# 1 BP 神经网络

## 1.1 神经网络的构造与前向传播

神经网络是由单个或多个神经元组成。下面是单个神经元的构造。

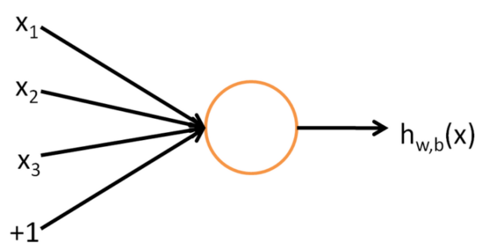



**图 1：** 神经元

该神经元的输入由三个数据 $x_1, x_2, x_3$ 以及偏置项 (bias)+1 组成，通过神经元后输出的表达式为

$$h_{W,b}(x) = f(W^T x + b) = f(\sum_{i=1}^{3} W_i x_i + b) \tag{1}$$

其中 $f$ 为激活函数。激活函数是为了将线性项 $W^T x$ 变换为非线性。在 BP 中，较常用的激活函数为 sigmoid 函数，其表达式如下

$$f(z) = \frac{1}{1 + \exp(-z)} \tag{2}$$

另外，令 $b = w_0$，则可重新定义 $W = (w_0, w_1, w_2, w_3)^T$，$x = (1, x_1, x_2, x_3)$，于是可将上式写为

$$h_{W,b}(x) = f(W^T x) \tag{3}$$

下面讨论神经网络。多个神经元可以组成一个层，多个层互相连接可以组成神经网络。其中，接受数据输入的层为输入层，数据计算后的数据的输出层，中间的层则称为隐含层。为下图是含有两个隐含层的神经网络。

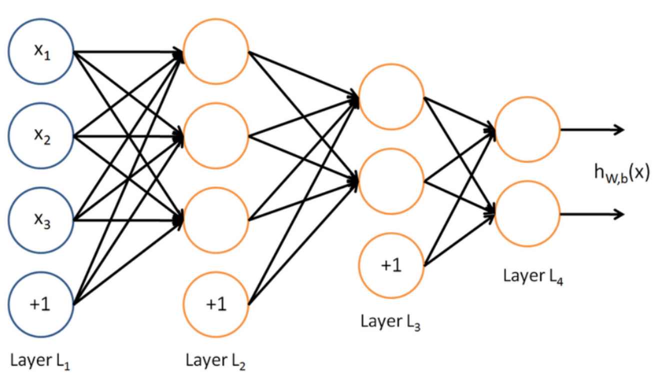



**图 2：** 含有两个隐含层的神经网络

如图，最左边的为输入层，即图中的 Layer L1，最右边的为输出层，即图中的 Layer L4，中间的所有层，即图中的 Layer L2，Layer L3 为隐含层。

我们用 $n_l$ 来表示网络的层数，记第 $i$ 层为 $L_i$，于是输入层为 $L_1$，输出层为 $L_{n_l}$。由于神经网络可以有任意多的隐层以及隐藏神经元，则我们记 $W_{ij}^{(l)}$ 为第 $l$ 层第 $j$ 单元以及第 $l+1$ 层第 $i$ 单元之间的连接权重，$b_i^{(l)}$ 为第 $L+1$ 层第 $i$ 单元的偏执。我们用 $a_i^{(l)}$ 表示第 $l$ 层第 $i$ 单元的激活值（输出值），则有

$$a_i^{(l+1)} = f(\sum_{j=1}^{S_l} W_{ij}^{(l)} a_j^{(l)} + b_i^{(l)}) \tag{4}$$

其中当 $l=1$ 时，$a^{(l)} = x$，$x$ 为输入向量 $(x_1, x_2, \cdots, x_{S_l})$，$S_l$ 指第 $l$ 层的神经元个数，我们用 $z_i^{(l+1)}$ 表示第 $l+1$ 层第 $i$ 单元输入加权和（包括偏置），即

$$z_i^{(l+1)} = \sum_{j=1}^{S_t} W_{ij}^{(l)} a_j^{(l)} + b_i^{(l)} \tag{5}$$

则有

$$a_i^{(l+1)} = f(z_i^{(l+1)}) \tag{6}$$

$$h_{W,b}(x) = a^{(n_l)} = f(z^{(n_l)}) \tag{7}$$

上述过程称为神经网络的前向传播。

## 1.2 神经网络的反向传播

根据上面的前向传播，我们设神经网络的各层表示为 $L_1, L_2, \cdots, L_{n_l}$，其中，$L_{n_l}$ 为输出层，对于输出层，假设输出层输出为 $t = a^{(n_l)}$，$y$ 为标签，则若为回归问题，则代价函数使用 MSE，即

$$J(W,b;x,y) = \frac{1}{2}||t-y||^2 \tag{8}$$

接下来计算输出层的残差

$$\begin{aligned}
\delta_i^{(n_l)} &= \frac{\partial}{\partial z_i^{(n_l)}} J(W,b;x,y) \\
&= \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2}||y - h_{W,b}(x)||^2 \\
&= \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \sum_{j=1}^{S} {}_{n_l}(y_i - a_j^{(n_l)})^2 \\
&= \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \sum_{j=1}^{S} {}_{n_l}(y_i - f(z_i^{(n_l)}))^2 \\
&= -(y_i - f(z_i^{(n_l)})) \cdot f'(z_i^{(n_l)}) \\
&= -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})
\end{aligned} \tag{9}$$

下面考虑残差的递推算法，以输出层前一层为例。由前向传播我们可以推导出

$$z_i^{(l+1)} = \sum_{j=1}^{S_l} W_{ij}^{(l)} f(z_i^{(l)}) + b_i^{(l)} \tag{10}$$

则有

$$z_i^{(n_i)} = \sum_{j=1}^{S_l} W_{ij}^{(n_l-1)} f(z_i^{(n_l-1)}) + b_i^{(n_l-1)} \tag{11}$$

于是有

$$\frac{\partial z_i^{(n_l)}}{\partial z_i^{(n_l-1)}} = \sum_{j=1}^{S_l} W_{ij}^{(n_l-1)} f'(z_i^{(n_l-1)}) \tag{12}$$

则可以得到输出层前一层的残差

$$\begin{aligned} \delta_i^{(n_l-1)} &= \frac{\partial}{\partial z_i^{(n_l-1)}} J(W,b;x,y) \\ &= \frac{\partial J(W,b;x,y)}{\partial z_i^{(n_l)}} \cdot \frac{\partial z_i^{(n_l)}}{\partial z_i^{(n_l-1)}} \\ &= \sum_{j=1}^{S_l} \delta_j^{(n_l)} W_{ij}^{(n_l-1)} f'(z_i^{(n_l-1)}) \end{aligned} \tag{13}$$

将 $n_l - 1$ 与 $n_l$ 的关系替换为 $l$ 与 $l+1$ 的关系，则可得到

$$\delta_i^{(l)} = \frac{\partial}{\partial z_i^{(l)}} J(W,b;x,y) = \left( \sum_{j=1}^{S_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}) \tag{14}$$

若取函数 $f$ 为 sigmoid 函数，则有

$$f'(z_i^{(l)}) = f(z_i^{(l)}) \circ (1 - f(z_i^{(l)})) = a_i^{(l)} \circ (1 - a_i^{(l)}) \tag{15}$$

其中 $\circ$ 代表点乘。于是可得到 $\sigma_j^{(l+1)}$ 到 $\sigma_j^{(l)}$ 的递推式：

$$\delta_i^{(l)} = \left( \sum_{j=1}^{S_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) (a_i^{(l)} \circ (1 - a_i^{(l)})) \tag{16}$$

反向传播，一般采用梯度下降法对每一层的权重进行调整，即

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) \tag{17}$$

其中，$\alpha$ 是学习率。因而需要求权重 $W_{ij}^{(l)}$ 对于代价函数的偏导，此时可使用当前层的残差来进行计算，即

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = \frac{\partial J(W,b;x,y)}{\partial z_i^{(l+1)}} \frac{z_i^{(l+1)}}{W_{ij}^{(l)}} \tag{18}$$

又有

$$\frac{z_i^{(l+1)}}{W_{ij}^{(l)}} = \frac{\left( \sum_{j=1}^{S_l} W_{ij}^{(l)} f(z_i^{(l)}) \right)}{W_{ij}^{(l)}} = f(z_i^{(l)}) = a_i^{(l)} \tag{19}$$

于是可得

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)} \tag{20}$$

综上，可以总结 BP 神经网络算法

**BP 神经网络算法**

输入：训练输入，训练输出，学习率

**while** 未达到收敛条件

  输入训练输入，训练输出，学习率

1. 初始化神经网络的权重与偏置
2. 对输入进行前向传播，得到除输入层外每一层（$L_2, \cdots, L_{n_l}$）的激活值 $a^{(2)}, \cdots, a^{(n_l)}$
3. 计算各层残差：

(1) 对输出层（第 $n_l$ 层）

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \cdot (a^{(l)} \circ (1 - a^{(l)})) \tag{21}$$

(2) 对于 $l = n_l - 1, \cdots, 2$ 各层，可递推得出残差值

$$\delta^{(l)} = ((W^{(l)})^T \delta^{(l+1)}) \cdot (a^{(l)}) \tag{22}$$

(3) 计算损失函数对每一层权重的偏导数值

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T \tag{23}$$

(4) 更新参数

$$W^{(l)} = W^{(l)} - \alpha \nabla_{W^{(l)}} J(W, b; x, y) \tag{24}$$

end

若为多分类问题，先对 $y$ 进行 one-hot 处理得到 $p$ 维向量 $(y_1, y_2, \cdots, y_p)$（假设 $y$ 有 $p$ 种取值），并将输出层的激活函数选为 softmax，即

$$a_i^{(n_l)} = f_s(z_i^{(n_l)}) = \frac{e^{z_i^{(n_l)}}}{\sum_j e^{z_j^{(n_l)}}} \tag{25}$$

并且代价函数使用交叉熵损失函数

$$J(W, b; x, y) = -\sum_i y_i \log a_i^{(n_l)} \tag{26}$$

则输出层残差为

$$\begin{aligned}
\delta_i^{(n_l)} &= \frac{\partial J}{\partial z_i^{(n_l)}} \\
&= \sum_i \frac{\partial J}{a_i^{(n_l)}} \cdot \frac{\partial a_i^{(n_l)}}{\partial z_i^{(n_l)}} \\
&= \sum_i \frac{\partial - \sum_i y_i \log a_i^{(n_l)}}{a_i^{(n_l)}} \cdot \frac{\partial a_i^{(n_l)}}{\partial z_i^{(n_l)}} \\
&= -\sum_i \frac{y_i}{a_i^{(n_l)}} \frac{\partial a_i^{(n_l)}}{\partial z_j^{(n_l)}}
\end{aligned} \tag{27}$$

当 $i=j$ 时，记 $e^{z_j^{(n_l)}}=e^A$，$\sum_{k\neq j} e^{z_k^{(n_l)}}=e^B$，显然有 $e^A+e^B=\sum_i e^{z_i^{(n_l)}}$，于是

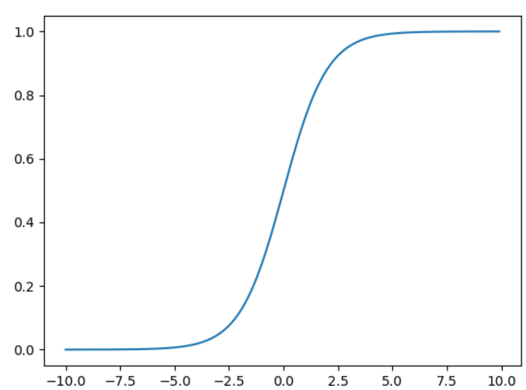
$$\begin{aligned}\frac{\partial a_i^{(n_l)}}{\partial z_j^{(n_l)}}=\frac{\partial a_j^{(n_l)}}{\partial z_j^{(n_l)}}\\&=\frac{\partial \frac{e^A}{e^A+e^B}}{\partial A}\\&=\frac{e^A(e^B+e^A)-e^{2A}}{(e^A+e^B)^2}\\&=\frac{e^Ae^B}{(e^A+e^B)^2}\\&=\frac{e^A}{e^A+e^B}\frac{e^B}{e^A+e^B}\\&=\frac{e^A}{e^A+e^B}(1-\frac{e^A}{e^A+e^B})\\&=a_j^{(n_l)}(1-a_j^{(n_l)})\end{aligned} \tag{28}$$

### 1.3 激活函数

**sigmoid** sigmoid 函数表达式如下

$$f(x)=\frac{1}{1+e^{-x}} \tag{29}$$

其图像如下图所示




sigmoid 激活函数考虑将输入值映射到 $(0,1)$ 的区间中，该函数在定义域内连续，且导数大于 0。它也有较为简单的求导结果

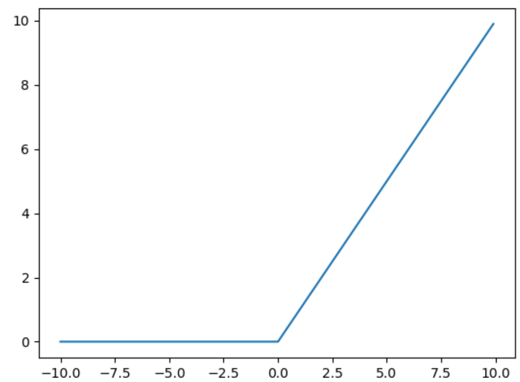
$$f'(x)=f(x)(1-f(x)) \tag{30}$$

但是在神经网络中，特别是对于层数较多的网络，通常不采用 sigmoid 作为激活函数，主要是因为其容易产生梯度消失的情况。当输入非常大或非常小的时候，其梯度趋近于 0，反向传播的过程中直接导致梯度无法传播，无法有效地调整权重。虽然做标准化可以让数据近似服从正态分布，但梯度消失仍有可能产生，在学习过程中可能会产生输入较大或较小的情况。或许这个问题可以用 batch-normalization 来缓解，但明显采取一种更佳的激活函数是较为可取的做法。

**ReLU** ReLU 函数表达式如下

$$f(x) = \max\{0, x\} \tag{31}$$

图像如下




其决定它有非常简单的求导结果

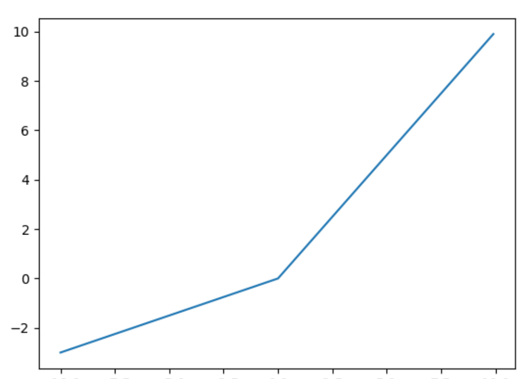
$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases} \tag{32}$$

RuLU 收敛能比 sigmoid 快的多，一方面其计算快，比起 sigmoid 函数的导数需要指数运算，RuLU 只需要做大小的比较。另一方面，其梯度经过多个层传播之后，多数能够保持原汁原味，比起 sigmoid 会梯度消失要好得多。然而，RuLU 也有弱点，当 $x < 0$ 时 $f(x)$ 为 0，梯度为 0，这直接导致该神经元失活。因而在训练过程中，要注意取较小的学习率。

**Leaky ReLU** Leaky ReLU 是针对 RuLU 的弱点而改进的，其考虑用一个比较小的数去替代 $x < 0$ 时的 $f(x) = 0$，即

$$f'(x) = \begin{cases} x, & x > 0 \\ ax, & x < 0 \end{cases} \tag{33}$$

图像如下



其求导结果为

$$f'(x) = \begin{cases} 1, & x > 0 \\ a, & x < 0 \end{cases} \tag{34}$$

这个方法可以使 $x < 0$ 处避免失活，但是额外引入了超参数 $a$。

**PReLU** PReLU 是针对 Leaky ReLU 的进一步优化，其考虑在反向传播过程中，也对 $a$ 进行学习，从而避免引入超参数 $a$。一些实验 [1] 证明这种优化能取到好的学习效果。

### 1.4 梯度下降法

梯度下降法的选取能影响收敛速度与质量，它也是模型构成的一部分。在应用中一般有如下的梯度下降法可供选择

**批量梯度下降法** 批量梯度下降法（Batch Gradient Descent ）考虑在计算了所有样本之后再对参数进行更新，即

$$W^{(l)} = \sum_{i=1}^{m} W^{(l)} - \alpha \nabla_{W^{(l)}} J(W, b; x^{(i)}, y^{(i)}) \tag{35}$$

由于通常训练的样本非常大，若在算法所有样本之后再进行参数更新，会让更新的速度减慢。另外，模型实现一般会采用矩阵运算，BGD 占的内存会非常多，从而影响计算速度。



## 2 RCNN

### 2.1 神经网络

### 2.2 卷积神经网络

#### 2.2.1 AlexNet

#### 2.2.2 VGGNet

### 2.3 Inception

### 2.4 ResNet

### 2.5 动态学习率

### 2.6 Batch Normalization

### 2.7 迁移学习

### 2.8 交并比

交并比 (Intersection-over-Union,Iou) 是用于衡量候选框（candidate bound）与标记框（ground truth bound）相似程度的指标，其是两者的交叠率。假设候选框区域为 $C$，标记框区域为 $G$，则定义交并比

$IoU$ 为

$$IoU = \frac{C \cap G}{C \cup G} \tag{36}$$

若 $IoU$ 越接近与 1，说明重叠程度越大，效果越好。

## 2.9 非极大值抑制

非极大值抑制算法 (Non-maximum suppression,NMS) 本质为搜索局部的极大值，并抑制附近非极大值的元素。

若给定一个 $n$ 个元素的一维数组 $A$, 该数组的顺序已定义了该数组的序。并定义领域 $\epsilon$，对于某个元素 $A[i]$，若 $A[i] > A[j], j \in [i-\epsilon, i]$ 且 $A[i] < A[j], j \in [i, i+\epsilon]$，则 $A[i]$ 为极大值，否则，跳出 $\epsilon$ 范围并重复上述操作，直到数组遍历完毕。其算法伪代码如下，其中，设 $\epsilon = 2$

```
Input: A
Output: MaximumSet
i=2
while i<=n−1
    if A[i]>A[i−1]
        if A[i]>A[i+1]
            MaximumSet = MaximumSet ∪ A[i]
    else
        i=i+1
        while i<=n−1 and A[i]<=A[i+1]
            i=i+1
        if i<=n−1
            MaximumSet = MaximumSet ∪ A[i]
    i=i+2
```

在物体检测中，由于初始化的候选边框数量很大，对于要检测的物体，其对应的边框很多，且边框之间的交叉重复特别严重，因此考虑用非极大值抑制来找到最佳的边框。

## 2.10 选择性搜索

## 2.11 候选区域变换

由于使用选择性搜索算法所产生的候选区域是长方形的，但其所包含的元素不定。又因为需要使用卷积神经网络对这些候选区域进行特征提取与降维，这就意味着需要将不定大小的候选区域变换为一个长宽（假设是 $a \times b$）固定的区域。常见的方法如下

**tightest square with context** 该方法考虑的，首先是采用各向同性的方法，将候选区域扩展为 $\max\{a, b\} \times \max\{a, b\}$ 的正方形区域。对于候选区域扩展后无法涉及的区域，用原来的图像对应的像素进行填补，之后再对该正方形进行裁剪，裁剪为 $a \times b$。该想法可理解为扩展后加入背景。

**tightest square without context** 该方法是 tightest square with context 的变体，其在候选区域扩展后无法涉及的区域的处理方法有所不同。其考虑的是对这部分区域不做处理。

**warp** 该方法采用的是各项异性的方法，直接把原来的图像，从长宽方向使用各自的比例进行放缩，直接放缩为 $a \times b$ 。

# 3 RCNN

## 3.1 神经网络

## 3.2 卷积神经网络

### 3.2.1 AlexNet

### 3.2.2 VGGNet

## 3.3 Inception

## 3.4 ResNet

## 3.5 迁移学习

## 3.6 交并比

交并比 (Intersection-over-Union,Iou) 是用于衡量候选框（candidate bound）与标记框（ground truth bound）相似程度的指标，其是两者的交叠率。假设候选框区域为 $C$，标记框区域为 $G$，则定义交并比 $IoU$ 为

$$IoU = \frac{C \cap G}{C \cup G} \tag{37}$$

若 $IoU$ 越接近与 1，说明重叠程度越大，效果越好。



## 3.7 非极大值抑制

非极大值抑制算法 (Non-maximum suppression,NMS) 本质为搜索局部的极大值，并抑制附近非极大值的元素。

若给定一个 $n$ 个元素的一维数组 $A$, 该数组的顺序已定义了该数组的序。并定义领域 $\epsilon$，对于某个元素 $A[i]$，若 $A[i] > A[j], j \in [i-\epsilon, i]$ 且 $A[i] < A[j], j \in [i, i+\epsilon]$，则 $A[i]$ 为极大值，否则，跳出 $\epsilon$ 范围并重复上述操作，直到数组遍历完毕。其算法伪代码如下，其中，设 $\epsilon = 2$

```
Input: A
Output: MaximumSet
i=2
while i<=n-1
    if A[i]>A[i-1]
```

```
        if A[i]>A[i+1]
            MaximumSet = MaximumSet ∪ A[i]
    else
        i=i+1
        while i<=n−1 and A[i]<=A[i+1]
            i=i+1
        if i<=n−1
            MaximumSet = MaximumSet ∪ A[i]
    i=i+2
```

在物体检测中，由于初始化的候选边框数量很大，对于要检测的物体，其对应的边框很多，且边框之间的交叉重复特别严重，因此考虑用非极大值抑制来找到最佳的边框。

### 3.8 选择性搜索

### 3.9 候选区域变换

由于使用选择性搜索算法所产生的候选区域是长方形的，但其所包含的元素不定。又因为需要使用卷积神经网络对这些候选区域进行特征提取与降维，这就意味着需要将不定大小的候选区域变换为一个长宽（假设是 $a \times b$）固定的区域。常见的方法如下

**tightest square with context** 该方法考虑的，首先是采用各向同性的方法，将候选区域扩展为 $\max\{a,b\} \times \max\{a,b\}$ 的正方形区域。对于候选区域扩展后无法涉及的区域，用原来的图像对应的像素进行填补，之后再对该正方形进行裁剪，裁剪为 $a \times b$。该想法可理解为扩展后加入背景。

**tightest square without context** 该方法是 tightest square with context 的变体，其在候选区域扩展后无法涉及的区域的处理方法有所不同。其考虑的是对这部分区域不做处理。

**warp** 该方法采用的是各项异性的方法，直接把原来的图像，从长宽方向使用各自的比例进行放缩，直接放缩为 $a \times b$ 。



## 4 参考文献

[1] Kaiming He,Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,https://arxiv.org/abs/1502.01852