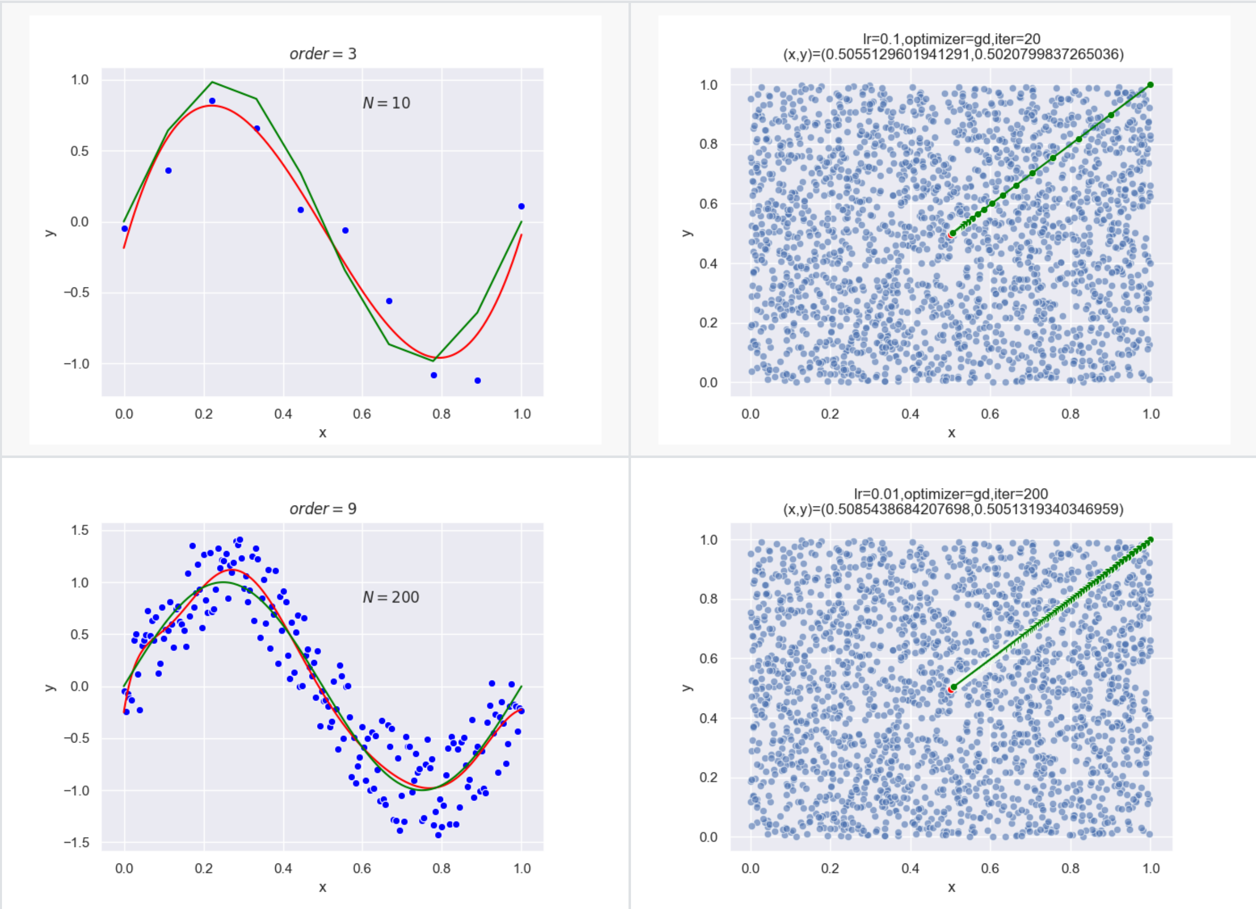


现代人工智能技术HW1

Code Released for Homework1 of Modern AI course.

由于github无法支持markdown嵌入latex公式，大家可以阅读pdf.

运行脚本位于 `code/run.sh`

如果本repo对你有帮助，请点亮star，有任何问题可以提出issue讨论，我会及时回复!

题目

> 编程模拟仿真：多项式回归
>
> 参见 textbook p4-12。完成以下任务：
>
> 1. 生成正弦序列s(n);
> 2. 使用噪声函数对正弦序列加噪x(n)=s(n)+w(n);
> 3. 使用多项式回归模型对x(n)进行拟合，并分析过拟合和欠拟合情况

> **注：参考误差函数式 1-2，带正则项的修正误差函数式 1-4，实验仿真生成图 1-6、图 1-7，并给出有关系数表。

> 1. Generate n = 2,000 points uniformly at random in the two-dimensional unit square. Which point do you expect the centroid to be?
> 2. What objective does the centroid of the points optimize?
> 3. Apply gradient descent (GD) to find the centroid.
> 4. Apply stochastic gradient descent (SGD) to find the centroid. Can you say in simple words, what the algorithm is doing?
>
> **In mathematics and physics, the centroid or geometric center of a plane figure is the arithmetic mean position of all the points in the figure. Informally, it is the point at which a cutout of the shape could be perfectly balanced on the tip of a pin.

HW1.1

(1) 生成正弦序列s(n)

首先生成n序列，然后通过x序列生成sin(n)序列

```
# s(n) = sin(n)
n = np.arange(0, 2 * np.pi, opt.point_interval)
s_n = np.sin(n)
```

(2) 使用噪声函数对正弦序列加噪 x(n)=s(n)+w(n)

这里定义噪声函数 w(n)为均匀分布的噪声，直接使用`numpy`随机数生成器生成；

```
np.random.seed(123)
x_n = s_n + (np.random.rand(s_n.size) - 0.5)
```

在上面我们可以看到，我们为了保证每次随机噪声相同，手动设置了`numpy`随机数生成的seed

(3) 使用多项式回归模型对x(x)进行拟合，并分析过拟合和欠拟合情况

多项式模型可以定义为如下形式：

其中n为多项式模型的阶数。

一般使用最小二乘法对其进行拟合

我们的优化目标为：

为了避免过拟合，我们有时需要在优化目标中加入正则项，则优化目标变为：

实现

首先定义目标函数：

```
def cost(p, x, y, lam):
    ret = y - np.poly1d(p)(x)
    reg = np.sqrt(math.exp(lam)) * (p * p)
    ret = np.append(ret, reg)
    return ret
```

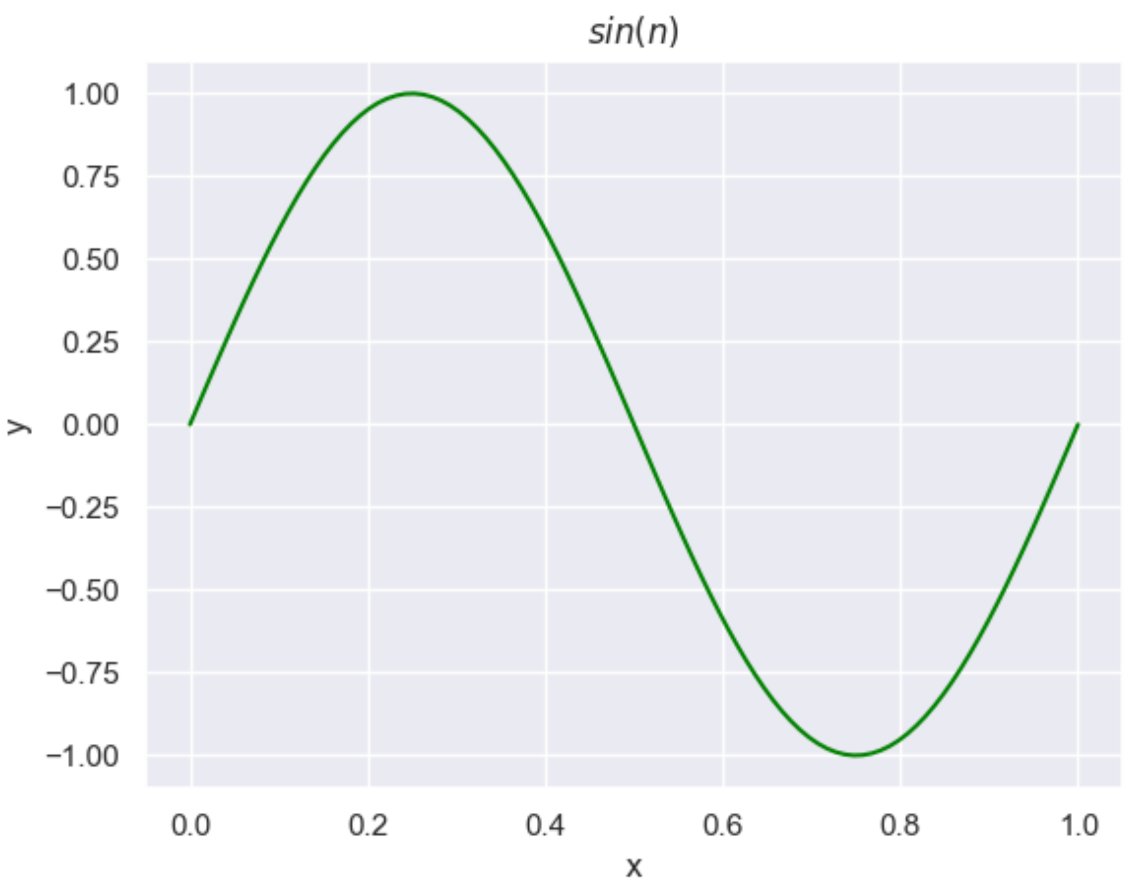
其中第一个参数`p`为多项式模型中的参数，`x`和`y`为要拟合的数据，`lam`为代表正则项系数，其中另正则项系数为\lambda，则ln(\lambda)=lam。

根据目标函数对多项式模型进行最小二乘拟合，这里使用`scipy.optimize`中实现的最小二乘：

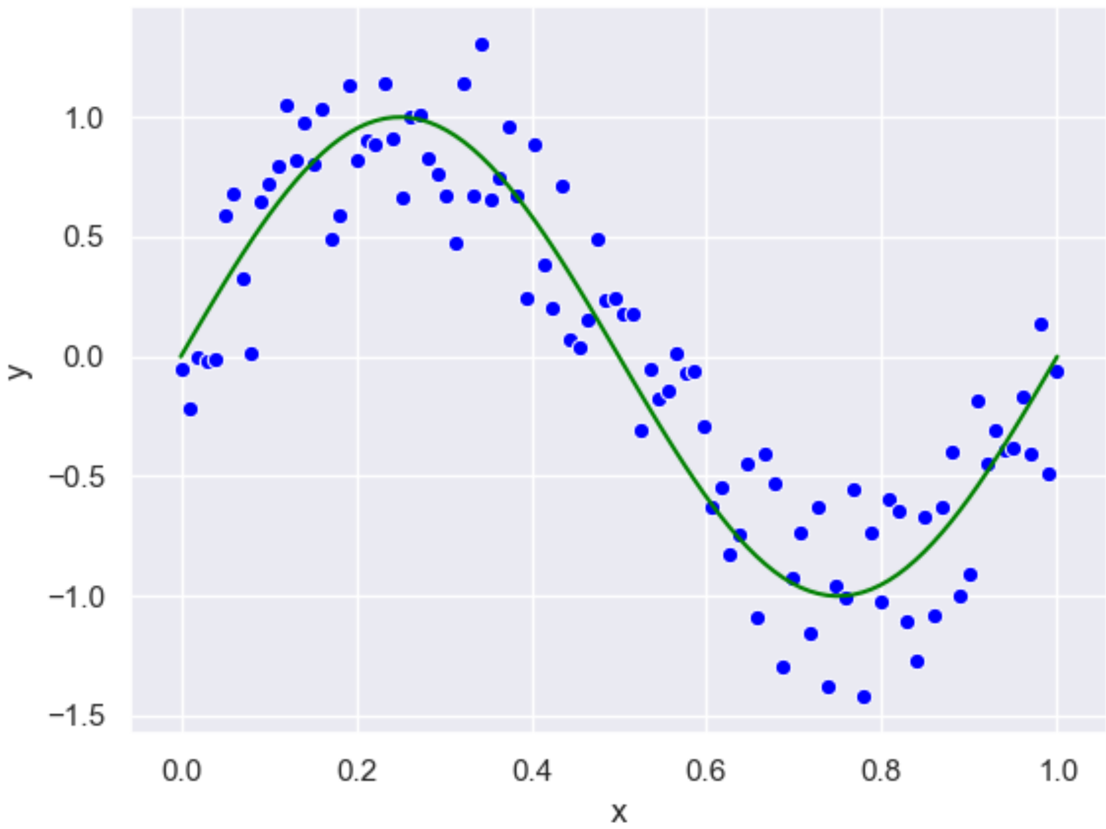
```
from scipy.optimize import leastsq
p = np.zeros((order+1,1)) # 初始化参数
coff = leastsq(cost, p, args=(x, y, lam)) # 最小二乘
```

结果分析

生成的sin(n)序列：

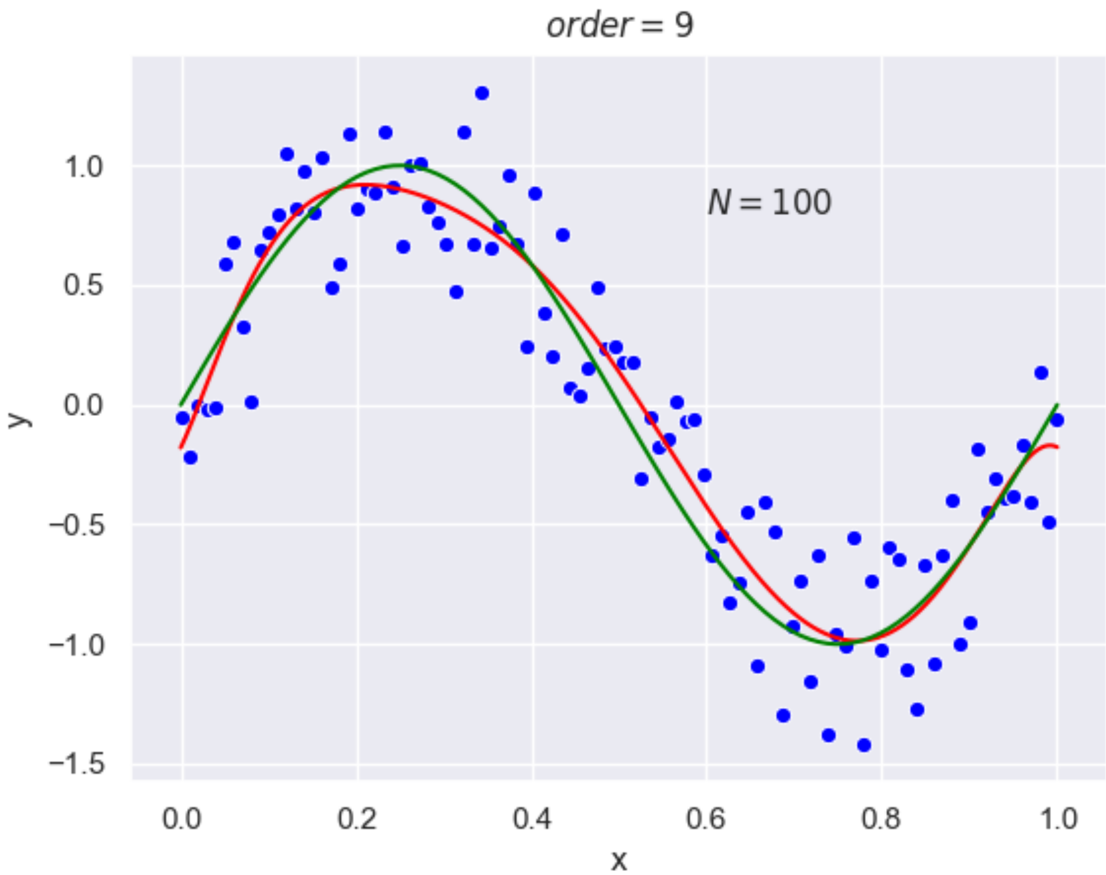


加噪声后x(n)序列：



1.0
0.5
0.0
−0.5
−1.0
−1.5
y
0.0
0.2
0.4
0.6
0.8
1.0
x

使用9次曲线拟合x(n):

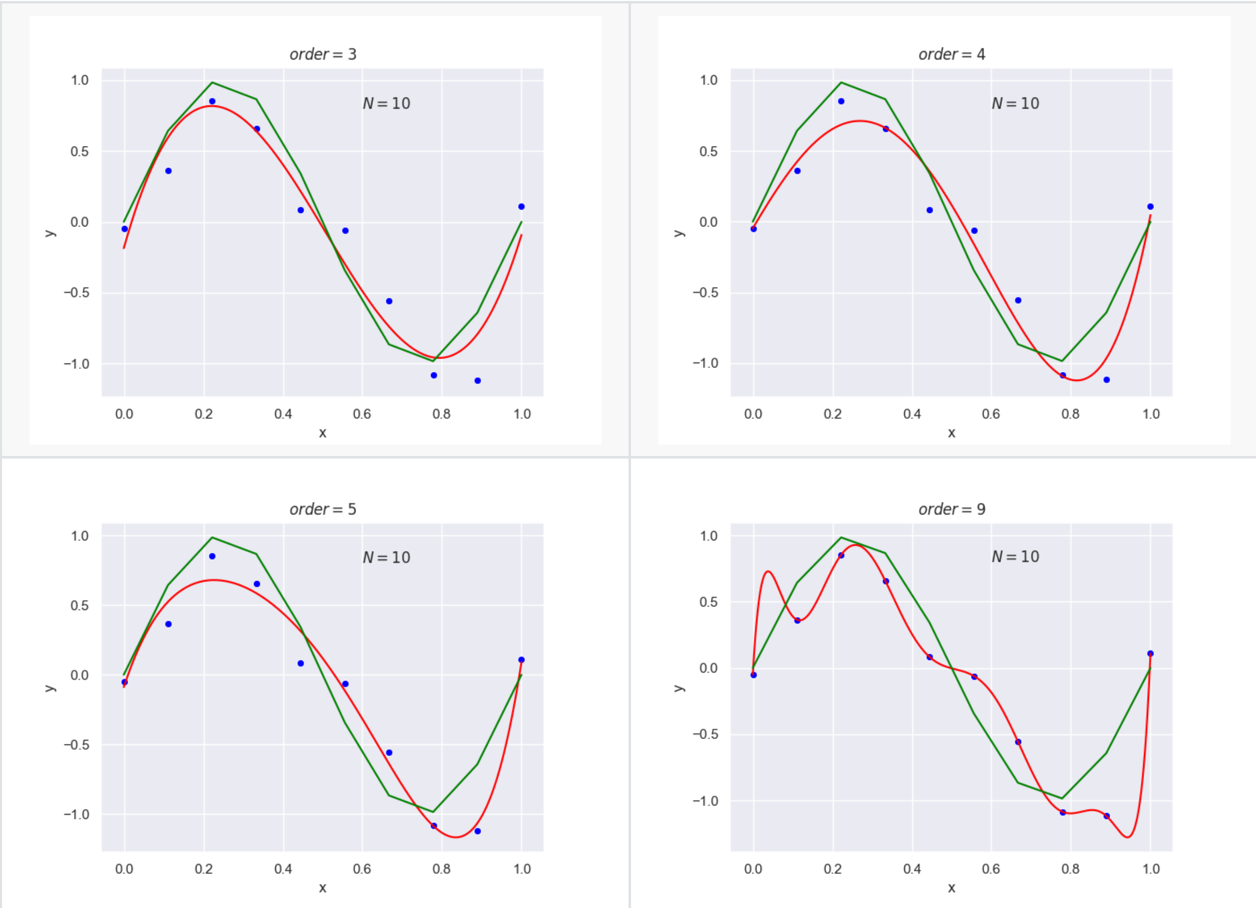


$order = 9$
$N = 100$
1.0
0.5
0.0
−0.5
−1.0
−1.5
y
0.0
0.2
0.4
0.6
0.8
1.0
x

分析过拟合和欠拟合的情况

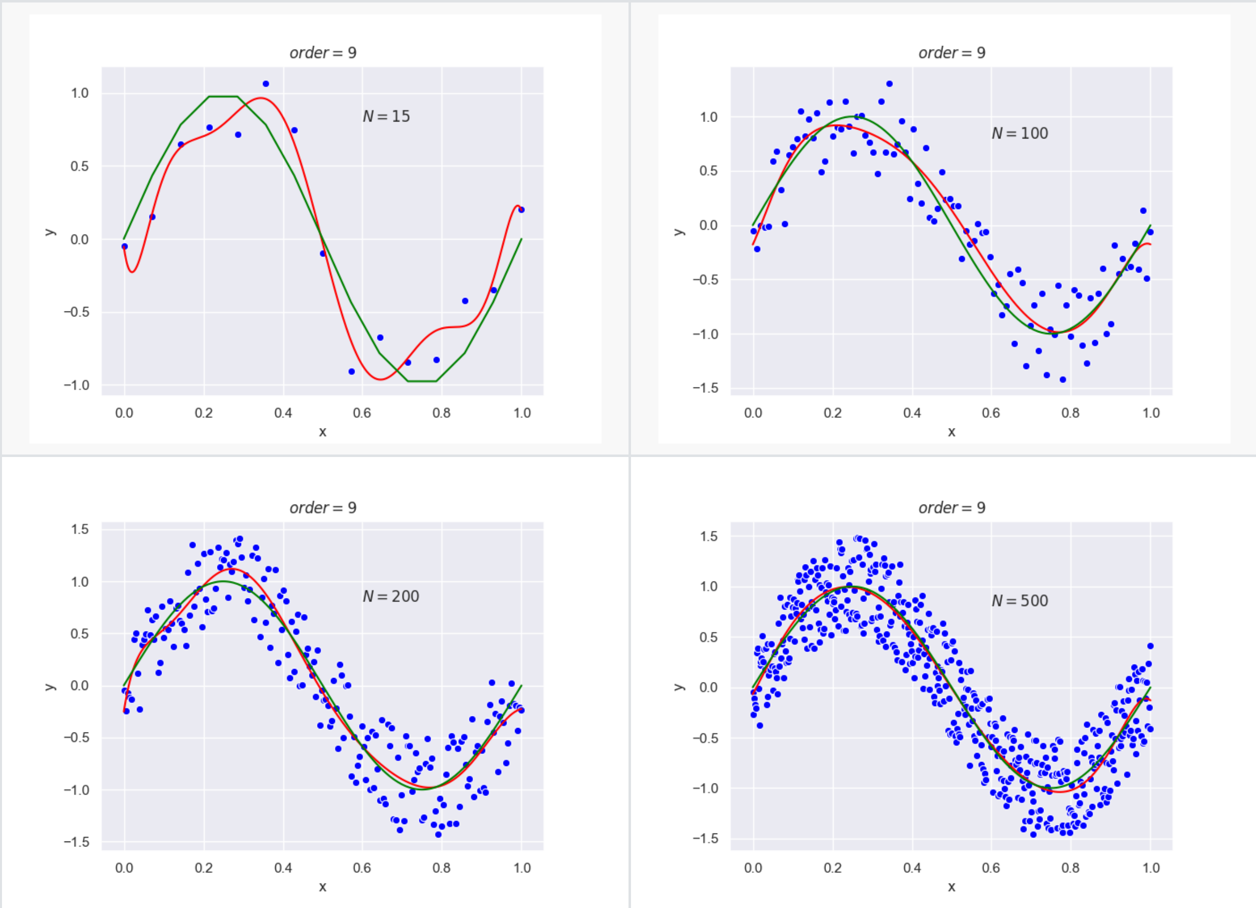
过拟合主要会由**多项式阶数过高或者数据过少**造成，而欠拟合可能由于**多项式阶数过低、正则项系数过大**造成

过拟合-阶数过高



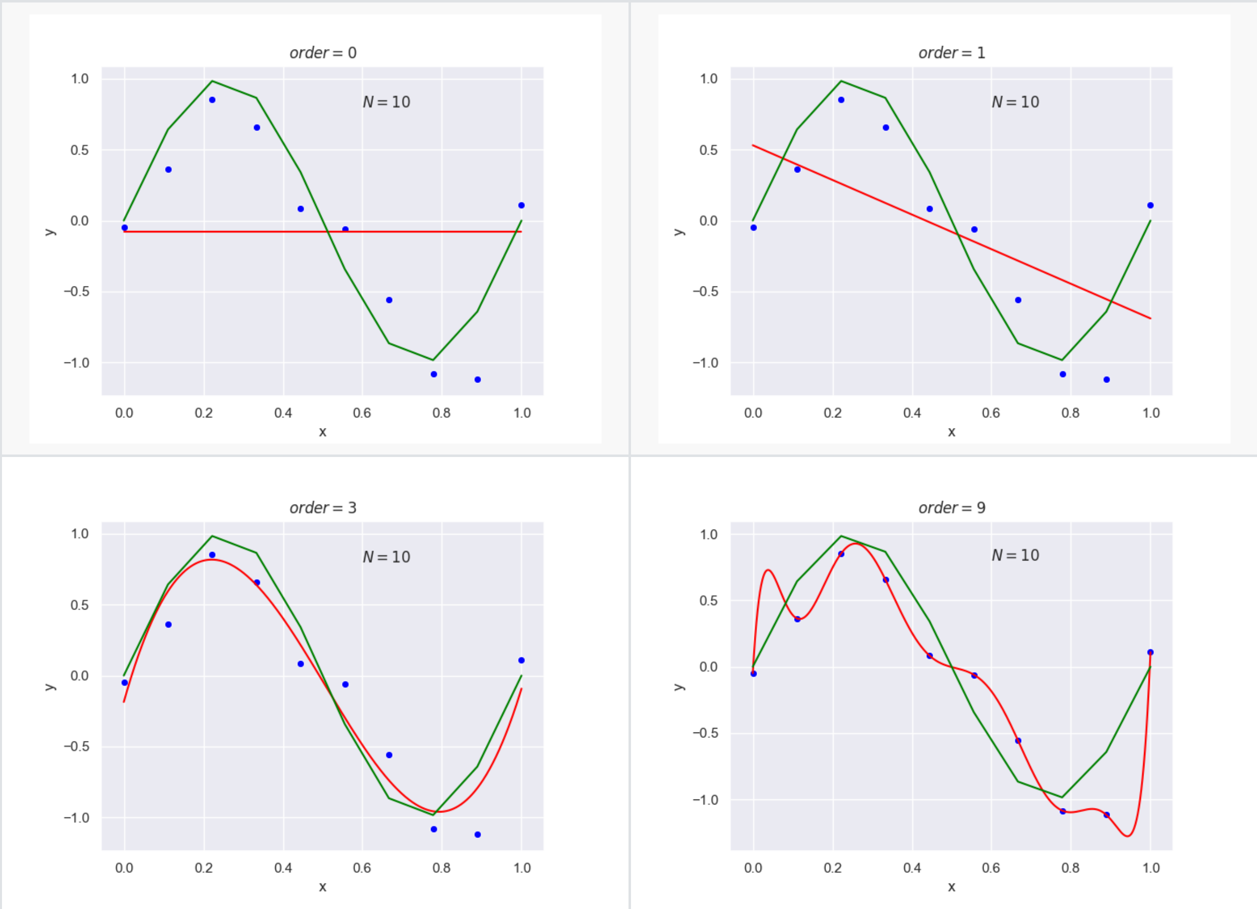
如上图，说明了由多项式系数过高引起的过拟合，可以看到随着阶数的提高，**过拟合现象提升**，在最后一张九阶曲线

过拟合-数据少



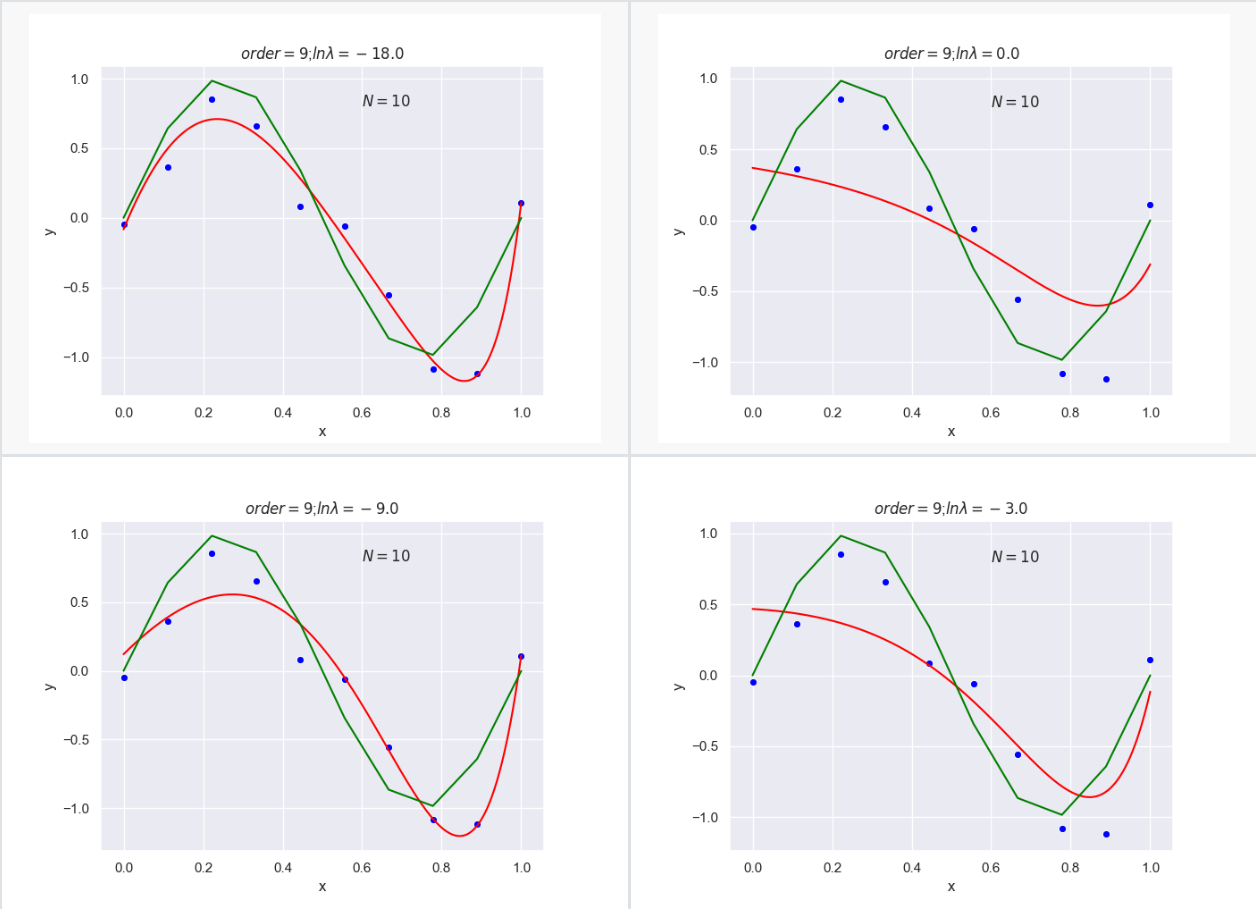
如上图，四张图所用多项式模型阶数相同，但是数据数量不同。第一张图点数只有15个点，明显曲线出现了过拟合的情况，剩下三张图通过增加数据量，有效**减小了过拟合**的现象。

欠拟合-阶数低



如上图，数据点数相同，分别使用不同阶数的曲线拟合，明显可见**低阶多项式无法很好的拟合数据**，出现了欠拟合现象。

正则的影响



从上图可以看出，同样对于九次曲线，加入正则项后可以明显的减少过拟合的现象，但是当正则项过大的时候，欠拟合现象也会出现（如图四）

HW1.2

1

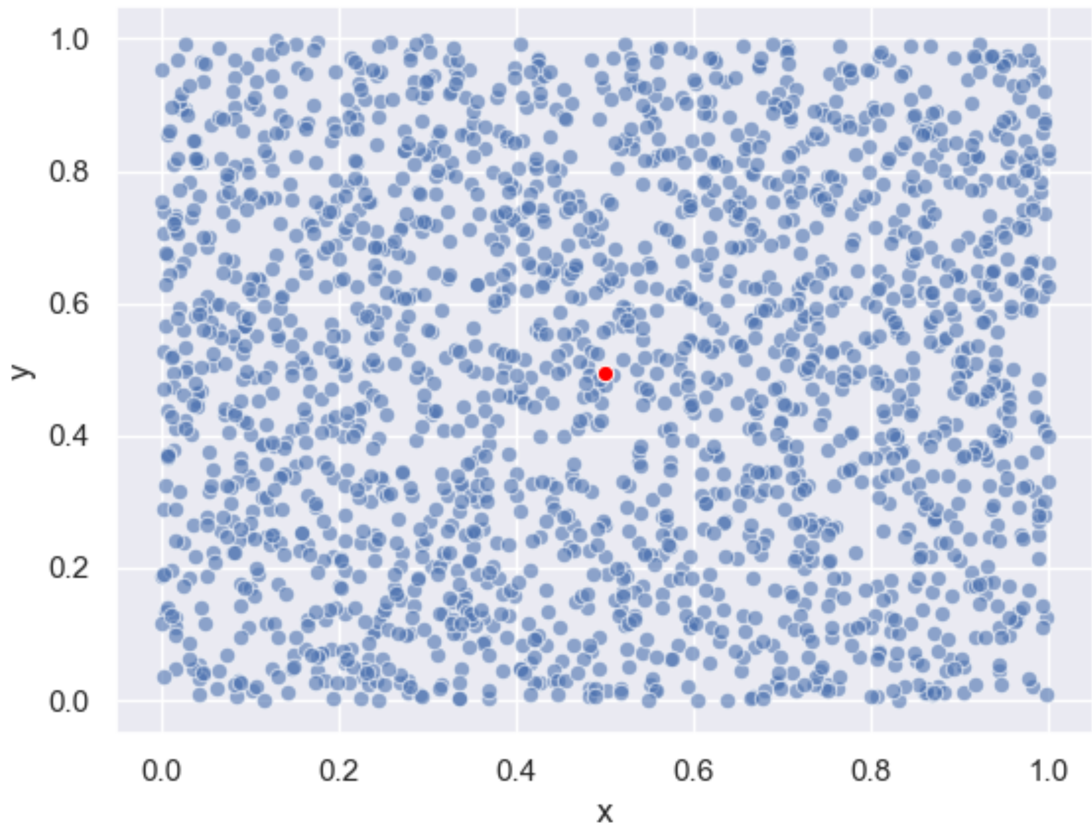
Generate n = 2,000 points uniformly at random in the two-dimensional unit square. Which point do you expect the centroid to be?

```
x = np.random.rand(opt.n)
y = np.random.rand(opt.n)
```

如上述代码所示，我们使用随机分布生成2000个点。他们的质心为如下：

```
gt_centroid_x = np.average(x) # 0.5
gt_centroid_y = np.average(y) # 0.5
```

由于生成的x y坐标都是在0～1的均匀分布，所以他们的质心应该是在 (0.5,0.5)



2

What objective does the centroid of the points optimize?

根据Wikipediahttps://zh.wikipedia.org/wiki/%E5%87%A0%E4%BD%95%E4%B8%AD%E5%BF%83对于质心性质的说明：

> 这个中心是空间中一点到这有限个点距离的平方和的唯一最小值点

目标函数

所以我们可以将目标函数定义为：

在代码中实现为：

```
def loss(x, y, centroid_x, centroid_y):
    l2_distance_square = (x - centroid_x) ** 2 + (y - centroid_y) ** 2
    return sum(l2_distance_square) / len(x)
```

求梯度

那么对目标函数求偏导即可

在代码中实现为：

```
# gradient function
def gradient(x, y, centroid_x, centroid_y):
    d_x = 2 * (centroid_x - x)
    d_y = 2 * (centroid_y - y)
    return sum(d_x) / len(x), sum(d_y) / len(y)
```

上述两个步骤，**定义目标函数和求梯度**就是梯度下降法的两个最重要的部分，接下来我们只需要在每次迭代中计算梯度，并根据梯度更新当前质心的位置即可。

3

Apply gradient descent (GD) to find the centroid.

使用梯度下降寻找质心有以下步骤：

1. 初始化质心

```
# init value for centroid
centroid_x, centroid_y = 1.0, 1.0
```

2. 计算梯度

```
loss_value = loss(sample_x, sample_y, centroid_x, centroid_y)
g_x, g_y = gradient(sample_x, sample_y, centroid_x, centroid_y)
```

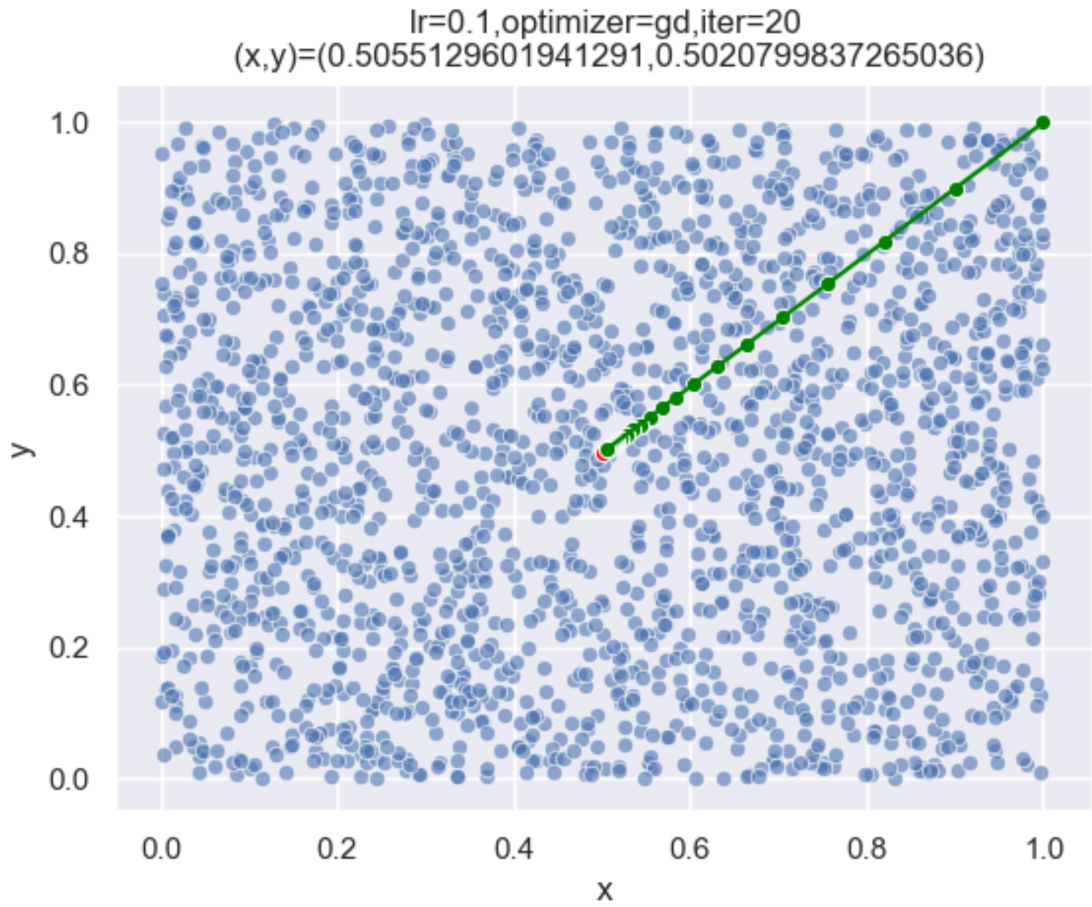
3. 更新质心

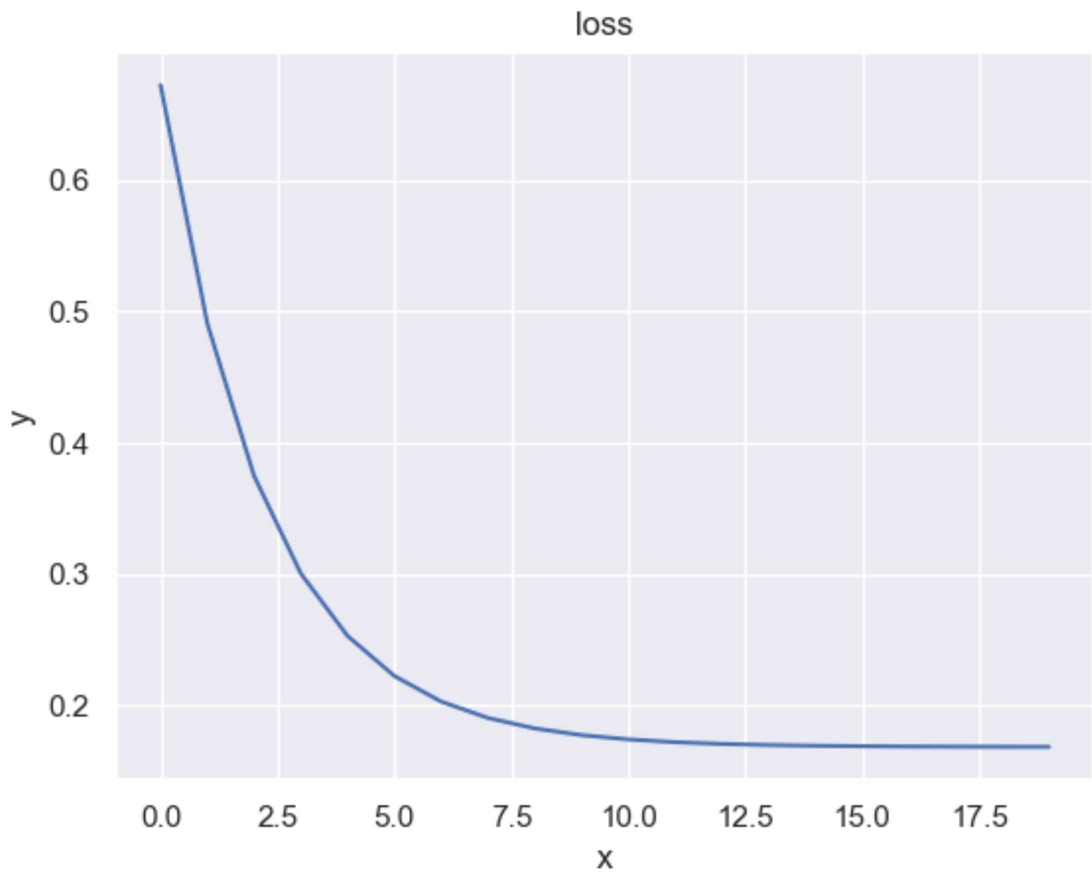
```
centroid_x, centroid_y = centroid_x - opt.lr * g_x, centroid_y - opt.lr *
g_y
```

4. 重复2和3直至loss收敛

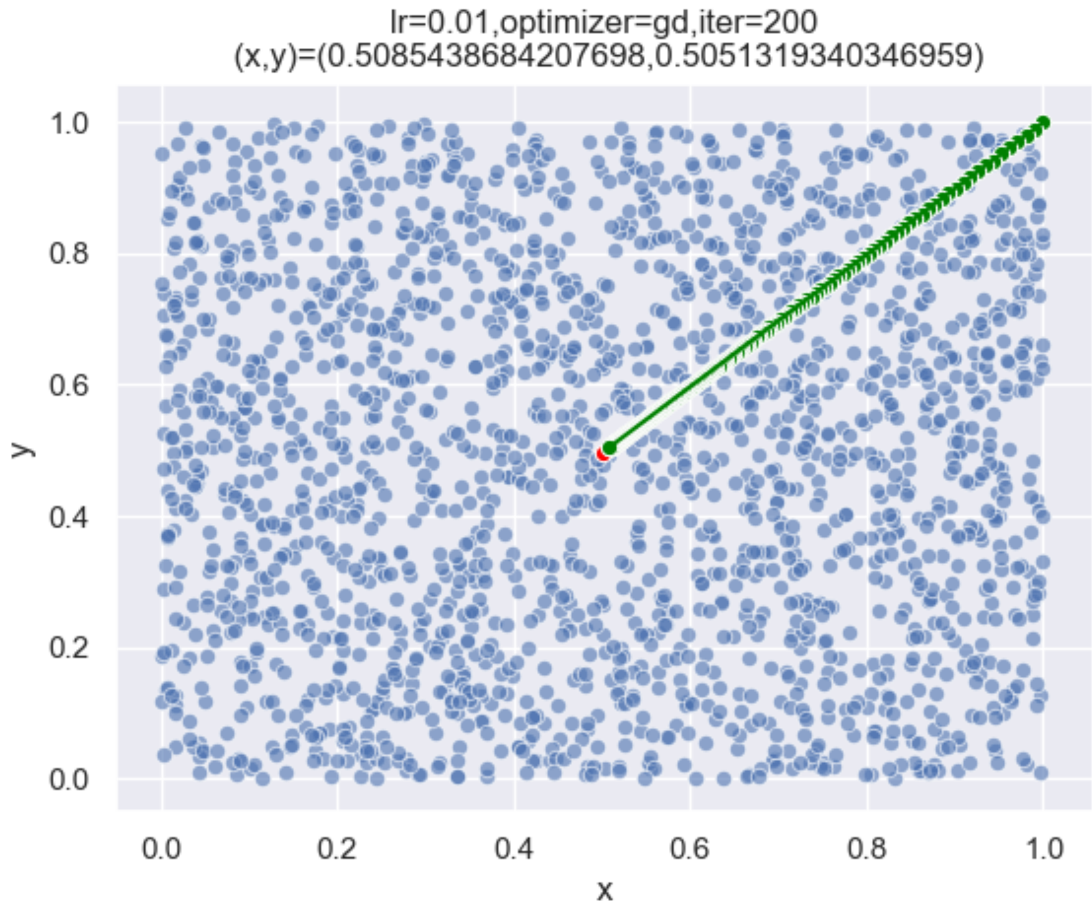
在上述步骤中，主要是有两个参数需要指定，一个为学习率(learning rate)，一个为迭代数。接下来分别调整以上两个参数做对照实验：

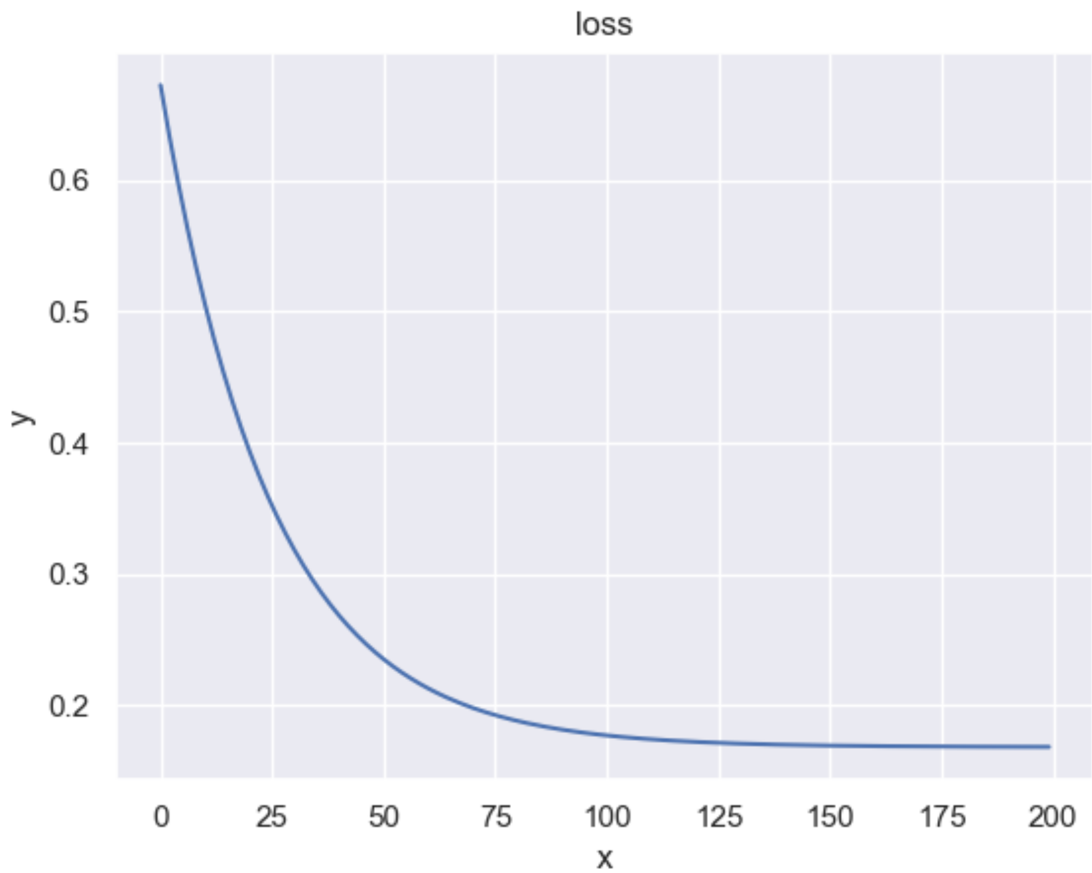
lr=0.1 iter=20





lr=0.01 iter=200


lr=0.01,optimizer=gd,iter=200
(x,y)=(0.5085438684207698,0.5051319340346959)
y
x
0.0
0.2
0.4
0.6
0.8
1.0


loss
y
x
0.2
0.3
0.4
0.5
0.6
0
25
50
75
100
125
150
175
200

Apply stochastic gradient descent (SGD) to find the centroid. Can you say in simple words, what the algorithm is doing?

随机梯度下降与梯度下降的区别在于，随机梯度下降每次使用**部分**数据参与计算梯度，而梯度下降使用**全部**数据参与计算梯度。随机梯度下降的优点包括减少过拟合，提升训练速度。

所以相比梯度下降，随机梯度下降步骤如下：（**唯一的区别就在于2**）

1. 初始化质心

```
# init value for centroid
centroid_x, centroid_y = 1.0, 1.0
```

2. 从点中抽样数据

```
start_index = (iter_index * opt.bs) % opt.n
sample_x, sample_y = x[start_index:start_index + opt.bs],
y[start_index:start_index + opt.bs]
```

3. 使用抽样数据计算梯度

```
loss_value = loss(sample_x, sample_y, centroid_x, centroid_y)
g_x, g_y = gradient(sample_x, sample_y, centroid_x, centroid_y)
```

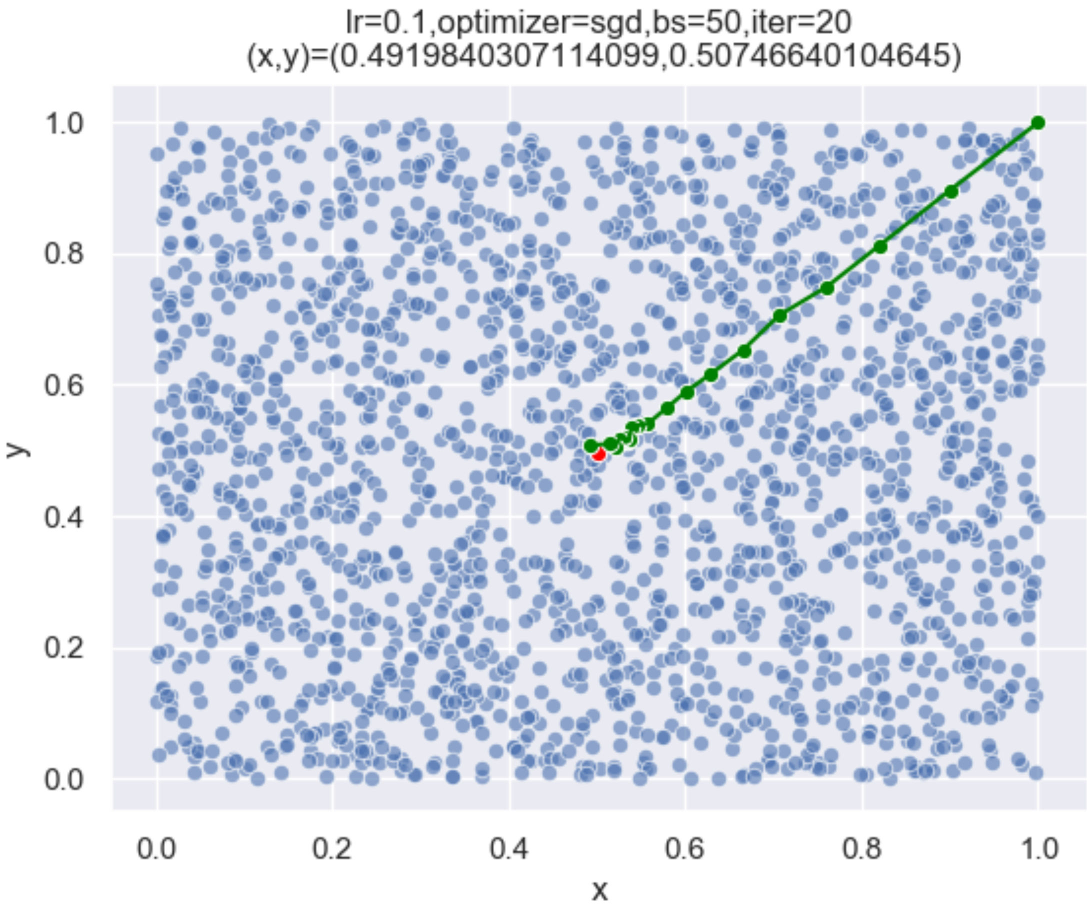
4. 更新质心

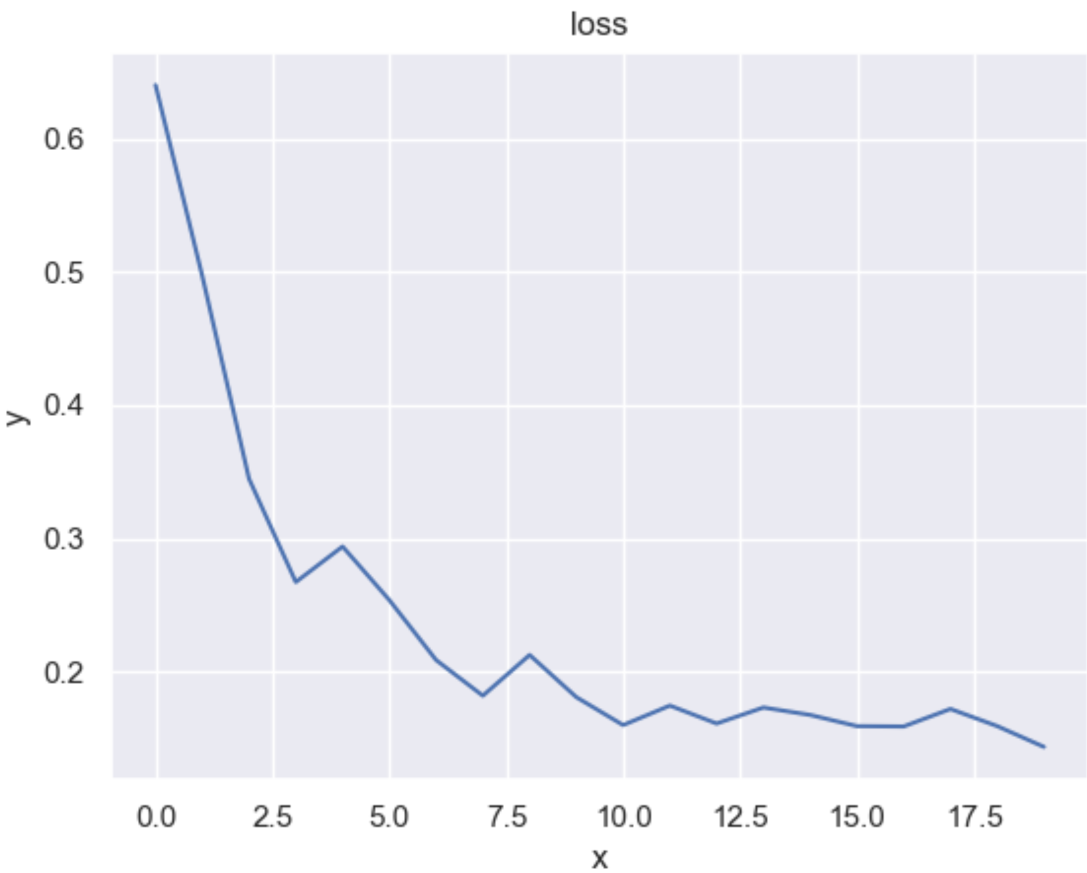
```
centroid_x, centroid_y = centroid_x - opt.lr * g_x, centroid_y - opt.lr *
g_y
```

5. 重复2、3、4直至loss收敛

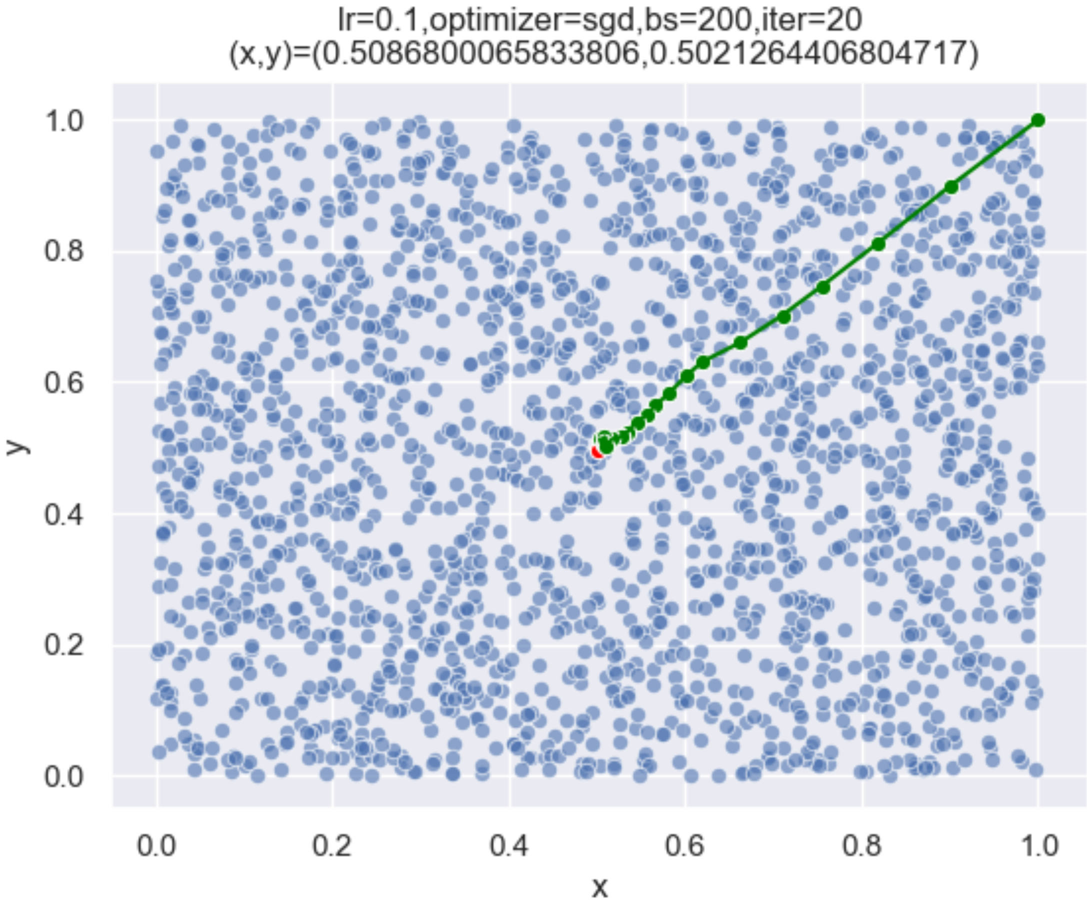
对于随机梯度下降，所需要的参数与梯度下降类似，但是多出一个batch size的参数，**意为每次用于计算梯度的点的个数**。接下来分别调整以上三个参数做对照实验：

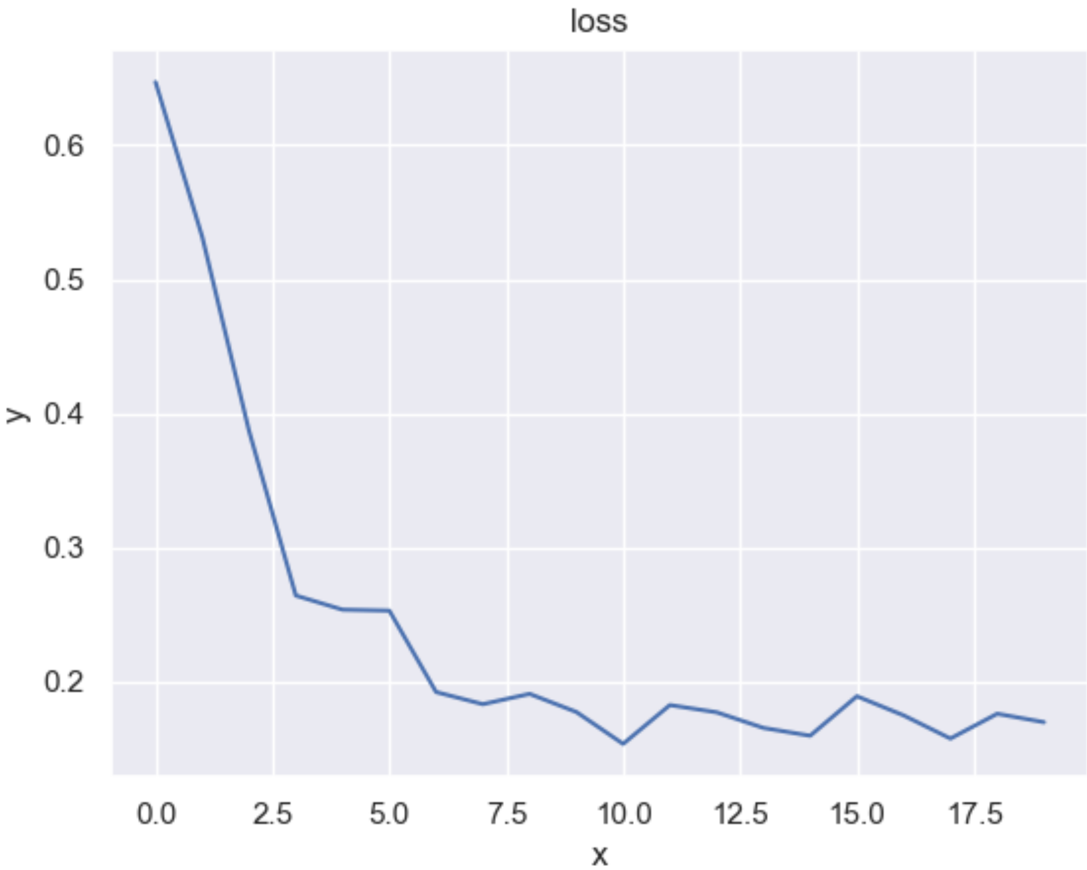
lr=0.1 iter=20 bs=50



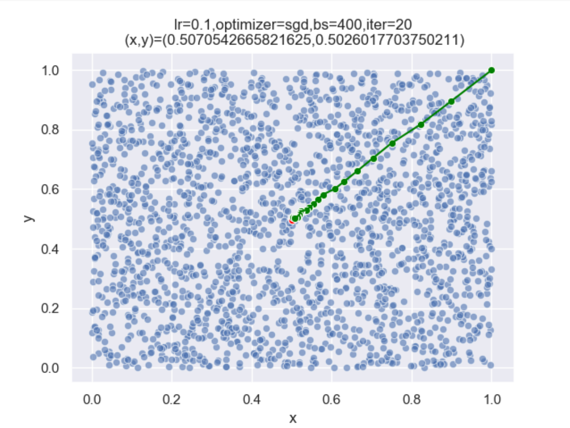


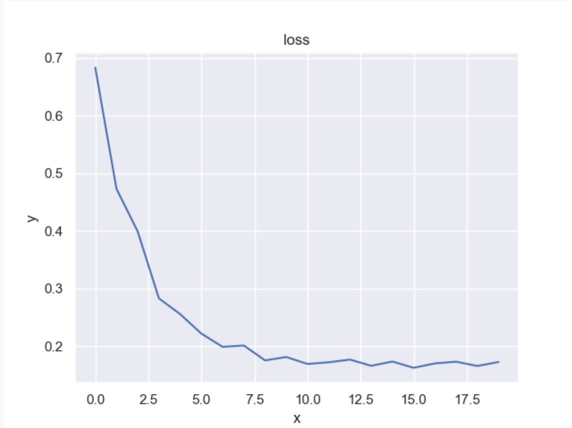
lr=0.1 iter=20 bs=200

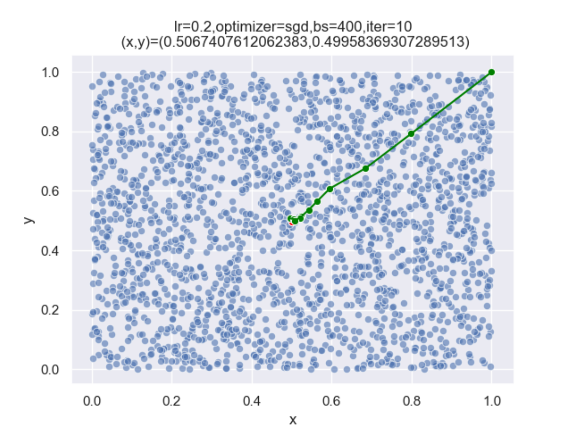

lr=0.1,optimizer=sgd,bs=200,iter=20
(x,y)=(0.5086800065833806,0.5021264406804717)
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x

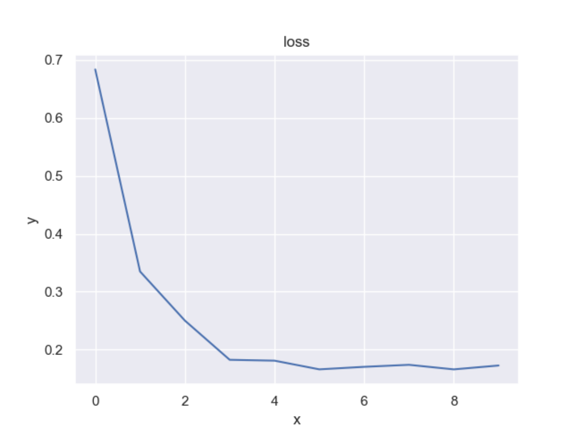

loss
0.6
0.5
0.4
0.3
0.2
y
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
x

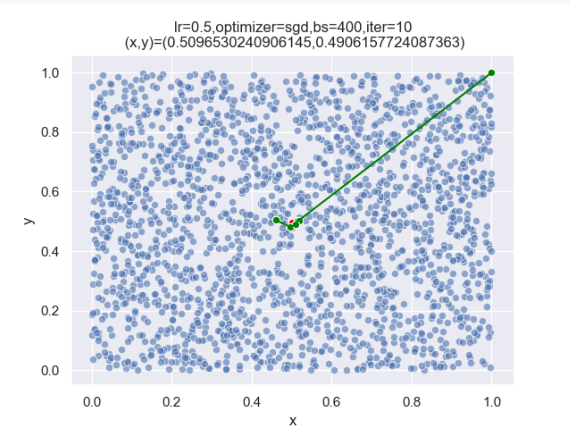
更多实验数据

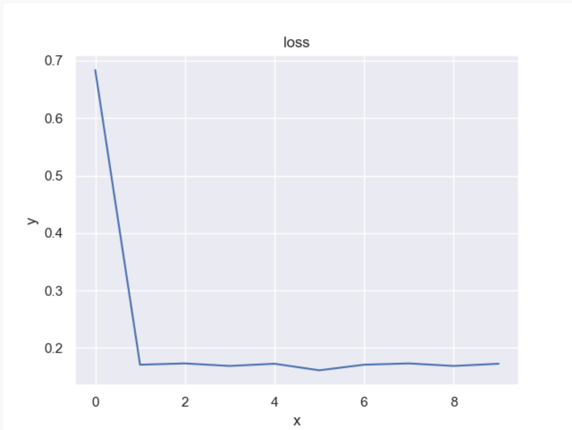

lr=0.1,optimizer=sgd,bs=400,iter=20
(x,y)=(0.5070542665821625,0.5026017703750211)
y
x

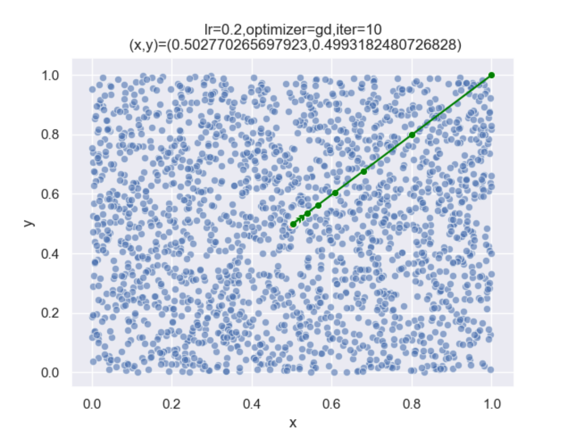

loss
y
x

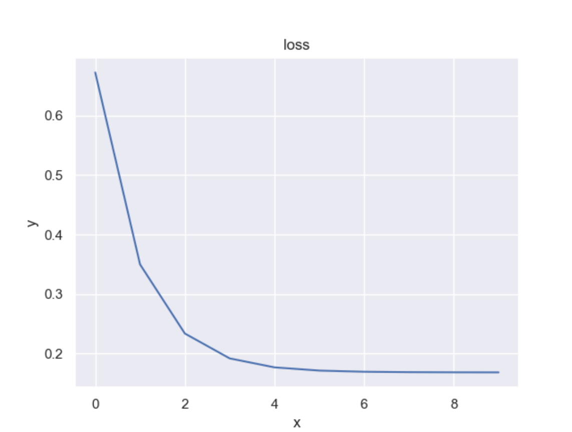

lr=0.2,optimizer=sgd,bs=400,iter=10
(x,y)=(0.5067407612062383,0.49958369307289513)
y
x

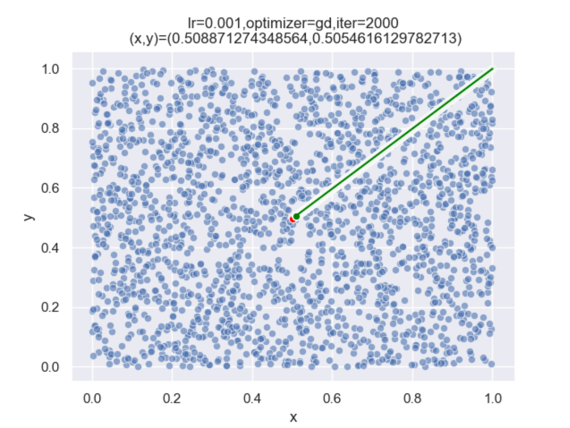

loss
y
x

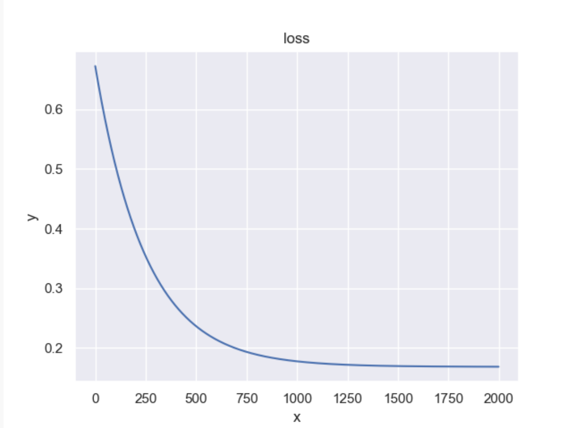

lr=0.5,optimizer=sgd,bs=400,iter=10
(x,y)=(0.5096530240906145,0.4906157724087363)
y
x


loss
y
x


lr=0.2,optimizer=gd,iter=10
(x,y)=(0.502770265697923,0.4993182480726828)
y
x


loss
y
x


lr=0.001,optimizer=gd,iter=2000
(x,y)=(0.508871274348564,0.5054616129782713)
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x


loss
0.6
0.5
0.4
0.3
0.2
y
0
250
500
750
1000
1250
1500
1750
2000
x