神经网络

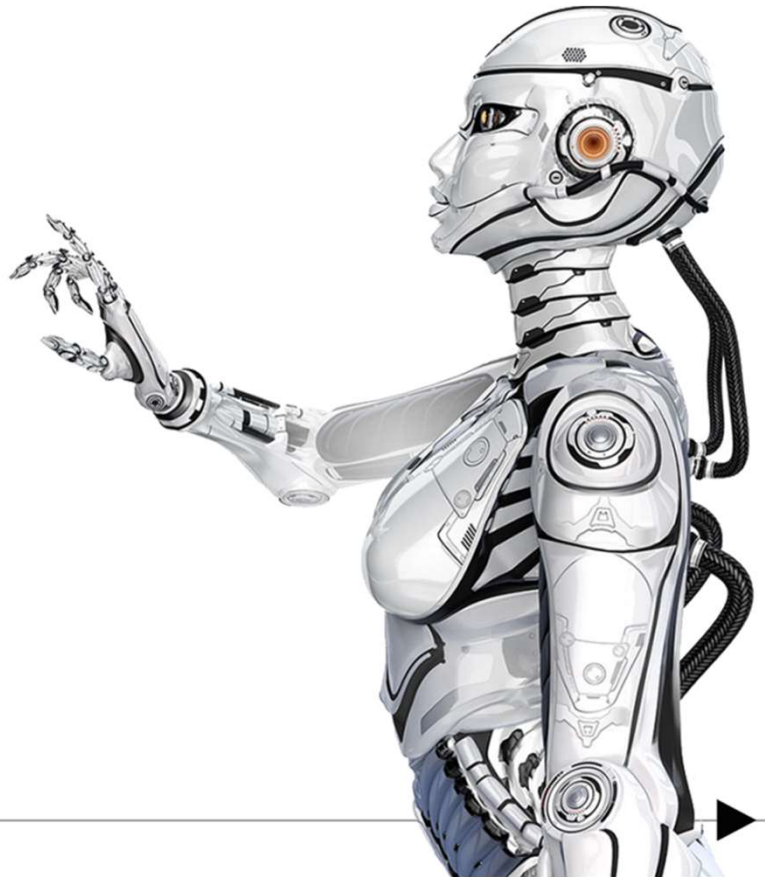
汇报人：孙静安　　时间：2020.12.03

目录

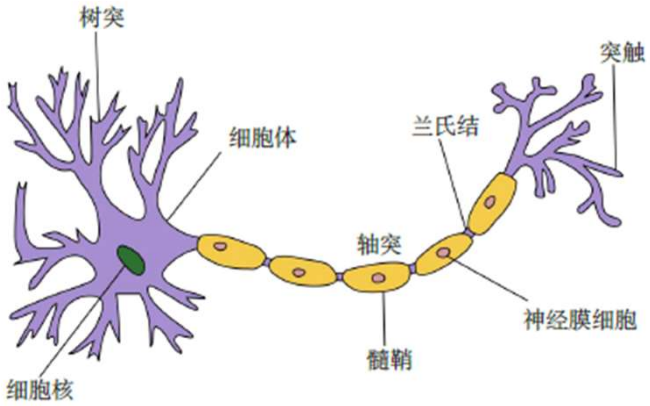
CONTENTS

01. 什么是神经网络

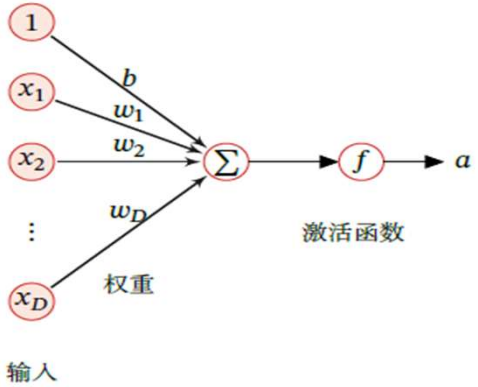
Overview of annual work



1.1 从人脑到人工



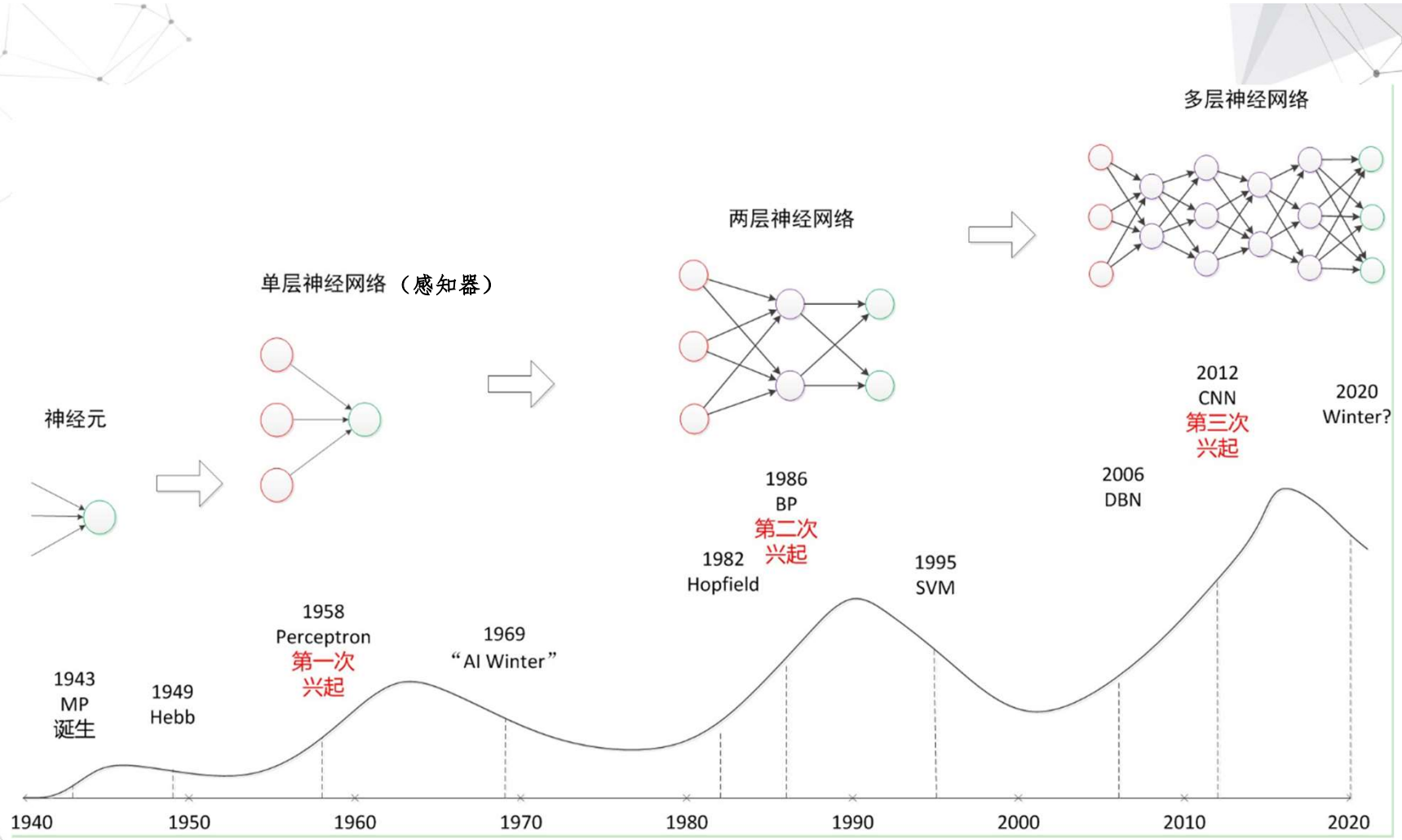
一个典型的神经元结构



一个典型的人工神经元（MP模型）

$$y = f(x_1 w_1 + x_2 w_2 + \cdots + x_l w_l + b) \quad = f(\boldsymbol{w}^T \boldsymbol{x} + b)$$

1.2 神经网络发展历史



三起三落的神经网络

1.3 感知机

感知机是一种**二分类线性分类**模型。可以看成**激活函数为$f(x)=\text{sign}(x)$的单层神经网络**

问题：给定一个数据集：

$$D=\{(x_1,y_1),(x_2,y_2),...(x_N,y_N)\}$$

其中x_i为特征信息，$y_i\in\{+1,-1\}$为相应的label，表示该实例属于哪一类，现在需要训练一个模型$\hat{y}_i=F(x_i)$，能对数据进行正确的分类。

策略：感知机采用线性分类。

假设存在一个超平面 w^Tx_i+b，它可以将实例正确地划分到它的两侧，对于y_i =+1的实例i，$w^Tx_i+b>0$；对于对于y_i =−1的实例i，$w^Tx_i+b<0$。对应到神经网络中就是将单层神经网络的激活函数取为$f(x)=\text{sign}(x)$，即：

$$\hat{y}_i=f(\mathbf{w^T}\mathbf{x}_i+b)=\begin{cases}+1 & if \quad \mathbf{w^T}\mathbf{x_i}+b>0\\ -1 & if \quad \mathbf{w^T}\mathbf{x}_i+b<0\end{cases}$$

损失函数：计算错误分类点到该超平面的距离。

首先一个点到超平面的距离为：

$$\frac{1}{||w||}|\boldsymbol{w}^T\boldsymbol{x}_i + b|$$

根据感知机的学习策略，所谓错误分类点就是对$y_i = +1$的实例，模型却算出，$\boldsymbol{w}^T\boldsymbol{x_i} + b < 0$，继而将其分为-1类；$y_i = -1$的实例，模型却算出，$\boldsymbol{w}^T\boldsymbol{x_i} + b > 0$，继而将其分为+1类，所以对错误分类点 $-y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) > 0$，恒成立，那么错误点到超平面的距离为：

$$-\frac{1}{||w||}y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b)$$

所有错误分类点到超平面距离为：

$$\sum_{i\in M} -\frac{1}{||w||}y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b)$$

不考虑$\frac{1}{||w||}$，就得到感知机的损失函数：

$$Loss = -y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b)$$

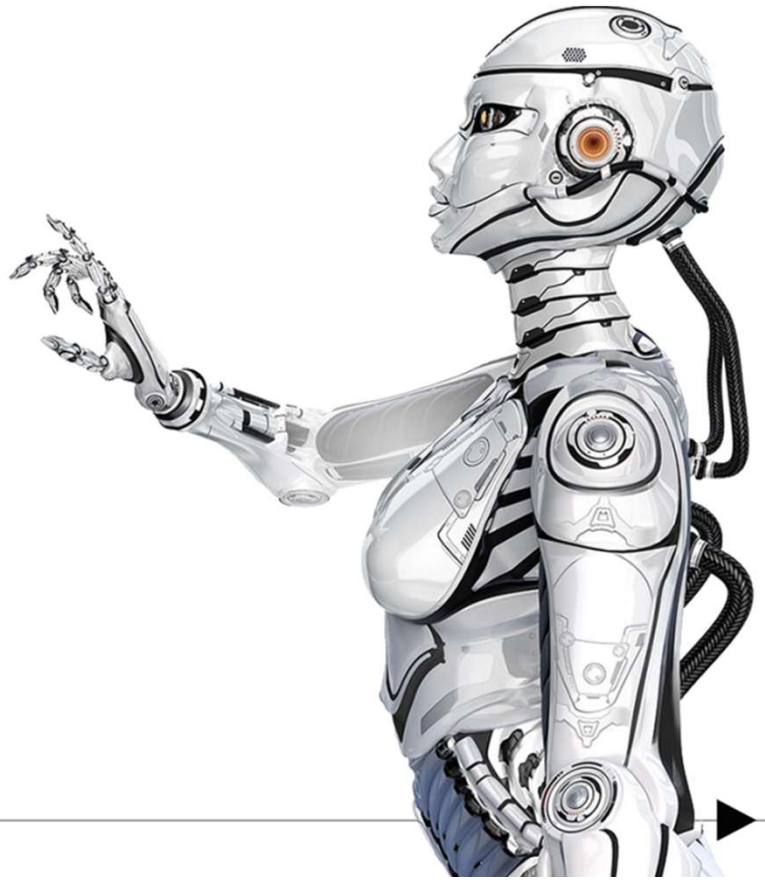
最终问题转化为求解$Loss$的极值问题，采用梯度下降法，如果某个$-y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) > 0$，即该实例为误分类点，参数进行更新：

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha\frac{\partial Loss}{\partial \boldsymbol{w}} = \boldsymbol{w} + \alpha y_i\boldsymbol{x}_i$$

$$b \leftarrow b - \alpha\frac{\partial Loss}{\partial b} = b + \alpha y_i$$

直到没有误分类点

02. 前馈神经网络



2.1 激活函数

激活函数的性质:

激活函数是神经元非常重要的一部分，为了加强网络的表示能力和学习能力，激活函数需要以下几点性质：

（1）连续并可导（允许少数点上不可导）的非线性函数．可导的激活函数可以直接利用数值优化的方法来学习网络参数．

（2） 激活函数及其导函数要尽可能的简单，有利于提高网络计算效率．

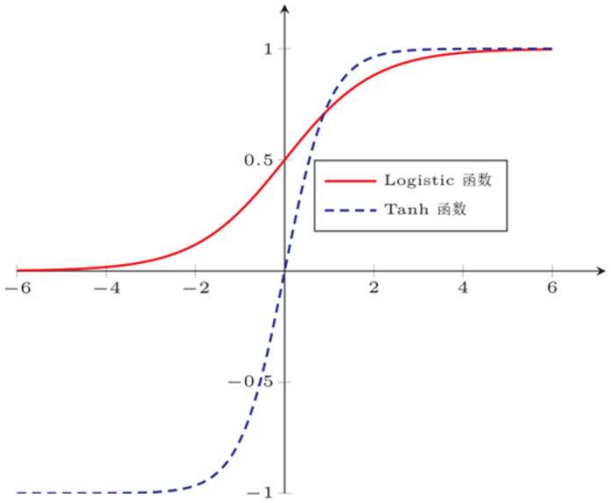
（3） 激活函数的导函数的值域要在一个合适的区间内，不能太大也不能太小，否则会影响训练的效率和稳定性．

常见的激活函数

Sigmoid型函数：

Logistic函数：$$\sigma(x) = \frac{1}{1+\exp(-x)}$$

Tanh函数：$$\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$$



因为Logistic 函数的性质，使得装备了Logistic 激活函数的神经元具有以下两点性质：

1）其输出直接可以看作概率分布，使得神经网络可以更好地和统计学习模型进行结合.

2）其可以看作一个软性门（Soft Gate），用来控制其他神经元输出信息的数量.

但Logistic函数的输出恒大于0. 而非零中心化的输出会使得其后一层的神经元的输入发生偏置偏移（Bias Shift），并进一步使得梯度下降的收敛速度变慢.

ReLU函数：

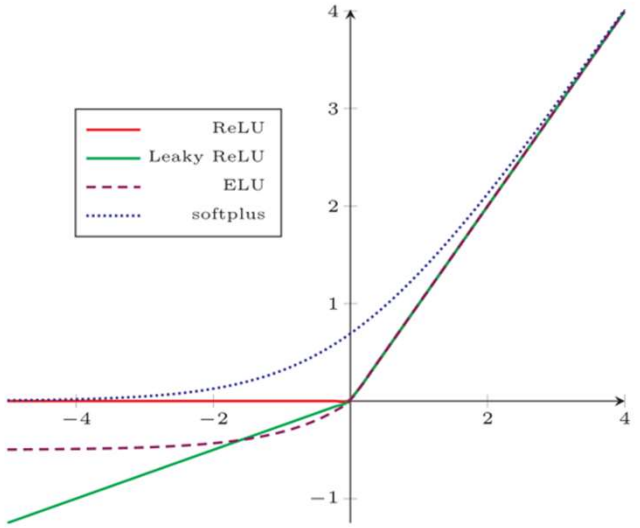
$$\text{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$
$$= \max(0, x).$$

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma x & \text{if } x \leq 0 \end{cases}$$

$$\text{PReLU}_i(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma_i x & \text{if } x \leq 0 \end{cases}$$

$$\text{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$
$$= \max(0, x) + \min(0, \gamma(\exp(x) - 1))$$

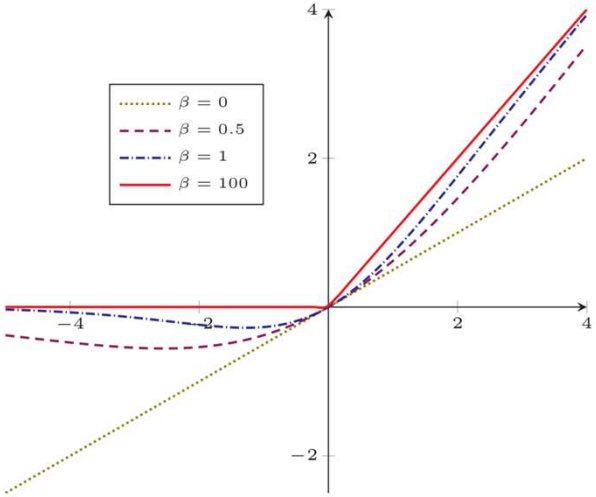
$$\text{softplus}(x) = \log(1 + \exp(x))$$



1）计算上更加高效
2）生物学合理性：单侧抑制、宽兴奋边界
3）它在$x > 0$时导数为1，在一定程度上缓解梯度消失问题，加速了梯度下降的收敛速度
1）输出非零中心化，给后层网络引入了偏置偏移
2）死亡ReLU：很有可能在参数更新后，某个神经元的输出为负数，这就导致后层全部神经元不能被激活，该神经元对自身参数的梯度也为0，这就意味着该神经元永远不能被激活

Swish函数

$$\text{swish}(x) = x\sigma(\beta x)$$



高斯误差线性单元（Gaussian Error Linear Unit，GELU）

$$\text{GELU}(x) = xP(X \leq x)$$

其中$P(X \leq x)$是高斯分布N(μ, σ^2)的累积分布函数，其中μ, σ为超参数，一般设$\mu = 0, \sigma = 1$即可

由于高斯分布的累积分布函数为S型函数，因此GELU可以用Tanh函数或Logistic函数来近似

$$\text{GELU}(x) \approx 0.5x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right)\right)$$

或 $$\text{GELU}(x) \approx x\sigma(1.702x).$$

2.2 前馈网络

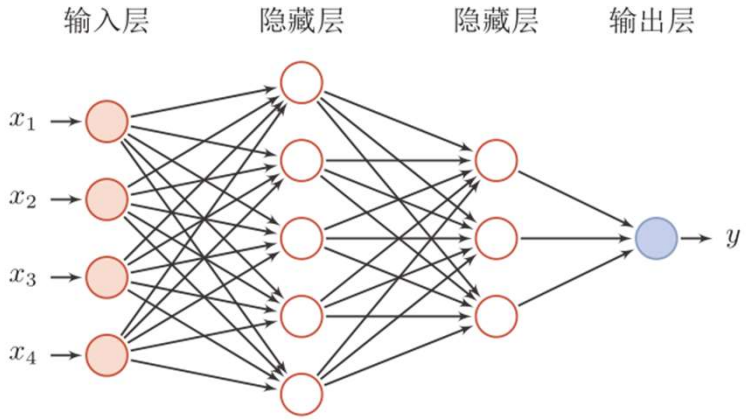
前馈神经网络（全连接神经网络、多层感知器）

特点：

1）各神经元分别属于不同的层，**层内无连接**。

2）相邻两层之间的神经元**全部两两连接**。

3）整个网络中**无反馈**，信号从输入层向输出层单向传播，可用一个有向无环图表示。



第0层称为输入层，最后一层称为输出层，其他中间层称为隐藏层

记号	含义
L	神经网络的层数
M_l	第l层神经元的个数
$f_l(\cdot)$	第l层神经元的激活函数
$\boldsymbol{W}^{(l)} \in \mathbb{R}^{M_l \times M_{l-1}}$	第$l-1$层到第l层的权重矩阵
$\boldsymbol{b}^{(l)} \in \mathbb{R}^{M_l}$	第$l-1$层到第l层的偏置
$\boldsymbol{z}^{(l)} \in \mathbb{R}^{M_l}$	第l层神经元的净输入（净活性值）
$\boldsymbol{a}^{(l)} \in \mathbb{R}^{M_l}$	第l层神经元的输出（活性值）

$$
\begin{aligned}
\mathbf{z}_1^{(1)} &= w_{11}x_1 + w_{12}x_2 + \cdots + w_{1M_0}x_{M_0} + b_1^{(1)} \\
\mathbf{z}_2^{(1)} &= w_{21}x_1 + w_{22}x_2 + \cdots + w_{2M_0}x_{M_0} + b_2^{(1)} \\
&\vdots \\
\mathbf{z}_{M_1}^{(1)} &= w_{M_1 1}x_1 + w_{M_1 2}x_2 + \cdots + w_{M_1 M_0}x_{M_0} + b_{M_0}^{(1)} \\
\mathbf{z}^{(1)} &= W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}
\end{aligned}
$$

设$\boldsymbol{a^{(0)} = x}$，前馈神经网络通过迭代下述公式进行信息传播:

$$
\begin{aligned}
\boldsymbol{z}^{(l)} &= \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}, \\
\boldsymbol{a}^{(l)} &= f_l(\boldsymbol{z}^{(l)}).
\end{aligned}
$$

首先通过第$\boldsymbol{l-1}$层神经元的活性值（输出） $\boldsymbol{a^{(l-1)}}$进行权重求和计算出第$\boldsymbol{l}$层神经元的净活性值（输入）$\boldsymbol{z^{(l)}}$，再代入该层激活函数，得到第$\boldsymbol{l}$层神经元的活性值$\boldsymbol{a^{(l)}}$

$$
\boldsymbol{x = a^{(0)} \rightarrow z^{(1)} \rightarrow a^{(1)} \rightarrow z^{(2)} \rightarrow \cdots \rightarrow a^{(L-1)} \rightarrow z^{(L)} \rightarrow a^{(L)} = \phi(x; W, b))}
$$

2.3 应用到机器学习

通用近似定理：

对于一个前馈神经网络，如果它至少有一层隐藏层，并且具有线性输出层，那么只要给予网络足够数量的神经元，便可以实现以足够高精度来逼近任意一个定义在某个R^n紧子集上的连续函数$f(\boldsymbol{x})$。

所谓线性输出层：输出层只有一个神经元，并且激活函数为恒等函数

也就是对于任意的$f(\boldsymbol{x})$，存在整数M，实数向量$\boldsymbol{w}_m$，一组实数v_m, b_m，使得对任意的$\epsilon > 0$，都有：

$$\left| \sum_{m=1}^{M} v_m \phi(\mathbf{w}_\mathbf{m}^\mathrm{T}\mathbf{x} + b_m) - f(\mathbf{x}) \right| < \epsilon$$

这里的$\boldsymbol{\phi}(\cdot)$就是隐藏层激活函数。

通用近似定理只是说明了神经网络的计算能力可以去近似一个给定的连续函数，但并没有给出如何找到这样一个网络，以及是否是最优的．此外，当应用到机器学习时，真实的映射函数并不知道，一般是通过经验风险最小化和正则化来进行参数学习．

神经网络如何应用到机器学习中：

在机器学习中，输入样本的特征对分类器的影响很大。以监督学习为例，好的特征可以极大提高分类器的性能。因此，要取得好的分类效果，需要将样本的原始特征向量$\boldsymbol{x}$转换到更有效的特征向量$\boldsymbol{\phi(x)}$，而多层前馈神经网络作为一种“万能”函数，刚好可以用来做特征转换或者逼近一个复杂的分布。

给定一个训练样本$(\boldsymbol{x},\boldsymbol{y})$，先用多层前馈神经网络把$\boldsymbol{x}$转换到$\boldsymbol{\phi(x)}$，然后再将$\boldsymbol{\phi(x)}$输入到分类器$\boldsymbol{g(\cdot)}$，即：

$$\hat{y} = g(\varphi(\mathbf{x}), \theta)$$

其中$\boldsymbol{\theta}$为分类器的参数。

特别地，如果分类器$\boldsymbol{g}(\cdot)$为Logistic 回归分类器或Softmax 回归分类器，那么$\boldsymbol{g}(\cdot)$也可以看成是网络的最后一层（输出层），即神经网络直接输出不同类别的条件概率$\boldsymbol{p(y|x)}$.

对于二分类问题$y \in \{0, 1\}$，并采用Logistic 回归， Logistic回归．那么Logistic回归分类器可以看成神经网络的最后一层．也就是说，网络的最后一层只用一个神经元，并且其激活函数为Logistic函数．网络的输出可以直接作为类别$y = 1$ 的条件概率：

$$p(y = 1|\boldsymbol{x}) = \boldsymbol{a}^{(L)}$$

其中$\boldsymbol{a}^{(L)} \in \mathbb{R}^{C}$ 为第L 层神经元的活性值．

对于多分类问题$y \in \{1, \cdots, C\}$，如果使用Softmax 回归分类器， Softmax回归相当于网络最后一层设置C 个神经元，其激活函数为Softmax函数，网络最后一层（第L 层）的输出可以作为每个类的条件概率，即

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{z}^{(L)})$$

其中$\boldsymbol{z}^{(L)} \in \mathbb{R}^{C}$为第$L$层神经元的净输入；$\hat{\boldsymbol{y}} \in \mathbb{R}^{C}$为第$L$层神经元的活性值（输出值），每一维分别表示不同类别标签的预测条件概率．

参数学习：

设对于训练样本$(\boldsymbol{x},\boldsymbol{y})$，损失函数为：$\mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})$。那么对于给定的训练集$D = \{(\boldsymbol{x}^{(n)},\boldsymbol{y}^{(n)})\}_{n=1}^{N}$，将每个样本$\boldsymbol{x}^{(n)}$输入给前馈神经网络，得到网络输出为 $\hat{\boldsymbol{y}}^{(n)}$，其在数据集D上的结构化风险函数为:

$$\mathcal{R}(W,\mathbf{b}) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\mathbf{y}^{(n)},\hat{\mathbf{y}}^{(n)}) + \frac{1}{2}\lambda\|W\|_F^2$$

其中$\boldsymbol{W}$表示网络中所有的权重矩阵，$\boldsymbol{b}$表示网络中所有的偏置；$\|\boldsymbol{W}\|_F^2$为正则化项，用来防止过拟合，一般采用Frobenius范数：

$$\|\boldsymbol{W}\|_F^2 = \sum_{l=1}^{L}\sum_{i=1}^{M_l}\sum_{j=1}^{M_{l-1}}(w_{ij}^{(l)})^2.$$

梯度下降法进行学习，第l层参数更新方式为：

$$\begin{aligned}
\boldsymbol{W}^{(l)} &\leftarrow \boldsymbol{W}^{(l)} - \alpha\frac{\partial\mathcal{R}(\boldsymbol{W},\boldsymbol{b})}{\partial\boldsymbol{W}^{(l)}} \\
&= \boldsymbol{W}^{(l)} - \alpha\left(\frac{1}{N}\sum_{n=1}^{N}(\frac{\partial\mathcal{L}(\boldsymbol{y}^{(n)},\hat{\boldsymbol{y}}^{(n)})}{\partial\boldsymbol{W}^{(l)}}) + \lambda\boldsymbol{W}^{(l)}\right) \\
\boldsymbol{b}^{(l)} &\leftarrow \boldsymbol{b}^{(l)} - \alpha\frac{\partial\mathcal{R}(\boldsymbol{W},\boldsymbol{b})}{\partial\boldsymbol{b}^{(l)}} \\
&= \boldsymbol{b}^{(l)} - \alpha\left(\frac{1}{N}\sum_{n=1}^{N}\frac{\partial\mathcal{L}(\boldsymbol{y}^{(n)},\hat{\boldsymbol{y}}^{(n)})}{\partial\boldsymbol{b}^{(l)}}\right),
\end{aligned}$$

2.4 反向传播算法

梯度下降法需要计算损失函数对参数的偏导数，如果通过链式法则逐一对每个参数进行求偏导比较低效．在神经网络的训练中经常使用**反向传播算法**来高效地计算梯度．

对于l层，我们需要计算两个偏微分：$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{W}^{(l)}}$和$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{b}^{(l)}}$．
根据链式法则：

$$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial w_{ij}^{(l)}} = \frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}},$$

$$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{b}^{(l)}} \frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}}.$$

其中$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}}$称为误差项，记为$\delta^{(l)}$，表示第$l$层神经元对最终损失的影响。

计算偏导数$\frac{\partial z^l}{\partial w_{ij}^{(l)}}$：因为$\mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$

$$\begin{aligned}
\frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} &= \left[\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_{m^{(l)}}^{(l)}}{\partial w_{ij}^{(l)}}\right] \\
&= \left[0, \cdots, \frac{\partial(\mathbf{w}_{i:}^{(l)}\mathbf{a}^{(l-1)} + b_i^{(l)})}{\partial w_{ij}^{(l)}}, \cdots, 0\right] \\
&= \left[0, \cdots, a_j^{(l-1)}, \cdots, 0\right] \\
&\triangleq \mathbb{I}_i(a_j^{(l-1)}) \quad \in \mathbb{R}^{m^{(l)}},
\end{aligned}$$

列向量关于标量的偏导数是行向量

计算偏导数$\frac{\partial z^l}{\partial \boldsymbol{b}^{(l)}}$：

$$\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} = \mathbf{I}_{m^{(l)}} \quad \in \mathbb{R}^{m^{(l)} \times m^{(l)}}$$

$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_q}{\partial x_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial y_1}{\partial x_p} & \cdots & \frac{\partial y_q}{\partial x_p} \end{bmatrix} \in \mathbb{R}^{p \times q}$$

向量关于向量的偏导数

计算误差项$\delta^{(l)} = \frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}}$:

根据:

$$\mathbf{a}^{(l)} = f_l(\mathbf{z}^{(l)})$$

$$\mathbf{z}^{(l+1)} = W^{(l+1)}\mathbf{a}^{(l)} + \mathbf{b}^{(l+1)}$$

$$\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} = \frac{\partial f_l(\mathbf{z}^{(l)})}{\partial \mathbf{z}^{(l)}} = \mathrm{diag}(f_l'(\mathbf{z}^{(l)}))$$

$$\frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} = (W^{(l+1)})^{\mathrm{T}}$$

又由链式法则，得:

$$\begin{aligned}\delta^{(l)} &\triangleq \frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \\ &= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{z}^{(l+1)}} \\ &= \mathrm{diag}(f_l'(\mathbf{z}^{(l)})) \cdot (W^{(l+1)})^{\mathrm{T}} \cdot \delta^{(l+1)} \\ &= f_l'(\mathbf{z}^{(l)}) \odot \big((W^{(l+1)})^{\mathrm{T}}\delta^{(l+1)}\big),\end{aligned}$$

$$
\begin{aligned}
\delta^{(l)} &\triangleq \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \\
&= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l+1)}} \\
&= \operatorname{diag}(f_l'(\mathbf{z}^{(l)})) \cdot (W^{(l+1)})^{\mathrm{T}} \cdot \delta^{(l+1)} \\
&= f_l'(\mathbf{z}^{(l)}) \odot \left((W^{(l+1)})^{\mathrm{T}} \delta^{(l+1)}\right),
\end{aligned}
$$

可以看出，第l层的误差项可以通过第$l+1$层的误差项计算得到，这就是误差的反向传播（BackPropagation，BP）．

反向传播算法的含义是：第l层的一个神经元的误差项（或敏感性）是所有与该神经元相连的第$l+1$层的神经元的误差项的权重和．然后，再乘上该神经元激活函数的梯度．

最终得到：

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} = \mathbb{I}_i(a_j^{(l-1)})\delta^{(l)} = \delta_i^{(l)} a_j^{(l-1)}.$$

进一步写为：

$$\left[\frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{W}^{(l)}}\right]_{ij} = \left[\delta^{(l)}(\boldsymbol{a}^{(l-1)})^{\mathrm{T}}\right]_{ij}.$$

于是对权重矩阵的偏导为：

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial W^{(l)}} = \delta^{(l)}(\mathbf{a}^{(l-1)})^{\mathrm{T}}.$$

而对偏置的偏导为：

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}.$$

小结：

使用误差反向传播算法的前馈神经网络训练过程可以分为以下三步：

（1） 前馈计算每一层的净输入$\boldsymbol{z}^{(l)}$和激活值$\boldsymbol{a}^{(l)}$，直到最后一层；

（2） 反向传播计算每一层的误差项$\delta^{(l)}$；

（3） 计算每一层参数的偏导数，并更新参数.

使用反向传播算法的随机梯度下降训练过程：

输入: 训练集$\mathcal{D}=\{(\boldsymbol{x}^{(n)},y^{(n)})\}_{n=1}^{N}$，验证集$\mathcal{V}$，学习率$\alpha$，正则化系数$\lambda$，网络层数$L$，神经元数量$M_l, 1\leq l\leq L$.

随机初始化$\boldsymbol{W},\boldsymbol{b}$；

repeat

  对训练集$\mathcal{D}$中的样本随机重排序；

  for $n=1\cdots N$ **do**

    从训练集$\mathcal{D}$中选取样本$(\boldsymbol{x}^{(n)},y^{(n)})$；

    前馈计算每一层的净输入$\boldsymbol{z}^{(l)}$和激活值$\boldsymbol{a}^{(l)}$，直到最后一层；

    反向传播计算每一层的误差$\delta^{(l)}$； // 公式(4.63)

    // 计算每一层参数的导数

    $\forall l,\quad \frac{\partial \mathcal{L}(y^{(n)},\hat{y}^{(n)})}{\partial \boldsymbol{W}^{(l)}}=\delta^{(l)}(\boldsymbol{a}^{(l-1)})^{\mathrm{T}}$； // 公式(4.68)

    $\forall l,\quad \frac{\partial \mathcal{L}(y^{(n)},\hat{y}^{(n)})}{\partial \boldsymbol{b}^{(l)}}=\delta^{(l)}$； // 公式(4.69)

    // 更新参数

    $\boldsymbol{W}^{(l)}\leftarrow\boldsymbol{W}^{(l)}-\alpha(\delta^{(l)}(\boldsymbol{a}^{(l-1)})^{\mathrm{T}}+\lambda\boldsymbol{W}^{(l)})$；

    $\boldsymbol{b}^{(l)}\leftarrow\boldsymbol{b}^{(l)}-\alpha\delta^{(l)}$；

  end

until 神经网络模型在验证集$\mathcal{V}$上的错误率不再下降；

输出: $\boldsymbol{W},\boldsymbol{b}$

03. 一个完整的神经网络模型训练

THANK