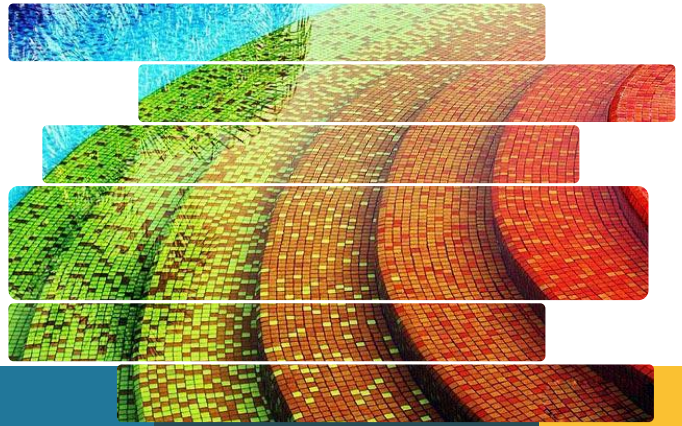
神经网络&深度学习 Google TENSORFLOW



10 梯度下降法

西安科技大学 牟琦
muqi@xust.edu.cn


tensorflow / tensorflow
TensorFlow: Open sou
Clone this repo:
git clone https://tensor
Branches
master

10.1 梯度下降法原理

- 求解**线性回归**模型——函数求极值
  - **解析解**

    根据严格的推导和计算得到，是方程的**精确解**

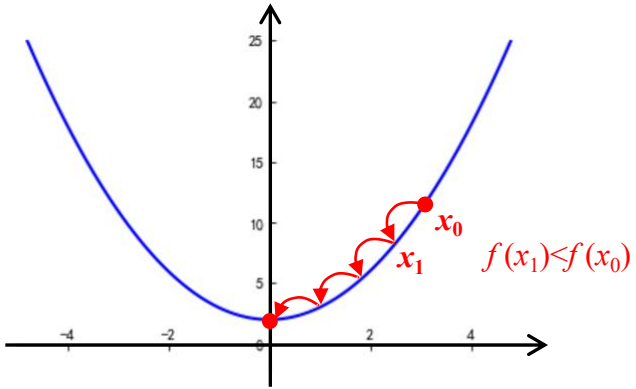
    能够在**任意精度**下满足方程
  - **数值解**

    通过某种**近似计算**得到的解

    能够在**给定的精度**下满足方程

一元凸函数求极值

$$f(x) = x^2 + 2$$



迭代法求极小值（步长=0.2）

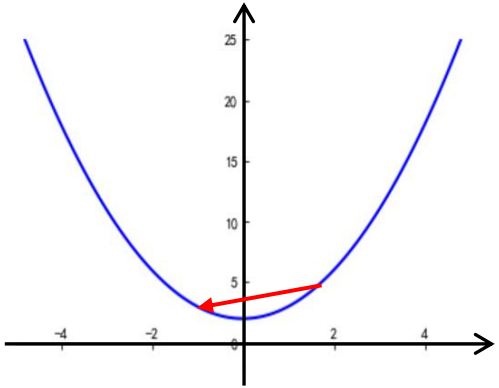
迭代次数	x_i	候选值	$y=x^2+2$	取值	迭代次数	x_i	候选值	$y=x^2+2$	取值
0	3	2.8 3.2	9.84 12.24	√	8	1.4	1.2 1.6	3.44 4.56	√
1	2.8	2.6 3	8.76 11	√	9	1.2	1 1.4	3 3.96	√
2	2.6	2.4 2.8	7.76 9.84	√	10	1	0.8 1.2	2.64 3.44	√
3	2.4	2.2 2.6	6.84 8.76	√	11	0.8	0.6 1	2.36 3	√
4	2.2	2 2.4	6 7.76	√	12	0.6	0.4 0.8	2.16 2.64	√
5	2	1.8 2.2	5.24 6.84	√	13	0.4	0.2 0.6	2.04 2.36	√
6	1.8	1.6 2	4.56 6	√	14	0.2	0 0.4	2 2.16	√
7	1.6	1.4 1.8	3.96 5.24	√	15	**0**	-0.2 0.2	2.04 2.04	

迭代法求极小值（步长=0.5）

迭代次数	x_i	候选值	$y=x^2+2$	移动方向
0	3	2.5	8.25	√
		3.5	14.25	
1	2.5	2	6	√
		3	11	
2	2	1.5	4.25	√
		2.5	8.25	
3	1.5	1	3	√
		2	6	
4	1	0.5	2.25	√
		1.5	4.25	
5	0.5	0	2	√
		1	3	
6	**0**	**-0.5**	**2.25**	
		0.5	**2.25**	

震荡

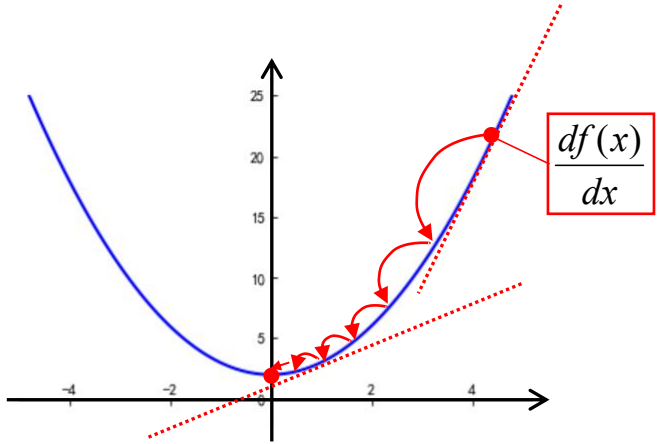
overshoot the minimum



迭代法求极小值（步长=**0.7**）

迭代次数	x_i	候选值	$y=x^2+2$	移动方向
0	3	2.3 3.7	7.29 15.69	√
1	2.3	1.6 3	4.56 11	√
2	1.6	0.9 2.3	2.81 7.29	√
3	0.9	0.2 1.6	2.04 4.56	√
4	0.2	-0.5 0.9	2.25 2.81	√
5	-0.5	-1.2 0.2	3.44 2.04	√
6	0.2	-0.5 0.9	2.25 2.81	√
7	-0.5	-1.2 0.2	3.44 2.04	√
8	0.2	-0.5 0.9	2.25 2.81	√

步长太小，迭代次数多，收敛慢
步长太大，引起震荡，可能无法收敛



步长 $= \eta \frac{df(x)}{dx}$　　η：学习率

$$x^{(k+1)} = x^{(k)} - \eta \frac{df(x)}{dx}$$

- 自动调节步长
- 自动确定下一次更新的方向
- 保证收敛性

二元凸函数求极值

$$x^{(k+1)} = x^{(k)} - \eta \frac{\partial f(x,y)}{\partial x}$$
$$y^{(k+1)} = y^{(k)} - \eta \frac{\partial f(x,y)}{\partial y}$$

$$\nabla = \begin{pmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{pmatrix}$$

模为方向导数的最大值

方向为取得最大方向导数的方向

梯度:

$$\overrightarrow{grad}\, f(x,y) = \frac{\partial f}{\partial x}\vec{i} + \frac{\partial f}{\partial y}\vec{j}$$

只要能够把**损失函数**描述成**凸函数**

那么就一定可以采用**梯度下降法**

以**最快**的速度更新**权值向量w**

找到使损失函数达到**最小值点**的位置

西安科技大学 计算机科学与技术学院

10.2 梯度下降法求解线性回归

10.2.1 梯度下降法求解线性回归

梯度下降法求解一元线性回归问题

$$Loss=\frac{1}{2}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2=\frac{1}{2}\sum_{i=1}^{n}(y_i-(wx_i+b))^2$$

极值问题：$\arg\min_{w,b} Loss(w,b)$

$$=\frac{1}{2}\sum_{i=1}^{n}({x_i}^2w^2+b^2+2x_iwb-2y_ib-2x_iy_iw+{y_i}^2)$$

$$=Aw^2+Bb^2+Cwb+Dw+Eb+F$$ **凸函数**

$$w^{(k+1)}=w^{(k)}-\eta\frac{\partial Loss(w,b)}{\partial w}$$

$$b^{(k+1)}=b^{(k)}-\eta\frac{\partial Loss(w,b)}{\partial b}$$

$$\frac{\partial Loss}{\partial w}=\sum_{i=1}^{n}(y_i-b-wx_i)(-x_i)$$

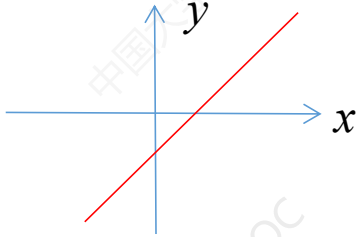
$$\frac{\partial Loss}{\partial b}=\sum_{i=1}^{n}(y_i-b-wx_i)(-1)$$

$$w^{(k+1)}=w^{(k)}-\eta\sum_{i=1}^{n}x_i(wx_i+b-y_i)$$

$$b^{(k+1)}=b^{(k)}-\eta\sum_{i=1}^{n}(wx_i+b-y_i)$$

一元线性回归问题

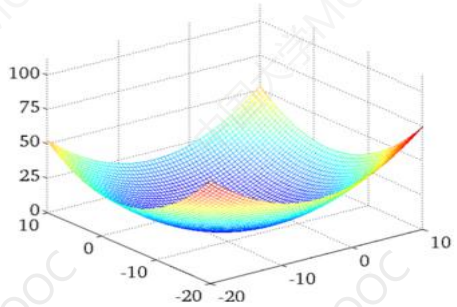
$$y = wx + b$$



二元求极值问题

$$\arg\min_{w,b} Loss(w,b)$$

$$Loss = \frac{1}{2}\sum_{i=1}^{n}(y_i - (wx_i + b))^2$$



$$w^{(k+1)} = w^{(k)} - \eta\frac{\partial Loss(w,b)}{\partial w}$$

$$b^{(k+1)} = b^{(k)} - \eta\frac{\partial Loss(w,b)}{\partial b}$$

平方损失函数

$$Loss = \frac{1}{2}\sum_{i=1}^{n}(y_i - (wx_i + b))^2$$

模型参数更新算法：

$$w^{(k+1)} = w^{(k)} - \eta\sum_{i=1}^{n}x_i(wx_i + b - y_i)$$

$$b^{(k+1)} = b^{(k)} - \eta\sum_{i=1}^{n}(wx_i + b - y_i)$$

均方差损失函数

$$Loss = \frac{1}{2n}\sum_{i=1}^{n}(y_i - (wx_i + b))^2$$

模型参数更新算法：

$$w^{(k+1)} = w^{(k)} - \frac{\eta}{n}\sum_{i=1}^{n}x_i(wx_i + b - y_i)$$

$$b^{(k+1)} = b^{(k)} - \frac{\eta}{n}\sum_{i=1}^{n}(wx_i + b - y_i)$$

梯度下降法求解多元线性回归问题

$$\hat{Y} = XW$$

$$Loss = \frac{1}{2}(Y - \hat{Y})^2 = \frac{1}{2}(Y - XW)^2$$

$$W = (w_0, w_1, \ldots, w_m)^T$$

$$X = (x^0, x^1, \ldots, x^m)^T$$

权值更新算法：

$$W^{(k+1)} = W^{(k)} - \eta \frac{\partial Loss(W)}{\partial W}$$

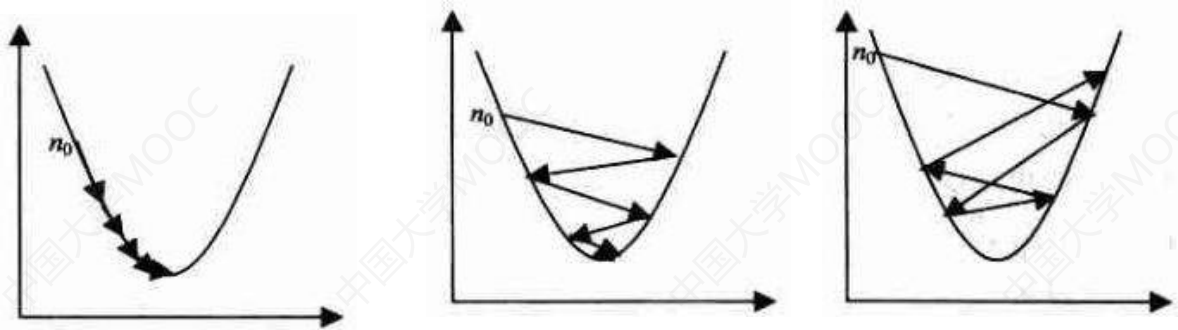
$$\frac{\partial Loss}{\partial W} = X^T(XW - Y)$$

$$W^{(k+1)} = W^{(k)} - \eta X^T(XW - Y)$$

学习率

$$w_i^{(k+1)} = w_i^{(k)} - \eta \frac{\partial Loss(\boldsymbol{w})}{\partial w_i}$$

对于**凸函数**，只要学习率设置的足够小，可以保证**一定收敛**



超参数：在开始**学习之前**设置，不是通过训练得到的

神经网络&深度学习
TENSORFLOW
Google

10.2.2 梯度下降法求解一元线性回归——NumPy实现

梯度下降法求解一元线性回归

平方损失函数：

$$Loss = \frac{1}{2}\sum_{i=1}^{n}(y_i - (wx_i + b))^2$$

模型参数更新算法：

$$w^{(k+1)} = w^{(k)} - \eta\sum_{i=1}^{n}x_i(wx_i + b - y_i)$$

$$b^{(k+1)} = b^{(k)} - \eta\sum_{i=1}^{n}(wx_i + b - y_i)$$

均方差损失函数：

$$Loss = \frac{1}{2n}\sum_{i=1}^{n}(y_i - (wx_i + b))^2$$

模型参数更新算法：

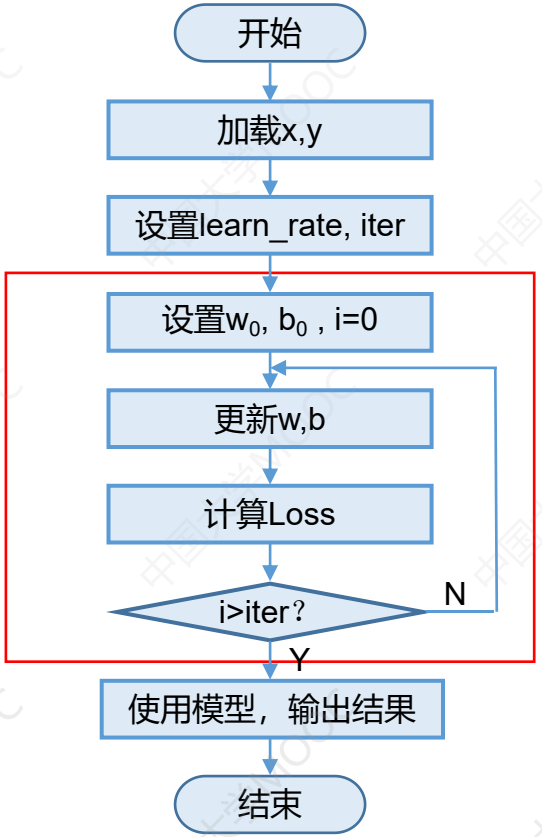
$$w^{(k+1)} = w^{(k)} - \frac{\eta}{n}\sum_{i=1}^{n}x_i(wx_i + b - y_i)$$

$$b^{(k+1)} = b^{(k)} - \frac{\eta}{n}\sum_{i=1}^{n}(wx_i + b - y_i)$$

商品房销售记录

序号	面积（平方米）	销售价格（万元）	序号	面积（平方米）	销售价格（万元）
1	137.97	145.00	9	106.69	62.00
2	104.50	110.00	10	138.05	133.00
3	100.00	93.00	11	53.75	51.00
4	124.32	116.00	12	46.91	45.00
5	79.20	65.32	13	68.00	78.50
6	99.00	104.00	14	63.02	69.65
7	124.00	118.00	15	81.26	75.69
8	114.00	91.00	16	86.21	95.30

- 加载样本数据 x, y
- 设置超参数 学习率，迭代次数
- 设置模型参数初值 w_0, b_0
- 训练模型 w, b
- 结果可视化



加载数据

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
x = np.array([137.97, 104.50, 100.00, 124.32, 79.20, 99.00, 124.00, 114.00,
              106.69, 138.05, 53.75, 46.91, 68.00, 63.02, 81.26, 86.21])
y = np.array([145.00, 110.00, 93.00, 116.00, 65.32, 104.00, 118.00, 91.00,
              62.00, 133.00, 51.00, 45.00, 78.50, 69.65, 75.69, 95.30])
```

- 设置**超参数**

```
In [3]: learn_rate=0.00001
        iter=100

        display_step=10
```

- 设置**模型参数初值**

```
In [4]: np.random.seed(612)
        w = np.random.randn()
        b = np.random.randn()
```

训练模型

```
In [5]: mse=[]

        for i in range(0, iter+1):

            dL_dw=np.mean(x*(w*x+b-y))
            dL_db=np.mean(w*x+b-y)

            w=w-learn_rate*dL_dw
            b=b-learn_rate*dL_db

            pred=w*x+b
            Loss= np.mean(np.square(y-pred))/2
            mse.append(Loss)

            if i % display_step == 0:
                print("i: %i, Loss:%f, w: %f, b: %f" % (i,mse[i], w, b))
```

$$\frac{\partial Loss}{\partial w}=\frac{1}{n}\sum_{i=1}^{n}x_i(wx_i+b-y_i)$$

$$\frac{\partial Loss}{\partial b}=\frac{1}{n}\sum_{i=1}^{n}(wx_i+b-y_i)$$

$$w^{(k+1)}=w^{(k)}-\eta\frac{\partial Loss}{\partial w}$$

$$b^{(k+1)}=b^{(k)}-\eta\frac{\partial Loss}{\partial b}$$

$$Loss=\frac{1}{2n}\sum_{i=1}^{n}(y_i-(wx_i+b))^2$$

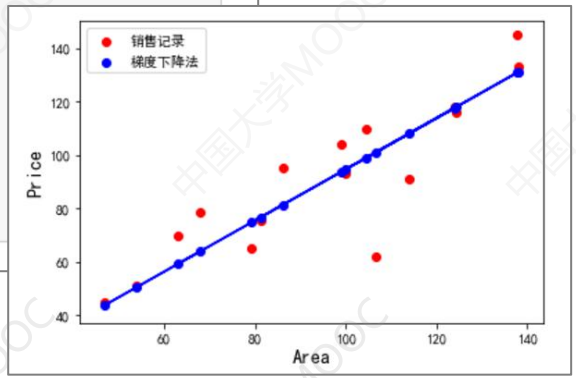
训练模型

```
In [5]: mse=[]

        for i in range(0, iter+1):

            dL_dw=np.mean(x*(w*x+b-y))
            dL_db=np.mean(w*x+b-y)

            w=w-learn_rate*dL_dw
            b=b-learn_rate*dL_db

            pred=w*x+b
            Loss= np.mean(np.square(y-pred))/2
            mse.append(Loss)

            if i % display_step == 0:
                print("i: %i, Loss:%f, w: %f, b: %f" % (i,mse[i], w, b))
```

```
i: 0, Loss:3874.243711, w: 0.082565, b: -1.161967
i: 10, Loss:562.072704, w: 0.648552, b: -1.156446
i: 20, Loss:148.244254, w: 0.848612, b: -1.154462
i: 30, Loss:96.539782, w: 0.919327, b: -1.153728
i: 40, Loss:90.079712, w: 0.944323, b: -1.153435
i: 50, Loss:89.272557, w: 0.953157, b: -1.153299
i: 60, Loss:89.171687, w: 0.956280, b: -1.153217
i: 70, Loss:89.159061, w: 0.957383, b: -1.153156
i: 80, Loss:89.157460, w: 0.957773, b: -1.153101
i: 90, Loss:89.157238, w: 0.957910, b: -1.153048
i: 100, Loss:89.157187, w: 0.957959, b: -1.152997
```

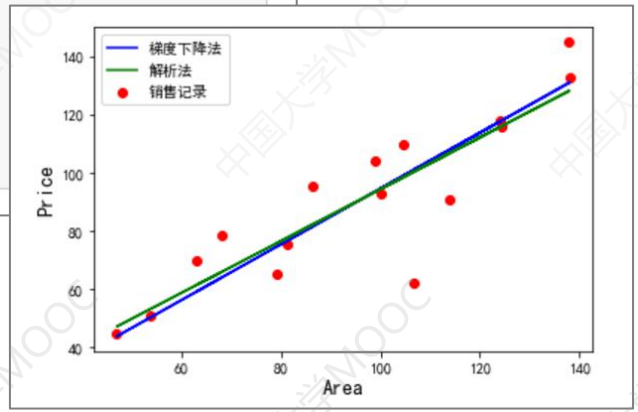
结果可视化——数据和模型

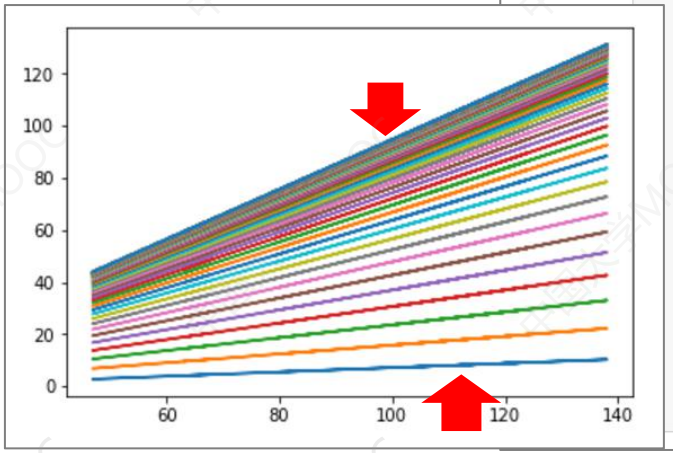
```
In [6]: plt.rcParams['font.sans-serif'] = ['SimHei']

        plt.figure()

        plt.scatter(x, y, color="red", label="销售记录")
        plt.scatter(x, pred, color="blue", label="梯度下降法")
        plt.plot(x, pred, color="blue")

        plt.xlabel("Area", fontsize=14)
        plt.ylabel("Price", fontsize=14)

        plt.legend(loc="upper left")
        plt.show()
```



解析解：w=0.8945604, b=5.4108505

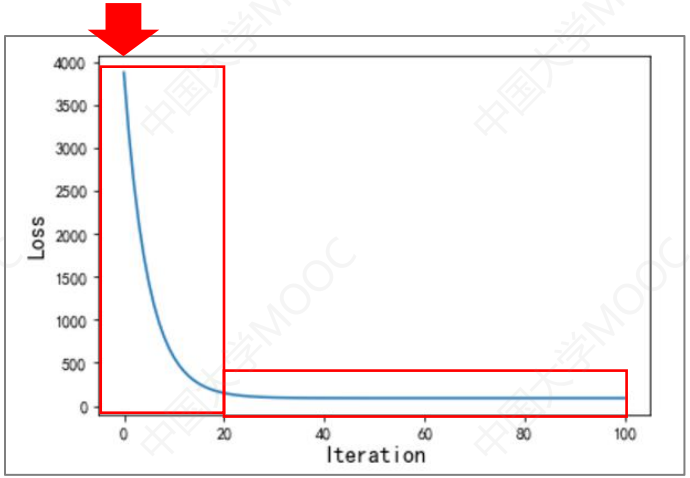
```
In [7]: plt.figure()

        plt.scatter(x, y, color="red", label="销售记录")
        plt.plot(x, pred, color="blue", label="梯度下降法")
        plt.plot(x, 0.89*x+5.41, color="green", label="解析法")

        plt.xlabel("Area", fontsize=14)
        plt.ylabel("Price", fontsize=14)

        plt.legend(loc="upper left")
        plt.show()
```






```
In [5]: mse=[]

for i in range(0, iter+1):

    dL_dw=np.mean(x*(w*x+b-y))
    dL_db=np.mean(w*x+b-y)

    w=w-learn_rate*dL_dw
    b=b-learn_rate*dL_db

    pred=w*x+b
    Loss= np.mean(np.square(y-pred))/2
    mse.append(Loss)

    plt.plot(x,pred)

    if i % display_step == 0:
        print("i: %i, Loss:%f, w: %f, b: %f" % (i,mse[i], w, b))
```

结果可视化——损失变化

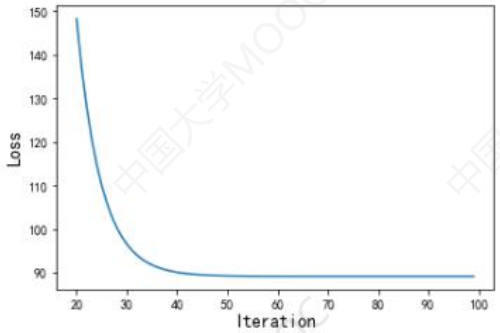


In [8]:

```
plt.figure()

plt.plot(mse)

plt.xlabel("Iteration",fontsize=14)
plt.ylabel("Loss",fontsize=14)

plt.show()
```

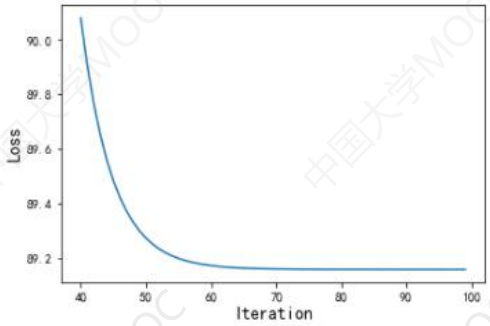
In [9]:

```
plt.figure()

plt.plot(range(20, 100), mse[20:100])

plt.xlabel("Iteration", fontsize=14)
plt.ylabel("Loss", fontsize=14)

plt.show()
```

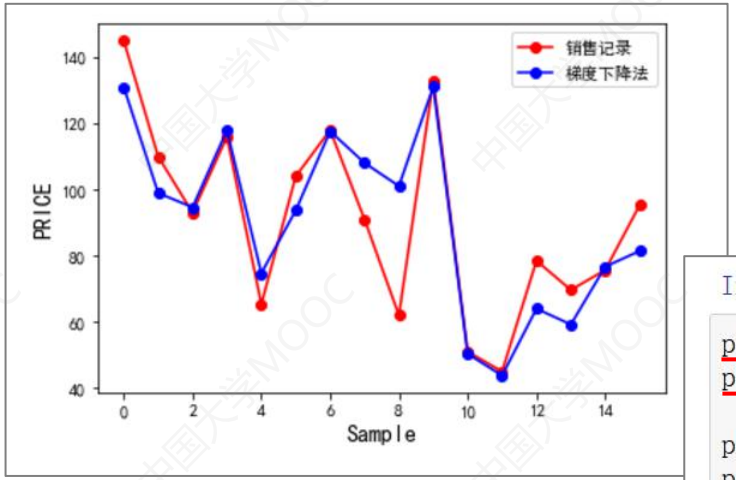


In [10]:

```
plt.figure()

plt.plot(range(40, 100), mse[40:100])

plt.xlabel("Iteration", fontsize=14)
plt.ylabel("Loss", fontsize=14)

plt.show()
```

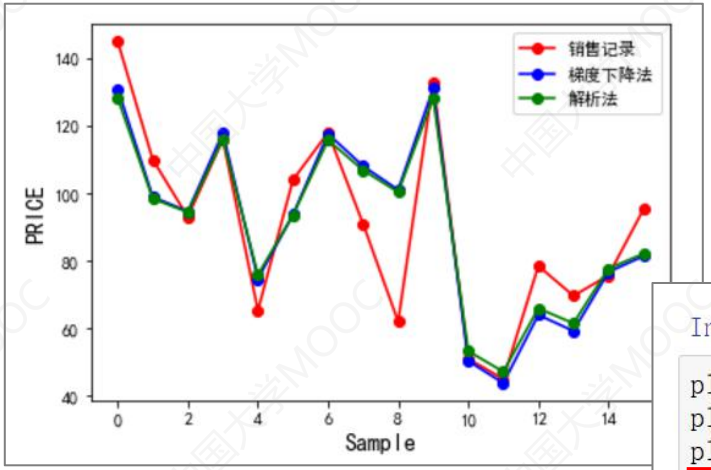


结果可视化——估计值 & 标签值



In [11]:

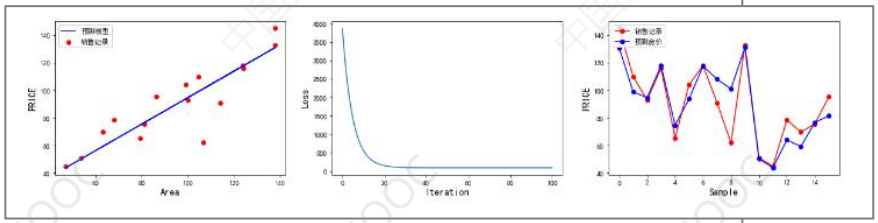
```
plt.plot(y, color="red", marker="o", label="销售记录")
plt.plot(pred, color="blue", marker="o", label="梯度下降法")

plt.legend()
plt.xlabel("Sample", fontsize=14)
plt.ylabel("PRICE", fontsize=14)
plt.show()
```



In [12]:

```
plt.plot(y, color="red", marker="o", label="销售记录")
plt.plot(pred, color="blue", marker="o", label="梯度下降法")
plt.plot(0.89*x+5.41, color="green", marker="o", label="解析法")

plt.legend()
plt.xlabel("Sample", fontsize=14)
plt.ylabel("PRICE", fontsize=14)
plt.show()
```

In [13]:
```
plt.rcParams['font.sans-serif'] = ['SimHei']

plt.figure(figsize=(20,4))

plt.subplot(1,3,1)
plt.scatter(x,y,color="red",label="销售记录")
plt.plot(x,pred,color="blue",label="预测模型")
plt.xlabel("Area",fontsize=14)
plt.ylabel("PRICE",fontsize=14)
plt.legend(loc="upper left")

plt.subplot(1,3,2)
plt.plot(mse)
plt.xlabel("Iteration",fontsize=14)
plt.ylabel("Loss",fontsize=14)

plt.subplot(1,3,3)
plt.plot(y,color="red",marker="o",label="销售记录")
plt.plot(pred,color="blue",marker="o",label="预测房价")
plt.legend()
plt.xlabel("Sample",fontsize=14)
plt.ylabel("PRICE",fontsize=14)
plt.legend(loc="upper left")

plt.show()
```



10.2.3 梯度下降法求解多元线性回归——NumPy实现

商品房销售记录

序号	面积（平方米）	房间数	销售价格（万元）	序号	面积（平方米）	房间数	销售价格（万元）
1	137.97	3	145.00	9	106.69	2	62.00
2	104.50	2	110.00	10	138.05	3	133.00
3	100.00	2	93.00	11	53.75	1	51.00
4	124.32	3	116.00	12	46.91	1	45.00
5	79.20	1	65.32	13	68.00	1	78.50
6	99.00	2	104.00	14	63.02	1	69.65
7	124.00	3	118.00	15	81.26	2	75.69
8	114.00	2	91.00	16	86.21	2	95.30

- **归一化 / 标准化**：将数据的**值**限制在**一定的范围**之内

使所有属性处于同一个**范围**、同一个**数量级**下

更快**收敛**到最优解

提高学习器的**精度**

线性归一化，标准差归一化，非线性映射归一化

- **线性归一化**：对原始数据的线性变换

$$x^* = \frac{x - \min}{\max - \min}$$

等比例缩放

所有的数据都被映射到[0,1]之间

- **标准差归一化**：将数据集归一化为**均值为0**，**方差为1**的标准正态分布

$$x^* = \frac{x - \mu}{\sigma}$$

- **非线性映射归一化**：对原始数据的**非线性**变换

指数、对数、正切

线性归一化

```
In [1]: import numpy as np

In [2]: area=np.array([137.97, 104.50, 100.00, 124.32, 79.20, 99.00, 124.00, 114.00,
                       106.69, 138.05, 53.75, 46.91, 68.00, 63.02, 81.26, 86.21])
        room=np.array([3, 2, 2, 3, 1, 2, 3, 2, 2, 3, 1, 1, 1, 1, 2, 2])

In [3]: x1=(area -area.min())/(area.max()-area.min())
        x2=(room -room.min())/(room.max()-room.min())

In [4]: x1, x2

Out[4]: (array([0.99912223, 0.63188501, 0.58251042, 0.84935264, 0.3542901 ,
                0.57153829, 0.84584156, 0.73612025, 0.65591398, 1.        ,
                0.07504937, 0.        , 0.23140224, 0.17676103, 0.37689269,
                0.43120474]),
         array([1. , 0.5, 0.5, 1. , 0. , 0.5, 1. , 0.5, 0.5, 1. , 0. , 0. , 0. ,
                0. , 0.5, 0.5]))
```

使用梯度下降法求解多元线性回归

- 加载样本数据 area, room, price
- 数据处理 归一化，X, Y
- 设置超参数 学习率，迭代次数
- 设置模型参数初值 W_0 (w_0,w_1,w_2)
- 训练模型 W

$$W^{(k+1)} = W^{(k)} - \eta X^T (XW - Y)$$

- 结果可视化

$$\frac{\partial Loss}{\partial W} = X^T (XW - Y)$$

$$W^{(k+1)} = W^{(k)} - \eta \frac{\partial Loss(W)}{\partial W}$$

加载样本数据

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']
```

In [2]:
```
area=np.array([137.97, 104.50, 100.00, 124.32, 79.20, 99.00, 124.00, 114.00,
               106.69, 138.05, 53.75, 46.91, 68.00, 63.02, 81.26, 86.21])
room=np.array([3, 2, 2, 3, 1, 2, 3, 2, 2, 3, 1, 1, 1, 1, 2, 2])
price = np.array([145.00, 110.00, 93.00, 116.00, 65.32, 104.00, 118.00, 91.00,
                  62.00, 133.00, 51.00, 45.00, 78.50, 69.65, 75.69, 95.30])
num = len(area)
```

数据处理

```
In [3]: x0 = np.ones(num)

        x1=(area -area.min())/(area.max()-area.min())
        x2=(room -room.min())/(room.max()-room.min())

        X =np.stack((x0,x1,x2), axis = 1)
        Y= price.reshape(-1,1)

In [4]: X.shape,Y.shape

Out[4]: ((16, 3), (16, 1))
```

设置超参数

```
In [5]: learn_rate=0.001
        iter=500

        display_step=50
```

设置模型参数初识值

```
In [6]: np.random.seed(612)
        W = np.random.randn(3,1)
```

训练模型

```
In [7]: mse=[]

        for i in range(0, iter+1):

            dL_dW= np.matmul(np.transpose(X),np.matmul(X,W)-Y)
            W=W-learn_rate*dL_dW

            PRED=np.matmul(X,W)
            Loss= np.mean(np.square(Y-PRED))/2
            mse.append(Loss)

            if i % display_step == 0:
                print("i: %i, Loss:%f" % (i,mse[i]))
```

$$\frac{\partial Loss}{\partial W}=X^T(XW-Y)$$

$$W^{(k+1)}=W^{(k)}-\eta\frac{\partial Loss(W)}{\partial W}$$

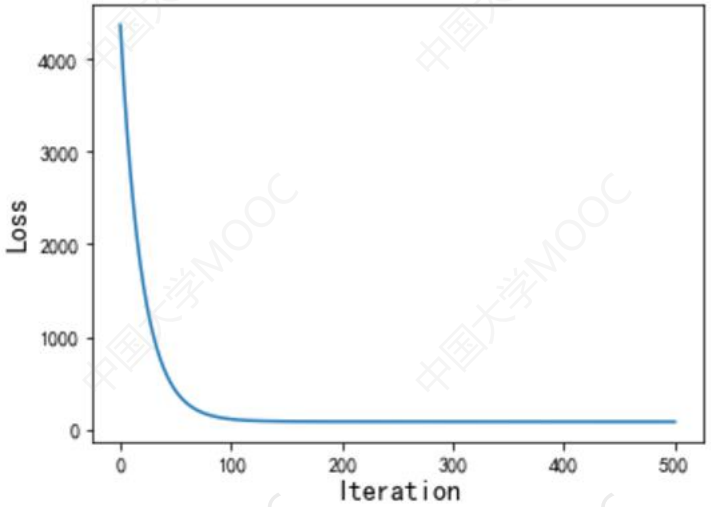
X (16,3) W (3,1) → (16,1)

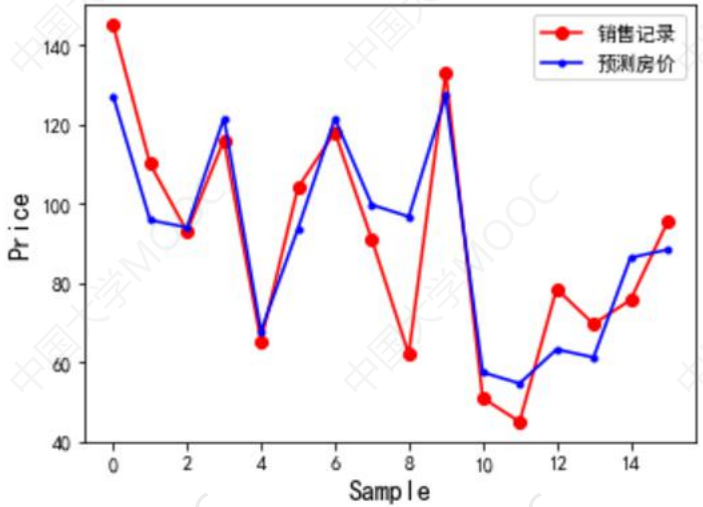
$$Loss=\frac{1}{2n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2$$

```
i: 0, Loss:4368.213908
i: 50, Loss:413.185263
i: 100, Loss:108.845176
i: 150, Loss:84.920786
i: 200, Loss:82.638199
i: 250, Loss:82.107310
i: 300, Loss:81.782545
i: 350, Loss:81.530512
i: 400, Loss:81.329266
i: 450, Loss:81.167833
i: 500, Loss:81.037990
```

结果可视化

```
In [8]: plt.figure(figsize=(12,4))

        plt.subplot(1,2,1)
        plt.plot(mse)
        plt.xlabel("Iteration",fontsize=14)
        plt.ylabel("Loss",fontsize=14)

        plt.subplot(1,2,2)
        PRED=PRED.reshape(-1)
        plt.plot(price,color="red",marker="o",label="销售记录")
        plt.plot(PRED,color="blue",marker=".",label="预测房价")
        plt.xlabel("Sample",fontsize=14)
        plt.ylabel("Price",fontsize=14)

        plt.legend()
        plt.show()
```


Loss
Iteration
4000
3000
2000
1000
0
0
100
200
300
400
500


销售记录
预测房价
Price
Sample
140
120
100
80
60
40
0
2
4
6
8
10
12
14

神经网络&深度学习
TENSORFLOW
Google

10.3 TensorFlow的自动求导机制

中国大学MOOC

10.3.1 TensorFlow的可训练变量

梯度下降法

损失函数

$$Loss = \frac{1}{2}\sum_{i=1}^{n}(y_i - (wx_i + b))^2$$

损失函数求导数

$$\frac{\partial Loss}{\partial w} = \frac{1}{n}\sum_{i=1}^{n} x_i(wx_i + b - y_i)$$

$$\frac{\partial Loss}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}(wx_i + b - y_i)$$

更新模型参数

$$w^{(k+1)} = w^{(k)} - \eta\frac{\partial Loss}{\partial w}$$

$$b^{(k+1)} = b^{(k)} - \eta\frac{\partial Loss}{\partial b}$$

- TensorFlow的**自动求导**机制
- **Variable**对象
  - 对Tensor对象的进一步封装
  - 在模型训练过程中**自动记录梯度信息**，由算法**自动优化**
  - **可以被训练**的变量
  - 在机器学习中作为**模型参数**

```
tf.Variable(initial_value,dtype)
```

数字
Python列表
ndarray对象
Tensor对象

创建可训练变量

```
In [1]: import tensorflow as tf
        print("TensorFlow version:", tf.__version__)

        TensorFlow version: 2.0.0

In [2]: import numpy as np

In [3]: tf.Variable(3)

Out[3]: <tf.Variable 'Variable:0' shape=() dtype=int32, numpy=3>

In [4]: tf.Variable([1,2])

Out[4]: <tf.Variable 'Variable:0' shape=(2,) dtype=int32, numpy=array([1, 2])>

In [5]: tf.Variable(np.array([1,2]))

Out[5]: <tf.Variable 'Variable:0' shape=(2,) dtype=int32, numpy=array([1, 2])>
```

```
tf.Variable(initial_value,dtype)
```

```
In [6]: tf.Variable(3.)
Out[6]: <tf.Variable 'Variable:0' shape=() dtype=float32, numpy=3.0>

In [7]: tf.Variable([1,2],dtype=tf.float64)
Out[7]: <tf.Variable 'Variable:0' shape=(2,) dtype=float64, numpy=array([1., 2.])>
```

- 将**张量**封装为**可训练变量**

```
In [8]: tf.Variable(tf.constant([[1,2],[3,4]]))
Out[8]: <tf.Variable 'Variable:0' shape=(2, 2) dtype=int32, numpy=
        array([[1, 2],
               [3, 4]])>

In [9]: tf.Variable(tf.zeros([2,3]))
Out[9]: <tf.Variable 'Variable:0' shape=(2, 3) dtype=float32, numpy=
        array([[0., 0., 0.],
               [0., 0., 0.]], dtype=float32)>

In [10]: tf.Variable(tf.random.normal([2,2]))
Out[10]: <tf.Variable 'Variable:0' shape=(2, 2) dtype=float32, numpy=
         array([[1.113321  , 0.79599154],
                [0.00233323, 0.33498392]], dtype=float32)>
```

- 使用**变量名**

```
In [11]: x=tf.Variable([1,2])

In [12]: x
Out[12]: <tf.Variable 'Variable:0' shape=(2,) dtype=int32, numpy=array([1, 2])>

In [13]: print(x.shape,x.dtype)
         (2,) <dtype: 'int32'>

In [14]: print(x.numpy())
         [1 2]
```

trainalbe属性

```
In [15]: x.trainable
Out[15]: True
```

ResourceVariable

```
In [16]: type(x)
Out[16]: tensorflow.python.ops.resource_variable_ops.ResourceVariable
```

可训练变量赋值

对象名.assign()　对象名.assign_add()　对象名.assign_sub()

```
In [17]: x=tf.Variable([1,2])

In [18]: x.assign([3,4])
Out[18]: <tf.Variable 'UnreadVariable' shape=(2,) dtype=int32, numpy=array([3, 4])>

In [19]: x.assign_add([1,1])
Out[19]: <tf.Variable 'UnreadVariable' shape=(2,) dtype=int32, numpy=array([4, 5])>

In [20]: x.assign_add([1,1])
Out[20]: <tf.Variable 'UnreadVariable' shape=(2,) dtype=int32, numpy=array([5, 6])>
```

```
In [21]: a=tf.constant(2)

In [22]: a.assign(3)
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-22-22a29cba3fee> in <module>
----> 1 a.assign(3)

AttributeError: 'tensorflow.python.framework.ops.EagerTensor' object has no attribute 'assign'

In [23]: a.trainable
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-23-d74019e38028> in <module>
----> 1 a.trainable

AttributeError: 'tensorflow.python.framework.ops.EagerTensor' object has no attribute 'trainable'
```

isinstance() 方法

```
In [22]: a=tf.range(5)

In [23]: x=tf.Variable(a)

In [24]: isinstance(a, tf.Tensor), isinstance(a, tf.Variable)
Out[24]: (True, False)

In [25]: isinstance(x, tf.Tensor), isinstance(x, tf.Variable)
Out[25]: (False, True)
```

10.3.2 TensorFlow的自动求导

自动求导——GradientTape

```
with GradientTape() as tape:
    函数表达式
grad=tape.gradient(函数,自变量)
```

$$y = x^2 \big|_{x=3}$$

$$y = 3^2 = 9$$

$$\frac{dy}{dx} = 2x = 6$$

```
In [26]: x = tf.Variable(3.)

In [27]: with tf.GradientTape() as tape:
             y = tf.square(x)

In [28]: dy_dx = tape.gradient(y, x)

In [29]: print(y)
         print(dy_dx)

tf.Tensor(9.0, shape=(), dtype=float32)
tf.Tensor(6.0, shape=(), dtype=float32)
```

```
GradientTape(persistent, watch_accessed_variables)
```

In [30]:

```
x = tf.Variable(3.)

with tf.GradientTape() as tape:
    y = tf.square(x)
    z = pow(x, 3)

dy_dx = tape.gradient(y, x)
dz_dx = tape.gradient(z, x)

print(y)
print(dy_dx)
print(z)
print(dz_dx)
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-30-5ace6fa66e0a> in <module>
      6
      7 dy_dx = tape.gradient(y, x)
----> 8 dz_dx = tape.gradient(z, x)
      9
     10 print(y)

D:\Anaconda3\lib\site-packages\tensorflow_core\python\eager\backprop.py in gradient(self, target, s
ources, output_gradients, unconnected_gradients)
    963     """
    964     if self._tape is None:
--> 965       raise RuntimeError("GradientTape.gradient can only be called once on "
    966                          "non-persistent tapes.")
    967     if self._recording:

RuntimeError: GradientTape.gradient can only be called once on non-persistent tapes.
```

```
In [31]: x = tf.Variable(3.)

         with tf.GradientTape(persistent=True) as tape:
             y = tf.square(x)
             z = pow(x, 3)

         dy_dx = tape.gradient(y, x)
         dz_dx = tape.gradient(z, x)

         print(y)
         print(dy_dx)
         print(z)
         print(dz_dx)

         del tape
```

```
tf.Tensor(9.0, shape=(), dtype=float32)
tf.Tensor(6.0, shape=(), dtype=float32)
tf.Tensor(27.0, shape=(), dtype=float32)
tf.Tensor(27.0, shape=(), dtype=float32)
```

$$z = 3^3 = 27$$

$$\frac{dz}{dx} = 3x^2 = 27$$

GradientTape(persistent,watch_accessed_variables)

In [32]:
```
x = tf.Variable(3.)

with tf.GradientTape(watch_accessed_variables=False) as tape:
    y = tf.square(x)

dy_dx = tape.gradient(y, x)

print(y)
print(dy_dx)
```

```
tf.Tensor(9.0, shape=(), dtype=float32)
None
```

添加监视——watch()

In [33]:
```
x = tf.Variable(3.)

with tf.GradientTape(watch_accessed_variables=False) as tape:
    tape.watch(x)
    y = tf.square(x)

dy_dx = tape.gradient(y,x)

print(y)
print(dy_dx)
```

```
tf.Tensor(9.0, shape=(), dtype=float32)
tf.Tensor(6.0, shape=(), dtype=float32)
```

监视非可训练变量

In [34]:
```
x = tf.constant(3.)

with tf.GradientTape(watch_accessed_variables=False) as tape:
    tape.watch(x)
    y = tf.square(x)

dy_dx = tape.gradient(y, x)

print(y)
print(dy_dx)
```

```
tf.Tensor(9.0, shape=(), dtype=float32)
tf.Tensor(6.0, shape=(), dtype=float32)
```

多元函数求偏导数

tape.gradient(函数, 自变量)

$$f(x,y)=x^2+2y^2+1$$

$$f(3,4)=42$$

$$\frac{\partial f(x,y)}{\partial x}|_{x=3}=2x=6$$

$$\frac{\partial f(x,y)}{\partial y}|_{y=4}=4y=16$$

In [35]:
```
x = tf.Variable(3.)
y = tf.Variable(4.)

with tf.GradientTape() as tape:
    f = tf.square(x)+2*tf.square(y)+1

df_dx, df_dy = tape.gradient(f, [x, y])

print(f)
print(df_dx)
print(df_dy)
```

```
tf.Tensor(42.0, shape=(), dtype=float32)
tf.Tensor(6.0, shape=(), dtype=float32)
tf.Tensor(16.0, shape=(), dtype=float32)
```

In [36]:
```
x = tf.Variable(3.)
y = tf.Variable(4.)

with tf.GradientTape() as tape:
    f = tf.square(x)+2*tf.square(y)+1

first_grads= tape.gradient(f,[x,y])

print(first_grads)
```

```
[<tf.Tensor: id=223, shape=(), dtype=float32, numpy=6.0>,
 <tf.Tensor: id=228, shape=(), dtype=float32, numpy=16.0
>]
```

In [37]:
```
x = tf.Variable(3.)
y = tf.Variable(4.)

with tf.GradientTape(persistent=True) as tape:
    f = tf.square(x)+2*tf.square(y)+1

df_dx = tape.gradient(f,x)
df_dy = tape.gradient(f,y)

print(f)
print(df_dx)
print(df_dy)

del tape
```

```
tf.Tensor(42.0, shape=(), dtype=float32)
tf.Tensor(6.0, shape=(), dtype=float32)
tf.Tensor(16.0, shape=(), dtype=float32)
```

求二阶导数

$$f(x,y)=x^2+2y^2+1$$

In [38]:
```
x = tf.Variable(3.)
y = tf.Variable(4.)

with tf.GradientTape(persistent=True) as tape2:

    with tf.GradientTape(persistent=True) as tape1:
        f = tf.square(x)+2*tf.square(y)+1

    first_grads = tape1.gradient(f, [x, y])

second_grads=[tape2.gradient(first_grads, [x, y])]

print(f)
print(first_grads)
print(second_grads)

del tape1
del tape2
```

$$\frac{\partial f(x,y)}{\partial x}|_{x=3}=2x=6$$

$$\frac{\partial f(x,y)}{\partial y}|_{y=4}=4y=16$$

$$\frac{\partial^2 f(x,y)}{\partial x^2}|_{x=3}=2$$

$$\frac{\partial^2 f(x,y)}{\partial y^2}|_{y=4}=4$$

```
tf.Tensor(42.0, shape=(), dtype=float32)
[<tf.Tensor: id=296, shape=(), dtype=float32, numpy=6.0>,
 <tf.Tensor: id=301, shape=(), dtype=float32, numpy=16.0>]
[[<tf.Tensor: id=308, shape=(), dtype=float32, numpy=2.0>,
 <tf.Tensor: id=309, shape=(), dtype=float32, numpy=4.0>]]
```

- 对向量求偏导

```
In [39]: x = tf.Variable([1.,2.,3.])
         y = tf.Variable([4.,5.,6.])

         with tf.GradientTape() as tape:
             f = tf.square(x)+2*tf.square(y)+1

         df_dx,df_dy = tape.gradient(f,[x,y])

         print(f)
         print(df_dx)
         print(df_dy)

         tf.Tensor([34. 55. 82.], shape=(3,), dtype=float32)
         tf.Tensor([2. 4. 6.], shape=(3,), dtype=float32)
         tf.Tensor([16. 20. 24.], shape=(3,), dtype=float32)
```

神经网络&深度学习
TENSORFLOW
Google

10.4 TensorFlow实现梯度下降法

中国大学MOOC

- **可训练**变量
  - **Variable**对象
  - 自动记录梯度信息
  - 由算法自动优化
- **GradientTape**——自动求导

```
with GradientTape() as tape:
    函数表达式
grad=tape.gradient(函数, 自变量)
```

```
tape.gradient(f, x)
```

```
tape.gradient(f, [x,y])
```

NumPy实现一元线性回归

加载数据

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

In [2]: x = np.array([137.97, 104.50, 100.00, 124.32, 79.20, 99.00, 124.00, 114.00,
                      106.69, 138.05, 53.75, 46.91, 68.00, 63.02, 81.26, 86.21])
        y = np.array([145.00, 110.00, 93.00, 116.00, 65.32, 104.00, 118.00, 91.00,
                      62.00, 133.00, 51.00, 45.00, 78.50, 69.65, 75.69, 95.30])
```

设置超参数

```
In [3]: learn_rate=0.00001
        iter=100

        display_step=10
```

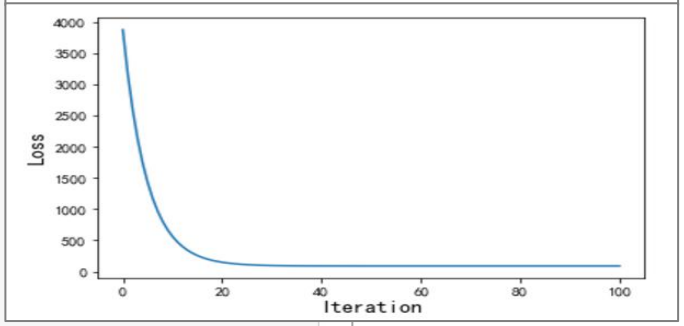
设置模型参数初值

```
In [4]: np.random.seed(612)
        w = np.random.randn()
        b = np.random.randn()
```

训练模型

```
In [5]: mse=[]

        for i in range(0, iter+1):

            dL_dw=np.mean(x*(w*x+b-y))
            dL_db=np.mean(w*x+b-y)

            w=w-learn_rate*dL_dw
            b=b-learn_rate*dL_db

            pred= w*x+b
            Loss= 0.5*np.mean(np.square(y-pred))
            mse.append(Loss)

            plt.plot(x,pred)

            if i % display_step == 0:
                print("i: %i, Loss:%f, w: %f, b: %f" % (i,mse[i], w, b))
```

```
i: 0, Loss:3874.243711, w: 0.082565, b: -1.161967
i: 10, Loss:562.072704, w: 0.648552, b: -1.156446
i: 20, Loss:148.244254, w: 0.848612, b: -1.154462
i: 30, Loss:96.539782, w: 0.919327, b: -1.153728
i: 40, Loss:90.079712, w: 0.944323, b: -1.153435
i: 50, Loss:89.272557, w: 0.953157, b: -1.153299
i: 60, Loss:89.171687, w: 0.956280, b: -1.153217
i: 70, Loss:89.159061, w: 0.957383, b: -1.153156
i: 80, Loss:89.157460, w: 0.957773, b: -1.153101
i: 90, Loss:89.157238, w: 0.957910, b: -1.153048
i: 100, Loss:89.157187, w: 0.957959, b: -1.152997
```

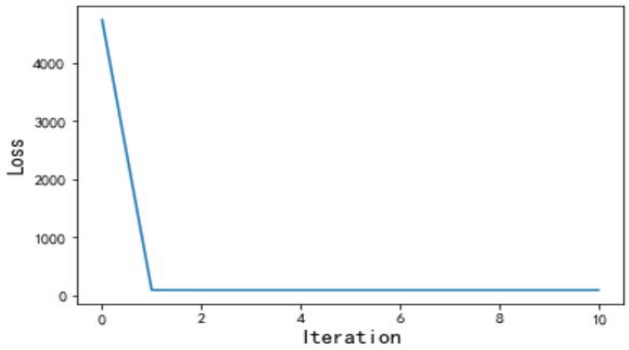


训练模型

```
In [3]: learn_rate=0.0001
        iter=10

        display_step=1
```

```
In [5]: mse=[]

        for i in range(0, iter+1):

            pred=w*x+b
            Loss= 0.5*np.mean(np.square(y-pred))
            mse.append(Loss)

            dL_dw=np.mean(x*(w*x+b-y))
            dL_db=np.mean(w*x+b-y)

            w=w-learn_rate*dL_dw
            b=b-learn_rate*dL_db

            if i % display_step == 0:
                print("i: %i, Loss:%f, w: %f, b: %f" % (i,mse[i], w, b))
```

```
i: 0, Loss:4749.362486, w: 0.946047, b: -1.153577
i: 1, Loss:89.861841, w: 0.957843, b: -1.153412
i: 2, Loss:89.157502, w: 0.957987, b: -1.153359
i: 3, Loss:89.157369, w: 0.957988, b: -1.153308
i: 4, Loss:89.157343, w: 0.957988, b: -1.153257
i: 5, Loss:89.157317, w: 0.957987, b: -1.153206
i: 6, Loss:89.157291, w: 0.957987, b: -1.153155
i: 7, Loss:89.157264, w: 0.957986, b: -1.153104
i: 8, Loss:89.157238, w: 0.957986, b: -1.153053
i: 9, Loss:89.157212, w: 0.957985, b: -1.153001
i: 10, Loss:89.157186, w: 0.957985, b: -1.152950
```



TensorFlow实现一元线性回归

In [1]:
```
import tensorflow as tf
print("TensorFlow version:",tf.__version__)
```

```
TensorFlow version: 2.0.0
```

In [2]:
```
import numpy as np
```

In [3]:
```
x = np.array([137.97,  104.50,  100.00,  124.32,  79.20,  99.00,  124.00,  114.00,
              106.69,  138.05,  53.75,   46.91,   68.00,  63.02,  81.26,   86.21])
y = np.array([145.00,  110.00,  93.00,  116.00,  65.32,  104.00,  118.00,  91.00,
              62.00,   133.00,  51.00,  45.00,   78.50,   69.65,  75.69,   95.30])
```

In [4]:
```
learn_rate = 0.0001
iter=10

display_step=1
```

In [5]:
```
np.random.seed(612)
w = tf.Variable(np.random.randn())
b = tf.Variable(np.random.randn())
```

In [6]:
```
mse=[]

for i in range(0,iter+1):

    with tf.GradientTape() as tape:
        pred = w*x+b
        Loss = 0.5*tf.reduce_mean(tf.square(y-pred))
    mse.append(Loss)

    dL_dw,dL_db = tape.gradient(Loss,[w,b])

    w.assign_sub(learn_rate*dL_dw)
    b.assign_sub(learn_rate*dL_db)

    if i % display_step == 0:
        print("i: %i,Loss: %f, w: %f, b: %f" % (i, Loss, w.numpy(), b.numpy()))
```

```
i: 0,Loss: 4749.362305, w: 0.946047, b: -1.153577
i: 1,Loss: 89.861855, w: 0.957843, b: -1.153412
i: 2,Loss: 89.157501, w: 0.957987, b: -1.153359
i: 3,Loss: 89.157379, w: 0.957988, b: -1.153308
i: 4,Loss: 89.157372, w: 0.957988, b: -1.153257
i: 5,Loss: 89.157318, w: 0.957987, b: -1.153206
i: 6,Loss: 89.157288, w: 0.957987, b: -1.153155
i: 7,Loss: 89.157265, w: 0.957986, b: -1.153104
i: 8,Loss: 89.157219, w: 0.957986, b: -1.153052
i: 9,Loss: 89.157211, w: 0.957985, b: -1.153001
i: 10,Loss: 89.157196, w: 0.957985, b: -1.152950
```

NumPy实现多元线性回归

加载样本数据

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']
```

In [2]:
```
area=np.array([137.97, 104.50, 100.00, 124.32, 79.20, 99.00, 124.00, 114.00,
               106.69, 138.05, 53.75, 46.91, 68.00, 63.02, 81.26, 86.21])
room=np.array([3, 2, 2, 3, 1, 2, 3, 2, 2, 3, 1, 1, 1, 1, 2, 2])
price = np.array([145.00, 110.00, 93.00, 116.00, 65.32, 104.00, 118.00, 91.00,
                  62.00, 133.00, 51.00, 45.00, 78.50, 69.65, 75.69, 95.30])
num = len(area)
```

In [3]:
```
x0 = np.ones(num)

x1=(area -area.min())/(area.max()-area.min())
x2=(room -room.min())/(room.max()-room.min())

X =np.stack((x0,x1,x2), axis = 1)
Y= price.reshape(-1,1)
```

In [4]:
```
learn_rate=0.01
iter=50

display_step=10
```

In [5]:
```
np.random.seed(612)
W = np.random.randn(3,1)
```

```
In [6]: mse=[]

        for i in range(0, iter+1):

            PRED = np.matmul(X,W)
            Loss = 0.5*np.mean(np.square(Y-PRED))
            mse.append(Loss)

            dL_dW= np.matmul(np.transpose(X),np.matmul(X,W)-Y)
            W=W-learn_rate*dL_dW

            if i % display_step == 0:
                print("i: %i, Loss:%f" % (i,mse[i]))
```

```
i: 0, Loss:4368.213908
i: 50, Loss:413.185263
i: 100, Loss:108.845176
i: 150, Loss:84.920786
i: 200, Loss:82.638199
i: 250, Loss:82.107310
i: 300, Loss:81.782545
i: 350, Loss:81.530512
i: 400, Loss:81.329266
i: 450, Loss:81.167833
i: 500, Loss:81.037990
```

训练模型

In [6]:
```
mse=[]

for i in range(0, iter+1):

    PRED = np.matmul(X,W)
    Loss = 0.5*np.mean(np.square(Y-PRED))
    mse.append(Loss)

    dL_dW= np.matmul(np.transpose(X),np.matmul(X,W)-Y)
    W=W-learn_rate*dL_dW

    if i % display_step == 0:
        print("i: %i, Loss:%f" % (i,mse[i]))
```

$$Loss=\frac{1}{2n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2$$

$$\frac{\partial Loss}{\partial W}=\frac{1}{n}X^T(XW-Y)=0$$

In [4]:
```
learn_rate=0.2
iter=50

display_step=10
```

In [6]:
```
mse=[]

for i in range(0, iter+1):

    PRED = np.matmul(X,W)
    Loss = 0.5*np.mean(np.square(Y-PRED))
    mse.append(Loss)

    dL_dW= np.matmul(np.transpose(X),np.matmul(X,W)-Y)/num
    W=W-learn_rate*dL_dW

    if i % display_step == 0:
        print("i: %i, Loss:%f" % (i,mse[i]))
```

```
i: 0, Loss:4593.851656
i: 50, Loss:264.489194
i: 100, Loss:90.497556
i: 150, Loss:82.899697
i: 200, Loss:82.113957
i: 250, Loss:81.718824
i: 300, Loss:81.427920
i: 350, Loss:81.207633
i: 400, Loss:81.040008
i: 450, Loss:80.911879
i: 500, Loss:80.813389
```

- TensorFlow实现**多元线性回归**

In [1]:
```
import tensorflow as tf
print("TensorFlow version:",tf.__version__)
```

```
TensorFlow version: 2.0.0
```

In [2]:
```
import numpy as np
```

In [3]:
```
area=np.array([137.97, 104.50, 100.00, 124.32, 79.20, 99.00, 124.00, 114.00,
               106.69, 138.05, 53.75, 46.91, 68.00, 63.02, 81.26, 86.21])
room=np.array([3, 2, 2, 3, 1, 2, 3, 2, 2, 3, 1, 1, 1, 1, 2, 2])
price = np.array([145.00, 110.00, 93.00, 116.00, 65.32, 104.00, 118.00, 91.00,
                  62.00, 133.00, 51.00, 45.00, 78.50, 69.65, 75.69, 95.30])
num = len(area)
```

In [4]:
```
x0 = np.ones(num)

x1=(area -area.min())/(area.max()-area.min())
x2=(room -room.min())/(room.max()-room.min())

X =np.stack((x0,x1,x2), axis = 1)
Y= price.reshape(-1,1)
```

In [5]:
```
learn_rate=0.2
iter=50

display_step=10
```

In [6]:
```
np.random.seed(612)
W =tf.Variable(np.random.randn(3,1))
```

In [7]:
```
mse=[]

for i in range(0,iter+1):

    with tf.GradientTape() as tape:
        PRED=tf.matmul(X,W)
        Loss=0.5* tf.reduce_mean(tf.square(Y-PRED))
    mse.append(Loss)


    dL_dW = tape.gradient(Loss,W)
    W.assign_sub(learn_rate*dL_dW)

    if i % display_step == 0:
        print("i: %i, Loss: %f" % (i, Loss))
```

```
i: 0, Loss: 4593.851656
i: 10, Loss: 85.480869
i: 20, Loss: 82.080953
i: 30, Loss: 81.408948
i: 40, Loss: 81.025841
i: 50, Loss: 80.803450
```


tensorflow / tensorflow
TensorFlow: Open sou
Clone this repo:
git clone https://tensor
Branches
master

10.5 模型评估

模型评估

误差 (error)：学习器的**预测输出**和样本的**真实标记**之间的差异

训练误差 (traning error): **训练集**上的误差

泛化误差 (generalization error): 在**新样本**上的误差

过拟合 (overfitting)：**学习过度**，在**训练集**上表现很好，在**新样本**上**泛化误差很大**

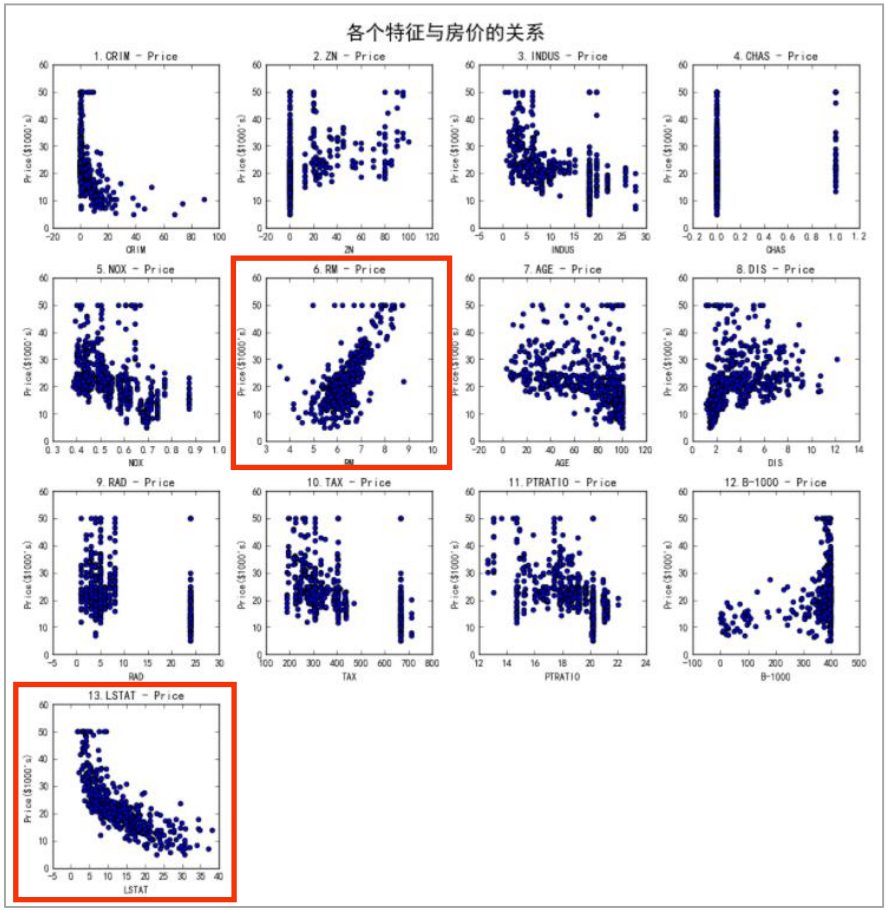
欠拟合 (underfitting): **学习不足**，没有学习到样本中的通用的特征

- 机器学习的目标：**泛化误差**小
- **训练集**和**测试集**
  - 训练集（training set）：训练模型
  - 测试集（testing set)：测试学习器在新样本上的预测或判断能力
  - 测试误差（testing error）：用来**近似**泛化误差

10.6 实例：波士顿房价预测

10.6.1 实例：波士顿房价预测(1)
一元线性回归

- 波士顿房价数据集


各个特征与房价的关系
1. CRIM - Price
2. ZN - Price
3. INDUS - Price
4. CHAS - Price
5. NOX - Price
6. RM - Price
7. AGE - Price
8. DIS - Price
9. RAD - Price
10. TAX - Price
11. PTRATIO - Price
12. B-1000 - Price
13. LSTAT - Price
Price($1000's)
CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PTRATIO
B-1000
LSTAT

加载数据集

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

In [2]:
```
boston_housing = tf.keras.datasets.boston_housing
(train_x, train_y), (test_x, test_y) = boston_housing.load_data()
```

In [3]:
```
train_x.shape, train_y.shape
```

Out[3]:
```
((404, 13), (404,))
```

In [4]:
```
test_x.shape, test_y.shape
```

Out[4]:
```
((102, 13), (102,))
```

数据处理

```
In [5]: x_train=train_x[:,5]
        y_train=train_y

In [6]: x_train.shape, y_train.shape
Out[6]: ((404,), (404,))

In [7]: x_test=test_x[:,5]
        y_test=test_y

In [8]: x_test.shape, y_test.shape
Out[8]: ((102,), (102,))
```

- 设置超参数

```
In [9]: learn_rate = 0.04
        iter = 2000
        display_step =200
```

- 设置模型参数初始值

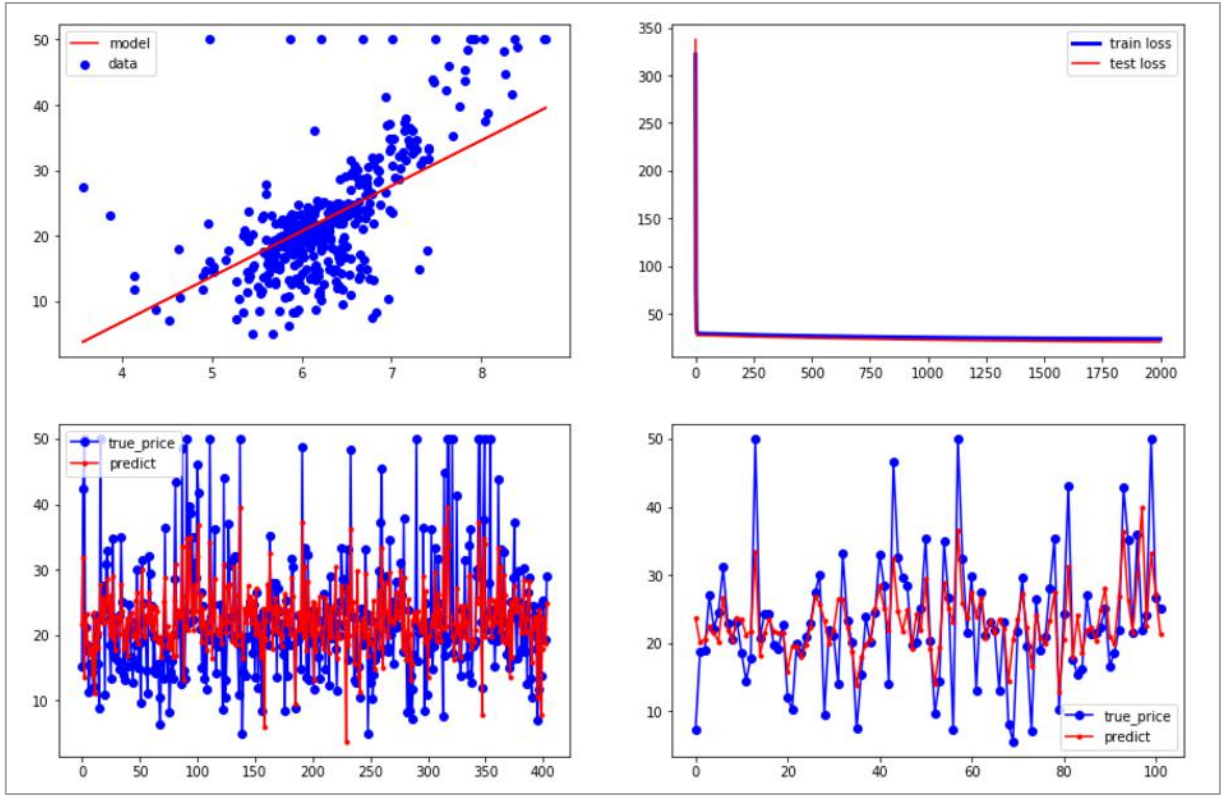
```
In [10]: np.random.seed(612)
         w = tf.Variable(np.random.randn())
         b = tf.Variable(np.random.randn())

In [11]: w.numpy().dtype, b.numpy().dtype
Out[11]: (dtype('float32'), dtype('float32'))
```

- 训练模型

In [12]:
```
mse_train=[]
mse_test=[]

for i in range(0,iter+1):

    with tf.GradientTape() as tape:

        pred_train = w*x_train+b
        loss_train = 0.5*tf.reduce_mean(tf.square(y_train-pred_train))

        pred_test = w*x_test+b
        loss_test = 0.5*tf.reduce_mean(tf.square(y_test-pred_test))

    mse_train.append(loss_train)
    mse_test.append(loss_test)

    dL_dw,dL_db = tape.gradient(loss_train,[w,b])
    w.assign_sub(learn_rate*dL_dw)
    b.assign_sub(learn_rate*dL_db)

    if i % display_step == 0:
        print("i: %i, Train Loss: %f, Test Loss: %f" % (i, loss_train,loss_test))
```

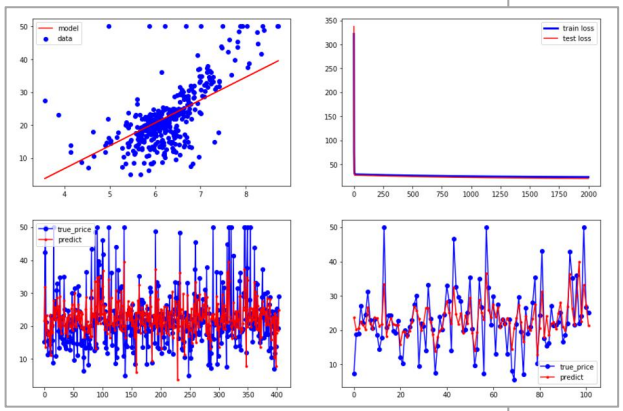
```
i: 0, Train Loss: 321.837585, Test Loss: 337.568665
i: 200, Train Loss: 28.122614, Test Loss: 26.237764
i: 400, Train Loss: 27.144741, Test Loss: 25.099329
i: 600, Train Loss: 26.341951, Test Loss: 24.141077
i: 800, Train Loss: 25.682898, Test Loss: 23.332981
i: 1000, Train Loss: 25.141848, Test Loss: 22.650158
i: 1200, Train Loss: 24.697674, Test Loss: 22.072004
i: 1400, Train Loss: 24.333027, Test Loss: 21.581432
i: 1600, Train Loss: 24.033665, Test Loss: 21.164263
i: 1800, Train Loss: 23.787907, Test Loss: 20.808695
i: 2000, Train Loss: 23.586145, Test Loss: 20.504940
```

- 可视化输出


model
data
train loss
test loss
true_price
predict
true_price
predict

10.6.1 波士顿房价预测：一元线性回归

In [13]:
```
plt.figure(figsize=(15, 10))

plt.subplot(221)
plt.scatter(x_train, y_train, color="blue", label="data")
plt.plot(x_train, pred_train, color="red", label="model")
plt.legend(loc="upper left")

plt.subplot(222)
plt.plot(mse_train, color="blue", linewidth=3, label="train loss")
plt.plot(mse_test, color="red", linewidth=1.5, label="test loss")
plt.legend(loc="upper right")

plt.subplot(223)
plt.plot(y_train, color="blue", marker="o", label="true_price")
plt.plot(pred_train, color="red", marker=".", label="predict")
plt.legend()

plt.subplot(224)
plt.plot(y_test, color="blue", marker="o", label="true_price")
plt.plot(pred_test, color="red", marker=".", label="predict")
plt.legend()

plt.show()
```



神经网络&深度学习 Google TENSORFLOW

10.6.2 实例：波士顿房价预测(2)
多元线性回归

中国大学MOOC

波士顿房价数据集

序号	变量名	说 明	示 例
1	CRIM	城镇人均犯罪率	0.00632
2	ZN	超过25000平方英尺的住宅用地所占比例	18.0
3	INDUS	城镇非零售业的商业用地所占比例	2.31
4	CHAS	是否被Charles河流穿过（取值1：是；取值0：否）	0
5	NOX	一氧化碳浓度	0.538
6	RM	每栋住宅的平均房间数	6.575
7	AGE	早于1940年建成的自住房屋比例	65.2
8	DIS	到波士顿5个中心区域的加权平均距离	4.0900
9	RAD	到达高速公路的便利指数	1
10	TAX	每10000美元的全值财产税率	296
11	PTRATIO	城镇中师生比例	15.3
12	B	反映城镇中的黑人比例的指标，越靠近0.63越小；$B=1000*(BK-0.63)^2$，其中BK是黑人的比例。	396.90
13	LSTAT	低收入人口的比例	7.68
14	MEDV	自住房屋房价的平均房价（单位为1000美元）	24.0

一维数组归一化

```
In [1]: import numpy as np

In [2]: area=np.array([137.97, 104.50, 100.00, 124.32, 79.20, 99.00, 124.00, 114.00,
                       106.69, 138.05, 53.75, 46.91, 68.00, 63.02, 81.26, 86.21])
        room=np.array([3, 2, 2, 3, 1, 2, 3, 2, 2, 3, 1, 1, 1, 1, 2, 2])

In [3]: x1=(area -area.min())/(area.max()-area.min())
        x2=(room -room.min())/(room.max()-room.min())

In [4]: x1, x2

Out[4]: (array([0.99912223, 0.63188501, 0.58251042, 0.84935264, 0.3542901 ,
                0.57153829, 0.84584156, 0.73612025, 0.65591398, 1.        ,
                0.07504937, 0.        , 0.23140224, 0.17676103, 0.37689269,
                0.43120474]),
         array([1. , 0.5, 0.5, 1. , 0. , 0.5, 1. , 0.5, 0.5, 1. , 0. , 0. , 0. ,
                0. , 0.5, 0.5]))
```

二维数组归一化——循环实现

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

In [2]:
```
boston_housing = tf.keras.datasets.boston_housing
(train_x, train_y), (test_x, test_y) = boston_housing.load_data()
```

In [3]:
```
train_x.shape, train_y.shape
```

Out[3]:
```
((404, 13), (404,))
```

In [4]:
```
test_x.shape, test_y.shape
```

Out[4]:
```
((102, 13), (102,))
```

```
In [1]: import numpy as np

In [2]: x = np.array([[3.,  10,  500],
                      [2.,  20,  200],
                      [1.,  30,  300],
                      [5.,  50,  100]])

In [3]: x.dtype, x.shape
Out[3]: (dtype('float64'), (4, 3))

In [4]: len(x)
Out[4]: 4

In [5]: x.shape[0], x.shape[1]
Out[5]: (4, 3)
```

```
In [5]: x.shape[0],x.shape[1]

Out[5]: (4, 3)

In [6]: for i in range(x.shape[1]):
            x[:,i]=(x[:,i]-x[:,i].min())/(x[:,i].max()-x[:,i].min())

In [7]: x

Out[7]: array([[0.5 , 0.  , 1.  ],
               [0.25, 0.25, 0.25],
               [0.  , 0.5 , 0.5 ],
               [1.  , 1.  , 0.  ]])
```

二维数组归一化——广播运算

```
In [8]: x = np.array([[3.,  10.,  500.],
                      [2.,  20.,  200.],
                      [1.,  30.,  300.],
                      [5.,  50.,  100.]])

In [9]: x.min(axis=0)
Out[9]: array([  1.,  10., 100.])

In [10]: x.max(axis=0)
Out[10]: array([  5.,  50., 500.])

In [11]: x.max(axis=0)- x.min(axis=0)
Out[11]: array([  4.,  40., 400.])
```

```
In [11]: x.max(axis=0)- x.min(axis=0)
Out[11]: array([  4.,   40., 400.])

In [12]: x-x.min(axis=0)
Out[12]: array([[  2.,   0., 400.],
                [  1.,  10., 100.],
                [  0.,  20., 200.],
                [  4.,  40.,   0.]])

In [13]: (x-x.min(axis=0)) /(x.max(axis=0)- x.min(axis=0))
Out[13]: array([[0.5 , 0.  , 1.  ],
                [0.25, 0.25, 0.25],
                [0.  , 0.5 , 0.5 ],
                [1.  , 1.  , 0.  ]])
```

波士顿房价数据多元线性回归

- 加载数据集

```
In [1]: import tensorflow as tf
        import numpy as np
        import matplotlib.pyplot as plt

In [2]: boston_housing = tf.keras.datasets.boston_housing
        (train_x, train_y), (test_x, test_y) = boston_housing.load_data()

In [3]: train_x.shape, train_y.shape
Out[3]: ((404, 13), (404,))

In [4]: test_x.shape, test_y.shape
Out[4]: ((102, 13), (102,))

In [5]: num_train=len(train_x)
        num_test=len(test_x)
```

数据处理

In [6]:
```
x_train=(train_x-train_x.min(axis=0))/(train_x.max(axis=0)- train_x.min(axis=0))
y_train=train_y

x_test=(test_x-test_x.min(axis=0))/(test_x.max(axis=0)- test_x.min(axis=0))
y_test=test_y
```

In [7]:
```
x0_train = np.ones(num_train).reshape(-1,1)
x0_test = np.ones(num_test).reshape(-1,1)
```

In [8]:
```
X_train=tf.cast(tf.concat([x0_train,x_train],axis=1),tf.float32)
X_test=tf.cast(tf.concat([x0_test,x_test],axis=1),tf.float32)
```

In [9]:
```
X_train.shape,X_test.shape
```

Out[9]: (TensorShape([404, 14]), TensorShape([102, 14]))

```
In [10]: Y_train=tf.constant(y_train.reshape(-1, 1),tf.float32)
         Y_test=tf.constant(y_test.reshape(-1, 1),tf.float32)

In [11]: Y_train.shape,Y_test.shape

Out[11]: (TensorShape([404, 1]), TensorShape([102, 1]))
```

```
In [10]: Y_train=tf.constant(y_train.reshape(-1, 1),tf.float32)
         Y_test=tf.constant(y_test.reshape(-1, 1),tf.float32)

In [11]: Y_train.shape,Y_test.shape

Out[11]: (TensorShape([404, 1]), TensorShape([102, 1]))
```

- 设置超参数

```
In [12]: learn_rate = 0.01
         iter= 2000
         display_step =200
```

- 设置模型变量初始值

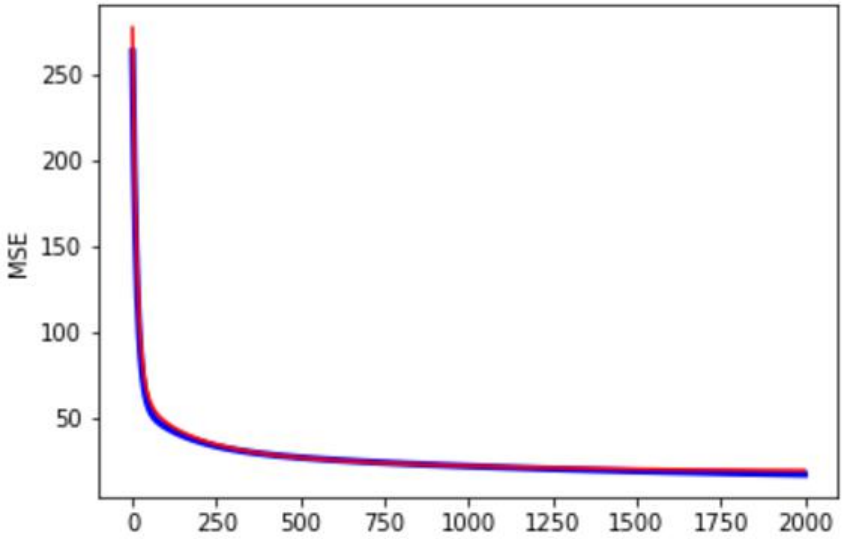
```
In [13]: np.random.seed(612)
         W = tf.Variable(np.random.randn(14,1),dtype=tf.float32)
```

训练模型

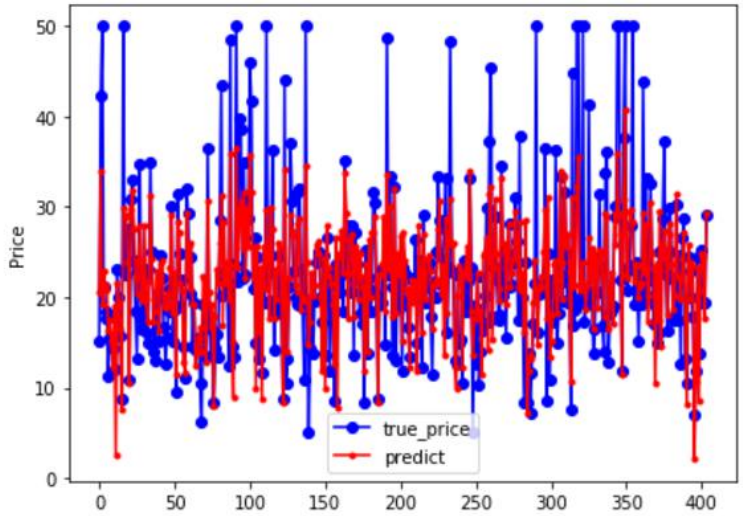
```
mse_train=[]
mse_test=[]

for i in range(0,iter+1):

    with tf.GradientTape() as tape:

        PRED_train=tf.matmul(X_train,W)
        Loss_train=0.5* tf.reduce_mean(tf.square(Y_train-PRED_train))

        PRED_test=tf.matmul(X_test,W)
        Loss_test=0.5* tf.reduce_mean(tf.square(Y_test-PRED_test))

    mse_train.append(Loss_train)
    mse_test.append(Loss_test)

    dL_dW = tape.gradient(Loss_train,W)
    W.assign_sub(learn_rate*dL_dW)

    if i % display_step == 0:
        print("i: %i, Train Loss: %f, Test Loss: %f " % (i, Loss_train,Loss_test))
```

```
i: 0, Train Loss: 263.193451, Test Loss: 276.994110
i: 200, Train Loss: 36.176552, Test Loss: 37.562954
i: 400, Train Loss: 28.789461, Test Loss: 28.952513
i: 600, Train Loss: 25.520697, Test Loss: 25.333916
i: 800, Train Loss: 23.460522, Test Loss: 23.340532
i: 1000, Train Loss: 21.887278, Test Loss: 22.039747
i: 1200, Train Loss: 20.596283, Test Loss: 21.124847
i: 1400, Train Loss: 19.510204, Test Loss: 20.467239
i: 1600, Train Loss: 18.587009, Test Loss: 19.997717
i: 1800, Train Loss: 17.797461, Test Loss: 19.671591
i: 2000, Train Loss: 17.118927, Test Loss: 19.456863
```

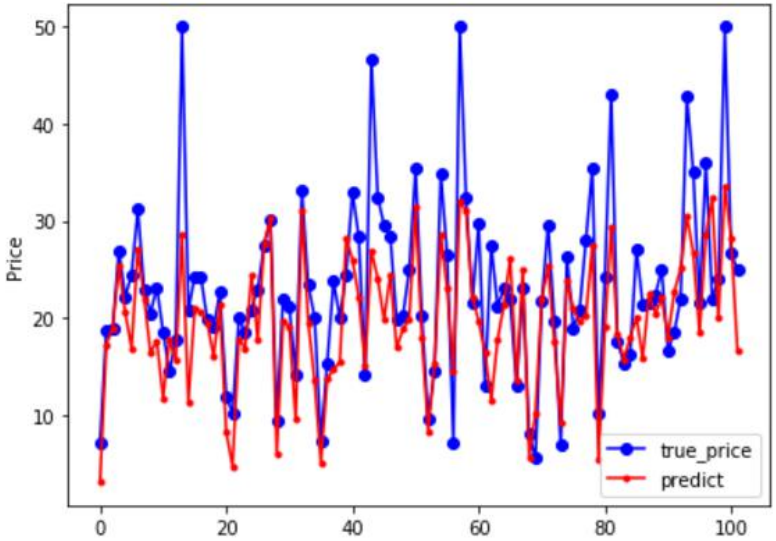
- 可视化输出




```
plt.plot(mse_train,color="blue",linewidth=3)
plt.plot(mse_test,color="red",linewidth=1.5)
```



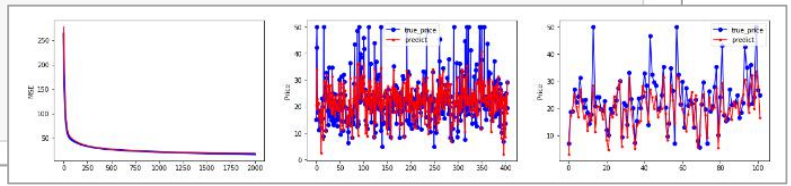

```
plt.plot(y_train,color="blue",marker="o",label="true_price")
plt.plot(PRED_train,color="red",marker=".",label="predict")
```




```
plt.plot(y_test,color="blue",marker="o",label="true_price")
plt.plot(PRED_test,color="red",marker=".",label="predict")
```

- 可视化输出

```
plt.figure(figsize=(20,4))

plt.subplot(131)
plt.ylabel("MSE")
plt.plot(mse_train,color="blue",linewidth=3)
plt.plot(mse_test,color="red",linewidth=1.5)

plt.subplot(132)
plt.plot(y_train,color="blue",marker="o",label="true_price")
plt.plot(PRED_train,color="red",marker=".",label="predict")
plt.legend()
plt.ylabel("Price")

plt.subplot(133)
plt.plot(y_test,color="blue",marker="o",label="true_price")
plt.plot(PRED_test,color="red",marker=".",label="predict")
plt.legend()
plt.ylabel("Price")

plt.show()
```



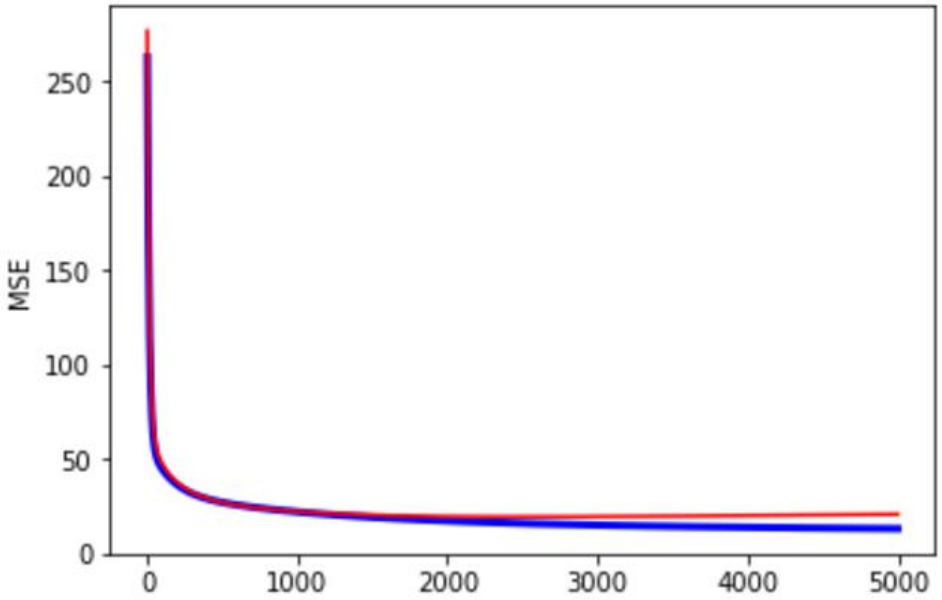
```
learn_rate = 0.01
iter= 8000
display_step =500
```

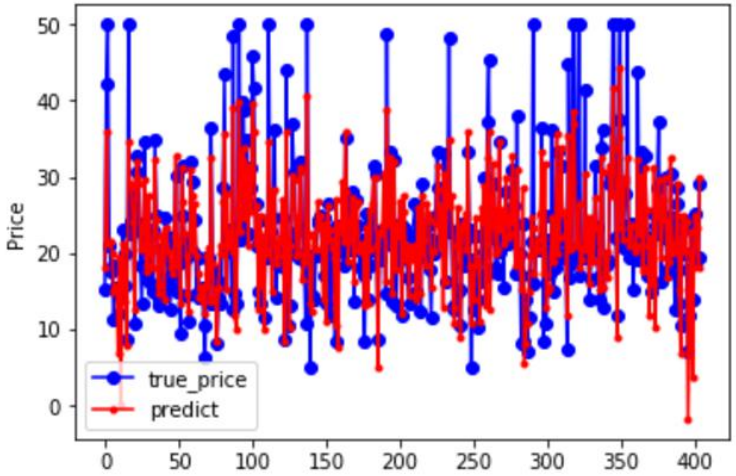
iter=**3000**
display=**100**

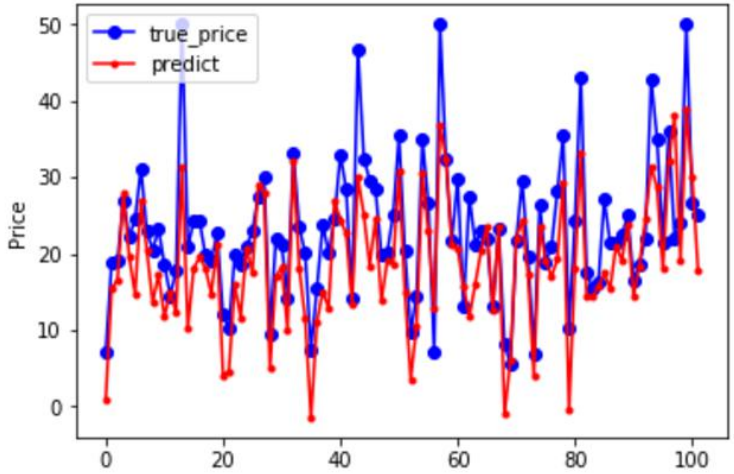
```
i: 0, Train Loss: 263.193451, Test Loss: 276.994110
i: 500, Train Loss: 26.911528, Test Loss: 26.827421
i: 1000, Train Loss: 21.887278, Test Loss: 22.039747
i: 1500, Train Loss: 19.030268, Test Loss: 20.212141
i: 2000, Train Loss: 17.118927, Test Loss: 19.456863
i: 2500, Train Loss: 15.797002, Test Loss: 19.260986
i: 3000, Train Loss: 14.858858, Test Loss: 19.365532
i: 3500, Train Loss: 14.177205, Test Loss: 19.623526
i: 4000, Train Loss: 13.671042, Test Loss: 19.949772
i: 4500, Train Loss: 13.287543, Test Loss: 20.295109
i: 5000, Train Loss: 12.991438, Test Loss: 20.631866
i: 5500, Train Loss: 12.758677, Test Loss: 20.945160
i: 6000, Train Loss: 12.572536, Test Loss: 21.227777
i: 6500, Train Loss: 12.421189, Test Loss: 21.477072
i: 7000, Train Loss: 12.296155, Test Loss: 21.693033
i: 7500, Train Loss: 12.191256, Test Loss: 21.877157
i: 8000, Train Loss: 12.101961, Test Loss: 22.031693
```

10.6.2 波士顿房价预测：多元线性回归

```
learn_rate = 0.01
iter= 8000
display_step =500
```




Price
true_price
predict
0
10
20
30
40
50
0
50
100
150
200
250
300
350
400


Price
true_price
predict
0
10
20
30
40
50
0
20
40
60
80
100