

# 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。

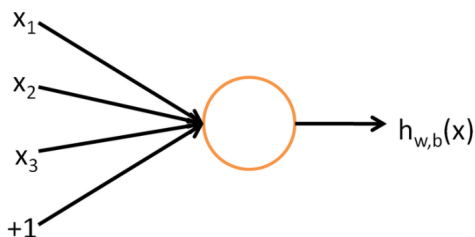


图 1: 神经元

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

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

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

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

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

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

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。

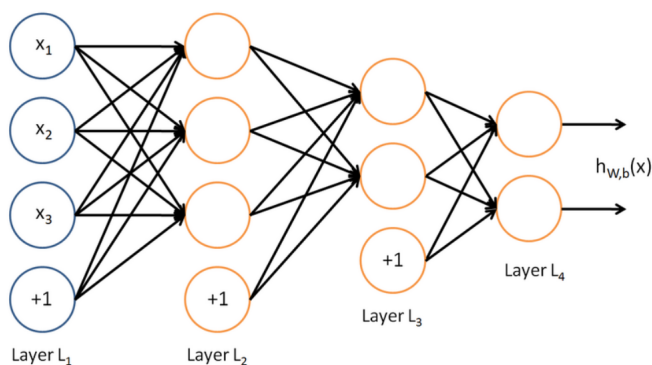


图 2: 含有两个隐含层的神经网络

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

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

其中当  $l=1$  时， $a^{(l)} = x$ ， $x$  为输入向量  $(x_1, x_2, \dots, x_{S_1})$ ， $S_l$  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

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

则有

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

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

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  $L_1, L_2, \dots, L_{n_l}$ ，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

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

接下来计算输出层的残差

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

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

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

则有

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

于是有

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

则可以得到输出层前一层的残差

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

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

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

若取函数  $f$  为 sigmoid 函数，则有

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

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

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

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

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

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

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

又有

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

于是可得

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

综上，可以总结 BP 神经网络算法

### BP 神经网络算法

1 输入：训练输入，训练输出，学习率

2 **while** 未达到收敛条件

3     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  $(L_2, \dots, L_{n_l})$  的激活值  $a^{(2)}, \dots, a^{(n_l)}$

      3. 计算各层残差：

      (1) 对输出层（第  $n_l$  层）

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

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

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

      (3) 计算损失函数对每一层权重的偏导数值

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

      (4) 更新参数

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

4 **end**

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

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

并且代价函数使用交叉熵损失函数

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

则输出层残差为

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

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

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

### 1.3 激活函数

### 1.4 梯度下降法

## 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

$$IoU = \frac{C \cap G}{C \cup G} \tag{29}$$

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  $\epsilon = 2$

```

1 Input: A
2 Output: MaximumSet
3 i=2
4 while i<=n-1
5     if A[i]>A[i-1]
6         if A[i]>A[i+1]
7             MaximumSet = MaximumSet ∪ A[i]
8     else
9         i=i+1
10        while i<=n-1 and A[i]<=A[i+1]
11            i=i+1
12        if i<=n-1
13            MaximumSet = MaximumSet ∪ A[i]
14        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

$$IoU = \frac{C \cap G}{C \cup G} \quad (30)$$

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  $\epsilon = 2$

```

1 Input: A
2 Output: MaximumSet
3 i=2
4 while i<=n-1
5     if A[i]>A[i-1]
6         if A[i]>A[i+1]
7             MaximumSet = MaximumSet ∪ A[i]
8     else

```

```
9      i=i+1
10     while i<=n-1 and A[i]<=A[i+1]
11         i=i+1
12     if i<=n-1
13         MaximumSet = MaximumSet ∪ A[i]
14     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  $a \times b$ 。