

# Regression: linear model

这里用的是 Adagrad，接下来的课程会再细讲，这里只是想显示 gradient descent 实作起来没有想像的那么简单，还有很多小技巧要注意

这里采用最简单的linear model:  $y\_data=b+w*x\_data$

我们要用gradient descent把b和w找出来

当然这个问题有closed-form solution，这个b和w有更简单的方法可以找出来；那我们假装不知道这件事，我们练习用gradient descent把b和w找出来

**数据准备：**

```
1 # 假设x_data和y_data都有10笔，分别代表宝可梦进化前后的cp值
2 x_data=[338.,333.,328.,207.,226.,25.,179.,60.,208.,606.]
3 y_data=[640.,633.,619.,393.,428.,27.,193.,66.,226.,1591.]
4 # 这里采用最简单的linear model: y_data=b+w*x_data
5 # 我们要用gradient descent把b和w找出来
```

**计算梯度微分的函数getGrad()**

```
1 # 计算梯度微分的函数getGrad()
2 def getGrad(b,w):
3     # initial b_grad and w_grad
4     b_grad=0.0
5     w_grad=0.0
6     for i in range(10):
7         b_grad+=(-2.0)*(y_data[i]-(b+w*x_data[i]))
8         w_grad+=(-2.0*x_data[i])*(y_data[i]-(b+w*x_data[i]))
9     return (b_grad,w_grad)
```

## 1、自己写的版本

当两个微分值b\_grad和w\_grad都为0时，gradient descent停止，b和w的值就是我们要找的最终参数

```
1 # 这是我自己写的版本，事实证明结果很糟糕。。。
2 # y_data=b+w*x_data
3 # 首先，这里没有用到高次项，仅是一个简单的linear model，因此不需要regularization版本的loss function
4 # 我们只需要随机初始化一个b和w，然后用b_grad和w_grad记录下每一次iteration的微分值；不断循环更新b和w直至两个微分值b_grad和w_grad都为0，此时gradient descent停止，b和w的值就是我们要找的最终参数
5
6 b=-120 # initial b
7 w=-4 # initial w
8 lr=0.00001 # learning rate
9 b_grad=0.0
10 w_grad=0.0
11 (b_grad,w_grad)=getGrad(b,w)
12
13 while(abs(b_grad)>0.00001 or abs(w_grad)>0.00001):
```

```

14     #print("b: "+str(b)+"\t\t\t w: "+str(w)+"\n"+"b_grad:
    "+str(b_grad)+"\t\t\t w_grad: "+str(w_grad)+"\n")
15     b-=lr*b_grad
16     w-=lr*w_grad
17     (b_grad,w_grad)=getGrad(b,w)
18
19     print("the function will be y_data="+str(b)+"+"+str(w)+"*x_data")
20
21     error=0.0
22     for i in range(10):
23         error+=abs(y_data[i]-(b+w*x_data[i]))
24     average_error=error/10
25     print("the average error is "+str(average_error))

```

```

1 the function will be y_data=-inf+nan*x_data
2 the average error is nan

```

上面的数据输出处于隐藏状态，点击即可显示

## 2、这里使用李宏毅老师的demo尝试

### 引入需要的库

```

1 import matplotlib
2 import matplotlib.pyplot as plt
3 matplotlib.use('Agg')
4 %matplotlib inline
5 import random as random
6 import numpy as np
7 import csv

```

### 准备好b、w、loss的图像数据

```

1 # 生成一组b和w的数据图，方便给gradient descent的过程做标记
2 x = np.arange(-200,-100,1) # bias
3 y = np.arange(-5,5,0.1) # weight
4 Z = np.zeros((len(x),len(y))) # color
5 X,Y = np.meshgrid(x,y)
6 for i in range(len(x)):
7     for j in range(len(y)):
8         b = x[i]
9         w = y[j]
10
11         # Z[j][i]存储的是loss
12         Z[j][i] = 0
13         for n in range(len(x_data)):
14             Z[j][i] = Z[j][i] + (y_data[n] - (b + w * x_data[n]))**2
15         Z[j][i] = Z[j][i]/len(x_data)

```

### 规定迭代次数和learning rate，进行第一次尝试

距离最优解还有一段距离

```

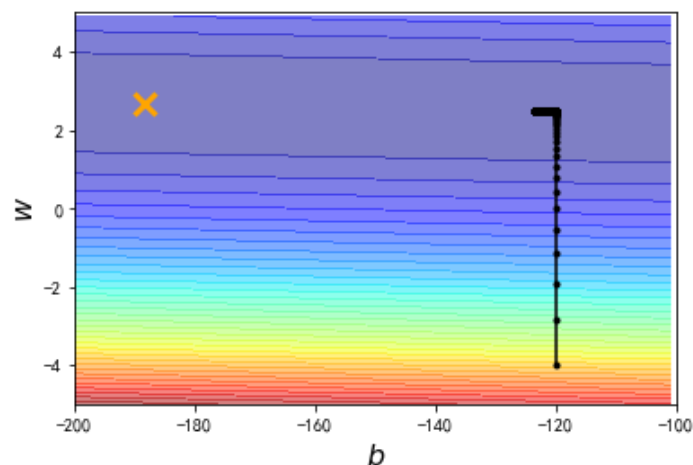
1 # y_data = b + w * x_data
2 b = -120 # initial b

```

```

3 w = -4 # initial w
4 lr = 0.0000001 # learning rate
5 iteration = 100000 # 这里直接规定了迭代次数，而不是一直运行到b_grad和w_grad都为0(事实证明这样做不太可行)
6
7 # store initial values for plotting, 我们想要最终把数据描绘在图上，因此存储过程数据
8 b_history = [b]
9 w_history = [w]
10
11 # iterations
12 for i in range(iteration):
13
14     # get new b_grad and w_grad
15     b_grad, w_grad = getGrad(b, w)
16
17     # update b and w
18     b -= lr * b_grad
19     w -= lr * w_grad
20
21     # store parameters for plotting
22     b_history.append(b)
23     w_history.append(w)
24
25 # plot the figure
26 plt.contourf(x, y, Z, 50, alpha=0.5, cmap=plt.get_cmap('jet'))
27 plt.plot([-188.4], [2.67], 'x', ms=12, markeredgewidth=3, color='orange')
28 plt.plot(b_history, w_history, 'o-', ms=3, lw=1.5, color='black')
29 plt.xlim(-200, -100)
30 plt.ylim(-5, 5)
31 plt.xlabel(r'$b$', fontsize=16)
32 plt.ylabel(r'$w$', fontsize=16)
33 plt.show()

```



## 把learning rate增大10倍尝试

发现经过100000次的update以后，我们的参数相比之前与最终目标更接近了，但是这里有一个剧烈的震荡现象发生

```

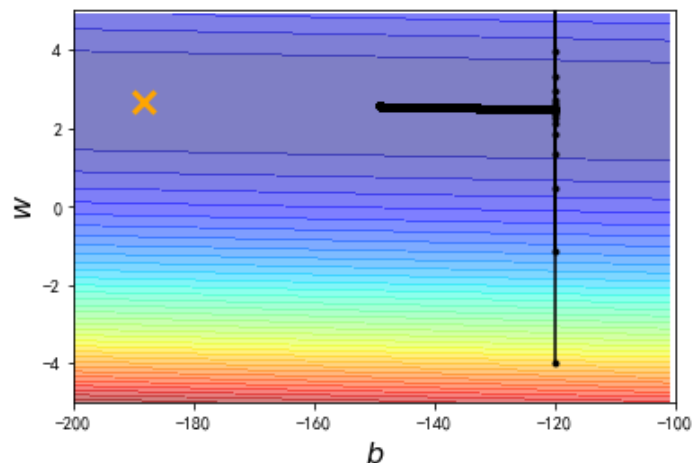
1 # 上图中，gradient descent最终停止的地方里最优解还差很远，
2 # 由于我们是规定了iteration次数的，因此原因应该是learning rate不够大，这里把它放大10倍
3
4 # y_data = b + w * x_data

```

```

5  b = -120 # initial b
6  w = -4 # initial w
7  lr = 0.000001 # learning rate 放大10倍
8  iteration = 100000 # 这里直接规定了迭代次数，而不是一直运行到b_grad和w_grad都为0(事实证明这样做不太可行)
9
10 # store initial values for plotting, 我们想要最终把数据描绘在图上，因此存储过程数据
11 b_history = [b]
12 w_history = [w]
13
14 # iterations
15 for i in range(iteration):
16
17     # get new b_grad and w_grad
18     b_grad,w_grad=getGrad(b,w)
19
20     # update b and w
21     b -= lr * b_grad
22     w -= lr * w_grad
23
24     #store parameters for plotting
25     b_history.append(b)
26     w_history.append(w)
27
28 # plot the figure
29 plt.contourf(x,y,Z,50,alpha=0.5,cmap=plt.get_cmap('jet'))
30 plt.plot([-188.4],[2.67],'x',ms=12,markeredgewidth=3,color='orange')
31 plt.plot(b_history,w_history,'o-',ms=3,lw=1.5,color='black')
32 plt.xlim(-200,-100)
33 plt.ylim(-5,5)
34 plt.xlabel(r'$b$',fontsize=16)
35 plt.ylabel(r'$w$',fontsize=16)
36 plt.show()

```



## 把learning rate再增大10倍

发现此时learning rate太大了，参数一update，就远远超出图中标注的范围了

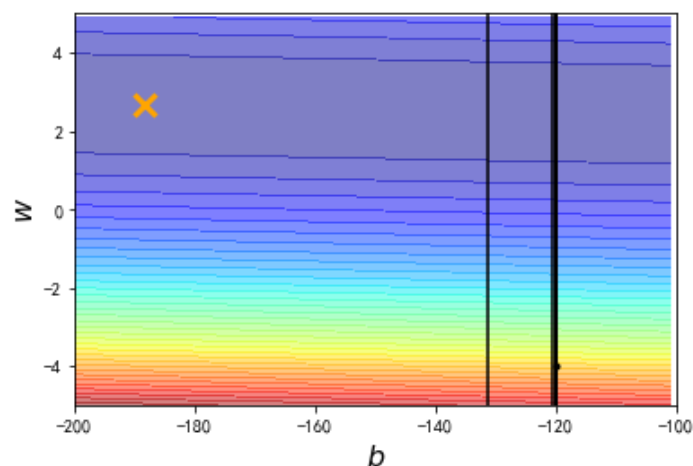
所以我们会发现一个很严重的问题，如果learning rate变小一点，他距离最佳解还是会具有一段距离；但是如果learning rate放大，它就会直接超出范围了

1 # 上图中，gradient descent最终停止的地方里最优解还是有一点远，

```

2  # 由于我们是规定了iteration次数的，因此原因应该是learning rate还是不够大，这里再把它放
   大10倍
3
4  # y_data = b + w * x_data
5  b = -120 # initial b
6  w = -4 # initial w
7  lr = 0.00001 # learning rate 放大10倍
8  iteration = 100000 # 这里直接规定了迭代次数，而不是一直运行到b_grad和w_grad都为0(事
   实证明这样做不太可行)
9
10 # store initial values for plotting, 我们想要最终把数据描绘在图上，因此存储过程数据
11 b_history = [b]
12 w_history = [w]
13
14 # iterations
15 for i in range(iteration):
16
17     # get new b_grad and w_grad
18     b_grad, w_grad = getGrad(b, w)
19
20     # update b and w
21     b -= lr * b_grad
22     w -= lr * w_grad
23
24     # store parameters for plotting
25     b_history.append(b)
26     w_history.append(w)
27
28 # plot the figure
29 plt.contourf(x, y, Z, 50, alpha=0.5, cmap=plt.get_cmap('jet'))
30 plt.plot([-188.4], [2.67], 'x', ms=12, markeredgewidth=3, color='orange')
31 plt.plot(b_history, w_history, 'o-', ms=3, lw=1.5, color='black')
32 plt.xlim(-200, -100)
33 plt.ylim(-5, 5)
34 plt.xlabel(r'$b$', fontsize=16)
35 plt.ylabel(r'$w$', fontsize=16)
36 plt.show()

```



这个问题明明很简单，可是只有两个参数b和w，gradient descent搞半天都搞不定，那以后做neural network有数百万个参数的时候，要怎么办呢

这个就是一室不治何以国家为的概念

## 于是这里就要放大招了!!! ——Adagrad

我们给b和w定制化的learning rate, 让它们两个的learning rate不一样

```
1  # 这里给b和w不同的learning rate
2
3  # y_data = b + w * x_data
4  b = -120 # initial b
5  w = -4 # initial w
6  lr = 1 # learning rate 放大10倍
7  iteration = 100000 # 这里直接规定了迭代次数, 而不是一直运行到b_grad和w_grad都为0(事实证明这样做不太可行)
8
9  # store initial values for plotting, 我们想要最终把数据描绘在图上, 因此存储过程数据
10 b_history = [b]
11 w_history = [w]
12
13 lr_b = 0
14 lr_w = 0
15
16 # iterations
17 for i in range(iteration):
18
19     # get new b_grad and w_grad
20     b_grad, w_grad = getGrad(b, w)
21
22     # get the different learning rate for b and w
23     lr_b = lr_b + b_grad ** 2
24     lr_w = lr_w + w_grad ** 2
25
26     # 这一招叫做adagrad, 之后会详加解释
27     # update b and w with new learning rate
28     b -= lr / np.sqrt(lr_b) * b_grad
29     w -= lr / np.sqrt(lr_w) * w_grad
30
31     # store parameters for plotting
32     b_history.append(b)
33     w_history.append(w)
34
35     # output the b w b_grad w_grad
36     # print("b: "+str(b)+"\t\t\t w: "+str(w)+"\n"+"b_grad: "+str(b_grad)+"\t\t\t w_grad: "+str(w_grad)+"\n")
37
38     # output the final function and its error
39     print("the function will be y_data="+str(b)+"+"+str(w)+"*x_data")
40     error=0.0
41     for i in range(10):
42         print("error "+str(i)+" is: "+str(np.abs(y_data[i]-(b+w*x_data[i]))))+")
43         error+=np.abs(y_data[i]-(b+w*x_data[i]))
44     average_error=error/10
45     print("the average error is "+str(average_error))
46
47     # plot the figure
48     plt.contourf(x, y, Z, 50, alpha=0.5, cmap=plt.get_cmap('jet'))
49     plt.plot([-188.4], [2.67], 'x', ms=12, markeredgewidth=3, color='orange')
50     plt.plot(b_history, w_history, 'o-', ms=3, lw=1.5, color='black')
```

```

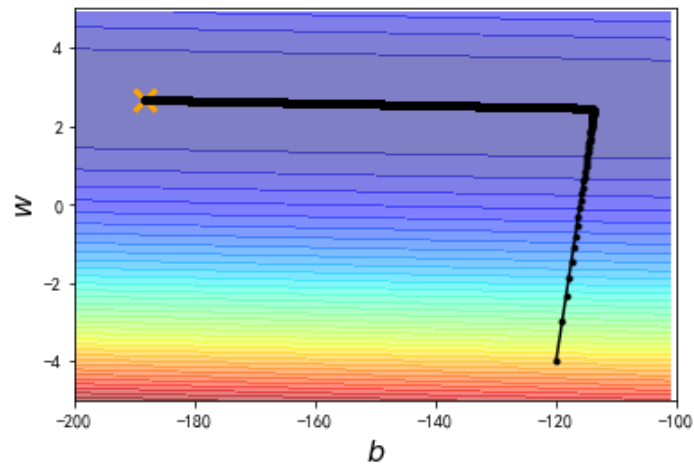
51 plt.xlim(-200,-100)
52 plt.ylim(-5,5)
53 plt.xlabel(r'$b$',fontsize=16)
54 plt.ylabel(r'$w$',fontsize=16)
55 plt.show()

```

```

1 the function will be y_data=-188.3668387495323+2.6692640713379903*x_data
2 error 0 is: 73.84441736270833
3 error 1 is: 67.4980970060185
4 error 2 is: 68.15177664932844
5 error 3 is: 28.8291759825683
6 error 4 is: 13.113158627146447
7 error 5 is: 148.63523696608252
8 error 6 is: 96.43143001996799
9 error 7 is: 94.21099446925288
10 error 8 is: 140.84008808876973
11 error 9 is: 161.7928115187101
12 the average error is 89.33471866905532

```



有了新的learning rate以后，从初始值到终点，我们在100000次iteration之内就可以顺利地完成了