吴恩达机器学习

1 监督学习

1.1 单变量线性回归(Linear Regression with One Variable)

输入特征 Input Feature x

输出标签 Output Label y

样本 Sample (x, y)

样本数量 Sample Number m

第 i 个样本 (x^i, y^i)

数据集 training set

model f or h

输出的结果 esitimate $\hat{y}$

损失函数 Cost function $J(...)$

how to represent the function f? We can konw f is a linear function,So we have

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

we call w, b as weights or parameters 权重或是参数.

Then we define a function called cost function $J(w, b)$ to measure the error between the real value and the estimated value. The cost function is defined as

$$\hat{y^i} = f_{w,b}(x^i)$$

$$\text{Cost} = J(x) = J(w, b) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^i - y^i)^2 \tag{1.1}$$

$\frac{1}{2}$ is just used to make the derivation easier.

We should found the best w, b to minimize the cost function $J(w, b)$. it called

$$\text{minimze}_{w,b} J(w, b)$$

先将 b 设置为 0，关于 w 的损失函数是个抛物线。

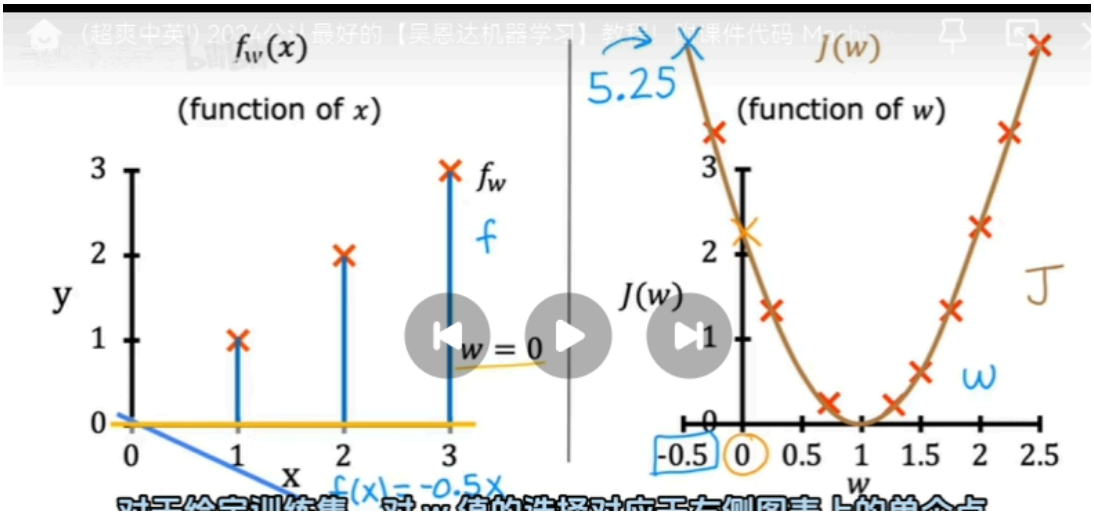

Abb. 1:

for tow parameters it can display in three dimensions

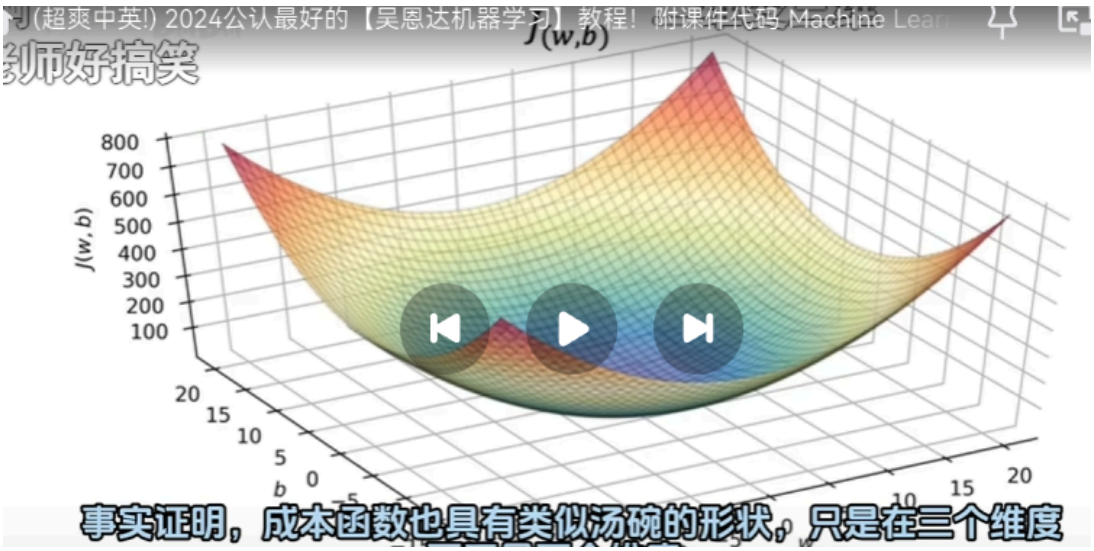

Abb. 2:

同理也可以用等高线图表示

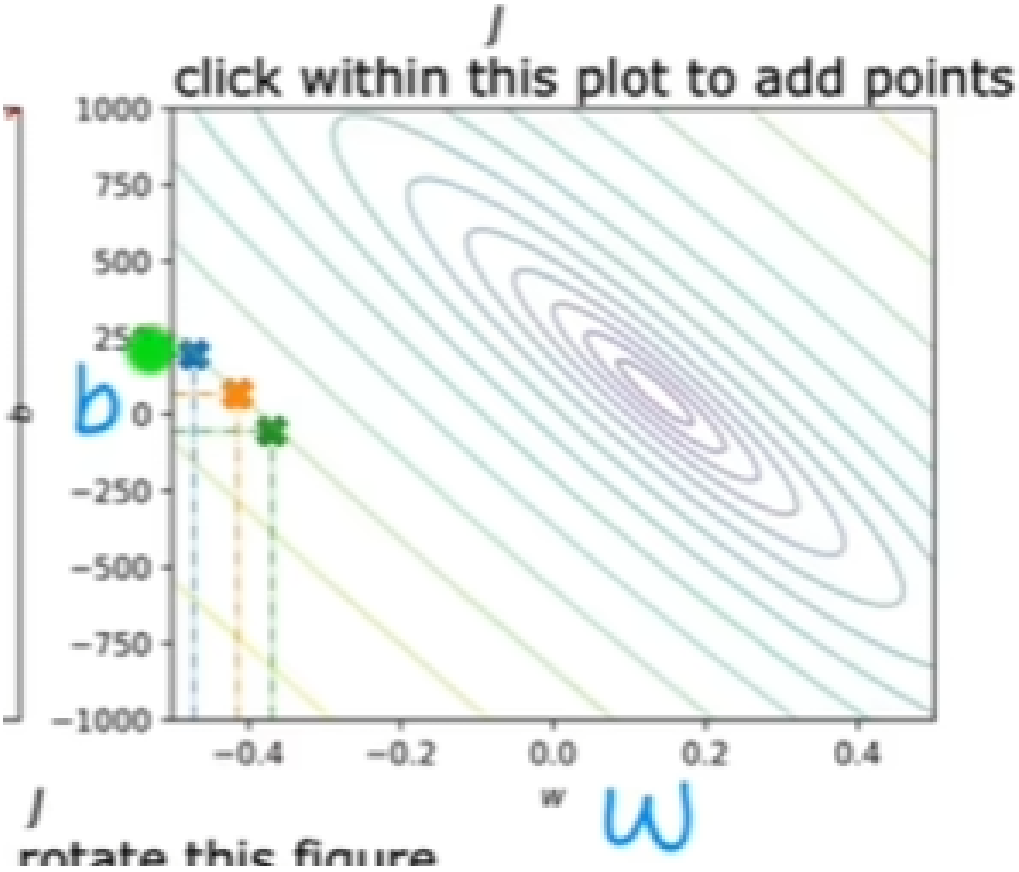

Abb. 3:

想象一座山来具象这个等高线图(Counter plot)。

1.1.1 梯度下降算法 gradient descent

Keep change w, b to reduce $J(w, b)$ to minimize it. 对于我们使用的平方损失函数,他总是上图所示的抛物面形状. 对于其他的损失函数,可能是各种各样的.

梯度下降对于不同的起点会有不同的结果,有可能进入局部最小值.

学习率 Learning rate α

先给出公式

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

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

正确的算法流程是

$$\text{tem}_w = w - \alpha \frac{\partial}{\partial w} J(w,b)$$

$$\text{tem}_b = w - \alpha \frac{\partial}{\partial b} J(w,b)$$

$$w = \text{tem}_w$$

$$b = \text{tem}_b$$

其中

1.1.1.1 数值求解积分

如[1],对于一个连续的一元函数，$f(x,y)$在$[a,b]$上的定积分

$$\int_a^b f(x) = f(b) - f(a)$$

但是如果$f(x)$是一个离散的如$f(x_i), i = 1,2,3...n$，只能使用估计值

$$I(f) = \int_a^b f(x)dx$$

$$I_n(f) = \sum_{i=1}^{n} f(x_i)h_i$$

如果有$I_n\left(x^k\right) - I\left(x^k\right) = 0$,就说具有 k 次精度.

1.1.1.1.1 New Cote,s 积分

将$[a,b]$分成n段，步长$h = \dfrac{b-a}{b}, x_i = a + ih$,构造拉格朗日插值多项式[1]$L(x)$

$$\int_a^b f(x)dx \approx \int_a^b L(x)dx$$

---

[1]拉格朗日插值多项式是一种通过给定的离散数据点来构造一个多项式的方法，使得该多项式在这些数据点上取值与给定的值相等。具体来说，如果给定$n+1$个数据点$(x_0,y_0),(x_1,y_1),...,(x_n,y_n)$，其中$x_i$互不相同，那么存在一个唯一不超过$n$次的多项式$P(x)$满足$P(x_i) = y_i$对所有$i = 0,1,...,n$。

拉格朗日插值多项式可以表示为:

$$P(x) = \sum_{i=0}^{n} y_i L_i(x)$$

其中$L_i(x)$是拉格朗日基多项式，定义为:

$$L_i(x) = \prod_{\substack{0 \le j \le n \\ j \ne i}} \frac{x - x_j}{x_i - x_j}$$

拉格朗日基多项式$L_i(x)$具有性质$L_i(x_j) = \delta_{ij}$，即当$i = j$时$L_i(x_j) = 1$，当$i \ne j$时$L_i(x_j) = 0$。因此，多项式$P(x)$在x_i处的值为y_i。

由此得到的数值积分称为牛顿 - 柯特斯积分。

梯形积分

取端点a,b构造拉格朗日多项式，显然，这是一个一次函数（$n+1=2,n=1$）,显然其积分等于面积，具有一阶代数精度。

$$\int_a^b f(x)dx \approx \int_a^b L_1(x)dx \approx \left[(f(a)+f(b)) \times \frac{b-a}{2}\right]$$

对于更普遍的,可以写成

```Python
def df(x, h=1e-10):
    return (f(x+h) - f(x-h)) / (2*h)
```

辛普森积分

具有三阶代数精度,对于次数不超过三次的多项式准确成立。 以$(a,f(a)),\left(\frac{a+b}{2},f\left(\frac{a+b}{2}\right)\right)$和$(b,f(b))$为插值节点构造二次插值函数 $L_2(x)$

$$\int_a^b f(x)dx \approx I_2(f)=S(f)=\frac{b-a}{6}\left[f(a)+4f\left[\frac{a+b}{2}\right]+f(b)\right]$$

```Python
def df_xps(x, h=1e-10):
    return (f(x+h) + 4*f(x)+f(x-h))*h/3
```

New Cote,s

表 2.1

n	$c_0^{(n)}$	$c_1^{(n)}$	$c_2^{(n)}$	$c_3^{(n)}$	$c_4^{(n)}$	$c_5^{(n)}$	$c_6^{(n)}$
1	$\frac{1}{2}$	$\frac{1}{2}$					
2	$\frac{1}{6}$	$\frac{4}{6}$	$\frac{1}{6}$				
3	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$			
4	$\frac{7}{90}$	$\frac{16}{45}$	$\frac{2}{15}$	$\frac{16}{45}$	$\frac{7}{90}$		
5	$\frac{19}{288}$	$\frac{25}{96}$	$\frac{25}{144}$	$\frac{25}{144}$	$\frac{25}{96}$	$\frac{19}{288}$	
6	$\frac{41}{840}$	$\frac{9}{35}$	$\frac{9}{280}$	$\frac{34}{105}$	$\frac{9}{280}$	$\frac{9}{35}$	$\frac{41}{840}$

Abb. 4: 牛顿科特斯积分的系数

$m=1$是梯形积分，$m=2$是辛普森积分。不过由插值的龙格现象可知,高阶牛顿 柯特斯积分不能保证等距数值积分系列的收敛性 ,同时可证(略)高阶牛顿 柯特斯积分的计算是不稳定的 。 因此,实际计算中常用低阶 复化梯形等积分公式

1.1.1.2 数值微分

分为前向微分，后向微分，和中心微分。有两点式和三点式。

$h = x_{i+1} - x_{i-1}$

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}x} f(x) &= \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} + o(h) \\
&= \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} + o(h) \\
&= \frac{f(x_{i+1}) - f(x_{i-1})}{x_{i+1} - x_{i-1}} + o(h^2) \\
&= \frac{3f(x_i) - 4f(x_{i+1}) + f(x_{i+2})}{-2h} + o(h^2) \\
&= \frac{3f(x_i) - 4f(x_{i-1}) + f(x_{i-2})}{2h} + o(h^2)
\end{aligned}$$

1.1.2 python 中一些常用的数值计算库

1.1.2.1 numpy

1.1.2.1.1 生成数组

array	创建数组	`np.array([1,2.3])`	`np.array([[1,2.3], [4,5,6]])`
zeros	全 0 数组	`np.zeros(shape=(3,4))` `# 3x4 matrix`	
ones	全 1 数组	`np.ones(shape=(3,4))` `# 3x4 matrix`	
empty	创建全空数组	…	
arrange	创建等长	`np.arange(a,b,c)`	$x_0 = a, x_{i+1} = x_i + c, x_i \in [a, b)$
linspace	创建 n 个等分	`np.linspace(a,b,c)`	$x_i \in [a, b], n = c, \forall j, x_{j+1} - x_j = x_j - x_{j-1}$
rand	随机数组	`np.random.rand(3,4)`	`np.random.randint(2,5,size=(3,4))`

1.1.3 matplotlib:pyplot

1.1.4 linear regration 梯度下降算法的实现

首先我们的损失函数如 Gleichung (1.1) 所示,Let,t we simplify it

$$J(x) = J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}^i - y^i\right)^2 = \frac{1}{2m}\sum_{i=1}^{m}\left(x_1 X_{i_1} + x_0 X_{i_0} - y_i\right)^2 \tag{1.2}$$

对 Gleichung (1.2) 求导

$$\frac{\mathrm{d}}{\mathrm{d}x_1}J(x) = \frac{1}{m}\sum_{i=1}^{m}\frac{\mathrm{d}}{\mathrm{d}x_1}\left(x_1 X_{i_1} + x_0 X_{i_0} - y_i\right) = \frac{1}{m}\sum_{i=1}^{m} X_{i_1}\left(x_1 X_{i_1} + x_0 X_{i_0} - y_i\right) \tag{1.3}$$

这是我们对原函数十分熟悉的时候，如果不熟悉就需要数值求解原函数的导数，下面我们分别给出两种解法。

1.1.4.1 解析解法

```Python
def gradientDescent(X, y, theta, alpha, iters):
    temp = np.matrix(np.zeros(theta.shape))
    parameters = int(theta.ravel().shape[1])
    cost = np.zeros(iters)
    for i in range(iters):
        for j in range(parameters):
            term = np.multiply((X * theta.T) - y, X[:,j])
            temp[0,j] = theta[0,j] - alpha*((1 / len(X)) * np.sum(term))
        theta = temp
        cost[i] = computeCost(X, y, theta)

    return theta, cost
```

上述的θ理解为我们文中的x。

同时，其求解得到的结果为

matrix([[−3.25095985, 1.12837093]])

1.1.4.2 数值解法

```Python
def gradientDescent_(X, y, theta, alpha, iters):
    temp = np.matrix(np.zeros(theta.shape))
    parameters = int(theta.ravel().shape[1])
    cost = np.zeros(iters)

    d_len=0.01
    for i in range(iters):
        theta_=np.matrix(np.ones(theta.shape)*d_len)
        for j in range(parameters):
```

```
            # 对于第 j 个变量 忽略其他
            theta_use=np.matrix(np.zeros(theta.shape))
            theta_use[0,j]=theta[0,j]
             term = (computeCost(X, y, theta+theta_use)-computeCost(X, y, theta-theta_use))/(2*d_len)
            temp[0,j] = theta[0,j] - alpha*term
        theta = temp
        cost[i] = computeCost(X, y, theta)

    return theta, cost
```

结果是一样的。

1.1.5 梯度下降算法的拓展

1.1.5.1 一般函数

比如$a\sin(\omega x)+2.09-a$

```python
np.random.seed(12891832)
a=0.780+0.265*np.random.rand()
omega=1.884+0.116*np.random.rand()

size=1000
def f_(x,a_=a,omega_=omega):
    return a_*np.sin(omega_*x)+2.09-a_
data=np.array([])
X=np.linspace(0,4,size)
Y=f_(X)+np.random.normal(0, 0.8, size)*a
X=np.matrix(X)
Y=np.matrix(Y)

def computeCost(X, y, theta):
    a_ = theta[0,0]
    omega_ = theta[0,1]
    inner = np.power(((a_*np.sin(omega_*X)+2.09-a_) - y), 2)
    return np.sum(inner) / (2 * len(X))
    # return 0
```

这里为了简单，我们还是直接使用了 MSE loss function，只需要修改 Cost 部分即可。

1.1.5.2 其他的损失函数及其应用场景

Regration Loss 回归损失

Mean Absolute Error $J(\theta) = |\hat{y} - y|$

- 也叫做 L1LOSS MAE 平均绝对五擦汗

优点 收敛速度慢，小的损失值和其他一样，不利于网络学习

缺点 鲁棒性好（对离群点和异常值）

Regration Loss 回归损失

Mean Squred Error $J(\theta) = (\hat{y} - y)^2$

- 也叫做 L2LOSS 均方误差 MSE

优点 收敛速度快

缺点 鲁棒性差

- Classification Loss 分类损失

1.1.5.3 梯度下降的优化

1.1.5.3.1 随机梯度下降

1.1.5.3.2 小批量梯度下降

1.1.5.4 损失函数的可视化

3d 图，我们还是用 matlab 画吧。

2 能量机关关键点检测实战

2.1 COCO 数据集及其定义

对于每张图，其注释如下所示

```
{"segmentation":[[0.43,299.58,2.25,299.58,9.05,287.78,32.66,299.13,jsonnet
"num_keypoints":1,
"area":1037.7819,
"iscrowd":0,
"keypoints":[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,9,277,2,
"image_id":397133,
"bbox":[0,262.81,62.16,36.77],
"category_id":1,
"id":1218137}
```

其中的 keypoints 部分，每三个数字为一组，代表一个关键点，三个值分别为 x 坐标、y 坐标、标志位，其中，标志位有三个值：

0：未标注 1：标注，但被遮挡 2：标注，未遮挡

2.2 搭建深度学习环境

使用 Docker Desktop 直接搭建 Docker 镜像。安装 Pytorch Transformer 自动标注工具，Dockerfile 如下所示

```dockerfile
FROM nvidia/cuda:11.0-cudnn8-devel-ubuntu18.04
RUN apt-get update && apt-get install -y python3 python3-pip
RUN pip3 install torch torchvision
```

参考文献

1. 张韵华, 奚梅成, 陈长松 & 数值计算. 数值计算方法和算法. 38–39 (科学出版社, 2000).