

分类

- 监督学习 (Supervised learning) : 回归、分类
- 无监督学习 (Unsupervised learning) : 聚类、异常检测、降维
- 强化学习 (Reinforcement learning)

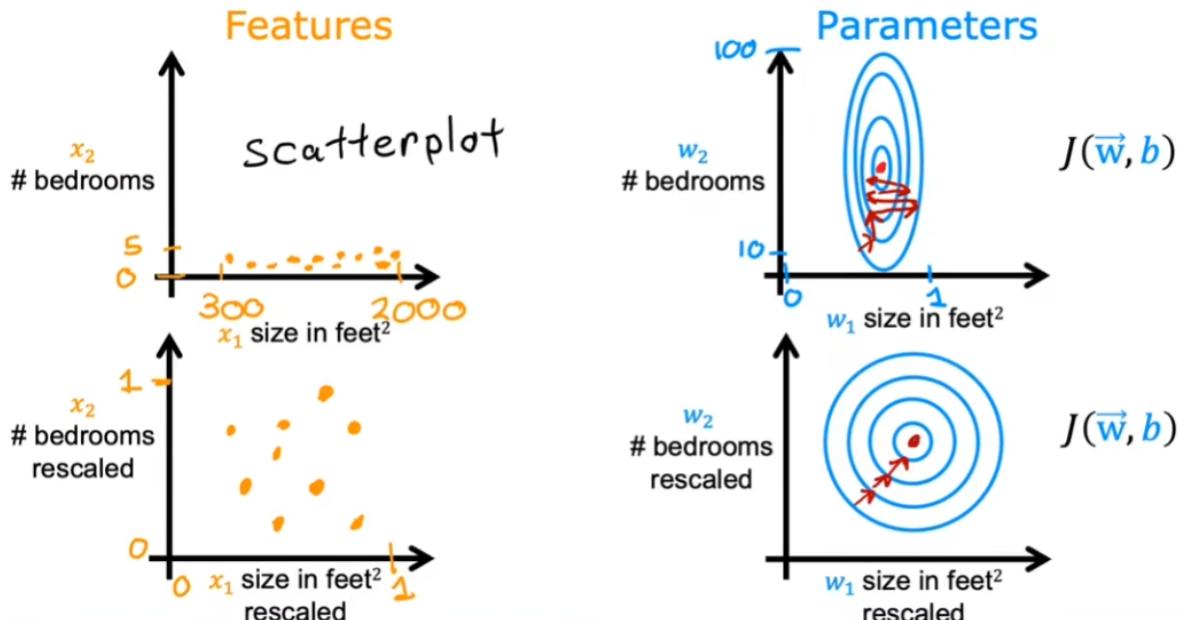
基础概念

过拟合欠拟合

- 过拟合: 指训练误差和测试误差之间的差距太大。模型在训练集上表现很好，但在测试集上却表现很差。模型对训练集“死记硬背”，没有理解数据背后的规律，泛化能力差。
- 欠拟合: 指模型不能在训练集上获得足够低的误差。模型复杂度低，模型在训练集上就表现很差，没法学到数据背后的规律。
- 解决方法:
 - 收集更多的数据
 - 减少使用特征
 - 正则化: 减少参数的大小

特征值缩放

特征值的大小会显著的影响到各自w系数的大小，为了提高机器学习的性能，避免特征值过大或过小，需要对特征值进行缩放。



名词解释：可以认为特征缩放包括归一化和标准化。

特征缩放

Feature Scaling

$$x_{i,scaled} = \frac{x_i}{x_{max}}$$

归一化

Normalization

特征值的均值 : μ_i

$$x_{i,scaled} = \frac{x_i - \mu_i}{x_{max} - x_{min}}$$

标准化

Standardization (Z-Score Normalization)

特征值的标准差 : σ_i

特征值的平均值 : \bar{x}_i

$$x_{i,scaled} = \frac{x_i - \bar{x}_i}{\sigma_i}$$

激活函数

线性函数

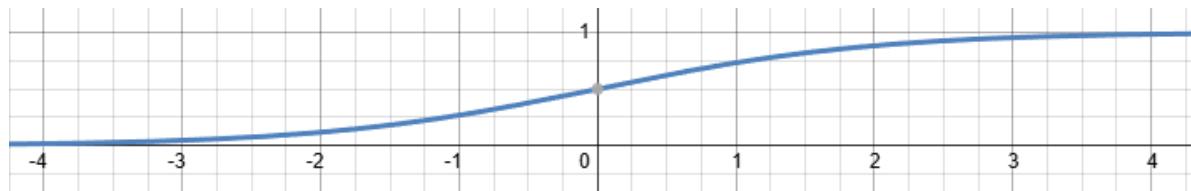
在回归任务中预测正负

$$g(x) = x$$

Sigmoid函数

多用于二分类问题

$$g(x) = \frac{1}{1 + e^{-x}}$$
$$0 < g(x) < 1$$



Softmax

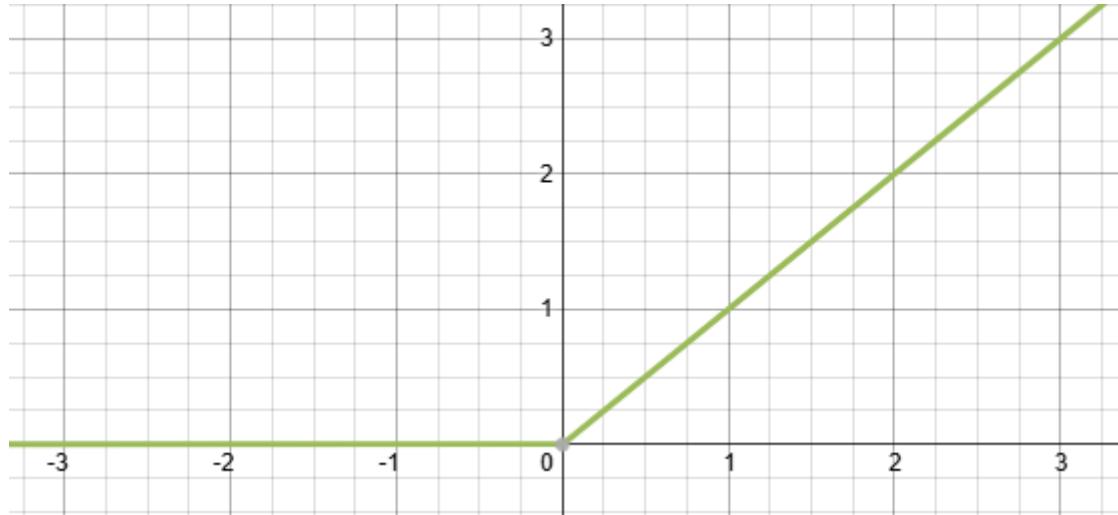
用于多类分类问题，一般只用在输出层

$$g(x) = \frac{e^{x_i}}{\sum_i e^{x_i}}$$

线性整流函数ReLU

在回归任务中 $g(x)$ 只有负值

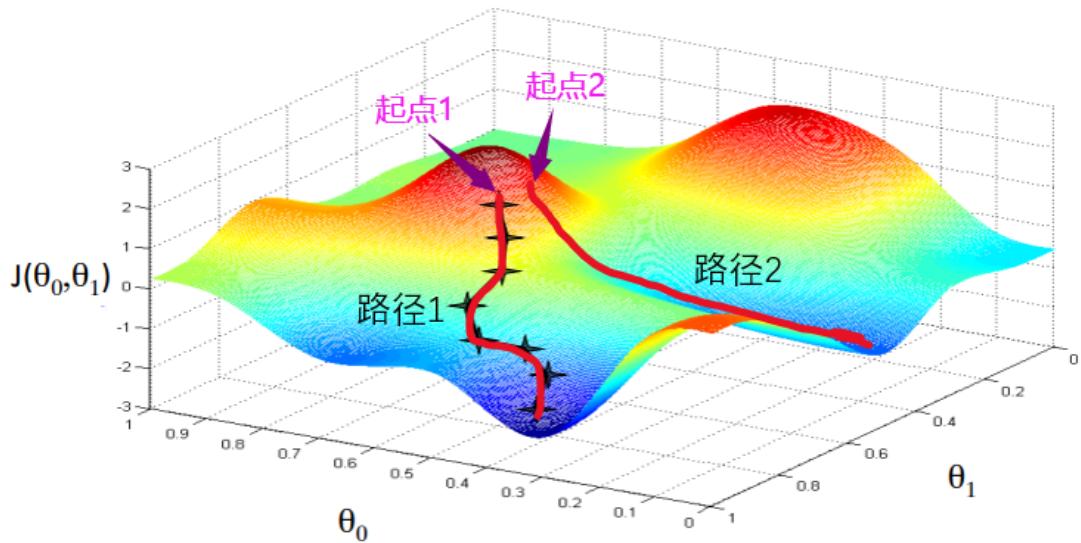
$$g(z) = \max(0, z)$$



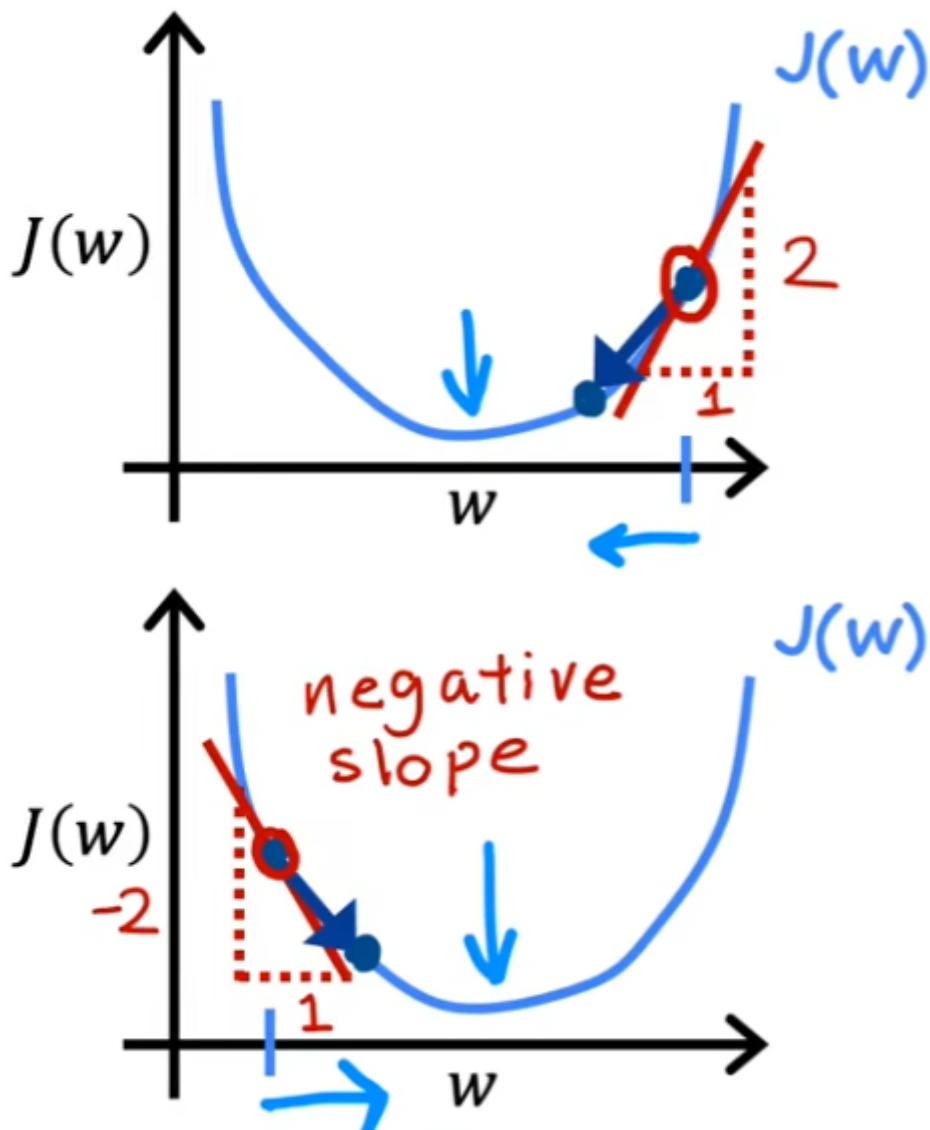
优化算法

梯度下降

梯度下降算法是用来快速确定函数中的最小值。



当起始点再最小值的右边， 斜率为正计算出的偏导数也是正数，更新后的参数会后移靠近最小值。反之同理。



Adam

根据梯度下降的情况动态的调整学习率一更快的达到最优状态。当下降方向保持不变时增大学习率快速下降，当下降方向来回摆动时减小学习率逼近最优点。

模型评估

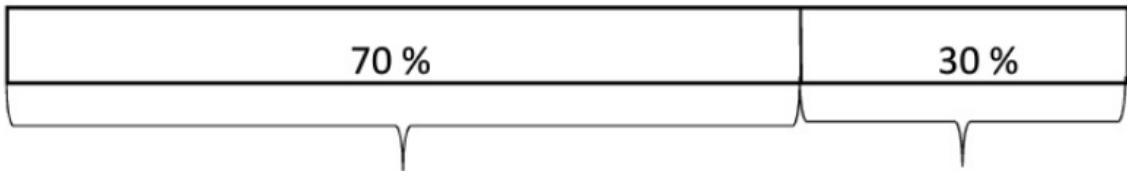
随着特征值的变多，绘制代价函数的困难程度也会增加（例如如何绘制四维图像），所以需要更系统的方式来评估模型。

将数据集分为三个部分：

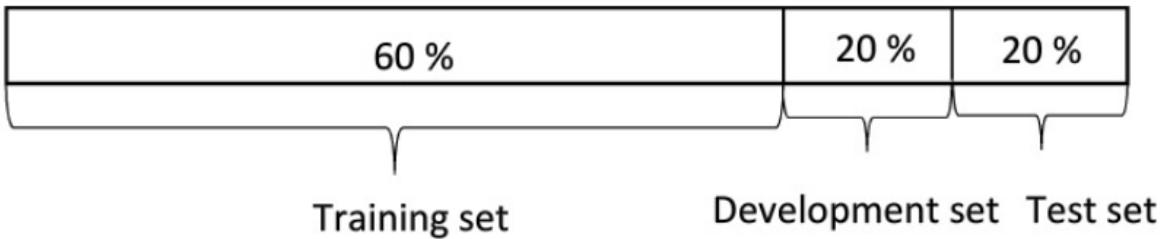
- **训练集** (train set)：用于训练模型
- **交叉验证集** (cross validation set)：用于评估模型效果
- **测试集** (test set)：用于测试模型泛化能力

留出法

在数据集中流出一部分用于测试集



Or



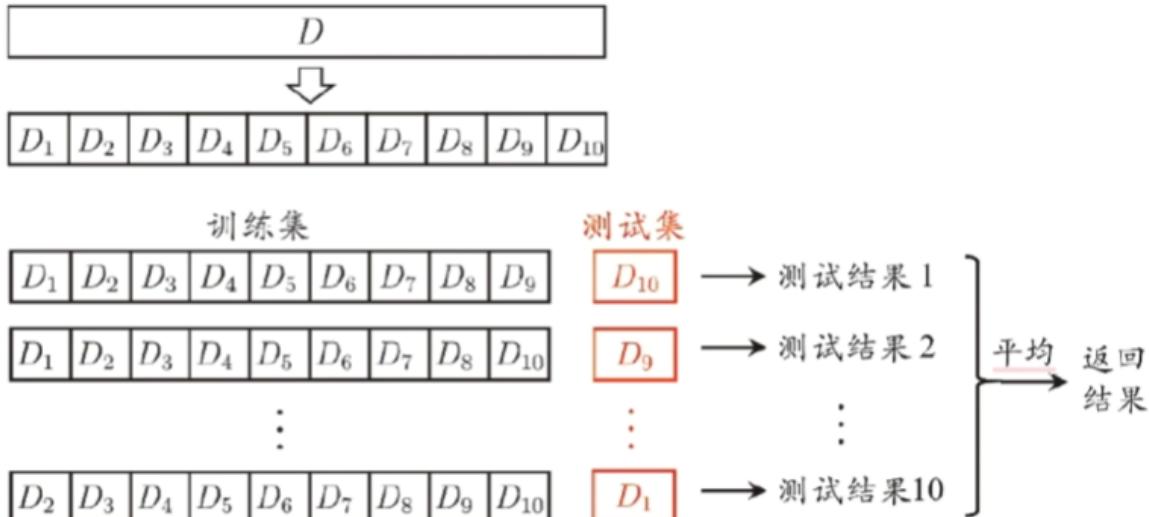
- 保持数据分布一致性（例如：分层采样）
- 多次重复划分（例如：100次随机划分）
- 测试集不能太大、不能太小（例如：1/5~1/3）
- 通过训练集训练的数据经测试集测试后，应将所有数据一起再训练得到最终模型

$$J_{test}(\vec{w}, b) = \frac{1}{2m_{test}} \left[\sum_{i=1}^{m_{test}} (\vec{f}_{\vec{w}, b}(\vec{x}_{test}^{(i)}) - \vec{y}_{test}^{(i)})^2 \right]$$

$$J_{cv}(\vec{w}, b) = \frac{1}{2m_{cv}} \left[\sum_{i=1}^{m_{cv}} (\vec{f}_{\vec{w}, b}(\vec{x}_{cv}^{(i)}) - \vec{y}_{cv}^{(i)})^2 \right]$$

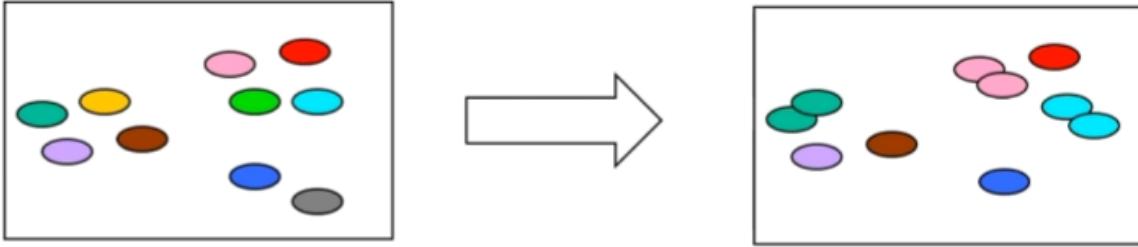
$$J_{train}(\vec{w}, b) = \frac{1}{2m_{train}} \left[\sum_{i=1}^{m_{train}} (\vec{f}_{\vec{w}, b}(\vec{x}_{train}^{(i)}) - \vec{y}_{train}^{(i)})^2 \right]$$

k-折交叉验证法



自助法

对于含有 m 个样本的数据集 D ，我们对它进行有放回的采样 m 次，最终得到一个含有 m 个样本的数据集 D' ，这个数据集 D' 会有重复的数据（约36.8%的数据不会出现），把它用作训练数据。

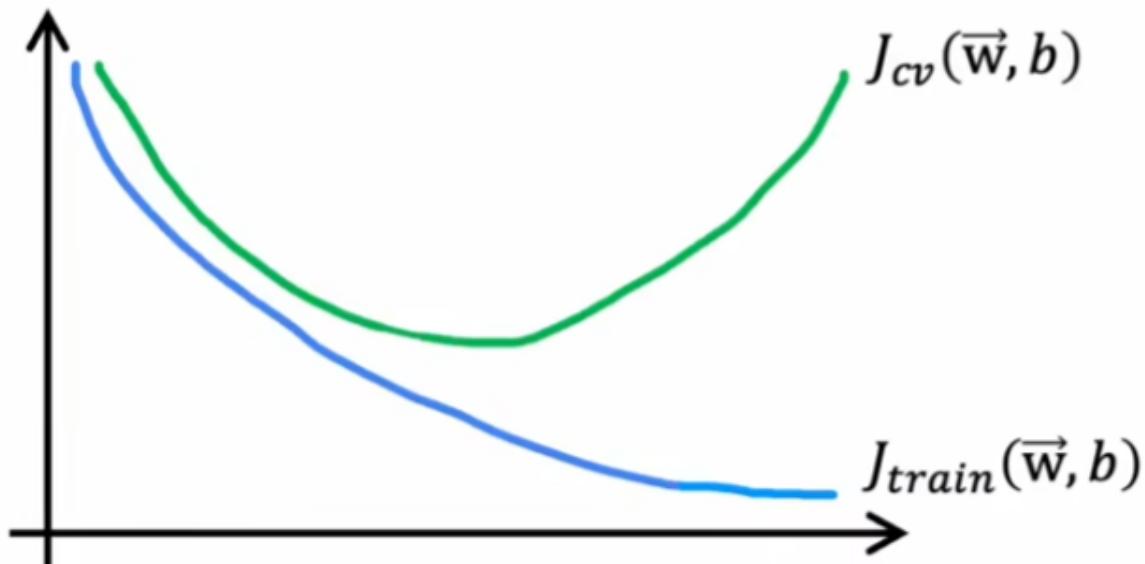


模型诊断

维度对代价的影响

$$\begin{aligned}f_{\vec{w}, b}(x) &= w_1 x + b \\f_{\vec{w}, b}(x) &= w_1 x + w_2 x^2 + b \\f_{\vec{w}, b}(x) &= w_1 x + w_2 x^2 + w_3 x^3 + b \\\dots \\f_{\vec{w}, b}(x) f_{\vec{w}, b}(x) &= w_1 x + w_2 x^2 + \dots + w_n x^n + b\end{aligned}$$

横轴为模型的特征维度，纵轴为模型代价：



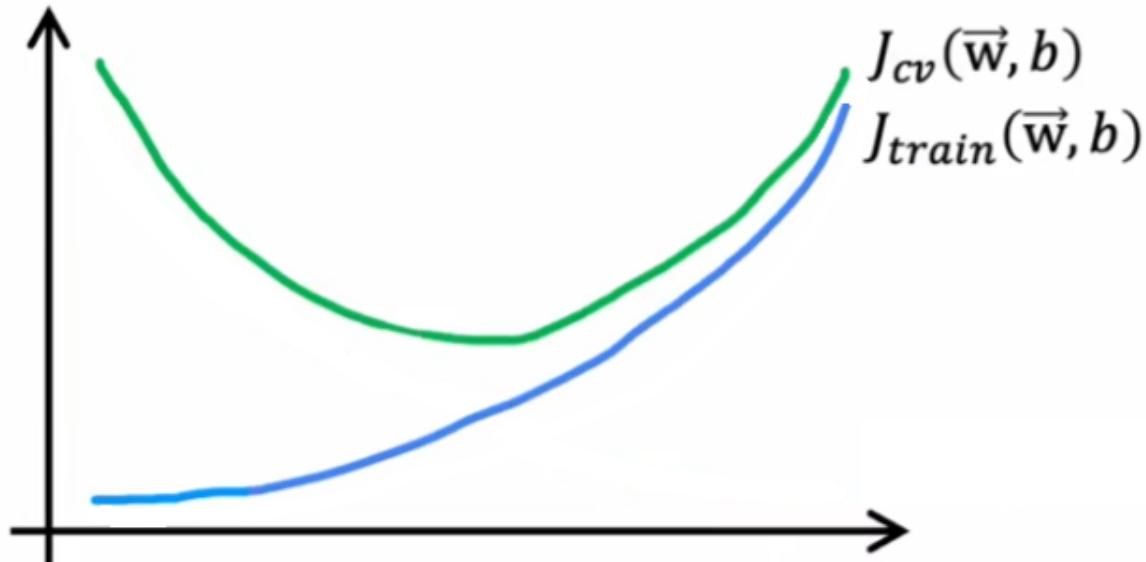
- 模型欠拟合时（图像左边部分），模型在训练集上表现不好 J_{train} 很大，同时在交叉验证集上表现也不好 J_{cv} 很大， J_{cv} 和 J_{train} 相似，存在高偏差。
- 模型过拟合时（图像右边部分），模型在训练集上表现很好 J_{train} 很小，但在交叉验证集上表现也不好 J_{cv} 很大， J_{cv} 远大于 J_{train} ，存在高方差。
- 模型同时存在高偏差和高方差（更多会出现在神经网络模型上）， J_{train} 很大， J_{cv} 远大于 J_{train}

正则化对代价的影响

$$f_{\vec{w}, b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^5 + b$$

$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}(i)) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^m w_j^2$$

横轴为lambda的取值，纵轴为模型代价：

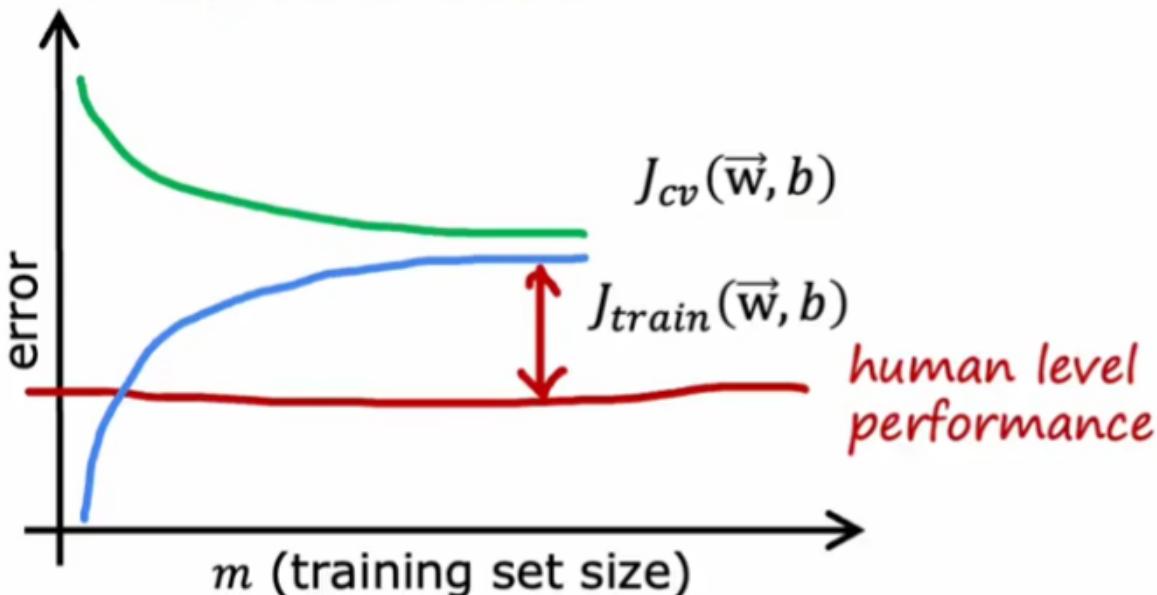


- 当lambda取值很小时，模型过拟合，存在高方差。
- 当lambda取值很大时，w接近于0，模型接近与常函数，模型欠拟合，存在高偏差。

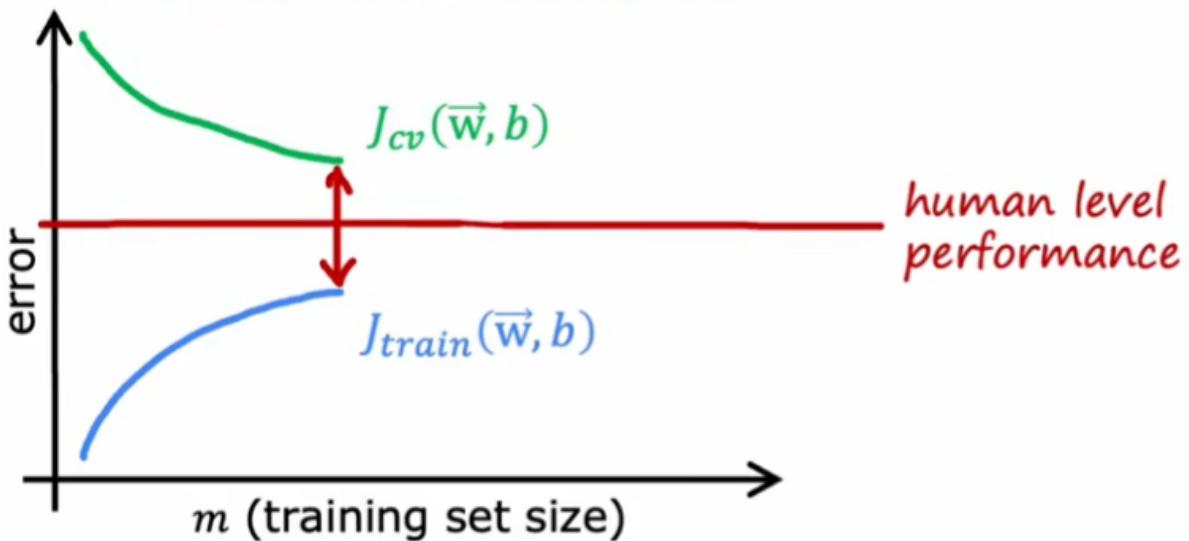
学习曲线

引入人类对这一任务的表现水平，通过对比人类水平 (Baseline performance)、训练错误 (Training error)、交叉错误 (Cross validation error) 三者来判别是否存在高偏差或高方差。

当Bp与Jtrain相差很大，而Jtrain和Jcv相差很小时，就时高偏差

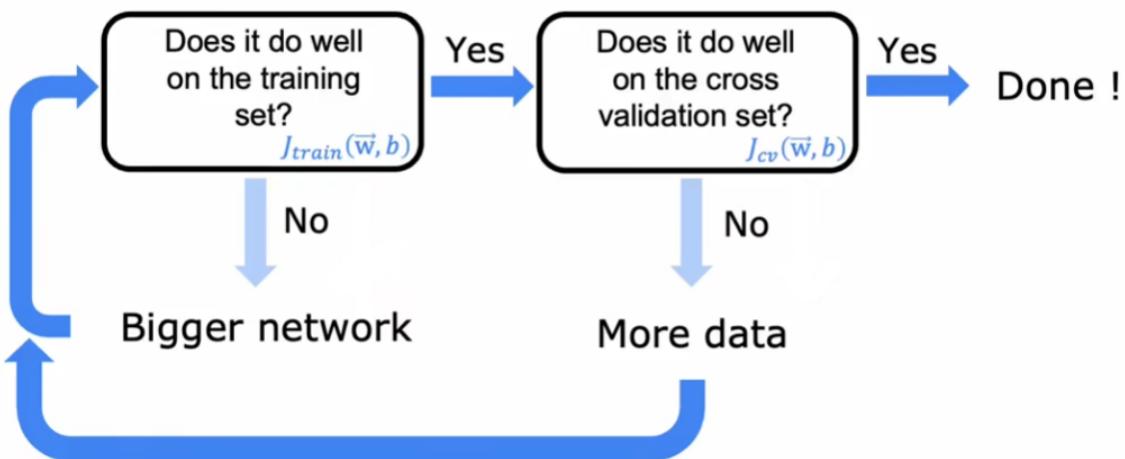


当Bp与Jtrain相差很小，而Jtrain和Jcv相差很大时，就时高方差



- 如何解决高偏差
 - 增加新的特征
 - 增加特征的维度
 - 减少lambda
- 如何解决高方差
 - 引入更多的训练集
 - 减少特征的维度
 - 增加lambda

使用简单的模型会导致高偏差，使用复杂的模型会导致高方差。这就是为什么神经网络流行的原因，只要有足够的大的神经网络和足够多的数据，神经网络总是能很好的拟合这些数据。



精确率&召回率

在分类问题中，将真实分类和预测分类的结果分为四类，由此分析精确率和召回率。其中精确率是评估分类准不准的，召回率是评估全不全的。通过F1得分可以总体权衡模型的精确率和召回率。

		Actual Class	
		1	0
Predict -ed Class	1	True positive 15	False positive 5
	0	False negative 10	True negative 70

$$\text{精确率 } P = \frac{\text{TruePositives}}{\text{TruePos} + \text{FalsePos}}$$

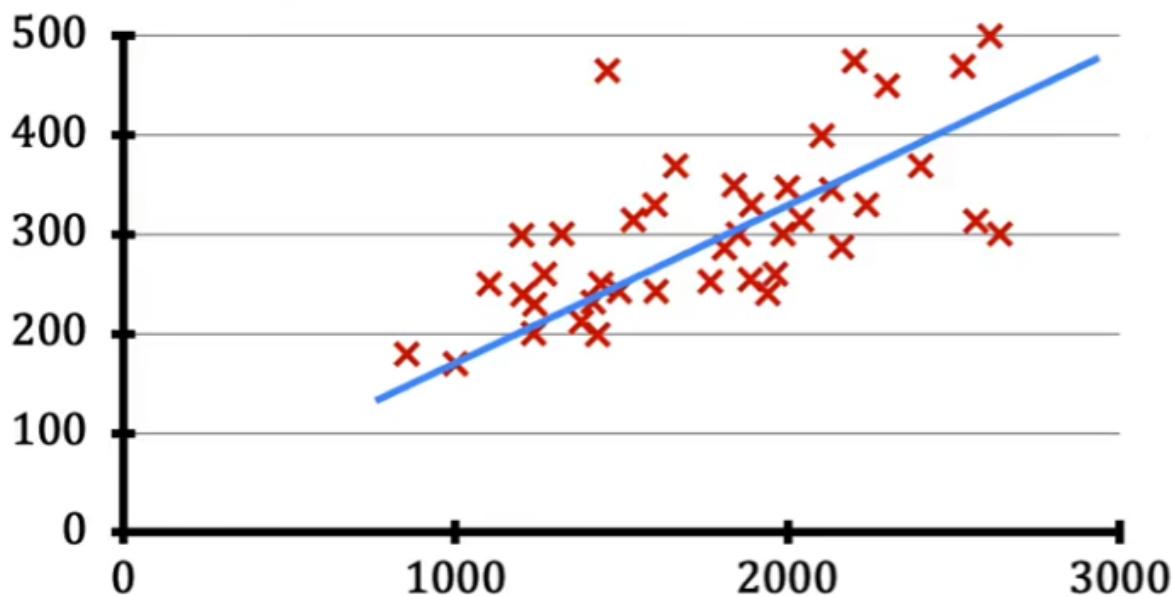
$$\text{召回率 } R = \frac{\text{TruePositives}}{\text{TruePos} + \text{FalseNeg}}$$

$$F1\ score = \frac{1}{\frac{1}{2}(\frac{1}{P} + \frac{1}{R})} = 2 \frac{PR}{P + R}$$

线性回归

目的

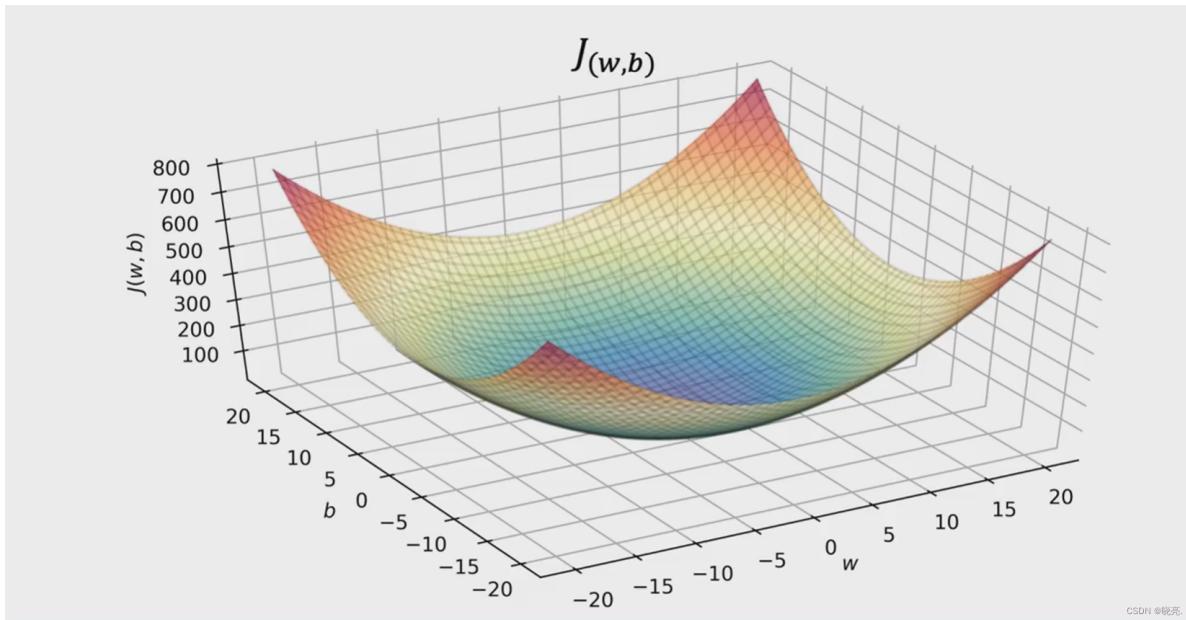
通过数据计算出一条尽可能符合数据的函数



代价函数

使用代价函数来评判当前函数的拟合程度，代价越低拟合程度越好。（一般使用二维图像和等高线表示）

机器学习的目的就是找到代价函数的最低点，即最小值。



CSDN @明亮

单特征值公式

函数模型：其中x和y来源于数据，w和b是待求解的参数

$$f_{w,b}(x) = wx + b$$

代价函数：用于评估w和b的代价，越低拟合程度越好

$$J(w, b) = \frac{1}{2m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})^2$$

梯度下降算法：用于快速的计算最优的w和b（其中的alpha就是学习率）

$$w = w - \alpha \frac{d}{dw} J(w, b)$$

$$b = b - \alpha \frac{d}{db} J(w, b)$$

将代价函数带入梯度下降算法并计算偏导数后的表达式为：

$$w = w - \alpha \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)}) x^{(i)}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})$$

求偏导过程如下：

$$\begin{aligned}
\frac{d}{dw} J(w, b) &= \frac{d}{dw} \frac{1}{2m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})^2 \\
&= \frac{d}{dw} \frac{1}{2m} \sum_{i=1}^m (wx^{(i)} + b - y^{(i)})^2 \\
&= \frac{1}{2m} \sum_{i=1}^m (wx^{(i)} + b - y^{(i)}) 2x^{(i)} \\
&= \frac{1}{m} \sum_{i=1}^m (wx^{(i)} + b - y^{(i)}) x^{(i)} \\
\frac{d}{db} J(w, b) &= \frac{d}{db} \frac{1}{2m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})^2 \\
&= \frac{d}{db} \frac{1}{2m} \sum_{i=1}^m (wx^{(i)} + b - y^{(i)})^2 \\
&= \frac{1}{2m} \sum_{i=1}^m (wx^{(i)} + b - y^{(i)}) 2 \\
&= \frac{1}{m} \sum_{i=1}^m (wx^{(i)} + b - y^{(i)})
\end{aligned}$$

多特征值公式

函数模型：

$$\begin{aligned}
f_{\vec{w},b}(\vec{x}) &= w_1 x_1 + \dots + w_n x_n + b \\
\vec{w} &= [w_1 \dots w_n] \\
\vec{x} &= [x_1 \dots x_n] \\
f_{\vec{w},b}(\vec{x}) &= \vec{w} \cdot \vec{x} + b
\end{aligned}$$

代价函数：

$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2$$

梯度下降算法：

$$\begin{aligned}
w_j &= w_j - \alpha \frac{d}{dw_j} J(\vec{w}, b) \\
b &= b - \alpha \frac{d}{db} J(\vec{w}, b)
\end{aligned}$$

将代价函数带入梯度下降算法并计算偏导数后的表达式为：

$$\begin{aligned}
w_n &= w_n - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_n^{(i)} \\
b &= b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})
\end{aligned}$$

设置学习率

对于学习率（0~1之间），若太小每次更新的距离太短导致迭代时间太长，若太大则有可能直接跨过最低点到另一边从而无法收敛。

通过设置不同的学习率，并绘制部分迭代过程中的代价值图像，通过观察图像再选择一条最大的可收敛的曲线作为最后的学习率

如：0.001 0.003 0.01 0.03 0.1 0.3 1

多项式回归

如果要拟合曲线则需要使用多项式的函数模型，如：

$$f_{\vec{w}, b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + \dots + b$$
$$f_{\vec{w}, b}(x) = w_1 x + w_2 \sqrt{x} + b$$

正则化

代价函数：

$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$$

梯度下降算法：

$$w_j = w_j - \alpha \left\{ \frac{1}{m} \sum_{i=1}^m [(f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)}] + \frac{\lambda}{m} w_j \right\}$$
$$b = b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})$$

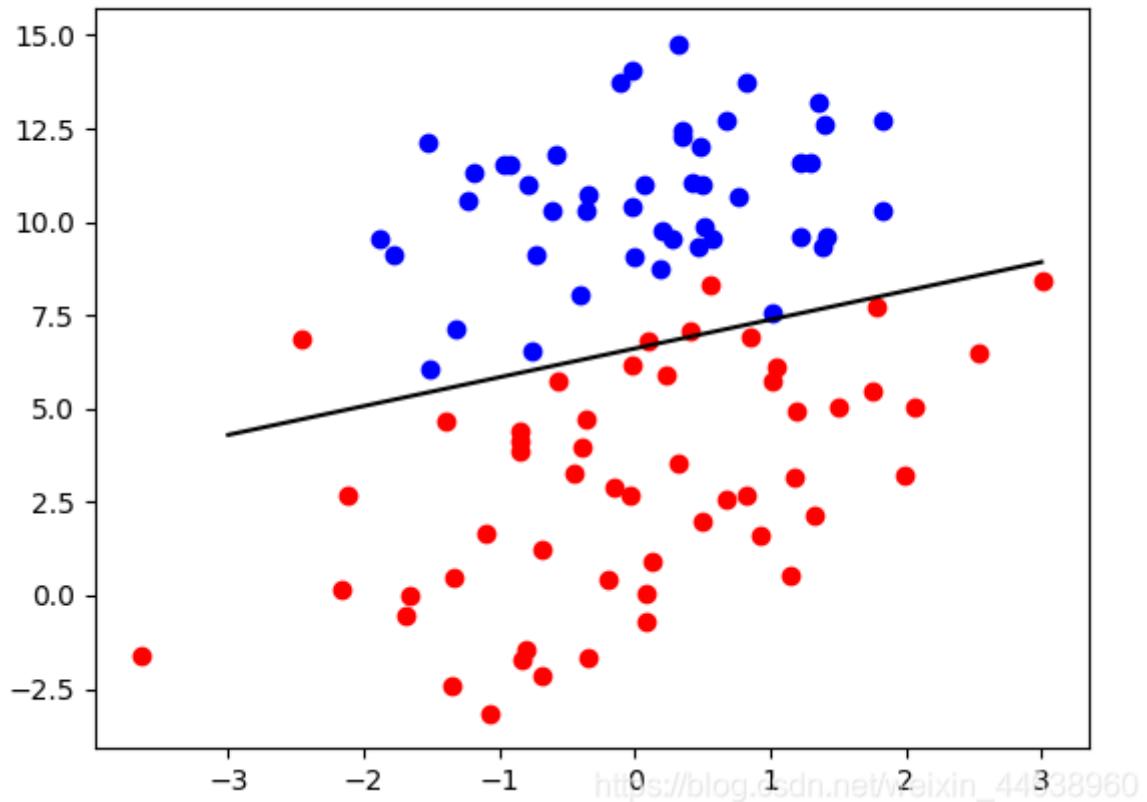
w项化简：

$$w_j = w_j \left(1 - \alpha \frac{\lambda}{m}\right) - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

逻辑回归

目的

不同与线性回归输出的值没有固定的范围，逻辑回归的输出有固定的几个值。如果使用函数输出固定值就需要用激活函数。通过函数尽量将不同的数据集划分开。



逻辑回归模型

这里的 $g()$ 表示Sigmoid函数

$$z = \vec{w} \cdot \vec{x} + b$$

$$f_{\vec{w}, b}(\vec{x}) = g(\vec{w} \cdot \vec{x} + b) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

通常会设置一个值(threshold)，当逻辑函数的结果大于这个值则输出1，反之则输出0

$$\text{is } f_{\vec{w}, b}(\vec{x}) \geq \text{threshold?}$$

$$\text{Yes : } y = 1$$

$$\text{No : } y = 0$$

$$\text{When is } f_{\vec{w}, b}(\vec{x}) \geq \text{threshold?}$$

$$g(z) \geq \text{threshold}$$

$$z \geq \text{threshold}$$

$$\vec{w} \cdot \vec{x} + b \geq \text{threshold}$$

$$y = 1$$

$$\vec{w} \cdot \vec{x} + b < \text{threshold}$$

$$y = 0$$

其中 z 的表达式就是边界函数，需要根据不同的情况进行调整。

逻辑回归公式

函数模型：

$$f_{\vec{w}, b}(\vec{x}) = g(\vec{w} \cdot \vec{x} + b) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

代价函数：

$$J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^m L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)})$$
$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} x = -\log(f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ y = -\log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$
$$J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))]$$

梯度下降算法：

$$w_j = w_j - \alpha \frac{d}{dw_j} J(\vec{w}, b)$$
$$b = b - \alpha \frac{d}{db} J(\vec{w}, b)$$

将代价函数带入梯度下降算法并计算偏导数后的表达式为：

$$w_n = w_n - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_n^{(i)}$$
$$b = b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})$$

正则化

代价函数：

$$J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))] + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$$

梯度下降算法：

$$w_j = w_j - \alpha \left\{ \frac{1}{m} \sum_{i=1}^m [(f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)}] + \frac{\lambda}{m} w_j \right\}$$
$$b = b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})$$

决策树

分类树

越靠近根节点越在意数据的一般性区别，越靠近叶节点越在意数据细节。

衡量标准-熵

熵是表示随机变量不确定性的度量（即物体内部的混乱程度）

$Ent(D)$ 表示还需要多少信息量才能把数据集D划分干净

$$Ent(D) = - \sum_{k=1}^{|y|} p_k \log_2 p_k$$
$$0 \leq Ent(D) \leq \log_2 |y|$$

- D表示数据集
- $|y|$ 表示类别数量
- p_k 表示每种类型出现的概率

属性划分准则

通过一些原则来判断前后两次划分后数据是否变纯净。

信息增益与基尼指数产生的结果，仅在约2%的情况下不同。

信息增益ID3

以属性a对数据集D进行划分所获得的信息增益

$$Gain(D, a) = Ent(D) - \sum_{v=1}^V \frac{|D^v|}{|D|} Ent(D^v)$$

- D^v 表示D中在a上取值= a^v 的样本集合

缺点：在面对ID这种唯一性的数据，信息增益会选择他作为跟节点划分，使得决策树丧失泛化性。

信息增益率C4.5

$$Gain_ratio(D, a) = \frac{Gain(D, a)}{IV(a)}$$
$$IV(a) = - \sum_{v=1}^V \frac{|D^v|}{|D|} \log_2 \frac{|D^v|}{|D|}$$

- 分支越多， $IV(a)$ 越大
- 划分效果越好， $Gain(D, a)$ 越大
- $Gain_ratio(D, a)$ 要求 $Gain(D, a)$ 越大的同时 $IV(a)$ 越小，即划分效果好的情况下分支还少

启发式（信息增益和分支的取舍）：先从候选划分属性中找出信息增益高于平均水平的，再从中选取增益率最高的

CART

基尼指数

$$Gini(D) = \sum_{k=1}^{|y|} \sum_{k' \neq k} = 1 - \sum_{k=1}^{|y|} p_k^2$$

$$Gini_index(D, a) = \sum_{v=1}^V \frac{|D^v|}{|D|} Gini(D^v)$$

- Gini(D)越小，数据集D的纯度越高

剪枝

在数据带噪时甚至可能将泛化性能提升25%

- 预剪枝 (pre-pruning)：通过限制深度，叶子节点个数，叶子节点样本数，信息增益量等，提前终止某些分支的生长。
- 后剪枝 (post-pruning)：生成一颗完全树，再进行剪枝。

$$C_a(T) = C(T) + \alpha |T_{leaf}|$$

- $C_a(T)$ 表示损失
- $C(T)$ 表示基尼指数
- α 表示平衡项
- T_{leaf} 表示叶子节点个数

回归树

不同于分类树在结点划分时考察分类后的信息熵，回归树要保证每次分类后数据的方差最小

随机森林

随机：是指每棵树的训练集是通过随机的有放回的抽取总样本而生成的。

森林：是指模型由多个简单的决策树构成。因为一棵决策树可能对样本的细微变化太过于敏感。

最后通过统计不同子树的结果得到最后随机森林的结果。

XGBoots

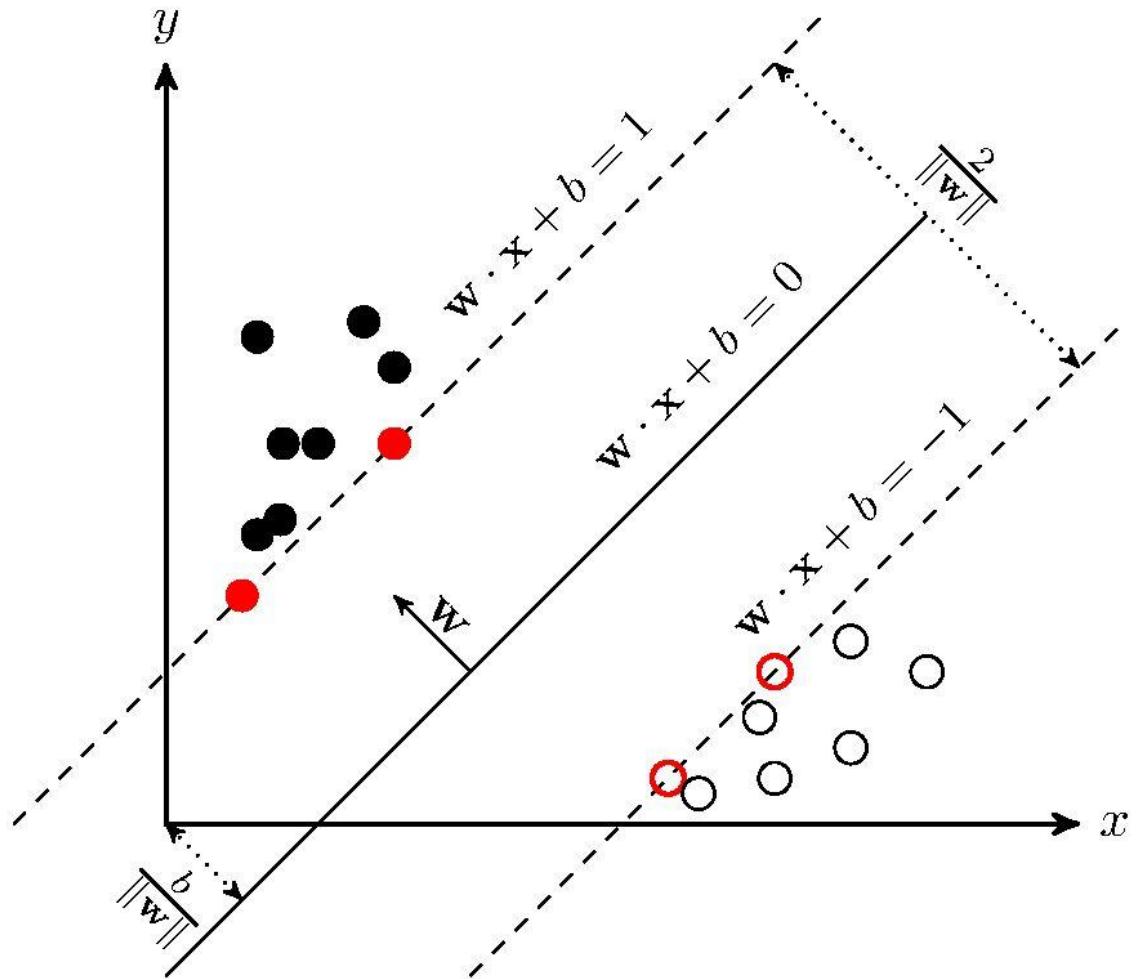
极致梯度提升 (eXtreme Gradient Boosting)，一种基于GBDT的算法。是为速度和性能而设计的梯度提升决策树的实现。

贝叶斯分类器

支持向量机

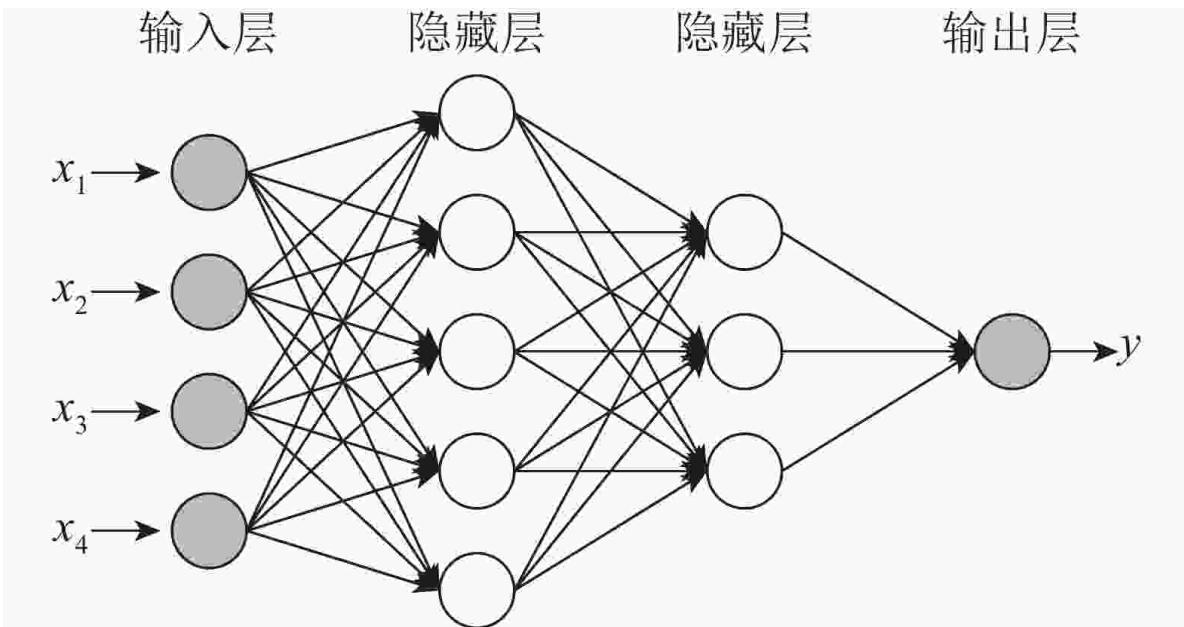
目的

使用一个超平面将两类数据分开，同时要留有一定的间隔。



神经网络

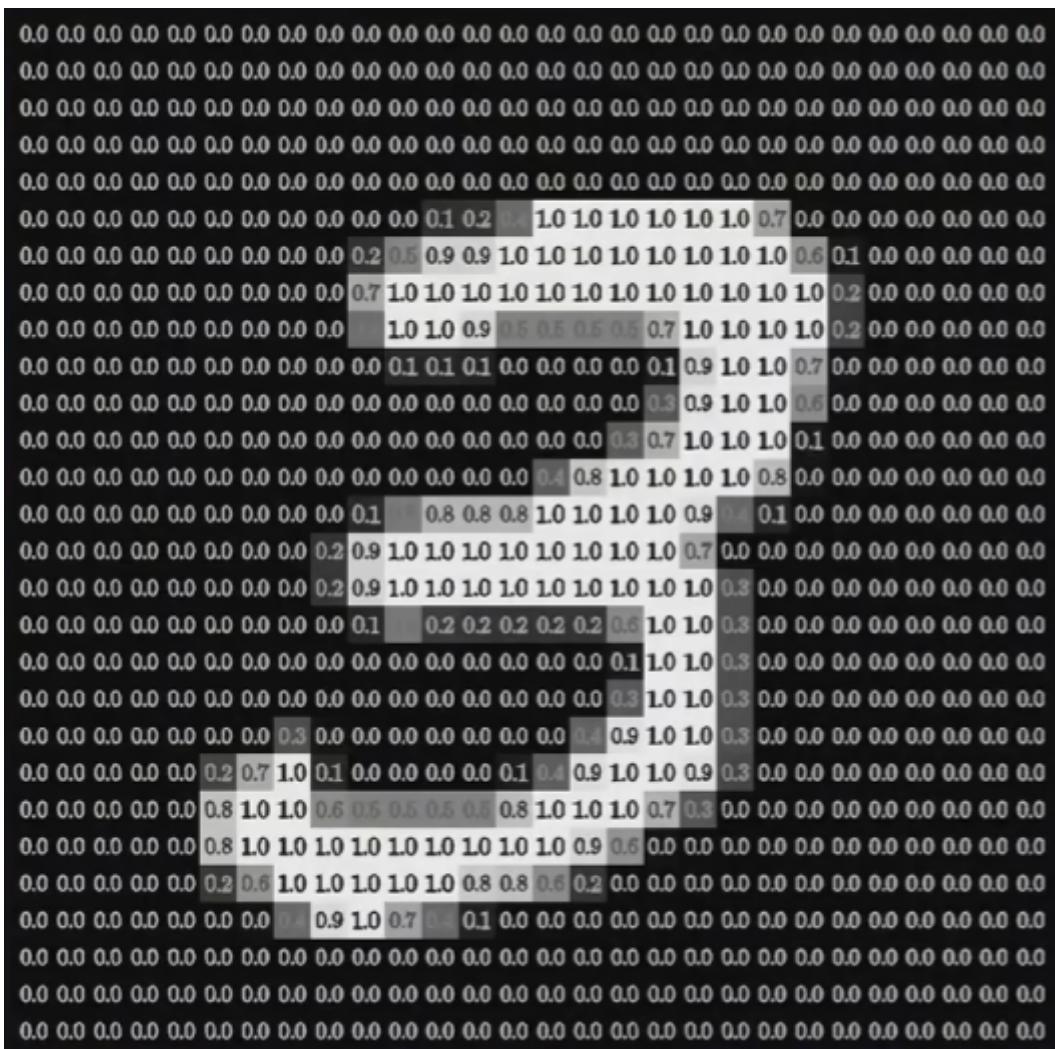
结构



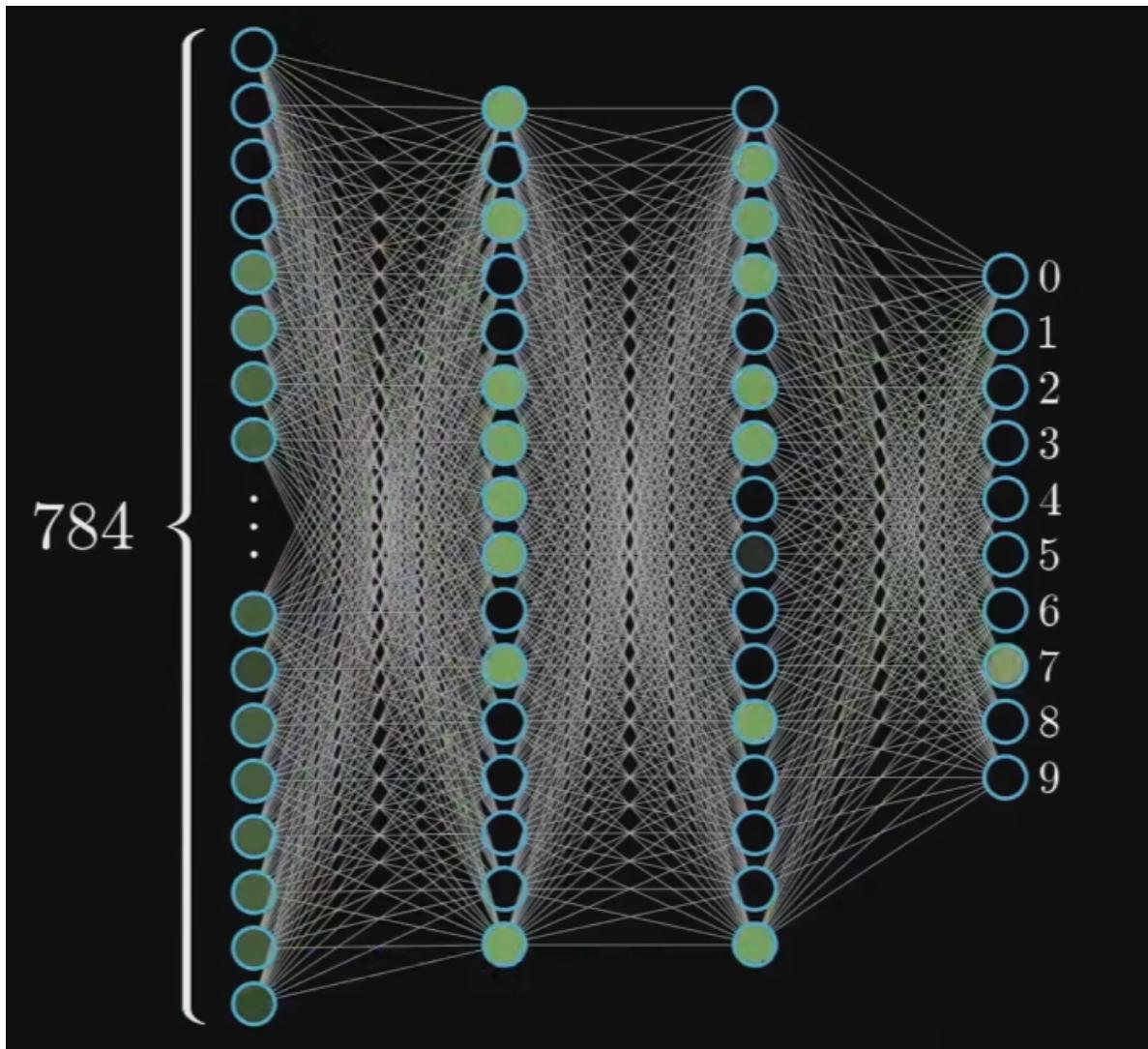
原理

来源于3Blue1Brown视频中的识别手写数字案例。

神经网络的输入是一个 28×28 大小的图片，其中每个像素的取值范围都是 $[0,1]$ ，0表示黑色，1表示白色。



神经网络的结构如下，第0层有784个节点对应每个像素点，第1和第2层有16个节点，输出层是0到9的概率。



每个节点之间的线会有的对应的权重 w ，在节点内部会进行一个激活函数的计算，为了控制再多大权值下才会激发就需要一个参数 b 来控制激发阈值。

单个节点内的公式： $g(w_1a_1 + w_2a_2 + w_3a_3 + \dots + w_na_n + b)$

一层的运算： $a^{(l)} = g \left(\begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ \dots & \dots & \dots & \dots \\ w_{n,0} & w_{n,1} & \dots & w_{n,n} \end{bmatrix} \begin{bmatrix} a_0^{(l-1)} \\ a_1^{(l-1)} \\ \dots \\ a_n^{(l-1)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \dots \\ b_n \end{bmatrix} \right)$

使用向量表示： $\vec{a}^{(l)} = g(\vec{w}^{(l)} \cdot \vec{a}^{(l-1)} + \vec{b}^{(l)})$

对于数字来说，一个数字就是各种抽象特征组合而成（如9是由圆圈和竖线组成、8是由两个圆圈组成），而抽象特征则是由更小的抽象特征组成（如圆圈是由多个小拐弯组成）。所以神经网络中隐藏层的任务就是，在第1层的节点中识别图像中小的抽象特征（如小拐弯、小竖线、小横线），在第2层的节点中识别图像中的抽象特征（如圆圈、竖线、直线），最后通过计算得出最有可能的数字是什么。

以上内容并非是神经网络真正的原理，更像是人们希望神经网络是这样的原理，实际的情况要更加抽象一些。神经网络中的隐藏层之所以这样命名就是因为人们并不知道这其中到底在干什么，这虽然不妨碍实现数字识别但整个过程就想黑箱一样我们无从得知。一方面是因为仅仅上述的网络中就有 $1.3w$ 的参数调整我们无法准确的解释，另一方面很多在数学上优化并不是为了提高可解释性而是为了高效，以及一些其他原因。

向前传播

给定输入，数据从输入层进入，通过网络的层层计算后，从输出层得到结果的过程。即神经网络的运行。

反向传播

通过对比神经网络的输出以及实际要求值的偏差，将结果反向传递到上一层从而修正上一层的w权值，使其更符合要求的输出。这个过程中会使用到梯度下降算法保证整个网络向着最优化的方向进行。即神经网络的学习。

实际中一个网络中一层的某个结点对应着下层的全部结点，有的结点希望上层结点权值变大有的希望变小。方向传播会将所有的正负值相加才得到最终的实际变动大小。

卷积层

卷积层(Convolutional layer)，其中每个结点只关注一部分数据

Scikit-Learn

六大功能

- 分类模型 (Classification)
- 回归模型 (Regression)
- 聚类模型 (Clustering)
- 降维算法 (Dimensionality reduction)
- 模型选择 (Model selection)
- 数据预处理 (Preprocessing)

数据预处理

数据切分

```
from sklearn.model_selection import train_test_split  
  
# 默认训练集和测试集比例为3: 1  
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=42)
```

超参数	默认值	描述
train_size	0.25	训练集的占比 (0~1)
test_size	None	测试集的占比 (0~1)
random_state	None	随机种子
shuffle	True	是否打乱数据 (False下stratify必须为None)
stratify	None	控制训练集和测试集不同类别样本所占比例

标准化

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
# 计算训练集的均值和方差等数据
scaler.fit(x)
# 利用训练集的均值和方差对测试集进行标准化处理
scaler.transform(x)

# 二合一
scaler.fit_transform(x)
```

超参数	默认值	描述
copy	True	是否拷贝新的副本
with_mean	True	是否减去均值
with_std	True	是否除以标准差

模型中的属性	描述
scaler.scale_	查看训练数据各列的标准差
scaler.mean_	查看训练数据各列的均值
scaler.var_	查看训练数据各列的方差
scaler.n_samples_seen_	总共有效的训练数据条数

01标准化

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit_transform(x)
```

超参数	默认值	描述
feature_range	(0,1)	标准化范围
axis	0	对行或对列
copy	True	是否拷贝新的副本

模型中的属性	描述
scaler.data_min_	查看训练数据各列的标准差
scaler.data_max_	查看训练数据各列的均值

归一化

```
from sklearn.preprocessing import Normalizer

normalize = Normalizer()
normalize.fit_transform(x)
```

超参数	默认值	描述
norm	l2	用于规格化每个非零样本的范数。 (l1, l2, max)
copy	True	是否拷贝新的副本

评估器

评估器就理解成一个个机器学习的模型，而sklearn的建模过程最核心的步骤就是围绕着评估器进行模型的训练。

第一步是实例化该对象，其次是围绕某数据进行模型训练。

线性回归评估器

```
model = LinearRegression()
model.set_params(params=val)
```

超参数	默认值	描述
fit_intercept	True	是否构建带有截距项的线性方程
normalize	False	是否进行正则化处理
copy_X	True	建模时是否带入训练数据的副本
n_jobs	None	设置工作时参与计算的CPU核数

模型中的属性	描述
model.coef_	线性方程自变量系数
model.intercept_	线性方程组截距项的系数
model.rank_	特征矩阵的秩
model.singular_	特征矩阵的奇异值

PyTorch

Dataset

提供一种方式去获取数据及其label

```
from torch.utils.data import Dataset

class MyData(Dataset):
    def __init__(self):

        def __getitem__(self, idx):
            # 重写获取单个数据的函数

        def __len__(self):
            # 重写获取数据总数的函数
```

Dataloader

为后面的网络提供不同的数据形式

```
from torch.utils.data import DataLoader

data_loader = DataLoader(dataset=data, batch_size=4, shuffle=True,
num_workers=0, drop_last=False)

for data in data_loader:
    imgs, targets = data
```

超参数	默认值	描述
dataset	None	传入的数据
batch_size	1	每次加载多少数据
shuffle	False	是否打乱数据集
num_workers	0	要使用多少子进程来处理数据
drop_last	False	最后数据不足batch_size时，是否要抛弃

TensorForms

可以将数据转换为Tensor算子，其中有一些属性和参数可以方便完成深度学习的训练

```
from torchvision import transforms

...
ToTensor
将PIL的Image或numpy.ndarray类型的图片格式转换为Tensor格式
...
transformsToTensor = transforms.ToTensor()
img_tensor = transformsToTensor(img)

...
Normalize
根据传入的mean和standard将Tensor格式的图片做归一化处理
...
transformsNormalize = transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
img_normal = transformsNormalize(img_tensor)

...
Resize
对PIL或Tensor类型图片从新调整大小，支持sequence和int型
sequence: (H, W) 高宽
...
transformsResize = transforms.Resize((100, 512))
img_resize = transformsResize(img)

...
RandomCrop
对PIL或Tensor类型图片进行随机裁剪，支持sequence和int型
sequence: (H, W) 高宽
...
transformsRandomCrop = transforms.RandomCrop(400)
img_crop = transformsRandomCrop(img)

...
Compose
将transforms中的各种图像操作串行结合
...
transformsCompose = transforms.Compose([
    ...
])
```

```
transformsToTensor,
transformsNormalize,
transformsResize,
transformsRandomCrop
])
img_compose = transformsCompose(i)
```

Tensorboard

是一组用于数据可视化的工具。TensorBoard是包含在TensorFlow中的一个子服务。但是后来其他深度学习框架也支持了TensorBoard的功能。

```
from torch.utils.tensorboard import SummaryWriter

# 初始化SummaryWriter
writer = SummaryWriter(tag)

...
记录点数据
tag数据表示
x轴数据
y轴数据
...
writer.add_scalar(tag, x, y)
...
记录图片数据
tag数据表示
img支持torch.Tensor / numpy.array / string
step步骤（第几张）
dataformats图像格式CHW / HWC / HW
...
writer.add_image(tag, img, step, dataformats='HWC')

# 关闭writer
writer.close()
```

打开Tensorboard网页

```
# directory_name为保存数据的目录。 默认是“logs”
# port对应端口
tensorboard --logdir=<directory_name> --port=<port>
```

公共数据集加载

```
import torchvision

test_data = torchvision.datasets.CIFAR10(
    "./dataset",      # 数据集所保存目录
    train=False,       # 是否为训练集
    transform=torchvision.transforms.ToTensor(),
    download=True     # 是否下载
)
```

神经网络框架

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

