Studies ML

Jacob Xie

2023-03-11

2 模型评估与选择

2.1 误差与拟合

误差与拟合		
名称	英文	描述
错误率	error rate	如果在 m 个样本中有 a 个样本分类错误，则错误率 $E = a/m$
精度	accuracy	$1 - a/m$
误差	error	学习器的实际预输出与样本的真实输出之间的差异
训练误差	training error	学习器在训练集上的误差
经验误差	empirical error	
泛化误差	generalization error	在新样本上的误差
过拟合	over fitting	学习器把训练样本自身的一些特点当做了所有潜在样本都会具有的一般性质，导致泛化性能下降
欠拟合	under fitting	与过拟合相对应
模型选择	model selection	

2.2 评估方法

评估方法		
测试集	testing set	
测试误差	testing error	
留出法	hold-out	直接将数据集 D 划分为两个互斥的集合，其中一个集合作为训练集 S，另一个作为测试集 T，即 $D = S \cup T, S \cap T = \varnothing$。在 S 上训练出模型后，用 T 来评估其测试误差，作为对泛化误差的估计。
采样	sampling	
分层采样	stratified sampling	保留类别比例的采样方式
保真性	fidelity	
交叉验证法	cross validation	将数据集 D 划分为 k 个大小相似的互斥子集，即 $D = D_1 \cup D_2 \cup \cdots \cup D_k, D_i \cap D_j = \varnothing \ (i \neq j)$。每个子集 D_i 都尽可能保持数据分布的一致性，即从 D 中通过分层采样的到。然后每次用 $k-1$ 个子集的并集作为训练集，余下的那个子集作为测试集；这样就可获得 k 组训练/测试集，从而可进行 k 次训练和测试，最终返回的是这 k 个测试结果的均值。
k 折交叉验证	k-fold cross validation	
留一法	leave-one-out	
自助法	bootstrapping	
自助采样法	bootstrap sampling	
包外估计	out-of-bag estimate	
参数	parameter	
调参	parameter tuning	
验证集	validation set	

2.3 性能度量

2.3.1 错误率与精度

性能度量（performance measure）：衡量模型泛化能力的评价标准。

均方误差（mean squared error）：

$$E(f;D)=\frac{1}{m}\sum_{i=1}^{m}(f(\boldsymbol{x}_i)-y_i)^2 \tag{2.2}$$

对于数据分布 $\mathcal{D}$ 和概率密度函数 $p(\cdot)$，均方误差可描述为：

$$E(f;\mathcal{D})=\int_{\boldsymbol{x}\ \mathcal{D}}(f(\boldsymbol{x}-y)^2)p(\boldsymbol{x})\,d\boldsymbol{x} \tag{2.3}$$

错误率是分类错误的样本数占样本总数的比例：

$$E(f;D)=\frac{1}{m}\sum_{i=1}^{m}\mathbb{I}(f(\boldsymbol{x}_i)\neq y_i) \tag{2.4}$$

精度则是分类正确的样本数占样本总数的比例：

$$\begin{aligned}acc(f;D)&=\frac{1}{m}\sum_{i=1}^{m}\mathbb{I}(f(\boldsymbol{x}_i)=y_i)\\&=1-E(f;D)\end{aligned} \tag{2.5}$$

对于数据分布 $\mathcal{D}$ 和概率密度函数 $p(\cdot)$，错误率与精度可分别描述为

$$E(f;\mathcal{D})=\int_{\boldsymbol{x}\ \mathcal{D}}\mathbb{I}(f(\boldsymbol{x})\neq y)p(\boldsymbol{x})\,d\boldsymbol{x} \tag{2.6}$$

$$\begin{aligned}acc(f;\mathcal{D})&=\int_{\boldsymbol{x}\ \mathcal{D}}\mathbb{I}(f(\boldsymbol{x})=y)p(\boldsymbol{x})\,d\boldsymbol{x}\\&=1-E(f;\mathcal{D})\end{aligned} \tag{2.7}$$

2.5 偏差与方差

偏差-方差分解（bias-variance decomposition）：对学习算法的期望泛化错误率进行拆解。

偏差-方差窘境（bias-variance dilemma）

3 线性模型

4 决策树

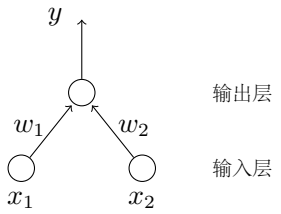
5 神经网络

5.1 神经元模型

神经元模型		
名称	英文	描述
神经网络	neural networks	由具有适应性的简单单元组成的广泛并行互联的网络，它的组织能够模拟生物神经系统对真实世界物体所作出的交互反应
神经元	neuron	神经网络中最基本的成分
阈值	threshold	公式中记作 θ
连接	connection	
激活函数	activation function	处理以产生神经元的输出
挤压函数	squashing function	

5.2 感知机与多层网络

感知机（Perceptron）由两层神经元组成。如图示，输入层接受外界输入信号后传递给输出层，输出层是 M-P 神经元，也称阈值逻辑单元（threshold logic unit），其中 $y = f(\sum_i w_i x_i - \theta)$，而 f 为激活函数。



一般而言，给定训练数据集，权重 $w_i\,(i = 1, 2, \ldots, n)$ 以及阈值 θ 可通过学习得到。而阈值 θ 可视为一个固定输入为 -1.0 的哑结点（dummy node）所对应的链接权重 w_{n+1}，因此权重和阈值的学习就可以统一为权重的学习。感知机的学习规则非常简单，对训练样例 $(\boldsymbol{x}, y)$ 而言，如果当前感知机的输出位 $\hat{y}$，则感知机权重将这样调整：

$$w_i \leftarrow w_i + \Delta w_i \tag{5.1}$$

$$\Delta w_i = \eta(y - \hat{y})x_i \tag{5.2}$$

其中 $\eta \in (0,1)$ 称为学习率（learning rate）。公式 5.2 为感知机学习算法中的参数更新式。

I 感知机模型

感知机模型的式可表示为:

$$\begin{aligned} y &= f\left(\sum_{i=1}^{n} w_i x_i - \theta\right) \\ &= f(\boldsymbol{w}^T \boldsymbol{x} - \theta) \end{aligned}$$

其中，$x \in \mathbb{R}^n$ 即样本的特征向量，是感知机模型的输入；$\boldsymbol{w}, \theta$ 是感知机模型的参数，权重 $w \in \mathbb{R}^n$，θ 为阈值。假定 f 为阶跃函数，那么感知机模型的式可以表示为:

$$y = \varepsilon(\boldsymbol{w}^T \boldsymbol{x} - \theta) = \begin{cases} 1, & \boldsymbol{w}^T \boldsymbol{x} - \theta \geq 0; \\ 0, & \boldsymbol{w}^T \boldsymbol{x} - \theta < 0. \end{cases}$$

由于 n 维空间中的超平面方程为

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b = \boldsymbol{w}^T \boldsymbol{x} + b = 0$$

因此感知机模型式中的 $\boldsymbol{w}^T \boldsymbol{x} - \theta$ 可视为 n 维空间中的一个超平面，将 n 维空间划分为 $\boldsymbol{w}^T \boldsymbol{x} - \theta \geq 0$ 与 $\boldsymbol{w}^T \boldsymbol{x} - \theta < 0$ 的两个子空间（试想一下三维空间下的一个平面将空间切分为两部分）。那么落在前一个子空间的样本对应的模型输出值为 1，而后者为 0，如此实现了分类功能。

II 学习策略

给定一个线性可分的数据集 T，感知机的学习目标是求得能对数据集 T 中的正负样本完全正确划分的分离超平面

$$\boldsymbol{w}^T \boldsymbol{x} - \theta = 0$$

假设此时误分类样本集合为 $M \subset T$，对任意一个误分类样本 $(\boldsymbol{x}, y) \in M$ 而言，当 $\boldsymbol{w}^T \boldsymbol{x} - \theta \geq 0$ 时，模型输出值为 $\hat{y} = 1$，样本真实标记为 $y = 0$；反之亦然。综上，以下式恒成立:

$$(\hat{y} - y)(\boldsymbol{w}^T \boldsymbol{x} - \theta) \geq 0$$

因此对于给定数据集 T，其损失函数可以定义为

$$L(\boldsymbol{w},\theta)=\sum_{\boldsymbol{x}\in M}(\hat{y}-y)(\boldsymbol{w}^T\boldsymbol{x}-\theta)$$

非负之和显然非负，因此损失函数为非负。当没有误分类点时，损失函数的值为 0；误分类点越少，误分类点离超平面越近，损失函数值就越小。因此对于给定的数据集 T，损失函数 $L(\boldsymbol{w},\theta)$ 是关于 $\boldsymbol{w},\theta$ 的连续可导函数（注意是关于 $\boldsymbol{w},\theta$ 的可导，意味着之后将要对其进行梯度下降算法，即对其使用导数计算）。

连续:

$$\lim_{x\to x_0}f(x)=f(x_0)$$

可导:

$$f'(x_0)=\lim_{\Delta x\to 0}\frac{\Delta y}{\Delta x}=\lim_{\Delta x\to 0}\frac{f(x_0+\Delta x)-f(x_0)}{\Delta x}$$

III 学习算法

感知机模型的学习问题可以转化为求解损失函数的最优化问题。给定数据集

$$T=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_N,y_N),\}$$

其中 $\boldsymbol{x}_i\in\mathbb{R},y_i\in\{0,1\}$，求参数 $\boldsymbol{w},\theta$ 使得损失函数最小化:

$$\min_{\boldsymbol{w},\theta}L(\boldsymbol{w},\theta)=\min_{\boldsymbol{w},\theta}\sum_{\boldsymbol{x}_i\in M}(\hat{y}_i-y_i)(\boldsymbol{w}^T\boldsymbol{x}-\theta)$$

其中 $M\subset T$ 为误分类样本集合。若将阈值 θ 视为一个固定输入为 -1 的“哑结点”（前文有提到），即:

$$-\theta=-1\cdot w_{n+1}=x_{n+1}\cdot w_{n+1}$$

那么 $\boldsymbol{w}^T\boldsymbol{x}-\theta$ 可简化为

$$\begin{aligned}\boldsymbol{w}^T\boldsymbol{x}-\theta&=\sum_{j=1}^{n}w_jx_j+x_{n+1}\cdot w_{n+1}\\&=\sum_{j=1}^{n+1}w_jx_j\\&=\boldsymbol{w}^T\boldsymbol{x_i}\end{aligned}$$

其中 $\boldsymbol{x_i} \in \mathbb{R}^{n+1}$, $\boldsymbol{w} \in \mathbb{R}^{n+1}$，有此可将最小化问题进一步简化：

$$\min_{\boldsymbol{w}} L(\boldsymbol{w}) = \min_{\boldsymbol{w}} \sum_{\boldsymbol{x_i} \in M} (\hat{y}_i - y_i)\boldsymbol{w}^T \boldsymbol{x_i}$$

假设误分类样本集合 M 固定，那么可以求得损失函数 $L(\boldsymbol{w})$ 的梯度

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}) = \sum_{\boldsymbol{x_i} \in M} (\hat{y}_i - y_i)\boldsymbol{x_i}$$

感知机的学习算法具体采用的是**随机梯度下降法**，即在最小化的过程中，不是一次使 M 中所有误分类点的梯度下降，而是一次随机选取一个误分类点，并使其梯度下降。所以权重 $\boldsymbol{w}$ 的更新式为

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta \boldsymbol{w},$$
$$\Delta \boldsymbol{w} = -\eta(\hat{y}_i - y_i)\boldsymbol{w} = \eta(y_i - \hat{y}_i)\boldsymbol{w}$$

即 $\boldsymbol{w}$ 中的某个分量 w_i 的更新式即式 5.2。

备注：这里随机的意义在于每次调整的权重 w 不会对某个/些样本点产生依赖，即常说的“过拟合”。

补充

I **梯度下降法（gradient descent）**是一种一阶（first-order）优化方法，其被用于求解无约束优化问题。假设一个无约束优化 $min_{\boldsymbol{x}} f(\boldsymbol{x})$，其中 $f(\boldsymbol{x})$ 为**连续可微函数**。若能构成一个序列 $\boldsymbol{x}^0, \boldsymbol{x}^1, \boldsymbol{x}^2, \ldots$ 满足：

$$f(\boldsymbol{x}^{t+1}) < f(\boldsymbol{x}^t), \quad t = 0, 1, 2, \ldots$$

则不断执行该过程即可收敛到局部极小点（注：收敛级数 series converges）。为了满足以上式，根据泰勒展开有

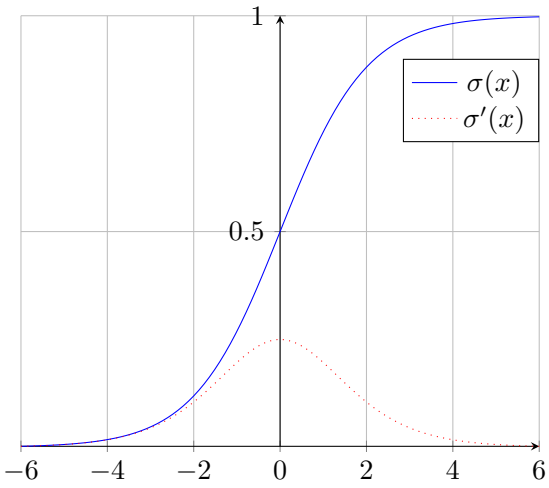
$$f(\boldsymbol{x} + \Delta \boldsymbol{x}) \simeq f(\boldsymbol{x}) + \Delta \boldsymbol{x}^T \nabla f(\boldsymbol{x})$$

因此，为了满足 $f(\boldsymbol{x} + \Delta \boldsymbol{x}) < f(\boldsymbol{x})$，可选择

$$\Delta \boldsymbol{x} = -\gamma \nabla f(\boldsymbol{x})$$

其中步长 γ 为一个小常数。这就是梯度下降法。

II Sigmoid 函数作为上述的连续可微函数（激活函数）



III ∇ 在数学中用于表示向量的微分计算。当 ∇ 应用在一维域的函数上时，即为标准的函数求导；当 ∇ 应用在一个场上（也就是多维域），它可能代表着如下三种含义中的一个：

i 梯度（gradient）：$f = \nabla f$

ii 向量的发散度（divergence）：$\operatorname{div} \overrightarrow{v} = \nabla \cdot \overrightarrow{v}$

iii 向量的旋度（curl）：$\operatorname{curl} \overrightarrow{v} = \nabla \times \overrightarrow{v}$

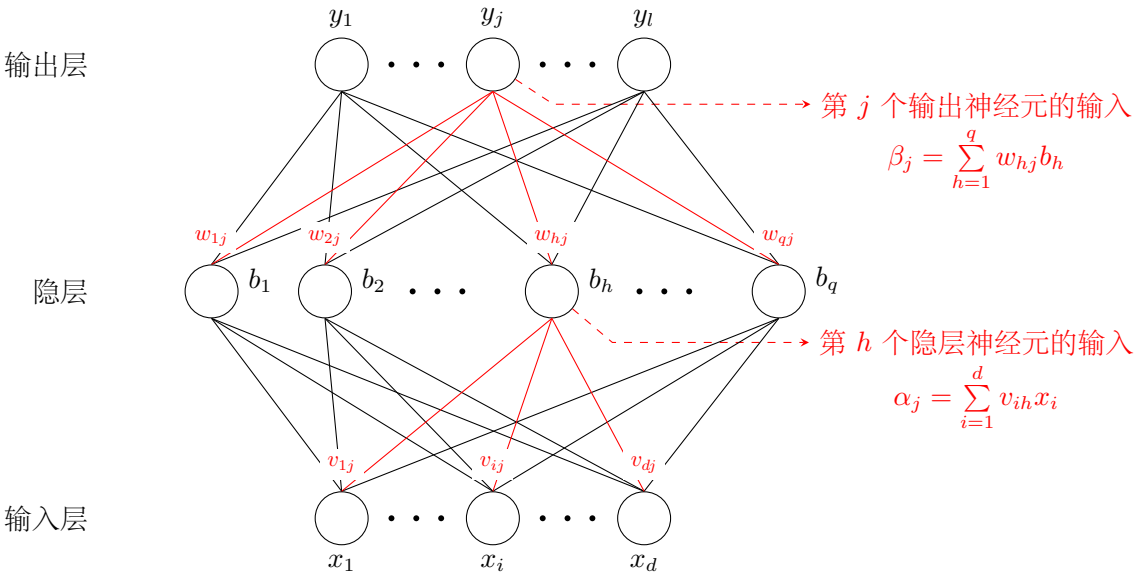
本章其余名词：

神经元模型	
名称	英文
功能神经元	functional neuron
线性可分	linearly separable
收敛	converge
震荡	fluctuation
隐层或隐含层	hidden layer
多层前馈神经网络	multi-layer feedforward neural networks
连接权	connection weight

5.3 误差反向传播算法

误差反向传播算法（error BackPropagation，简称 BP）。

给定训练集 $D = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \ldots, (\boldsymbol{x}_m, \boldsymbol{y}_m)\}$, $\boldsymbol{x}_i \in \mathbb{R}^d$, $\boldsymbol{y}_i \in \mathbb{R}^l$，即 input 为 d 维，output l 维。如图：



如图所示，该神经网络是一个拥有 d 个输入神经元，l 个输出神经元，q 个隐层神经元的多层前馈网络结构。

其中输出层第 j 个神经元的**阈值**用 θ_j 表示，隐层第 h 个神经元的**阈值**用 γ_h 表示；输入层第 i 个神经元与隐层第 h 个神经元之间的**连接权**为 v_{ih}，隐层第 h 个神经元与输出层第 j 个神经元之间的**连接权**为 w_{hj}。

其中隐层的第 h 个神经元接收到的**输入** 为 $\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$（图中输入层与隐层之间的三条红线所示）；输出层的第 j 个神经元接收到的**输入** 为 $\beta_j = \sum_{h=1}^{q} w_{hj} b_h$（图中隐层与输出层之间的四条红线所示），其中 b_h 为隐层第 h 个神经元的**输出**。

对训练例 $(\boldsymbol{x}_k, \boldsymbol{y}_k)$ 假设神经网络的输出为 $\hat{\boldsymbol{y}}_k = (\hat{y}_1^k, \hat{y}_2^k, \ldots, \hat{y}_l^k,)$，即

$$\hat{y}_j^k = f(\beta_j - \theta_j) \tag{5.3}$$

这里的真实值 $y_j^k = (y_1^k, y_2^k, \ldots, y_l^k)^2$，那么对于某第 i 个输出层的神经元而言，其均方误差即

$$\frac{(\hat{y}_i^k - y_i^k)}{2}$$

那么将所有神经元加总，就有了网络在 $(\boldsymbol{x}_k, \boldsymbol{y}_k)$ 上的均方误差为：

$$E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2 \tag{5.4}$$

另外上图的网络中有 $(d+l+l)q+l$ 个参数需确定：输入层到隐层的 $d \times q$ 个权值、隐层到输出层的 $q \times l$ 个权值、q 个隐层神经元的阈值、l 个输出层神经元的阈值。BP 是一个迭代学习算法，在迭代的每一轮中采用广义的感知机学习规则对参数进行更新估计，即与式 5.1 类似，任意参数 v 的更新估计式为

$$v \leftarrow v + \Delta v. \tag{5.5}$$

BP 算法基于 **梯度下降（gradient descent）** 策略，以目标的负梯度方向对参数进行调整。对式 5.4 的误差 E_k，给定学习率 η，有

$$\Delta w_{hj} = -\eta \frac{\partial E_k}{\partial w_{hj}} \tag{5.6}$$

这里提到的负梯度，是利用了导数趋近于零时可以得到最小值的特性，去求得均方误差 E_k 的最小值。回顾上文中的感知机学习算法：

> ... 随机选取一个误分类点并使其梯度下降，所以权重 $\boldsymbol{w}$ 的更新式为
>
> $$\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta\boldsymbol{w},$$
> $$\Delta\boldsymbol{w} = -\eta(\hat{y}_i - y_i)\boldsymbol{w} = \eta(y_i - \hat{y}_i)\boldsymbol{w}$$

式 5.6 与感知机的最小化损失函数的不同之处在于：感知机是有两层神经元构成的，且输出层只有一个神经元，因此只需要考虑一层的权重变化，即 $(\hat{y}_i - y_i)\boldsymbol{w}$；而对于由若干个感知机所构成的神经网络而言，所有权重的变化则变为了 $\frac{\partial E_k}{\partial w_{hj}}$。

那么对于多层的神经网络，式 5.4 仅表现出了最后一层隐层与输出层的权重变化，那么就有了疑问：其它层之间的权重好像并不会被式 5.6 所改变？我们接着往下看，注意到 w_hj 先影响到第 j 个输出层神经元的输入值 β_j。根据图示其为隐层所有神经元与其本身的表达式，即

$$\beta_j = \sum_{h=1}^{q} w_{hj} b_h$$

接着 w_{hj} 再影响到其他输出值 $\hat{y}_j^k$，最后再影响到 E_k，那么根据求导的链式法则有：

$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}} \tag{5.7}$$

根据 β_j 的定义，有：

$$\begin{aligned} \frac{\partial \beta_j}{\partial w_{hj}} &= \frac{\partial (\sum_{h=1}^{q} w_{hj} b_h)}{\partial w_{hj}} \\ &= \frac{\partial (w_{1j} b_1 + w_{2j} b_2 + \cdots + w_{hj} b_h + \cdots + w_{qj} b_q)}{\partial w_{hj}} \\ &= 0 + 0 + \cdots + b_h + \cdots + 0 \\ &= b_h \end{aligned} \tag{5.8}$$

而又根据 Sigmoid 函数的一个很好的性质：

$$f'(x) = f(x)(1 - f(x)) \tag{5.9}$$

根据式 5.4 与式 5.3 有：

$$\begin{aligned} g_j &= -\frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \\ &= -(\hat{y}_j^k - y_j^k) f'(\beta_j - \theta_j) \\ &= \hat{y}_j^k (1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k) \end{aligned} \tag{5.10}$$

将式 5.10 与式 5.8 代入式 5.7，再代入式 5.6，就得到了 BP 算法中关于 w_{hj} 的更新公式：

$$\begin{aligned} \Delta w_{hj} &= -\eta \frac{\partial E_k}{\partial w_{hj}} \\ &= -\eta \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}} \\ &= \eta \hat{y}_j^k (1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k) \cdot b_h \\ &= \eta g_j b_h \end{aligned} \tag{5.11}$$

类似可得：

$$\Delta \theta_j = -\eta g_j \tag{5.12}$$

$$\Delta v_{ih} = \eta e_h x_i \tag{5.13}$$

$$\Delta \gamma_h = -\eta e_h \tag{5.14}$$

式 5.13 与 5.14 中

$$
\begin{aligned}
e_n &= -\frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} \\
&= -\sum_{j=1}^{i} \frac{\partial E_k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} f'(\alpha_h - \gamma_h) \\
&= \sum_{j=1}^{l} w_{hj} g_j f'(\alpha_h - \gamma_h) \\
&= b_h(1 - b_h) \sum_{j=1}^{l} w_{hj} g_j
\end{aligned}
\tag{5.15}
$$

式 5.12 解析

因为:

$$
\Delta\theta_j = -\eta \frac{\partial E_k}{\partial \theta_j}
$$

同时有:

$$
\begin{aligned}
\frac{\partial E_k}{\partial \theta_j} &= \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \theta_j} \\
&= \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial [f(\beta_j - \theta_j)]}{\partial \theta_j} \\
&= \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot f'(\beta_j - \theta_j) \times (-1) \\
&= \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot f(\beta_j - \theta_j) \times [1 - f(\beta_j - \theta_j)] \times (-1) \\
&= \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \hat{y}_j^k (1 - \hat{y}_j^k) \times (-1) \\
&= \frac{\partial [\frac{1}{2} \sum_{j=1}^{l} (\hat{y}_j^k - y_j^k)^2]}{\partial \hat{y}_j^k} \cdot \hat{y}_j^k (1 - \hat{y}_j^k) \times (-1) \\
&= \frac{1}{2} \times 2(\hat{y}_j^k - y_j^k) \times 1 \cdot \hat{y}_j^k (1 - \hat{y}_j^k) \times (-1) \\
&= (\hat{y}_j^k - y_j^k) \hat{y}_j^k (1 - \hat{y}_j^k) \\
&= g_j
\end{aligned}
$$

所以:

$$\Delta\theta_j = -\eta g_j = -\eta \frac{\partial E_k}{\partial \theta_j}$$

式 5.13 解析

因为:

$$\Delta v_{ih} = -\eta \frac{\partial E_k}{\partial v_{ih}}$$

同时有:

$$\begin{aligned}
\frac{\partial E_k}{\partial v_{ih}} &= \sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} \cdot \frac{\partial \alpha_h}{\partial v_i h} \\
&= \sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} \cdot x_i \\
&= \sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} \cdot f'(\alpha_h - \gamma_h) \cdot x_i \\
&= \sum_{j=1}^{l} \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot w_{hj} \cdot f'(\alpha_h - \gamma_h) \cdot x_i \\
&= \sum_{j=1}^{l} (-g_j) \cdot w_{hj} \cdot f'(\alpha_h - \gamma_h) \cdot x_i \\
&= -f'(\alpha_h - \gamma_h) \cdot \sum_{j=1}^{l} (g_j) \cdot w_{hj} \cdot x_i \\
&= -b_h(1 - b_h) \cdot \sum_{j=1}^{l} (g_j) \cdot w_{hj} \cdot x_i \\
&= -e_h \cdot x_i
\end{aligned}$$

所以:

$$\Delta v_{ih} = -\eta \frac{\partial E_k}{\partial v_{ih}} = \eta e_h x_i$$

式 5.14 解析

因为:

$$\Delta\gamma_h = -\eta\frac{\partial E_k}{\partial\gamma_h}$$

同时有:

$$\begin{aligned}
\frac{\partial E_k}{\partial\gamma_h} &= \sum_{j=1}^{l}\frac{\partial E_k}{\partial\hat{y}_j^k}\cdot\frac{\partial\hat{y}_j^k}{\partial\beta_j}\cdot\frac{\partial\beta_j}{\partial b_h}\cdot\frac{\partial b_h}{\partial\gamma_h} \\
&= \sum_{j=1}^{l}\frac{\partial E_k}{\partial\hat{y}_j^k}\cdot\frac{\partial\hat{y}_j^k}{\partial\beta_j}\cdot\frac{\partial\beta_j}{\partial b_h}\cdot f'(\alpha_h-\gamma_h)\cdot(-1) \\
&= -\sum_{j=1}^{l}\frac{\partial E_k}{\partial\hat{y}_j^k}\cdot\frac{\partial\hat{y}_j^k}{\partial\beta_j}\cdot w_{hj}\cdot f'(\alpha_h-\gamma_h) \\
&= -\sum_{j=1}^{l}\frac{\partial E_k}{\partial\hat{y}_j^k}\cdot\frac{\partial\hat{y}_j^k}{\partial\beta_j}\cdot w_{hj}\cdot b_h(1-b_h) \\
&= \sum_{j=1}^{l} g_j\cdot w_{hj}\cdot b_h(1-b_h) \\
&= e_h
\end{aligned}$$

所以:

$$\Delta\gamma_h = -\eta\frac{\partial E_k}{\partial\gamma_h} = -\eta e_h$$

BP 算法的工作流程:

输入:

训练集 $D = \{(\boldsymbol{x}_k, \boldsymbol{y_k})\}_{k=1}^{m}$;

学习率 η.

过程:

1: 在 $(0, 1)$ 范围内随机初始化网络中所有连接权和阈值

2: **repeat**

3. **for all** $(\boldsymbol{x}_k, \boldsymbol{y}_k) \in D$ **do**

4. 根据当前参数和式 5.3 计算当前样本的输出 $\hat{\boldsymbol{y}}_k$;

$$\hat{y}_j^k = f(\beta_j - \theta_j)$$

5. 根据式 5.10 计算输出层神经元的梯度项 g_j;

$$g_j = \hat{y}_j^k(1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k)$$

6. 根据式 5.15 计算隐层神经元的梯度项 e_h;

$$e_h = b_h(1 - b_h)\sum_{j=1}^{l} w_{hj}g_j$$

7. 根据式 5.11 - 5.14 更新连接权 w_{hj}, v_{ih} 与阈值 θ_j, γ_h

$$\begin{cases} \Delta w_{hj} = \eta g_j b_h \\ \Delta\theta_j = -\eta g_j \\ \Delta v_{ih} = \eta e_h x_i \\ \Delta\gamma_h = -\eta e_h \end{cases}$$

8. **end for**

9. **until** 达到停止条件

输出: 连接权与阈值确定的多层前馈神经网络

需要注意，BP 算法的目标是要最小化训练集 D 上的累积误差（m 即训练集的样本个数）

$$E = \frac{1}{m}\sum_{k=1}^{m} E_k \tag{5.16}$$

但是上面介绍的“标准 BP 算法”每次仅针对一个训练样例更新连接权和阈值，意味着上述工作流程中的更新规则是基于单个 E_k 推导而得。如果类似推导出基于累积误差最小化的更新规则，那么就得到了**累积误差反向传播（accumulated error backpropagation）**算法。

那么来比较一下两种算法的差异：

BP 算法		
差异	标准 BP 算法	累积 BP 算法
参数更新	只针对单个样例	直接针对累积误差最小化
更新频率	非常频繁，且对不同样例进行更新可能会出现“抵消”	很低，因为是读取整个训练集后才对参数进行更新
迭代次数	多	少
类比	随机梯度下降	标准梯度下降

注：随机梯度下降（stochastic gradient descent）在训练集非常大时，累积误差下降到一定程度后，进一步下降会非常缓慢，这时切换为标准 BP 可以更快获得较好的解。

两种策略来缓解 BP 网络的过拟合：

1. **早停（early stopping）**：将数据分成训练集和验证集，训练集用来计算梯度、更新连接权和阈值，验证集用来估计误差，若训练集误差降低但验证集误差升高，则停止训练，同时返回具有最小验证集误差的连接权和阈值。

2. **正则化（regularization）**：在误差目标函数中增加一个用于描述网络复杂度的部分，例如连接权与阈值的平方和。仍然令 E_k 表示第 k 个训练样例上的误差，w_i 表示连接权和阈值，则误差目标函数 5.16 变为

$$E = \lambda\frac{1}{m}\sum_{k=1}^{m} E_k + (1-\lambda)\sum_{i} w_i^2 \tag{5.17}$$

   其中 $\lambda \in (0,1)$ 用于对经验误差与网络复杂度这两项进行折中，通常会使用交叉验证法来估计。

注：引入正则化策略的神经网络与第 6 章的 SVM 非常相似。

补充

BP 算法的计算流程。

- x 输入，即特征向量；
- y 目标输出。对于分类问题而言，输出是一个类型概率的向量（例 $(0.1, 0.7, 0.2)$），而目标输出是某个特定的类型，由 one-hot/dummy variable 编码（例 $(0, 1, 0)$）；
- C 损失函数或代价函数。对于分类问题而言，这通常是交叉熵 cross entropy（XC），而对回归问题而言则是平方错误损失 squared error loss（SEL）；
- L 神经网络的层数；
- $W^l = w^l_{jk}$ 神经网络中 $l-1$ 层与 l 层之间的权重，而 w^l_{jk} 则是在 $l-1$ 层的第 k 个节点与 l 层第 j 个节点之间的权重；
- f^l 神经网络中 l 层的激活函数。对于分类问题，二元分类的最后一层通常是对数函数，多种类分类则是 softmax 函数，而传统做法中，隐层的节点通常是 sigmoid 函数，今天更多样化了，线性整流函数 rectified 更为普遍（ramp，ReLU）。

整个网络实际上是一个关于**函数组合 function composition** 与**矩阵乘法 matrix multiplication** 的结合：

$$g(x) := f^L(W^L f^{L-1}(W^{L-1} \cdots f^1(W^1 x) \cdots))$$

对于训练集而言，输入-输出对 $\{(x_i, y_i)\}$ 视为一个集；而模型在每个输入-输出对 (x_i, y_i) 的损失实际上是预测值 $g(x_i)$ 与目标输出 y_i 的差异：

$$C(y_i, g(x_i))$$

注意这里的差异：在模型评估时，权重是固定的，而输入是变化的（且目标输出可能是不可知的），此时神经网络结束于输出层（它并不包含损失函数）；而在模型训练时，输入-输出对是固定的，而权重是变化的，此时神经网络结束于损失函数。

反向传递计算出固定的输入-输出对 (x_i, y_i) 的梯度，其中权重 w^l_{jk} 是可变的。每个独立的梯度 $\partial C / \partial w^l_{jk}$ 可以被链式计算；而分开计算每个权重是低效的。反向传递通过避免重复计算以及不去计算非必要的中间值，从而可以高效的计算出梯度，即计算每层的梯度 – 具体而言，是从后向前的，每层加权输入的梯度，以 δ^l 表示。

这里的关键点在于，在 W^l 中的权重影响损失是通过它在下一层的作用，又因为整个过程是线性的，δ^l 是唯一一个在 l 层中需要计算的数据，有此可以计算上一层 δ^{l-1} 并递归下去。这样就在两个层面避免了低效：首先，它避免了重复，因为当计算 l 层的梯度时，不需要每次都重新计算之后所有 $l+1, l+2$ 层的导数；其次，它避免了非必要的中间计算，因为每个阶段它直接计算了最终输出（损失）权重的梯度，而不是不必要的计算隐层中关于权重变化 $\partial a^{l'}_{j'}/\partial w^l_{jk}$ 数值的导数。

I 矩阵乘法

给定一个输入-输出对 (x, y)，其损失为：

$$C(y, f^L(W^L f^{L-1}(W^{L-1} \cdots f^2(W^2 f^1(W^1 x)) \cdots)))$$

计算损失首先从输入 x 开始并前向进行；将每个隐层的加权输入表示为 z^l，同时将隐层 l 的输出记为激活 a^l。对于反向传播，激活 a^l 以及导数 $(f^l)'$（在 z^l 计算）必须被缓存。

关于输入的损失导数由链式法则给出；注意每一项都是**全微分**，计算在神经网络（每个节点）的值：

$$\frac{dC}{da^L} \circ \frac{da^L}{dz^L} \cdot \frac{dz^L}{da^{L-1}} \circ \frac{da^{L-1}}{dz^{L-1}} \cdot \frac{dz^{L-1}}{da^{L-2}} \circ \cdots \circ \frac{da^1}{dz^1} \cdot \frac{\partial z^1}{\partial x}$$

注 *1* 函数 f 在某一点的全微分是指该函数在该点附近关于其自变量的最佳线性近似；与偏微分不同，全微分反映了函数关于其所有自变量的线性近似，而非单个自变量。

$$f(x) = \int_0^x f'(t)dt$$

注 *2* 对于组合函数 $g \circ f$ 在 a 的全微分，满足链式法则：

$$d(g \circ f)_a = dg_{f(a)} \cdot df_a$$

注 *3* 这里的 $\circ$ 为 Hadamard product，即元素一一相乘。

这些项分别为：损失函数的导数；激活函数的导数；以及矩阵的权重：

$$\frac{dC}{da^L} \circ (f^L)' \cdot W^L \circ (f^{L-1}) \circ \cdots \circ (f^l)' \cdot W^l$$

梯度 ∇ 是输出（由输入表示的）的导数转置，因此矩阵式转置的，且乘法的顺序是相反的，但是入口是一致的：

$$\nabla_x C = (W^{-1}) \cdot (f^1)' \circ \cdots \circ (W^{L-1})^T \cdot (f^{L-1})' \circ (W^L)^T \cdot (f^L)' \circ \nabla_{a^L} C$$

反向传播则是该表达式，从右向左计算所构成的（相当于从左到右乘上前一个表达式的导数），有此计算每一次的梯度；另外还有额外的一步加法，这是因为权重的梯度不仅仅是一个子表达式：还有一步额外的乘法。

对于部分乘积（从右至左）的辅助值 δ^l，表述为"l 层的错误"，且定义 l 层输入值的梯度：

$$\delta^l := (f^l)' \circ (W^{l+1})^T \circ \cdots \circ (W^{L-1})^T \cdot (f^{L-1})' \circ (W^L)^T \cdot (f^L)' \circ \nabla_{a^L} C$$

注意 δ^l 是一个向量，其长度是 l 层神经元的数量；每部分都被表述为"该节点的损失属性"

那么 l 层权重的梯度则为：

$$\nabla_{W^l} C = \delta^l (a^{l-1})^T$$

这里的 a^{l-1} 是因为在 $l-1$ 与 l 层直接的权重 W^l 按比例影响 l 层的输入（激活）：输入是固定的，权重是可变的。

那么 δ 就可以很容易的递归计算出来，从右至左：

$$\delta^{l-1} := (f^{l-1})' \circ (W^l)^T \cdot \delta^l$$

那么权重的梯度可以通过每层的矩阵乘法而计算出来；这便是反向传播。

II **导数**注：以下资料以另一个角度来解读式 5.11。

梯度下降方法牵涉到计算损失函数相对于网络权重的导数。这通常是通过反向传播算法完成的。假设一个输出神经元，有方差函数

$$E = L(t, y)$$

其中

- L 是输出 y 与目标值 t 的损失；
- t 是训练样本的目标输出；
- y 是输出神经元的实际输出。

对于每个神经元 j，其输出 O_j 被定义为

$$o_j = \varphi(\mathrm{net}_j) = \varphi\left(\sum_{k=1}^{n} w_{kj} x_k\right)$$

其中激活函数 φ 是非线性的，同时在激活区域是可微的（ReLU 在一个点上不可微）。经典的激活函数是对数函数：

$$\varphi = \frac{1}{1+e^{-x}}$$

其可以方便的进行求导：

$$\frac{d\varphi(z)}{dz} = \varphi(z)(1-\varphi(z))$$

这里一个神经元的输入 net_j 是前一层神经元们输出 o_k 加权之和。如果神经元在神经网络输入层之后的第一层，那么输入层的 o_k 即为神经网络的输入 x_k。神经元的输入单元数量是 n。变量 w_{kj} 表示前一层中神经元 k 与当前层神经元 j 之间的权重。

求误差的导数

计算误差相对于权重 w_{ij} 的偏微分是通过两次链式法则：

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net}_j}\frac{\partial \mathrm{net}_j}{\partial w_{ij}} \tag{I}$$

上述表达式右侧的最后一个元素，仅有一项在 net_j 之和中依赖 w_{ij}，因此有

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=1}^{n} w_{ki}o_k\right) = \frac{\partial}{\partial w_{ij}} w_{ij}o_i = o_i \tag{II}$$

如果神经元在输出层之后的一层，那么 o_i 就是 x_i。

神经元 j 的输出相对于其输入的倒数就是一个简单的激活函数导数：

$$\frac{\partial o_j}{\partial \mathrm{net}_j} = \frac{\varphi(\mathrm{net}_j)}{\partial \mathrm{net}_j} \tag{III}$$

而对于对数激活函数而言有

$$\frac{\partial o_j}{\partial \mathrm{net}_j} = \frac{\partial}{\partial \mathrm{net}_j}\varphi(\mathrm{net}_j) = \varphi(\mathrm{net}_j)(1-\varphi(\mathrm{net}_j)) = o_j(1-o_j)$$

这也是为什么反向传播需要激活函数是可微的。（尽管如此，ReLU 激活函数，在 0 处并不可微，如今更加普遍，例如在 AlexNet 中）。注：AlexNet，即 CNN 架构，卷积神经网络。

如果神经元位于输出层，那么计算很简单，因为 $o_j = y$：

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y} \tag{IV}$$

如果平方差用作于损失函数，那么可以重写为

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y} = \frac{\partial}{\partial y}\frac{1}{2}(1-y)^2 = y - t$$

然而，如果 j 是神经网络中隐层中的神经元，那么寻找 E 相对于 o_j 的导数就没那么明显了。将 E 视为一个由神经元 j 的得到的输入为全部神经元 $L = \{u, v, \ldots, w\}$ 的函数：

$$\frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(\mathrm{net}_u, \mathrm{net}_v, \ldots, \mathrm{net}_w)}{\partial o_j}$$

求相对于 o_j 的全微分，那么就得到了一个导数的递归表达式：

$$\frac{\partial E}{\partial o_j} = \sum_{t \in L} \left(\frac{\partial E}{\partial \mathrm{net}_t} \frac{\partial \mathrm{net}_t}{\partial o_j} \right) = \sum_{t \in L} \left(\frac{\partial E}{\partial o_t} \frac{\partial o_t}{\partial \mathrm{net}_t} \frac{\partial \mathrm{net}_t}{\partial o_j} \right) = \sum_{t \in L} \left(\frac{\partial E}{\partial o_t} \frac{\partial o_t}{\partial \mathrm{net}_t} w_{jl} \right) \tag{V}$$

因此，当相对下一层的输出 o_l 全部可微时，那么相对于 o_j 的微分则可被计算。[注意，如果 L 中任何神经元没有连接至神经元 j，它们则会独立于 w_{ij}，同时与和相关的偏微分则会消失为 0]。

将 (II)，(III)，(IV)，(V) 代入进 (I) 中，获得：

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \mathrm{net}_j} \frac{\partial \mathrm{net}_j}{\partial w_{ij}} \\ &= \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \mathrm{net}_j} o_i \\ &= o_i \delta_j \end{aligned}$$

这其中

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \mathrm{net}_j} = \begin{cases} \frac{\partial L(o_j, t)}{\partial o_j} \frac{d\varphi(\mathrm{net}_j)}{d\mathrm{net}_j} & \text{如果 } j \text{ 是一个输出层的神经元，} \\ (\sum_{l \in L} w_{jl} \delta_j) \frac{d\varphi(\mathrm{net}_j)}{d\mathrm{net}_j} & \text{如果 } j \text{ 是一个隐层的神经元。} \end{cases}$$

如果 φ 是一个对数函数，那么误差则为平方差：

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \mathrm{net}_j} = \begin{cases} (o_j - t_j) o_j (1 - o_j) & \text{如果 } j \text{ 是一个输出层的神经元} \\ (\sum_{l \in L} w_{jl} \delta_j) o_j (1 - o_j) & \text{如果 } j \text{ 是一个隐层的神经元} \end{cases}$$

通过梯度下降法来更新权重 w_{ij}，那么就必须选择一个学习率，$\eta > 0$。根据 w_{ij} 的增减，权重的变化必须反应在 E 上。如果 $\frac{\partial E}{\partial w_{ij}} > 0$，那么 w_{ij} 增加则 E 增加；相反的，如果 $\frac{\partial E}{\partial w_{ij}} < 0$，那么 w_{ij} 增加则 E 减少。新的 Δw_{ij} 被添加到就权重上，且学习率与梯度之积再乘上 -1 则能保证 w_{ij} 的变化总是能减少 E。换言之，下述等式 $-\eta \frac{\partial E}{\partial w_{ij}}$ 总是能在 w_{ij} 改变时令 E 减少：

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta o_i \delta_j$$

III 损失函数

损失函数是一个函数，它将一个或多个变量的值映射到一个实数上，直观地表示与这些值相关的一些“成本”。对于反向传播而言，在一个训练样本通过正向传播后，损失函数的计算了神经网络输出与期望输出之间的差异。

损失函数的数学表达式必须满足两个条件，才有可能用于反向传播。首先，对于 n 个独立的训练模型 x，可以被写成一个由误差函数 E_x 所构成的均值 $E = \frac{1}{n}\sum_x E_x$。该假设的原因是，对于单个训练样本，反向传播算法需要计算误差函数的梯度，再被泛化到整个误差函数；其次，它可以写成神经网络输出的函数。

5.4 全局最小与局部最小

WIP

6 支持向量机

7 贝叶斯分类器

8 集成学习

9 聚类

10 降维与度量学习

11 特征选择与稀疏学习

12 计算学习理论

13 半监督学习

14 概率图模型

15 规则学习

16 强化学习