前馈神经网络

从机器学习的角度来看，神经网络一般可以看作是一个非线性模型，其基本组成单位为具有非线性激活函数的神经元，通过大量神经元之间的连接，使得神经网络成为一种高度非线性的模型。神经元之间的连接权重就是需要学习的参数，可以通过梯度下降方法来进行学习。

神经元

假设一个神经元接受 d 个输入 $x_1, x_2, \cdots, x_d$，令向量 $\mathbf{x} = [x_1; x_2; \cdots; x_d]$ 来表示这组输入，并用净输入（Net Input）$z \in \mathbb{R}$ 表示一个神经元所获得的输入信号 $\mathbf{x}$ 的加权和，

$$z = \sum_{i=1}^{d} w_i x_i + b \tag{4.1}$$

$$= \mathbf{w}^{\mathrm{T}}\mathbf{x} + b, \tag{4.2}$$

其中 $\mathbf{w} = [w_1; w_2; \cdots; w_d] \in \mathbb{R}^d$ 是 d 维的权重向量，$b \in \mathbb{R}$ 是偏置。

净输入 z 在经过一个非线性函数 $f(\cdot)$ 后，得到神经元的活性值（Activation）a，

$$a = f(z), \tag{4.3}$$

其中非线性函数 $f(\cdot)$ 称为激活函数（Activation Function）。

激活函数：为了增强网络的表示能力和学习能力，激活函数需要具备以下几点性质：

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。

激活函数

Sigmoid 型函数是指一类 S 型曲线函数，为两端饱和函数。常用的 Sigmoid 型函数有 Logistic 函数和 Tanh 函数。

Logistic 函数：1）其输出直接可以看作是概率分布，使得神经网络可以更好地和统计学习模型进行结合。2）其可以看作是一个软性门（Soft Gate），用来控制其它神经元输出信息的数量。

Tanh 函数：放大并平移的 Logistic 函数

Tanh 函数的输出是零中心化的（Zero-Centered），而 Logistic 函数的输出恒大于 0。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。都是在中间（0 附近）近似线性，两端饱和。

修正线性单元（Rectified Linear Unit， ReLU）

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

优点:采用 ReLU 的神经元只需要进行加、乘和比较的操作，计算上更加高效。Sigmoid 型激活函数会导致一个非稀疏的神经网络，而 ReLU 却具有很好的稀疏性，大约 50%的神经元会处于激活状态。在优化方面，相比于 Sigmoid 型函数的两端饱和，ReLU 函数为左饱和函数，且在 x>0 时导数为 1，在一定程度上缓解了神经网络的梯度消失问题，加速梯度下降的收敛速度。

缺点:ReLU 函数的输出是非零中心化的，给后一层的神经网络引入偏置偏移，会影响梯度下降的效率。此外，ReLU 神经元在训练时比较容易“死亡”。在训练时，如果参数在一次不恰当的更新后，第一个隐藏层中的某个 ReLU 神经元在所有的训练数据上都不能被激活，那么这个神经元自身参数的梯度永远都会是 0，在以后的训练过程中永远不能被激活。这种现象称为死亡 ReLU 问题（Dying ReLU Problem），并且也有可能会发生在其它隐藏层。

带泄露的 ReLU（Leaky ReLU）

在输入 x<0 时，保持一个很小的梯度 λ。这样当神经元非激活时也能有一个非零的梯度可以更新参数，避免永远不能被激活.

$$LeakyReLU(x)=max(\gamma x,x)=\max(0,x)+\gamma\min(0,x)=\begin{cases} x, x>0 \\ \gamma x, x\leq 0 \end{cases}$$

带参数的 ReLU（Parametric ReLU， PReLU）

引入一个可学习的参数，不同神经元可以有不同的参数。对于第 i 个神经元，其 PReLU 的定义为

$$PReLU_i(x)=\max(0,x)+\gamma_i\min(0,x)=\begin{cases} x, x>0 \\ \gamma_i x, x\leq 0 \end{cases}$$

指数线性单元（Exponential Linear Unit， ELU）

一个近似的零中心化的非线性函数,超参数决定 x≤0 时的饱和曲线，并调整输出均值在 0 附近。

$$ELU(x)=\max(0,x)+\min(0,\gamma(e^x-1))=\begin{cases} x, x>0 \\ \gamma(e^x-1), x\leq 0 \end{cases}$$

Softplus 函数

可以看作是 rectifier 函数的平滑版本,其导数刚好是 Logistic 函数。Softplus 函数虽然也有具有单侧抑制、宽兴奋边界的特性，却没有稀疏激活性。

$$Softplus(x)=\log(1+\exp(x))$$

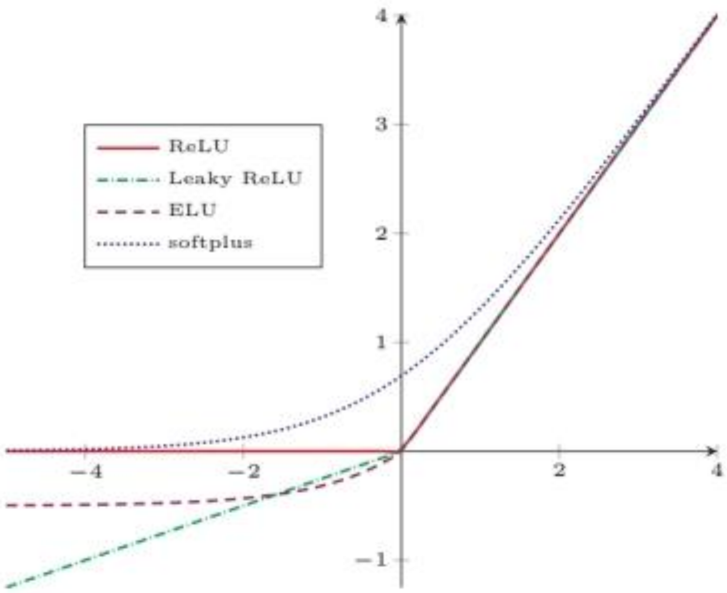

图 4.4 ReLU、Leaky ReLU、ELU 以及 Softplus 函数

Swish 函数

$$swish(x) = x\sigma(\beta x)$$

一种自门控（Self-Gated）激活函数，β 为可学习的参数或一个固定超参数。σ(·)∈(0,1)可以看作是一种软性的门控机制。当 σ(βx)接近于 1 时，门处于“开”状态，激活函数的输出近似于 x 本身；当 σ(βx)接近于 0 时，门的状态为“关”，激活函数的输出近似于 0。

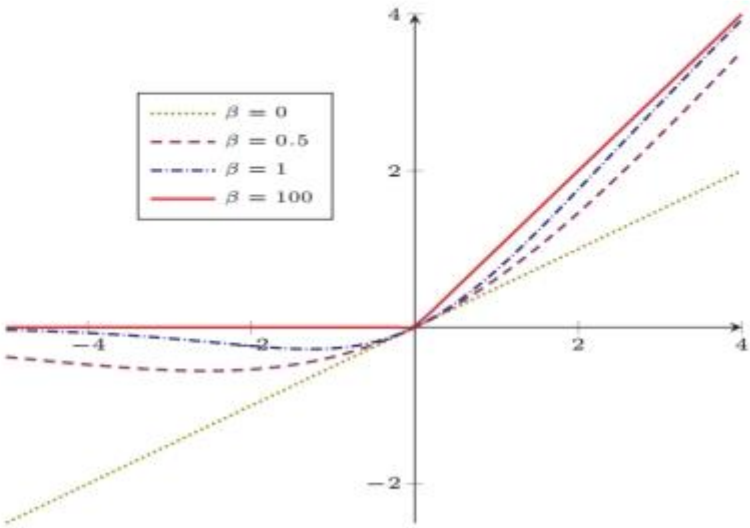

图 4.5 Swish 函数

当 β=0 时，Swish 函数变成线性函数 x/2。当 β=1 时，Swish 函数在 x>0 时近似线性，在 x<0 时近似饱和，同时具有一定的非单调性。当 β→+∞时，σ(βx)趋向于离散的 0-1 函数，Swish 函数近似为 ReLU 函数。因此，Swish 函数可以看作是线性函数和 ReLU 函数之间的非线性插值函数，其程度由参数 β 控制。

Maxout 单元

一种分段线性函数。Sigmoid 型采用 maxout 单元的神经网函数、ReLU 等激活函数的输入是神经元的净输入 z，是一个标量。而 maxout 络也就做 maxou 网络。单元的输入是上一层神经元的全部原始输入，是一个向量。

输入 $\mathbf{x}$，可以得到 K 个净输入 $z_k, 1 \leq k \leq K$。

$$z_k = \mathbf{w}_k^{\mathrm{T}} \mathbf{x} + b_k, \tag{4.27}$$

其中 $\mathbf{w}_k = [w_{k,1}, \cdots, w_{k,d}]^{\mathrm{T}}$ 为第 k 个权重向量。

Maxout 单元的非线性函数定义为

$$\operatorname{maxout}(\mathbf{x}) = \max_{k \in [1,K]} (z_k). \tag{4.28}$$

Maxout 单元不单是净输入到输出之间的非线性映射，而是整体学习输入到输出之间的非线性映射关系。Maxout 激活函数可以看作任意凸函数的分段线性近似，并且在有限的点上是不可微的。

网络结构

通过一定的连接方式或信息传递方式进行协作的神经元。

前馈网络

各个神经元按接受信息的先后分为不同的组。每一层中的神经元接受前一层神经元的输出，并输出到下一层神经元。整个网络中的信息是朝一个方向传播，没有反向的信息传播。

全连接前馈网络和卷积神经网络等。

可以看作一个函数，通过简单非线性函数的多次复合，实现输入空间到输出空间的复杂映射。这种网络结构简单，易于实现。

反馈网络

反馈网络中神经元不但可以接收其它神经元的信号，也可以接收自己的反馈信号。和前馈网络相比，反馈网络中的神经元具有记忆功能，在不同的时刻具有不同的状态。反馈神经网络中的信息传播可以是单向或双向传递。

循环神经网络，Hopfield 网络、玻尔兹曼机等。

为了增强记忆网络的记忆容量，可以引入外部记忆单元和读写机制，用来保存一些网络的中间状态，称为记忆增强网络（Memory-Augmented Neural Network），比如神经图灵机和记忆网络等。

图网络

定义在图结构数据上。节点之间的连接可以是有向的，也可以是无向的。每个节点可以收到来自相邻节点或自身的信息。

图网络是前馈网络和记忆网络的泛化，比如图卷积网络（Graph

Convolutional Network, GCN)、消息传递网络 (Message Passing Neural Network, MPNN) 等

前馈神经网络 (Feedforward Neural Network, FNN)

由多层的 Logistic 回归模型（连续的非线性函数）组成。

- L：表示神经网络的层数；
- $m^{(l)}$：表示第 l 层神经元的个数；
- $f_l(\cdot)$：表示 l 层神经元的激活函数；
- $W^{(l)} \in \mathbb{R}^{m^{(l)} \times m^{l-1}}$：表示 $l-1$ 层到第 l 层的权重矩阵；
- $\mathbf{b}^{(l)} \in \mathbb{R}^{m^l}$：表示 $l-1$ 层到第 l 层的偏置；
- $\mathbf{z}^{(l)} \in \mathbb{R}^{m^l}$：表示 l 层神经元的净输入（净活性值）；
- $\mathbf{a}^{(l)} \in \mathbb{R}^{m^l}$：表示 l 层神经元的输出（活性值）。

前馈神经网络通过下面公式进行信息传播，

$$\mathbf{z}^{(l)} = W^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \tag{4.29}$$

$$\mathbf{a}^{(l)} = f_l(\mathbf{z}^{(l)}). \tag{4.30}$$

通用近似定理

定理 4.1 – 通用近似定理（Universal Approximation Theorem）[Cybenko, 1989, Hornik et al., 1989]：令 $\varphi(\cdot)$ 是一个非常数、有界、单调递增的连续函数，$\mathcal{I}_d$ 是一个 d 维的单位超立方体 $[0,1]^d$，$C(\mathcal{I}_d)$ 是定义在 $\mathcal{I}_d$ 上的连续函数集合。对于任何一个函数 $f \in C(\mathcal{I}_d)$，存在一个整数 m，和一组实数 $v_i, b_i \in \mathbb{R}$ 以及实数向量 $\mathbf{w}_i \in \mathbb{R}^d$，$i = 1, \cdots, m$，以至于我们可以定义函数

$$F(\mathbf{x}) = \sum_{i=1}^{m} v_i \varphi(\mathbf{w}_i^{\mathrm{T}} \mathbf{x} + b_i), \tag{4.34}$$

作为函数 f 的近似实现，即

$$|F(\mathbf{x}) - f(\mathbf{x})| < \epsilon, \forall \mathbf{x} \in \mathcal{I}_d. \tag{4.35}$$

其中 $\epsilon > 0$ 是一个很小的正数。

对于具有线性输出层和至少一个使用“挤压”性质的激活函数的隐藏层组成的前馈神经网络，只要其隐藏层神经元的数量足够，它可以以任意的精度来近似任何从一个定义在实数空间 $\mathbb{R}^d$ 中的有界闭集函数。

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。通过经验风险最小化和正

则化来进行参数学习，因为神经网络的强大能力，反而容易在训练集上过拟合。

应用到机器学习

特征抽取：将样本的原始特征向量 x 转换到更有效的特征向量 φ(x)。

多层前馈神经网络也可以看成是一种特征转换方法，其输出 φ(x)作为分类器的输入进行分类。

可以使用 Logistic 回归分类器或 softmax 回归分类器。

参数学习

如果采用交叉熵损失函数，对于样本 $(\mathbf{x}, y)$，其损失函数为

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\mathbf{y}^{\mathrm{T}} \log \hat{\mathbf{y}}, \tag{4.39}$$

其中 $\mathbf{y} \in \{0,1\}^C$ 为标签 y 对应的 one-hot 向量表示。

给定训练集为 $\mathcal{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^N$，将每个样本 $\mathbf{x}^{(n)}$ 输入给前馈神经网络，得到网络输出为 $\hat{\mathbf{y}}^{(n)}$，其在数据集 $\mathcal{D}$ 上的结构化风险函数为：

$$\mathcal{R}(W, \mathbf{b}) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)}) + \frac{1}{2} \lambda \|W\|_F^2, \tag{4.40}$$

$$= \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)}) + \frac{1}{2} \lambda \|W\|_F^2, \tag{4.41}$$

其中 W 和 $\mathbf{b}$ 分别表示网络中所有的权重矩阵和偏置向量；$\|W\|_F^2$ 是正则化项，用来防止过拟合；λ 是为正数的超参数。λ 越大，W 越接近于 0。这里的 $\|W\|_F^2$ 一般使用 Frobenius 范数：

$$\|W\|_F^2 = \sum_{l=1}^{L} \sum_{i=1}^{m^{(l)}} \sum_{j=1}^{m^{(l-1)}} (W_{ij}^{(l)})^2. \tag{4.42}$$

有了学习准则和训练样本，网络参数可以通过梯度下降法来进行学习。在梯度下降方法的每次迭代中，第 l 层的参数 $W^{(l)}$ 和 $\mathbf{b}^{(l)}$ 参数更新方式为

$$W^{(l)} \leftarrow W^{(l)} - \alpha \frac{\partial \mathcal{R}(W, \mathbf{b})}{\partial W^{(l)}}, \tag{4.43}$$

$$= W^{(l)} - \alpha \left(\frac{1}{N} \sum_{n=1}^{N} \left(\frac{\partial \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)})}{\partial W^{(l)}} \right) + \lambda W^{(l)} \right), \tag{4.44}$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \alpha \frac{\partial \mathcal{R}(W, \mathbf{b})}{\partial \mathbf{b}^{(l)}}, \tag{4.45}$$

$$= \mathbf{b}^{(l)} - \alpha \left(\frac{1}{N} \sum_{n=1}^{N} \frac{\partial \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)})}{\partial \mathbf{b}^{(l)}} \right), \tag{4.46}$$

其中 α 为学习率。

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效率比较低。在神经网络的训练中经常使用反向传播算法来计算高效地梯度。

反向传播算法

采用随机梯度下降进行神经网络参数学习，要进行参数学习就需要计算损失函数关于每个参数的导数。

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial W_{ij}^{(l)}} = \left(\frac{\partial \mathbf{z}^{(l)}}{\partial W_{ij}^{(l)}}\right)^{\mathrm{T}} \frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}, \tag{4.47}$$

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \left(\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}}\right)^{\mathrm{T}} \frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}. \tag{4.48}$$

公式 (4.47) 和 (4.48) 中的第二项是都为目标函数关于第 l 层的神经元 $\mathbf{z}^{(l)}$ 的偏导数，称为**误差项**，因此可以共用。我们只需要计算三个偏导数，分别为 $\frac{\partial \mathbf{z}^{(l)}}{\partial W_{ij}^{(l)}}$，$\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}}$ 和 $\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$。

（1）计算偏导数 $\frac{\partial \mathbf{z}^{(l)}}{\partial W_{ij}^{(l)}}$ 因为 $\mathbf{z}^{(l)}$ 和 $W_{ij}^{(l)}$ 的函数关系为 $\mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$，因此偏导数

$$\frac{\partial \mathbf{z}^{(l)}}{\partial W_{ij}^{(l)}} = \frac{\partial (W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})}{\partial W_{ij}^{(l)}} \tag{4.49}$$

$$= \begin{bmatrix} \frac{\partial (W_{1:}^{(l)}\mathbf{a}^{(l-1)}+\mathbf{b}^{(l)})}{\partial W_{ij}^{(l)}} \\ \vdots \\ \frac{\partial (W_{i:}^{(l)}\mathbf{a}^{(l-1)}+\mathbf{b}^{(l)})}{\partial W_{ij}^{(l)}} \\ \vdots \\ \frac{\partial (W_{m^{(l)}:}^{(l)}\mathbf{a}^{(l-1)}+\mathbf{b}^{(l)})}{\partial W_{ij}^{(l)}} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ \boxed{a_j^{(l-1)}} \\ \vdots \\ 0 \end{bmatrix} \leftarrow \text{第 } i \text{ 行} \tag{4.50}$$

$$\triangleq \mathbb{I}_i(a_j^{(l-1)}), \tag{4.51}$$

其中 $W_{i:}^{(l)}$ 为权重矩阵 $W^{(l)}$ 的第 i 行。

（2）计算偏导数 $\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}}$ 因为 $\mathbf{z}^{(l)}$ 和 $\mathbf{b}^{(l)}$ 的函数关系为 $\mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$，因此偏导数

$$\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} = \mathbf{I}_{m^{(l)}}, \tag{4.52}$$

（3）计算误差项 $\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$ 我们用 $\delta^{(l)}$ 来定义第 l 层神经元的误差项，

$$\delta^{(l)}=\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}\in \mathbb{R}^{m^{(l)}}. \tag{4.53}$$

误差项 $\delta^{(l)}$ 来表示第 l 层神经元对最终损失的影响，也反映了最终损失对第 l 层神经元的敏感程度。误差项也间接反映了不同神经元对网络能力的贡献程度，从而比较好地解决了“贡献度分配问题”。

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

$$\frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}}=(W^{(l+1)})^{\mathrm{T}}. \tag{4.54}$$

根据 $\mathbf{a}^{(l)}=f_l(\mathbf{z}^{(l)})$，其中 $f_l(\cdot)$ 为按位计算的函数，因此有

$$\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}}=\frac{\partial f_l(\mathbf{z}^{(l)})}{\partial \mathbf{z}^{(l)}} \tag{4.55}$$

$$=\mathrm{diag}(f_l'(\mathbf{z}^{(l)})). \tag{4.56}$$

因此，根据链式法则，第 l 层的误差项为

$$\delta^{(l)}\triangleq\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \tag{4.57}$$

$$=\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}}\cdot\frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}}\cdot\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{z}^{(l+1)}} \tag{4.58}$$

$$=\mathrm{diag}(f_l'(\mathbf{z}^{(l)}))\cdot(W^{(l+1)})^{\mathrm{T}}\cdot\delta^{(l+1)} \tag{4.59}$$

$$=f_l'(\mathbf{z}^{(l)})\odot\left((W^{(l+1)})^{\mathrm{T}}\delta^{(l+1)}\right), \tag{4.60}$$

其中 $\odot$ 是向量的点积运算符，表示每个元素相乘。

从公式(4.60)可以看出，第 1 层的误差项可以通过第 1+1 层的误差项计算得到，这就是误差的反向传播。反向传播算法的含义是：第 1 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 1+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。

在计算出上面三个偏导数之后，公式(4.47)可以写为

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial W_{ij}^{(l)}}=\mathbb{I}_i(a_j^{(l-1)})^{\mathrm{T}}\delta^{(l)}=\delta_i^{(l)}a_j^{(l-1)}. \tag{4.61}$$

进一步，$\mathcal{L}(\mathbf{y},\hat{\mathbf{y}})$ 关于第 l 层权重 $W^{(l)}$ 的梯度为

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial W^{(l)}}=\delta^{(l)}(\mathbf{a}^{(l-1)})^{\mathrm{T}}. \tag{4.62}$$

同理可得，$\mathcal{L}(\mathbf{y},\hat{\mathbf{y}})$ 关于第 l 层偏置 $\mathbf{b}^{(l)}$ 的梯度为

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}}=\delta^{(l)}. \tag{4.63}$$

1. 前馈计算每一层的净输入 $\mathbf{z}^{(l)}$ 和激活值 $\mathbf{a}^{(l)}$，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。

算法4.1给出使用随机梯度下降的误差反向传播算法的具体训练过程。

算法 4.1: 基于随机梯度下降的反向传播算法

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

随机初始化 $W,\mathbf{b}$;

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 $(\mathbf{x}^{(n)},y^{(n)})$;

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

    反向传播计算每一层的误差 $\delta^{(l)}$; // 公式 (4.60)

    // 计算每一层参数的导数

    $\forall l,\quad \frac{\partial\mathcal{L}(\mathbf{y}^{(n)},\hat{\mathbf{y}}^{(n)})}{\partial W^{(l)}}=\delta^{(l)}(\mathbf{a}^{(l-1)})^{\mathrm{T}}$; // 公式 (4.62)

    $\forall l,\quad \frac{\partial\mathcal{L}(\mathbf{y}^{(n)},\hat{\mathbf{y}}^{(n)})}{\partial \mathbf{b}^{(l)}}=\delta^{(l)}$; // 公式 (4.63)

    // 更新参数

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

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

  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: $W,\mathbf{b}$

自动梯度计算

几乎所有的主流深度学习框架都包含了自动梯度计算的功能。

数值微分（Numerical Differentiation）

用数值方法来计算函数的导数： $f'(x)=\lim_{\Delta x\to 0}\frac{f(x+\Delta x)-f(x)}{\Delta x}$.

如果 Δx 过小，会引起数值计算问题，比如舍入误差；如果 Δx 过大，会增加截断误差，使得导数计算不准确。因此，数值微分的实用性比较差。

$$f'(x)=\lim_{\Delta x\to 0}\frac{f(x+\Delta x)-f(x-\Delta x)}{2\Delta x}.$$

数值微分的另外一个问题是计算复杂度。假设参数数量为 n，则每个参数都需要单独施加扰动，并计算梯度。假设每次正向传播的计算复杂度为 O(n)，则计算数值微分的总体时间复杂度为 $O(n^2)$。

符号微分（Symbolic Differentiation）

对输入的表达式，通过迭代或递归使用一些事先定义的规则进行转换。当转换结果不能再继续使用变换规则时，便停止计算。

自动微分（Automatic Differentiation， AD）

前向模式和反向模式。

优化问题

非凸优化问题

神经网络的优化问题是一个非凸优化问题。

梯度消失问题

在神经网络中误差反向传播的迭代公式为

$$\delta^{(l)} = f_l'(\mathbf{z}^{(l)}) \odot \left(W^{(l+1)}\right)^{\mathrm{T}} \delta^{(l+1)}, \tag{4.86}$$

误差从输出层反向传播时，在每一层都要乘以该层的激活函数的导数。当我们使用 Sigmoid 型函数：Logistic 函数 $\sigma(x)$ 或 Tanh 函数时，其导数为

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big) \in [0, 0.25] \tag{4.87}$$

$$\tanh'(x) = 1 - \big(\tanh(x)\big)^2 \in [0, 1]. \tag{4.88}$$

由于 Sigmoid 型函数的饱和性，饱和区的导数更是接近于 0。这样，误差经过每一层传递都会不断衰减。当网络层数很深时，梯度就会不停的衰减，甚至消失，使得整个网络很难训练。这就是所谓的梯度消失问题（Vanishing Gradient Problem），也叫梯度弥散问题。

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

表 4.2 常见激活函数及其导数