计算机模式识别与机器学习

——人工神经网络与深度学习

主讲：图像处理与模式识别研究所

赵群飞

邮　　箱：zhaoqf@sjtu.edu.cn

办 公 室：电院 **2-441**

电　　话：**13918191860**

- 本节学习目标

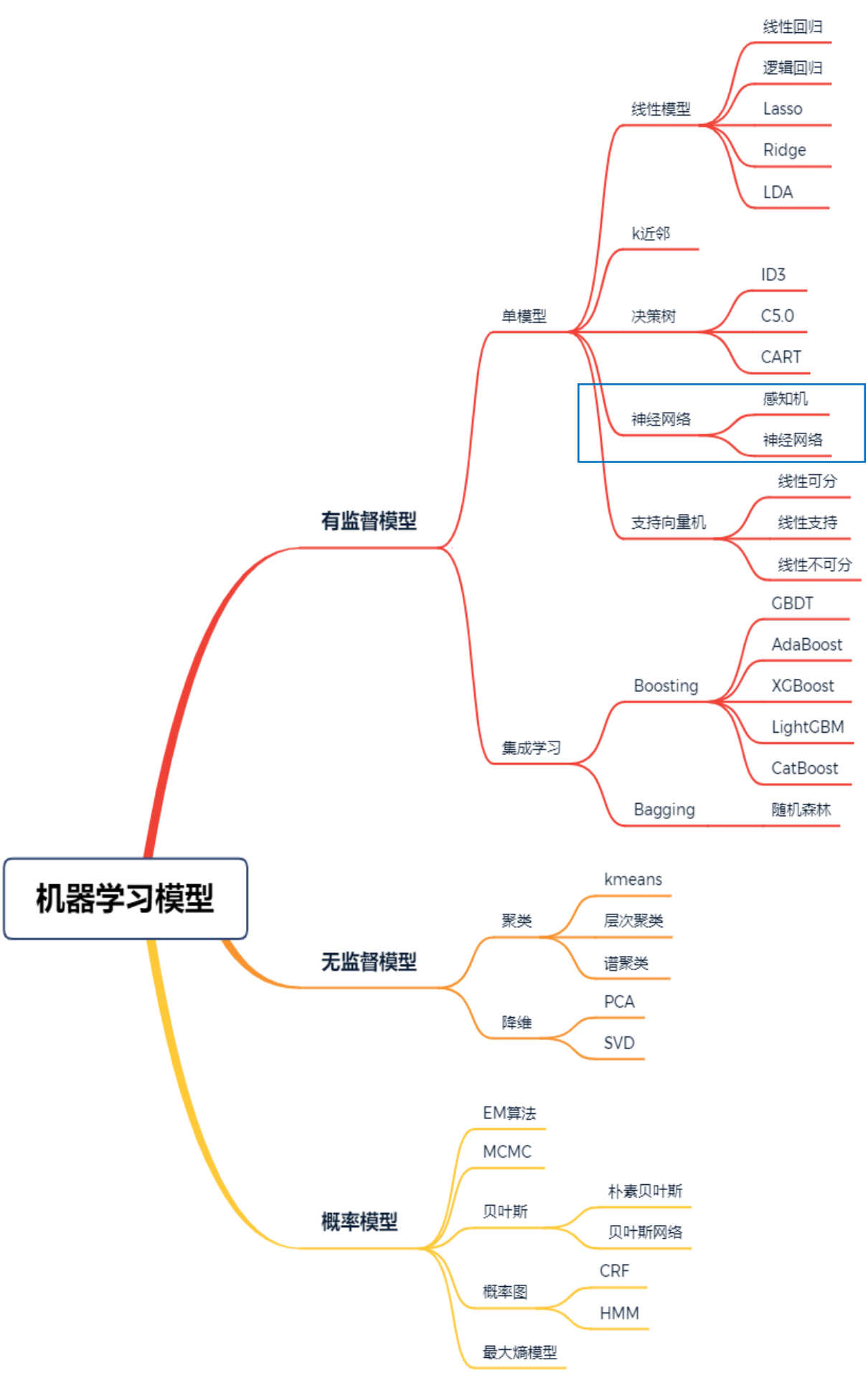
  ✓ 掌握感知机模型和学习算法

  ✓ 掌握多层神经网络模型和误差反向传播训练算法

  ✓ 理解深度神经网络的典型挑战问题

  ✓ 了解一些常见的深度神经网络

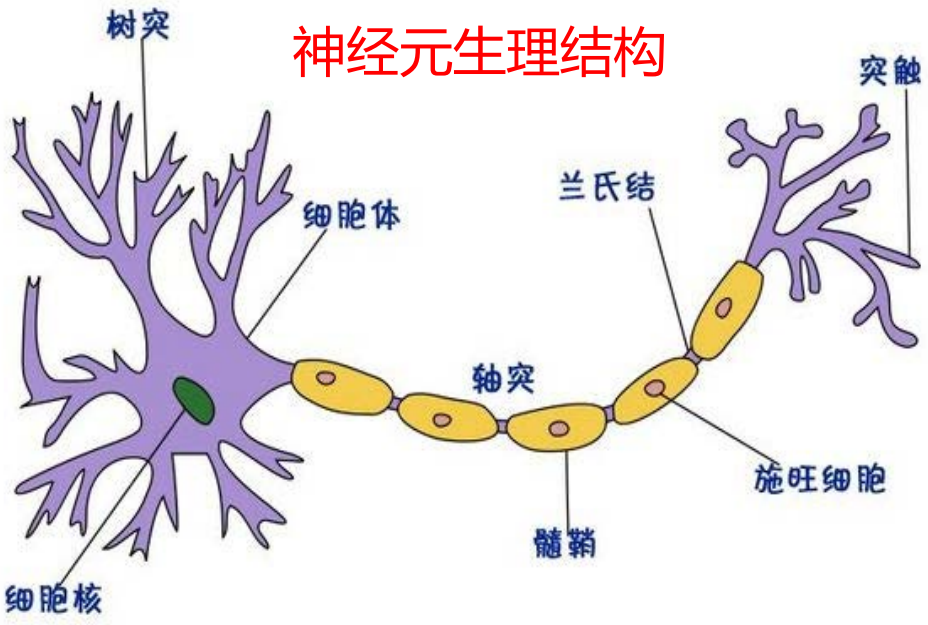
机器学习模型


机器学习模型
有监督模型
单模型
线性模型
线性回归
逻辑回归
Lasso
Ridge
LDA
k近邻
决策树
ID3
C5.0
CART
神经网络
感知机
神经网络
支持向量机
线性可分
线性支持
线性不可分
集成学习
Boosting
GBDT
AdaBoost
XGBoost
LightGBM
CatBoost
Bagging
随机森林
无监督模型
聚类
kmeans
层次聚类
谱聚类
降维
PCA
SVD
概率模型
EM算法
MCMC
贝叶斯
朴素贝叶斯
贝叶斯网络
概率图
CRF
HMM
最大熵模型

人工神经网络发展历史

- **神经元又称神经细胞**，是构成生物神经系统最基本的结构。
- 根据神经元的机能分类：感觉(传入)神经元、运动(传出)神经元和联络神经元。
- 各种神经元，可按功能分区:输入(感受)区、整合(触发冲动)区、冲动传导区、输出(分泌)区。

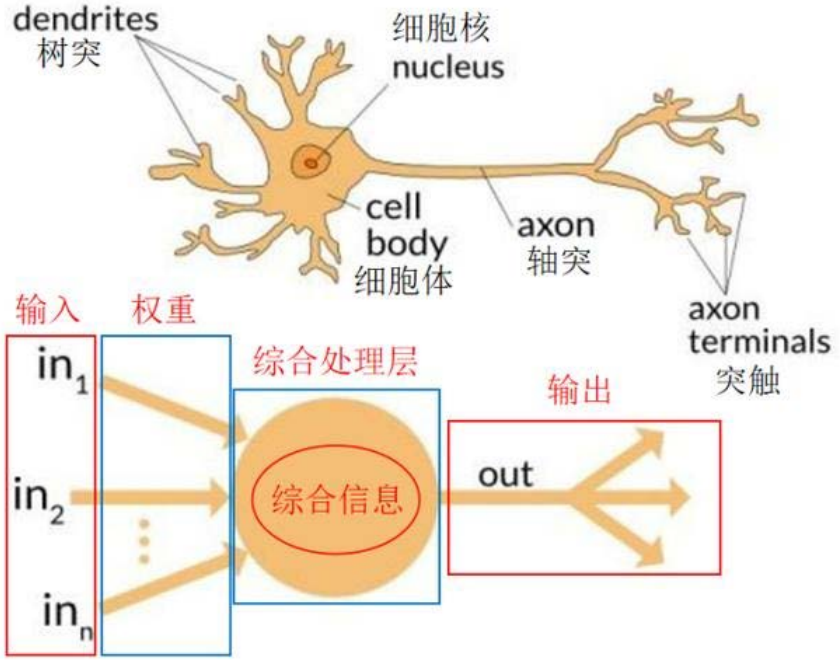
神经元生理结构



- 人类中枢神经系统中约含1000亿个神经元，仅大脑皮层中就约有140亿。
- 神经元互相联结形成**神经网络**，人体通过感觉器官和神经接受来自身体内外的各种信息，传递至中枢神经系统内，经过对信息的分析和综合，再通过运动神经发出控制信息，以此来实现机体与内外环境的联系，协调全身的各种机能活动。

1943年，美国心理学家McCulloch和逻辑学家Pitts建立神经网络的数学模型，即MP模型。

如右图，是一个包含输入，输出与计算处理功能的模型。输入可以类比为神经元的树突，而输出可以类比为神经元的轴突，计算则可以类比为细胞核。

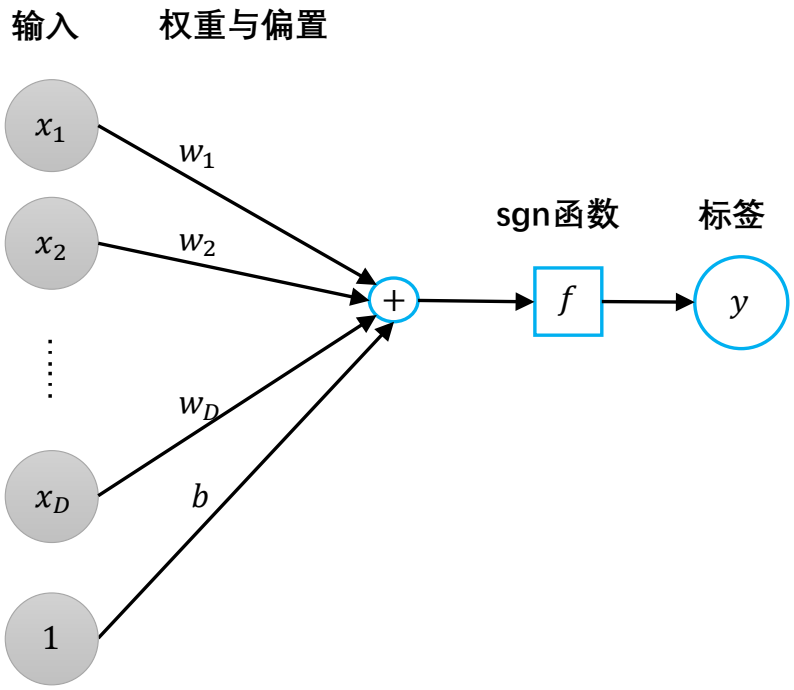


神经元数学模型

- MP模型奠定了人工神经网络的基础

目录

1958年，美国科学家Rosenblatt提出了由两层神经元组成的神经网络，命名为“感知机”（Perceptron）。是当时首个可以学习的人工神经网络ANN。与神经元模型不同，感知器中的权值是通过训练得到的。因此，感知机类似一个逻辑回归模型，可以做线性分类任务。



假设输入样本 $\mathbf{x} \in R^D$，标签 $y \in \{0,1\}$，感知机要拟合一个函数f，满足

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w}^T\mathbf{x} + b) = \begin{cases} 1 & \mathbf{w}^T\mathbf{x} + b > 0, \\ 0 & \text{otherwise}, \end{cases}$$

其中，$\mathbf{w} \in R^D$是实值权重向量，$b \in R$是偏置项，$sgn(\cdot)$ 是符号函数：

$$\text{sgn}(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} > 0, \\ 0 & \mathbf{x} \le 0. \end{cases}$$

- 给定一个有N个样本的训练集 $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$，其中 $\mathbf{x}_i \in R^D$，$y_i \in \{0,1\}$。
- 感知机学习是要根据训练集求解出**模型参数**$\mathbf{w}, b$。
- 感知机的**损失函数**是所有错分类样本到分类超平面的距离和，其**优化目标**为

$$\arg\min_{\mathbf{w},b} L(\mathbf{w},b) = -\frac{1}{\|\mathbf{w}\|}\sum_{i=1}^{N} I(y_i\ (\mathbf{w}^T\mathbf{x}_i+b) < 0) y_i\ (\mathbf{w}^T\mathbf{x}_i+b)$$

其中$I(\cdot)$是单位函数，在$(\cdot)$的条件满足时取值为1，否则为0.

由于||w||的大小并不影响分类结果，可得优化目标

$$\arg\min_{\mathbf{w},b} L(\mathbf{w},b) = -\sum_{i=1}^{N} I(y_i\ (\mathbf{w}^T\mathbf{x}_i+b) < 0) y_i\ (\mathbf{w}^T\mathbf{x}_i+b)$$

- 感知机使用随机梯度下降的方法来学习模型的参数，其中单个样本$(\mathbf{x}_i, y_i)$的损失函数$L_i(\mathbf{w}, b)$关于模型参数$\mathbf{w}, b$的梯度为

$$\frac{\partial L_i(\mathbf{w},b)}{\partial \mathbf{w}} = -I(y_i\ (\mathbf{w}^T\mathbf{x}_i+b) < 0) y_i \mathbf{x}_i,$$

$$\frac{\partial L_i(\mathbf{w},b)}{\partial b} = -I(y_i\ (\mathbf{w}^T\mathbf{x}_i+b) < 0) y_i.$$

  - 感知机把输入经过权重向量转化为输出，是最简单的**前馈ANN**，也称为**单层ANN**。

算法 8-1 感知机学习算法

输入：训练数据 $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$

1：随机初始化模型参数 $\mathbf{w}, b$；

2：REPEAT

3：从训练集中选取训练数据 $(\mathbf{x}_i, y_i)$；

4：如果 $y_i(\mathbf{w}^T \mathbf{x}_i + b) \le 0$（表示该点被错分），则更新模型参数：

$$\mathbf{w}_{new} = \mathbf{w}_{old} + \eta y_i \mathbf{x}_i,$$
$$b_{new} = b_{old} + \eta y_i,$$

其中 $\eta \in (0,1]$ 是学习率；

5：UNTIL 训练集中没有错分的数据点

输出：参数 $\mathbf{w}$， b

- **若线性可分，算法收敛，无穷多解，取决于初始条件和迭代过程。**

目录

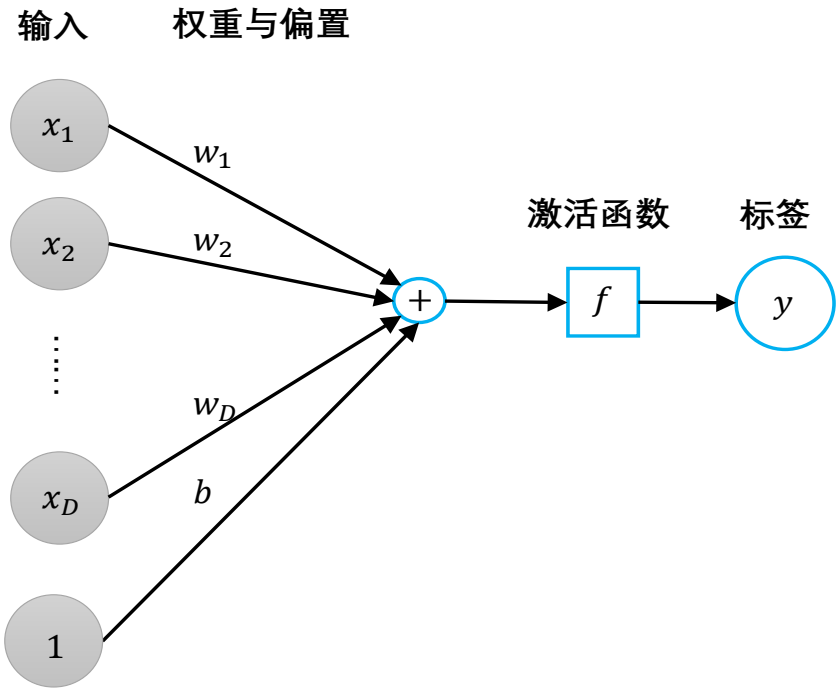
- 感知机只包含一层输入和一层输出，所进行的变换十分有限。训练结果的好坏取决于输入向量的质量，即选的特征是否重要。
- 在输入向量无法改变的前提下，要提升分类的性能，需要使用多层的网络，而且每一层都是非线性映射。
- 多层网络中，输入层和输出层之间的隐含层，可以为下一层提取重要的特征，克服感知机过于简单的局限性。
- 受生物神经系统的启发，多层神经网络使用大量简单的计算单元“神经元”互相连接形成网络，隐含层中的每一个神经元的输入，是其前一层所有神经元输出的加权和，而其输出又作为后一层神经元的输入。
- 多层神经网络模型的目的是经过学习训练，拟合一个更复杂的非线性函数。

- 神经元

神经元以$\boldsymbol{x} \in R^D$为输入，其输出为

$$h_{\mathbf{w},b}(\mathbf{x}) = f(\sum_{d=1}^{D} w_d x_d + b) = f(\mathbf{w}^T \mathbf{x}),$$

其中，$\boldsymbol{w}$和b分别为权重和偏置项，$f(\cdot)$称为激活函数。



常用的激活函数

- sigmoid函数：

$$f(z)=\frac{1}{1+e^{-z}},$$

值域范围为(0,1)。

sigmoid函数的导数为

$$f'(z)=f(z)(1-f(z)).$$

- 如果激活函数选为sigmoid函数，神经元正好对应于逻辑回归定义的输入和输出间的映射。

常用的激活函数

- 双曲正切(tanh)函数:

$$f(z)=\tanh(z)=\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}},$$

  - 值域范围为(-1,1) 。
  - 双曲正切函数的导数为

$$f'(z)=1-(f(z))^{2}.$$

常用的激活函数

- 修正线性单元（ReLU）：

$$f(z) = \max(0, z).$$

- 值域范围为$(0, +\infty)$。
- 修正线性单元的导数为

$$f'(z) = \begin{cases} 1 & z > 0, \\ 0 & z < 0. \end{cases}$$

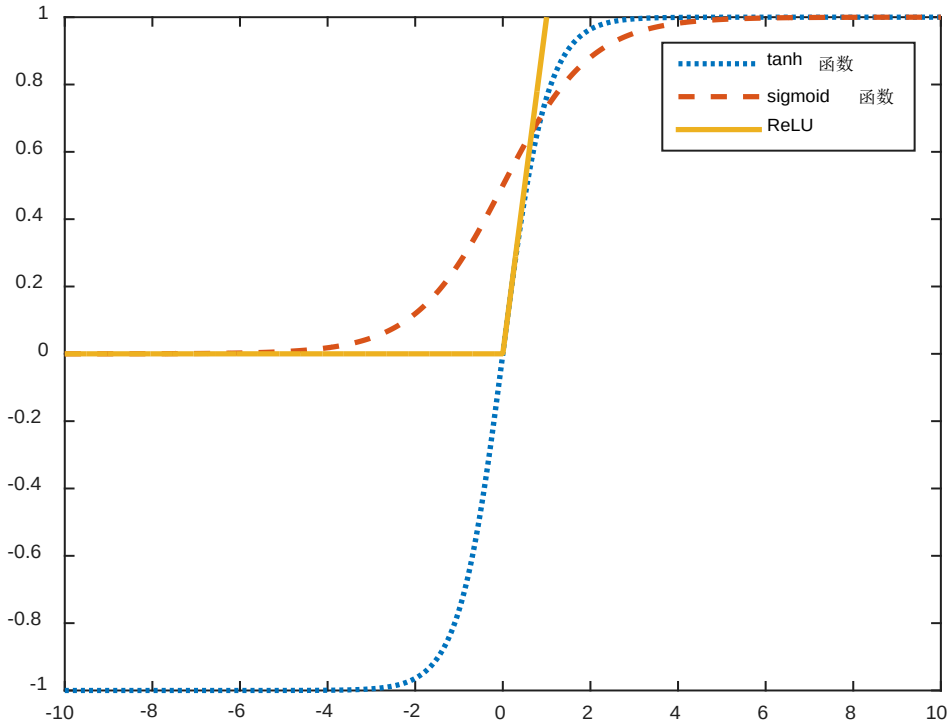


图 三种激活函数的形态

激活函数	函数	导数
Logistic 函数	$f(x)=\frac{1}{1+\exp(-x)}$	$f'(x)=f(x)\big(1-f(x)\big)$
Tanh 函数	$f(x)=\frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$	$f'(x)=1-f(x)^2$
ReLU 函数	$f(x)=\max(0,x)$	$f'(x)=I(x>0)$
ELU 函数	$f(x)=\max(0,x)+\min\Big(0,\gamma\big(\exp(x)-1\big)\Big)$	$f'(x)=I(x>0)+I(x\leq 0)\cdot\gamma\exp(x)$
SoftPlus 函数	$f(x)=\log\big(1+\exp(x)\big)$	$f'(x)=\frac{1}{1+\exp(-x)}$

●多层神经网络

- 多层神经网络也叫多层感知机，是一种基本的前馈神经网络。
- 多个神经元分层相连，形成一个输入层、一个输出层，一个或多个隐含层，把输入层记为第0层，连接输入层的第一个隐含层记为网络的第1层，以此向后排序。
- 前一层所有神经元输出的加权和作为后一层每个神经元的输入。每个神经元都看作是一个把输入映射到输出的非线性函数，而多层神经元网络是多个非线性函数的复合。
- 多层神经网络经过多次非线性变换，可以解决线性不可分问题等复杂函数的拟合难题。

表 8-1　多层神经网络中的符号表示

符　　号	说　　明
$X=\{\boldsymbol{x}^1,\boldsymbol{x}^2,\cdots,\boldsymbol{x}^N\}$	数据集输入
$Y=\{\boldsymbol{y}^1,\boldsymbol{y}^2,\cdots,\boldsymbol{y}^N\}$	数据集对应标签
N	数据集样本数量
D	数据集输入特征维度
n_ℓ	第ℓ层的节点个数
L	神经网络的层数(不包括输入层)
$\boldsymbol{W}^{(\ell)}\in\mathbb{R}^{n_{\ell-1}\times n_\ell}$	第ℓ层的权重矩阵,由第$\ell-1$层的节点指向第ℓ层的节点
$w_{ij}^{(\ell)}$	第$\ell-1$层的第i个节点到第ℓ层的第j个节点之间的连接权重
$\boldsymbol{b}^{(\ell)}=[b_1^{(\ell)},b_2^{(\ell)},\cdots,b_{n_\ell}^{(\ell)}]^{\mathrm{T}}$	第ℓ层的偏置项,由第$\ell-1$层指向第ℓ层
$\boldsymbol{z}^{(\ell)}=[z_1^{(\ell)},z_2^{(\ell)},\cdots,z_{n_\ell}^{(\ell)}]^{\mathrm{T}}$	第ℓ层的节点激活之前的状态
$\boldsymbol{a}^{(\ell)}=[a_1^{(\ell)},a_2^{(\ell)},\cdots,a_{n_\ell}^{(\ell)}]^{\mathrm{T}}$	第ℓ层的节点激活之后的状态,作为第$\ell+1$层的输入

- 当$\ell=0$, $\boldsymbol{a}^{(0)}=\boldsymbol{x}$, 即 $a_i^{(0)}=x_i$;
- 当 $\ell=\mathrm{L}$, $a_i^{(L)}$ 是神经网络的第i个输出，$\boldsymbol{h}_{w,b}(\boldsymbol{x})=\boldsymbol{a}^{(\mathrm{L})}$;
- 当 $1\leq\ell\leq\mathrm{L}$，第 ℓ 层第j个节点激活之前的状态

$$z_j^{(\ell)}=\sum_1^{n_{\ell-1}} w_{ij}^{(\ell)} a_i^{(\ell-1)}+b_i^{(\ell)}$$

激活之后的状态为 $a_j^{(\ell)}=f\left(z_j^{(\ell)}\right)=f(\sum_1^{n_{\ell-1}} w_{ij}^{(\ell)} a_i^{(\ell-1)}+b_i^{(\ell)})$

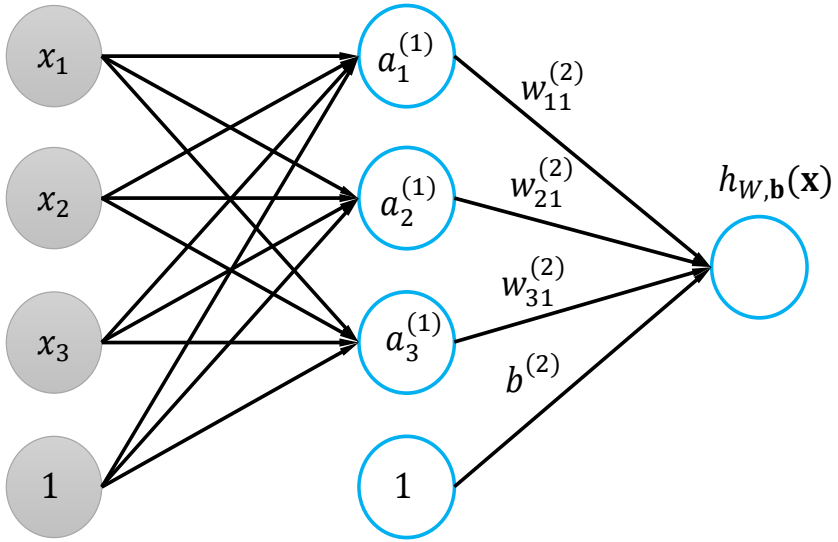
本层所有节点激活后的状态向量为 $\boldsymbol{a}^{(\ell)}=(a_1^{(\ell)},a_2^{(\ell)},\ldots,a_{n_\ell}^{(\ell)})$;

- 第$\ell+1$层激活后的状态向量就可以用下式计算：

$$\boldsymbol{a}^{(\ell+1)}=f\left(\boldsymbol{z}^{(\ell+1)}\right)=f([\boldsymbol{W}^{(\ell+1)}]^{\boldsymbol{T}}\boldsymbol{a}^{(\ell)}+\boldsymbol{b}^{(\ell)})$$

➢ **从第1层到第L层，逐层计算激活前和激活后状态的步骤称为正向传播。**

两层正向传播网络



$$
\begin{cases}
a_1^{(1)} = f(w_{11}^{(1)}x_1 + w_{21}^{(1)}x_2 + w_{31}^{(1)}x_3 + b_1^{(1)}), \\
a_2^{(1)} = f(w_{12}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{32}^{(1)}x_3 + b_2^{(1)}), \\
a_3^{(1)} = f(w_{13}^{(1)}x_1 + w_{23}^{(1)}x_2 + w_{33}^{(1)}x_3 + b_3^{(1)}), \\
h_{w,\mathbf{b}}(\mathbf{x}) = a_1^{(2)} = f(w_{11}^{(2)}a_1^{(1)} + w_{21}^{(2)}a_2^{(1)} + w_{31}^{(2)}a_3^{(1)} + b^{(2)}),
\end{cases}
$$

- 反向传播算法

设训练集包括N个数据点$\{(\mathbf{x}^1,\mathbf{y}^1),(\mathbf{x}^2,\mathbf{y}^2),\cdots,(\mathbf{x}^N,\mathbf{y}^N)\}$，

$\boldsymbol{W}=\{\boldsymbol{W}^{\{\ell\}}\}_{\ell=1}^{L}$，$\mathbf{b}=\{\mathbf{b}^{\{\ell\}}\}_{\ell=1}^{L}$表示神经网络中的所有参数，

$J(\boldsymbol{W},\mathbf{b};\mathbf{x}^n,\mathbf{y}^n)$表示神经网络关于单个数据点$(\mathbf{x}^n,\mathbf{y}^n)$的损失函数。

- 对于回归任务，神经网络关于单个数据点$(\mathbf{x}^n,\mathbf{y}^n)$的损失函数为

$$J(W,\mathbf{b};\mathbf{x}^n,\mathbf{y}^n)=\frac{1}{2}\|\mathbf{h}_{W,\mathbf{b}}(\mathbf{x}^n)-\mathbf{y}^n\|^2.$$

- 对于多个独立的二类分类任务，假设每个数据点的标签是元素为0或1的向量，神经网络关于单个数据点$(\mathbf{x}^n,\mathbf{y}^n)$的损失函数为

$$J(W,\mathbf{b};\mathbf{x}^n,\mathbf{y}^n)=-(\mathbf{y}^n)^T\ln\mathbf{h}_{W,\mathbf{b}}(\mathbf{x}^n)-(\mathbf{1}-\mathbf{y}^n)^T\ln(\mathbf{1}-\mathbf{h}_{W,\mathbf{b}}(\mathbf{x}^n)),$$

- 对于多类分类任务，假设每个数据点的标签是指示类别的one-hot向量，神经网络关于单个数据点$(\mathbf{x}^n, \mathbf{y}^n)$的损失函数为

$$J(W, \mathbf{b}; \mathbf{x}^n, \mathbf{y}^n) = -(\mathbf{y}^n)^{\mathrm{T}} \ln \mathbf{h}_{W,\mathbf{b}}(\mathbf{x}^n).$$

- 神经网络的总损失等于所有单个数据点损失的平均。为了防止过拟合，通常还会在损失目标中增加对权重参数的正则化约束。因此，对于整个数据集的总损失函数为

$$J(W, \mathbf{b}) = \frac{1}{N}\sum_{n=1}^{N} J(W, \mathbf{b}; \mathbf{x}^n, \mathbf{y}^n) + \frac{\lambda}{2}\sum_{l=1}^{L}\sum_{i=1}^{n_{l-1}}\sum_{j=1}^{n_l}(w_{ij}^{(l)})^2.$$

- 式中第一项是训练误差项，第二项是正则化项（也叫权重衰减项，一般不用于偏置项b），其作用是防止过拟合。权重衰减参数λ控制两项的重要性。

给定优化目标$J(W,\mathbf{b})$后，梯度下降方法训练神经网络，每次迭代按如下公式分别更新参数W和$\mathbf{b}$：

$$w_{ij}^{(l)} = w_{ij}^{(l)} - \alpha \frac{\partial}{\partial w_{ij}^{(l)}} J(W,\mathbf{b}),$$

$$b_{j}^{(l)} = b_{j}^{(l)} - \alpha \frac{\partial}{\partial b_{j}^{(l)}} J(W,\mathbf{b}),$$

$\alpha \geqq 0$ 是学习率。

在得到单个数据点$(\mathbf{x}^n,\mathbf{y}^n)$上的损失函数的偏导数 $\partial J(W,\mathbf{b};\mathbf{x}^n,\mathbf{y}^n)/\partial w_{ij}^{(l)}$ 和 $\partial J(W,\mathbf{b};\mathbf{x}^n,\mathbf{y}^n)/\partial b_{j}^{(l)}$ 后，整体损失函数$J(W,\mathbf{b})$的偏导数计算如下：

$$\frac{\partial}{\partial w_{ij}^{(l)}} J(W,\mathbf{b}) = \frac{1}{N}\sum_{n=1}^{N} \frac{\partial}{\partial w_{ij}^{(l)}} J(W,\mathbf{b};\mathbf{x}^n,\mathbf{y}^n) + \lambda w_{ij}^{(l)},$$

$$\frac{\partial}{\partial b_{j}^{(l)}} J(W,\mathbf{b}) = \frac{1}{N}\sum_{n=1}^{N} \frac{\partial}{\partial b_{j}^{(l)}} J(W,\mathbf{b};\mathbf{x}^n,\mathbf{y}^n).$$

将单个样本的损失$J(W,\mathbf{b};\mathbf{x}^n,\mathbf{y}^n)$简记为$J^n(W,\mathbf{b})$。

引入中间变量$z^{(\ell)}$，表示第ℓ层的节点激活之前的状态。根据定义

$$z_j^{(l)} = \sum_{i=1}^{n_{l-1}} w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)}, a_i^{(l-1)} = f(z_i^{(l-1)}),$$

那么，可以得到关于$\boldsymbol{z}^{(\ell)}$各分量的偏导

$$\frac{\partial J^n(W,\mathbf{b})}{\partial z_j^{(l)}} = \frac{\partial J^n(W,\mathbf{b})}{\partial z_1^{(l+1)}} \frac{\partial z_1^{(l+1)}}{\partial z_j^{(l)}} + \frac{\partial J^n(W,\mathbf{b})}{\partial z_2^{(l+1)}} \frac{\partial z_2^{(l+1)}}{\partial z_j^{(l)}} + \ldots + \frac{\partial J^n(W,\mathbf{b})}{\partial z_{n_{l+1}}^{(l+1)}} \frac{\partial z_{n_{l+1}}^{(l+1)}}{\partial z_j^{(l)}}$$

$$= \frac{\partial J^n(W,\mathbf{b})}{\partial z_1^{(l+1)}} f'(z_j^{(l)}) w_{j1}^{(l+1)} + \ldots + \frac{\partial J^n(W,\mathbf{b})}{\partial z_{n_{l+1}}^{(l+1)}} f'(z_j^{(l)}) w_{jn_{l+1}}^{(l+1)}$$

$$= \sum_{k=1}^{n_{l+1}} \frac{\partial J^n(W,\mathbf{b})}{\partial z_k^{(l+1)}} f'(z_j^{(l)}) w_{jk}^{(l+1)}.$$

将 $\partial J^n(W,\mathbf{b})/\partial z_j^{(l)}$ 记为 $\delta_j^{(l)}$，其含义是第ℓ层第j个节点对输出误差 $J^n(W,\mathbf{b})$ 的影响程度，也称为节点残差。节点残差的递推关系为

$$\delta_j^{(l)}=\left(\sum_{k=1}^{n_{l+1}} w_{jk}^{(l+1)}\delta_j^{(l+1)}\right) f'(z_j^{(l)}),\ \ell=L-1,L-2,\ldots,1.$$

起始值 $\delta_j^{(l)}$ 可以通过损失函数直接计算得出

$$\delta_j^{(L)}=\frac{\partial J(W,\mathbf{b};\mathbf{x}^n,\mathbf{y}^n)}{\partial z_j^{(L)}}.$$

计算出所需的针对单个数据点的损失函数的偏导数

$$\frac{\partial}{\partial w_{ij}^{(l)}}J^n(W,\mathbf{b})=a_i^{(l-1)}\delta_j^{(l)},\ \frac{\partial}{\partial b_j^{(l)}}J^n(W,\mathbf{b})=\delta_j^{(l)}.$$

然后计算总损失函数关于权重矩阵$\boldsymbol{W}^{(\ell)}$和偏置向量 $\boldsymbol{b}^{(\ell)}$的梯度：

$$\Delta\boldsymbol{W}^{(\ell)}=\frac{1}{N}\sum_1^N\frac{\partial J^n(\boldsymbol{W},\boldsymbol{b})}{\partial\boldsymbol{W}^{(\ell)}}+\lambda\boldsymbol{W}^{(\ell)}\qquad\Delta\boldsymbol{b}^{(\ell)}=\frac{1}{N}\sum_1^N\frac{\partial J^n(\boldsymbol{W},\boldsymbol{b})}{\partial\boldsymbol{b}^{(\ell)}}$$

基于反向传播的多层神经网络训练算法

输入：训练数据集 $\{(\mathbf{x}^1,\mathbf{y}^1),(\mathbf{x}^2,\mathbf{y}^2),\cdots,(\mathbf{x}^N,\mathbf{y}^N)\}$

1：初始化参数 $\boldsymbol{W}=\{\boldsymbol{W}^{\{\ell\}}\}_{\ell=1}^{L}$，$\boldsymbol{b}=\{\boldsymbol{b}^{\{\ell\}}\}_{\ell=1}^{L}$；

2：REPEAT

3：使用正向传播算法计算每一个样本在网络每一层激活前与后的状态；

4：利用反向传播算法计算参数的梯度，依次计算$\delta_j^{(\mathrm{L})}$，$\delta_j^{(\ell)}$，$\Delta\boldsymbol{W}^{(\ell)}$ 和 $\Delta\boldsymbol{b}^{(\ell)}$，

for ℓ=L-1,L-2,...,1；

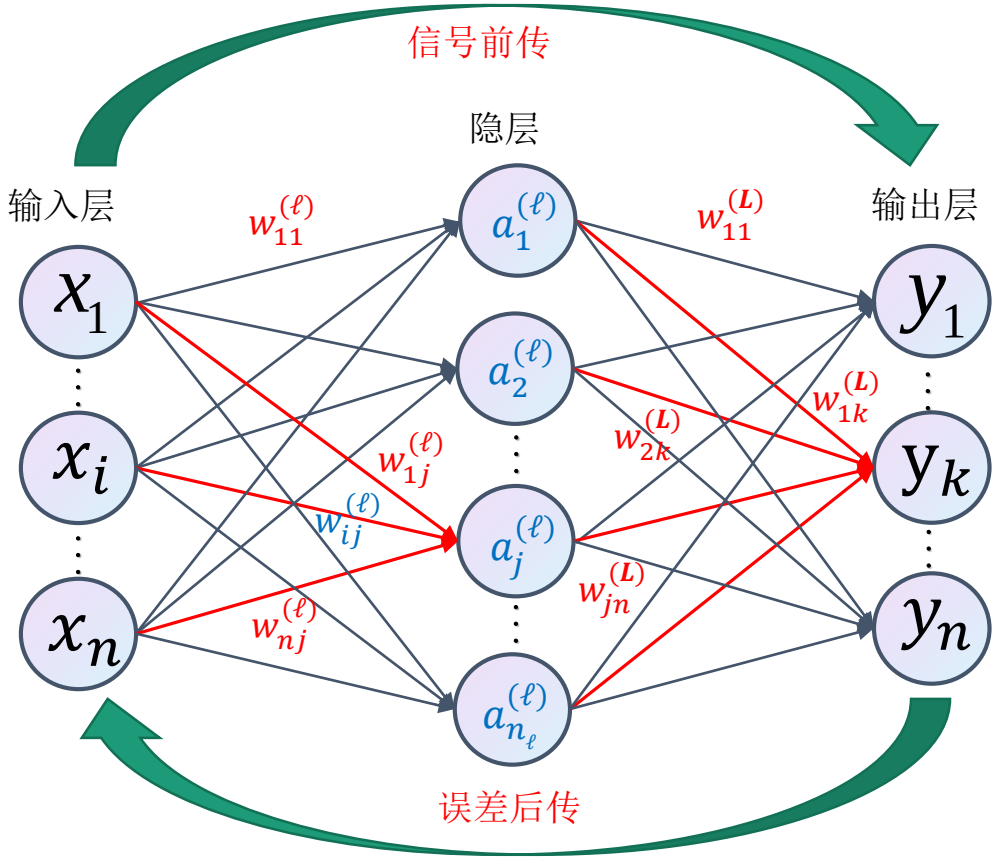
5：更新所有参数：$\boldsymbol{W}^{(\ell)}=\boldsymbol{W}^{(\ell)}-\alpha\Delta\boldsymbol{W}^{(\ell)}$　$\boldsymbol{b}^{(\ell)}=\boldsymbol{b}^{(\ell)}-\alpha\Delta\boldsymbol{b}^{(\ell)}$

6：UNTIL 算法收敛（条件设置为参数或损失函数的变化小于某个很小阈值）

输出：参数 $\{\boldsymbol{W}^{\{\ell\}},\mathbf{b}^{\{\ell\}}\}_{\ell=1}^{L}$

算法流程回味：

1. 将输入样本提供给输入层神经元；
2. 逐层将信号前传至隐层、输出层，产生输出层的结果；
3. 计算输出层误差；
4. 将误差反向传播至隐藏层神经元；
5. 根据隐层神经元对连接权重和阈值进行调整；
6. 上述过程循环进行，直至达到某些停止条件为止。



算法流程回味：

优点：

1.自适应、自主学习。BP可以根据预设能够参数更新规则，通过不断调整神经网络中的参数，已达到最符合期望的输出。

2.拥有很强的非线性映射能力。

3.误差的反向传播采用的是成熟的链式法则，推导过程严谨且科学。

4.算法泛化能力很强。

缺点：

1.BP神经网络参数众多，每次迭代需要更新较多数量的阈值和权值，故收敛速度比较慢。

2.网络中隐层含有的节点数目没有明确的准则，需要不断设置节点数字试凑，根据网络误差结果最终确定隐层节点个数。

3.BP算法是一种速度较快的梯度下降算法，容易陷入局部极小值的问题。

参数的随机初始化

据吴恩达老师讲，如果设所有参数初始值为0，虽然在逻辑回归中行得通，但是对于神经网络来说是行不通的。因为如果我们令所有的初始参数为0（或全部参数都为一样的值），那么这就意味着第二层的所有神经元都会有相同的值。

激活函数的选择

- 在整个神经网络中，隐含层和输出层的神经元的激活函数的选择有一些不同的。
  - 隐含层上的激活函数选择双曲正切函数。
  - 输出层上的激活函数选择
    - ✓ 二分类问题：sigmoid函数
    - ✓ 多分类问题：softmax函数

$$\mathrm{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{l=1}^{k} \exp(z_l)}$$

- 另外，在使用反向传播更新参数权重时，计算过程中会涉及到激活函数的求导计算，所以激活函数的选择将会影响收敛速度，而且使用sigmoid函数的过程本身就会产生如下问题：
  - •容易出现梯度消失
  - •函数输出不是zero-centered
  - •幂运算相对比较耗时

损失函数的选择

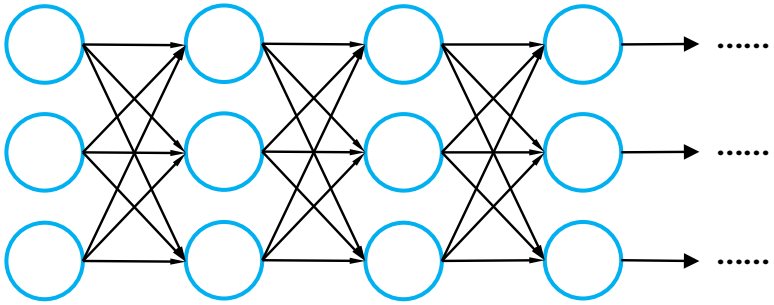
- 根据sklearn官方文档所说，对于分类问题，使用交叉熵（Cross-Entropy）代价函数，对于回归问题，则使用均方误差损失函数。
- 在参数更新的梯度推导中，则以均方误差作为损失函数，而事实上，如果是分类问题的话，使用交叉熵代价函数得到的效果会更好，梯度下降的速度会更快。这是因为交叉熵有一个特性，它会在模型误差较大的时候梯度下降得更快，误差较小的时候梯度下降得更慢。


神经网络
感知机和神经网络
感知机
异或问题与多层感知机
多层前馈神经网络结构
反向传播算法（BP）
前向传播
后向传播
伪代码
算法实现时需要注意的要点
参数的随机初始化
激活函数的选择
隐含层神经元激活函数
sigmoid函数
tanh函数（双曲正切）
输出层神经元激活函数
分类问题
二分类问题
sigmoid函数
多分类问题
softmax函数
回归问题
输出仍然是f(x)，因此激活函数仅仅是一个identity函数
在回归问题中，有隐藏结点激活函数，却没有输出节点激活函数
激活函数更多的选择：ReLU
损失函数的选择
分类问题
交叉熵损失函数
回归问题
均方误差损失函数
实际使用时的其他小技巧
预处理时使用feature scaling
收敛速度
收敛最优点的讨论

目录

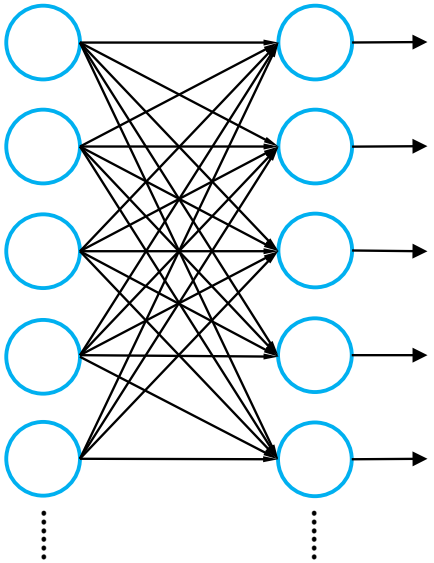
• 浅层与深度神经网络

- 深度神经网络是指隐含层多于一层的神经网络。
- 深度神经网络与浅层神经网络的基本原理是相同的。
- 随着网络层数的增加，遇到许多挑战，需要更好的技术应对，如更巧妙的网络结构设计、更加稳定的优化方法等。
- 给定相同的参数，窄的深度网络性能更优，可用较少的数据量得到更好的拟合。



thin+tall

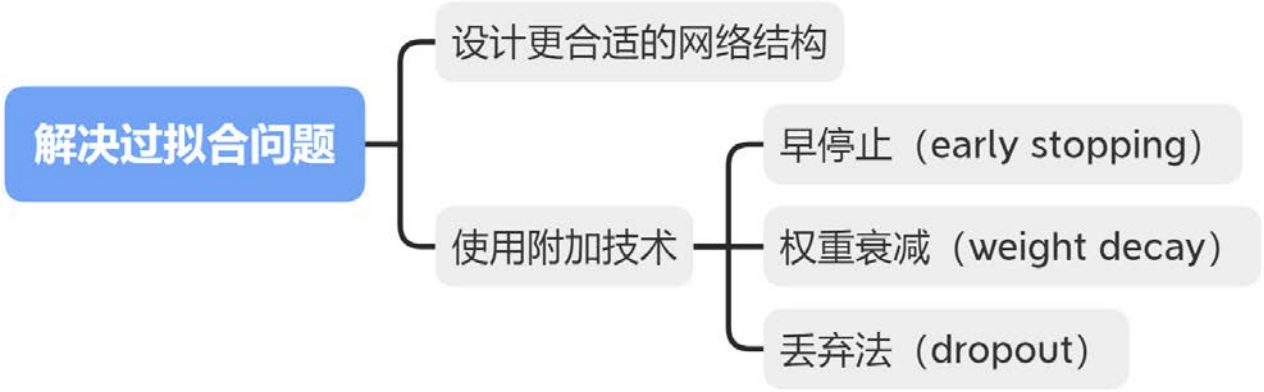
VS



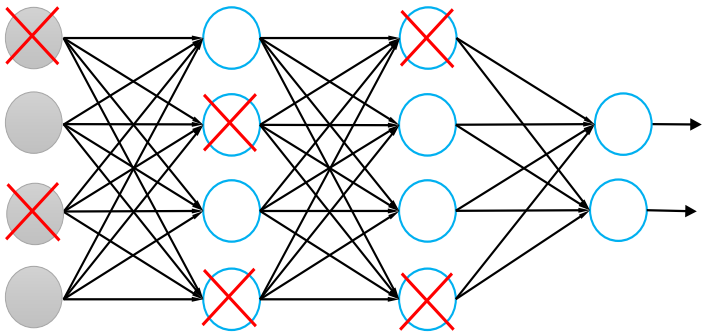
fat+short

- 过拟合问题

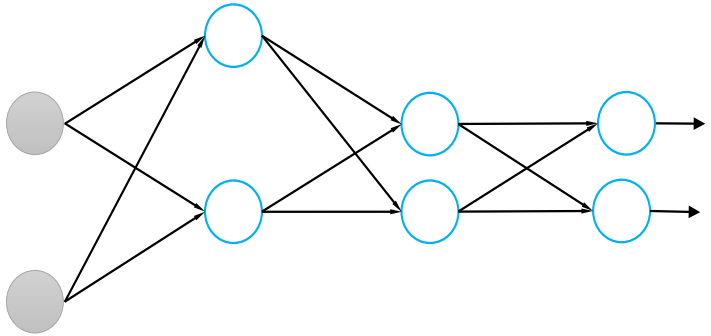
过拟合问题是深度神经网络的主要挑战之一，其主要原因是模型过于复杂或者训练集过少。



- **早停止**是指在模型训练过程中，可通过观察验证集上的预测性能来决定何时停止对参数的优化，从而可以在产生过拟合之前停止训练。
- **权重衰减**是指为了防止得到的权重参数过大，而采取的在每步迭代中少量减少权重的方法。
- **丢弃法**是指在训练过程中，对于网络中的神经单元（包括节点以及与之连接的边），按照一定的概率将其暂时从网络中丢弃。



（a）丢弃情况



（b）丢弃后剩余的网络结构

- 局部极值问题
  - 随机梯度下降

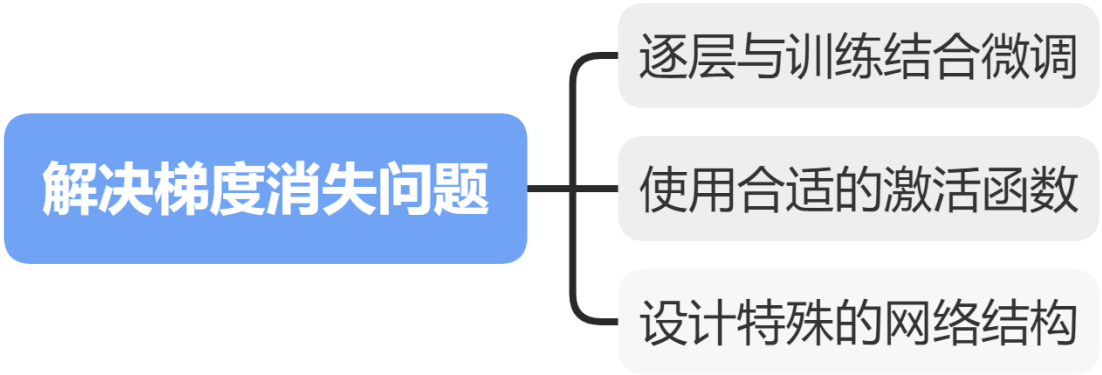
    使用随机梯度下降代替批量的梯度下降，不仅使得优化速度得以提升，而且还可以提高模型的整体性能。性能提高的主要原因是每次用于迭代的随机梯度并非梯度的确切方向，使得模型容易跳出局部极值点。
  - 基于动量的梯度下降

    基于动量的梯度下降的做法是每次进行梯度下降时，在当前梯度方向上增加历史梯度的加权值。动量方法能够使得梯度更新的大小可以根据上一步的梯度进行适当调节，增加跳出局部极值点的几率。
  - 多次随机初始化

    假设损失函数的曲面具有许多局部极值点，多次随机初始化待优化的参数值可以增加离开局部极值的可能性，有助于找到更好的解。

- 梯度消失问题

当使用反向传播方法求解梯度时，使用sigmoid函数或者tanh函数作为激活函数的深度神经网络随着网络层数的增加，从输出层到网络最初几层的反向传播得到的梯度的幅度值可能会急剧增大（**梯度爆炸**）或减小（**梯度消失**）。



逐层与训练结合微调

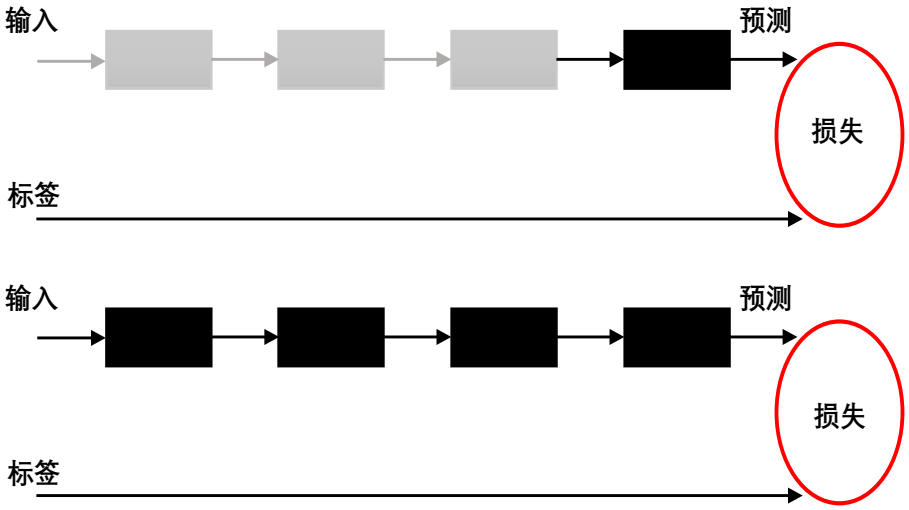


图8-6 逐层预训练加微调方法示意图
(黑色方框表示需要微调的结构。)

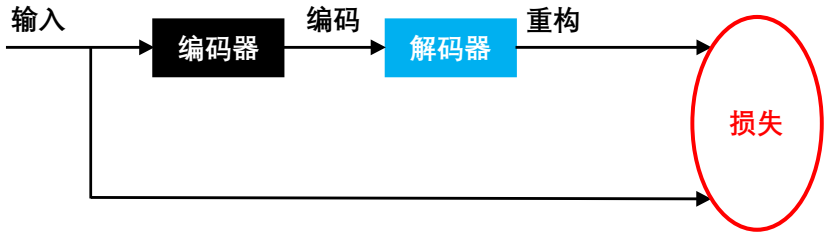


图8-7 预训练网络中的自编码器结构示意图

使用合适的激活函数

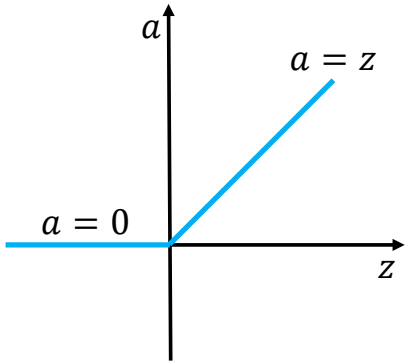


图8-8 ReLU函数

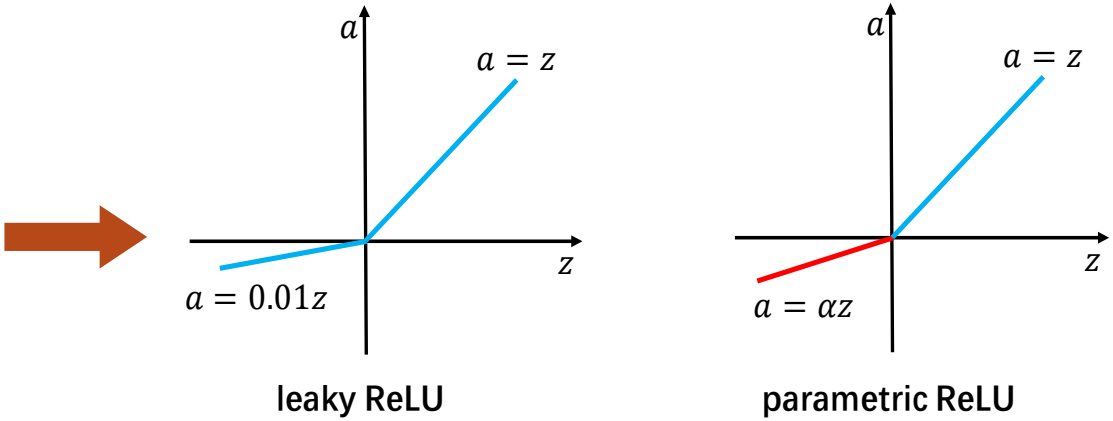


图8-9 ReLU函数的变体

使用合适的激活函数

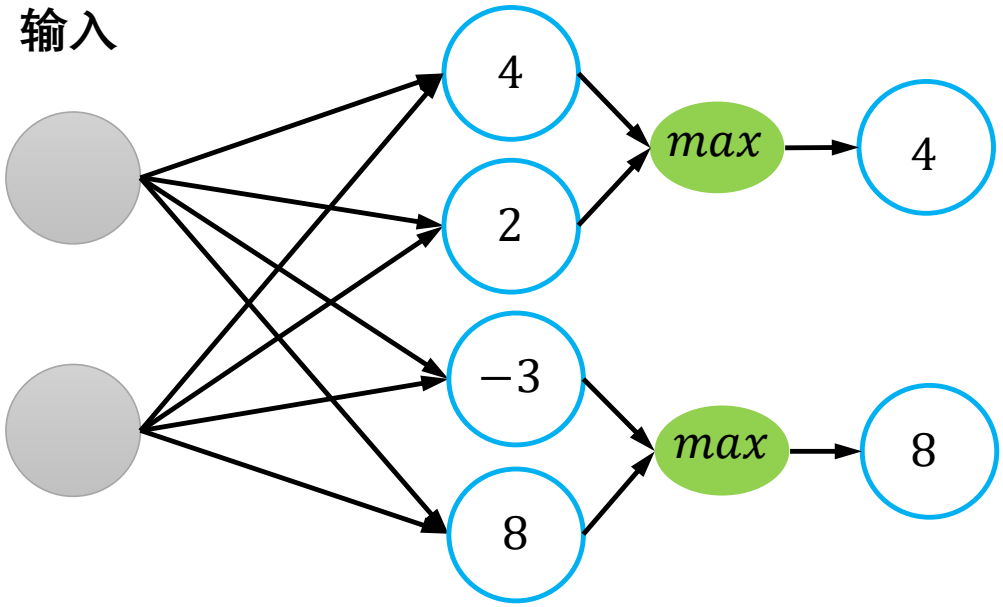


图8-10 Maxout函数原理示意图
(将每层的节点分组，并选择组内最大数的作为下一层的输入。)

目录

常用的深度神经网络并不局限于全连接网络。

- 自编码网络

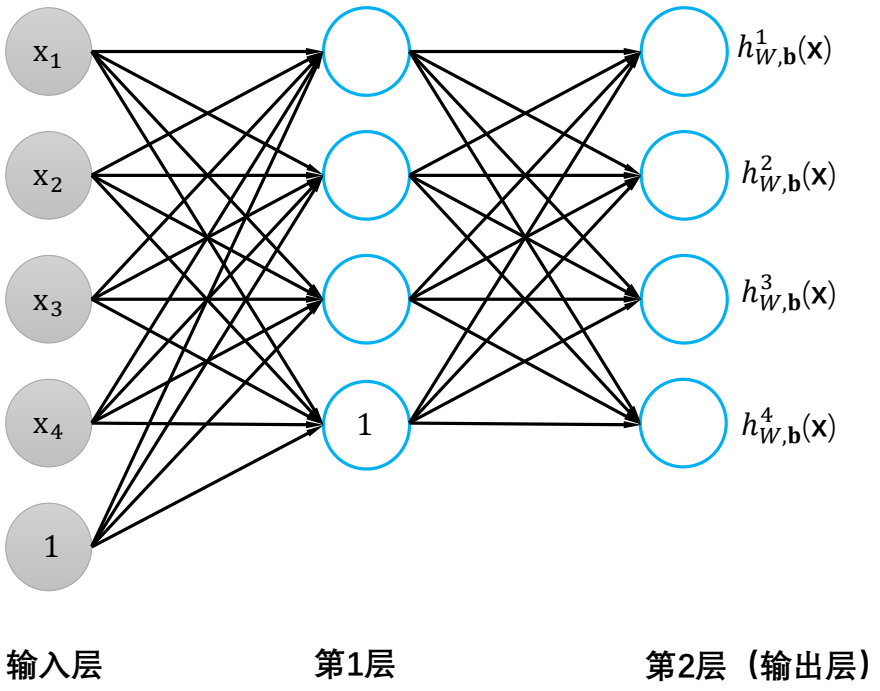


图8-11 自编码器架构

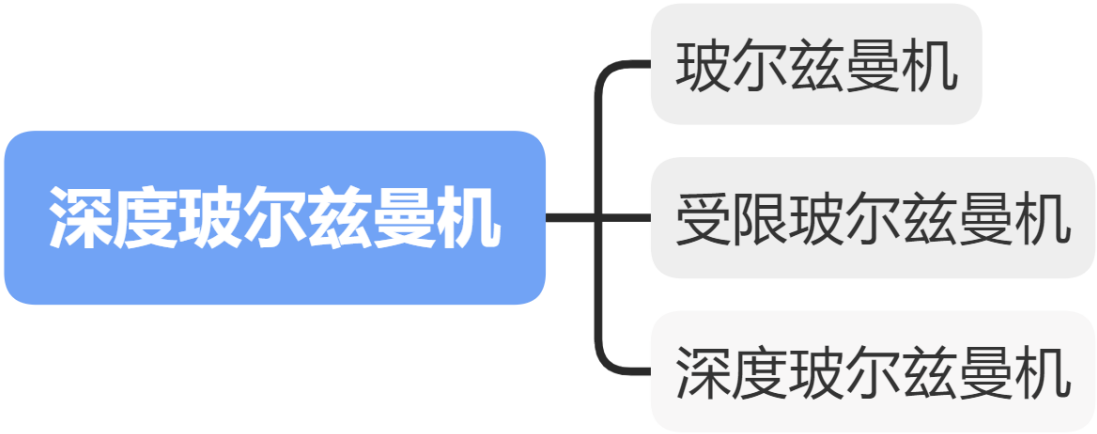
➢ 自编码网络巧妙地使用重构误差实现对数据的非监督编码，也是进行逐层训练的重要组成部分。

➢ 通过对网络加入一些约束，如限制隐含单元数目。如果隐含单元数目小于原始特征数目，学习到数据的一种降维表示。

- 深度玻尔兹曼机

➢ 深度波尔兹曼机完全使用无向图模型实现一个生成式概率神经网络。

➢ 可用于对复杂数据进行密度估计，也可以实现对数据的深层编码。



波尔兹曼机

无向概率图模型，所有节点之间互相连接，样本分布服从玻尔兹曼分布，用于学习数据的固有内在表示。

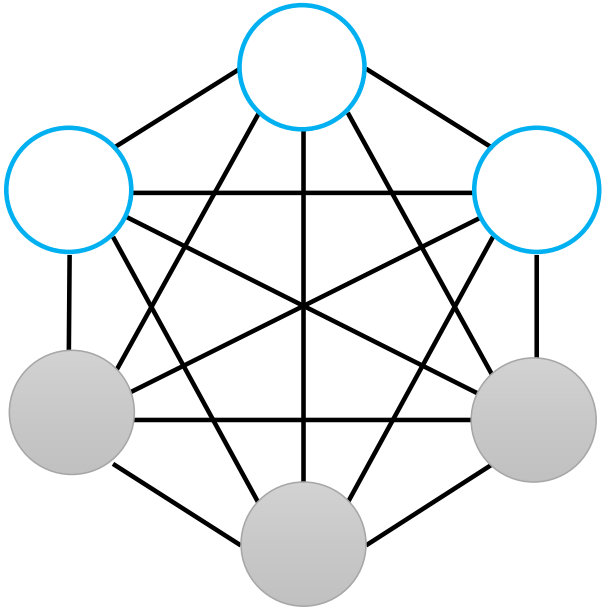


图8-12 玻尔兹曼机示例

受限波尔兹曼机

特殊的无向概率图模型，由一层观测单元和一层隐含单元组成，只有观测单元和隐含单元之间才存在连接，同类单元都不连接。用于学习观测数据的表示。

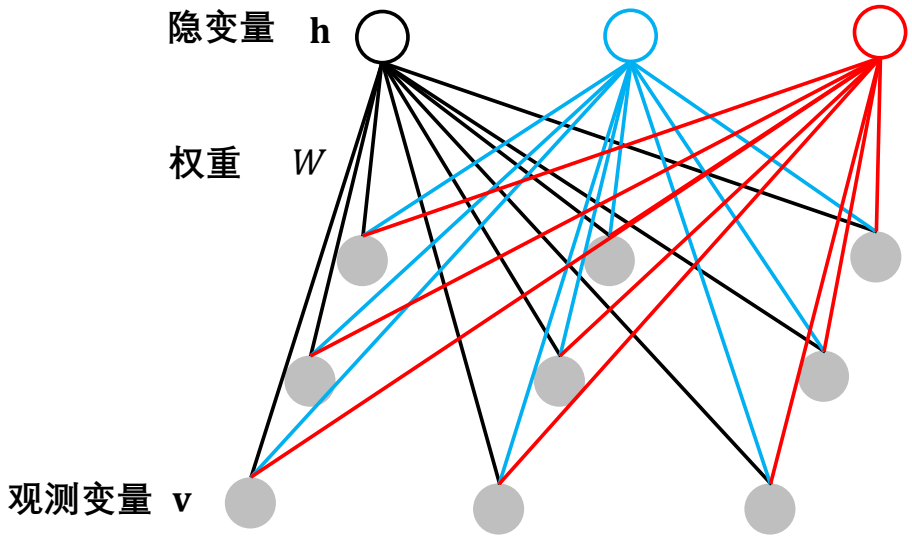


图8-13 受限玻尔兹曼机示例

深度波尔兹曼机

由一层观测单元和多层隐含单元组成的无向图模型，也是生成式概率图模型，也是无监督模型，可用于深度神经网络的预训练网络。

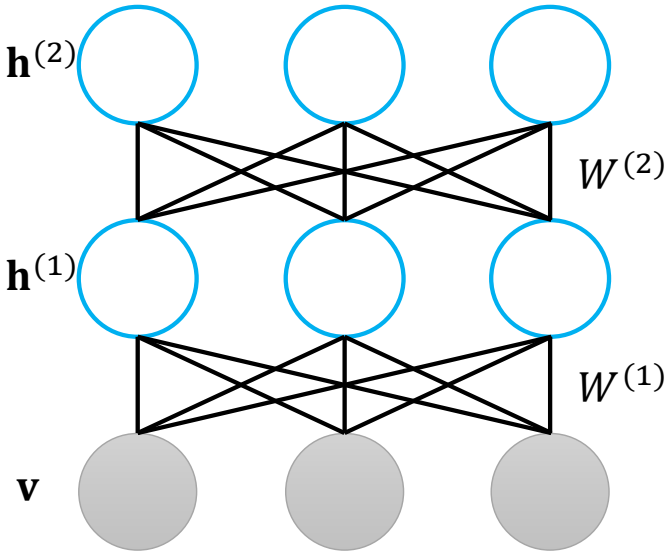


图8-13 深度玻尔兹曼机示例

• 深度信念网络

又称为深度置信网络，包含一层观测单元和多层隐含单元，同一层单元不连接。隐含单元通常是二值的，观测单元可以是实数或二值。是一种结合了无向图和有向图的生成式概率图模型，可用于无监督学习。

训练过程比较复杂，既不能使用反向传播，又不能使用最大边缘似然估计，常用逐层训练。

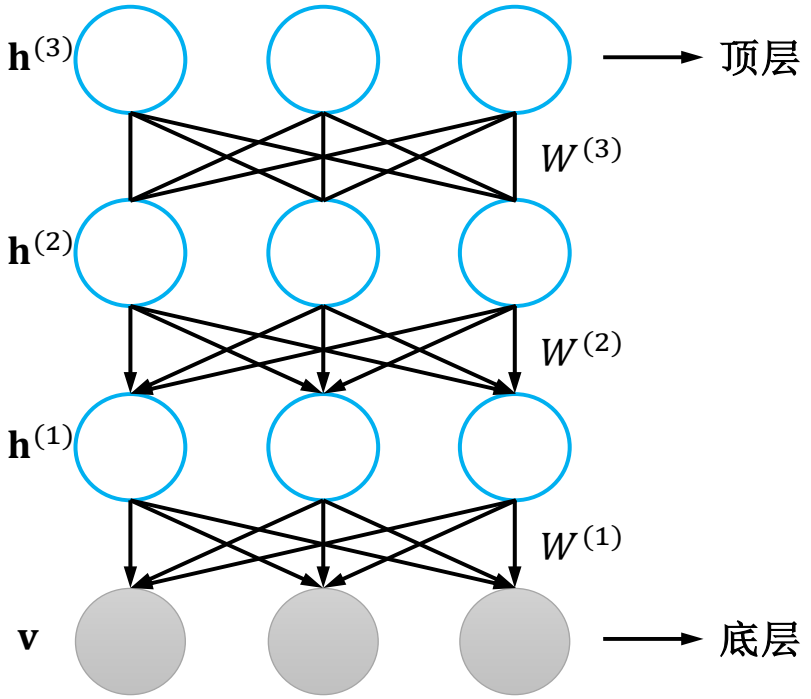


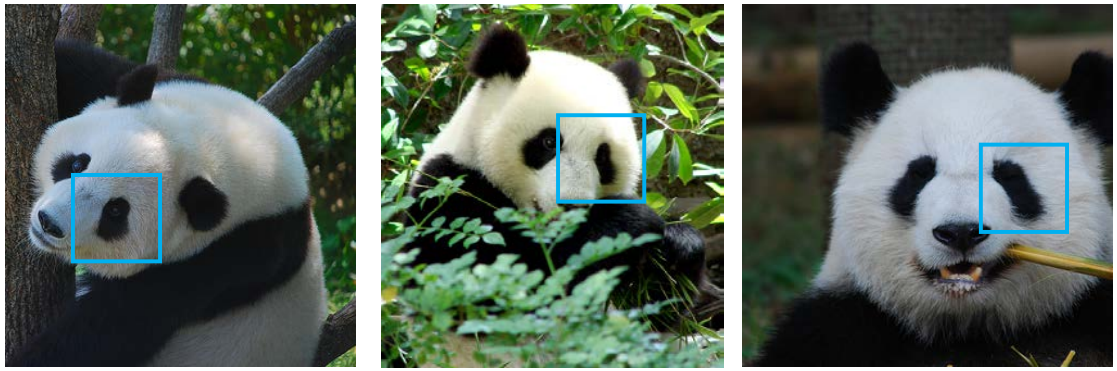
图8-14 深度信念网络示例

- 卷积神经网络（Convolutional Neural Networks: CNN）

三条性质：

① 某些模式总是存在于局部区域

② 相同的模式会出现在多个区域



③ 对图像中的像素做下采样（subsampling）不会影响物体的识别

- CNN是一种前馈神经网络，卷积层与一般的全连接层不同，不再使用权重矩阵表示所有神经元节点在相邻网络层之间的一一对应关系，而是使用多组共享参数来构建两个网络层之间的联系。
- 在卷积网络中，共享参数叫做卷积核。
- 由一个或多个卷积层与一个或多个全连接层构建，其中图像经过卷积后获得的表示经常要进行下采样操作，即池化操作。

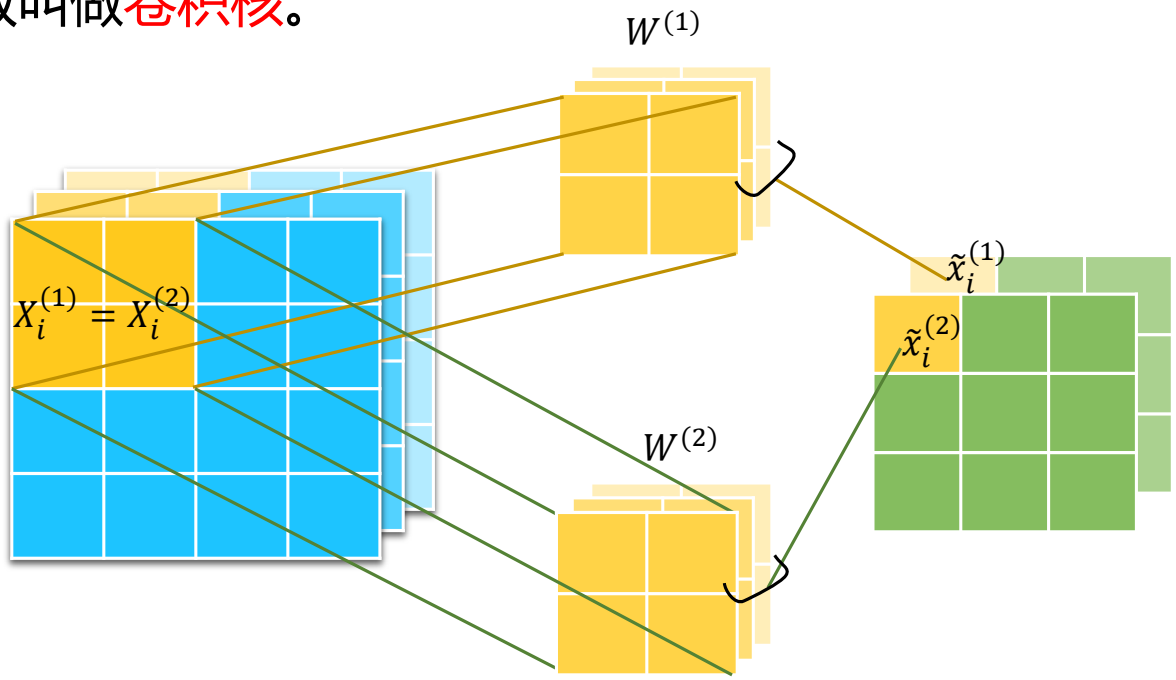


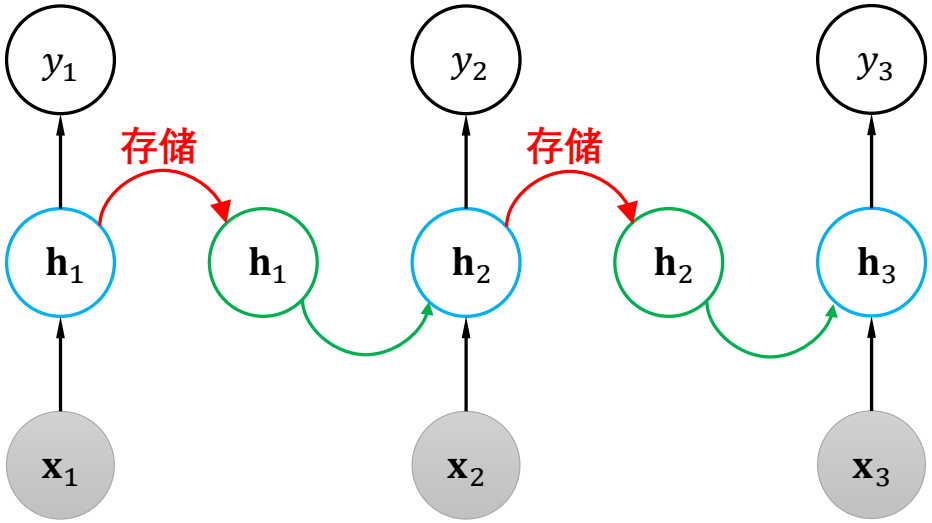
图8-18 卷积操作的原理示意图

- 循环神经网络（Recurrent Neural networks: RNN）

RNN是带有“记忆”的神经网络。隐含层前一时刻的输出会和下一时刻的输入一起传递下去。每个时刻的“记忆”$\boldsymbol{h}_t$也称作状态信息，假设每个时刻的输入为$\boldsymbol{x}_t \in \boldsymbol{R}^D$，状态信息的计算为

$$\boldsymbol{h}_t = f(\boldsymbol{W}[\boldsymbol{x}_t^T, \boldsymbol{h}_{t-1}^t]^{\boldsymbol{T}} + \boldsymbol{b})$$

其中$\boldsymbol{W} \in \boldsymbol{R}^{M\times(M+D)}$和$\boldsymbol{b} \in R^M$是网络的权重和偏置项，$f(\cdot)$是激活函数，通常使用tanh函数。循环神经网络可以在每个时刻输出标签y_t。



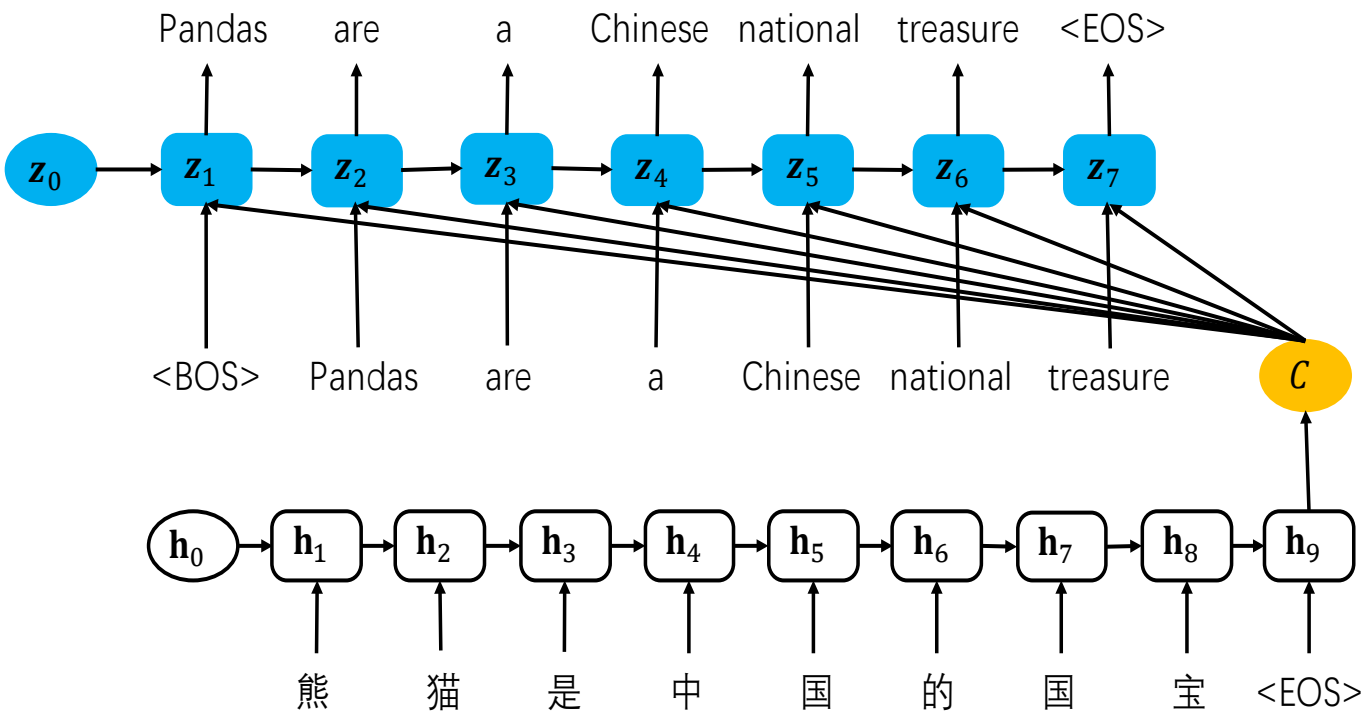


图8-20 使用seq2seq模型进行机器翻译示意图

长短期记忆（long short-term memory，LSTM）

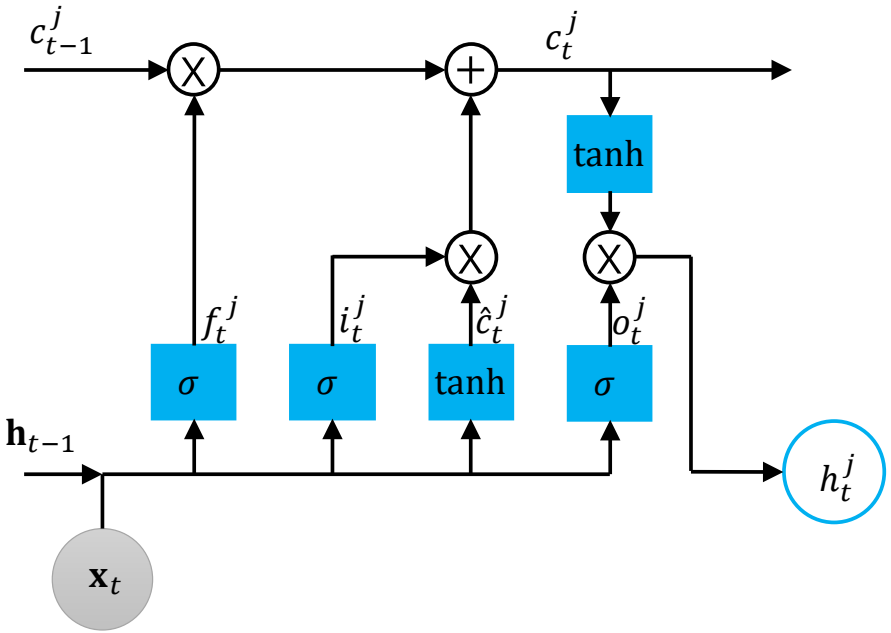
- 一种典型的门模型，它使用一些门操作来控制“记忆”的取舍。
- LSTM每个时刻t的输入为$\mathbf{x}_t \in R^D$，传播的信息有传统RNN中的状态信息$\mathbf{h}_t \in R^M$和长时的“记忆”$\mathbf{c}_t \in R^M$。

遗忘门(forget gate) $\mathbf{f}_t$

输出门(output gate) $\mathbf{o}_t$

输入门(input gate) $\mathbf{i}_t$

门被用于控制LSTM中各个变量之间的比例关系，当前时刻的“记忆”$\mathbf{c}_t$由遗忘门和输入门决定。



- LSTM通过引入一个变量存储“记忆”，增强了RNN把握长距离关系的能力，也可以缓解梯度消失问题。
- 常用于处理时间序列分析。
- 除了LSTM之外，门循环单元（gated recurrent unit，GRU）也是RNN的变体，它进一步简化了LSTM的结构，能使用更少的参数达到近似的性能。

- Transformer
  - Transformer是一种seq2seq模型，其核心思想是使用注意力（attention）和自注意力（self-attention）机制。
  - **注意力机制**用于捕获输入序列和输出序列之间的关系。
  - **自注意力机制**用于捕获文本序列内部的依赖关系，构建对原始文本的语义表示。
  - 其中的自注意力是一种特殊的注意力模型。

注意力

- 注意力作为组件被嵌入在seq2seq神经机器翻译模型中，用于学习序列到序列的对齐关系。
- 假设序列数据的长度为T，序列的每一个元素记为向量$\mathbf{x}_i$，那么序列可以表示为$\{\mathbf{x}_i\}_{i=1}^T$。在注意力语境下，输入称为value变量。除了输入，注意力机制中还会引入key变量$\{\mathbf{k}_i\}_{i=1}^T$和query变量$\mathbf{q}$，它们通常是$\{\mathbf{x}_i\}_{i=1}^T$的函数。首先，对于query变量$\mathbf{q}$，注意力计算出每一个输入$\mathbf{x}_i$权重α_i

$$\alpha_i = \text{softmax}(\text{s}(\mathbf{k}_i, \mathbf{q})),$$

其中s是评分函数。一种常用的评分函数是加性注意力

$$\text{s}(\mathbf{k}_i, \mathbf{q}) = \text{s}(\mathbf{x}_i, \mathbf{q}) = \mathbf{v}^T \tanh(\mathbf{W}\mathbf{x}_i + \mathbf{U}\mathbf{q}),$$

得到权重$\{\alpha_i\}_{i=1}^T$之后，注意力机制根据权重计算所有输入的加权平均值

$$\mathbf{a} = \text{attention}(\mathbf{X}; \mathbf{K}, \mathbf{q}) = \sum_{i=1}^{T} \alpha_i \mathbf{x}_i,$$

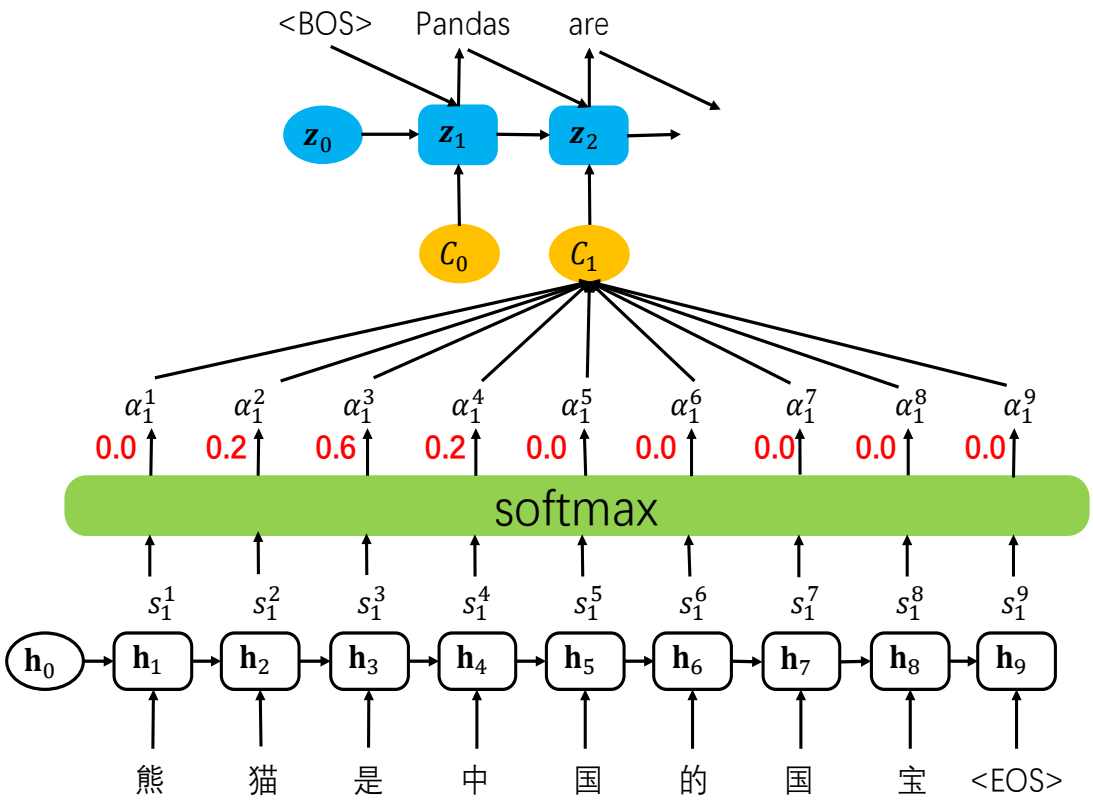


图8-22 有注意力机制的seq2seq模型进行机器翻译的示意图

自注意力

所谓自注意力，是指输入序列中的每个单词（或字）都要和该序列中的所有单词（或字）进行注意力计算。好处是学习序列内部的单词（或字）的依赖关系，捕获句子的内部结构。

Transformer

- 编码网络包含“多头自注意力（multi-head self-attention）”子结构，用于表示多组不同的注意力分配机制。这个子结构的实现方式是同时构建多个自注意力单元，并在最后汇总。
- Transformer也用到了在解码器中增加“遮蔽的多头自注意力（masked multi-head self-attention）”和在输入层增加位置编码等技巧

谢 谢!

本课件制作过程中，多处引用了国内外同行的网页、教材、以及课件PPT的内容或图片，没有随处标注，特此说明，并在此向各位作者表示感谢！