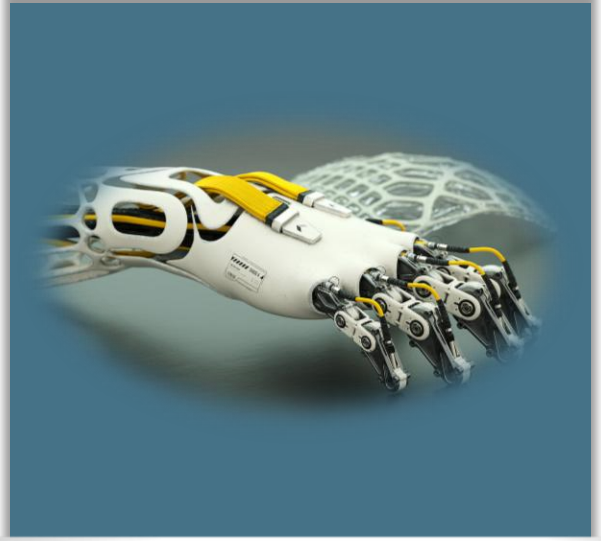
机器学习 第四讲

授课人：王闻博

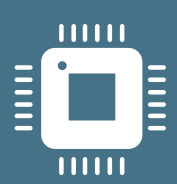
Email: wenbo_wang@kust.edu.cn

昆明理工大学 机电工程学院

2026年04月01-10日



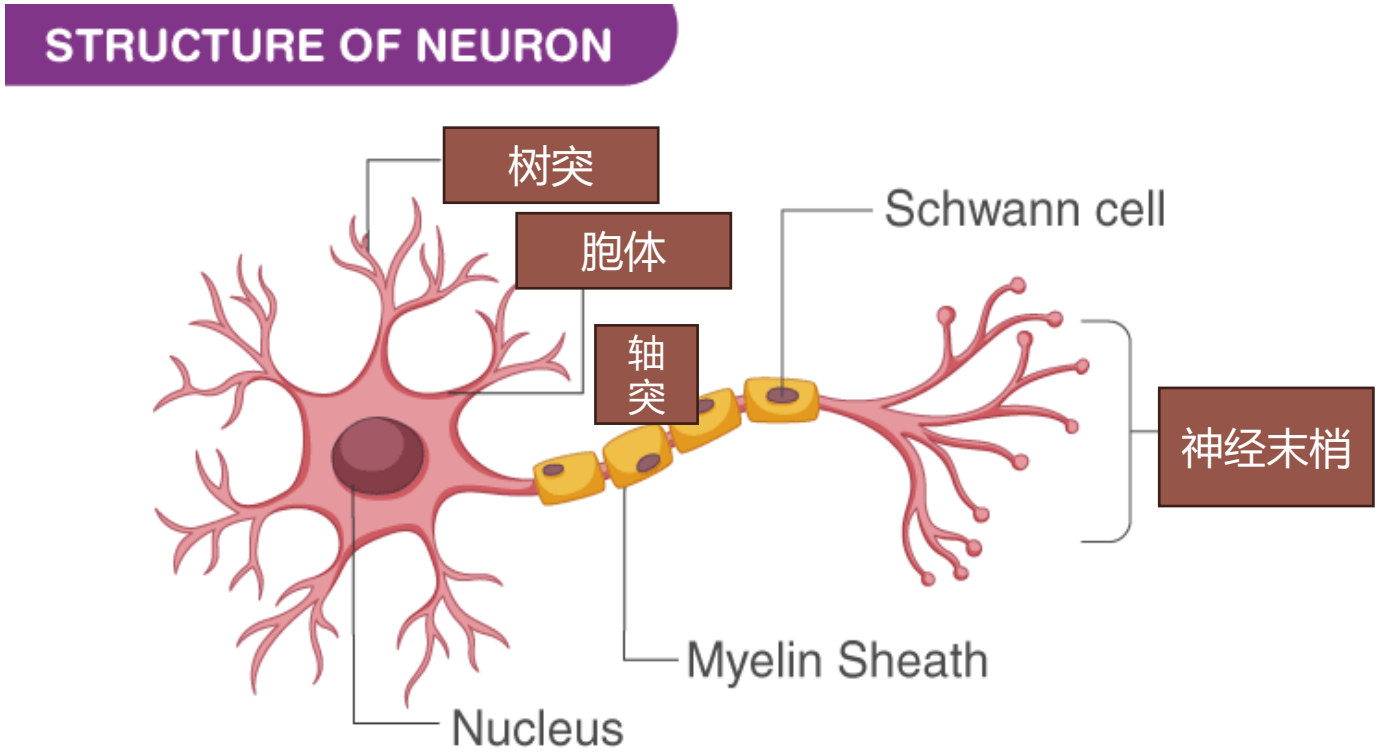
前馈神经网络




1. 从神经元模型到前馈神经网络
2. 损失函数和反向传播算法
3. 贝叶斯神经网络
4. 神经网络的模型训练流程

从生物神经网络到人工神经网络

- **生物神经元的基础模型**
  - 树突：接收多个输入信号；
  - 胞体：处理信号；
  - 轴突：传递信号（单输出）；
  - 神经末梢：神经元与其他细胞通信；
  - 神经元间建立新的连接或修改已有连接为生物神经网络的学习过程。



- 树突
- 胞体
- 轴突

→

- 输入
- 处理
- 输出

从生物神经网络到人工神经网络

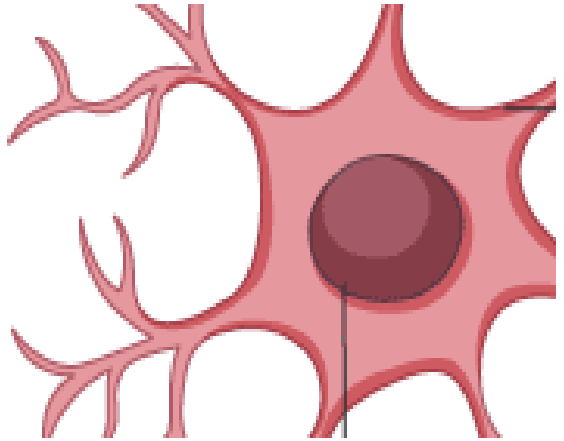
- **单神经元模型（Perceptron）：适用于二分类任务（一个输出）**
  - 神经元计算模型

$$y = f\left(\sum_{i=1}^{M} w_i x_i + w_0\right)$$

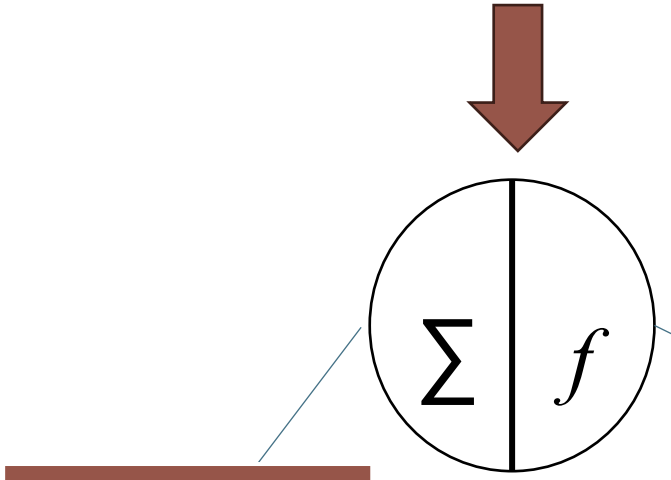
  - 模型向量化：$y = f(\tilde{\mathbf{W}}\tilde{\mathbf{x}})$;
  - 对比第三讲（回归模型）和第四讲（线性分类模型）中的带核函数的广义线性模型：

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

  - 其中 $a = \sum_{i=1}^{m} w_i x_i + w_0$ 称为激活信号（Activation），$f(\cdot)$ 是非线性激活函数（见下页）。



胞体对输入信息进行处理



输入加权求和

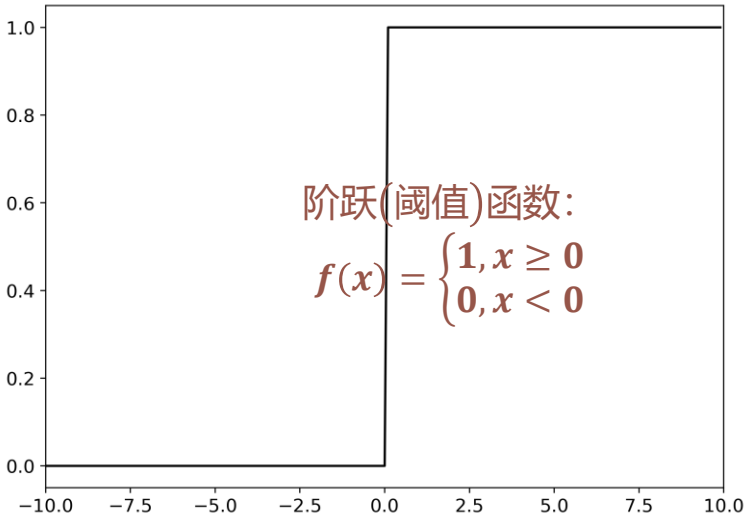
信号处理/激活

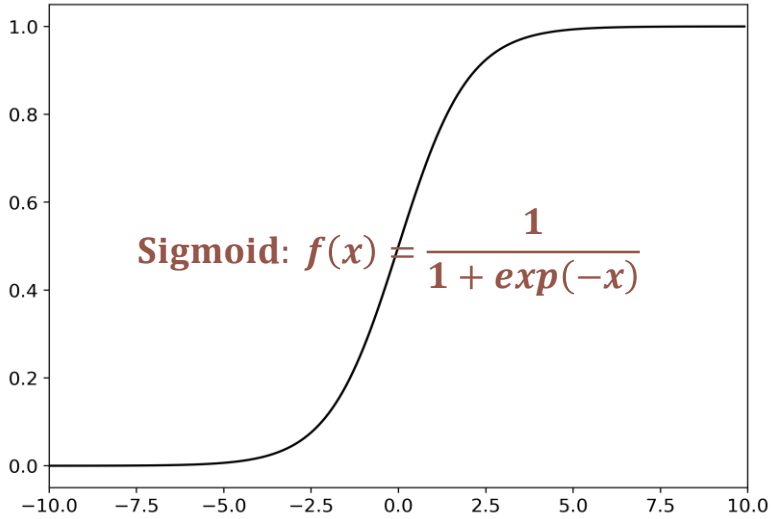
常见的激活函数

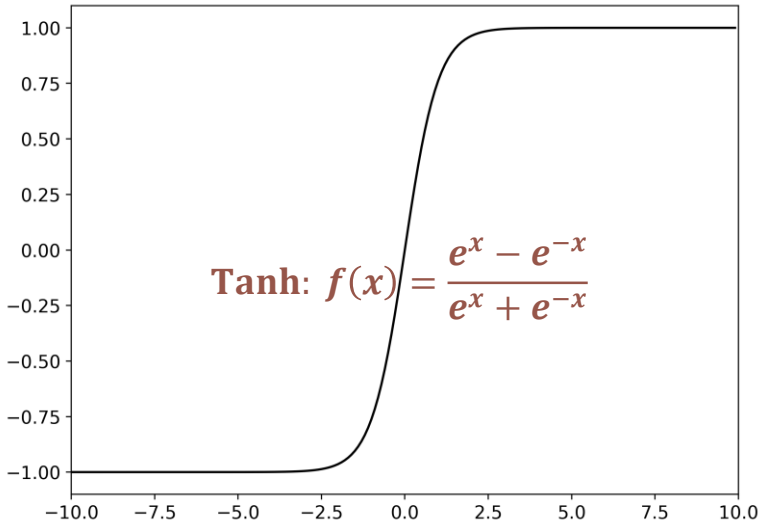
- **激活信号一般通过可微的非线性激活函数$f(\mathrm{a})$ 变换得到神经元的输出：**

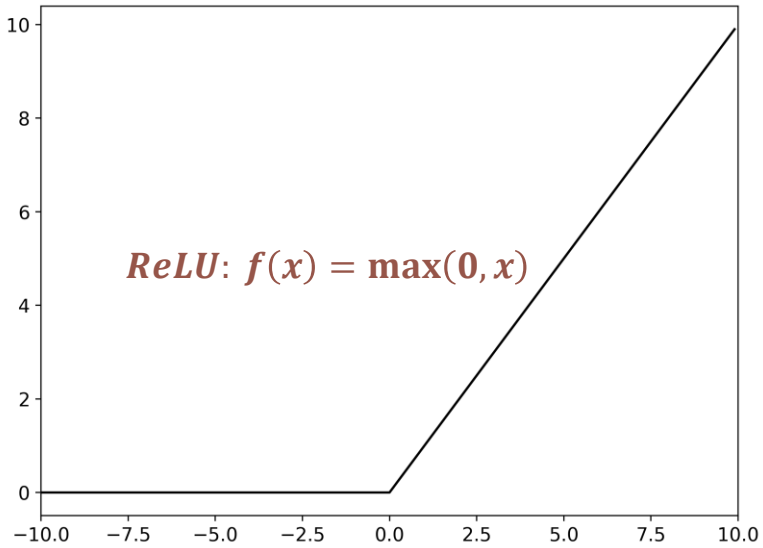
$$\mathrm{z} = f(a)$$

  - 思考：单一神经元的能力范围在何处?





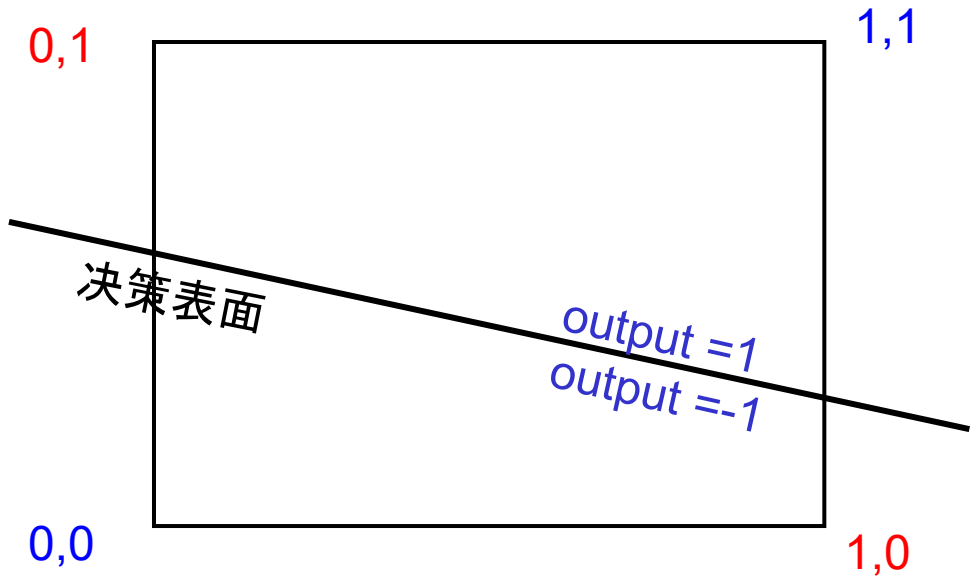


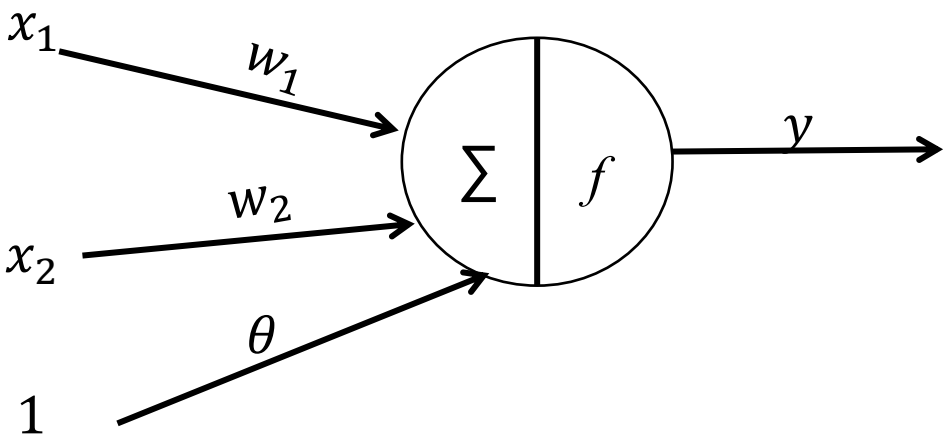


单一神经元的局限

- **回忆：广义线性模型，分类决策表面总是一个M-1维关于输入特征空间的超平面：**
  - $\mathbf{w}^{\mathrm{T}}\mathbf{x} = \theta$
- **XOR运算符号下的二维特征向量分类任务**
  - Positive类（标签为+1）：(1,1) → 1; (0,0) → 1；
  - Negative类（标签为-1）：(1,0) → 0; (0,1) → 0;
  - 决策表面：$f(\mathbf{x}) = 0$，见右上图
  - 如右上图所示，决策表面无法同时满足关于Positive和Negative类的如下条件（输出如右下图所示）：

$$w_1 + w_2 \geq \theta, \quad 0 \geq \theta$$
$$w_1 < \theta, \quad w_2 < \theta$$





引入隐藏层：把多个神经元连接起来

- **多层前馈神经网络**
  - 网络结构：

输入层 → 隐藏层 → 输出层

  - 第1层第j个神经元的激活信号：

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

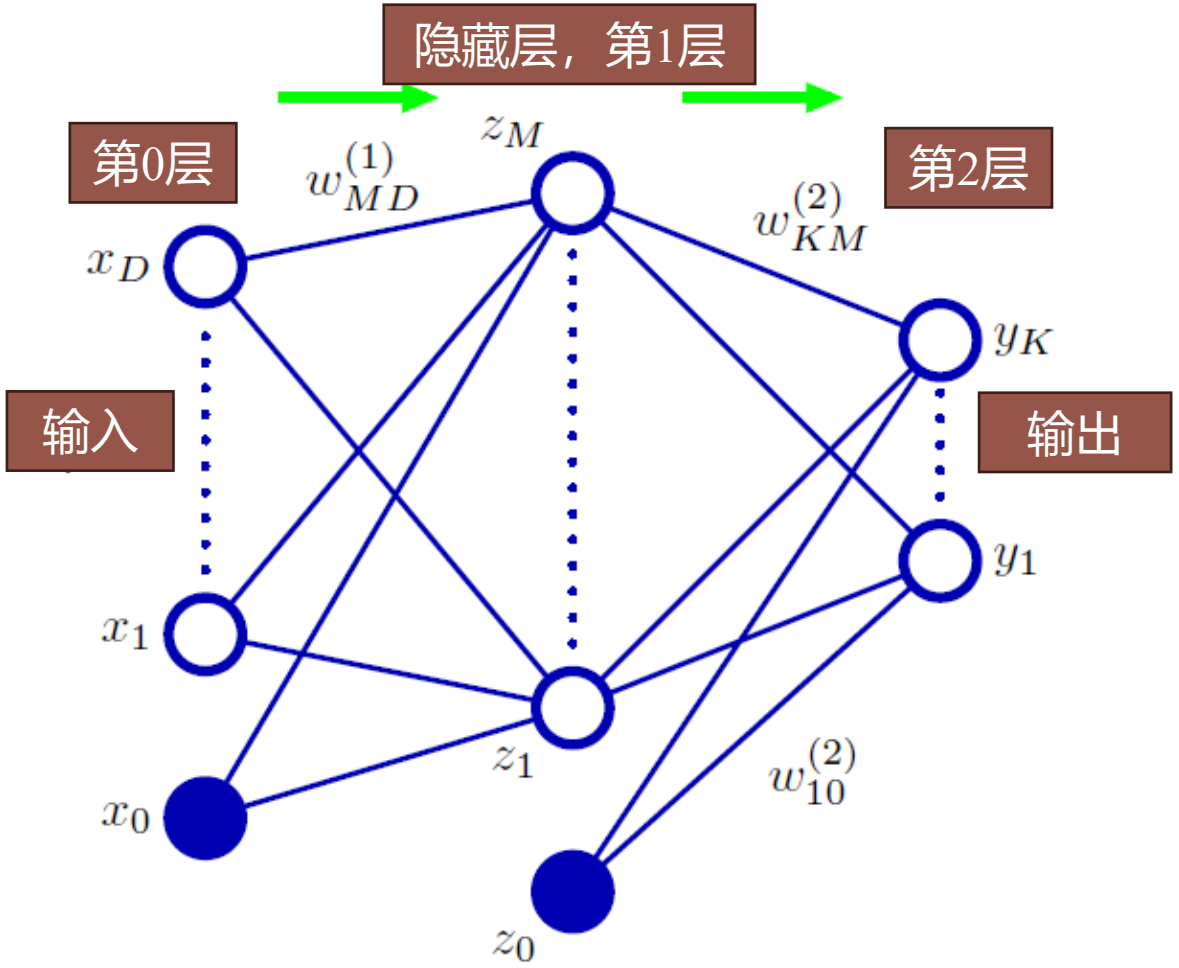
注意：按本讲座约定，此处角标中前一层的被连接节点（父节点）ID在后（j← i）

  - 第1层第j个神经元的激活函数输出：

$$z_j = h(a_j)$$

  - 第2层（输出层）第k个节点的激活信号：

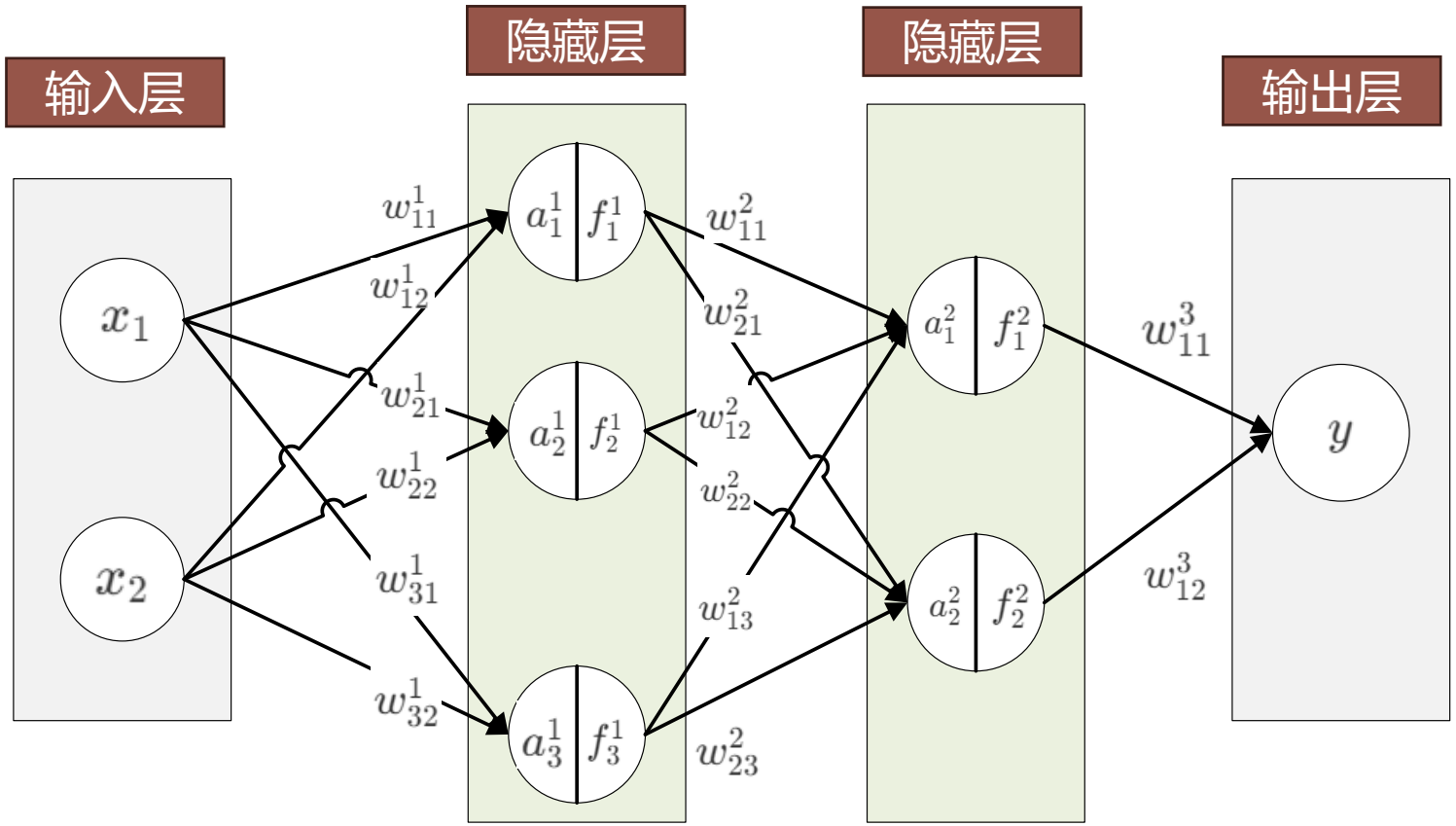
$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$



  - 第2层（输出层）的输出 $y_k = \sigma(a_k)$ ，二分类任务可用sigmoid函数，多分类任务可用Softmax函数。（回忆第四讲内容）

多层前馈神经网络的分层表示

- **用于回归任务的多层前馈神经网络**
  - **注意1：前馈网络一般是全连接网络，一定是无环网络，各神经元节点可以采用不同的激活函数；**
  - **注意2：（与下图所用下标相反）不排除某些文献中，w_{kj}代表前一层第k个神经元节点到后一层第j个神经元节点的连接权重（k→ j）。**



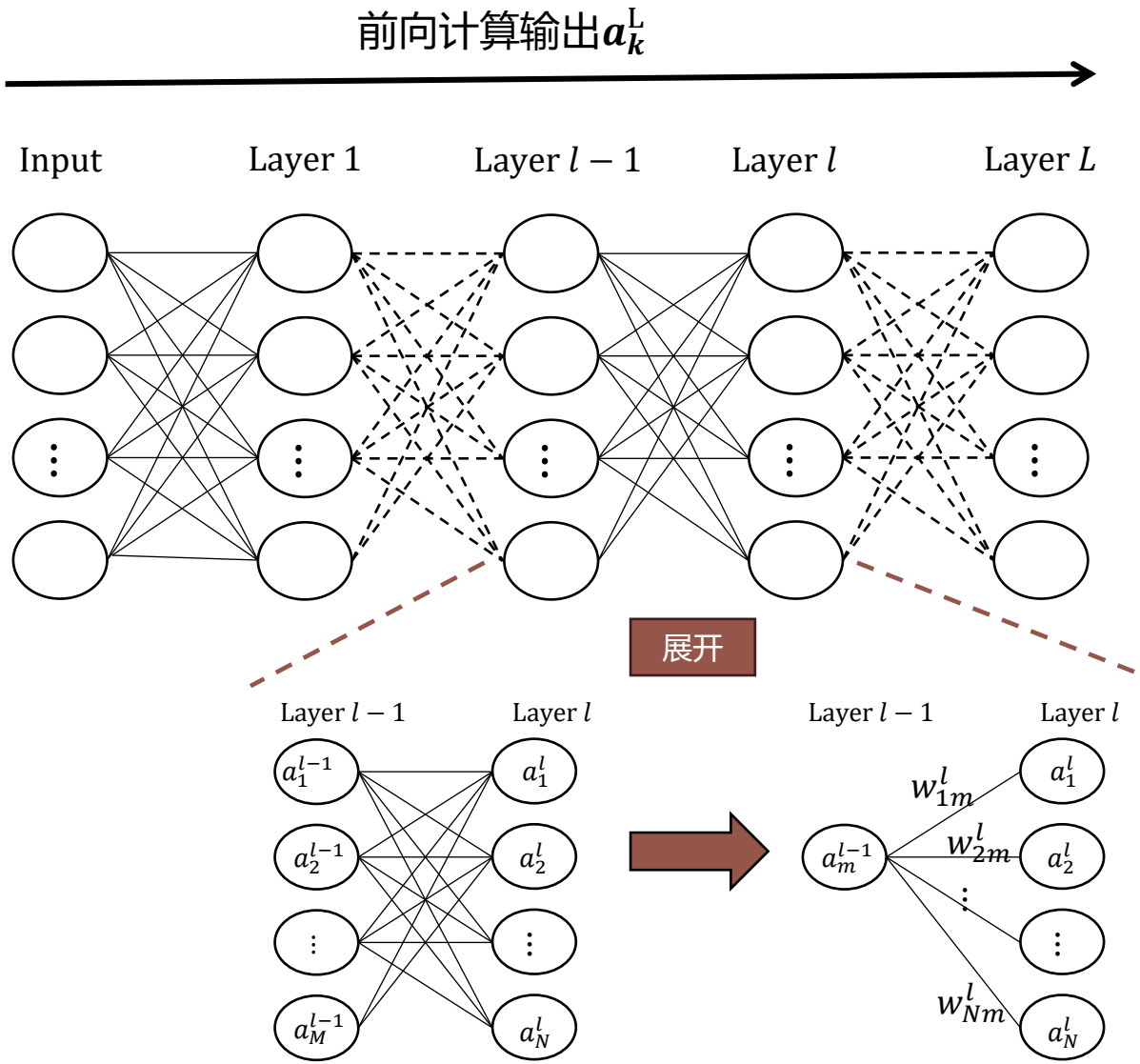
前向传播（Feed-forward Propagation）算法

- **权重已知，从输入到输出的计算（见右图）**
- **假设$l-1$层有M个神经元， l层有N个神经元，对第l层的第n个神经元，其输出的迭代展开公式如下：**

$$z_n^l = f_n^l\left(a_n^l = \sum_{m=1}^{M} w_{nm}^l z_m^{l-1} + w_{n0}^l\right)$$

- **向量化（把偏置项吸收进权重项）**

$$\mathbf{z}^l = f^l\left(a_n^l = \mathrm{W}^l \mathbf{z}^{l-1}\right)$$



前向传播算法（续）

- **前向传播计算向量化（接上页）**

$$z_n^l = f_n^l\left(\sum_{m=1}^{M} w_{nm}^l z_m^{l-1} + w_{n0}^l\right)$$

向量化

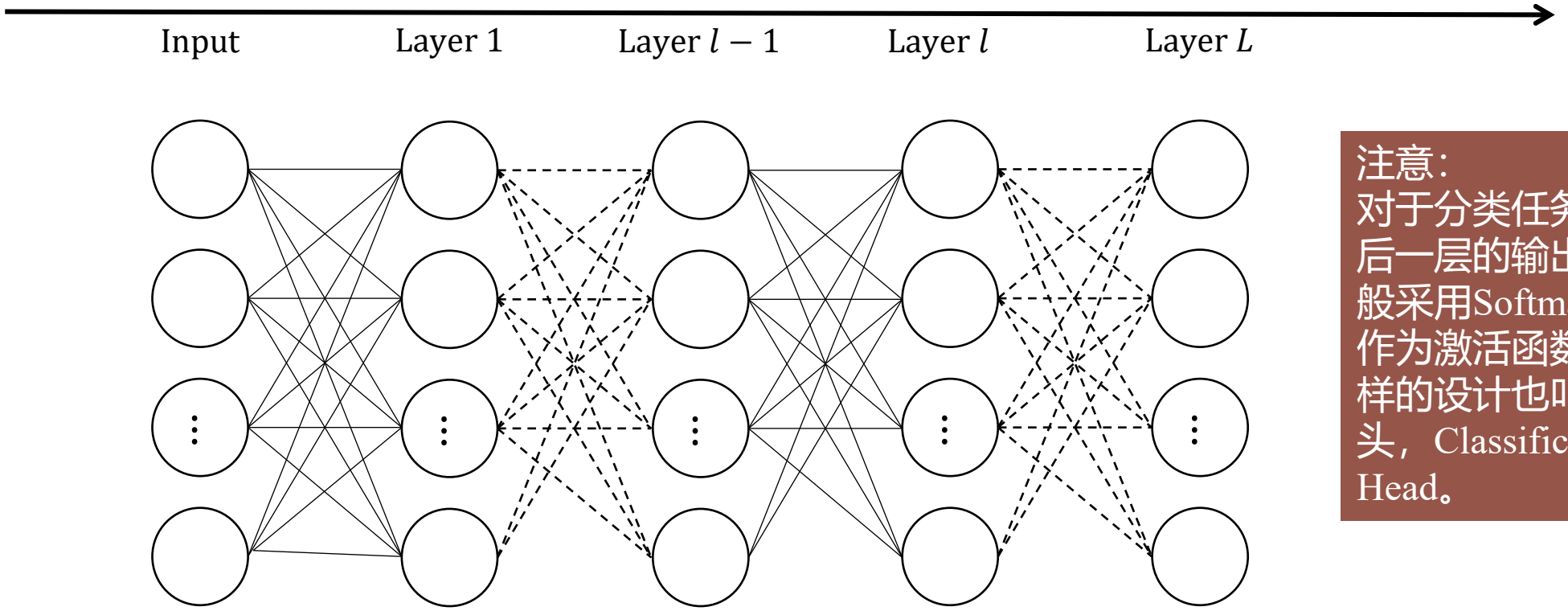
$$\mathbf{z}^l = f^l\left(\mathrm{W}^l\mathbf{z}^{l-1}\right)$$

$$z^{l-1} = \begin{bmatrix} 1 \\ z_1^{l-1} \\ z_2^{l-1} \\ \vdots \\ z_M^{l-1} \end{bmatrix} \qquad \boldsymbol{W}^l = \begin{bmatrix} w_{10}^l & w_{11}^l & w_{12}^l & \cdots & w_{1M}^l \\ w_{20}^l & w_{21}^l & w_{22}^l & \cdots & w_{2M}^l \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{N0}^l & w_{N1}^l & w_{N2}^l & \cdots & w_{NM}^l \end{bmatrix}$$

一般情况：L层前馈神经网络

- **前向传播计算的一般迭代公式**
  - 第L层输出(合并偏置项)的递归形式表达：

$$\mathrm{z}^{L}(\mathbf{x},\mathbf{W})=f^{L}\left(\mathbf{W}^{L}\mathrm{z}^{L-1}\right)=f^{L}\left(\mathbf{W}^{L}f^{L-1}\left(\mathbf{W}^{L-1}f^{L-2}\left(\mathbf{W}^{L-3}\ldots f(\mathbf{W}^{1}\mathbf{x})\right)\right)\right)$$



注意：
对于分类任务，最后一层的输出层一般采用Softmax函数作为激活函数。这样的设计也叫分类头，Classification Head。

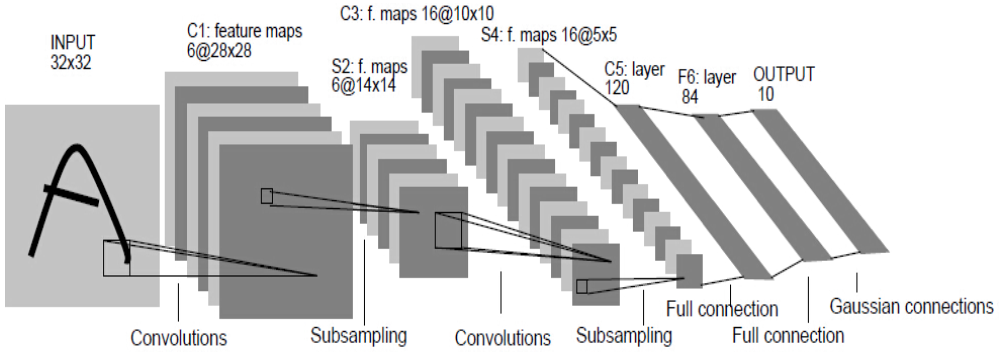
适用于前馈神经网络的标签编码

- **回归任务无需编码：”标签（目标值）”本身为连续数值**
  - 输出层设计：单个输出节点 + 无激活函数。
- **监督学习任务**
  - **二分类任务（极简任务）：标签直接设为0或1。**
    - 输出层设计：使用单个输出节点 + sigmoid激活函数。
  - **多分类任务**
    - 独热编码（One-Hot Encoding）：k维向量表示k个类别，即向量中类对应的一个元素为1，其他元素为0的形式；输出层使用softmax激活函数，输出概率分布。
    - 多热编码（Multi-Hot Encoding）：每个样本对应多个二进制标签，因此k维向量可表示2^k个类别；输出层每个类别设置一个输出节点 + sigmoid激活函数。

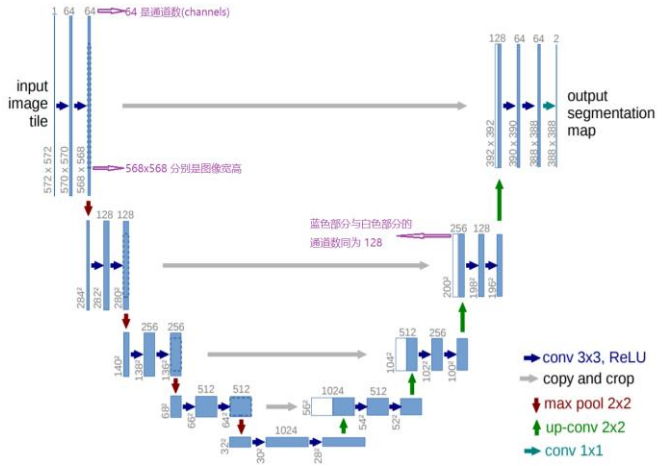
常见多层前馈神经网络一瞥（详细讨论见后续第五讲）

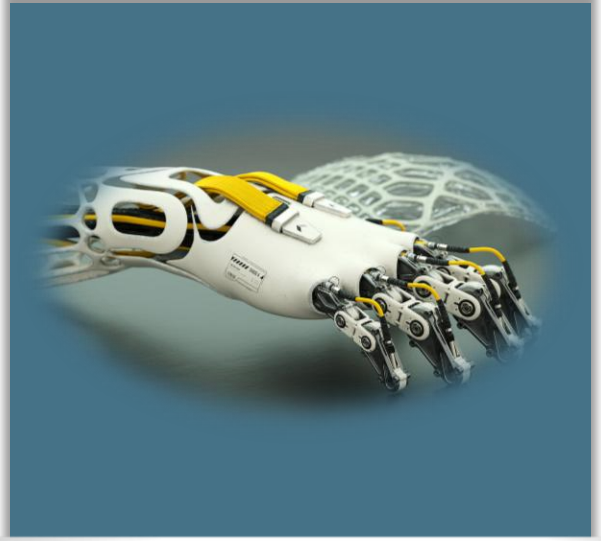
- **线性分类器/回归器可看做是单层（输入层+输出层）神经网络**
  - 基于线性判别函数的多分类分类器：不带激活函数的单层神经网络。
  - 二分类任务下的逻辑回归分类器：激活函数为Logistic Sigmoid函数的单层神经网络。
  - 多分类任务下的逻辑回归分类器：激活函数为Softmax函数的单层神经网络。
- **多层感知机（MLP, Multi-Layer Perceptron）：层与层之间为神经元全连接形式，多层堆叠（见上页图）。**
- **处理二维（空间）特征输入的卷积神经网络（CNN, Convolutional Neural Network）。**
- **自动编码器（Autoencoder）：输入层**→ [特征压缩] →**隐藏层** → [输出重建] →**输出层。**

LeNet：一种典型的CNN网络

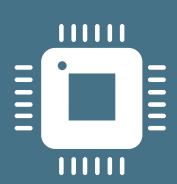


U-Net：一种典型的自动编码器网络





前馈神经网络




1. 从神经元模型到前馈神经网络
2. 损失函数和反向传播算法
3. 贝叶斯神经网络
4. 神经网络的模型训练流程

网络训练：神经网络的参数学习——损失函数构建

- **训练集形式：**$\{\mathbf{x}_n\}$ + $\{\mathbf{t}_n\}$。
- **构建损失函数（参照线性回归和分类判别模型）：平方和误差**

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2$$

- **神经网络训练和线性模型的联系——回归任务（参见第三讲I）：考虑输出层预测值**
  - **仍旧假设目标值采样符合高斯分布：** $p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(\mathbf{t}|\mathbf{y}(\mathbf{x}, \mathbf{w}), \beta^{-1}\mathbf{I}\right)$
  - **基于所有训练样本估计似然函数：**

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \beta)$$

  - **由对数似然函数取负值得到SSE损失函数项+常数项，最大对数似然法与最小平方和误差法等价：**

$$\mathbf{w}^* = \text{argmin}_{\mathbf{w}} \ \frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi)$$

损失函数构建（续）

- **K分类任务：考虑Softmax激活函数构成的输出层（独热编码对应K个输出）**
  - 以参数集和训练数据为条件的标签输出概率（似然函数）：

$$p(\mathbf{t}|\mathbf{x},\mathbf{w}) = \prod_{k=1}^{K} y_k(\mathbf{x},\mathbf{w})^{t_k} \left[1 - y_k(\mathbf{x},\mathbf{w})\right]^{1-t_k}$$

  - 得到交叉熵损失：

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\{t_{nk}\ln y_{nk} + (1-t_{nk})\ln(1-y_{nk})\}$$

  - 上式中，y_{nk}对应输出层第k个神经元的输出 $y_k(\mathbf{x}_n,\mathbf{w})$
  - 注：输出层根据Softmax函数，第k个神经元的输出形式如下：

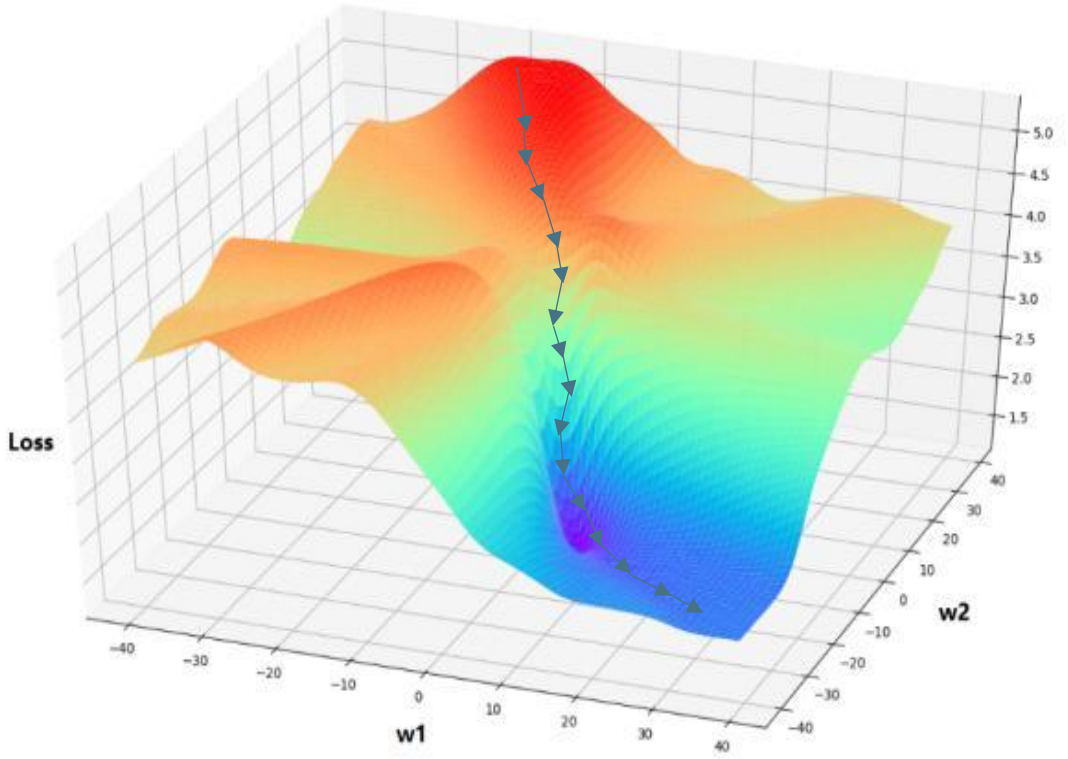
$$y_k(\mathbf{x},\mathbf{w}) = \frac{\exp(a_k(\mathbf{x},\mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x},\mathbf{w}))}$$

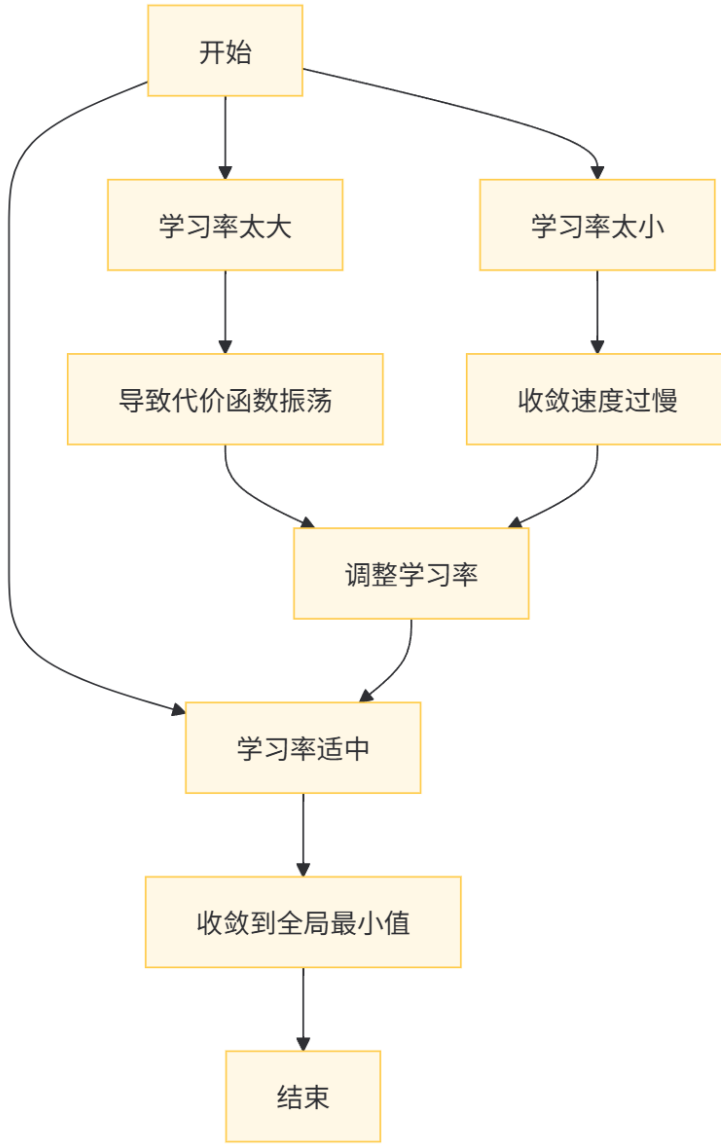
基于损失函数构建参数最优化问题

- **（回忆）参数最优化问题的主要元素**
  - 优化目标： $\min \mathrm{E}(\mathbf{X}; \mathbf{w})$
  - $\mathbf{X}$ – 训练样本集
  - $\mathbf{W}$ – 向量化的神经网络参数
- **一阶优化方法：梯度下降法**
  - **最终目标（极值点一阶条件）：** $\nabla E(\mathbf{w}) = 0$
  - **顺序迭代算法——梯度下降：** $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$

回顾：基本梯度下降法和超参数

- **学习率η的选择**
  - **学习率过大**：收敛速度可能快，但容易导致振荡或发散。
  - **学习率过小**：收敛速度慢，但更稳定。





理想情况

三种梯度下降算法

- **随机梯度下降（Stochastic Gradient Descent，SGD）**
  - 梯度下降的每一步参数更新只用到一个样本，即在每一次计算之后便更新参数，而不需要首先将所有的训练集求和。
- **批量梯度下降（Batch Gradient Descent，BGD）**
  - 梯度下降的每一步参数更新都用到了所有的训练样本。
- **小批量梯度下降（Mini-Batch Gradient Descent，MBGD）**
  - 梯度下降的每一步中，用到了一定批量的训练样本。

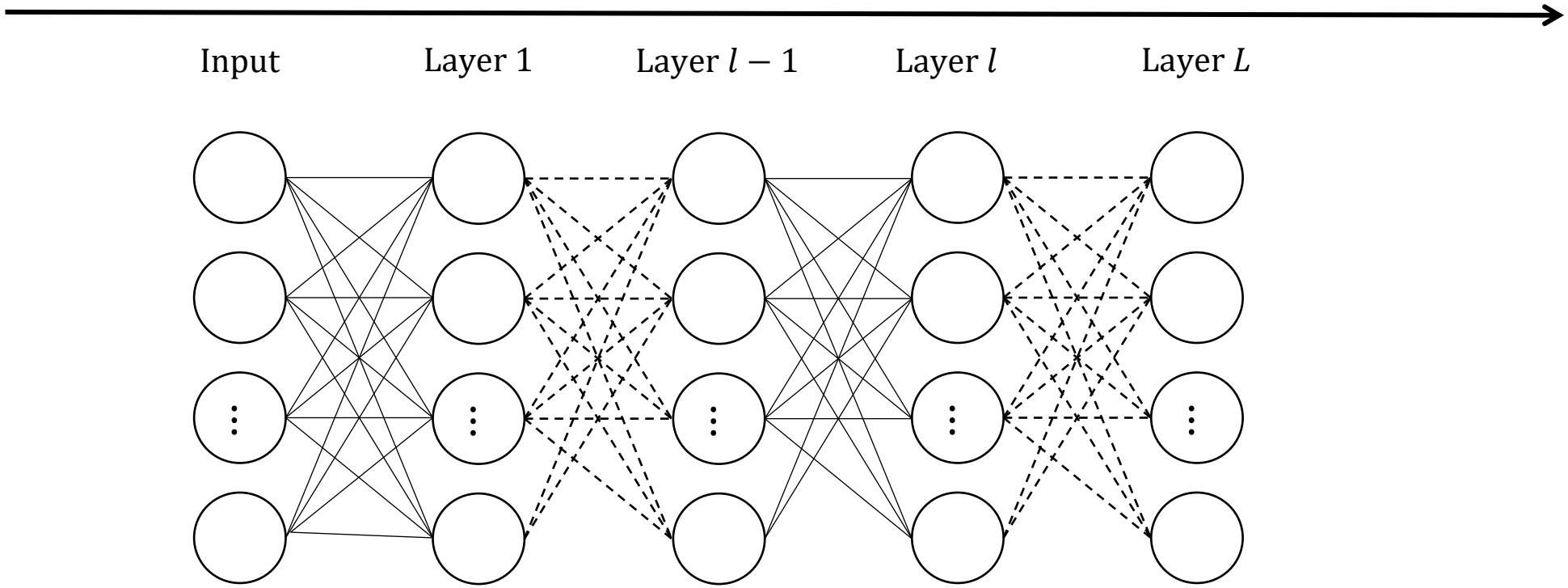
三种梯度下降算法（续）

- **随机梯度下降（Stochastic Gradient Descent，SGD）**
  - $E_n(\mathbf{w};\mathbf{x}_n,t_n)=\frac{1}{2}(\mathrm{y}(\mathbf{x}_n)-t_n)^2$
  - $\mathbf{w}^{t+1}=\mathbf{w}^t-\eta\nabla E_n(\mathbf{w};\mathbf{x}_n,t_n)=\mathbf{w}^t-\eta(\mathrm{y}(\mathbf{x}_n)-t_n)\frac{\partial E_n}{\partial \mathbf{x}}|_{\mathbf{x}_n}$
- **批量梯度下降（Batch Gradient Descent，BGD）**
  - $E(\mathbf{w};\mathbf{x}_n,t_n)=\frac{1}{2}\sum_{n=1}^{N}(\mathrm{y}(\mathbf{x}_n)-t_n)^2$
  - $\mathbf{w}^{t+1}=\mathbf{w}^t-\eta\nabla E(\mathbf{w};\mathbf{x}_n,t_n)=\mathbf{w}^t-\eta\sum_{n=1}^{N}(\mathrm{y}(\mathbf{x}_n)-t_n)\frac{\partial E}{\partial \mathbf{x}}|_{\mathbf{x}_n}$
- **小批量梯度下降（Mini-Batch Gradient Descent，MBGD）:每次对一个样本集合$\mathrm{B_i}$计算梯度**
  - $E_b\ (\mathbf{w};\mathbf{x}_n,t_n)=\frac{1}{2}\sum_{i\in \mathrm{B_i}}(\mathrm{y}(\mathbf{x}_i)-t_i)^2$
  - $\mathbf{w}^{t+1}=\mathbf{w}^t-\eta\nabla E_b(\mathbf{w};\mathbf{x}_n,t_n)=\mathbf{w}^t-\eta\sum_{i\in \mathrm{B_i}}(\mathrm{y}(\mathbf{x}_i)-t_i)\frac{\partial E_b}{\partial \mathbf{x}}|_{\mathbf{x}_i}$。

L层前馈神经网络的训练问题

- **需求：计算损失函数对每一层参数$\mathbf{W}^l$的梯度，由于其递归表达形式，并非简易的向量式计算过程；**
- **解决方案：基于有向无环图的自动微分机。**

$$\mathrm{z}^L(\mathbf{x},\mathbf{W}) = f^L\left(\mathbf{W}^L\mathrm{z}^{L-1}\right) = f^L\left(\mathbf{W}^L f^{L-1}\left(\mathbf{W}^{L-1} f^{L-2}\left(\mathbf{W}^{L-3}\ldots f(\mathbf{W}^1\mathbf{x})\right)\right)\right)$$



反向传播算法（Back Propagation）：正向模式自动微分机

- **反向传播（backprop）：用来计算损失函数关于神经网络中某一层参数的梯度**
  - 在某些语境下，反向传播也被描述为“误差的反向传播”。此处，误差特指模型采用损失函数为SSE损失时，输出端的误差：

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

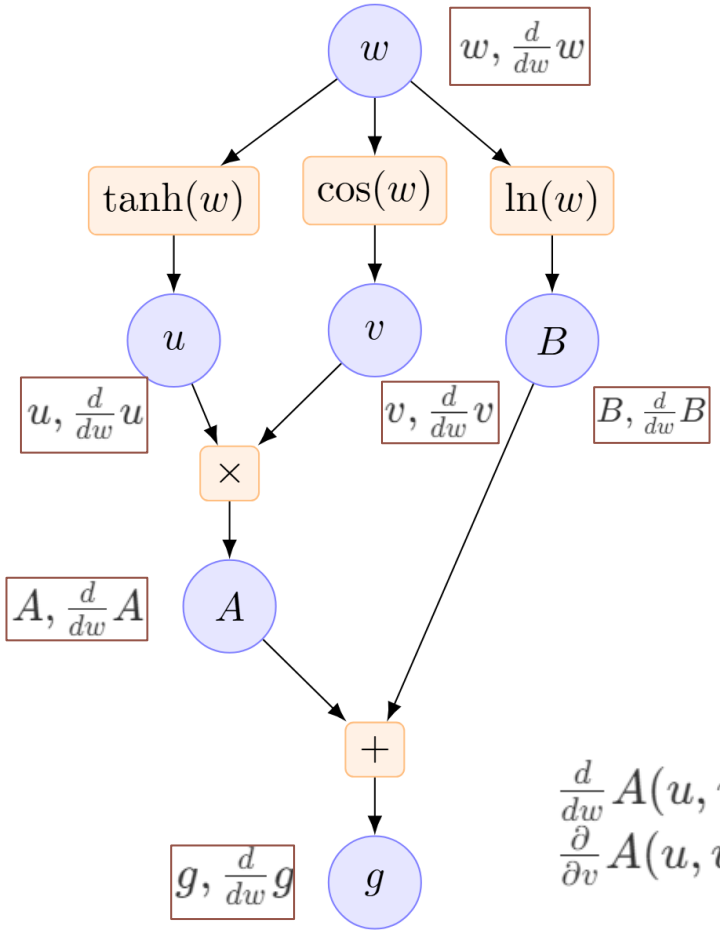
对单个样本而言，误差

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$

$$\delta_k = y_k - t_k$$

  - 反向传播用于描述在计算图上程序化计算梯度的一种方式。
  - 广义的“自动微分机”算法包含前向自动微分和后向自动微分，反向传播指代后一模式。

- **计算图示例（前向自动微分）**
  - 考虑函数：$g(w) = \tanh(w)\cos(w) + \ln(w)$



- 图中，每个“子节点”（箭头指向节点）表示为其“父节点”的函数；
- 由此溯源，所有节点都是输入节点w的函数；
- 函数本体的计算流向由输入向前闯过每一组“父-子”节点，直到汇聚节点；
- 输出对输入的微分（多输入形式：梯度）计算根据梯度计算的链式法则，可通过其父节点的梯度计算结果和“父-子节点运算关系”确定，如对节点A，有：

$$\frac{d}{dw}A(u,v) = \frac{\partial}{\partial u}A(u,v) \times \frac{d}{dw}u(w) + \frac{\partial}{\partial v}A(u,v) \times \frac{d}{dw}v(w)$$

反向传播算法（Back Propagation）：反向模式自动微分机

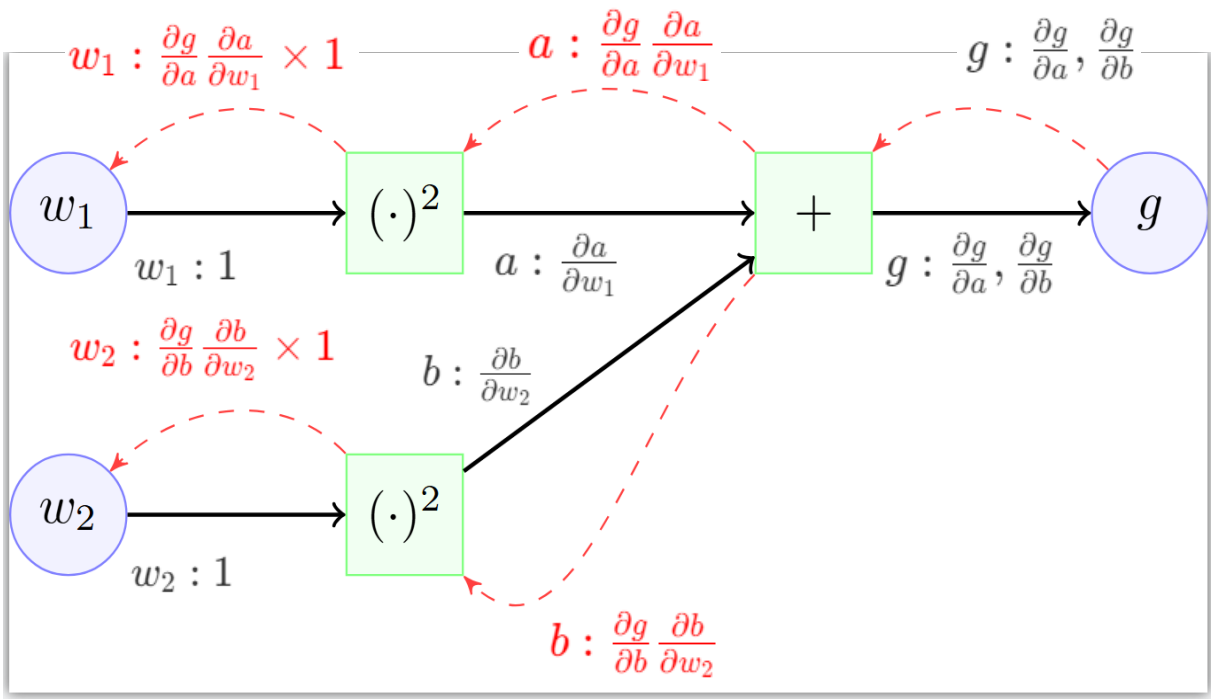
- **回忆：梯度计算的链式法则**

$$\frac{d(y=f(g(u(x),v(x))))}{dx}=\frac{\partial f}{\partial g}\cdot\left(\frac{\partial g}{\partial u}\cdot\frac{du}{dx}+\frac{\partial g}{\partial v}\cdot\frac{dv}{dx}\right)$$

- **前向模式自动微分的缺点**
  - 对多输入函数而言，在计算图的每个节点都要计算节点输出对所有输入的梯度，而不只是与节点输出相关的输入的梯度；

- **反向模式自动微分**
  - 计算过程由对函数计算图的一个正向遍历和一个反向遍历组成。
  - 正向遍历：计算当前节点输出对其父节点的梯度；
  - 反向遍历：更新当前节点的梯度，即

  （左乘）子节点偏导 × 当前节点偏导



- **上图中：**
  - 考虑示例计算图$w_1^2+w_2^2$；
  - 黑色实线-前向遍历，计算节点对父节点输出的梯度；
  - 红色虚线-后向遍历，更新目标值对当前节点所关联参数的梯度。

神经网络中的反向传播算法（适用于随机梯度下降）

- **基于误差传播的梯度计算：从损失函数开始**

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) ; \quad E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$

  - 对第l层的第j个神经元，有：$a_j^l = \sum_i w_{ji}^l z_i^{l-1}$ ；
  - 其激活输出为：$z_j^l = h(a_j^l)$ ；
  - 对其所在的父-子节点关系中的连接权重 w_{ji}^l，有

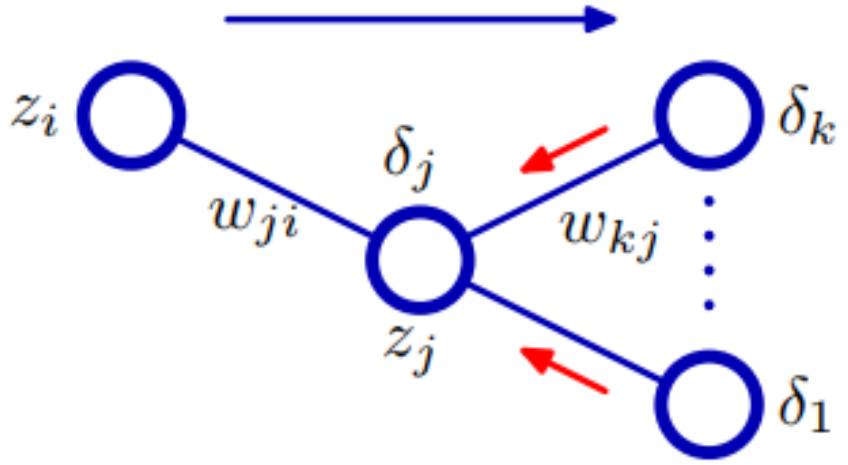
$$\frac{\partial E_n}{\partial w_{ji}^l} = \frac{\partial E_n}{\partial a_j^l}\frac{\partial a_j^l}{\partial w_{ji}^l}$$

  - 由输出误差的定义 $\delta_k = y_k - t_k$，进一步定义当前节点（神经元）误差：损失函数对当前节点激活值的梯度

$$\delta_j^l = \frac{\partial E_n}{\partial a_j^l}$$

误差反向传播部分即是自动微分机反向遍历模式下的左乘的子节点梯度（见上页）

示例：计算图对应某神经网络截取的一段子图



根据本页左侧定义，有对节点j，其误差可展开为

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}$$

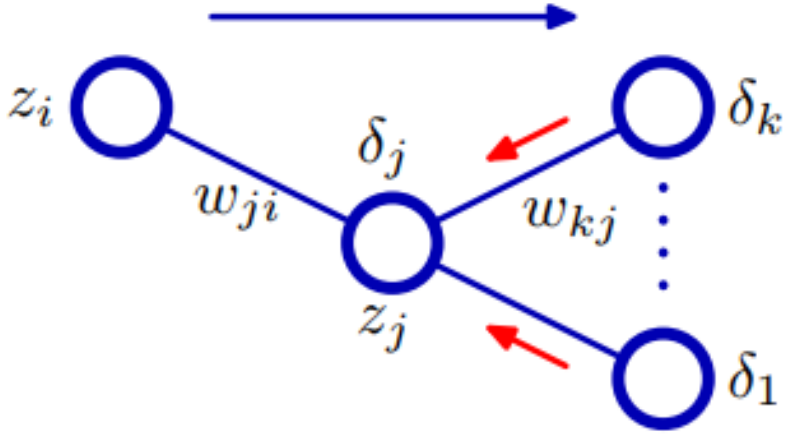
子节点传播的误差（左乘部分）

子节点激活值对当前节点激活的偏导

神经网络中的反向传播算法（适用于随机梯度下降，续）

- **节点j的误差计算（见右图）**

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}$$



- 对节点j的单个子节点k，其激活值写为所有父节点激活值的函数，有：

$$a_k = \sum_{i \in Pred(i)} w_{ki} h(a_i) \quad \text{其中} \quad z_i = h(a_i)$$

- 则，对父节点j，其误差值可以进一步展开为：

$$\delta_j = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial a_k}{\partial a_j} = h'(a_j)\sum_k w_{kj}\delta_k$$

链式法则展开

$$\frac{\partial a_k}{\partial a_j} = \frac{\partial a_k}{\partial z_j}\frac{\partial z_j}{\partial a_j}$$

子节点向后传播的梯度

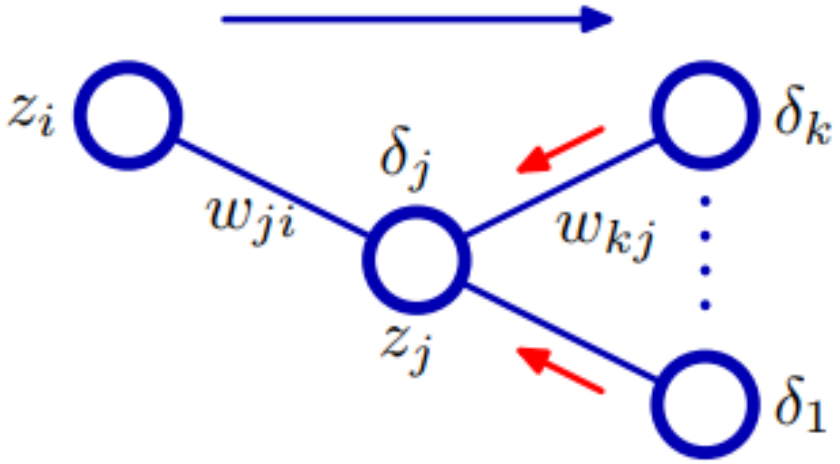
注意：对批量梯度下降方法而言，我们只需要将输出节点的误差用对应损失函数替换即可，如

$$\frac{\partial E}{\partial a_j} = \sum_{n \in B_i} \frac{\partial E_n}{\partial a_j}$$

神经网络中的反向传播算法（适用于随机梯度下降，续）

- **节点j的相关权重参数计算（见右图）**

一般形式 $\dfrac{\partial E_n}{\partial w_{ji}^l} = \dfrac{\partial E_n}{\partial a_j^l}\dfrac{\partial a_j^l}{\partial w_{ji}^l}$ ➡ $\dfrac{\partial E_n}{\partial w_{ji}^l} = \delta_j^l z_i^{l-1}$



- 由上式则有梯度更新公式，其中α 是学习率（超参数）：

$$\mathrm{w}_{ji}^{(l)} = \mathrm{w}_{ji}^{(l)} - \alpha \frac{\partial E_n}{\partial \mathrm{w}_{ji}^{(l)}}$$

课堂练习（作业）：一个三层反向传播（BP）神经网络，其输入层（含偏置项）、隐含层、输出层的节点维度分别为[D, M, K]，层与层之间为全连接结构，中间层的激活函数为h，输出层的激活函数为σ，损失函数为平方和误差函数。请根据反向传播算法，分别写出在随机梯度下降方式下，隐含层和输出层任意节点权重一次更新时，梯度的标量和形式的展开表达式。

综合：反向传播算法的向量形式表示

- **对一般前馈神经网络，定义**
  - 网络共 L 层，输入层为 l=0，输出层为 l=L；
  - 激活值（第l层的加权输入，未经激活函数激活）：$\mathbf{a}^l = \mathbf{W}^l \mathbf{z}^{l-1}$
  - 激活输出：$\mathbf{z}^l = \sigma(\mathbf{a}^l)$，其中$\sigma$为激活函数；
  - 损失函数L（如平方和误差，交叉熵等）
- **上述网络的反向传播公式的向量形式**
  - 输出层误差（⊙：Hadamard 积）：$\delta^L = \dfrac{\partial \mathcal{L}}{\partial \mathbf{a}^L} = \dfrac{\partial \mathcal{L}}{\partial \mathbf{z}^L} \odot \sigma'(\mathbf{a}^L)$
  - 隐藏层误差：$\delta^l = \left((\mathbf{W}^{l+1})^\top \delta^{l+1}\right) \odot \sigma'(\mathbf{a}^l)$
    - 其中 $\mathbf{W}^{l+1} \in \mathbb{R}^{n^{l+1} \times n^l}$ 为第l+1层的权重矩阵；
    - $\delta^{l+1} \in \mathbb{R}^{n^{l+1} \times 1}$ 为第 l+1 层的误差向量；
- **权重梯度矩阵（**利用误差 δ^l 和上一层激活输出 z^{l-1} 计算梯度）：

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} = \delta^l (\mathbf{z}^{l-1})^\top$$

- **公式构建过程说明**
  - 假设损失函数为均方误差：$\mathcal{L} = \frac{1}{2}\|\mathbf{t} - \mathbf{z}^L\|^2$
  - 则 $\dfrac{\partial \mathcal{L}}{\partial \mathbf{z}^L} = \mathbf{z}^L - \mathbf{t}$
  - 由定义得输出层误差：

$$\delta^L = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^L} = (\mathbf{z}^L - \mathbf{t}) \odot \sigma'(\mathbf{a}^L)$$

  - 通过链式法则，误差从l+1 层传递到 l 层，当前层误差：

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}^l} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{l+1}} \frac{\partial \mathbf{a}^{l+1}}{\partial \mathbf{z}^l} \frac{\partial \mathbf{z}^l}{\partial \mathbf{a}^l} \right)$$

展开到矩阵形式

根据标量形式分析（见之前讨论）：$\dfrac{\partial \mathbf{a}^{l+1}}{\partial \mathbf{z}^l} = \mathbf{W}^{l+1}$　$\dfrac{\partial \mathbf{z}^l}{\partial \mathbf{a}^l} = \mathrm{diag}(\sigma'(\mathbf{a}^l))$

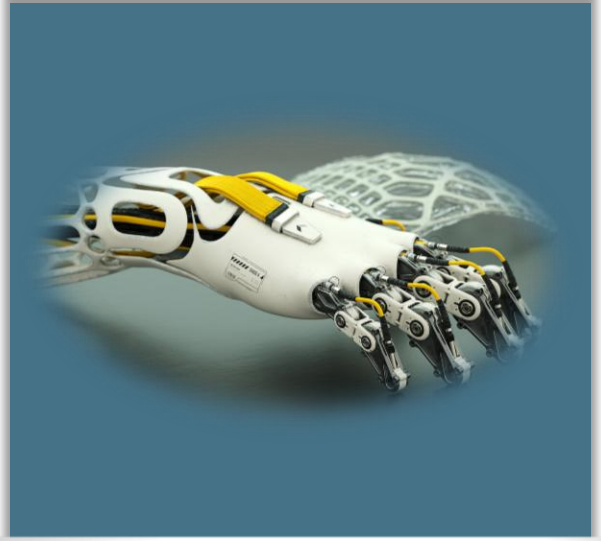
  - 权重W^l的梯度由误差 δ^l 和输入 z^{l-1} 的外积决定

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^l} = \delta_i^l \cdot z_j^{l-1}$$

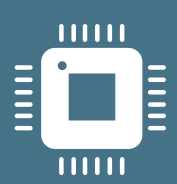
推广到矩阵形式

阶段性总结：多层前馈神经网络

- **多层前馈神经网络（或称多层感知机），是一种由若干独立神经元，分层构成的单向信号处理网络架构。在分类任务中，它可以看成是直接判别模型的一种（见第三、四讲讨论）。**
- **多层前馈神经网络层级连接规则和拓扑结构**
  - 仅相邻层之间通过带权重的有向边进行全连接（即每个神经元与下一层的所有神经元相连）；
  - 同一层内及跨层（非相邻层）的神经元间无直接连接。
  - 网络以有向无环图（DAG） 形式表示，信号严格向前单向传播（输入层→隐藏层→输出层）。
- **多层前馈神经网络的积木式搭建**
  - 单一输入层：每个输入层节点代表输入特征的一个分量。
  - 多层隐藏层：每个隐藏层节点代表一个线性模型和一个激活函数的组合。
  - 单一输出层：输出层的节点并不一定代表独立神经元，可能是单纯的线性层输出，可能是Sigmoid或Softmax等对应不同分类任务的函数。
  - **无隐藏层的情况**：输入直接经带权连接传递至输出层，模型退化为线性分类或回归模型。



前馈神经网络




1. 从神经元模型到前馈神经网络
2. 损失函数和反向传播算法
3. 贝叶斯神经网络
4. 神经网络的模型训练流程

贝叶斯神经网络：基本模型构造（以回归任务为例）

- **Bayesian神经网络可看做是概率生成模型的一种（见第三、四讲讨论）**
  - 基于人为假设的先验网络参数分布 $p(\mathbf{w}|\boldsymbol{\alpha})$，和给定样本特征条件下的连续目标值分布 $p(\boldsymbol{t}|\mathbf{x},\mathbf{w},\boldsymbol{\beta})$；
  - 估计后验概率$p(\mathbf{w}|\boldsymbol{D},\boldsymbol{\alpha},\boldsymbol{\beta})$，其中$D$为训练样本-标签集。
- **参数先验分布和条件概率预测分布假设**
  - 网络权重参数**w**的分布为高斯分布：$p(\mathbf{w}|\alpha)=\mathcal{N}(\mathbf{w}|\mathbf{0},\alpha^{-1}\mathbf{I})$；
  - 单样本特征条件下的标签分布为高斯分布：$p(t|\mathbf{x},\mathbf{w},\beta)=\mathcal{N}(t|y(\mathbf{x},\mathbf{w}),\beta^{-1})$；
  - 待估计网络权重参数后验概率分布：$p(\mathbf{w}|\mathcal{D},\alpha,\beta)$，其中$D$为所有训练样本-标签对的集合；
  - 根据贝叶斯公式，后验概率分布与两个分布假设的关系为：$p(\mathbf{w}|\mathcal{D},\alpha,\beta)\propto p(\mathbf{w}|\alpha)p(\mathcal{D}|\mathbf{w},\beta)$；
  - 考虑训练集中的样本为独立同分布（i.i.d）情况，由样本特征条件下的标签分布可得似然函数：

$$p(\mathcal{D}|\mathbf{w},\beta)=\prod_{n=1}^{N}\mathcal{N}(t_n|y(\mathbf{x}_n,\mathbf{w}),\beta^{-1})$$

  - 对后验概率求针对权重集**w**的最大化（局部最大化）值（即最大后验概率MAP），求对数有（推导略）：

$$\ln p(\mathbf{w}|\mathcal{D})=-\frac{\alpha}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}-\frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n,\mathbf{w})-t_n\}^2+\text{const}$$

(1) (2)

(1) 来自参数先验分布；
(2) 来自似然函数

贝叶斯神经网络：后验概率的高斯近似

- **后验概率的估计由MAP参数优化问题导出$\mathbf{w}_{\mathrm{MAP}}$：**
  - 对数后验概率（见上页）：$\ln p(\mathbf{w}|\mathcal{D}) = -\frac{\alpha}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} - \frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n,\mathbf{w}) - t_n\}^2 + \mathrm{const}$ （(1)、(2)）
  - $\mathbf{w}_{\mathrm{MAP}}$的求解：根据对数后验概率目标函数，使用标准反向传播算法配合一阶优化方法求解。
- **基于$\mathbf{w}_{\mathrm{MAP}}$对后验概率的高斯近似：**
  - 由对数后验概率的二阶导，得到高斯近似分布的协方差矩阵：；

$$\mathbf{A} = -\nabla\nabla \ln p(\mathbf{w}|\mathcal{D},\alpha,\beta) = \alpha\mathbf{I} + \beta\mathbf{H}$$

其中，H为对数后验概率分项（2）针对w的Hessian矩阵。

  - 由高斯分布近似的后验概率（基于拉普拉斯近似，推导略）：$q(\mathbf{w}|\mathcal{D}) = \mathcal{N}(\mathbf{w}|\mathbf{w}_{\mathrm{MAP}}, \mathbf{A}^{-1})$.
  - 根据高斯近似分布，对新样本的连续目标值概率估计（边缘分布）如下：

$$p(t|\mathbf{x},\mathcal{D}) = \int p(t|\mathbf{x},\mathbf{w})q(\mathbf{w}|\mathcal{D})\,\mathrm{d}\mathbf{w}$$

贝叶斯神经网络：处理标签估计概率无闭合表达式的情况

- **新样本目标值边缘概率估计——预测分布（接上页）：**

$$p(t|\mathbf{x},\mathcal{D}) = \int p(t|\mathbf{x},\mathbf{w})q(\mathbf{w}|\mathcal{D})\,\mathrm{d}\mathbf{w}.$$

- **对上式的估计如下展开：**
  - 对神经网络的输出针对网络权重参数做一阶泰勒展开，有：

$$y(\mathbf{x},\mathbf{w}) \simeq y(\mathbf{x},\mathbf{w}_{\mathrm{MAP}}) + \mathbf{g}^{\mathrm{T}}(\mathbf{w}-\mathbf{w}_{\mathrm{MAP}}) \quad \text{其中} \quad \mathbf{g} = \nabla_{\mathbf{w}} y(\mathbf{x},\mathbf{w})|_{\mathbf{w}=\mathbf{w}_{\mathrm{MAP}}}$$

  - 由样本特征条件下的标签分布 $p(\boldsymbol{t}|\mathbf{x},\mathbf{w},\boldsymbol{\beta})$ 为高斯分布，有：

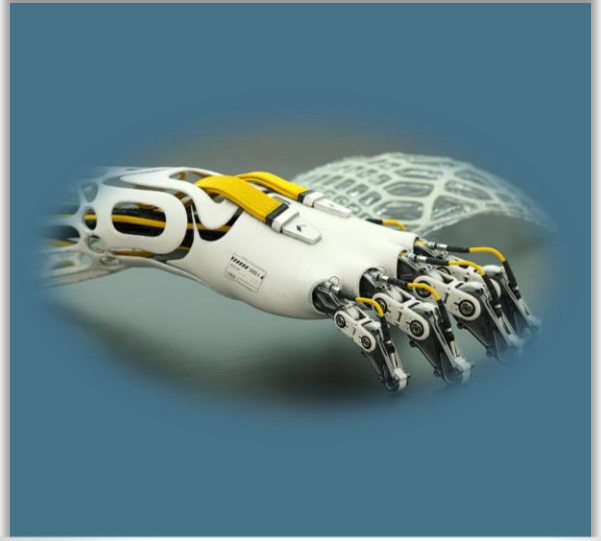
$$p(t|\mathbf{x},\mathbf{w},\beta) \simeq \mathcal{N}\left(t|y(\mathbf{x},\mathbf{w}_{\mathrm{MAP}}) + \mathbf{g}^{\mathrm{T}}(\mathbf{w}-\mathbf{w}_{\mathrm{MAP}}),\beta^{-1}\right)$$

对比原始分布假设 $p(t|\mathbf{x},\mathbf{w},\beta) = \mathcal{N}(t|y(\mathbf{x},\mathbf{w}),\beta^{-1})$

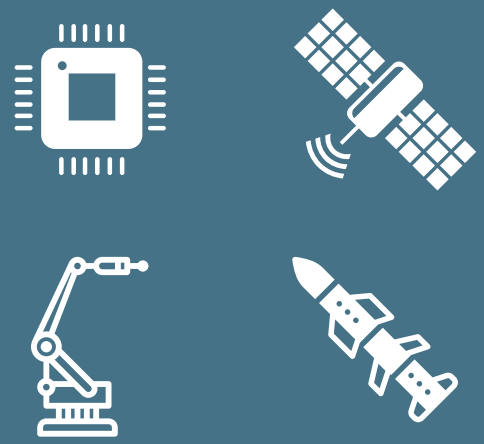
  - 目标预测值的分布可由以下形式估计：

$$p(t|\mathbf{x},\mathcal{D},\alpha,\beta) = \mathcal{N}\left(t|y(\mathbf{x},\mathbf{w}_{\mathrm{MAP}}),\sigma^2(\mathbf{x})\right) \quad \text{其中} \quad \sigma^2(\mathbf{x}) = \beta^{-1} + \mathbf{g}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{g}.$$

注：先验概率分布精度参数值α和条件概率分布参数值β的估计可根据 $p(\mathcal{D}|\alpha,\beta) = \int p(\mathcal{D}|\mathbf{w},\beta)p(\mathbf{w}|\alpha)\,\mathrm{d}\mathbf{w}$，即基于训练集数据的似然函数的拉普拉斯近似，在$\mathbf{w}_{\mathrm{MAP}}$已知的情况下分别针对$\alpha$和$\beta$求取最大值得到。（具体推导参见本课程参考书Bishop，PRML）



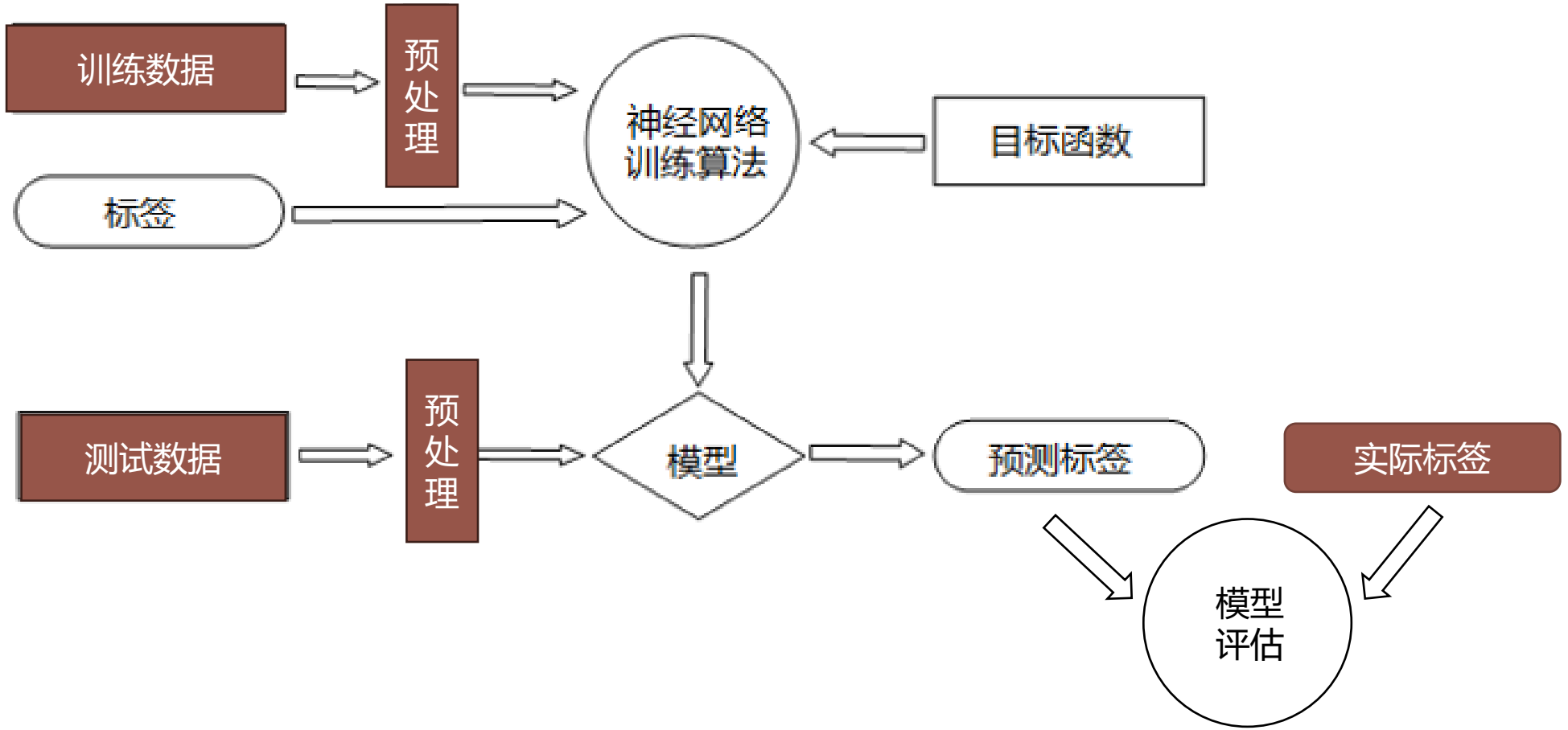
前馈神经网络



1. 从神经元模型到前馈神经网络
2. 损失函数和反向传播算法
3. 贝叶斯神经网络
4. 神经网络的模型训练流程

神经网络的模型训练流程

构建一个神经网络模型大致可分为数据准备与预处理、模型初始化、确定优化目标函数、模型训练（模型优化求解）和模型评估（验证模型性能）这五个基本步骤。



数据准备与预处理

- **Step1、针对任务需求收集样本并对其进行标注**
  - 常见数据标注工具：LabelImg（图像），CVAT（视频），BRAT（文本），SUSTechPoints（3D点云）。
- **Step2、通过适当的样本集增强方式实现对训练样本集扩充。**
- **Step3、将带标注样本集划分为两部分**
  - 其中一部分样本子集作为训练集用于模型训练；
  - 其余部分样本作为测试集用于验证模型性能。
- **Step4、对数据进行特征提取等预处理。**
- **Step5、采用合适方式对标签数据进行编码。**

程序框架示例（以算法库Pytorch为例）：设计DataLoader模块完成数据预处理

- **DataLoader模块的作用**
  - 在一般的有监督人工智能模型中，需要设计Dataloader模块，以负责训练数据的读取、预处理、批处理和打乱顺序。
- **通用DataLoader设计模式下所含核心流程**
  - 数据集定义：加载原始数据并定义样本结构。
  - 预处理：数据清洗、归一化、增强。
  - 批处理：将样本打包为批量数据。
  - 迭代器：按批次返回数据，支持随机打乱和多进程加载。
- **以Pytorch框架下的DataLoader设计模式为例**
  - 自有数据集定义类CustomDataset的设计（右图）。
  - CustomDataset类的目的是为DataLoader提供数据存储与单样本访问的标准化接口，而DataLoader基于此接口创建可调用的采样迭代器，实现批量加载与多进程优化。

```python
import torch
from torch.utils.data import Dataset, DataLoader

class CustomDataset(Dataset):
    def __init__(self, data_path, transform=None):
        """
        初始化：加载原始数据（如CSV、图像路径等）
        - data_path: 数据存储路径
        - transform: 数据预处理/增强函数
        """
        self.data = load_data(data_path)  # 自定义数据加载函数
        self.transform = transform

    def __len__(self):
        """返回数据集的总样本数"""
        return len(self.data)

    def __getitem__(self, idx):
        """
        根据索引返回单个样本（自动被DataLoader调用）
        - idx: 样本索引
        """
        sample = self.data[idx]
        if self.transform:
            sample = self.transform(sample)  # 应用预处理
        return sample  # 返回 (feature, label)
```

框架示例：设计DataLoader模块完成数据预处理——使用第三方算法库Pytorch（续）

- **（接上页）CustomDataset模块中使用Transforms类实现数据的预处理（见右图代码）**
  - Transforms类所提供的预处理功能：归一化、数组等类型到张量（tensor）的转换；随机裁剪、翻转、旋转、缩放；以及自定义的基于以上操作的多个操作顺序构成的组合变换；
  - 以图像样本（如MNIST数据集中的手写体图像）为例；
  - 关键步骤：调整尺寸 → 数据增强 → 转 Tensor → 标准化。
- **创建DataLoader类的实例：加载过程分以下部分（对应初始化参数，见右图代码）**
  - Dataset：存储样本及其对应的标签，提供基本的数据访问接口。
  - Sampler：定义数据采样策略（如随机采样、顺序采样等）。
  - Collate_fn：定义如何将单个样本组合成一个 batch。
  - Worker：负责实际的数据加载和预处理任务。

```python
from torchvision import transforms

# 示例：图像数据预处理（组合操作）
transform = transforms.Compose([
    transforms.Resize(256),            # 调整大小
    transforms.RandomHorizontalFlip(),  # 数据增强
    # 转为Tensor，以便硬件加速（如GPU加速）
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])  # 归一化
])
```

```python
# 初始化数据集
dataset = CustomDataset(data_path='./data', transform=transform)

# 创建 DataLoader
dataloader = DataLoader(
    dataset,
    batch_size=32,       # 批量大小
    shuffle=True,        # 是否打乱数据（训练集通常为True）
    num_workers=4,       # 多进程加载的线程数
    drop_last=False,     # 是否丢弃最后不足一个batch的数据
    pin_memory=True      # 是否将数据拷贝到CUDA固定内存（加速GPU传输）
)
```

回顾：样本标准-归一化（见线性模型部分讲座）

- **对数据集中的M维样本x进行归一化：**
  - 关键步骤：先做均值中心化（标准化），再利用样本的标准差对每个样本进行缩放（归一化）。
  - 定义样本集中第n个样本$\boldsymbol{x}_n$的第m个分量为$\boldsymbol{x}_{n,m}$；
  - 标准-归一化公式：

$$\hat{\mathbf{x}}_{n,m} = \frac{\mathbf{x}_{n,m} - \mu_m}{\sigma_m}$$

  - 样本均值计算公式：

$$\mu_m = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_{n,m}$$

  - 样本标准差计算：

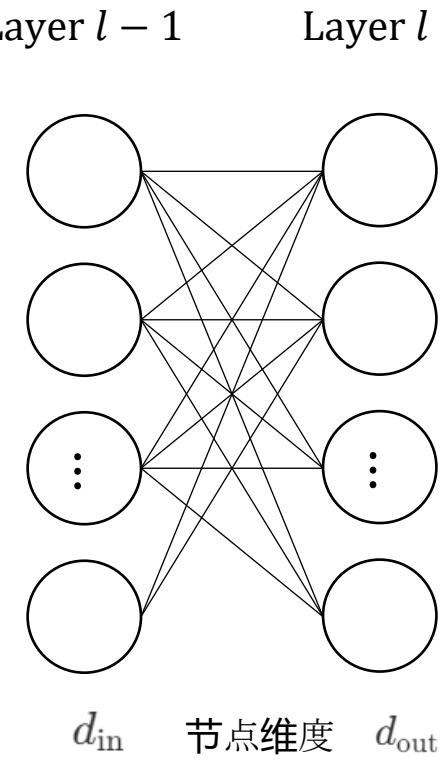
$$\sigma_m = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_{n,m} - \mu_m)^2}$$

神经网络的训练

- **模型初始化**
  - 模型初始化参数一般有：连接权重+偏置项的初始化、超参数的选择等。
  - 模型初始化过程确定了模型优化过程从何处开始，从一组较好的模型参数开始的训练过程通常能够避免参数陷入局部最优并获得性能较好的优化模型。
- **确定优化目标**
  - 针对不同类型的实际任务，通常所使用的目标函数（损失函数）形式也有所不同。
- **训练过程中缓解过拟合问题**
  - 在模型优化的目标函数中添加正则化项以约束模型参数的取值（参见线性回归专题）；
- **模型训练**
  - 优化算法主要是依据梯度和误差进行参数更新；
  - 通过网络模型前向计算求得目标函数值，进而使用梯度下降等方法对目标函数进行迭代优化；
  - 反向传播算法通过自动微分机计算大规模网络参数梯度。

模型初始化方法

- **Xavier初始化方法**
  - 核心思想：保持每一层输入和输出的方差一致，从而确保信号在前向传播和反向传播过程中既不被过度放大（爆炸）也不被过度缩小（消失）。
  - 基本假设：
    - 激活函数对称且在零点附近邻域近似线性（如tanh）。
    - 每层的输入和初始化权重分别为为独立分布（如标准高斯分布），均值为0。



- **原理**
  - 对任意层的输入（激活值），其写为矩阵形式有 $\mathbf{a}^l = W\mathbf{x}^{l-1} + \mathbf{b}^l$。
  - 激活值每个元素的方差（忽略偏置项）：$\mathrm{Var}(a_i^l) = \mathrm{Var}\left(\sum_{j=1}^{d_{\text{in}}} W_{i,j}^l x_j^{l-1}\right)$
  - 由于独立分布，有：

$$\mathrm{Var}(a_i^l) = \sum_{j=1}^{d_{\text{in}}} \mathrm{Var}(W_{i,j}^l) \cdot \mathrm{Var}(x_j^{l-1}) = d_{\text{in}} \cdot \sigma_W^2 \cdot \sigma_{\mathbf{x}^{l-1}}^2$$

- **初始化操作**
  - **令 $\sigma_W^2 = \dfrac{1}{d_{\text{in}}}$，由此保证激活与输入的方差一致。**

（对比）Kaiming初始化：专用于ReLU激活函数后：

$\sigma_W^2 = \dfrac{2}{d_{\text{in}}}$ 这是用于补偿ReLU函数的方差减半效应。

减少对初始化的敏感并缓解过拟合：添加归一化层

- **归一化操作的作用**
  - 加速收敛：使得神经网络每层的输入具有相似的分布，并控制激活值的范围，从而减少梯度消失（逐渐趋近于0）或爆炸（逐渐变大）的可能性，加速模型的收敛。
  - 避免过拟合：归一化层通过对数据进行标准化处理，减少了模型对特定输入分布的依赖性，从而增强了模型的鲁棒性和泛化能力。
  - 减少对初始化的敏感性：通过对每一层的输入进行标准化，降低了模型对初始权重的敏感性，使得模型更容易训练。
- **归一化方法分类**
  - 批归一化，Batch Normalization (BN)：对每个小批量数据进行归一化，因此批量大小必须大于1
    - 注意：在推理时，并不存在样本小批量集的概念，BN层直接使用训练阶段积累的移动平均均值和方差，而非当前输入样本的统计量。
  - 层归一化，Layer Normalization （LN）：相当于在相邻神经元层中添加操作，对均值和方差的计算来自当前层所有神经元的线性输入。

归一化操作方法

- **批归一化**
  - 训练时，对每个批次样本的特征分量维度计算统计特征，对含m个样本的批次B的所有样本中同一分量（分量ID略）而言：
    - 计算批次均值：$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$
    - 计算批次方差：$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$
  - 标准化-归一化：

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

ϵ 为防止除零的小常数

  - 为保证在相似输入时不同神经元的激活程度有所不同，引入可学习调整参数γ，β：

$$y_i = \gamma\hat{x}_i + \beta$$

缩放与平移操作

- **层归一化**
  - 对某层输入$\mathbf{x}=[x_1, x_2, \ldots, x_H]$（H 为该层神经元数量），计算均值和方差
    - 该层所有神经元的均值：$\mu = \frac{1}{H}\sum_{i=1}^{H} x_i$
    - 该层所有神经元的方差：$\sigma^2 = \frac{1}{H}\sum_{i=1}^{H}(x_i - \mu)^2$
  - 标准化-归一化：

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

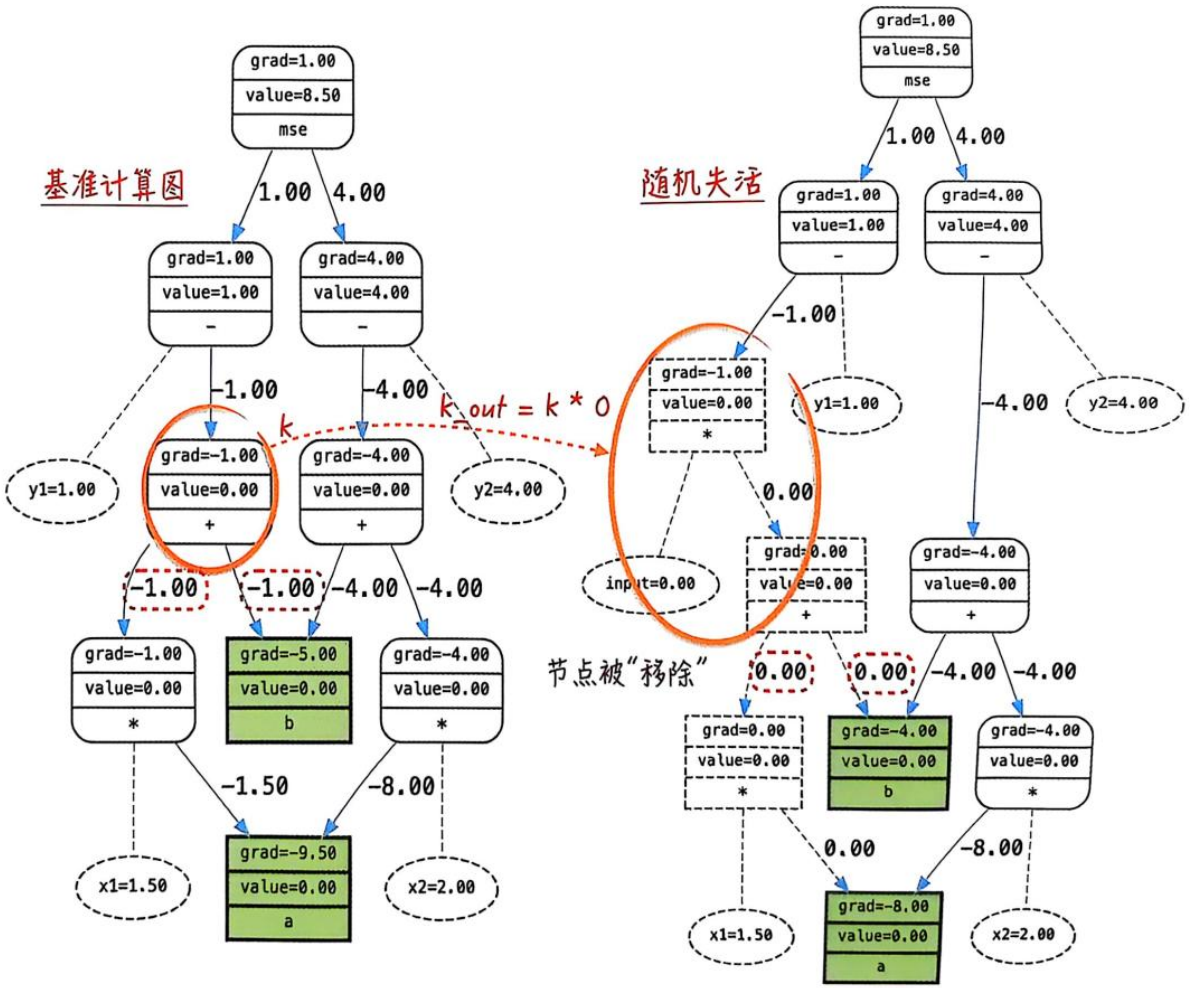
  - 为保证在相似输入时不同神经元的激活程度有所不同，引入可学习调整参数γ，β：

$$y_i = \gamma\hat{x}_i + \beta$$

缓解过拟合：随机失活（Dropout）

- **随机失活（Dropout）通过在训练过程中以概率p随机“丢弃”一部分神经元（即将其输出乘以0），从而减少神经网络对特定神经元的依赖，增强模型的泛化能力**
  - 作用1：降低模型复杂度，使得参与计算的网络参数减少，从而避免模型过于复杂而过度拟合训练数据。
  - 作用2：通过随机关闭部分神经元，减少了神经元之间的隐性协同适应（co-dependence），使每个神经元都更加独立地工作，提升了模型的鲁棒性。
  - 作用3：实现模型集成效果，每次训练中，随机失活相当于训练了一个不同的子网络，最终的结果可以视为多个子网络的集成预测，从而提高了泛化性能。
  - 节点失活导致反向传播的梯度（误差值）也变为0，等价于在计算图中删除失活节点（如右图）

Dropout导致反向传播中计算图的改变



图来源：《解构大语言模型：从线性回归到通用人工智能》

神经网络模型的验证

- **模型的验证**
  - 模型的验证往往是一个冗长的过程：若模型性能未达到任务需求，则需重新设定超参数并构造优化模型。
  - 对不同类型的任务而言，用于描述模型性能的度量指标也有所不同
    - 分类任务性能度量指标：正确率和错误率、查准率和查全率、F_1值等（参见线性分类器专题）；
    - 回归任务性能度量指标：均方误差、决定系数R^2等（参见线性回归模型专题）；
- **Cross Validation**
  - 由于模型的随机性（来自训练数据集和输入数据），采用N-fold交叉验证的形式估计模型性能；注意对训练集划分为N等分，其中选取（N-1）份大小的训练集及1份大小的验证集。
  - 交叉验证不代表所获得的N次训练后模型的估计是相互独立的。

模块综合：基于第三方算法库实现多层感知机

- **基于Pytorch设计所需模块（以类的形式封装）**
  - **MLP模块：负责定义多层感知机的结构和前向传播过程。初始化时需要指定输入层、隐藏层和输出层的维度大小；**
  - 数据处理模块：负责对数据进行预处理和加载。创建DataLoader对象，用于模型训练和评估的数据导入。
  - ModelTrainer模块：负责实现模型的训练过程。包括模型的初始化、优化器和损失函数的设置、训练循环的执行等。还提供了保存和加载模型的方法。
  - 模型评估模块：用于评估模型的性能。需要根据指定的评估指标，对模型在测试数据上的表现进行评估，并返回评估结果。

```python
# 模型定义模块
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dims, output_dim):
        """
        参数:
            input_dim (int): 输入层的维度大小。
            hidden_dims (list): 一个整数列表，指定了每个隐藏层的节点数。
            output_dim (int): 输出层的维度大小。
        """
        super(MLP, self).__init__()
        pass

    def forward(self, x):
        """
        定义前向传播过程。
        参数:
            x (Tensor): 输入数据。
        返回:
            Tensor: 输出数据。
        """
        pass
```

模块综合：基于第三方算法库实现多层感知机

- **基于Pytorch设计所需模块（以类的形式封装）**
  - 模型定义模块：负责定义多层感知机的结构和前向传播过程。初始化时需要指定输入层、隐藏层和输出层的维度大小；
  - DataProcessor**模块：负责对数据进行预处理和加载。创建DataLoader对象（见前文讨论），用于模型训练和评估的数据导入。**
  - 模型训练模块：负责实现模型的训练过程。包括模型的初始化、优化器和损失函数的设置、训练循环的执行等。还提供了保存和加载模型的方法。
  - 模型评估模块：用于评估模型的性能。需要根据指定的评估指标，对模型在测试数据上的表现进行评估，并返回评估结果。

```python
# 数据处理模块
class DataProcessor:
    def __init__(self, x, y, batch_size=32):
        """
        参数:
            x: 特征数据; y: 标签数据。
            batch_size (int): 每个批次的样本数量。
        """
        pass

    def create_data_loader(self):
        """
        创建并返回PyTorch的DataLoader对象。
        返回: DataLoader: 数据加载器。
        """
        pass

    def preprocess_data(self, x, y):
        """
        对数据进行预处理，如标准化、归一化等。
        返回: tuple: 预处理后的特征和标签数据。
        """
        pass
```

模块综合：基于第三方算法库实现多层感知机

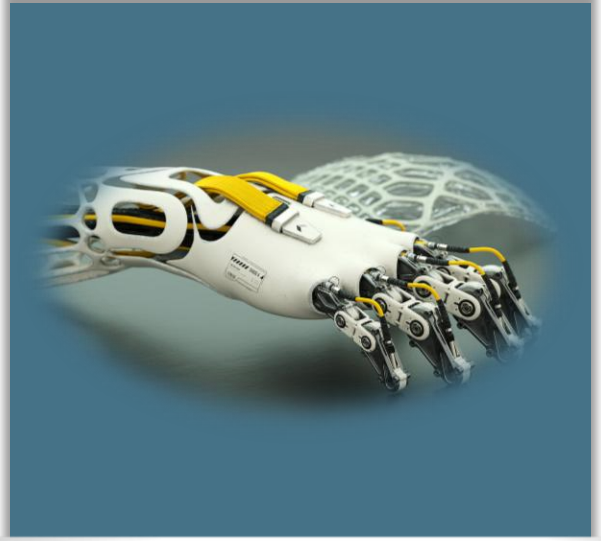
- **基于Pytorch设计所需模块（以类的形式封装）**
  - 模型定义模块：负责定义多层感知机的结构和前向传播过程。初始化时需要指定输入层、隐藏层和输出层的维度大小；
  - 数据处理模块：负责对数据进行预处理和加载。创建DataLoader对象，用于模型训练和评估的数据导入。
  - **ModelTrainer模块：负责实现模型的训练过程。包括模型的初始化、优化器和损失函数的设置、训练循环的执行等。还提供了保存和加载模型的方法。**
  - 模型评估模块：用于评估模型的性能。需要根据指定的评估指标，对模型在测试数据上的表现进行评估，并返回评估结果。

```python
# 模型训练模块
class ModelTrainer:
    def __init__(self, model, optimizer, criterion, scheduler=None):
        """
        参数:
            model (MLP): 要训练的多层感知机模型。
            optimizer (torch.optim.Optimizer): 优化器。
            criterion (nn.Module): 损失函数。
            scheduler (torch.optim.lr_scheduler._LRScheduler): 学习率调度器。
        """
        pass

    # 训练模型
    def train(self, train_loader, epochs, val_loader=None):
        pass

    # 保存训练好的模型。
    def save_model(self, model, path)
        pass

    def load_model(self, model, path):
        """
        加载预训练的模型。
        返回:
            MLP: 加载了预训练权重的模型。
        """
        pass
```

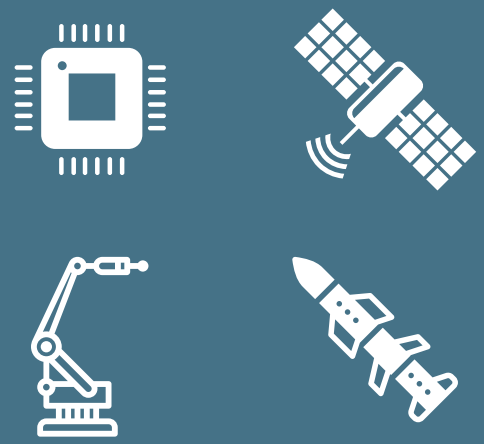
模块综合：基于第三方算法库实现多层感知机

- **基于Pytorch设计所需模块（以类的形式封装）**
  - 模型定义模块：负责定义多层感知机的结构和前向传播过程。初始化时需要指定输入层、隐藏层和输出层的维度大小；
  - 数据处理模块：负责对数据进行预处理和加载。创建DataLoader对象，用于模型训练和评估的数据导入。
  - 模型训练模块：负责实现模型的训练过程。包括模型的初始化、优化器和损失函数的设置、训练循环的执行等。还提供了保存和加载模型的方法。
  - ModelEvaluator**模块：用于评估模型的性能。需要根据指定的评估指标，对模型在测试数据上的表现进行评估，并返回评估结果。**

```python
# 模型评估模块
class ModelEvaluator:
    def __init__(self, model, metrics):
        """
        参数:
            model (MLP): 要评估的多层感知机模型。
            metrics (dict): 评估指标。
        """
        pass

    def evaluate(self, test_loader):
        """
        评估模型性能。
        参数:
            test_loader (DataLoader): 测试数据加载器。
        返回:
            dict: 包含各种评估指标的结果。
        """
        pass
```



前馈神经网络



讨论