Lecture10 感知器&神经网络

1. 感知器 Perceptron

介绍

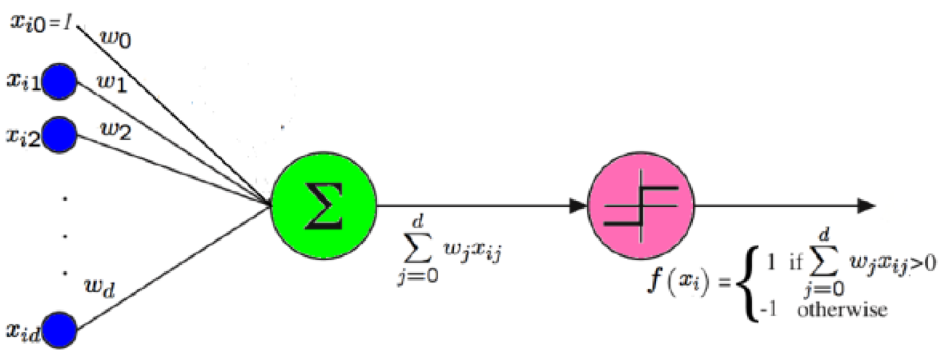
- 属于**神经网络 Neural Networks** 类算法（试图模拟大脑功能的算法）
- 神经网络中的第一个算法，就是**感知器**
- 在**识别**方面非常有效
  - 手写文字识别 (LeCun et a. 1989)
  - 语音识别 (Lang et al. 1990)
  - 面部识别(Cottrel 1990)
- **神经网络 NN** 在90年代很受欢迎，但后来失去了一些人气，但是现在有回到了深度学习的视野中

感知器模仿生物中神经元的样子

给定 n 个样本，每个样本有 d 维的特征

$$f(x_i) = sign(\sum_{j=0}^{d} w_j x_{ij})$$

- 其中，$x_{i0} = 1$，不是样本的维度特征
- w_j 是每个维度的权重，给每个维度分配权重方便加总
- $f(x_i)$ 使用阶梯函数进行激活



- 只有 $\sum_{j=0}^{d} w_j x_{ij} > 0$，$f(x_i)$ 被激活，等于 1

参数调整

这种感知器在线性可分的时候，效果可以，如果线性不可分，由于阶梯函数不可导，函数不收敛

现在我们要调整每一个 w_j，可以从一个随机的超平面开始，来调整训练数据，迭代调整

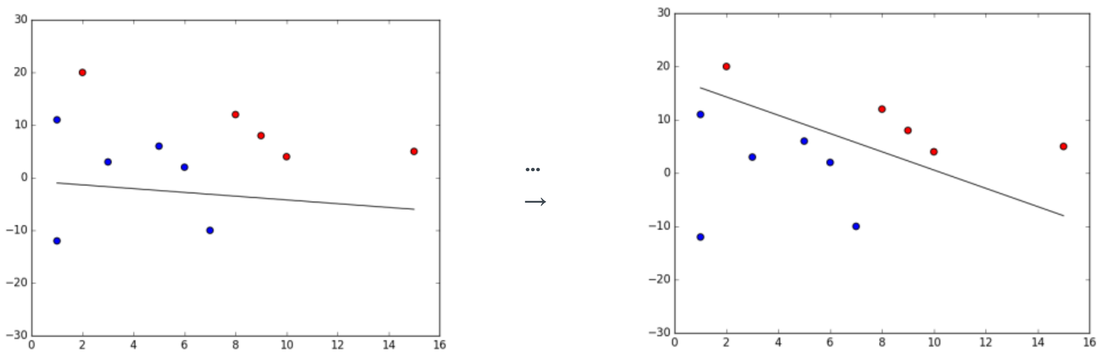
- 输入：一系列样本 $(x_1, y_1), \ldots, (x_n, y_n)$
  - $x_i \in \mathbb{R}^d$
  - $y_i \in [-1, 1]$
- 输出：一个由 $(w_0, w_1, \ldots, w_d)$ 定义好的感知器

`Perceptron Algorithm`

1. 初始化所有的权重 $w_j = 0 \quad \forall j \in \{0, \ldots, d\}$
2. 重复直到收敛
3. 　　对于每一个样本 $x_i \quad \forall i \in \{1, \ldots, n\}$
4. 　　if $y_i f(x_i) \leq 0$ // 判断当前样本是否分类错误
5. 　　　　更新所有的 w_j，使得 $w_j := w_j + y_i x_{ij}$ // 如果分类错误，将权值调整一下

观察算法可以得到

- $w_1, \ldots, w_d$ 决定了超平面的斜率
- w_0 确定决策边界的偏移量，有时候也会被记作 b
- 收敛发生在所有的权重 w_j 都不再改变的时候

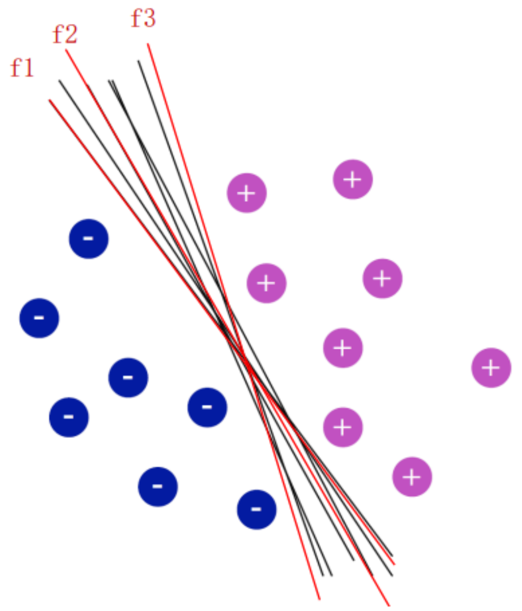


总结

每一个 w_j 表示了样本 x_i 的第 j 个特征 x_{ij} 对于分类的贡献度

$-w_0$ 也被称为**阈值**，因为 $\sum_{j=1}^{d} w_j x_{ij} + w_0 > 0 \rightarrow \sum_{j=1}^{d} w_j x_{ij} > -w_0$ 才可以激活 $f(x_i)$

一个需要注意的是，感知器调整的参数只能把所有的样本划分开，但是它不会在分类的时候选择一个更好的分类标准



- 如上图，感知器可能会通过调整参数到 f_1 或者 f_3，但是只要收敛了后，它就不会继续计算
- 不会算出 f_2 这样一个可能效果更好的解
    - 这里 SVM（支持向量机）就可以找到

2. 神经网络 Neural Network

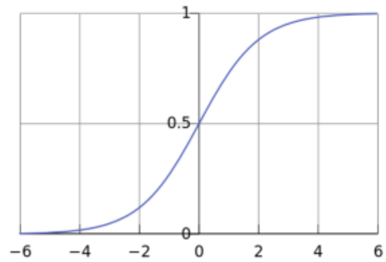
从感知器到神经网络

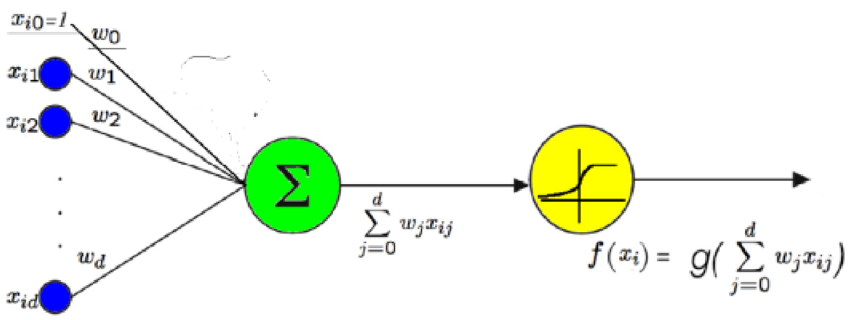
- 神经网络使用**感知器**的能力来表示基本函数，并将它们组合在一个层网络中
- 然而，线性函数级联仍然是线性的，我们希望网络能够代表高度**非线性的函数**

与逻辑回归模型相似，我们将感知器的**阶梯函数**更换成 **sigmoid 函数**

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

$$g(z) \to 1 \text{ when } z \to +\infty$$
$$g(z) \to 0 \text{ when } z \to -\infty$$





给定 n 个样本，每个样本有 d 维的特征，对于给定的样本 x_i，激活函数 $f(x_i)$ 为

$$f(x_i) = \frac{1}{1+e^{-\sum_{j=0}^{d} w_j x_{ij}}}$$

神经网络的逻辑表达

感知器可以表达许多布尔函数 AND、OR、NAND、NOR、NOT（不过不能表达 XOR）

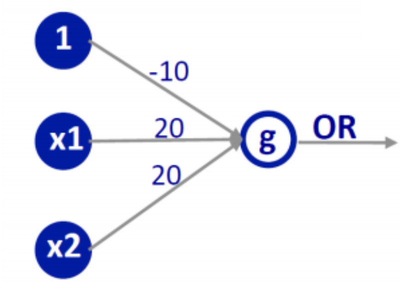
首先观察，sigmoid 函数有一个特点

$$g(z) = \frac{1}{1+e^{-z}}$$
$$g(10) = 0.99995 \quad g(-10) = 0.00004$$

如果按照 0.5 为激活函数 = 1 或 = 0 的划分的话

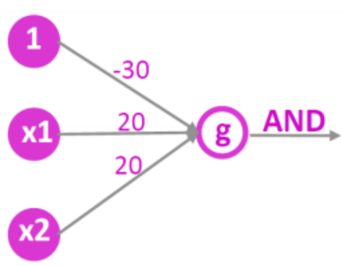
- 对于 $z \geq 10, g(z) \rightarrow 1$
- 对于 $z \leq 10, g(z) \rightarrow 0$

逻辑或 OR



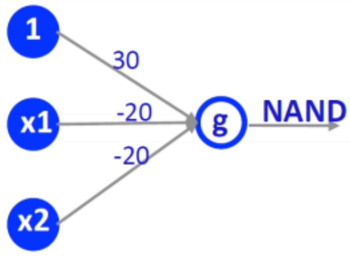
x_{i1}	x_{i2}	x_{i1} **OR** x_{i2}	$g(z) = g(w_0 + w_1x_1 + w_2x_2)$
0	0	0	$g(-10)$
0	1	1	$g(10)$
1	0	1	$g(10)$
1	1	1	$g(30)$

逻辑与 AND



x_{i1}	x_{i2}	x_{i1} **AND** x_{i2}	$g(z) = g(w_0 + w_1x_1 + w_2x_2)$
0	0	0	$g(-30)$
0	1	0	$g(-10)$
1	0	0	$g(-10)$
1	1	1	$g(10)$

逻辑与非 NAND

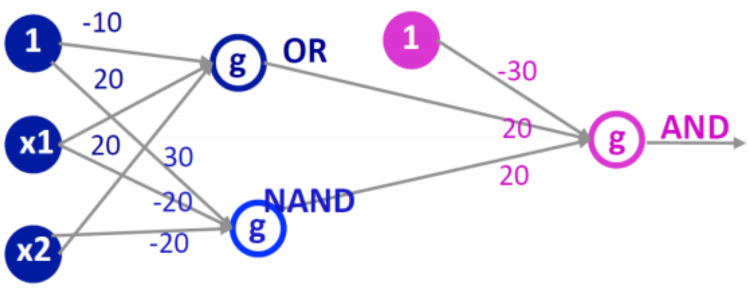


x_{i1}	x_{i2}	x_{i1} **NAND** x_{i2}	$g(z) = g(w_0 + w_1x_1 + w_2x_2)$
0	0	1	$g(30)$
0	1	1	$g(10)$
1	0	1	$g(10)$
1	1	0	$g(-10)$

逻辑异或 XOR

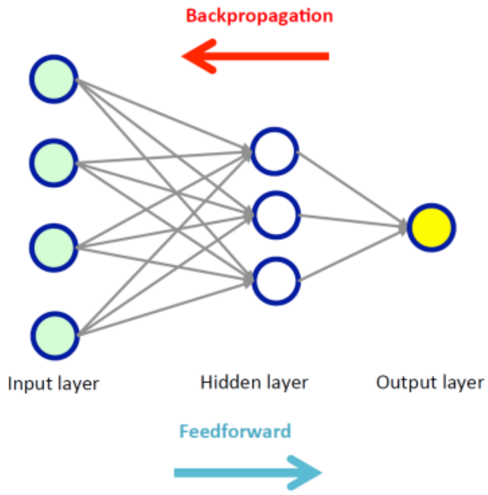
现在我们可以用基本逻辑感知器来构建神经网络 NN，来表示异或的逻辑

x_1	x_2	x_1 XOR x_2	$(x_1$ OR $x_2)$ AND $(x_1$ NAND $x_2)$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0



XOR 是 3 个基本的感知器的组合

3. 反向传播 Backpropagation



- 前馈神经网络（与递归网络相反）没有环路的连接
- 但是我们想要同时学习**多层网络的权值**
- Backpropagation 的意思是“误差的反向传播”
- 给定一个具有固定结构的网络（神经元和互连）
- 使用梯度下降来最小化神经网络的输出值 o 和标签 y 的均方误差
- 我们假设有多个输出 k
- 搜索网络中所有神经元的所有可能的权重值

符号定义

在将接下来的东西之前，先定义一组符号

- x_{ij}：神经元 j 的第 i 个输入
- w_{ij}：神经元 j 接收第 i 个输入时的权重参数
- $z_j = \sum w_{ij}x_{ij}$：神经元 j 的输入加权和
- o_j：神经元 j 的激活函数激活函数结果
    - $o_j = g(z_j) = g(\sum w_{ij}x_{ij} + w_0)$
- $output$：输出层的神经元集合
- $Succ(j)$：一组神经元集合，其中每个神经元都会将神经元 j 作为输入参数

反向传播规则

我们假设有 k 个输出

对于一个由 (x, y) 定义的样本 e，对于网络中的 k 个神经元输出的误差加和为

$$E_e(\boldsymbol{w}) = \frac{1}{2}\sum_k (y_k - o_k)^2$$

- y_k：样本的第 k 个输出标签
- o_k：样本通过神经网络的第 k 个输出

- $E_e(\boldsymbol{w})$：样本 e 在一组参数 $\boldsymbol{w}$ 下的误差

梯度下降每一次计算完一样本后进行一次迭代的更新，降低误差的梯度

$$\Delta w_{ij} = -\alpha \frac{\partial E_e(w)}{\partial w_{ij}} g$$

根据链式法则，我们得到

$$\frac{\partial E_e}{\partial w_{ij}} = \frac{\partial E_e}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} = \frac{\partial E_\mathrm{e}}{\partial z_\mathrm{j}} x_{ij}$$
$$\Delta w_{ij} = -\alpha \frac{\partial E_\mathrm{e}}{\partial z_\mathrm{j}} x_{ij}$$

现在我们要考虑计算 $\frac{\partial E_\mathrm{e}}{\partial z_\mathrm{j}}$ 的两种情况，我们定义梯度符号如下：

$$\delta_j = -\frac{\partial E}{\partial z_j}$$
$$\Delta w_{ij} = \alpha \delta_j x_{ij}$$

神经元 j 是输出神经元

再次根据链式法则

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial z_j}$$

对于 $\frac{\partial E}{\partial o_j}$ 来说

$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_k (y_k - o_k)^2$$
$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} (y_j - o_j)^2$$
$$\frac{\partial E}{\partial o_j} = \frac{1}{2} 2 (y_j - o_j) \frac{\partial (y_j - o_j)}{\partial o_j}$$
$$\frac{\partial E}{\partial o_j} = -(y_j - o_j)$$

- 损失函数用的是最小二乘损失函数

对于 $\frac{\partial o_j}{\partial z_j}$ 来说

$$
\begin{aligned}
& o_j = g\left(z_j\right) \\
& \frac{\partial o_j}{\partial z_j} = \frac{\partial g\left(z_j\right)}{\partial z_j} \\
& \frac{\partial o_j}{\partial z_j} = o_j\left(1 - o_j\right)
\end{aligned}
$$

- 激活函数用的是 sigmoid 函数

所以

$$
\begin{aligned}
\frac{\partial E}{\partial z_j} & = \delta_j = -\left(y_j - o_j\right) o_j\left(1 - o_j\right) \\
\Delta w_{ij} & = \alpha(y_j - o_j) o_j (1 - o_j) x_{ij}
\end{aligned}
$$

神经元 j 是隐藏层神经元

$$
\begin{aligned}
& \frac{\partial E}{\partial z_j} = \sum_{k \in \operatorname{succ}\{j\}} \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial z_j} = \sum_{k \in succ\{j\}} -\delta_k \frac{\partial z_k}{\partial z_j} \\
& \frac{\partial E}{\partial z_j} = \sum_{k \in \operatorname{succ}\{j\}} -\delta_k \frac{\partial z_k}{\partial o_j} \frac{\partial o_j}{\partial z_j} \\
& \frac{\partial E}{\partial z_j} = \sum_{k \in \operatorname{succ}\{j\}} -\delta_k w_{jk} \frac{\partial o_j}{\partial z_j} \\
& \frac{\partial E}{\partial z_j} = \sum_{k \in \operatorname{succ}\{j\}} -\delta_k w_{jk} o_j\left(1 - o_j\right) \\
& \delta_j = -\frac{\partial E}{\partial z_j} = o_j\left(1 - o_j\right) \sum_{k \in \operatorname{succ}\{j\}} \delta_k w_{jk}
\end{aligned}
$$

反向传播算法

- 输入
    - 训练样本 $(x_1, y_1), \ldots, (x_n, y_n), x_i \in \mathbb{R}^d, y_i \in \mathbb{R}^k$
    - 学习率 α
    - 输入层网络神经元个数 n_i，隐层网络神经元个数 n_h，输出层网络神经元个数 n_o
- 输出
    - 一个有着 n_i 个神经元的输入层，有着 n_h 个神经元的隐层，n_o 个神经元的输出层以及各个神经元之间的权重

`Backpropagation Algorithm - BP`

1. 创建前向网络 (n_i, n_h, n_o)
2. 初始化所有的权重，让它们是一个比较小的随机数，例如 $[-0.2, 0.2]$
3. 重复下面的步骤直到收敛
4. 　　对于每个训练样本 (x_i, y_i)
5. 　　　　**前馈 Feed forward**：将每个样本 x_i 通过网络层，计算每一个神经元的输出 o_j

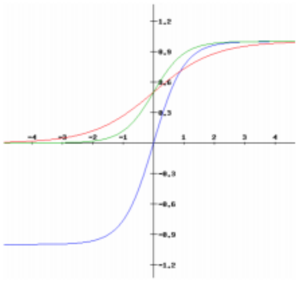
6. **反向传播 Propagate backward**：将误差向后传播
7. Case1：对于输出神经元 k，它的误差为 $\delta_k = -(y_k - o_k)o_k(1 - o_k)$
8. Case2：对于隐层神经元 h，它的误差为 $\delta_h = o_h(1 - o_h)\sum_{k \in \mathrm{succ}\{h\}} \delta_k w_{hk}$
9. **更新权重 Update each weight**：$w_{ij} := w_{ij} + \alpha\delta_j x_{ij}$

4. 补充

Hyperbolic tangent 函数

其实还有其它的比较好的激活函数，比如 Hyperbolic tangent 函数对于神经网络也比较好，它的输出范围是 $[-1, 1]$

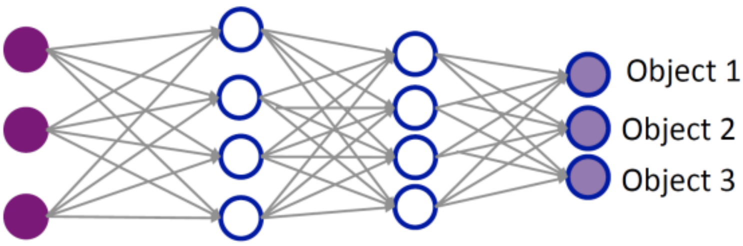
- sigmoid 函数：$g(x) = \frac{e^{kx}}{1+e^{kx}}$，$k$ 是一个调整的常数
- hiperbolic tan 函数：$g(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$（它其实是 sigmoid 函数的一个缩放）



$g(x) = sigmoid(x) = \frac{e^{kx}}{1+e^{kx}}$ for $k = 1$, $k = 2$, etc.

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

多神经网络层的结构



如今，两层以上的网络，又称**深度网络 deep networks**，已经在许多领域被证明是非常有效的

深度网络的例子

- 受限玻尔兹曼机器 Restricted Boltzman machines
- 卷积神经网络 convolutional NN
- 自动编码器 auto encoders

MNIST 数据集

- MNIST 手写数字数据库
- 训练集有 60,000 个样本，测试集有 10,000 个样本
- 输入样本的维度是是 $\mathbb{R}^{784}$（28 x 28 像素）
- 标签的维度是它们代表的数字数值（0-9）$\mathbb{R}^{10}$
- 有各种方法已经用这个训练集和测试集进行了测试
    - 线性模型 7% ~ 12% 的误差
    - KNN：0.5% ~ 5% 的误差

- 神经网络：0.35% ~ 4.7% 的误差
- 卷积神经网络：0.23% ~ 1.7% 的误差