第六讲 神经网络学习初步

2025年3月

概要

回顾线性可分支持向量机和逻辑斯谛回归模型

- 都涉及到$x = (x^{(1)}, x^{(2)}, \cdots, x^{(n)})^T$各维特征的线性组合

$$w \cdot x + b = \sum_{i=1}^{n} w_i x^{(i)} + b,$$

这里$w = (w_1, w_2, \cdots, w_n)^T \in \mathbf{R}^n$且$b \in \mathbf{R}$为参数.

- 如果将此各维特征的线性组合$w \cdot x + b$看作是由原来的特征衍生出的新特征，并记为z，则
  - 相应的感知机模型可以表示为：

$$y = \text{sign}(z),$$

  - 可以认为感知机模型是基于衍生特征的变换来进行预测的模型.

- 二项逻辑斯谛模型中给定x的条件下$y=1$的对数几率可以表示为

$$\text{logit}(y=1)=g_l(z),$$

这里$g_l(z)$是恒等变换，而$y=1$的条件概率为

$$P(y=1|x)=\frac{e^z}{1+e^z}.$$

如果我们引进sigmoid 函数

$$\sigma(z)=\frac{1}{1+e^{-z}},$$

则

$$P(y=1|x)=\sigma(z).$$

- 这相当于对衍生特征z利用sigmoid函数进行了一个非线性变换.

更一般地，

- 如果对x的各维特征进行不同的线性组合得到不同的衍生特征;
- 并基于不同的衍生特征的非线性变换得到新的特征;
- 对新的特征向量进行类似线性组合和非线性变换;
- .....
- 进行若干次这样的嵌套操作就可以来逼近一些比较复杂的模型.

本讲介绍的神经网络学习就是基于这样的原理的一类学习方法.

概要

- 在上述逼近复杂模型的方法中，最基本的操作就是对当前的特征进行线性组合并进行(非线性)变换.
- 不妨设当前的（衍生）特征向量为

$$z = (z^{(1)}, z^{(2)}, \cdots, z^{(m)})^T,$$

进行如下线性组合

$$v \cdot z - \theta = \sum_{j=1}^{m} v_j z^{(j)} - \theta,$$

之后进行非线性变换（通常采用sigmoid函数）

$$t = g(v \cdot z - \theta),$$

- 可以用如下图所示的McCulloch-Pitts（简称M-P）神经元模型来刻画这个基本操作:

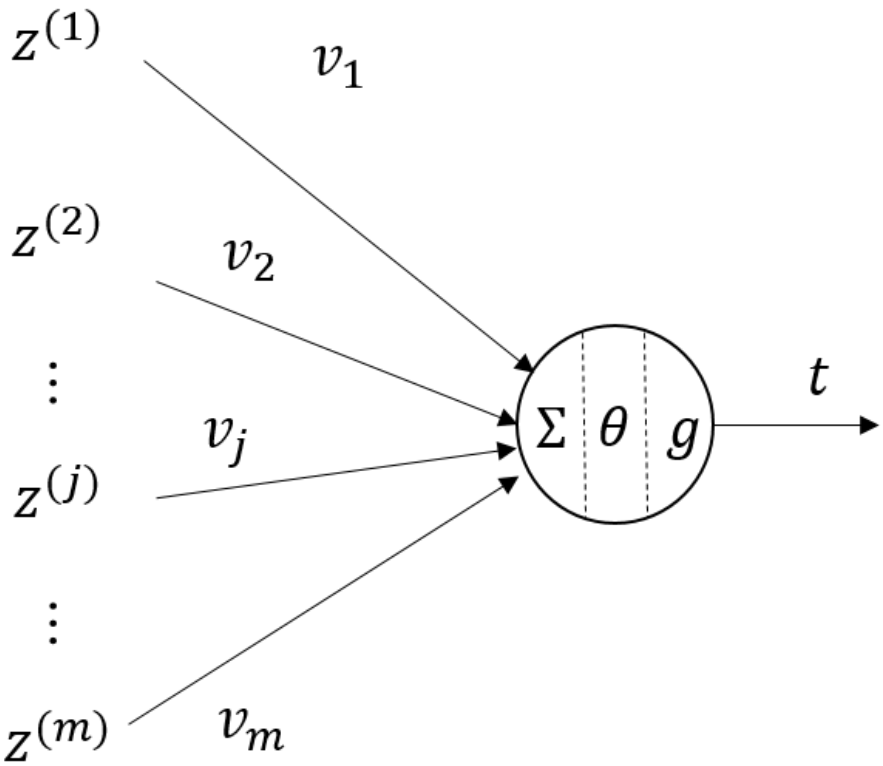


Figure: 神经元模型

在M-P神经元模型中

- 以当前的每一维特征$z^{(j)}$作为神经元接受到的通过带权重v_j的连接进行传播的输入信号;
- 以当前特征的加权和

$$\sum_{j=1}^{m} v_j z^{(j)}$$

作为神经元接受到的总输入;

- 以θ作为神经元的阈值;
- 变换g可以看作是神经元的激活或者响应函数;
- $t = g(v \cdot z - \theta)$为神经元对接受到的输入信号经激活函数处理后产生的输出.

- 对M-P神经元模型来说，比较简洁直观的激活函数为阶跃函数sgn，

$$\mathrm{sgn}(v \cdot z - \theta) = \begin{cases} 1, & v \cdot z \geq \theta; \\ 0, & v \cdot z < \theta. \end{cases}$$

即

  - 在神经元接受到的总输入$v \cdot z$达到阈值θ时神经元兴奋， 阶跃函数输出1值;
  - 在总输入未达阈值时神经元抑制，阶跃函数输出0 值.

- 但在神经网络学习方面，一般用数学性质更好的函数如sigmoid 函数来替代阶跃函数作为神经元的激活函数.

- 要表示前面所讨论的基本操作的嵌套，需要以某种形式将若干个神经元连接起来形成神经网络.
- 多层前馈神经网络(multi-layer feedforeward neural networks)是常见的神经网络结构之一:
  - 该结构逐层排列神经元
    - 神经元的互连仅限于相邻层神经元之间的完全互连;
    - 无同层或者跨层的神经元互连.
  - 用以接受外界输入信号的神经元都处在同一层，称为输入层神经元:
    - 这些神经元并不对输入信号进行激活函数处理，只将接受到的输入信号传递到下一层神经元.
  - 如果以输入层作为第一层，则最后一层神经元为输出神经元，输出最终信号处理结果;
  - 输入层和输出层之间的神经元层,被称为隐层(hidden layer).
  - 隐层和输出层的神经元都对接受到的信号进行激活函数处理，属于功能神经元.

- 两层网络或单隐层前馈神经网络: 只有一个隐层.
- 感知机(Perceptron): 没有隐层神经元.
- 回顾线性可分情形下基于分离超平面(w, b)的分类器就可以表示成一个如下图所示的感知机:
  - 输入层的每个神经元对应x的一维特征$x^{(i)}$;
  - 其与输出神经元之间的连接权重为法向量w的分量w_i;
  - 输出层只有一个M-P 神经元，其阈值为$-b$;
  - 激活函数为$y = \mathrm{sgn}(w \cdot x - (-b))$.

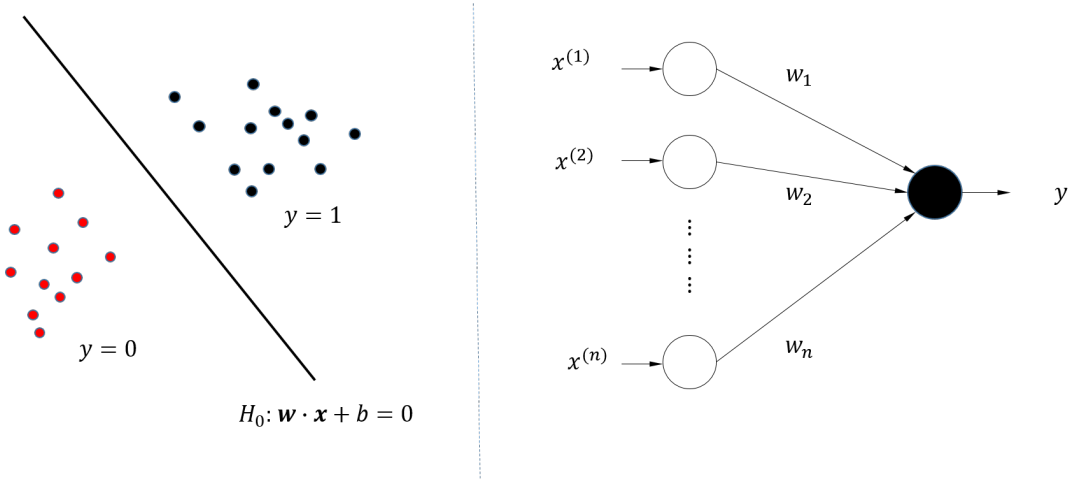


Figure: 分离超平面对应的感知机

- 以阶跃函数

$$y = \mathrm{sgn}(w_1 x^{(1)} + w_2 x^{(2)} - \theta)$$

作为激活函数,感知机能实现逻辑与($\wedge$)、或($\vee$)、非($\neg$)运算.

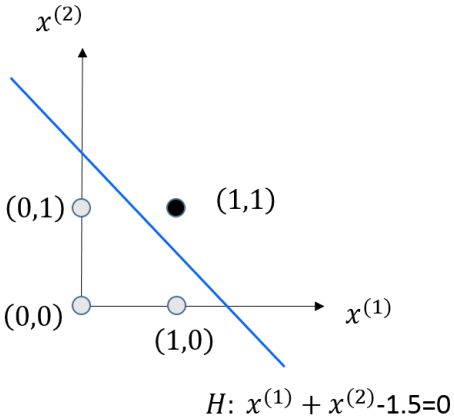
- 对逻辑与来说, $x^{(1)} \wedge x^{(2)} = 1 \iff x^{(1)} = x^{(2)} = 1$, 因此

$$\begin{aligned} w_1 \cdot 1 + w_2 \cdot 1 - \theta \geq 0 &\implies w_1 + w_2 \geq \theta \\ w_1 \cdot 1 + w_2 \cdot 0 - \theta < 0 &\implies w_1 < \theta \\ w_1 \cdot 0 + w_2 \cdot 1 - \theta < 0 &\implies w_2 < \theta \\ w_1 \cdot 0 + w_2 \cdot 0 - \theta < 0 &\implies \theta > 0 \end{aligned}$$

- 显然 $w_1 = w_2 = 1$, $\theta = 1.5$ 满足上述要求，因此我们可以构造如下图所示的感知机来实现逻辑与运算，且相应的超平面

$$H: \ x^{(1)} + x^{(2)} - 1.5 = 0$$

正好可以分割$x^{(1)} \wedge x^{(2)}$结果分别为0和1的点$(x^{(1)}, x^{(2)})$.



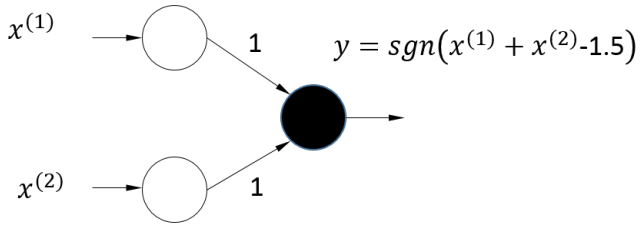
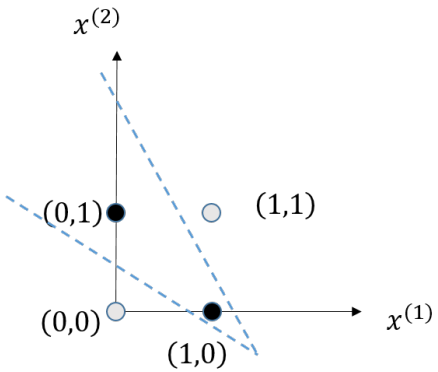


Figure: 逻辑与运算对应的感知机

- 由于感知机只有一层功能神经元，其学习能力非常有限.
- 比如感知机不能实现非线性可分的异或运算.
- 但如果增加一层隐层神经元，构建如下图所示的两层神经网络，就可以实现异或运算:



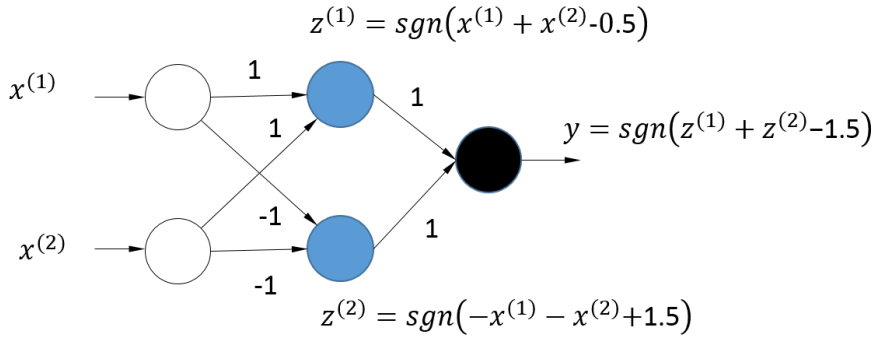


Figure: 异或运算运算对应的两层网络

单隐层前馈神经网络的学习算法.

给定训练数据集 $D = \{(x_i, y_i)\}_{i=1}^{N}$，其中$x_i \in \mathbf{R}^n$，$y_i \in \mathbf{R}^k$。假定我们构造如下单隐层前馈神经网络，其中

- 输入层有n个神经元分别用于接受数据x的n维特征向量$(x^{(1)}, x^{(2)}, \cdots, x^{(n)})^T$的每个分量；
- 输出层有k个功能神经元分别对应于输出$y = (y^{(1)}, y^{(2)}, \cdots, y^{(k)})^T$的每个分量，且输出层第$l$个神经元的阈值为$\theta_l$；
- 隐层有m个功能神经元，且隐层第t个神经元的阈值为γ_t；
- 输入层第j个神经元与隐层第t个神经元之间的连接权值为w_{jt}；
- 隐层第t个神经元与输出层第l个神经元之间的连接权值为v_{tl}；

- 隐层第t个神经元接受到的总输入为$\alpha_t(x)=\sum_{j=1}^{n} w_{jt}x^{(j)}$，输出为$z^{(t)}(x)=\sigma(\alpha_t(x)-\gamma_t)$；
- 输出层第l个神经元接受到的总输入为$\beta_l(x)=\sum_{t=1}^{m} v_{tl}z^{(t)}(x)$，输出为$y^{(l)}=\sigma(\beta_l(x)-\theta_l)$.

我们以Θ表示该单隐层前馈网络中的参数集，即

$$\Theta=(\bigcup_{t=1}^{m}\{w_{jt}\}_{j=1}^{n})\cup(\bigcup_{l=1}^{k}\{v_{tl}\}_{t=1}^{m})\cup\{\theta_l\}_{l=1}^{k}\cup\{\gamma_t\}_{t=1}^{m},$$

其中共包含$(n+k+1)m+k$个参数.

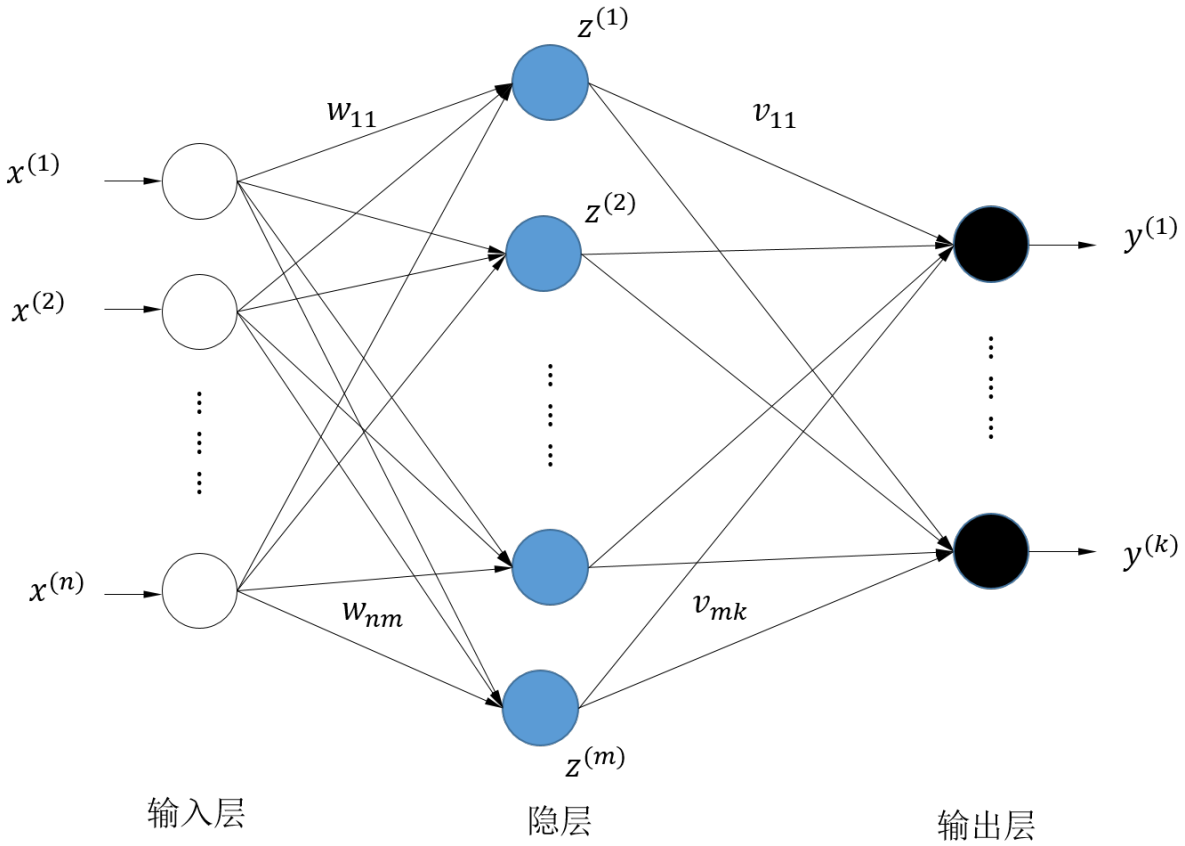


Figure: 单隐层前馈神经网络

概要

- 对训练数据(x_i, y_i)，我们以$\hat{y}_i = (\hat{y}_i^{(1)}, \hat{y}_i^{(2)}, \cdots, \hat{y}_i^{(k)})^T$表示上述单隐层前馈网络的输出，其中

$$\begin{aligned}
\hat{y}_i^{(l)} &= \sigma(\beta_l(x_i) - \theta_l) \\
&= \sigma(\sum_{t=1}^{m} v_{tl} z^{(t)}(x_i) - \theta_l) \\
&= \sigma(\sum_{t=1}^{m} v_{tl} \sigma(\sum_{j=1}^{n} w_{jt} x_i^{(j)} - \gamma_t) - \theta_l),\ l = 1, 2, \ldots, k.
\end{aligned}$$

- 采用平方误差作为预测损失函数，则该网络在(x_i, y_i)上的损失为

$$R_i(\Theta) = \| y_i - \hat{y}_i \|_2^2 = \sum_{l=1}^{k} (y_i^{(l)} - \hat{y}_i^{(l)})^2.$$

- 在训练数据集D上的损失为

$$R(\Theta) = \sum_{i=1}^{N} R_i(\Theta).$$

- 如果采用经验风险最小化策略，则网络参数的估计$\hat{\Theta}$应该满足

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}}\, R(\Theta).$$

如果考虑采用梯度下降法来求参数的估计，则需要求出$R(\Theta)$对各参数的偏导数:

- 先计算$R(\Theta)$对v_{tl}的偏导数

$$\frac{\partial R(\Theta)}{\partial v_{tl}} = \sum_{i=1}^{N} \frac{\partial R_i(\Theta)}{\partial v_{tl}},$$

其中

$$\begin{aligned}
\frac{\partial R_i(\Theta)}{\partial v_{tl}} &= \frac{\partial R_i(\Theta)}{\partial \hat{y}_i^{(l)}} \frac{\partial \hat{y}_i^{(l)}}{\partial v_{tl}} = -2(y_i^{(l)} - \hat{y}_i^{(l)}) \frac{\partial \hat{y}_i^{(l)}}{\partial v_{tl}} \\
&= -2(y_i^{(l)} - \hat{y}_i^{(l)}) \frac{\partial \hat{y}_i^{(l)}}{\partial \beta_l} \frac{\partial \beta_l}{\partial v_{tl}} \\
&= -2(y_i^{(l)} - \hat{y}_i^{(l)}) \hat{y}_i^{(l)} (1 - \hat{y}_i^{(l)}) z^{(t)}(x_i).
\end{aligned}$$

$R(\Theta)$对θ_l的偏导数

$$\frac{\partial R(\Theta)}{\partial \theta_l} = \sum_{i=1}^{N} \frac{\partial R_i(\Theta)}{\partial \theta_l},$$

其中

$$\begin{aligned}
\frac{\partial R_i(\Theta)}{\partial \theta_l} &= \frac{\partial R_i(\Theta)}{\partial \hat{y}_i^{(l)}} \frac{\partial \hat{y}_i^{(l)}}{\partial \theta_l} \\
&= -2(y_i^{(l)} - \hat{y}_i^{(l)}) \frac{\partial \hat{y}_i^{(l)}}{\partial \theta_l} \\
&= 2(y_i^{(l)} - \hat{y}_i^{(l)}) \hat{y}_i^{(l)} (1 - \hat{y}_i^{(l)}).
\end{aligned}$$

我们令

$$\delta_i^{(l)} = -2(y_i^{(l)} - \hat{y}_i^{(l)})\hat{y}_i^{(l)}(1 - \hat{y}_i^{(l)}),$$

则

$$\frac{\partial R_i(\Theta)}{\partial v_{tl}} = \delta_i^{(l)} z^{(t)}(x_i),$$
$$\frac{\partial R_i(\Theta)}{\partial \theta_l} = -\delta_i^{(l)}.$$

$R(\Theta)$对w_{jt}的偏导数 $\frac{\partial R(\Theta)}{\partial w_{jt}} = \sum_{i=1}^{N} \frac{\partial R_i(\Theta)}{\partial w_{jt}}$，其中

$$\frac{\partial R_i(\Theta)}{\partial w_{jt}} = -2\sum_{l=1}^{k}(y_i^{(l)} - \hat{y}_i^{(l)})\frac{\partial \hat{y}_i^{l}(\Theta)}{\partial w_{jt}}$$

$$= -2\sum_{l=1}^{k}(y_i^{(l)} - \hat{y}_i^{(l)})\frac{\partial \hat{y}_i^{(l)}(\Theta)}{\partial \beta_l}\frac{\partial \beta_l}{\partial w_{jt}}$$

$$= -2\sum_{l=1}^{k}(y_i^{(l)} - \hat{y}_i^{(l)})\hat{y}_i^{(l)}(1-\hat{y}_i^{(l)})v_{tl}z^{(t)}(x_i)(1-z^{(t)}(x_i))x_i^{(j)}$$

$$= z^{(t)}(x_i)(1-z^{(t)}(x_i))\left[\sum_{l=1}^{k} -2(y_i^{(l)} - \hat{y}_i^{(l)})\hat{y}_i^{(l)}(1-\hat{y}_i^{(l)})v_{tl}\right]x_i^{(j)}$$

$$= z^{(t)}(x_i)(1-z^{(t)}(x_i))\left[\sum_{l=1}^{k}\delta_i^{(l)}v_{tl}\right]x_i^{(j)}.$$

$R(\Theta)$对γ_t的偏导数 $\frac{\partial R(\Theta)}{\partial \gamma_t} = \sum_{i=1}^{N} \frac{\partial R_i(\Theta)}{\partial \gamma_t}$, 其中

$$\begin{aligned}
\frac{\partial R_i(\Theta)}{\partial \gamma_t} &= -2\sum_{l=1}^{k}(y_i^{(l)} - \hat{y}_i^{(l)})\frac{\partial \hat{y}_i^{l}(\Theta)}{\partial \gamma_t} \\
&= -2\sum_{l=1}^{k}(y_i^{(l)} - \hat{y}_i^{(l)})\frac{\partial \hat{y}_i^{(l)}(\Theta)}{\partial \beta_l}\frac{\partial \beta_l}{\partial \gamma_t} \\
&= 2\sum_{l=1}^{k}(y_i^{(l)} - \hat{y}_i^{(l)})\hat{y}_i^{(l)}(1 - \hat{y}_i^{(l)})v_{tl}z^{(t)}(x_i)(1 - z^{(t)}(x_i)) \\
&= -z^{(t)}(x_i)(1 - z^{(t)}(x_i))\left[\sum_{l=1}^{k}\delta_i^{(l)}v_{tl}\right].
\end{aligned}$$

我们令

$$s_i^{(t)} = z^{(t)}(x_i)(1 - z^{(t)}(x_i)) \left[\sum_{l=1}^{k} \delta_i^{(l)} v_{tl} \right],$$

则

$$\begin{aligned} \frac{\partial R_i(\Theta)}{\partial w_{jt}} &= s_i^{(t)} x_i^{(j)}, \\ \frac{\partial R_i(\Theta)}{\partial \gamma_t} &= -s_i^{(t)}. \end{aligned}$$

给定学习率η, 梯度下降法相应的权值更新公式为：

$$w_{jt} = w_{jt} - \eta \sum_{i=1}^{N} s_i^{(t)} x_i^{(j)},$$

$$v_{tl} = v_{tl} - \eta \sum_{i=1}^{N} \delta_i^{(l)} z^{(t)}(x_i);$$

相应的阈值更新公式为：

$$\theta_l = \theta_l + \eta \sum_{i=1}^{N} \delta_i^{(l)},$$

$$\gamma_t = \gamma_t + \eta \sum_{i=1}^{N} s_i^{(t)},$$

这里 $j = 1, 2, \ldots, n; t = 1, 2, \ldots, m; l = 1, 2, \ldots, k$.

在参数更新公式中

- 每个$x_i^{(j)}$属于输入信号.
- $z^{(t)}(x_i)$为隐层第t个神经元基于连接权值和阈值参数的当前值对接受到的总输入处理以后的输出.
- $\delta_i^{(l)}$和$s_i^{(t)}$分别为输出层第l个神经元和隐层第t个神经元基于当前参数值的误差.
- $s_i^{(t)}$可以由$\delta_i^{(l)}$按照式

$$s_i^{(t)} = z^{(t)}(x_i)(1 - z^{(t)}(x_i)) \left[\sum_{l=1}^{k} \delta_i^{(l)} v_{tl} \right]$$

来计算.

隐层神经元误差的计算相当于

- 先计算隐层神经元的输出$z^{(t)}(x_i)$和输出层神经元的输出$\hat{y}_i^{(l)}$;
- 得到输出层神经元的输出后计算输出神经元的误差$\delta_i^{(l)}$;
- 然后将每个$\delta_i^{(l)}$通过隐层神经元和输出层神经元之间的连接反向传播到隐层各神经元来计算各隐层神经元误差$s_i^{(t)}$.

因此该估计参数的梯度下降法也被称为误差反向传播算法.

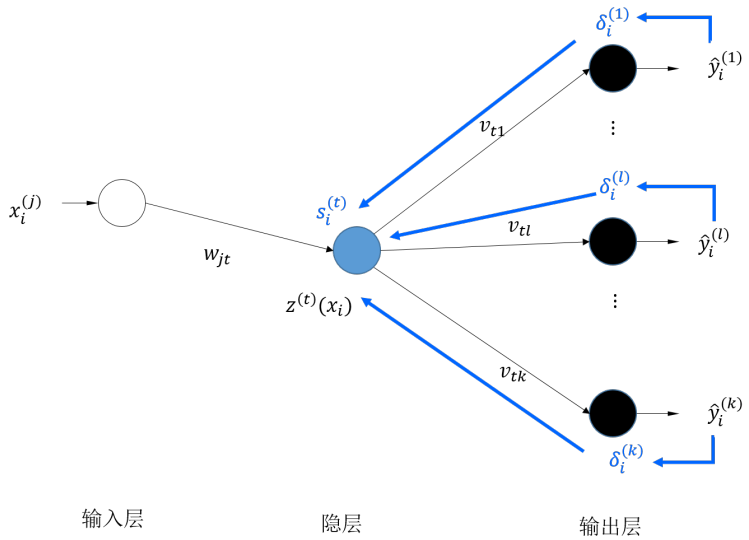


Figure: 误差反向传播示意

- 误差反向传播算法中参数初始化操作通常随机选择$(0,1)$区间比较接近0的值作为参数的初始值.
- 与随机梯度下降法类似,也可以针对每个训练样本进行一次参数更新,即
  - 每次用$R_i(\Theta)$代替$R(\Theta)$作为目标函数,
  - 只考虑误差$\delta_i^{(l)}$的反向传播来更新参数.

  进而在整个训练数据集上循环进行该迭代过程, 直到满足停止条件.
  - 可以处理训练数据集规模比较大的情形.
  - 在获得新训练数据的时候也比较方便更新网络连接权值和神经元阈值参数.

- 采用正则化策略来缓解神经网络学习的过拟合风险:

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}} \left[R(\Theta) + \lambda J(\Theta) \right],$$

这里$\lambda \geq 0$为正则化系数, 其中

$$J(\Theta) = \sum_{j=1}^{n} \sum_{t=1}^{m} w_{jt}^2 + \sum_{t=1}^{m} \sum_{l=1}^{k} v_{tl}^2 + \sum_{t=1}^{m} \gamma_t^2 + \sum_{l=1}^{k} \theta_l^2,$$

- 早停(early stopping)也是缓和过拟合的策略之一.

- 对sigmoid函数$\sigma(x)$来说，

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)),$$

则当$\sigma(x) \to 0$或者$\sigma(x) \to 1$时，

$$\sigma'(x) \to 0.$$

- 除了sigmoid函数，可以采用

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

作为隐层神经元的激活函数.

- 对tanh函数来说，

$$\tanh'(x) = 1 - \tanh^2(x)$$

则当$\tanh(x) \to 1$或者$\tanh(x) \to -1$时，

$$\tanh'(x) \to 0.$$

- 这意味着如果以上述两个函数作为多层前馈神经网络隐层神经元的激活函数，在有些情况下可能导致梯度趋于0而参数更新很慢.

激活函数

- 整流线性rectified linear函数:

$$g_r(x) = \max(0, x).$$

  - 在$x < 0$时，$g_r'(x) = 0$;而在$x > 0$时，$g_r'(x) = 1$.
  - 这使得相关的梯度计算比较方便.

- 带泄露的整流线性（leaky rectified linear）函数:

$$g_\lambda(x) = \begin{cases} x, & x > 0 \\ \lambda x, & x \leq 0 \end{cases},$$

这里$\lambda \in (0, 1)$是一个常数.

  - 在$x > 0$时，$g_\lambda'(x) = g_r'(x) = 1$;
  - 而在$x < 0$时，$g_\lambda'(x) = \lambda$也是一个常数.

小结

- M-P神经元模型
- 多层前馈神经网络
- 感知机
- 单隐层前馈神经网络
- 误差反向传播算法