最优化理论homework1

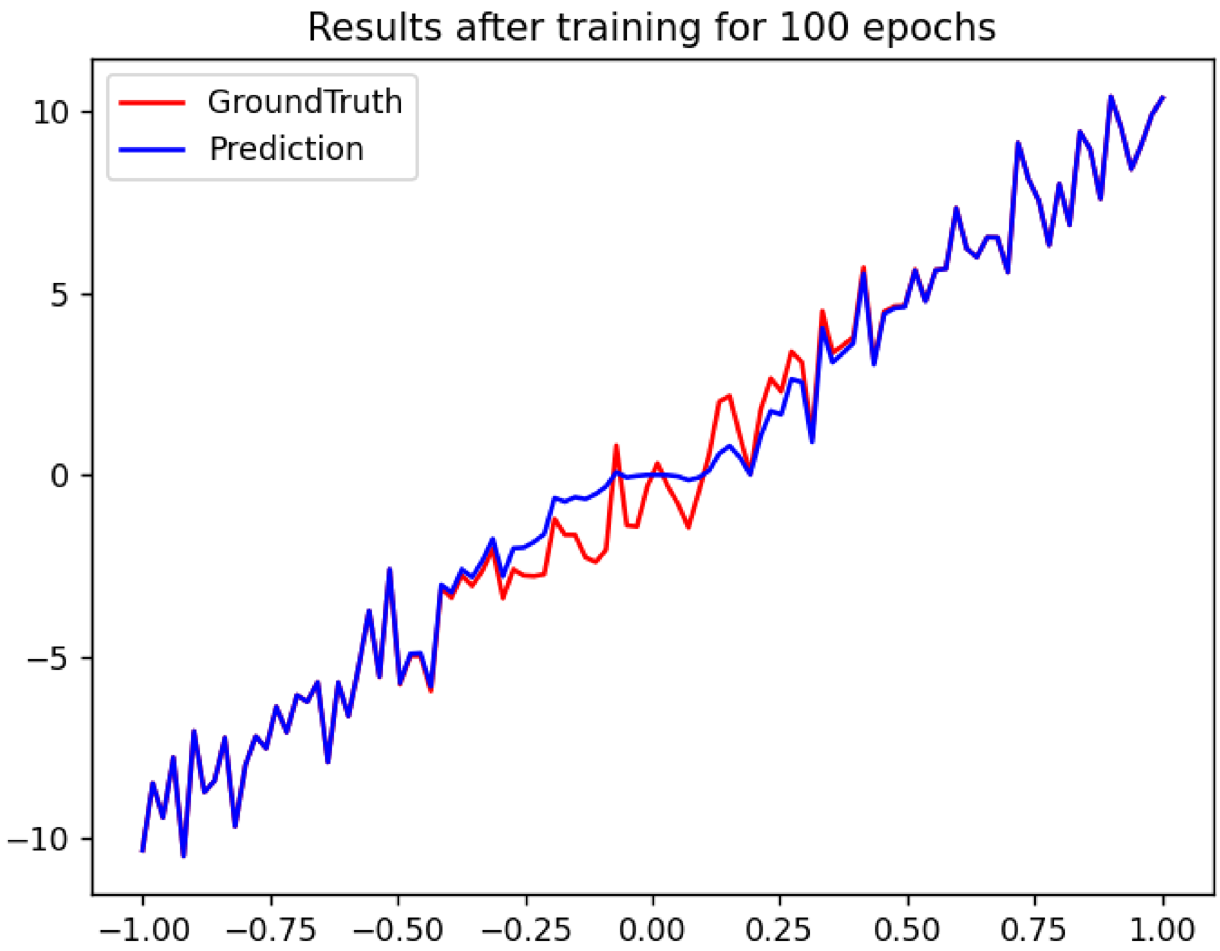
姓名	学号
崔璨明	20337025

1、补全代码

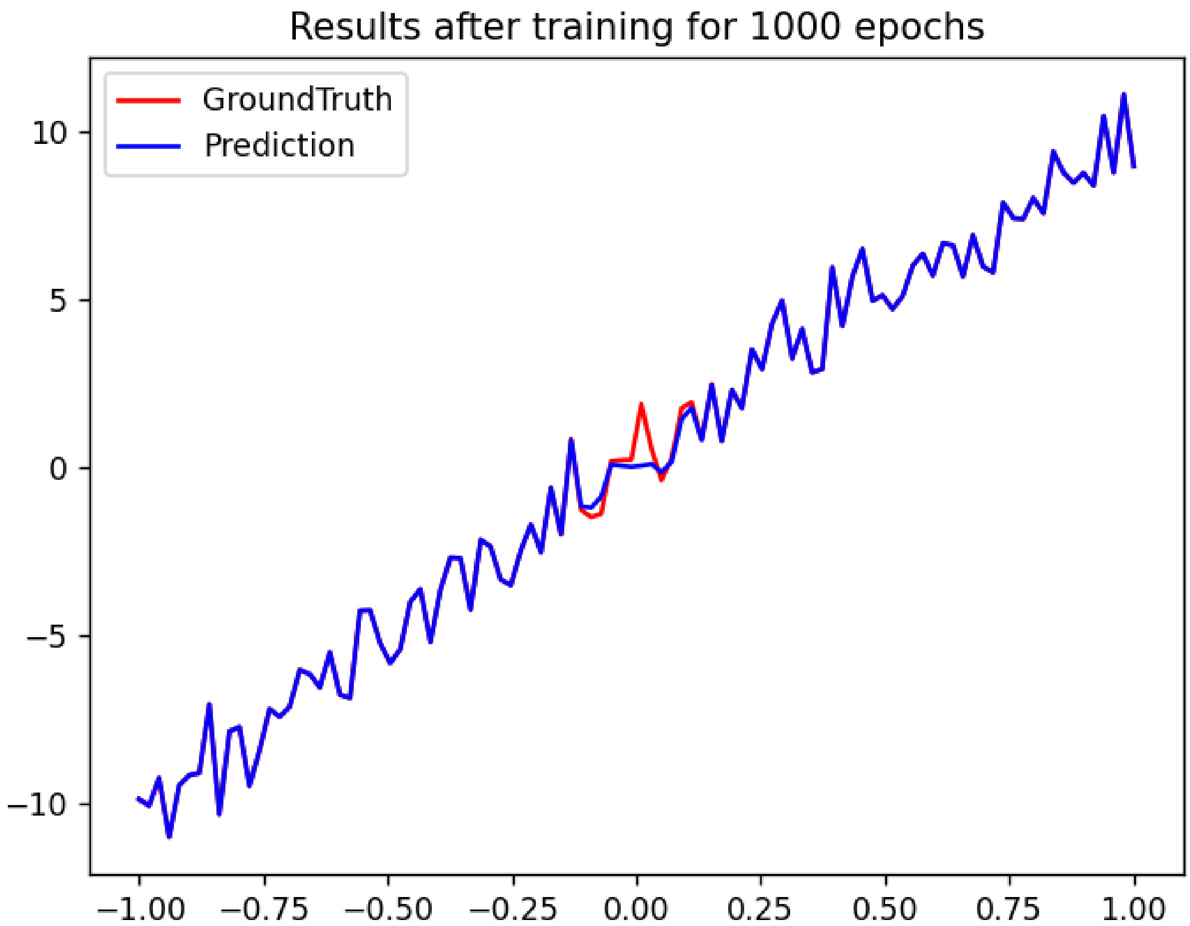
按照题目要求，使用梯度下降方面补全homeworks1.py中的代码，具体内容如下：

```
...
for step in range(epochs):
    pred = W * X
    #Optimization: gradient descend
    grad = 2*(pred-Y)*X
    W = W-lr*grad
    loss = ((Y - pred) **2).mean()
    print("[%d/%d]LOSS:%.3f" %(step + 1, epochs, loss))
...
```

代码运行结果如下，蓝色的函数图像为拟合的函数，红色的为目标函数Y，可以看到在取步长 $lr = 0.1, epochs = 100$ 时，拟合的结果较好，最后的损失函数值为0.298。



当增加迭代次数时，可以得到更好的拟合结果（$lr = 0.1, epochs = 1000$）：



尝试多种不同的迭代次数和步长的组合，进行多次实验，以损失函数为评价拟合效果的标准，

得到结果如下（表格中横轴为步长，纵轴为迭代次数，对应格的值为几次实验拟合的损失函数的平均值）：

迭代次数/步长	0.10	0.07	0.04	0.01
100	0.335	0.514	0.968	5.496
400	0.088	0.112	0.187	0.996
700	0.082	0.103	0.079	0.417
1000	0.030	0.048	0.099	0.335

由此可见，步长lr越小则收敛得越慢，但步长过大又容易引起收敛速度太快，可能一下子就越过极值点，导致发散的问题；而迭代次数越多则拟合效果越好，但会花费不必要的时间（函数早已收敛），因此需要选择合适的步长和迭代次数的组合。

2、选择合适的步长

1. 可以采用**line search**的方法来选择步长，即每次试一个步长，如果用该步长走的话，观察函数值会不会比当前点下降一定的程度，如果没有，就按比例减小步长，继续测试直至迭代次数结束。编写代码如下：

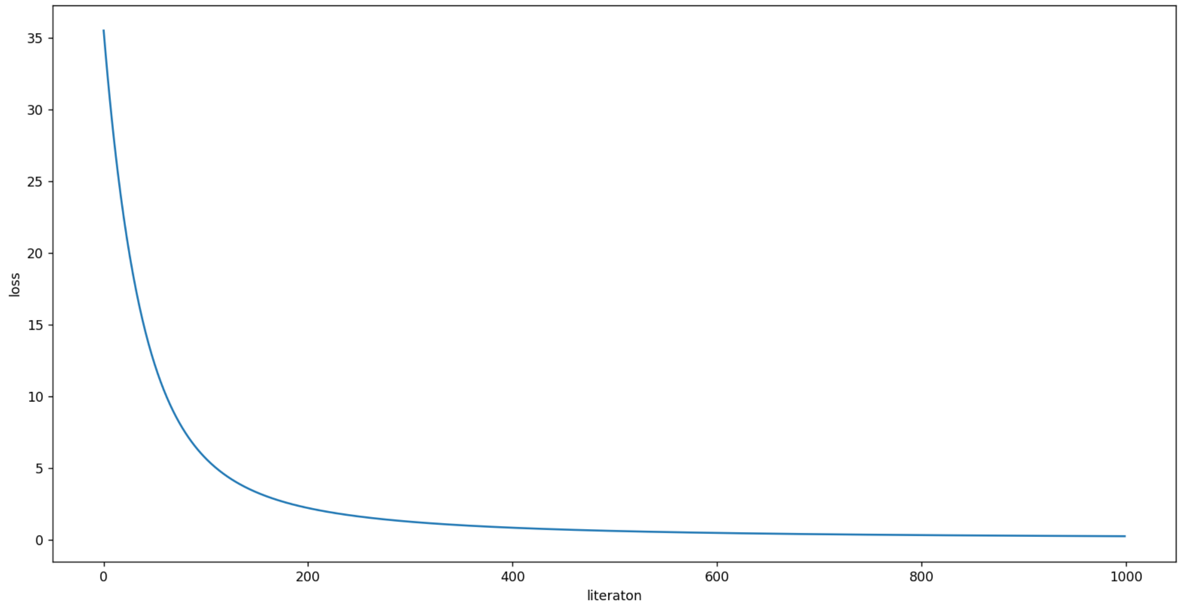
```
pre_loss=loss   #储存上一次迭代的损失函数值
for step in range(epochs):
    pred = W * X
    #Optimization: gradient descend
    grad = 2*(pred-Y)*X
    W = W-lr*grad
    loss = ((Y - pred) **2).mean()
    if loss>=pre_loss:
        lr*=0.8
    pre_loss=loss
    print("[%d/%d]LOSS:%.3f" %(step + 1, epochs, loss),lr)
```

在$lr = 0.1, epochs = 100$ 时，进行多次实验，可以发现若采用line search方法进行步长更新，最后得到的损失函数值要小于不对步长进行更新得到的损失函数值，但运行的速度要较慢。

2. 一般都是手动调整，通用的做法就是从较小的学习率开始尝试，如果遇到不平稳现象，那就调小学习率，或者按三倍来调整，确定范围之后再微调。

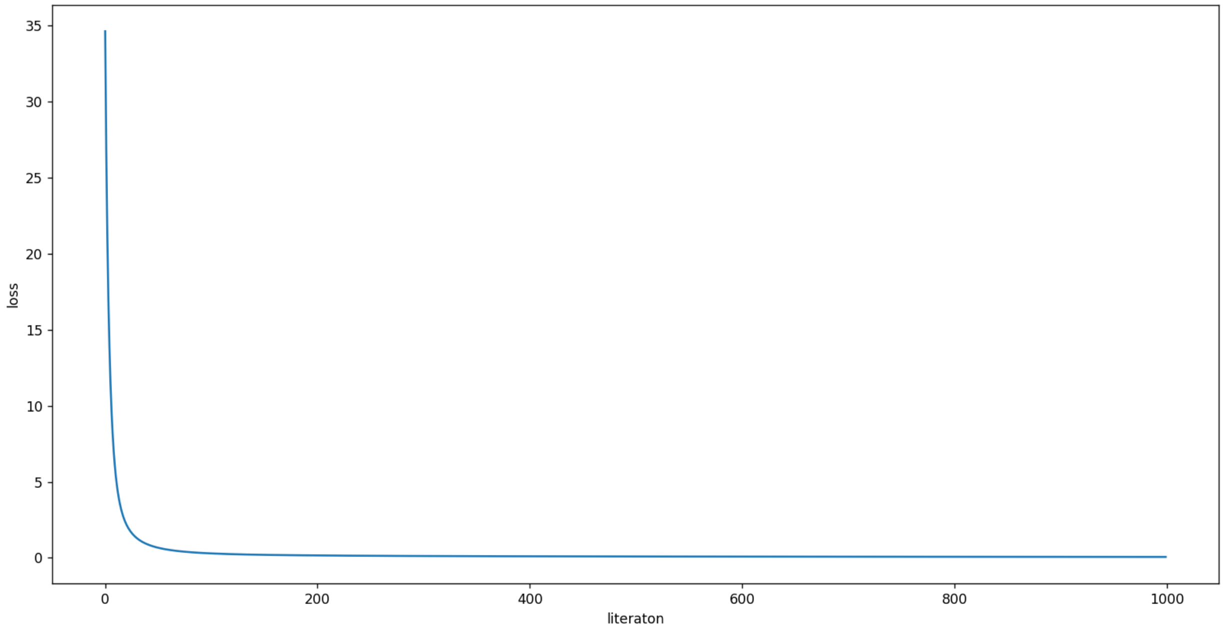
3、迭代次数

在确定了合适的步长后，我们可以绘制损失函数的变化图，通过观察函数图上的拐点的位置来确定合适的迭代次数，以$lr=0.01,epochs=1000$为例，绘制损失函数变化图如下：



由图可知，损失函数在迭代次数接近400时便降低得很慢，几乎没有什么变化，于是我们可以将迭代次数定为400，既能取得不错的拟合结果，又大大减少了运行的时间，减少计算的花费。

当$lr=0.1,epochs=1000$时，绘制损失函数变化图如下：



由图可知，损失函数在迭代次数接近70时便降低得很慢，几乎没有什么变化，于是我们可以将迭代次数定为70。

而在其他问题中也可以利用这种方法，绘制损失函数图来确定要迭代多少次才能有一个比较好的解。