



# 线性模型和梯度下降算法

# 线性模型

# 1、模型定义

线性模型，即假定输入和输出的关系是线性的。通过属性的线性组合来进行预测,线性回归模型可以表示为：

$$\hat{y} = x * \omega + b$$

或者向量形式

$$f(x) = \omega^T x + b$$

学习方式：根据数据集，确定各个属性的权重 $\omega$ 和 $b$

x (hours)	y (points)
1	2
2	4
3	6
4	?



Linear Model

$$\hat{y} = x * \omega + b$$

x (hours)	y (points)
1	2
2	4
3	6
4	?



Linear Model

$$\hat{y} = x * \omega$$

To simplify the model

线性模型形式简单，易于建模，并且有很好的可解释性。

## 2、一元线性回归

为分析模型，先考虑数据集只有一个属性，线性模型为

$$f(x_i) = \omega * x_i + b$$

模型求解：数据集中的样本点不可能都落在假定的线性模型上，每个样本点与预测值之间的误差可以用：

$$\text{loss} = (\hat{y} - y)^2 = (x * \omega + b - y)^2$$

来表示。

模型的好坏可以用均平方误差函数MSE(mean square error):

$$\text{cost} = \frac{1}{N} \sum_{n=1}^N (\hat{y}_n - y_n)^2$$

来进行评价。

基于上述公式，**均方误差最小化**来进行模型求解的方法称为最小二乘法。最小二乘法就是试图找一条直线，是所有样本到直线上的欧氏距离之和最小。

## Mean Square Error

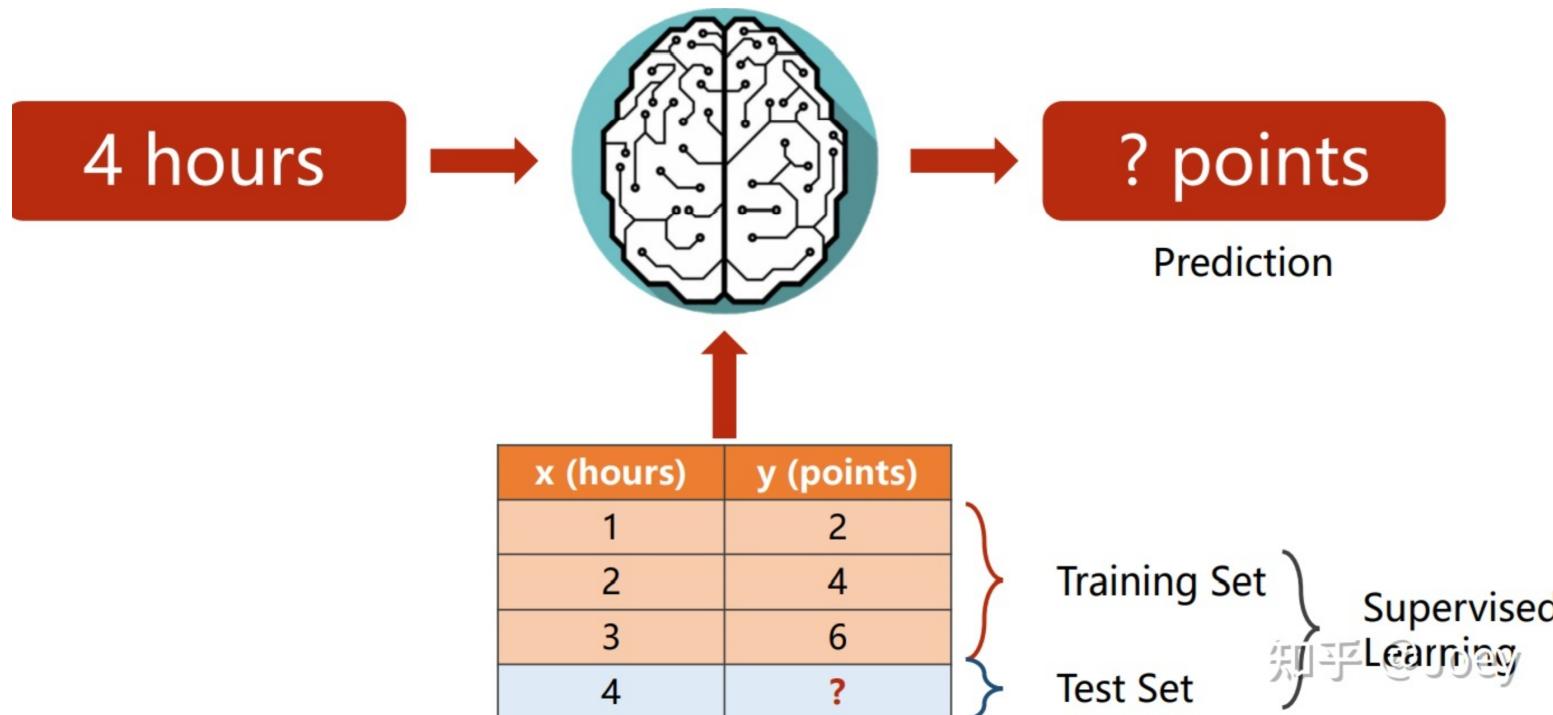
$$cost = \frac{1}{N} \sum_{n=1}^N (\hat{y}_n - y_n)^2$$

x (Hours)	Loss (w=0)	Loss (w=1)	Loss (w=2)	Loss (w=3)	Loss (w=4)
1	4	1	0	1	4
2	16	4	0	4	16
3	36	9	0	9	36
MSE	18.7	4.7	0	4.7	18.7



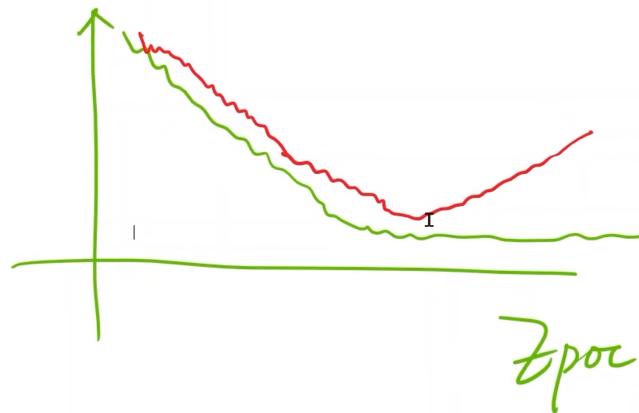
## 2.1求解 $\omega$ 和 $b$

实例：给定数据集：三组x、y，以及一组x，通过三组x、y得到y与x的映射关系，进而对第四组x对应的y进行预测



## 2.2 穷举法

### 1. 穷举法：



通过计算机穷举 $\omega$ 和 $b$ ，模型会逐渐实现收敛，在训练集中误差会逐渐减少，训练出最优的收敛点。后面会使用训练轮数轮数来判定是否收敛成功。

## 2.3 计算 $\omega$

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 # 引入numpy和matplotlib两个包
4
5 x_data = [1.0, 2.0, 3.0]
6 y_data = [2.0, 4.0, 6.0]
7 # 数据集
8
9 def forward(x):
10     global w
11     return x * w
12
13 def loss(x,y):
14     # 损失函数
15     y_pred = forward(x)
16     return (y_pred - y) * (y_pred - y)
17
18 # 记录权重和权重的损失值
19 w_list = []
20 mse_list = []
```

```
1 for w in np.arange(0.0,4.1,0.1):
2     print('w=',w)
3     l_sum = 0
4     # zip函数，把多个列表按顺序依次拼接为多个元组，返回一个
5     for x_val, y_val in zip(x_data,y_data):
6         # 计算预测值和损失
7         y_pred_val = forward(x_val)
8         loss_val = loss(x_val,y_val)
9         # 求损失和
10        l_sum += loss_val
11        print('\t',x_val,y_val,y_pred_val,loss_val)
12
13    print('MSE=',l_sum / 3)
14    # 把权重和损失值记录在列表中
15
16    w_list.append(w)
17    mse_list.append(l_sum / 3)
18
19 plt.plot(w_list,mse_list)
20 plt.ylabel('Loss')
21 plt.xlabel('w')
22 plt.show()
```

## 2.4 穷举结果

输出结果如图所示，容易看出 $w=2$ 是一个最小值，故模型为 $y=2*x$ .

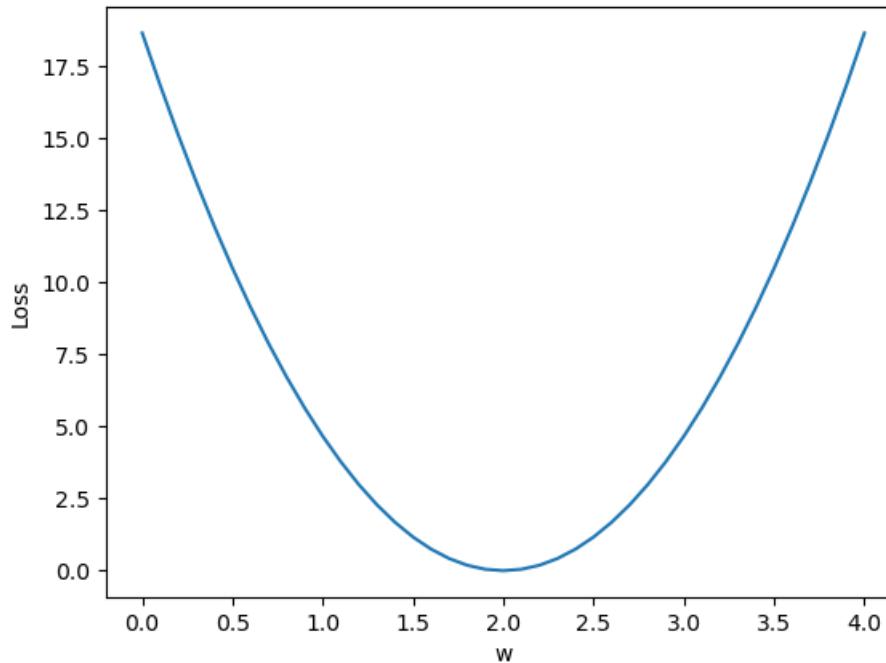


图2.4 MSE-w结果示意图

### 3、多元回归模型

#### 3.1 多元线性模型

更一般的情形是：假设数据集D，样本由d个属性描述。此时我们试图学得

$$f(x) = \omega^T x + b$$

这称为“多元线性回归”，同样多元线性回归模型也可以用最小二乘法来做：

$$\hat{\omega}^* = \arg(\hat{w}) \min(y - X\hat{\omega})^T (y - X\hat{\omega}))$$

即使得函数 $(y - X\hat{\omega})^T (y - X\hat{\omega})$ 取最小值时的参数 $\hat{\omega}^*$ 是最优解。

上述函数可以通过穷举或者数学的方式算出最优解：对 $\hat{\omega}$ 求导可得

$$\frac{\partial E_{\hat{\omega}}}{\partial \hat{\omega}} = 2X^T(X\hat{\omega} - y)$$

另上式为零，可以得到 $\hat{\omega}$ 最优解的闭式解。

## 4、线性模型的拓展

### 4.1 对数线性回归

之前的模型都是直接把预测值逼近 $y$ , 如果数据集对应的输出标记是在指数尺度上变化, 可以把输出标记的对数作为线性模型的逼近目标。

$$\ln y = \omega^T x + b$$

这个模型被称为“对数线性回归”, 虽然形式上是线性模式, 但实质上是求输入空间到输出空间的非线性函数映射

### 4.2 广义线性模型

更进一步, 如果一个单调可微函数 $g(x)$ , 令

$$y = g^{-1}(\omega^T x + b)$$

这个模型称为广义线性模型

参数估计也可以通过加权最小二乘法来进行

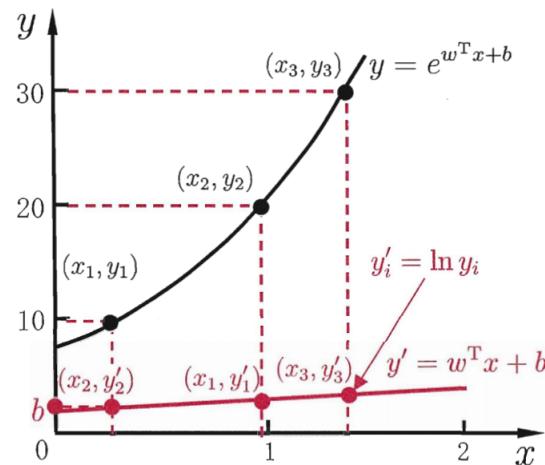


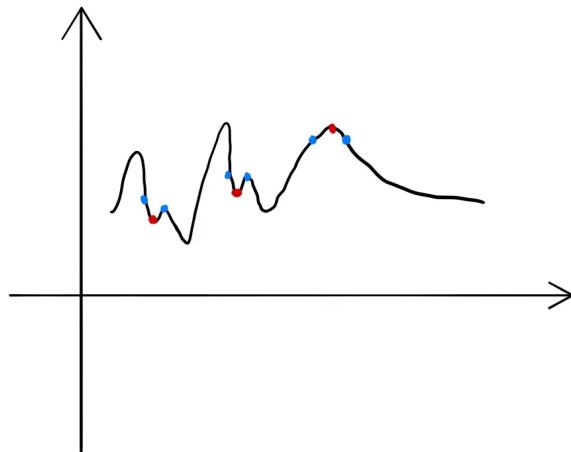
图 3.1 对数线性回归示意图

# 梯度下降算法

# 1.1 梯度下降算法的引入

上一章我们在线性模型(也即最小二乘法)，对于参数 $w$ 的范围选择是基于图像以及对于二次函数(MSE)的先验知识而确定的，但是面对一个复杂且难以通过肉眼观测最小值范围的损失函数时，模型参数的范围比较难确定。

- 穷举法：当自变量维度增加，搜索的区间范围就增加，很难搜索最小值
- 分治法：容易陷入**局部最优**，且容易错失最优解

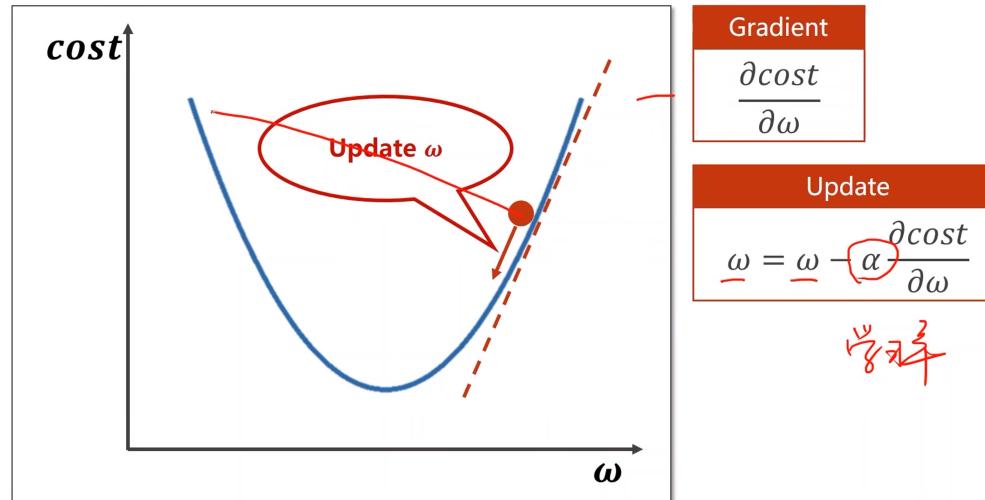


# 1.2 梯度下降

百度百科：梯度(gradient)的本意是一个向量（矢量），表示某一函数在该点处的方向导数沿着该方向取得最大值，即函数在该点处沿着该方向（此梯度的方向）变化最快，变化率最大（为该梯度的模）。

## Gradient Descent Algorithm

刘二大人 bilibili



Lecturer : Hongpu Liu

Lecture 3-11

PyTorch Tutorial @ SLAM Research



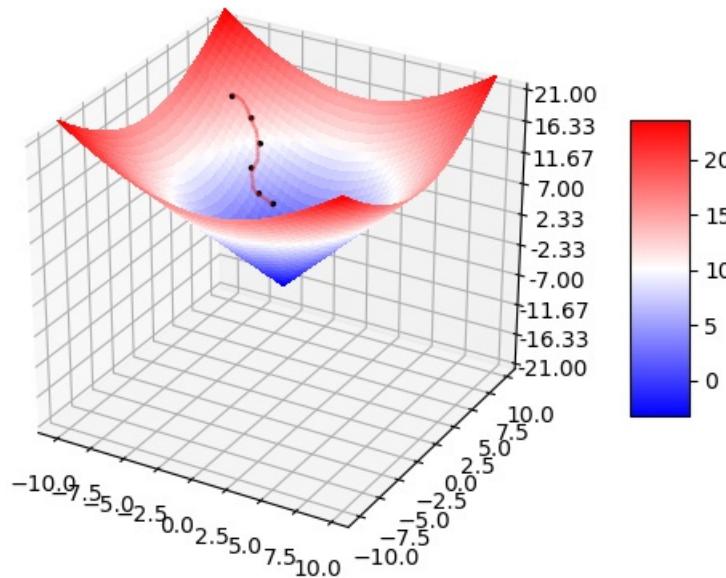
梯度指向的是函数增长的方向，那么我们想要得到损失函数逐渐减少，那么我们每次就得让模型的参数往梯度相反的方向走，这样才能使得损失函数不断下降，以期得到最小值

## 2.1 梯度下降算法

核心概念：不断迭代每一步都向梯度下降的方向前进.梯度下降可以理解为你站在山的某处，想要下山，此时最快的下山方式就是你环顾四周，哪里最陡峭，朝哪里下山，一直执行这个策略，在第N个循环后，你就到达了山的最低.

算法步骤：

1. 给定待优化连续可微分的函数 $J(\theta)$ ,步长, $\alpha$
2. 计算待优化函数的梯度
3. 更新迭代
4. 再次计算梯度
5. 向量模小于一定值或达到目标迭代次数



## 2.2 梯度下降迭代公式

### Gradient Descent Algorithm

刘二大人 bilibili

#### Derivative

$$\begin{aligned}\frac{\partial \text{cost}(\omega)}{\partial \omega} &= \frac{\partial}{\partial \omega} \frac{1}{N} \sum_{n=1}^N (x_n \cdot \omega - y_n)^2 \\ &= \frac{1}{N} \sum_{n=1}^N \frac{\partial}{\partial \omega} (x_n \cdot \omega - y_n)^2 \\ &= \frac{1}{N} \sum_{n=1}^N 2 \cdot (x_n \cdot \omega - y_n) \frac{\partial (x_n \cdot \omega - y_n)}{\partial \omega} \\ &= \frac{1}{N} \sum_{n=1}^N 2 \cdot x_n \cdot (x_n \cdot \omega - y_n)\end{aligned}$$

#### Gradient

$$\frac{\partial \text{cost}}{\partial \omega}$$

#### Update

$$\omega = \omega - \alpha \frac{\partial \text{cost}}{\partial \omega}$$

#### Update

$$\omega = \omega - \alpha \frac{1}{N} \sum_{n=1}^N 2 \cdot x_n \cdot (x_n \cdot \omega - y_n)$$



## 3.1 批量梯度下降算法

前面所讨论中使用的梯度下降算法公式为

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{n} \sum_{i=1}^n (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

可以看出，计算机会每次从所有数据中计算梯度，然后求平均值，作为一次迭代的梯度，对于高维数据，计算量相当大，因此，把这种梯度下降算法称之为**批量梯度下降算法**。

## 3.2 随机梯度下降

随机梯度下降算法是利用批量梯度下降算法每次计算所有数据的缺点，**随机抽取某个数据来计算梯度作为该次迭代的梯度，梯度计算公式：**

$$\frac{\partial J(\theta)}{\partial \theta_j} = (h - \theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

迭代公式：

$$\theta = \theta - \alpha \cdot \nabla_{\theta} J(\theta; x^{(i)}; y^{(i)})$$

由于随机选取某个点，省略了求和和求平均的过程，降低了计算复杂度，提升了计算速度，但由于随机选取的原因，存在较大的震荡性

### 3.3 小批量梯度下降算法

小批量梯度下降算法是综合了批量梯度下降算法和随机梯度下降算法的优缺点，随机选取样本中的一部分数据，梯度计算公式

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{k} \sum_i^{i+k} (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

迭代公式：

$$\theta = \theta - \alpha \cdot \nabla_{\theta} J(\theta; x^{(i:i+k)}, y^{(i:I+k)})$$

# 4 代码及效果

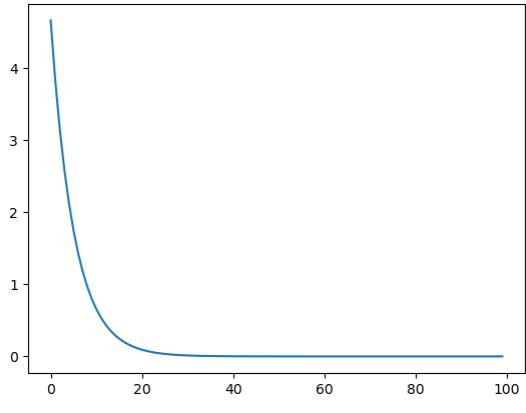
## 批量梯度下降算法

```
1  def cost(xs,ys):# 计算MSE
2      cost = 0
3      for x,y in zip(xs,ys):
4          y_pred = forward(x)
5          cost += (y_pred - y) ** 2
6      return cost /len(xs)
7  def gradient(xs,ys):# 梯度函数,算每个样本点的平均梯度
8      grad = 0
9      for x,y in zip(xs,ys):
10         grad += 2 * x * (x * w - y)
11     return grad /len(xs)
12 #更新权重 新权重 = 旧权重 - 学习率 * 梯度
13 for epoch in range(100):
14     cost_val = cost(x_data,y_data) #先算成本
15     grad_val = gradient(x_data,y_data) #训练过程
16     w -= a * grad_val
17     epoch_list.append(epoch)
18     w_list.append(w)
19     cost_list.append(cost_val)
20     # 输出训练日志
```

# 随机批次梯度下降算法

```
1 def loss(x,y):#定义损失函数
2     y_pred = forward(x)
3     return (y_pred-y)**2
4 #梯度函数
5 def gradient(x,y):
6     return 2*x*(x*w-y)
7
8 print('Predict (before training)',4,forward(4))
9 epoch_list=[]
10 loss_list=[]
11 #epoch为迭代次数,共训练100轮次
12 for epoch in range(100):
13     for x,y in zip(x_data,y_data):
14         #梯度
15         grad = gradient(x,y)
16         #若在某一点梯度为0, 则停止迭代, 对应的w值即稳定不变
17         w -= 0.01 * grad
18         print("\tgrad:",x,y,grad)
19         l =loss(x,y)
20         epoch_list.append(epoch)
21         loss_list.append(l)
22         print("progress:",epoch,"w=",w,"loss=",l)
```

## 批量梯度下降算法



## 随机批次梯度下降算法

