3.5 神经网络

从大量数据中找特征机器比人更擅长

3.3 节我们介绍了**对数几率回归**，将其作为基本单元（称为**神经元**），我们就能够构建出一个**全连接多层神经网络**（Neural Network，后面简称为 NN）。如图 1 所示，如同搭积木一样，我们把多个对数几率回归基本单元按照从左到右的顺序一层层叠起来，层与层之间所有神经元均连接在一起，最右边用一个多类对数几率回归（Softmax）完成最终的多类分类，这就得到了一个很常见的全连接多层 NN。由于层与层之间所有神经元均连接在一起，所以称为“**全连接**”（fully connected）。

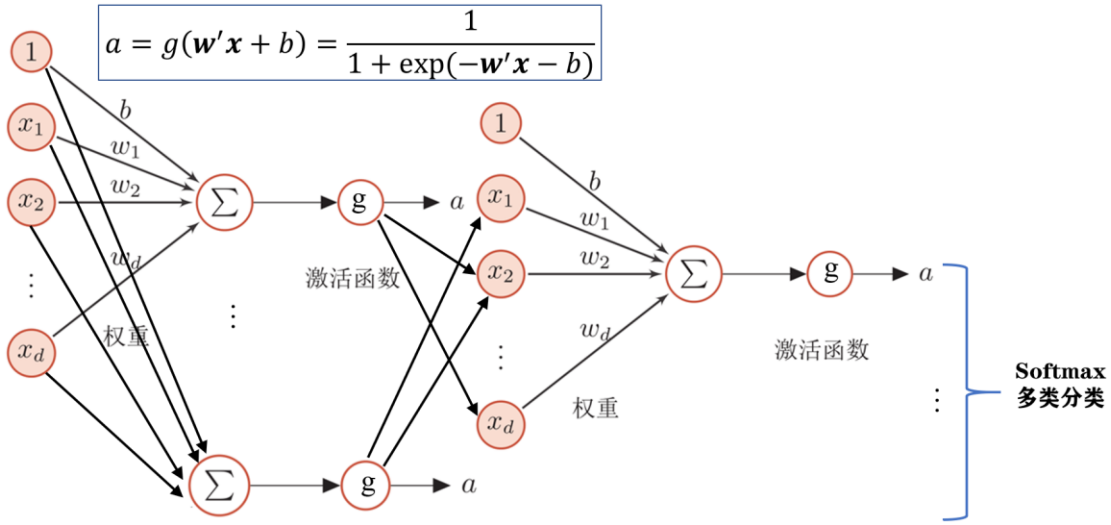

图 1 全连接多层神经网络

图 1 中，我们按照 NN 的习惯，将函数$g()$称为**激活函数**（activation functions），相应的将函数$g()$的输出值a称为**激活值**（activation value）。图 1 的最左边是整个 NN 的第 1 层——输入层（input layer），用输入向量$(1, x_1, x_2, \ldots, x_d)^T$表示。最右边是整个 NN 的最后一层——输出层（output layer），给出多类分类的类别

值。其余中间的层次均接受前一层的输出作为输入，并将本层的输出作为后一层的输入。图 2 给出了一个整体上更清楚的三层全连接 NN 网络结构图，其中忽略了神经元的细节。如图 2 所示，中间层我们一般也称为“**隐藏层**”（hidden layer）。本节我们将采用类似这样的一个 NN 来解决 Mnist 手写数字识别问题。

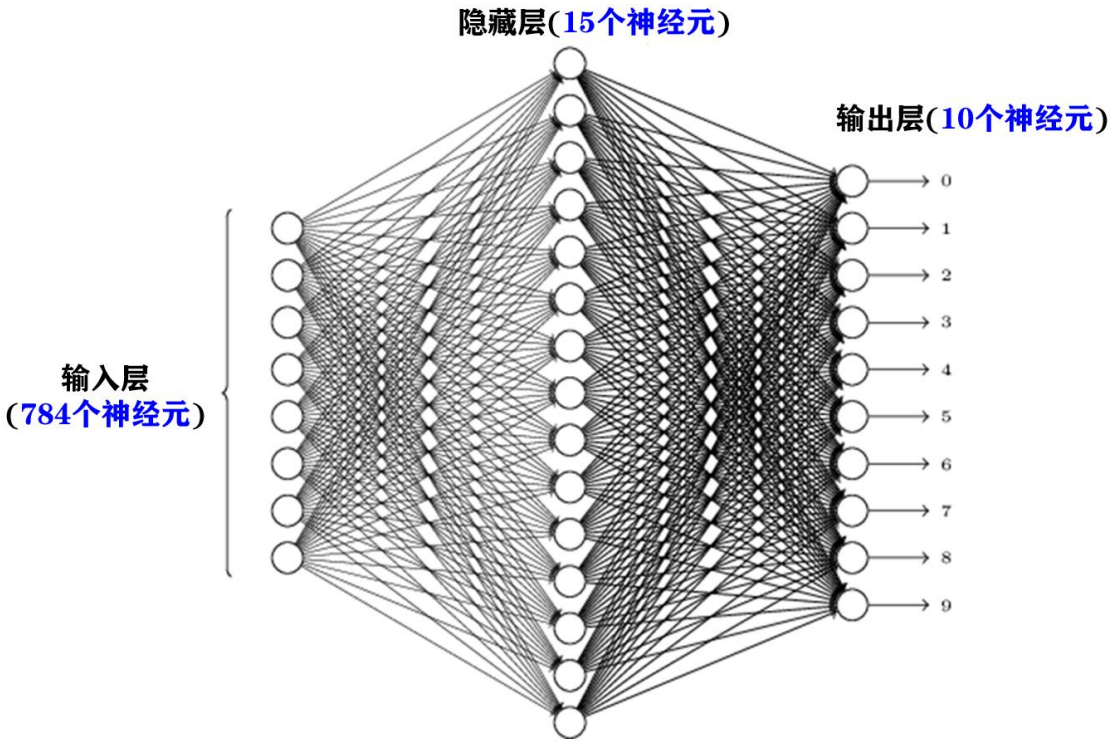

图 2 一个三层全连接多层神经网络

3.5.1 万能逼近定理

早在 1989 年，Hornik 等人[1]就证明了一个定理，称为 NN 的万能逼近定理。这个定理表明：单隐层 NN 与任意连续 S 型函数能够以**任意精度**逼近任何目标连续函数，只要不对网络的结点数加以限制。这个定理的理论意义不言而喻，相当于图灵机在计算机科学中的地位——一个通用的能解决任何可计算问题的理论计算机。这个定理断言，只需要满足很弱的条件，我们可以用类似图 2 的单隐层 NN 完美解决任何机器学习问题。

然而，万能逼近定理对于实际中如何设计 NN 起不到有效的指导作用，因为

实际中我们不可能只用单个隐层而对其结点数不加以限制——计算复杂度过高。或许对于本节将涉及的 Mnist 手写数字识别问题，我们确实可以如定理所要求的，采用一个单隐层 NN，那么这也存在一个基本问题——隐层应该采用多少个神经元才能达到多高的精度？万能逼近定理也解答不了我们这个基本问题。

3.5.2 NN 的学习方法——梯度下降

对于手写数字识别问题，我们的类别标签取值为{0,1,⋯,9}。为了设计 NN 的方便，此处我们将第i个训练样本的类别标签$\boldsymbol{y_i}$定义为一个 10 维的单位向量$(\dots,1,\dots)^T$，如果其第k维为 1，我们就说其类别为k，或者说为数字k。这样定义的类别标签一般也称为 **one-hot 向量**。

类似的，我们要求网络的输出层也输出这样一个 **one-hot 向量**。这样，我们就可以方便的定义一个**二次损失（也称为代价）函数**，用以衡量输出层的输出与类别标签是否一致。对于每个训练样本，如果输出$\boldsymbol{a}$与类别标签$\boldsymbol{y_i}$一致，则损失为 0；否则损失为 2。见公式（1），其中 N 为总的训练样本数。注意，此

$$L(\boldsymbol{w},\boldsymbol{b})=\frac{1}{2N}\sum_{i=1}^{N}\|\boldsymbol{y_i}-\boldsymbol{a}\|^2 \qquad (1)$$

处我们对求和的值除以 N，以起到归一化的效果（对比 3.3 节的相应公式）。至于 N 前面的 2 倍则纯粹是为了后面求梯度得到一个更简洁的数学形式（3.4 节我们也采用了类似的小技巧）。

要使损失$L(\boldsymbol{w},\boldsymbol{b})$最小，我们可以采用 3.3 节介绍的梯度下降方法。首先计算$L(\boldsymbol{w},\boldsymbol{b})$的梯度$\nabla_{\boldsymbol{w}}L(\boldsymbol{w},\boldsymbol{b})$和$\nabla_{\boldsymbol{b}}L(\boldsymbol{w},\boldsymbol{b})$。然后用公式$\boldsymbol{w}=\boldsymbol{w}-\alpha\nabla_{\boldsymbol{w}}L(\boldsymbol{w},\boldsymbol{b})$和$\boldsymbol{b}=\boldsymbol{b}-\alpha\nabla_{\boldsymbol{b}}L(\boldsymbol{w},\boldsymbol{b})$更新参数，直到$\boldsymbol{w}$和$\boldsymbol{b}$收敛或者达到最大迭代次数。不同点在于，公式（1）中的$\boldsymbol{a}$实际上本身也是一个关于$\boldsymbol{w}$和$\boldsymbol{b}$的函数，因此我们必然要涉及到**多层复合函数的链式求导规则**。NN 中将这个求导的过程形象的称为误差**反向传播算**

法（BP 算法）。

3.5.3 NN 的学习方法——BP 算法

为了讨论的方便，先约定一些数学符号。用$l = 1, \ldots, L$表示一个 NN 的**层**，其中第1层是输入层，第L层是最后一层--输出层。用$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$表示第$l$层第$j$个神经元的**权重输入**。把第$l$层的所有神经元的权重输入记为向量$\boldsymbol{z}^l$，则有$\boldsymbol{z}^l = \boldsymbol{w}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l$。用$\boldsymbol{a}^l = \sigma(\boldsymbol{z}^l) = \sigma(\boldsymbol{w}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l)$表示第$l$层神经元的输出激活值，激活函数此处采用对率函数$\sigma()$。图 3 给出了一个简单 NN 的图示。

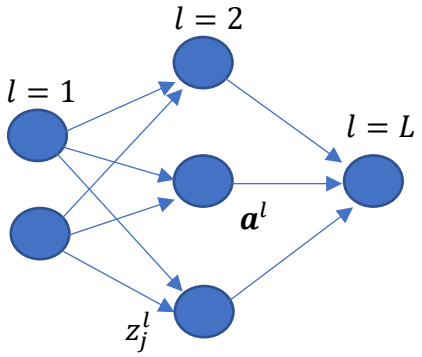

图 3 一个简单 NN 的图示

由此，公式（1）的二次损失函数可重写为$C = \frac{1}{2N}\sum_{i=1}^{N} \|\boldsymbol{y_i} - \boldsymbol{a^L}\|^2$。为了避免与第$L$层的$L$搞混，我们将损失函数重新记为$C$。我们定义$\delta_j^l = \frac{\partial C}{\partial z_j^l}$表示第$l$层第$j$个神经元的**误差**。反向传播的过程就是为了计算这个误差，然后基于误差就能得到梯度$\nabla_{\boldsymbol{w}} C$和$\nabla_{\boldsymbol{b}} C$。

我们先来看第一个反向传播方程。由上面的定义有$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \sum_k \frac{\partial C}{\partial a_k^L}\frac{\partial a_k^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$，这就得到了方程（BP1）——输出层的误差。第二步推导就是复合函数的链式求导。第三步是由于$\frac{\partial a_k^L}{\partial z_j^L}$当$k \neq j$时为 0，因为同层的不同神经元之间无连接。

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'\left(z_j^L\right) \qquad \text{(BP1)}$$

继续推导第二个方程。$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1}\frac{\partial z_k^{l+1}}{\partial z_j^l}$，又$z_k^{l+1} =$

$\sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$，$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l)$，代入则得到

$$\delta_j^l = \sum_k \delta_k^{l+1} w_{kj}^{l+1} \sigma'(z_j^l) \quad \text{(BP2)}$$

方程（BP2）是第l层相对于第$l+1$层的误差，实际上就是反向传播第$l+1$层的误差，从而得到第l层的误差。由方程（BP2）和（BP1）即可从最后一层L开始得到任一层l的误差：$\boldsymbol{\delta}^L \to \boldsymbol{\delta}^{L-1} \to \cdots \to \boldsymbol{\delta}^1$。类似，很容易推导出方程（BP3）和（BP4）：

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad \text{(BP3)}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \quad \text{(BP4)}$$

这就是我们最终需要的梯度$\nabla_{\boldsymbol{w}} C$和$\nabla_{\boldsymbol{b}} C$。

练习：

1、试计算图 2 所示的 NN 有多少个参数。

2、试证明方程（BP3）和（BP4）。

3、试分析全连接 NN 的局限性。

参考文献：

【1】Multilayer feedforward networks are universal approximators, by Kurt Hornik, Maxwell Stinchcombe, and Halbert White (1989)

【2】Neural Networks and Deep Learning, by Michael Nielson, (2015)

[http://neuralnetworksanddeeplearning.com/index.html]