.6 \\ &= 0.664 * 0.212 * 0.5 + 0.668 * 0.202 * 0.6 = 0.151\end{aligned}$$

公式4的第2项: $\frac{\partial out_{h_1}}{\partial net_{h_1}} = \sigma(net_{h_1})(1 - \sigma(net_{h_1})) = 0.595 * (1 - 0.595) = 0.241$

公式4的第3项: $\frac{\partial net_{h_1}}{\partial a_{11}} = \frac{\partial}{\partial a_{11}} (out_{I_1} * a_{11} + out_{I_2} * a_{21} + \theta_1) = out_{I_1} = x_1 = 0.05$

综上所述, $\Delta a_{11} = \frac{\partial E}{\partial a_{11}} = 0.151 * 0.241 * 0.05 = 0.002$

同理可得, $a_{12}^{new} = 0.199$, $a_{21}^{new} = 0.298$, $a_{22}^{new} = 0.398$, $\theta_1^{new} = 0.307$ 。

偏置的更新方法与第(2)步中的方法相同, 可求得: $\theta_1^{new} = 0.307$ 。

5.3.2 BP学习算法

- ◆ **第1个训练样本** ($x_1=0.05$, $x_2=0.1$)，目标输出为 $[0.03, 0.05]$ ，实际输出为 $[0.694, 0.718]$ ，总误差为**0.444**；
- ◆ 完成第1轮参数更新；
- ◆ 继续将**第1个训练样本** ($x_1=0.05$, $x_2=0.1$) 输入到已更新参数的神经网络中，进行第二次训练，得到实际输出为 $[0.667, 0.693]$ ，总误差为**0.44356**；
- ◆ 可见，总误差在逐渐减小，随着迭代次数的增加，输出值会越来越接近目标值，直到总误差小于设定的阈值，**再输入下一个训练样本**，继续训练，直到输入所有训练样本训练完毕为止。

BP神经网络及学习算法的局限性

- (1) BP 学习算法是有监督学习，需要大量带标签的训练数据。
- (2) BP神经网络中的参数量大，收敛速度慢，需要较长的训练时间，学习效率低。
- (3) BP 学习算法采用梯度下降法更新学习参数，容易陷入局部极值，从而找不到全局最优解。
- (4) 尚无理论指导如何选择网络隐藏层的层数和神经元的个数，一般是根据经验或通过反复实验确定。因此，网络往往存在很大的冗余性，在一定程度上也增加了网络学习的负担。

以前长期困扰研究者的，但目前已基本解决的问题：

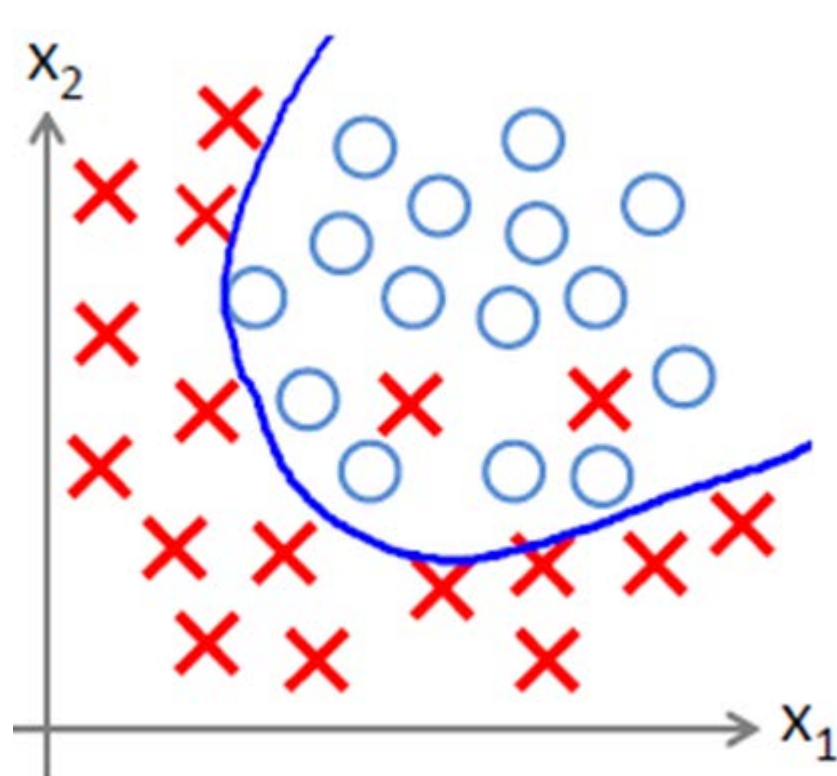
- (5) 过拟合问题
- (6) “梯度消失”或“梯度爆炸”问题

(1) 过拟合问题

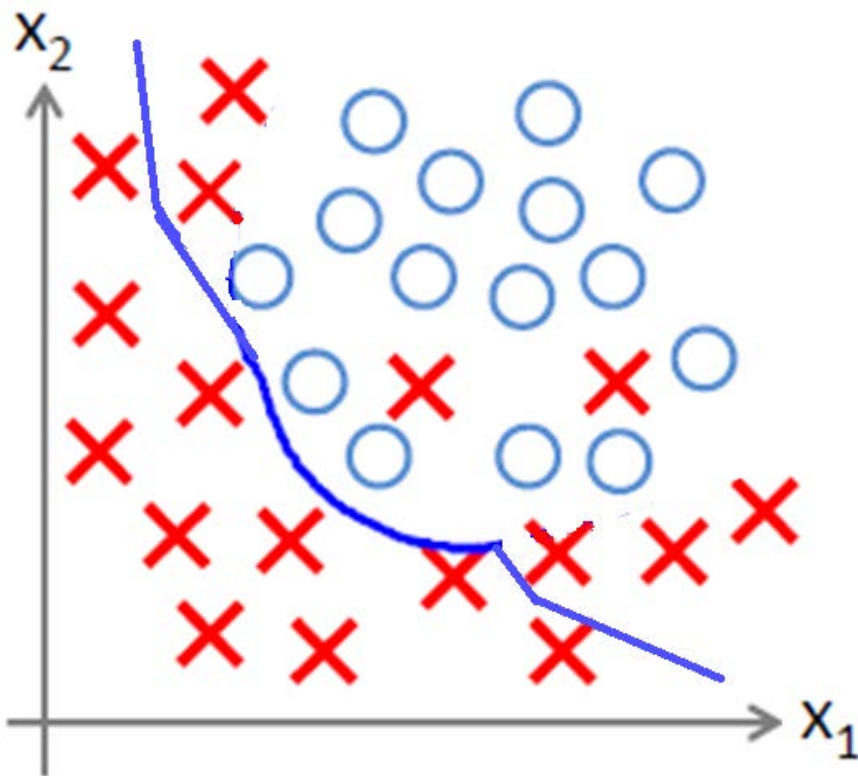
过拟合问题 (Overfitting Problem)

- ◆ 一般情况下，随着训练能力的提高，预测能力会得到提高。
- ◆ 但这种趋势不是固定的，有一个极限，当达到此极限时，随着训练能力的提高，预测能力反而会下降，称为“**过拟合**”现象，即**训练误差**变小，**测试误差**也随之减小，然而减小到某个值后，**测试误差**却反而开始增大。
- ◆ 在训练**数据不够多**或**模型过于复杂**时，常会导致模型对训练集**过拟合**。
- ◆ **预测能力**也称**泛化能力**或**推广能力**，而**训练能力**也称**逼近能力**或**学习能力**。

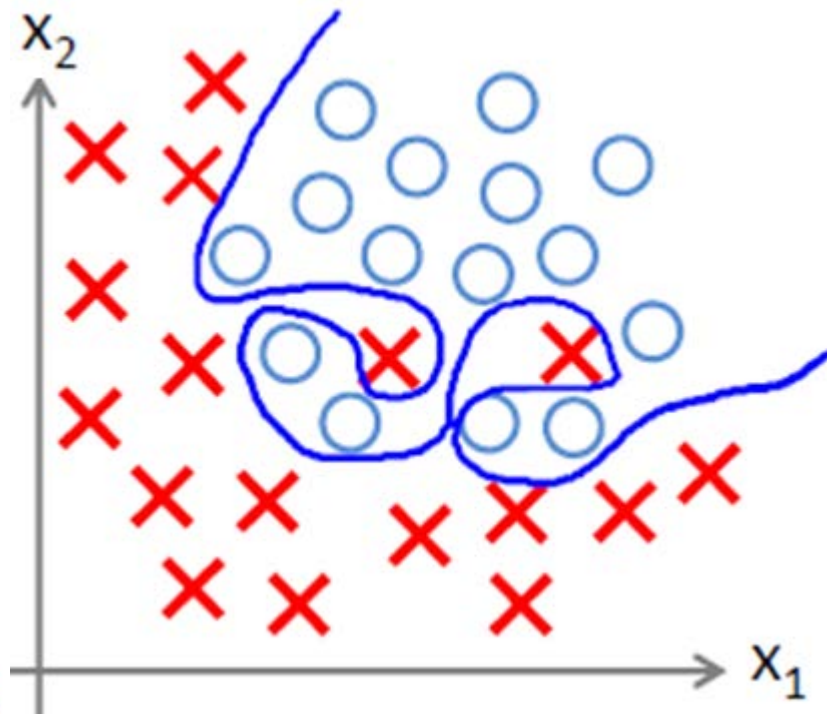
分类问题中的拟合



(a) Good fit 良拟合



(b) under-fit 欠拟合



(c) Over-fit 过拟合

解决策略:

- ◆ Early-stopping (早停法)
- ◆ Dropout(0.3)(随机失活/抛弃30%的神经元)
- ◆ Data enhancement (数据增强)
- ◆ Weight regularization (权重正则化, 可以降低模型的复杂度)

采用 Dropout 技术解决“过拟合”

- ◆ 随机失活是对深度神经网络进行优化的方法。
- ◆ 在神经网络的学习过程中，将隐含层的部分权重随机归零；
- ◆ 由于每次迭代受归零影响的节点不同，因此各节点的“重要性”会被平衡。
- ◆ 引入随机失活后，神经网络的每个节点都会贡献内容，不会出现少数高权重节点完全控制输出结果的情况，因此降低了网络的结构风险。
- ◆ 研究表明，随机失活提高了神经网络在视觉、语音识别、文档分类和计算生物学中监督学习任务的性能，在许多基准数据集上获得了最先进的结果。

(2) “梯度消失” 或 “梯度爆炸” 问题

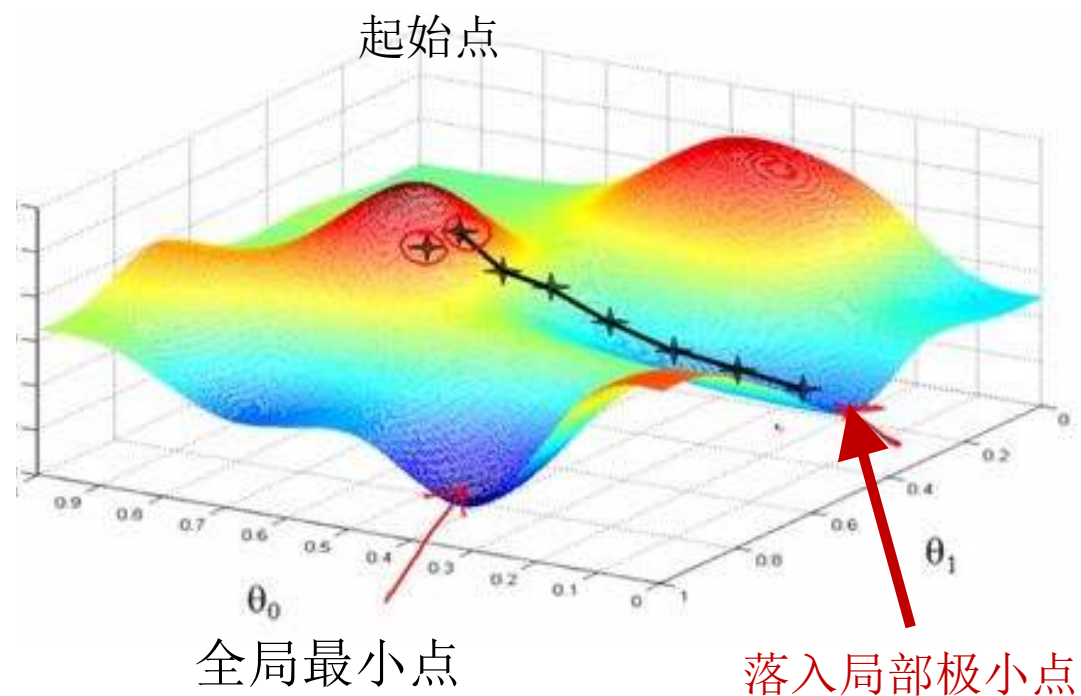
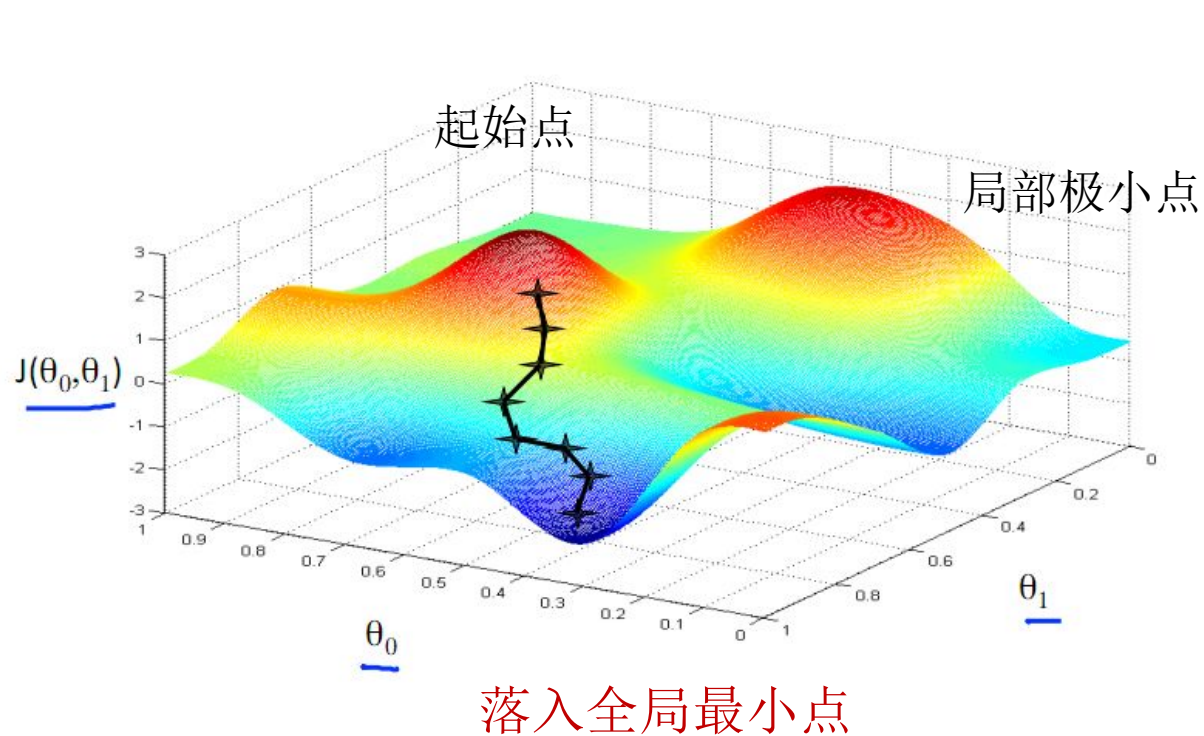
“梯度消失” 或 “梯度爆炸” (Gradient vanishing / explosion problem)

- ◆ BP算法使用**梯度下降法**更新权重，误差的梯度是在更新中**累积相乘**的。
- ◆ 当梯度值**小于 1.0**，越靠近输入层时，重复相乘越会导致梯度呈指数级下降，**接近于0**，使得**梯度消失**，这会导致后面层的**权重基本不更新**，训练过程几乎停顿，导致了BP神经网络算法**收敛速度慢**的现象。
- ◆ 当梯度值**大于 1.0**，重复相乘会导致梯度呈指数级**增长**，导致网络权重的大幅更新，即**梯度爆炸**，会使网络变得不稳定。
- ◆ **解决方法**：更换激活函数ReLU（导数为常数1）

(3) 局部极小值问题

局部极小值问题 (Local minimum problem)

BP算法是一种局部搜索的优化方法，在训练多层神经网络时，可能会陷入局部极小值，而非全局最小值。

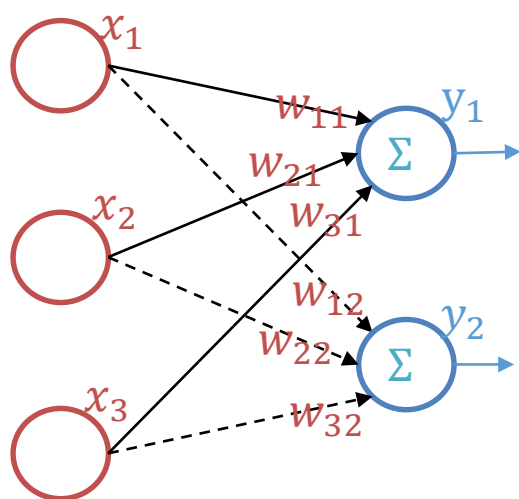


解决策略:采用启发式搜索算法，如模拟退火，以一定的概率接受“次优解”，概率随时间降低.

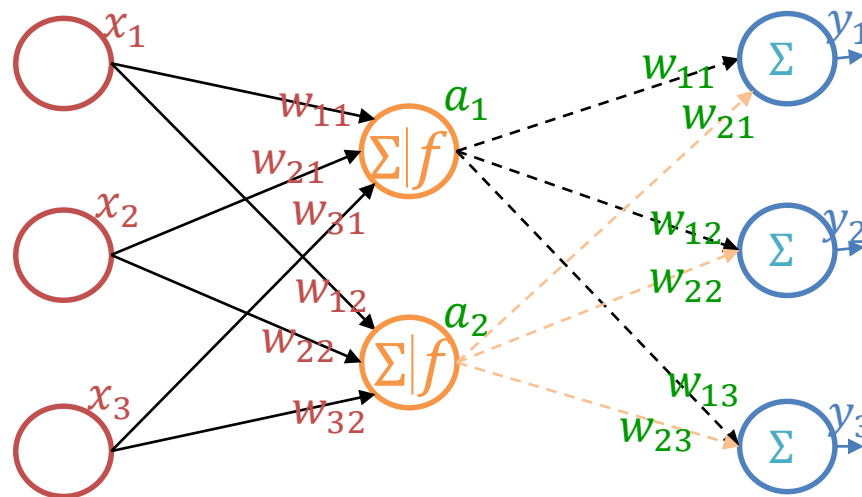
(4) 参数量巨大的问题

◆BP-NN采用全连接，权值数量多，参数量巨大。

◆应对之道：采用局部连接和权值共享方法，用以减少权值个数。



单层全连接网络



多层全连接网络

CNN减少参数量的两种方法

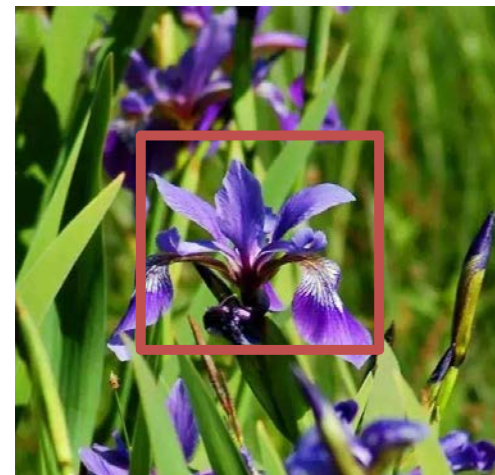
卷积神经网络采用以下**两种**方法减少参数量：

◆ 局部连接 (local connections)

- 卷积即**局部感知野**，用于局部连接。
- 因为在图像空间中，邻近像素的关联较紧密，而相距较远的像素相关性较弱。
- 因此，每个神经元没必要感知全局图像，只需感知其邻近的局部像素，然后在更高层网络将局部信息综合起来可得到全局信息。

◆ 权值共享 (weight sharing)

Translation invariance
卷积：具有平移不变性



5.4 卷积神经网络

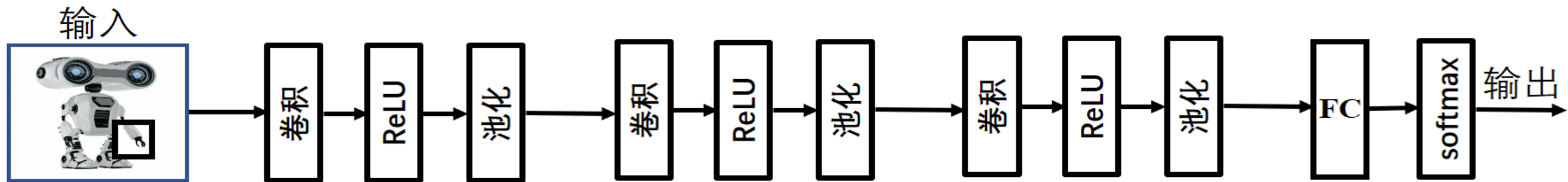
- ◆ 卷积神经网络（Convolutional Neural Network, CNN）是一种特殊的多层前馈神经网络，常用于监督学习。
- ◆ 早期的BP神经网络是全连接的，需要学习的参数量巨大，网络训练时间较长。
- ◆ 科学家发现：生物神经元有**感受野**（Receptive Field）机制，研究人员受此启发，提出了卷积神经网络。
- ◆ 卷积神经网络模仿了感受野的机制，采用卷积运算，使得人工神经元只连接其周围一定范围内的神经元，连接数量减少了，权重（参数）量也减少了。
- ◆ 卷积神经网络最早用来完成图像处理任务，特别是在图像分类、目标检测和语义分割等任务上不断取得突破性进展。

5.4 卷积神经网络

- ◆ 对卷积神经网络的研究最早可以追溯到1979年，日本学者**福岛邦彦**(Kunihiko Fukushima)提出了命名为“neocognitron”的神经网络，它是一个具有深层结构的神经网络，也是最早被提出的深度学习算法之一。
- ◆ 1989年，杨乐昆（Yann LeCun）等人在《Backpropagation Applied to Handwritten Zip Code》一文中第一次使用“convolution”和“kernel”术语。提出采用随机梯度下降法训练神经网络模型，并首次使用了“卷积神经网络”一词，因此，杨乐昆被誉为“**卷积神经网络之父**”。

5.4.1 卷积神经网络的整体结构

- ◆ 一个CNN可以包含多个网络层，每层网络由多个独立的神经元组成。
- ◆ CNN中的网络层主要分为3种类型：**卷积层**、**池化层**和**全连接层**。
- ◆ 通常，每个卷积层与最后一个全连接层之后都会采用激活函数。
- ◆ 与多层前馈神经网络一样，CNN也可以像搭积木一样，通过叠加多个网络层来组装。



- ◆ 多层前馈神经网络可以看成是由“**全连接层+ReLU层**”组合拼装而成的，而CNN可以看成是由“**卷积层+ReLU层+池化层**”（有时也省略池化层）组合拼装而成的。
- ◆ CNN具有**3个重要特性**：**权重共享**、**局部感知**和**亚采样**。
- ◆ 在卷积神经网络中，输入层输入的数据是训练样本或测试样本，其他网络层输入/输出的数据均称为**特征图**（Feature Map）。

5.4.1 卷积神经网络的整体结构

- ◆ 卷积神经网络不需要额外的特征工程，可以直接从图像中自动提取视觉特征，称为**学习式特征**；不用像传统图像处理技术那样提取**手工式特征**（如HOG、SIFT特征等）。
- ◆ CNN可识别具有极端可变性的模式，如手写体字符，且具有一定程度的**扭曲、平移、旋转、缩放不变性**，从而可以保留图像的空间特性，在图像处理方面具有显著优势。
- ◆ 与几乎所有其他的神经网络一样，**CNN也采用反向传播算法训练模型**。
- ◆ CNN通常包括
 - 1个输入层
 - 多个“卷积层+ReLU层+池化层”组合块，也可能是【n个“卷积层+ReLU层”+1个池化层】的组合
 - 多个连续的全连接层（中间不加激活函数）
 - 1个采用softmax函数的输出层。

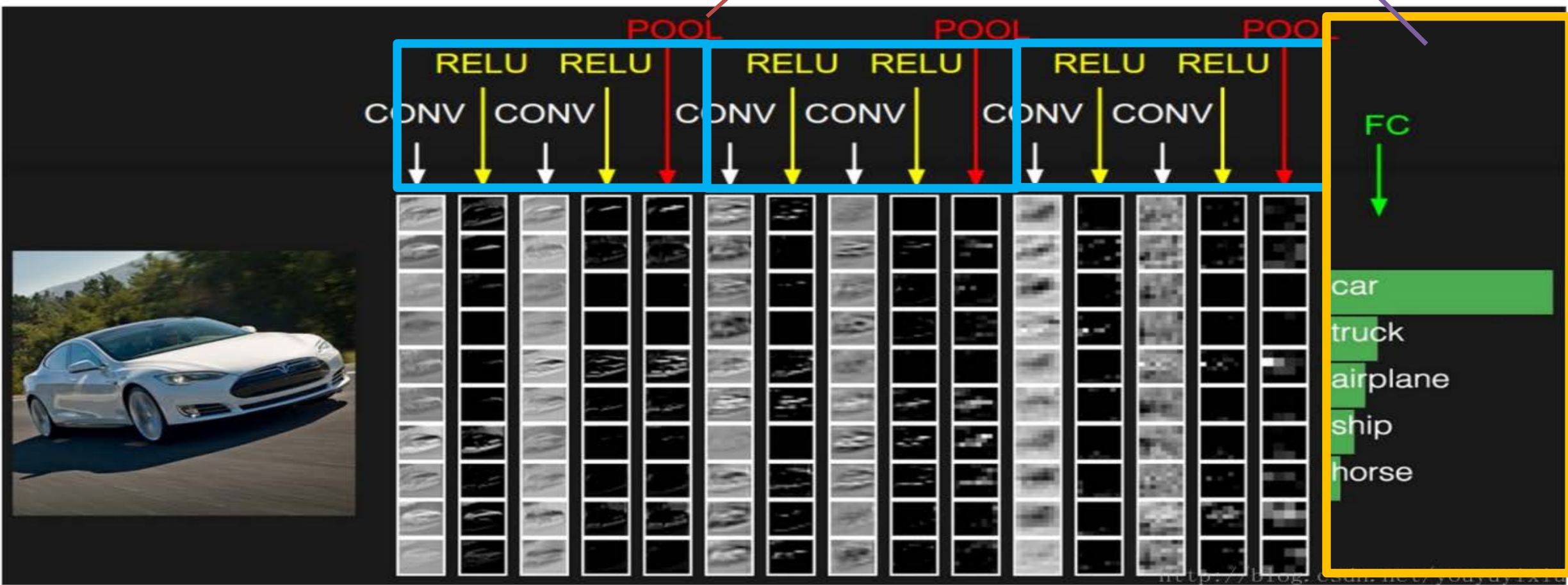
CNN的整体结构

Convolution + Activation + Pooling + Full connection

卷积层 + 激活函数 + 池化层 + 全连接层

池化出现多次，用于提取特征

全连接在网络模型
后部出现一次或多
次，用于做分类



5.4.1 卷积神经网络的整体结构

(1) 输入层。

- 灰度图： $W \times H$ 矩阵，其中 W 为宽度， H 为高度，图像深度为1；
- 彩色图： $W \times H \times 3$ 像素矩阵，“3”表示R、G、B三个颜色通道，即图像深度。

(2) 卷积层。

- 卷积操作就是点积运算。
- **卷积层的功能**：用卷积操作对输入的原始数据/特征图**提取特征**，输出卷积运算后产生的特征图。
- **卷积层是CNN的核心部分**，一个卷积层可以包含若干个卷积核（Kernel），一个卷积核就是一个二维的、高度和宽度相同的权重矩阵，记为Conv $F \times F$ ， $F \ll W$ 。
- $F \times F$ 的区域就是卷积核的**感受野**，卷积核与输入的图像或特征图只有 $F \times F$ 大小的局部连接；
- 在全连接神经网络中，隐藏层的神经元的感受视野是输入图像或特征图的全部大小，进行的是全局连接。

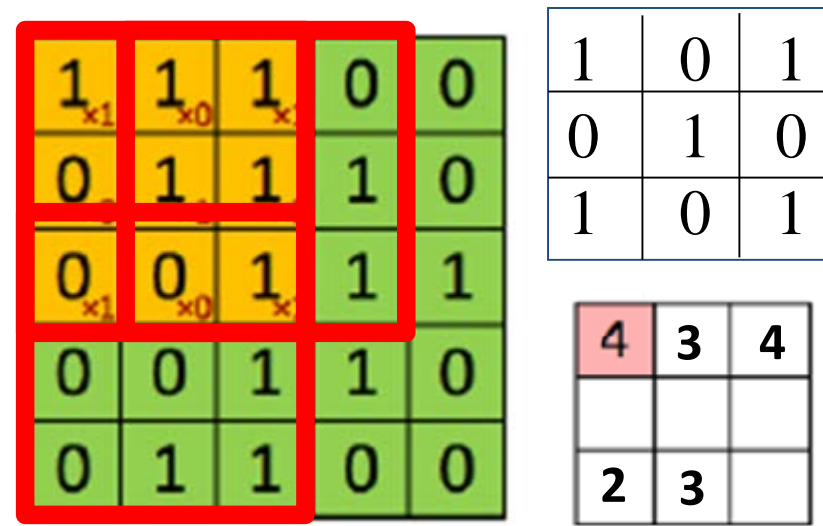
(2) 卷积层 (Convolution Layer)

◆ 卷积层包含一组可学习的滤波器，每个滤波器输出一个2维特征映射图。

◆ 每个滤波器就是一个神经元，滤波器覆盖的范围称为**感受野** (receptive field, 窗口)。

◆ 与卷积层相关的术语：

- Depth (**深度**) : RGB image, depth=3
- Stride (**步长**) : (窗口一次滑动的长度) 水平与垂直方向的步长相同
- zero-padding (以0填充)



Convolution with a 3x3 filter
用3x3滤波器进行卷积

(2) 卷积层

- 对灰色图像作卷积操作，用一个卷积核即可，因为一张灰色图像只有一个颜色通道；
- 对彩色图像作卷积操作，则同时需要有3个大小相同的卷积核，分别对彩色图像的R、G、B三个通道进行卷积运算，那么这个由若干个大小（Size，也称为尺寸）相同的卷积核堆叠而成的卷积核组就称为一个**滤波器**，记为 $\text{Conv } F \times F \times D$ ，其中D为该滤波器中包含的卷积核的个数，称为**滤波器的深度**（Depth），也称为**滤波器的通道数**，记为 #channel。
- 滤波器的深度取决于输入的图像或特征图的深度。
- 对于灰度图像，一个卷积核可以看作是深度为1的滤波器。

(2) 卷积层

- 一个卷积层中可以包含多个滤波器，而滤波器的个数是人为设定的，一个滤波器就是一个神经元。
- 通常，同层上的多个滤波器的大小、深度都相同，也是人为设定的，即**超参数**；
- 只是权重和偏置不同，需要训练得到，即**学习参数**。
- **每个滤波器，即神经元**，只对输入图像或特征图的部分区域进行卷积运算，负责提取图像或特征图中的局部特征。
- **靠近输入层的卷积层提取的特征往往是局部的，越靠近输出层，卷积层提取的特征越是全局化的。**
- 一个滤波器只关注一个特征，不同的滤波器用于提取不同的特征，如边缘、轮廓、纹理、颜色等。
- 每个滤波器只产生一层特征图，故有多少个滤波器，就会产生多少层特征图。一个神经网络中的所有神经元，就是构成了整张图像的特征提取器的集合。

5.4.1 卷积神经网络的整体结构

(3) 池化层。

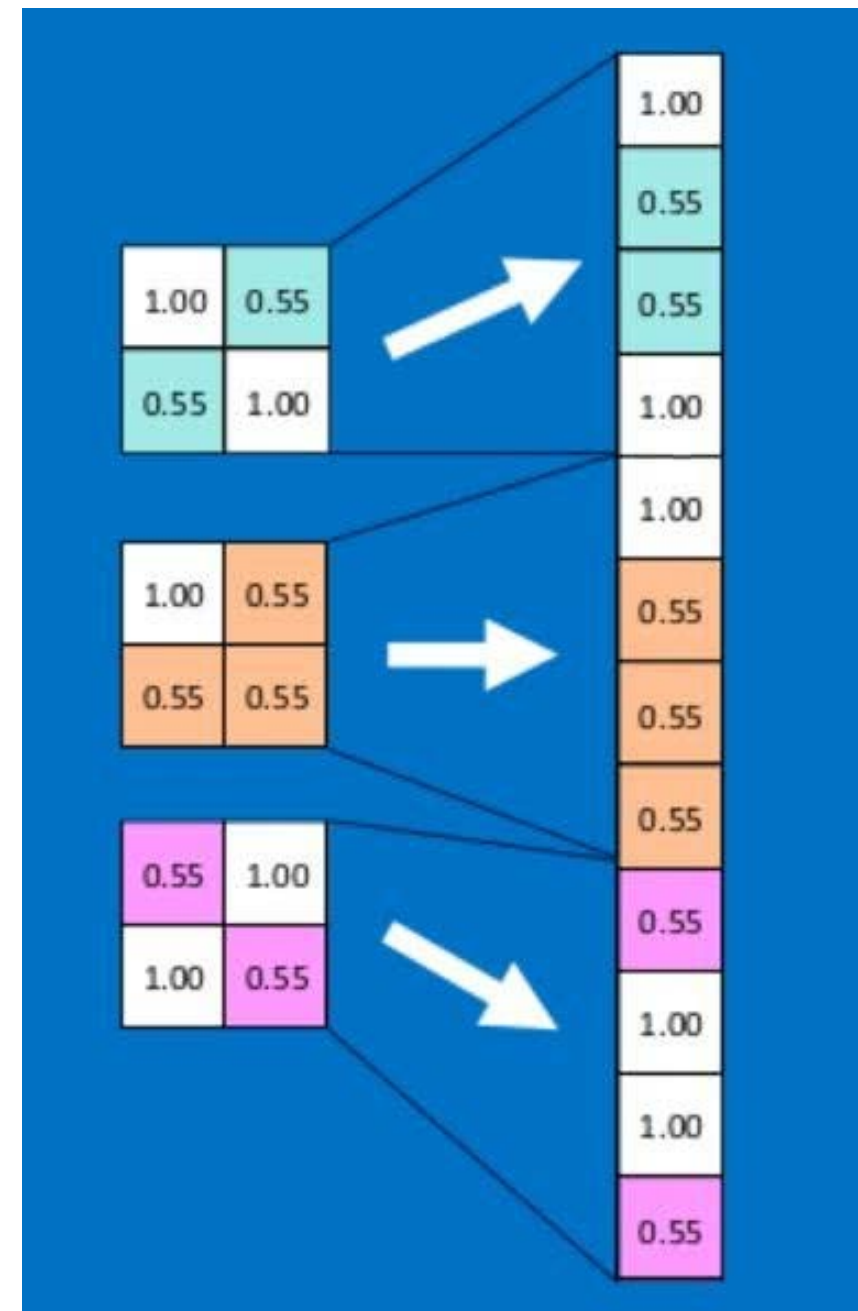
- 池化层也称为下采样层、子采样、亚采样。
- 池化层的**作用**：选择具有代表性的特征，去除不重要的或冗余的特征，用以降低特征维数，从而减少参数量和计算开销。

(4) 全连接层。

- 全连接层**通常置于CNN尾部**，它与传统的全连接神经网络的连接方式相同，都是**相邻两层上的所有神经元两两相连**，每个连接都有权重。
- 两个相邻全连接层之间只进行线性转换，不用激活函数。
- **全连接层的作用**主要是对卷积层和池化层提取的局部特征进行重新组合，得到全局信息，以减少特征信息的丢失。

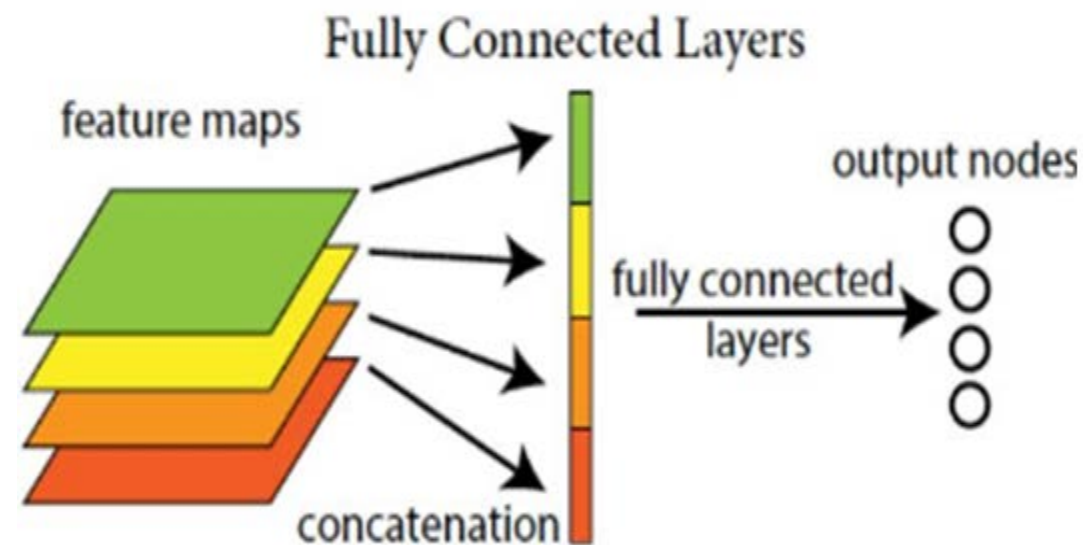
全连接的操作方式

- ◆ 首先，将经过卷积、激活函数、池化的多层特征向量抻直成一个一维的向量；
- ◆ 然后，采用全连接的方式将上述一维向量与输出层连接。



全连接层的作用

- ◆ 打破卷积特征的空间限制
- ◆ 对卷积层获得的不同的特征进行加权
- ◆ **全连接层**在整个卷积神经网络中**起到“分类器”的作用**，即通过**卷积、激活函数、池化**等网络层后，再经过全连接层对结果进行识别、分类。
- ◆ 最终目的：**得到属于不同类别的得分（即概率）**
- ◆ **输出层**则输出**每个类别的得分(概率)**



5.4.1 卷积神经网络的整体结构

(5) 输出层。

- 输出层用于输出结果，实际上就是最后一个全连接层之后的激活函数层。
- 根据回归或分类问题的需要选择不同激活函数，以获得最终想要的结果。
- 例如，二分类任务可以选择sigmoid作为激活函数，若是多分类任务，可以选择softmax作为激活函数。

◆ CNN的结构比传统多层全连接神经网络具有以下**两个方面优势**。

- (1) **连接方式不同，CNN的局部连接可大大减少参数数量和计算量。**
- (2) **对输入图像处理的方式不同，CNN保留了图像固有的空间特征。**

5.4.2 卷积运算

卷积是泛函分析中一种重要的运算。图像处理中的卷积是二维的。

1. 灰色图像上的卷积运算

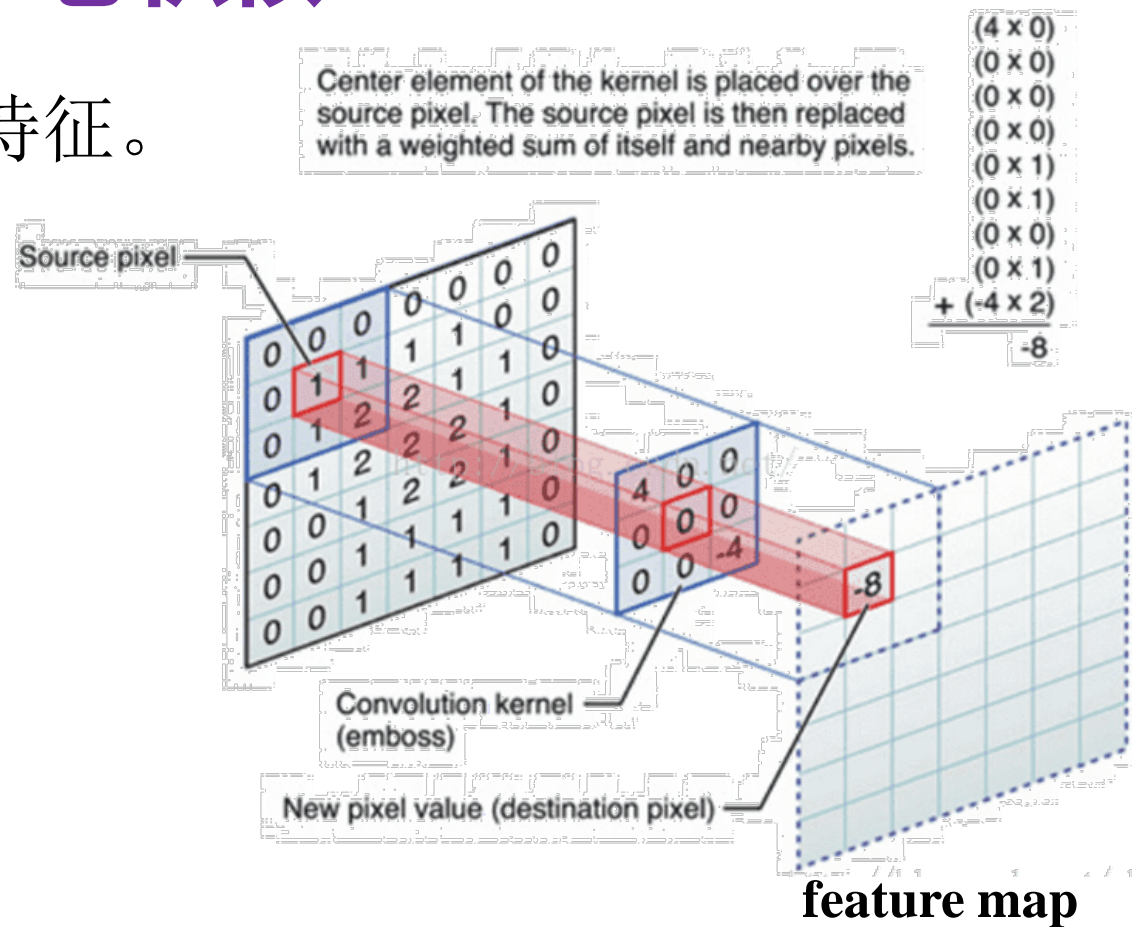
- 灰度图像： $W \times H$ 的像素矩阵，是单通道图像。其中每个像素的取值范围为 $[0, 255]$ ，表示像素的**强度**。
- 图像处理的底层就是对图像像素点进行操作。
- 卷积运算是对像素矩阵与卷积核矩阵重叠部分作**内积**的操作，即计算两个重叠矩阵中对应元素的乘积之和。
- 在图像上作卷积操作是用同一个卷积核在图像上**从左到右、从上到下**，按某个固定的步长进行**“Z”字形滑动**，遍历图像的每一个局部位置，并将每个位置的内积运算结果保存到输出特征图的相应位置。
- **步长**是指卷积核窗口在像素矩阵上滑动的位置间隔，以像素为计数单位。

灰度图上的单个卷积核

◆ 在灰度图像上使用单卷积核：抽取单个特征。

◆ Terms (术语)

- ✓ x : input, image (输入)
- ✓ **filter=kernel**, 只有在单通道/灰度图
像上：卷积核=滤波器，等价
- ✓ feature map (特征图)



灰度图上的单个卷积核

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

像素矩阵

⊗

1	0	1
0	1	0
1	0	1

卷积矩阵

=

4		

特征图

filter= Convolution Kernel

过滤器=卷积核

灰度图上的单个卷积核

3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

input, gray image

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

feature map

0	1	2
2	2	0
0	1	2

filter= Convolution Kernel

灰度图上单个卷积核的卷积运算示例

在下列灰度图像上，假设给定 4×4 的像素矩阵，卷积核大小为 2×2 ，滑动步长为2。

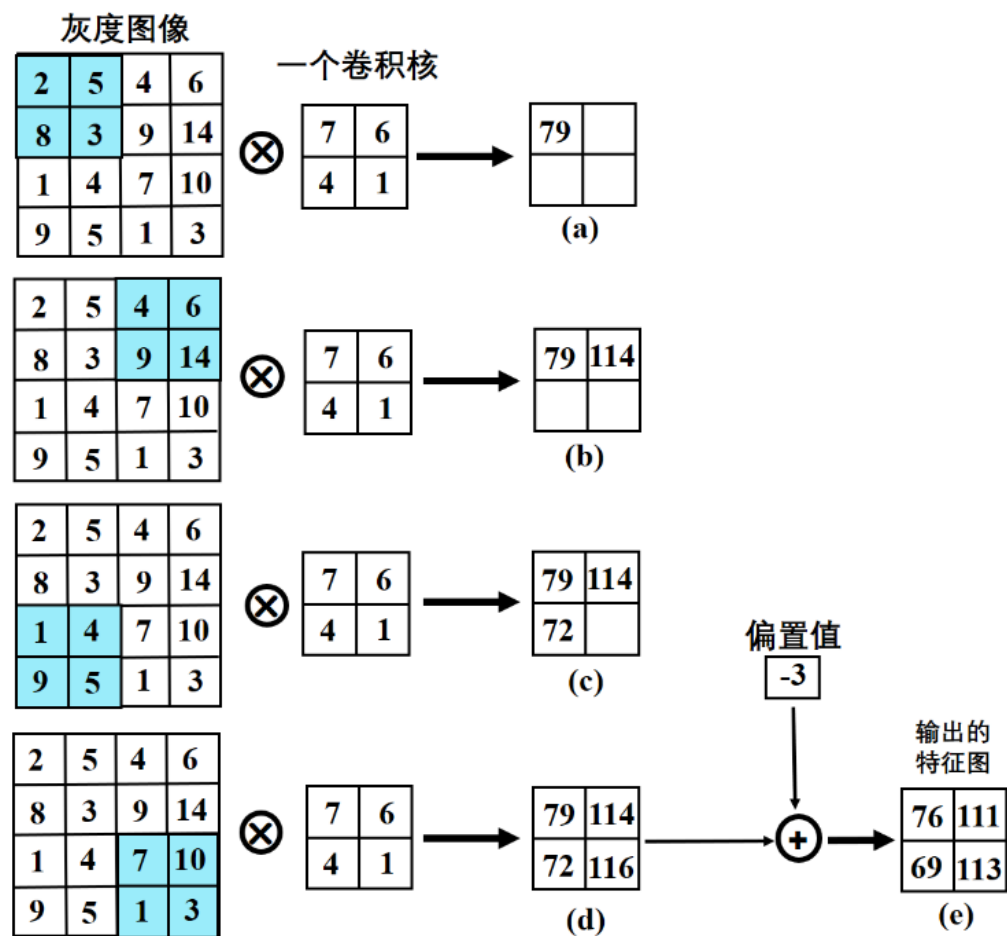
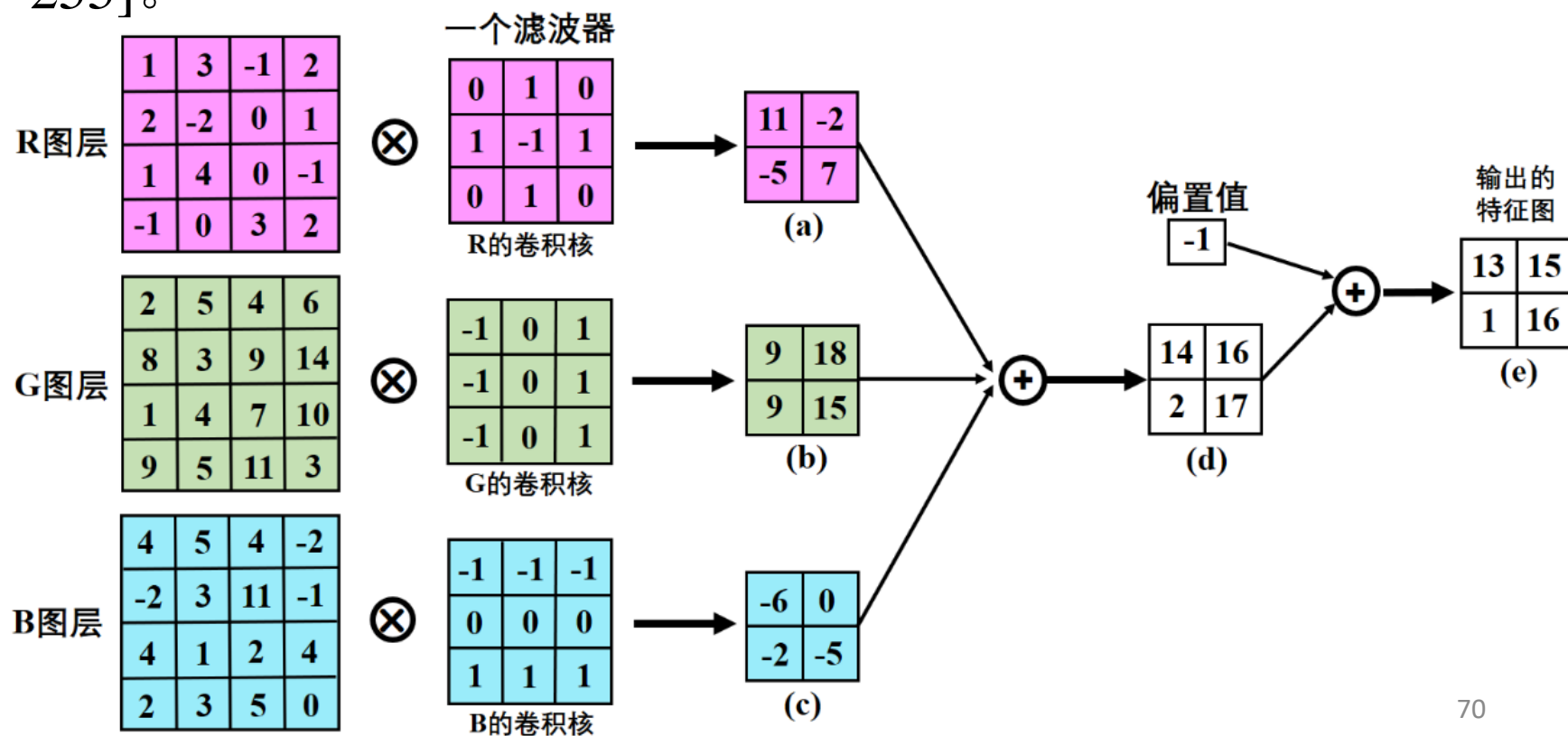


图5.12 灰度图像的卷积运算示例

5.4.2 卷积运算

2. RGB图像上的卷积运算

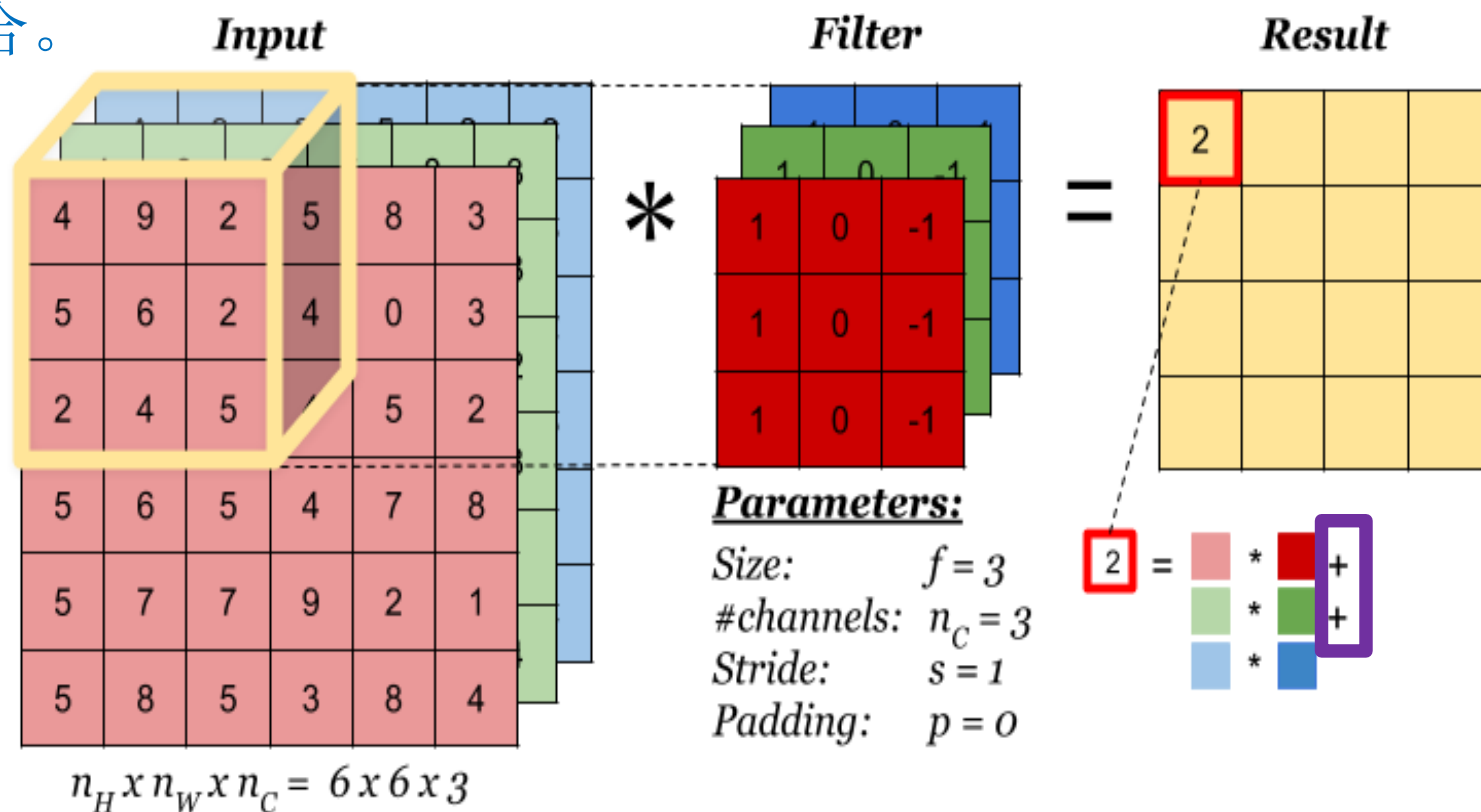
- 输入的彩色图像通常表示为 $W \times H \times 3$ 的像素矩阵，其中“深度”也称为通道。
- 彩色图像可以看作3个二维像素矩阵，每个像素矩阵存放所有像素的一种颜色值，像素取值范围为[0~255]。



RGB图上的单个过滤器

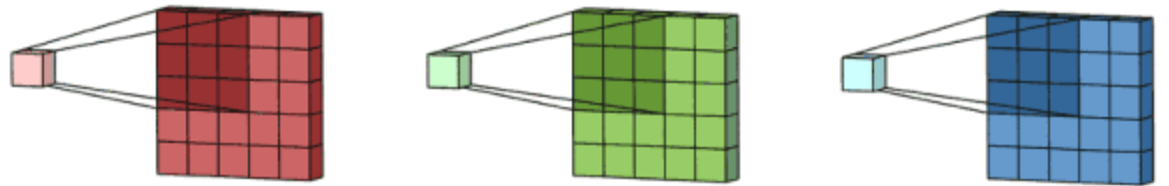
- ◆ 大多数输入图像都是 3 通道的。
- ◆ RGB 图像上使用单个滤波器，其作用是：抽取单个特征。
- ◆ RGB 输入通道的每个图层都有一个唯一的卷积核. 如 3×3 的卷积核，共 3 个卷积核。
- ◆ 滤波器就是各图层上卷积核的集合。

- ◆ 每个卷积核的大小: 3×3 ,
Height \times Width
- ◆ 滤波器的深度 = 卷积核的个数
= 前一层输出数据的深度
即 Depth = 3



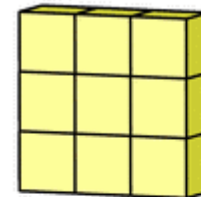
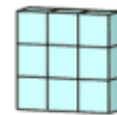
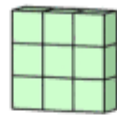
RGB图上的单个过滤器---Steps 1

- ◆滤波器的每个卷积核在各自的输入通道上「滑动」，产生各自的计算结果。
- ◆某个卷积核可能比其他卷积核具有更大的权重，以强调其所对应的输入通道的特征。
- ◆例如，滤波器的红色通道卷积核可能比其他通道的卷积核的权重更大，因此，对红色通道特征的反应要强于其他通道。



RGB图上的单个过滤器--Steps 2

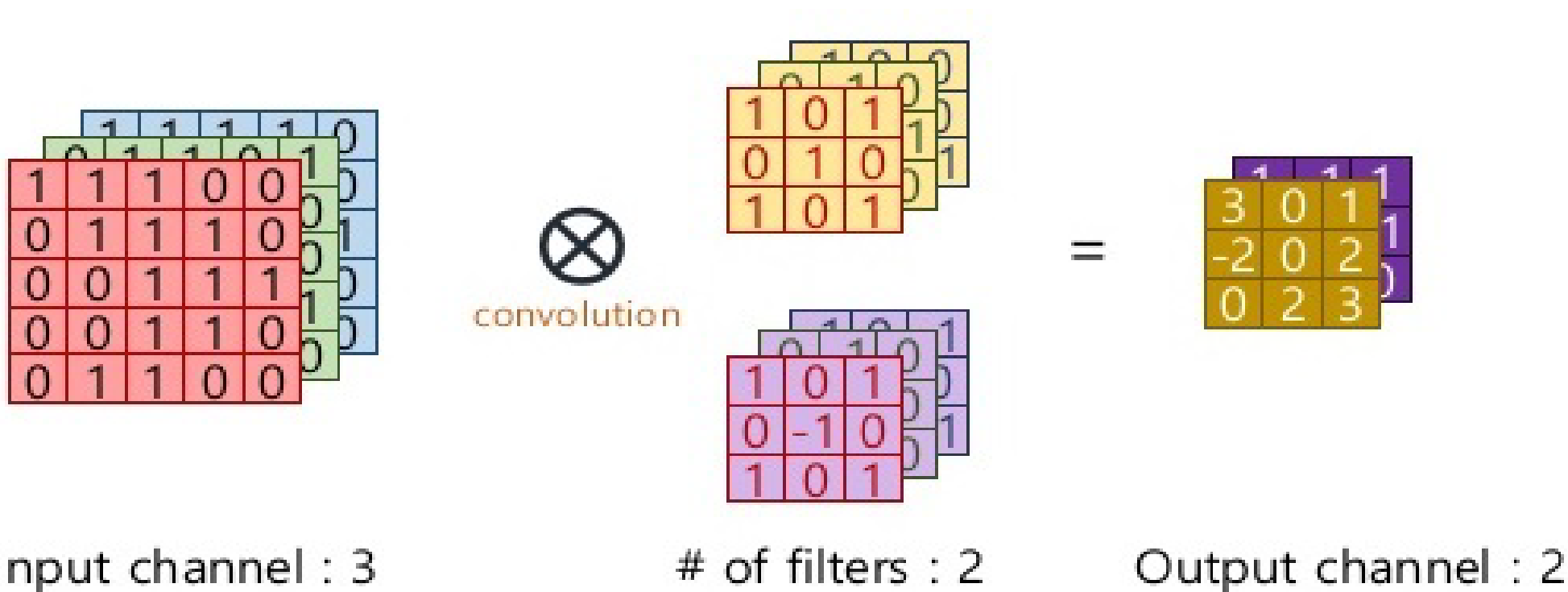
- ◆然后将滤波器的每个卷积核各自产生的结果汇在一起，形成一个总的输出通道。
- ◆即一个滤波器（不管其中包含几个卷积核）只产生一个输出结果。
- ◆偏置的作用是对每个滤波器的输出增加偏置项，以便产生最终输出通道。



bias

RGB图上的多个过滤器

- ◆RGB图像上使用多个滤波器：用于提取多个不同特征。
- ◆一个滤波器提取图像的一种局部特征。
- ◆在一个卷积层同时使用多个滤波器，可提取多种不同的局部特征。



RGB图上2个滤波器的卷积运算过程

R

Input Volume (+pad 1) (7x7x3)

$x[:, :, 0]$

0	0	0	0	0	0	0
0	0	1	1	2	2	0
0	0	1	1	0	0	0
0	1	1	0	1	0	0
0	1	0	1	1	1	0
0	0	2	0	1	0	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

$w0[:, :, 0]$

1	1	-1
-1	0	1
-1	-1	0

$w0[:, :, 1]$

-1	0	-1
0	0	-1
1	-1	0

$w0[:, :, 2]$

0	1	0
1	0	1
0	-1	1

Bias b0 (1x1x1)

$b0[:, :, 0]$

1

第1个滤波器，
即第1个神经元

Filter W1 (3x3x3)

$w1[:, :, 0]$

-1	-1	0
-1	1	0
-1	1	0

$w1[:, :, 1]$

1	-1	0
-1	0	-1
-1	0	0

$w1[:, :, 2]$

-1	0	1
1	0	1
0	-1	0

Bias b1 (1x1x1)

$b1[:, :, 0]$

0

第2个滤波器，
即第2个神经元

Output Volume (3x3x2)

$o[:, :, 0]$

1	0	-3
-6	1	1
4	-3	1

第1个滤波器的输出

$o[:, :, 1]$

-1	-6	-4
-2	-3	-4
-1	-3	-3

第2个滤波器的输出

滤波器的偏差

G

$x[:, :, 1]$

0	0	0	0	0	0	0
0	1	1	1	2	0	0
0	0	2	1	1	2	0
0	1	2	0	0	2	0
0	0	2	1	2	1	0
0	2	0	1	2	0	0
0	0	0	0	0	0	0

$x[:, :, 2]$

0	0	0	0	0	0	0
0	2	0	2	0	2	0
0	0	0	1	2	1	0
0	1	0	2	2	1	0
0	2	0	2	0	0	0
0	0	0	1	1	2	0
0	0	0	0	0	0	0

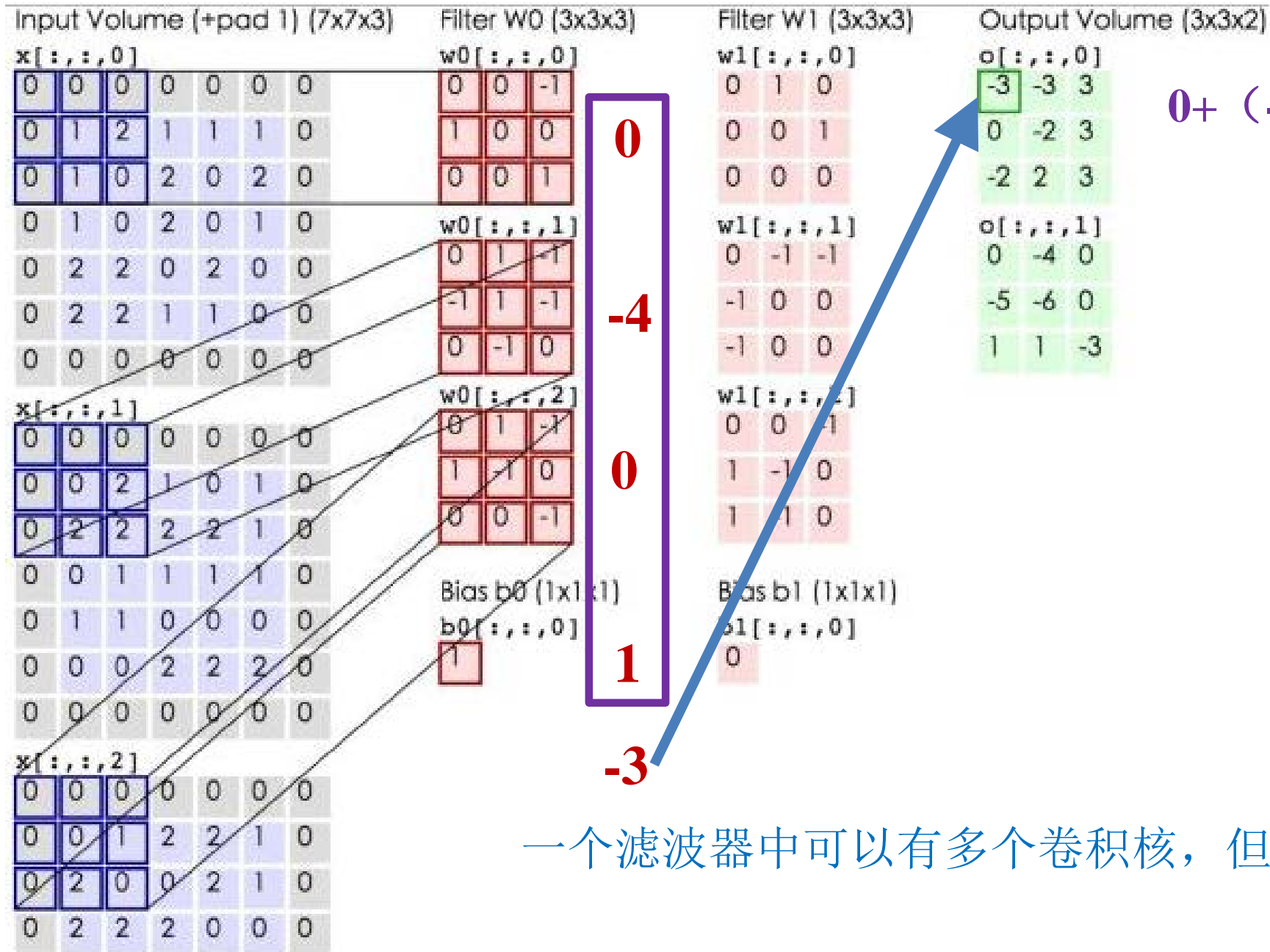
B

输入RGB图像

◆ 粉色矩阵是两个滤波器/神经元: ($w0, w1$)

◆ 滤波器的步长设为 2

RGB图上2个滤波器的卷积运算过程 (1)



一个滤波器中可以有多多个卷积核，但共有一个偏置值。

RGB图上2个滤波器的卷积运算过程 (2)

Input Volume (+pad 1) (7x7x3)

$x[:, :, 0]$

0	0	0	0	0	0	0
0	1	2	1	1	1	0
0	1	0	2	0	2	0
0	1	0	2	0	1	0
0	2	2	0	2	0	0
0	2	2	1	1	0	0
0	0	0	0	0	0	0

$x[:, :, 1]$

0	0	0	0	0	0	0
0	0	2	1	0	1	0
0	2	2	2	2	1	0
0	0	1	1	1	1	0
0	1	1	0	0	0	0
0	0	0	2	2	2	0
0	0	0	0	0	0	0

$x[:, :, 2]$

0	0	0	0	0	0	0
0	0	1	2	2	1	0
0	2	0	0	2	1	0
0	2	2	2	0	0	0

Filter W0 (3x3x3)

$w0[:, :, 0]$

0	0	-1
1	0	0
0	0	1

$w0[:, :, 1]$

0	1	-1
-1	1	-1
0	-1	0

$w0[:, :, 2]$

0	1	-1
1	-1	0
0	0	-1

Bias b0 (1x1x1)

$b0[:, :, 0]$

1

Filter W1 (3x3x3)

$w1[:, :, 0]$

0	1	0
0	0	1
0	0	0

$w1[:, :, 1]$

0	-1	-1
-1	0	0
-1	0	0

$w1[:, :, 2]$

0	0	-1
1	-1	0
1	-1	0

Bias b1 (1x1x1)

$b1[:, :, 0]$

0

Output Volume (3x3x2)

$o[:, :, 0]$

-3	-3	3
0	-2	3
-2	2	3

$o[:, :, 1]$

0	-4	0
-5	-6	0
1	1	-3

$$2 + (-2 + 1 - 2) + (1 - 2 - 2) + 1(\text{bias}) = 2 - 3 - 3 + 1 = -3,$$

Function of Convolution (卷积的作用)

◆ Detect the edges of an image (检测图像的边缘)

◆ Convolution operation = feature extraction

卷积运算 = 特征抽取

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

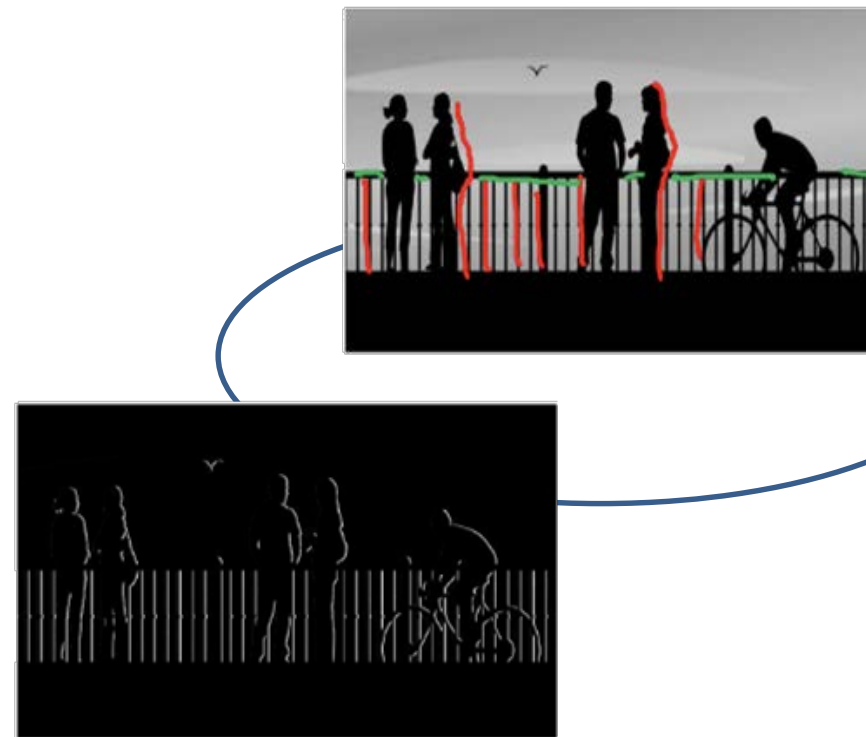
grey image
灰度图像

1	0	-1
1	0	-1
1	0	-1

Vertical edge filter
垂直边缘滤波器

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

Feature image
特征图像



vertical edges

5.4.2 卷积运算

- CNN的每个卷积层中都可以使用多个滤波器（其个数由人为设定）。
- 在每个卷积层上，一个滤波器只能输出一个通道数为1的特征图；
- 若有 K 个滤波器，则输出一个通道数为 K 的特征图。
- 针对输入的RGB 图像，第一个卷积层的所有滤波器的深度必须是3，因为颜色通道为3。
- 在其后的卷积层上，滤波器的深度取决于前一层输出的特征图的通道数。
- 一个滤波器提取一种局部特征，多个滤波器可以提取多种不同的局部特征。

5.4.2 卷积运算

3. 图像填充 (padding)

- 每做一次卷积运算，输出的特征图的尺寸都会减少若干个像素，经过若干个卷积层后，特征图尺寸会变得非常小。
- 另外，在卷积核移动的过程中，图像边缘的像素参与卷积运算的次数远少于图像内部的像素，这是因为边缘上的像素永远不会位于卷积核的中心，而卷积核也不能扩展到图像边缘区域以外，因此会导致图像边缘的大部分信息丢失。
- 若想尽可能多地保留原始输入图像的信息，可以在卷积操作之前，在原图像的周围填充p圈固定的常数，例如，填充常数0，这种操作称为**填充** (padding)。

在原特征图的四周填补一圈0值

0	0	0	0	0	0
0	1	3	-1	2	0
0	2	-2	0	1	0
0	1	4	0	-1	0
0	-1	0	3	2	0
0	0	0	0	0	0

⊗

卷积核

0	1	0
1	-1	1
0	1	0



输出保持原来尺寸的特征图

4	-5	6	-2
-2	11	-2	0
4	-5	6	4
2	6	-1	0

5.4.2 卷积运算

- **填充的主要目的**是调整输入数据的大小，使得输出数据的形状保持与输入数据一致。需要根据具体情况来确定超参数 p 的值。
- 在实践中，当设置padding = valid时，表示不填充0值，当padding = same时，表示自动计算 p 值来填补0值，使得卷积运算前、后的特征图的尺寸相同。
- 采用填充技术有以下**两个作用**：
 - ①CNN的深度不再受卷积核大小的限制，CNN可以不断地堆叠卷积层。若不作填充，当前一层输出的特征图的尺寸比卷积核还小时，就无法再进行卷积操作，也就无法再增加卷积层了。
 - ②可以充分利用图像的边界信息，避免遗漏图像边界附近的重要信息。

5.4.2 卷积运算

4. 卷积运算后特征图尺寸的计算公式

- 填充的 p 值和步长 s 的值都会影响卷积输出特征图的大小。
- 设当前卷积层中滤波器的个数为 K ，输入特征图的尺寸为 $H \times W \times D$ （ D 为通道数），卷积核的尺寸为 $F \times F$ ，填充为 p ，步长为 s ，则执行卷积运算后，输出特征图的尺寸 $H' \times W' \times D'$ 的计算公式为：

$$H' = \frac{H+2p-F}{s} + 1, \quad W' = \frac{W+2p-F}{s} + 1, \quad D'=K$$

其中，主要超参数包括：每个卷积层中滤波器的个数 K 、卷积核或滤波器的大小 F 、步长 s 、填充 p 。当前卷积层中学习参数的个数为 $(F \times F \times D + 1) \times K$ 。

卷积的步长

- ◆ 一次滑动的步长
- ◆ 有height上和width上的步长
- ◆ 一般，高、宽上的步长取相同的值。
- ◆ 右图中的stride = 2, 指在两个维度上的步长都为2
- ◆ 若将步长的值设置大于1，就相当于在stride=1的卷积结果中作了**下采样**
- ◆ 实际上，是跳过去、不计算某些像素，能够成倍减少计算量。

10	10	10	0	0
10	10	10	0	0
10	10	10	0	0
10	10	10	0	0
10	10	10	0	0

Stride = 1: 一次滑动1格

1	0	-1	0	30	30
1	0	-1	0	30	30
1	0	-1	0	30	30

10	10	10	0	0
10	10	10	0	0
10	10	10	0	0
10	10	10	0	0
10	10	10	0	0

Stride = 2: 一次滑动2格

1	0	-1	0	30
1	0	-1	0	30

卷积的填充 (Padding)

◆ Padding = valid (不补0)

若不进行补零操作，每卷积一次，宽和高方向的数据维度下降(F-1)，其中F为卷积核大小。

◆ Padding = same (补0)

- ✓ 在输入的边界周围进行0或复制填充
- ✓ 卷积前、后特征图的高与宽不变

5 * 5

10	10	10	0	0
10	10	10	0	0
10	10	10	0	0
10	10	10	0	0
10	10	10	0	0

padding: valid

卷积操作后，图像尺寸: 3 * 3

1	0	-1
1	0	-1
1	0	-1

0	30	30
0	30	30
0	30	30

padding: same

0	0	0	0	0	0	0
0	10	10	10	0	0	0
0	10	10	10	0	0	0
0	10	10	10	0	0	0
0	10	10	10	0	0	0
0	10	10	10	0	0	0
0	0	0	0	0	0	0

7 * 7

1	0	-1
1	0	-1
1	0	-1

-20	0	20	20	0
-30	0	30	30	0
-30	0	30	30	0
-30	0	30	30	0
-20	0	20	20	0

5 * 5

- ◆ 原图H=W=5，Padding=same，pad=1，H=W=7
- ◆ F = 3，stride = 1，卷积后 H=W=5

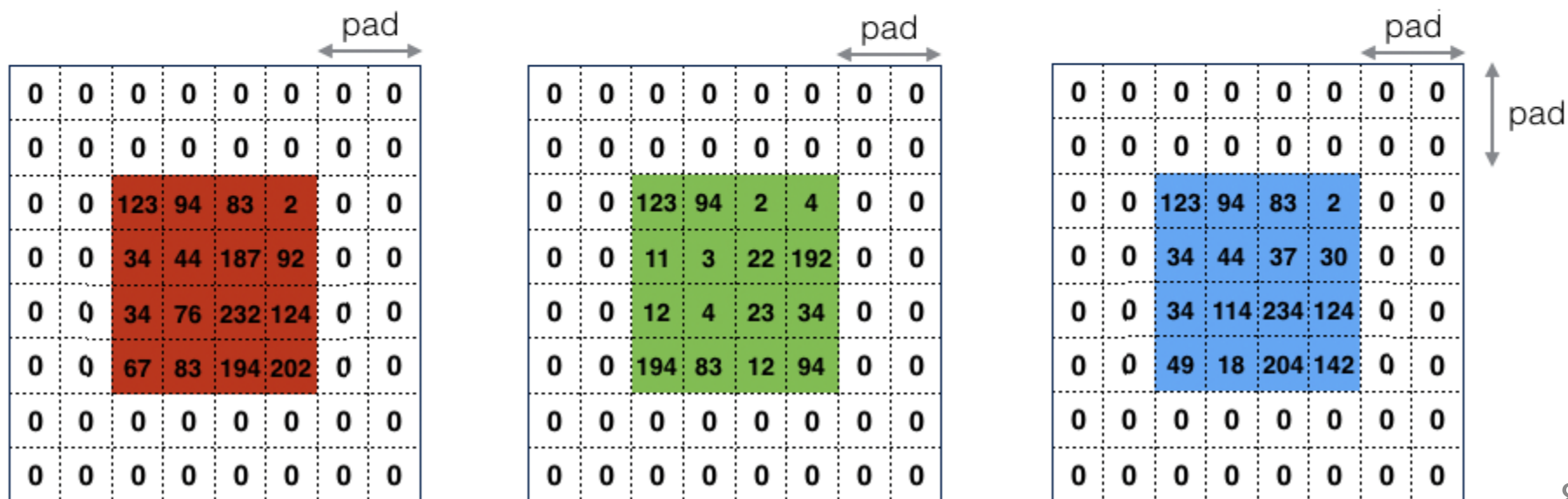
Padding, pad=2



==

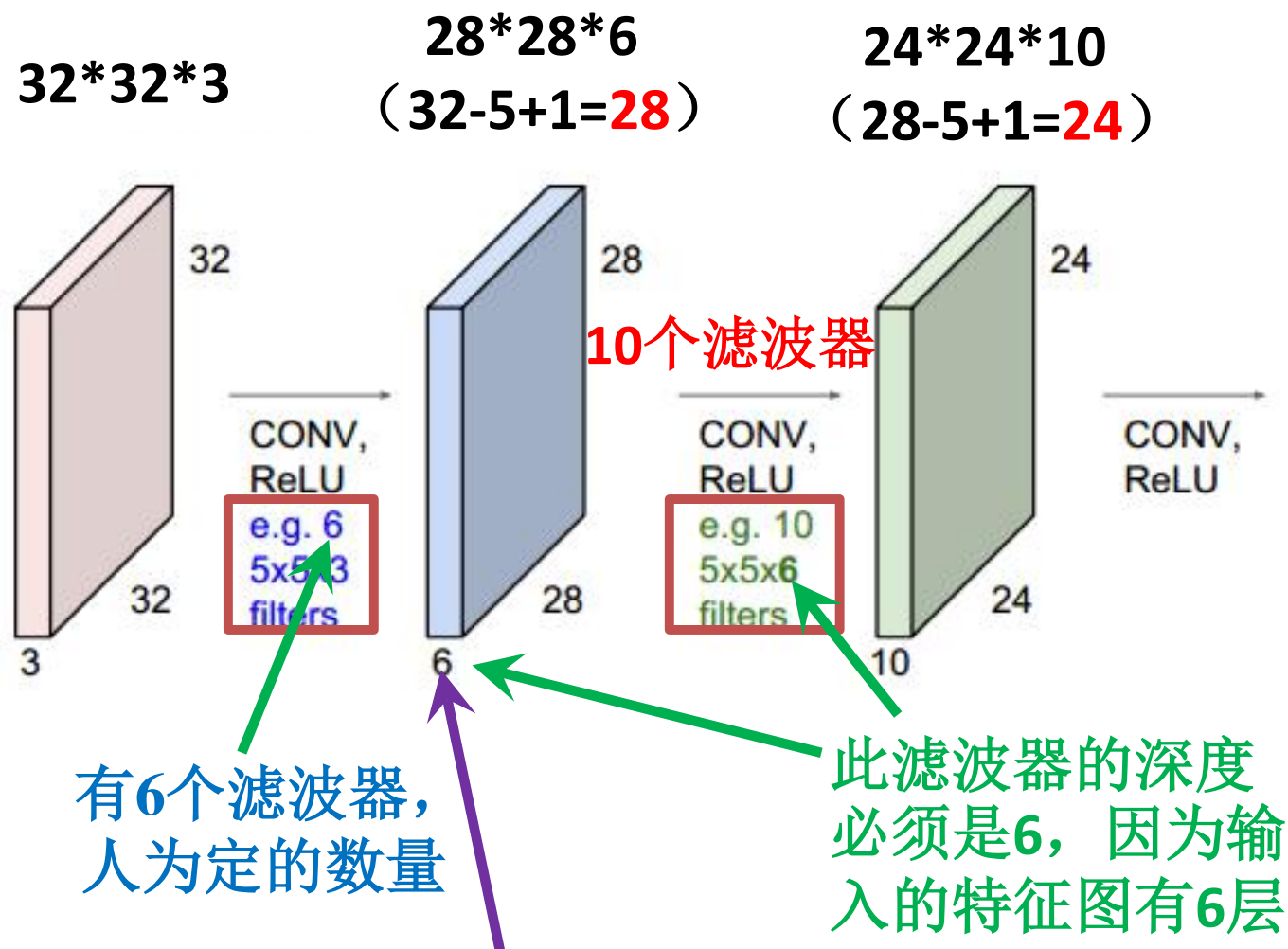
	Blue				
Green	123	94	83	2	
Red	123	94	83	4	30
	123	94	83	2	92
	34	44	187	92	124
	34	76	232	124	142
	67	83	194	202	

按0填充



卷积层

- ◆ 上一层滤波器的个数
= 下一层输入数据的深度
= 下一卷积层 滤波器的深度
- ◆ 滤波器的个数 = 提取特征的数量
每层滤波器的个数都是超参数，
程序员可以自己调节。



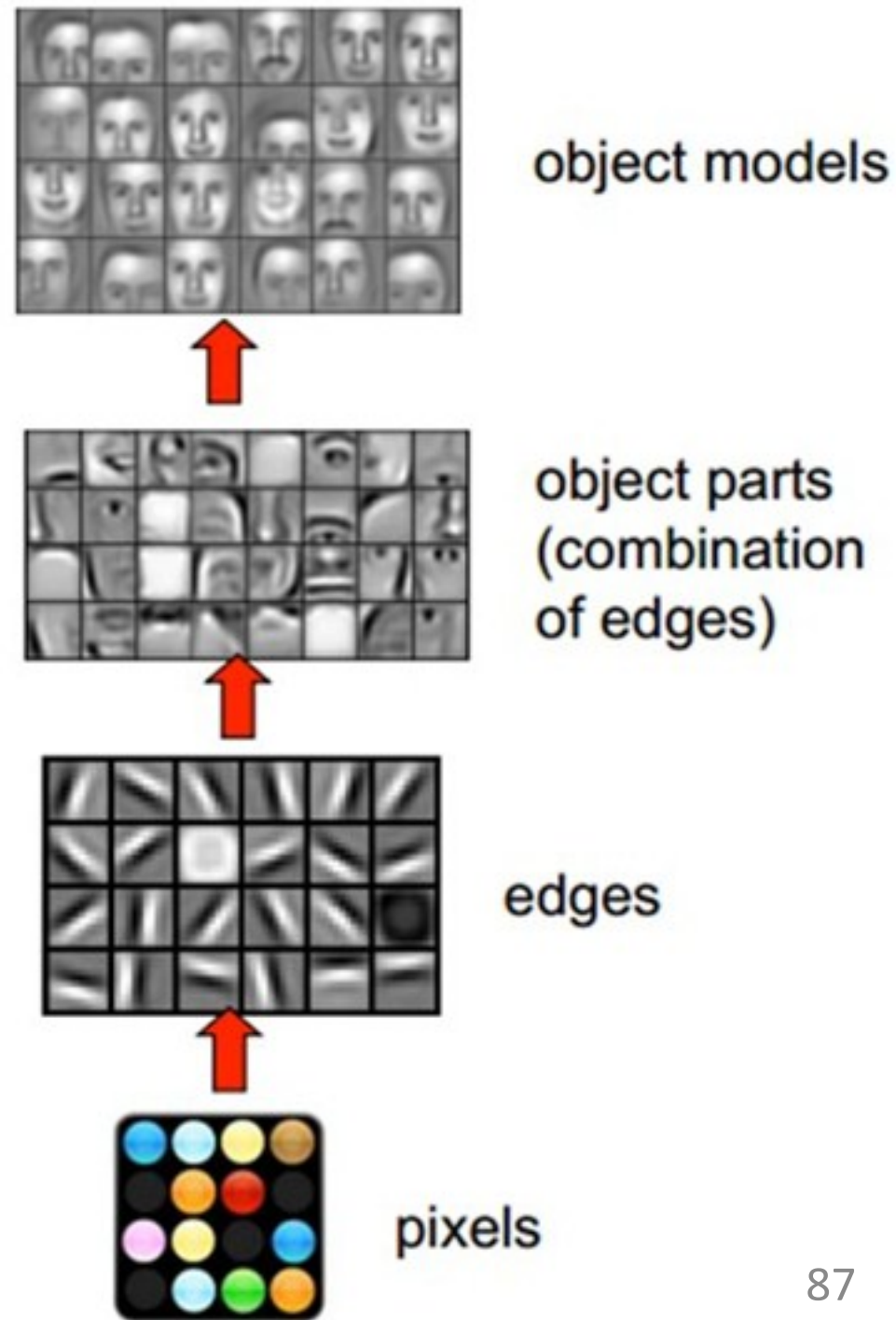
- ◆ 用一个滤波器对输入图像的RGB三通道分别进行卷积，得到3个特征图层。
- ◆ 将这3层叠加，得到该滤波器对应的1层输出图层，故6个滤波器就有6个输出的特征图层。

隐层的卷积

隐层的卷积：可提供特征的组合

◆多层卷积：

- 一层卷积得到的特征往往是局部的
- 层数越高，学到的特征就越全局化



卷积层的总结

✓ Input: $W_1 \times H_1 \times D_1$

✓ 超参数：过滤器个数: K

过滤器维度: F

步长: S

padding: P

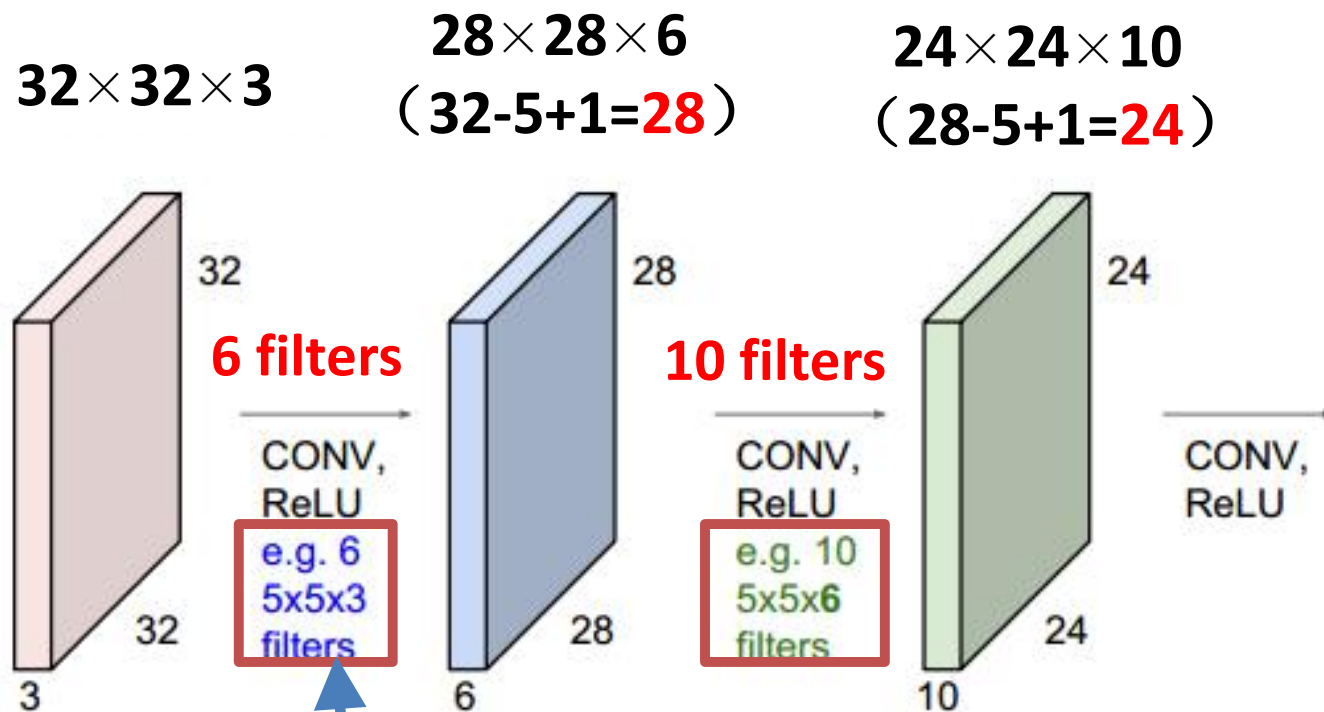
✓ Output: 下一层输出图像尺寸: $W_2 \times H_2 \times D_2$

$$\text{其中: } W_2 = \frac{W_1 + 2P - F}{S} + 1, H_2 = \frac{H_1 + 2P - F}{S} + 1, D_2 = K$$

✓ 一个卷积层中参数个数: $(F \times F \times D_1 + 1) \times K$

第一层卷积层中参数个数: $(5 \times 5 \times 3 + 1) \times 6 = 456$,

C1层, 若不用卷积, 而用全连接, 则参数个数是 $(32 \times 32 \times 3 + 1) \times 6 = 18438$ 个。



总结：卷积、滤波器、学习参数的个数

- ◆ 在一个卷积层中，可以人为设置 N （超参）个滤波器。
- ◆ 一个神经元就是一个滤波器。
- ◆ 每个滤波器中可包含 d 个卷积核， d 就是该滤波器的深度， d 的值等于前一层输出的特征图层数。（ d 不是超参）
- ◆ 若一个卷积核尺寸（大小）是 $n*n$ ，其中 $n*n$ 个值叫做权重，是训练获得的，但 n 的值本身是超参。卷积就是权重的集合。
- ◆ 一个滤波器的尺寸（大小）往往写成 $n*n*d$ 。
- ◆ 由于一个滤波器的 d 个卷积核共用同一个偏置值，所以包含 N 个滤波器的卷积层中需要学习（即训练）的参数（不是超参）有 $(n*n*d+1) * N$ 个。

5.4.2 卷积运算

5.转置卷积

- 卷积操作可看作是下采样。
- 对于某些特定任务，需要将图像恢复到原来的尺寸，这个将图像由小尺寸转换为大尺寸的操作称为上采样。
- 传统的上采样方法有：最近邻插值法、线性插值法和双线性插值法等。
- 然而，这些上采样方法都是基于人们已有的先验经验设计的。但在很多应用场景中，人们并不具有正确的先验知识，因此上采样的效果不理想。
- 我们希望神经网络能够自动学习如何更好地进行上采样，转置卷积就是一种自动上采样的方法。转置卷积又称为反卷积或逆卷积。

转置卷积的示例

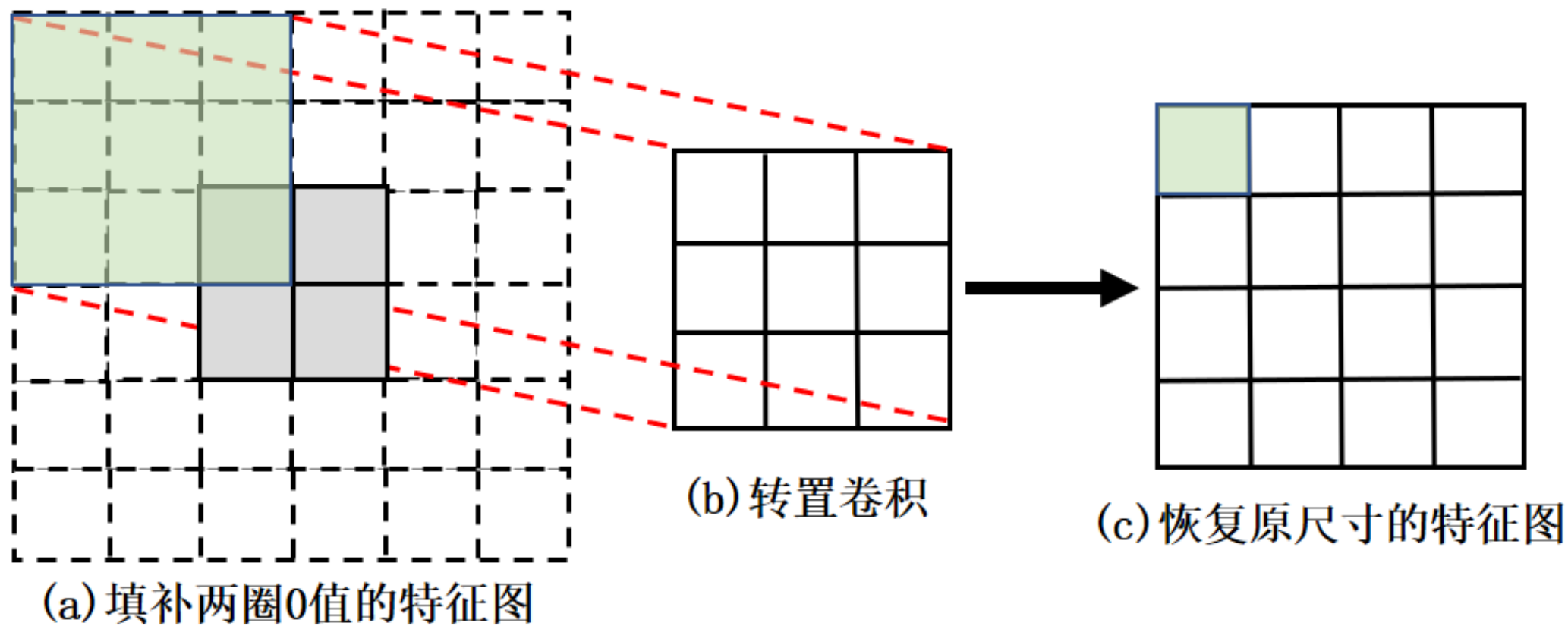


图5.16 $F=3$, $s=1$, $p=2$ 转置卷积的运算过程

5. 转置卷积

- 确定 p 值的公式为 $p=F-1$ 。
- 卷积是用一个小窗口看大世界，而转置卷积是用一个大窗口的一部分去看小世界，即卷积核比原输入图像尺寸大。
- 需要注意的是：卷积操作和转置卷积并不是互逆的两个操作。
- 一个特征图A经过卷积操作后，得到特征图B，而B再经过转置卷积操作后，并不能恢复到A中原始的元素值，只是保留了原始的形状，即大小相同而已。所以，转置卷积虽然又叫逆卷积，但事实上，它不是原来卷积操作的逆运算。

5.4.2 卷积运算

6.卷积神经网络结构的特点

(1) 局部连接。

- 人类对外界的认知一般是从局部到全局，先感知局部，再逐步认知全局。卷积即局部感受野。
- CNN模仿了人类的认识模式：一个神经元只与特征图局部区域中的元素相连。
- 每个卷积层的输入特征图或卷积核的大小是不同的，卷积核大小的不同意味着感受野范围的不同。
- 随着网络的加深，神经元的感受野范围逐层扩大，所提取图像特征的全局化程度越来越高，直到全连接层，全连接层中每个神经元的感受野覆盖了前一网络层的全部输出，得到的就是全局特征。

5.4.2 卷积运算

CNN是：

- 在卷积层，先用感受野小的卷积提取图像的局部特征；
- 在全连接层，再用全连接将所有特征重新组合在一起，提取全局特征。
- 局部连接实际上减少了神经元之间的连接数，也就减少了参数量，起到了降低计算量的作用。

5.4.2 卷积运算

(2) 权值共享。


- 权值共享是指卷积核在滑过整个图像时，其参数是固定不变的。
- 计算同一个通道的特征图的不同窗口时，卷积核中的权值是共享的，这样可以极大地减少参数量。
- 需要指出的是：同一卷积核只是针对同一通道的特征图共享权值，不同滤波器在同一通道上的卷积核不共享权值，不同通道上的卷积核也不共享权值。
- 一个滤波器的多个卷积核共享同一个偏置值，即一个神经元共用一个偏置值。
- 在CNN的隐藏层中，共享卷积核的参数可以减少学习参数的数量，降低处理高维数据的计算压力。例如：AlexNet 的参数有1亿，采用共享权值后，减至 6000万。

可见，卷积神经网络的两个特点（局部连接和权值共享），都是减少学习参数量的方法。

权值共享

◆在卷积层中, 每个神经元连接数据窗口的权重是固定的, **每个神经元只关注一个特征。**

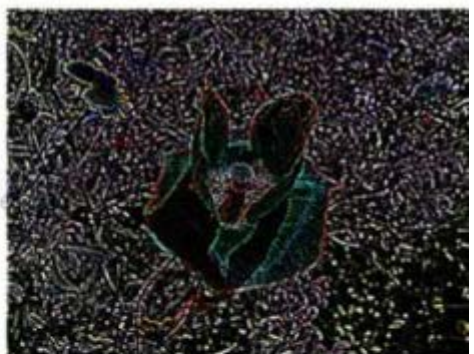
◆神经元就是图像处理中的**滤波器**, 比如边缘检测专用的Sobel滤波器;




filter

-1	-1	-1
-1	8	-1
-1	-1	-1

=

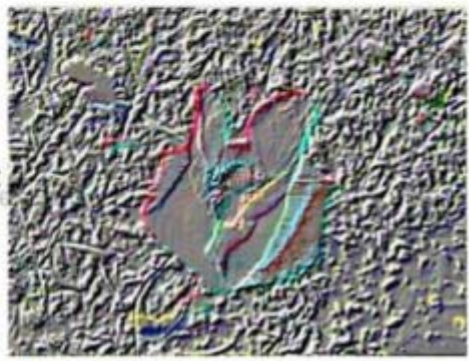


中心点边缘



-1	-1	0
-1	0	1
0	1	1

=



对角线边缘

◆不同的神经元（即滤波器）有不同的权值, 作用于同一图像上, 所提取的特征不同。

◆**每个滤波器都会专门关注一个图像特征,** 比如垂直边缘、水平边缘、颜色、纹理等;

↓

1	0	-1
1	0	-1
1	0	-1

↓

1	1	1
0	0	0
-1	-1	-1

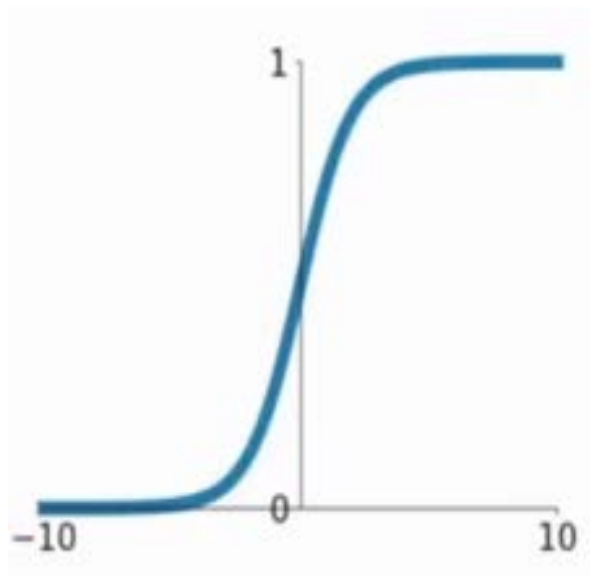
卷积神经网络的局限性

- ① 训练网络模型时，**不仅需要大量的训练样本，还需要高性能算力**，例如需要使用GPU，还需要花大量的时间调试超参数；
- ② **所提取特征的物理含义不明确**，即不知道每个卷积层提取到的特征表示什么含义，神经网络本身就是一种难以解释的“黑箱模型”；
- ③ 深度神经网络**缺乏完备的数学理论证明**，这也是深度学习一直面临的问题，目前仍无法解决。

5.4.3 激活函数

- ◆ 在多层CNN中，第 i 个卷积层的输出与第 $i+1$ 个卷积层的输入之间有一个函数映射，即激活函数。
- ◆ 激活函数是NN中的重要组成部分，它是对网络层输出的线性组合结果做**非线性映射**。
- ◆ 若激活函数是线性函数，则无论神经网络有多少层，整个网络的功能等价于感知机，网络的逼近能力十分有限。
- ◆ 选择**非线性激活函数**，可大幅度提升深度神经网络的表达能力，使其几乎可以逼近任何函数。
- ◆ **激活函数的作用**就是给网络模型提供非线性的建模能力。
- ◆ 在现代CNN中，每个卷积层后面都有非线性激活函数，目的是向模型中加入非线性元素，以解决非线性问题。**一般在同一个网络中使用同一种激活函数**。
- ◆ 早期，CNN的隐藏层大多采用sigmoid函数作为激活函数，自从2012年起，几乎所有的CNN均采用ReLU系列函数做激活函数了。

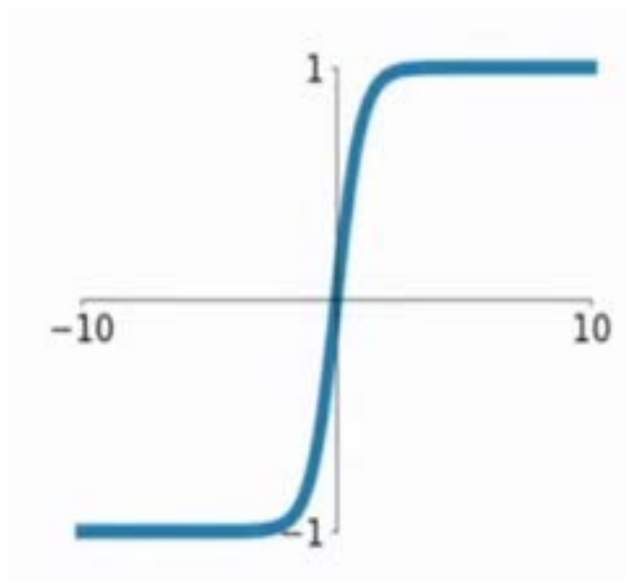
激活函数



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid function

指数运算，效率低

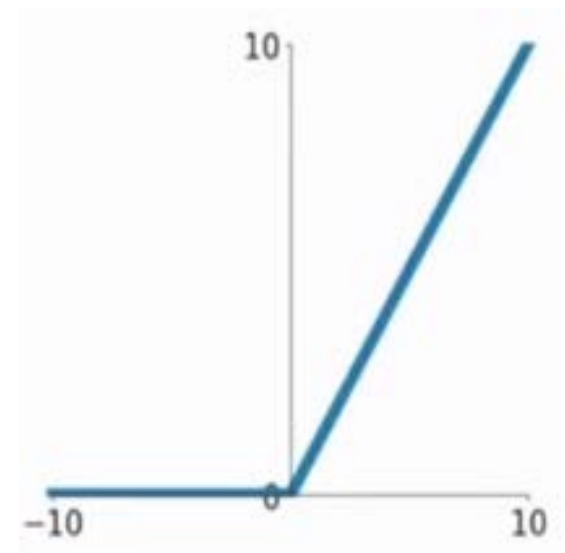


$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

tanh function

双曲正切函数

指数运算，效率低



$$f(x) = \max(0, x)$$

ReLU

线性运算，效率极高，
最常用

5.4.3 激活函数

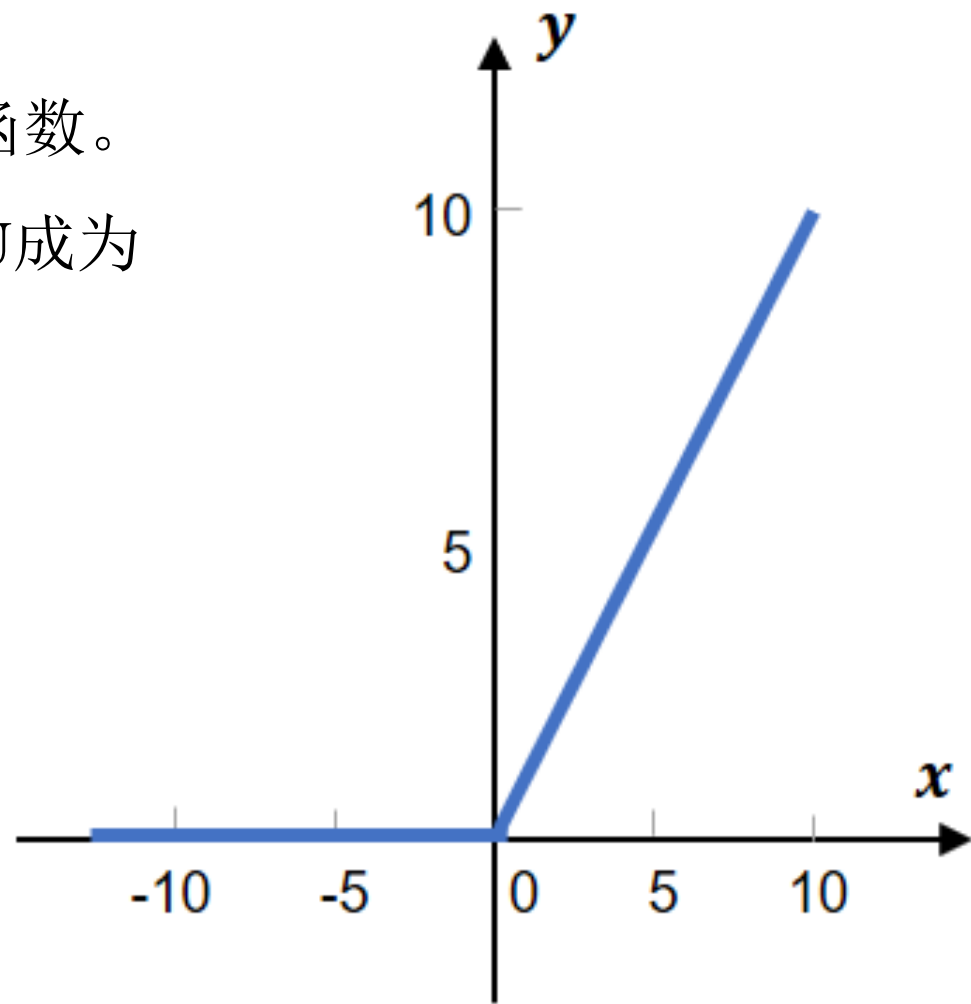
1. ReLU函数

- ReLU是修正线性单元的简称，是最常用的激活函数。
- AlexNet模型采用的激活函数是ReLU，从此ReLU成为深度神经网络模型中应用最广泛的激活函数。
- ReLU是一个简单的分段线性函数，但从整体看，ReLU是一个非线性函数：

$$y = f(x) = \max(x, 0) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

- ReLU函数是分段可导的，并人为规定在0处的梯度为0，其导数形式如下：

$$\frac{\partial y}{\partial x} = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$



ReLU函数的优点

- (1) 计算简单且快，求梯度简单，收敛速度比 sigmoid 与 tanh 函数快得多，ReLU 仅需要做简单的阈值运算。
- (2) S型函数在 x 趋近于正负无穷时，函数的导数趋近于零，而ReLU的导数为0或常数，在一定程度上缓解了梯度消失的问题。
- (3) ReLU具有生物上的可解释性，有研究表明：人脑中同一时刻大概只有 1%~4% 的神经元处于激活状态，同时只响应小部分输入信号，屏蔽了大部分信号。sigmoid 函数和 tanh 函数会导致形成一个稠密的神经网络；而ReLU函数在 $x < 0$ 的负半区的导数为0，当神经元激活函数的值进入负半区时，该神经元不会被训练，使得网络具有稀疏性。可见，ReLU 只有大约50%的神经元保持处于激活状态，引入了稀疏激活性，使神经网络在训练时会有更好的表现。

ReLU函数的缺点

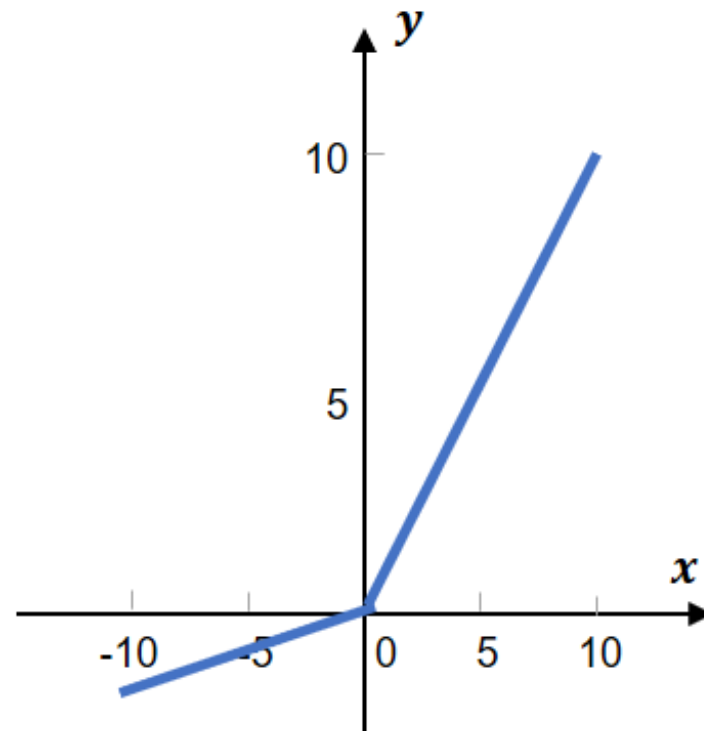
- (1) ReLU函数的**输出是非零中心化的**，使得后一层神经网络的偏置偏移，影响梯度下降的速度。
- (2) 采用ReLU函数，**神经元在训练时比较容易“死亡”**，即在某次不恰当地更新参数后，所有输入都无法激活某个神经元，则该神经元的梯度固定为0，导致无法更新参数，而且在之后的训练中，此神经元再也不会被激活，这种现象称为“**神经元死亡**”问题。在实际使用中，为了避免上述情况，提出了若干ReLU的变种，如 LeakyReLU 函数等。

5.4.3 激活函数

2. LeakyReLU函数

- LeakyReLU称为带泄露的ReLU，简记为LReLU，
- 它在ReLU梯度为0的区域保留了一个很小的梯度，以维持参数更新。
- LReLU函数的数学表达式：

$$y = f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$$



其中， $\alpha \in (0,1)$ 是一个很小的常数，如0.01，当 $\alpha < 1$ 时，LReLU也可以写作：

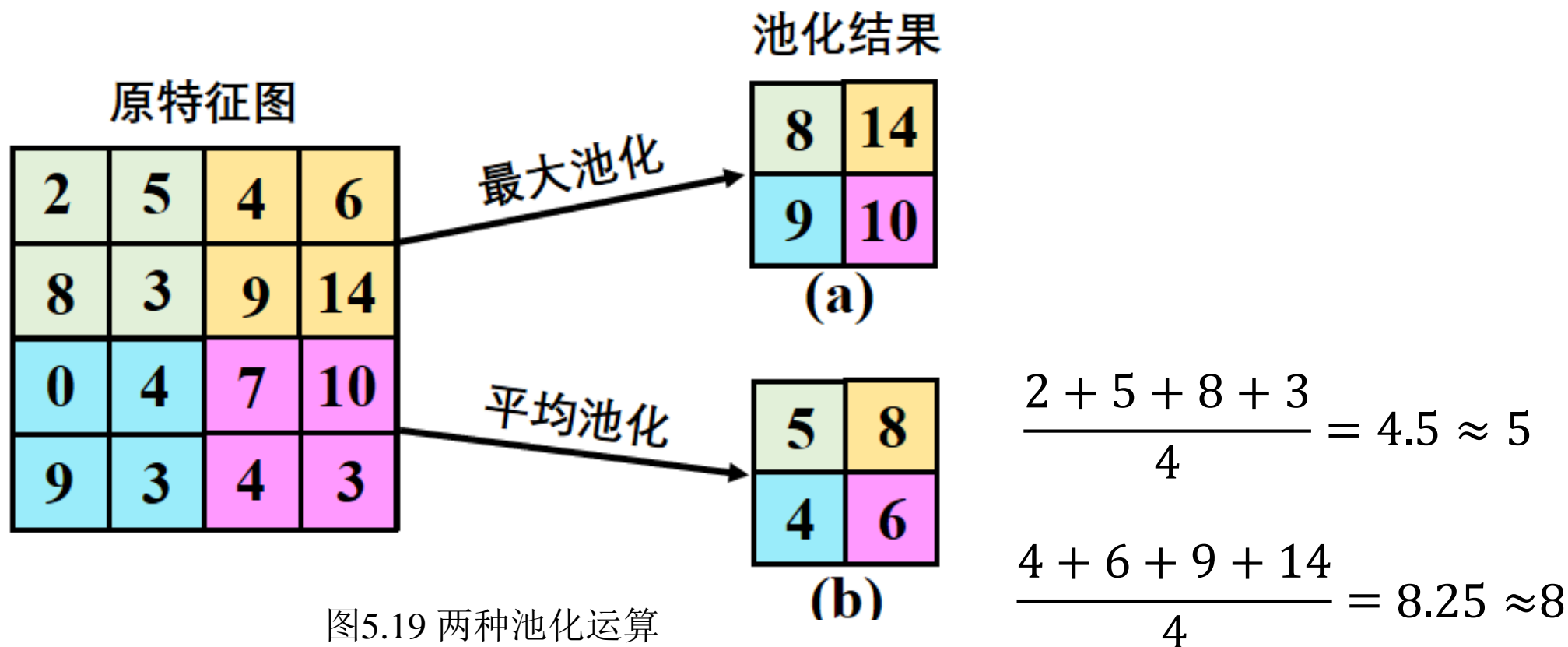
$$f(x) = \max(x, \alpha x)$$

- LReLU函数的导数形式为： $\frac{\partial y}{\partial x} = \begin{cases} 1, & x > 0 \\ \alpha, & x \leq 0 \end{cases}$
- LReLU函数的导数总是不为零，**解决了一部分神经元死亡的问题。**
- 但在实际使用的过程中，LReLU函数并非总是优于ReLU函数。

5.4.4 池化运算

- ◆ 池化是一种非线性下采样，池化层也称为下采样层。
- ◆ 最常见的池化运算采用大小为 2×2 、步长为2的滑动窗口操作，有时窗口尺寸为3，更大的窗口尺寸比较罕见。
- ◆ 实际上，池化也是一个特殊的卷积运算，它与普通卷积的**区别**有以下两点：
 - ① 池化卷积核的步长等于卷积核的大小，即 $s=F$ ，称为**不重叠的池化**；
 - ② 池化卷积核的权重不是通过学习获得的，而是允许人为选择进行下采样的方式，所以，**池化核的权重不是学习参数**。
- ◆ 两种常用的池化运算
 - **最大池化**（Max Pooling），它取滑窗内所有元素中的最大值；
 - **平均池化**（Average Pooling），它取滑窗内所有元素的平均值。

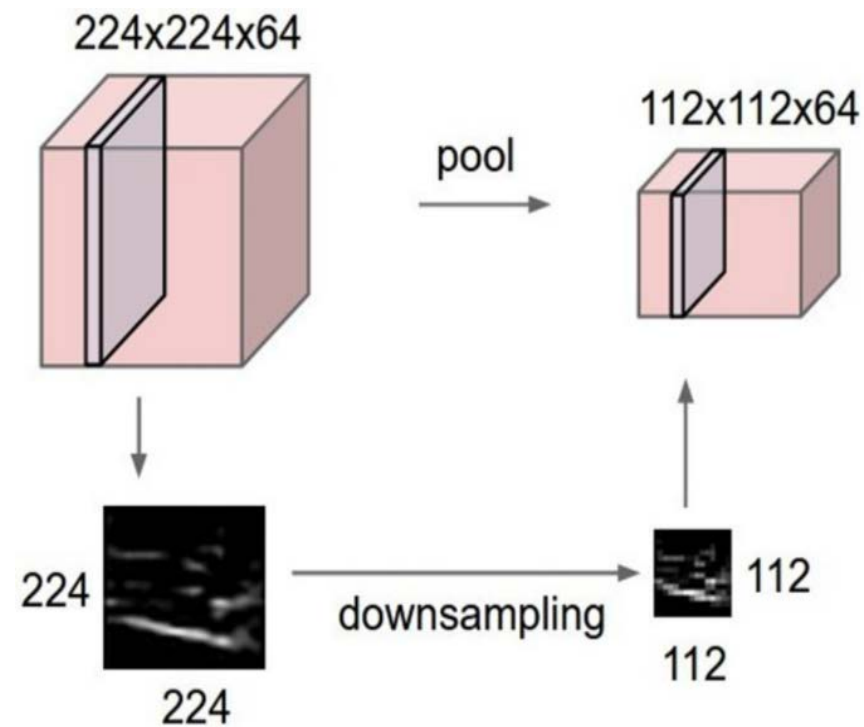
5.4.4 池化运算



- ◆ **平均池化**操作可以较好地保留图像的背景信息，但是图像中的物体边缘会被钝化。
- ◆ **最大池化**操作可以更清晰地保留图像的纹理信息，在提取目标轮廓等特征时会更有效。

池化层（Pooling layer）

- ◆ 池化将输入图像分割成一组不重叠的矩形，相当于卷积核的**stride**（步长）=**F**（尺寸）
- ◆ 池化层夹在连续的卷积层中间，用于压缩数据和参数的量，减小过拟合。
- ◆ 如果输入是图像的话，那么池化层的最主要作用就是压缩图像。
- ◆ 在宽和高维度上进行下采样，不改变深度的维度；
- ◆ 右图的**池化**相当于对输入数据使用了 2×2 ， $\text{stride} = 2$ 的卷积核，但是该卷积核不是通过学习获得，而是**人为定义的，不算是参数**。
- ◆ 2×2 的池化能够减少一半计算量
- ◆ 相比于卷积，池化层允许你自行选择进行下采样的方式。



5.4.4 池化运算

池化层夹在连续的“卷积层+ReLU”组合块中间，其作用如下。

(1) 保持尺度不变性。

池化操作是对特征图中的元素进行选择，保留图像的重要特征，去掉一些冗余的信息，保留下来的信息具有尺度不变性的特征，可以很好地表达图像的特征。

(2) 降低特征维度。

池化操作分别独立作用于特征图的每个通道上，降低所有特征图层的宽度和高度，但不改变图像的深度（通道数）。可以起到降低特征维度、防止过拟合的作用，同时可以减少参数量和计算开销。

池化层总结

✓ 输入：图像尺寸 $W_1 \times H_1 \times D_1$

✓ Hyper parameters (超参数) :

滤波器尺寸: F

stride (步长) : $S=F$ (因为不重叠)

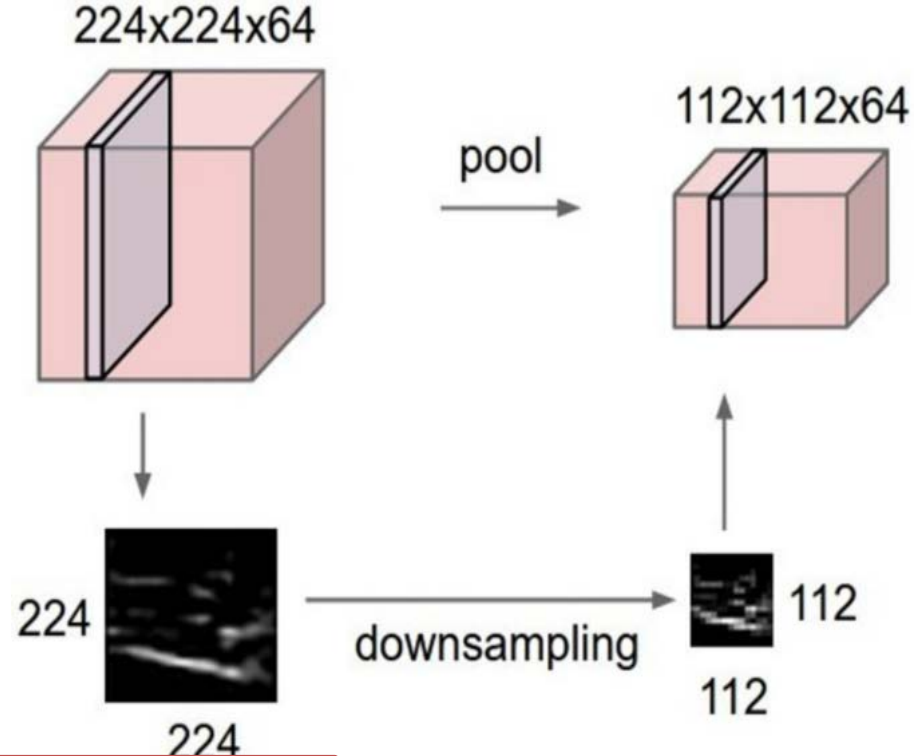
✓ 输出: $W_2 \times H_2 \times D_2$

$$\text{where } W_2 = \frac{W_1 - F}{S} + 1 = \frac{W_1}{F}, \quad H_2 = \frac{H_1 - F}{S} + 1 = \frac{H_1}{F}, \quad D_2 = D_1$$

✓ parameters :

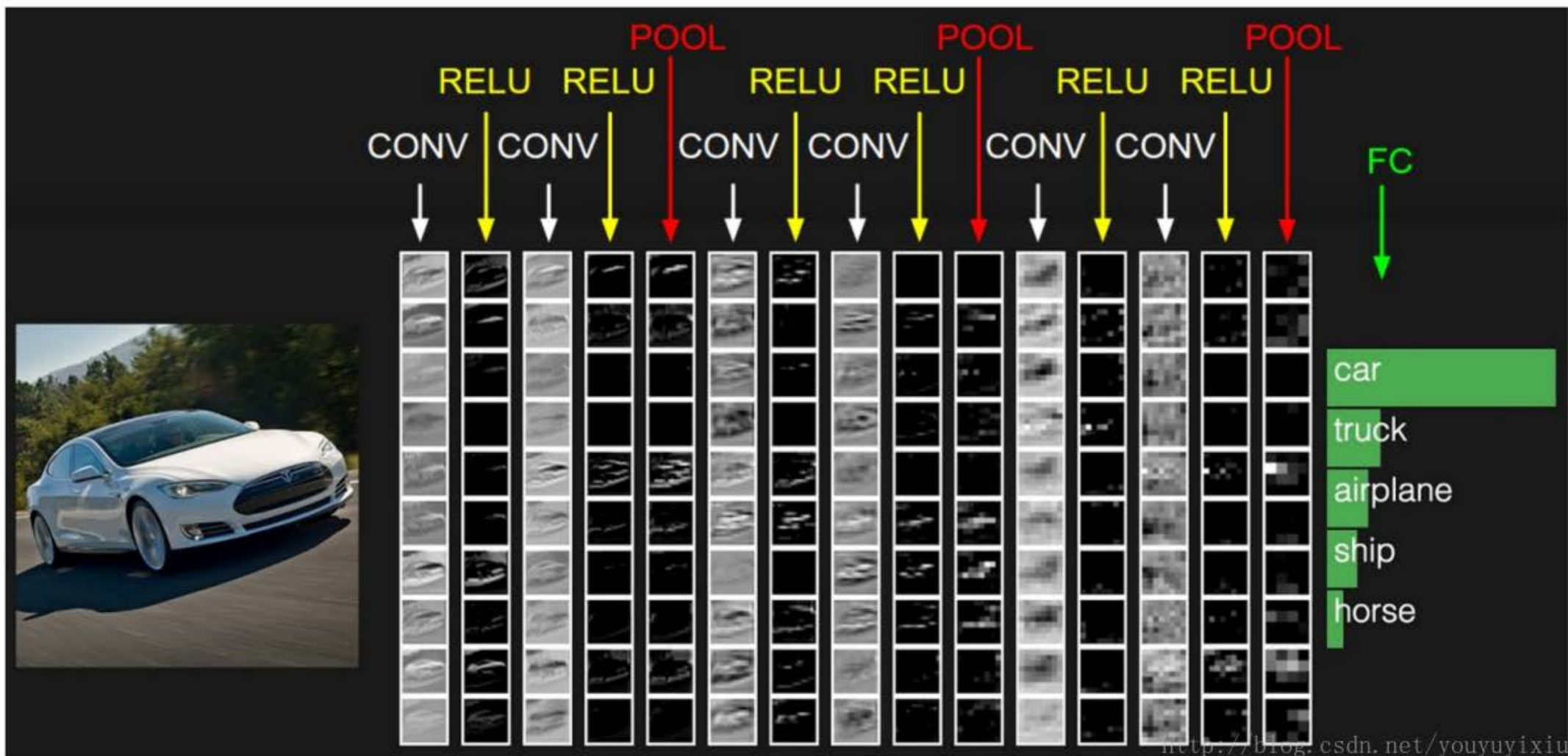
常用池化函数: max-pooling 和 mean-pooling, 没有参数.

但有一些池化方式中是有参数的.



卷积神经网络的整体结构

1. （卷积层+ReLU+ 池化层）的组合多次出现：用于提取特征
2. 多个全连接 或 特殊的CNN结构作为输出层：用作分类器/检测器/分割器



CNN

◆卷积神经网络之训练算法（学习算法）

1. 与一般机器学习算法相同：先定义Loss function，衡量**输出**和**真实结果**之间差距。
2. 找到最小化损失函数的**模型参数**，CNN中用的优化算法是**SGD**（Stochastic Gradient Descent，**随机梯度下降**）。

◆卷积神经网络之优缺点

➤ 优点

- (1) 局部连接，共享卷积核参数，对高维数据处理无压力
- (2) 无需手动选取特征，训练好权重，即得特征，分类效果好

➤ 缺点

- (1) **需要调参，需要大样本量，训练时需要高性能算力**，如GPU。
- (2) 物理含义不明确（也就说，我们并不知道每个卷积层到底提取到的是什么特征，而且神经网络本身就是一种**难以解释**的“**黑箱模型**”）

深度神经网络

◆模型复杂度包括以下两方面：

- **Model width**（模型**宽度**）：隐层中神经元的数目，有最多神经元那个隐层的宽度就是模型的宽度。
- **Model depth**（模型**深度**）：隐层的数目+1（输出层）

◆从增加模型复杂度的角度来看，**增加隐层的数目比增加隐层中神经元的数目更有效。**

◆这是因为增加隐含层的数目不仅增加了拥有激活函数的神经元数目，还**增加了激活函数的个数。**

深度神经网络

◆深度神经网络的学习算法中最优秀的是BP算法。但它有缺陷。

➤ **BP算法本身的缺陷**：局部极小、过拟合、梯度消失

◆影响深度学习模型训练的因素：

➤ **BP算法的本身缺陷**

➤ **算力不够**，导致模型训练的过程很慢，效率低下

➤ **数据量不够**，出现过拟合现象（训练误差较小，但测试误差较大）

◆ So researchers are working hard to improve BP algorithm.

5.5 本章小结

1. 感知机与神经网络

- 神经元的数学模型—MP模型包括线性组合和非线性激活函数两部分。
- 感知机的结构与MP模型相同，但有以下**两点不同之处**：
 - ①采用的激活函数不同，MP模型采用阶跃函数，感知机采用sigmoid函数；
 - ②MP模型的参数都是人为设定的，没有“学习”的机制；而感知机中引入了“学习”的概念，参数都是通过学习得到的。
- 感知机是单层人工神经网络，不含隐藏层；而多层人工神经网络包含至少一个隐藏层。
- 根据**神经元之间的连接范围**，多层人工神经网络分为**全连接神经网络**和**部分连接神经网络**。
- 根据网络层之间的连接方式，多层人工神经网络分为前馈神经网络和反馈神经网络。

5.5 本章小结

2. BP神经网络及其学习算法

- BP神经网络是一种多层前馈神经网络，由于前馈神经网络大多采用反向传播学习算法进行训练模型，故被称为**BP神经网络**。
- 神经网络的学习是指利用训练数据集不断地修改神经网络的所有连接权值和偏置值，使神经网络的实际输出尽可能逼近真实值（Ground Truth）。可见，**BP学习算法是一种有监督学习**。
- BP学习算法包括正向传播和反向传播两个阶段。

5.5 本章小结

3. 卷积神经网络

- 卷积神经网络是一种特殊的多层前馈神经网络，常用于监督学习。
- **卷积神经网络结构**包括一个输入层、多个“卷积层+ReLU层+池化层（池化层有时可以省略）”组合块、多个连续的全连接层（中间不加激活函数）、一个采用softmax函数的输出层。
- 卷积运算前后特征图尺寸的变化可通过公式计算。
- **卷积神经网络具有局部连接和权值共享的特点**。每个卷积层可以有多个滤波器，一个滤波器就是一个神经元，一个滤波器可以包含多个卷积核，一个滤波器内的多个卷积核共用一个偏置值。
- **池化层的作用**是保持尺度不变性和降低特征维度。
- **激活函数**必须是非线性的，目的是大幅度提升深度神经网络的表达能力，用以逼近任何函数，提供非线性的建模能力。每个卷积层后面都有一个激活函数。