学号	21820236	姓名	于烨泳		
评分项					
充分性（20%）	准确性（20%）	逻辑性（20%）	深刻性（20%）	规范性（20%）	总分

深入剖析反向传播算法

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

从机器学习的角度来看，神经网络一般可以看作一个非线性模型，线性神经元和非线性激活函数极大增强网络的表示能力和学习能力的，神经元之间的连接权重就是需要学习的参数，可以在机器学习的框架下通过梯度下降方法来进行学习[1]。前馈神经网络（FNN）作为一种极其重要的神经网络结构，其他网络结构如记忆网络，图网络的前馈思想是以 FNN 为基础的。根据通用近似定理[2]，前馈神经网络具有很强的拟合能力，常见的连续非线性函数都可以用前馈神经网络来近似[1]。

在前馈神经网络中，每一层的神经元可以接收前一层神经元的信号，并产生信号输出到下一层。第 0 层称为输入层，最后一层称为输出层，中间层称为隐藏层。整个网络中无反馈，信号从输入层向输出层单向传播，如图 1 所示。

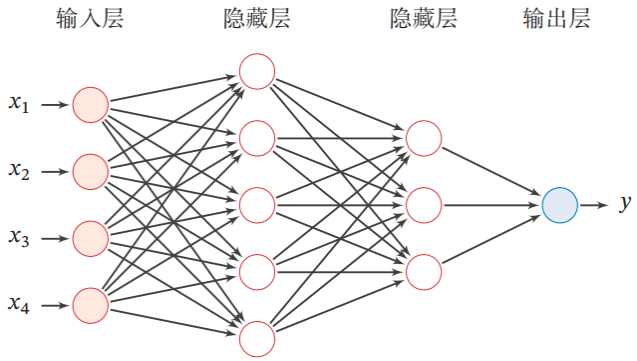

图 1 多层前馈神经网络[1]

上图中，令$\boldsymbol{a}^{(0)} = \mathbf{x}$（x 代表(x1,x2,x3,x4)向量），前馈神经网络通过公式(1)进行信息前向传播：

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

其中 $\boldsymbol{z}^{(l)}$ 代表 l 层的净活性值（未被激活），f_l 为激活函数，$\boldsymbol{a}^{(l)}$ 则是经过 f_l 激活 $\boldsymbol{z}^{(l)}$ 的活性值，通过这种方式前馈神经网络逐层传递信息，信息传递到输出层就得到了网络的预测值 $\hat{\boldsymbol{y}}$，通过特定的损失函数$\mathcal{L}\left(\boldsymbol{y}^{(n)}, \hat{\boldsymbol{y}}^{(n)}\right)$可以计算出此次前向传播的损失（未考虑正则化项）：

$$\mathcal{R}(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}\left(\boldsymbol{y}^{(n)}, \hat{\boldsymbol{y}}^{(n)}\right) \tag{2}$$

在传统的机器学习任务中，目标是最小化迭代训练误差，网络参数$\boldsymbol{W}$和$\boldsymbol{b}$就可以通过梯度下降方法公式(3)，(4)（也有机器学习任务是梯度上升，将公式(3)，(4)中的“-”替换成“+”）来进行更新学习：

$$W^{(l)} \leftarrow W^{(l)} - \alpha \frac{\partial \mathcal{R}(W,b)}{\partial W^{(l)}} = W^{(l)} - \alpha \left(\frac{1}{N} \sum_{n=1}^{N} \left(\frac{\partial \mathcal{L}\left(y^{(n)}, \hat{y}^{(n)}\right)}{\partial W^{(l)}} \right) + \lambda W^{(l)} \right) \tag{3}$$

$$b^{(l)} \leftarrow b^{(l)} - \alpha \frac{\partial \mathcal{R}(W,b)}{\partial b^{(l)}} = b^{(l)} - \alpha \left(\frac{1}{N} \sum_{n=1}^{N} \frac{\partial \mathcal{L}\left(y^{(n)}, \hat{y}^{(n)}\right)}{\partial b^{(l)}} \right) \tag{4}$$

其中$\frac{\partial \mathcal{R}(\boldsymbol{W},\boldsymbol{b})}{\partial \boldsymbol{W}^{(l)}}$ 是梯度上升最快的方向，乘以-1 则可以得到梯度下降最快的方向，α是学习率，两者相乘即可得到参数$W^{(l)}$和参数$b^{(l)}$更新的幅度，同时我们需要注意到$-\frac{\partial \mathcal{R}(W,b)}{\partial W^{(l)}}$是当前梯度下降最快的方向，并不一定是最优的方向，这就是导致模型陷入局部最优的问题根源。但是此梯度下降法需要计算损失函数对参数的偏导数，如果通过链式法逐一对每一层每一个参数进行求偏导，更新的效率非常低，所以在神经网络的参数训练过程中常使用反向传播算法（Backpropagation, BP）来更新参数。

2. 反向传播算法（Backpropagation, BP）

梯度下降法是一种通用的优化算法，中心思想是沿着目标函数梯度的方向更新参数值以希望达到目标函数最小（或最大），反向传播法是梯度下降法在深度网络上的具体实现方式。由于深度学习网络按层深入，层层嵌套的特点，且现有网络结构大多为多输入少输出（在此场景下前向传播的方式不够高效），所以对深度网络目标函数计算梯度的时候，使用用反向传播的方式由深到浅倒着计算以及更新参数将会更高效。

BP 算法将$\frac{\partial \mathcal{L}(\boldsymbol{W},\boldsymbol{b})}{\partial \boldsymbol{W}^{(l)}}$和$\frac{\partial \mathcal{L}(\boldsymbol{W},\boldsymbol{b})}{\partial \boldsymbol{b}^{(l)}}$梯度通过链式法则公式(5)，(6)分解：

$$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial w_{ij}^{(l)}} = \frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}} \tag{5}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{b}^{(l)}} \frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}} \tag{6}$$

其中$\boldsymbol{z}^{(l)}$是 l 层净活性值，根据公式 $\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}$ 计算得到，所以$\frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}}$可以根据上一层的活性值$\mathrm{a}^{(l-1)}$计算得出，而$\frac{\partial \mathrm{z}^{(l)}}{\partial \mathrm{b}^{(l)}}$因为系数为 I，所以求偏导结果就是$M_l \times M_l$单位矩阵，M_l是第 l 层神经元的个数，具体计算见公式(7)，(8)。

因为 l 层的第 i 个神经元的净活性值$z_i^{(l)}$是由 $l-1$ 层所有神经元的活性值$\boldsymbol{a}^{(l-1)}$经过仿射变换$\boldsymbol{w}_{i:}^{(l)}\boldsymbol{a}^{(l-1)} + b_i^{(l)}$得到的，$\boldsymbol{w}_{i:}^{(l)}$为权重矩阵$\mathrm{W}^{(l)}$第 i 行，即第 l 层中第 i 个神经元中的所有权重$\boldsymbol{w}$，其中的每一个权重$w_{ij}^{(l)}$只与 $l-1$ 层第 j 个神经元的激活值$a_j^{(l-1)}$相乘，其余的偏导都为 0，所以具

体到每一层的参数$w_{ij}^{(l)}$的偏导上，$\frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}}$得到的结果是$\mathbb{I}_i\left(a_j^{(l-1)}\right)$，表示第 i 个元素为$a_j^{(l-1)}$，其余都为 0 的行向量。

$$\begin{aligned}\frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}} &= [\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}},\cdots,\frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}},\cdots,\frac{\partial z_{M_l}^{(l)}}{\partial w_{ij}^{(l)}}] \\ &= \left[0,\cdots,\frac{\partial\left(\boldsymbol{w}_{i:}^{(l)}\boldsymbol{a}^{(l-1)}+b_i^{(l)}\right)}{\partial w_{ij}^{(l)}},\cdots,0\right] \\ &= \left[0,\cdots,a_j^{(l-1)},\cdots,0\right] \\ &\triangleq \mathbb{I}_i\left(a_j^{(l-1)}\right) \in \mathbb{R}^{1\times M_l},\end{aligned} \tag{7}$$

$$\frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{I}_{M_l} \in \mathbb{R}^{M_l\times M_l} \tag{8}$$

对比公式(5)，(6)，可以发现所需计算的仅剩$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}}$，代表第 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[1]，[1]中定义其为误差项$\delta^{(l)}$：

$$\delta^{(l)} \triangleq \frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}} \in \mathbb{R}^{M_l} \tag{9}$$

对比直接计算$\frac{\partial \mathcal{L}(\mathrm{y}^{(\mathrm{n})},\hat{y}^{(\mathrm{n})})}{\partial \mathrm{W}^{(\mathrm{l})}}$，误差项$\delta^{(l)}$可以找到递推关系，即可以通过下一层的$\delta^{(l+1)}$来直接计算$\delta^{(l)}$，见公式(10)：

$$\begin{aligned}\delta^{(l)} &\triangleq \frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}} \\ &= \frac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{z}^{(l)}}\cdot\frac{\partial \boldsymbol{z}^{(l+1)}}{\partial \boldsymbol{a}^{(l)}}\cdot\left(\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l+1)}}\right) \\ &= \mathrm{diag}\left(f_l'\left(\boldsymbol{z}^{(l)}\right)\right)\cdot\left(\boldsymbol{W}^{(l+1)}\right)^{\mathsf{T}}\cdot\delta^{(l+1)} \\ &= f_l'\left(\boldsymbol{z}^{(l)}\right)\odot\left(\left(\boldsymbol{W}^{(l+1)}\right)^{\mathsf{T}}\delta^{(l+1)}\right) \in \mathbb{R}^{M_l},\end{aligned} \tag{10}$$

其中⊙是向量的 Hadamard 积运算符，表示每个元素相乘，diag 表示$f_l'\left(\boldsymbol{z}^{(l)}\right)$的对角线元素。反向传播的含义就是第 l 层的一个神经元的误差项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再乘上该神经元激活函数的梯度[1]。简而言之就是$\delta^{(l)}$可以根据下一层的$\delta^{(l+1)}$计算得出，综合之前计算得到的$\frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}}$和$\frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{b}^{(l)}}$就可以得到参数更新的方向$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial w_{ij}^{(l)}}$和$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{b}^{(l)}}$，与学习率 α 相乘即可更新参数，无需逐层通过链式法则计算参数更新的方向，大大提升了参数的更新效率。

反向传播算法的思想我感觉与前馈神经网络有“异曲同工”之妙，不过 FNN 传播的是信息，BP 传播的是误差，两者传播的信息都可以基于本层的信息直接传递，而不用依赖之前层（反向传播是之后层）的信息；且 FNN 前馈的$\boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)}$和激活函数$f_l\left(\boldsymbol{z}^{(l)}\right)$与 BP 算法的$\left(\boldsymbol{W}^{(l+1)}\right)^{\mathsf{T}}\delta^{(l+1)}$ 和激活函数的导数$f_l'\left(\boldsymbol{z}^{(l)}\right)$都是镜像的逆操作；不过 BP 误差的传递还需要前馈时保留的每一层的激活值$\boldsymbol{a}^{(l)}$和净激活值$\boldsymbol{z}^{(l)}$。

3. 反向传播与动态规划（Dynamic Programming，DP）

作为计算机专业的学生，相信对动态规划思想一定不会陌生，在《Introduction to Algorithms》[3]中动态规划做以下定义：动态规划通过组合子问题的解来求解原问题（在这里，“programming”指的是一种表格法，并非编写计算机程序）。动态规划应用于子问题重叠的情况，即不同的子问题具有公共子子问题（子问题的求解是递归进行的，将其划分为更小的子子问题），动态规划算法对每个子问题只求解一次，将其保存在一个表格中，从而无需每次求解一个子子问题时都需重新计算，避免了这种不必要的计算工作。

对比 FNN 中传统的梯度下降的方法计算$\frac{\partial \mathcal{R}(W,b)}{\partial W^{(l)}}$和$\frac{\partial \mathcal{R}(W,b)}{\partial b^{(l)}}$，这其中包含了很多重复的子问题计算工作，导致了参数更新的低效。而 BP 则应用了动态规划的核心思想——通过组合子问题的解（后面层神经元的误差项）来求解每一个参数更新的梯度。在 BP 中最核心的点就是其巧妙利用求导的链式法则找到了误差反向传播的递推反向，即误差传播的状态方程，见公式(11)：

$$\boldsymbol{\delta}^{(l)} = f_l'\left(\boldsymbol{z}^{(l)}\right) \odot \left(\left(\boldsymbol{W}^{(l+1)}\right)^{\mathrm{T}} \delta^{(l+1)}\right) \tag{11}$$

具体的例子如图 2，在 BP 中可以根据$\delta^{3,1}$、$\boldsymbol{W}^{(3)}$和$\boldsymbol{z}^{(2)}$来计算出$\delta^{2,1}$，$\delta^{2,2}$，$\delta^{2,3}$，保存在动态规划表格中，以此类推来计算第一层的$\boldsymbol{\delta}^{(1)}$，并且根据链式法则来计算最终所需要的$\frac{\partial \mathcal{R}(W,b)}{\partial W^{(l)}}$和$\frac{\partial \mathcal{R}(W,b)}{\partial b^{(l)}}$用于梯度下降更新参数。

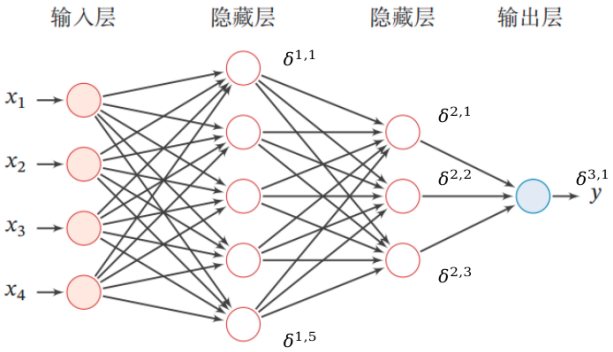

图 2 反向传播算法中的动态规划思想

但是 BP 与 DP 有一个根本上的区别就是 DP 是用于求解最优化问题，DP 没有陷入局部最优的问题，而 BP 算法受限于梯度的局部性，所以是一种局部搜索的优化方法，但它要解决的问题为求解复杂非线性函数的全局极值，因此，算法很有可能陷入局部极值，即局部最优解。

4. 反向传播与强化学习（Reinforcement Learning，RL）

在我深入理解反向传播算法之后，发现反向传播算法不仅可以传播误差，还可以传递奖励，来实现梯度上升算法。之前的强化学习 Q-Learning 和 Sarsa 采用动态规划来传递奖励，改善每个状态上智能体的动作选择。当 DQN 出现，深度学习引入强化学习之后，科研者很自然的想到使用反向传播算法来传播奖励，然后利用梯度上升来改进智能体的动作选择。在我理解中，反向传播就像从输出端角度出发的信号前馈，前馈的信号是损失或奖励，以修正参数来优化整个网络。

参考文献

[1] 邱锡鹏，神经网络与深度学习，机械工业出版社，https://nndl.github.io/, 2020.

[2] Hornik K, Stinchcombe M, White H, 1989. Multilayer feedforward networks are universal approximators[J]. Neural networks, 2(5):359-366.

[3] Cormen, Thomas H. (EDT), Introduction to Algorithms, Mit Pr, 2005