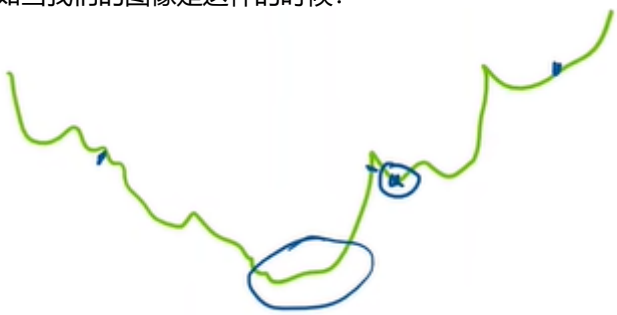
梯度下降算法

笔记本:	pytorch		
创建时间:	2023/3/23 21:51	**更新时间:**	2023/3/24 10:56
作者:	22qdnlbn		

1、
上一节线性模型我们使用的其实是**穷举法**，但是如果w变多时，穷举法就不再合适了，事件复杂度太高。
此时，出现了分治法，比如有w1，w2，我们可以将其各自分成4份，其实就是找了16个点，在这16个点中
找最小值，之后在这个最小值的点周围在使用**分治法**，依次去找最小值。

但是这里有个弊端：
比如当我们的图像是这样的时候：



我们就会错过全局最优解，找到的是局部最优解。

如果只有一个最优解，使用分治，是可以找到最优解的。

还有一个弊端就是如果w很多，那么事件复杂度也会很高。

2、
梯度下降算法：梯度的方向是上升的方向，但其实我们是要往下降的方向，所以要减去。α是学习率。
使用梯度来更新我们的参数。

梯度下降 贪心算法，每次是往当前最好的方向前进。梯度下降也有可能找到的是局部最优解，但是大量的数据表明，
我们算法中的损失函数一般不会包含很多参数，即不会有很多局部最优解，所以我们经常使用梯度下降法。

Derivative

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

Gradient

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

Update

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

Update

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

还有一种情况：就是鞍点，即梯度为0的点，这时候会导致我们无法更新参数了。

3、
使用梯度下降不是每一次损失函数都是减少的，它也有可能会出现增加的情况，但是它的大方向应该是减小的。
这种情况下，我们可以使用指数加权平均来使得我们的损失曲线更平滑。

4、随机梯度下降 sgd

我们可以一次只使用一个样本来计算梯度，之后利用该梯度来更新参数。
这样做的好处是，我们可以减少出现鞍点这种情况，因为一个样本它有噪音，可能就会让我们跳过鞍点。但是也有可能出现噪声太大的情况。

缺点是：sgd每一个样本点都依赖于上一个样本点计算出来的梯度，每个样本点之间并不能并行计算。
但是梯度下降，它整个数据集中的样本点依赖的是上一次迭代中数据集计算出来的梯度，每个样本之间是没有依赖关系的，可以并行。

5、mini-batch

综合可知，我们可以使用mini-batch梯度下降，既不会让某一个样本点的噪声太过突出，可以并行了，也不会说每次必须计算出所有样本点的损失之后
才能进行参数更新。

```
import numpy as np
import matplotlib.pyplot as plt


x_data=[1.0,2.0,3.0]
y_data=[2.0,4.0,6.0]
w=1.0
def forward(x):
    return x*w
mse_list=[]
epoch_list=[]
def loss(xs,ys):
    l_sum=0
    for x,y in zip(xs,ys):
           l_sum+=(forward(x)-y)**2
    return l_sum/3


def gradient(xs,ys):
    grad=0
    for x,y in zip(xs,ys):
        grad+=2*x*(x*w-y)
    return grad/len(xs)
```

```
for epoch in range(101):
    epoch_list.append(epoch)
    cost=loss(x_data,y_data)
    mse_list.append(cost)
    grad=gradient(x_data,y_data)
    w=w-0.05*grad
    print('epoch=',epoch, 'w=',w,'cost=',cost)
plt.plot(epoch_list,mse_list)
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
```