招新题解 1: 基础的线性回归

1. 生成数据点：

- 定义真实的函数模型，x和y从均匀分布中抽取，计算z值,绘制散点图。
- 使用n_point设置数据批量大小，np.random.uniform()进行均匀取点（我之前错用np.linspace生成的是等间隔的点导致loss很大）
- 绘制散点图用plt.figure()创建图像，用fig.add_subplot()建坐标轴，用ax.scatter()绘制散点图，也可以用camp=让她更美观

2. 使用Matplotlib库绘制出函数

- 这个要用np.linspace（连贯的）定义函数表达式，设置标签，用ax.plot_surface绘制曲面，用plot_3d绘制函数

3. 训练模型 定义拟合模型函数，初始化参数a, b, c, d随机值。定义超参数lr,epochs大小，利用之前生成的数据，使用梯度下降法最小化MSE损失。每20次迭代打印损失。

- 梯度下降法：采用平方损失公式loss = np.mean((Z_hat-Z)**2)，对参数求偏导
- 计算梯度时a = a- lr * da 和a = (1-k*lr)*a- lr * da的对比

```
epoch 0, loss: 0.09492
epoch 20, loss: 0.06620
epoch 40, loss: 0.04622
epoch 60, loss: 0.03230
epoch 80, loss: 0.02260
epoch 100, loss: 0.01583
epoch 120, loss: 0.01112
epoch 140, loss: 0.00782
epoch 160, loss: 0.00551
epoch 180, loss: 0.00390
```

```python
print(f"a = {a.item():.6f}
print(f"b = {b.item():.6f}
print(f"c = {c.item():.6f}
print(f"d = {d.item():.6f}
```

```
a = 2.999322 (true: 3.0)
b = 1.003022 (true: 1.0)
c = 1.016801 (true: 1.0)
d = 4.914977 (true: 5.0)
```

```python
    if epoch % 20 == 0:
        print(f'epoch {epoch }, loss: {loss:.5f}')
```

```
epoch 0, loss: 97.91282
epoch 20, loss: 1.72177
epoch 40, loss: 0.82885
epoch 60, loss: 0.39961
epoch 80, loss: 0.19295
epoch 100, loss: 0.09336
epoch 120, loss: 0.04531
epoch 140, loss: 0.02208
epoch 160, loss: 0.01083
epoch 180, loss: 0.00536
```

4. 调整学习率并观察变化。lr过小，形象地说就是梯度下降的步长太小，下降速度太慢，梯度消失，200 epochs后

loss仍较大；lr过小，就是梯度下降的步长太大，下降速度太快导致梯度爆炸，loss极大

```python
def model (x,y,a,b,c,d):
    return a*x**3+ b*y**4 +c*np.exp(y)+d
a,b,c,d= np.random.normal (0,0.01,4)
losses =[]
```

```python
lr = 0.001
num_epochs = 200


for epoch in range(num_epochs):
    Z_hat = model(x,y,a,b,c,d)
    loss =  np.mean((Z_hat-Z)**2)
    losses.append(loss)

    if epoch % 20 == 0:
        print(f'epoch {epoch }, loss: {loss:.5f}')
    e = Z_hat - Z
    da = np.mean(2 * e * x**3)
    db = np.mean(2 * e * y**4)
    dc = np.mean(2 * e * np.exp(y))
    dd = np.mean(2 * e)
    a -= lr * da
    b -= lr * db
    c -= lr * dc
    d -= lr * dd
```

```
epoch 0, loss: 152.87705
epoch 20, loss: 72.19058
epoch 40, loss: 50.36656
epoch 60, loss: 42.74826
epoch 80, loss: 39.81566
epoch 100, loss: 38.52836
epoch 120, loss: 37.83935
epoch 140, loss: 37.38269
epoch 160, loss: 37.02876
epoch 180, loss: 36.73022
```

```python
print(f"a = {a:.6f} (true: 3.0)")
```

```python
print(f"b = {b:.6f} (true: 1.0)")
print(f"c = {c:.6f} (true: 1.0)")
print(f"d = {d:.6f} ")
```

```
a = 2.036442 (true: 3.0)
b = -0.089948 (true: 1.0)
c = 1.882061 (true: 1.0)
```

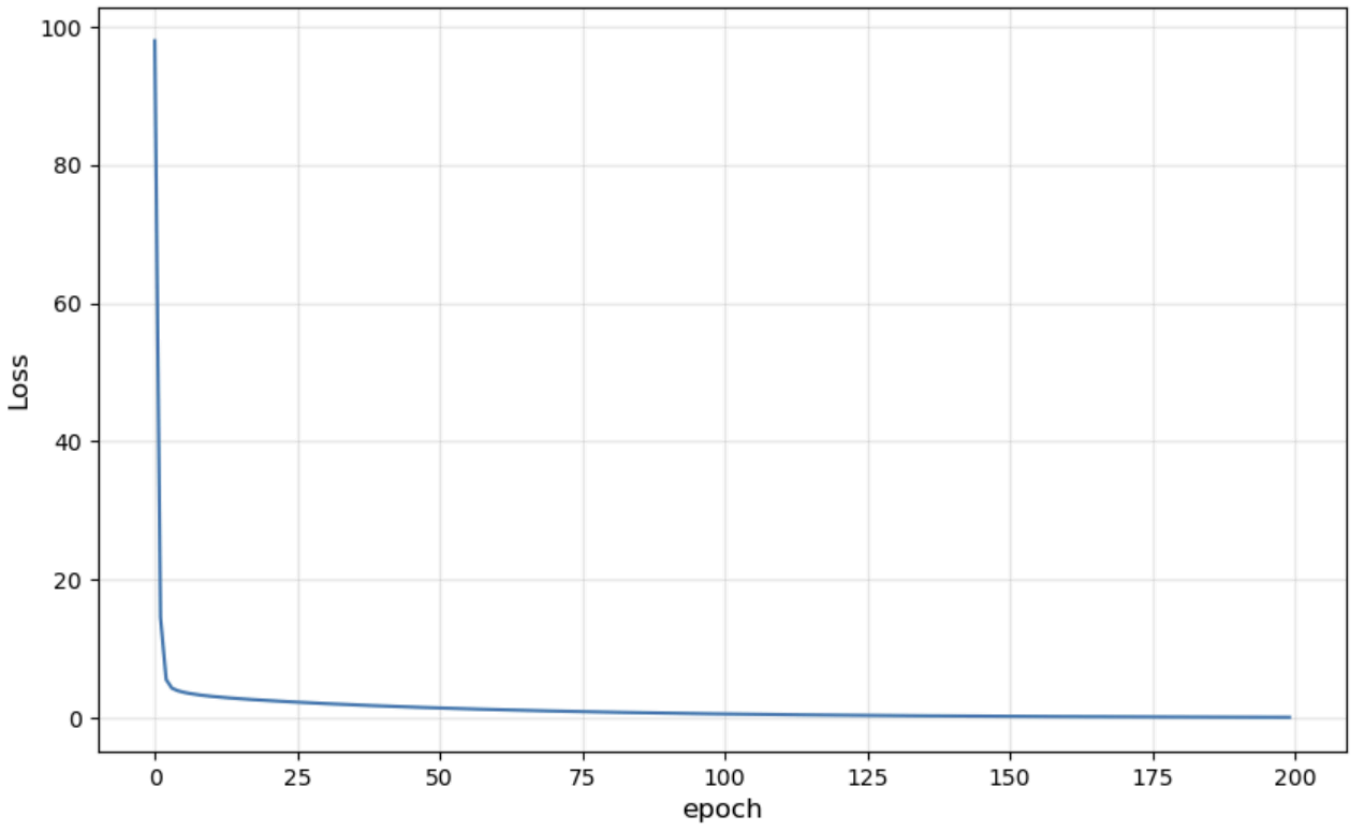
```python
lr = 0.2
num_epochs = 200
k =0.001

#定义循环
for epoch in range(num_epochs):
    Z_hat = model(x,y,a,b,c,d)
    #采用平方损失
    loss =  np.mean((Z_hat-Z)**2)
    losses.append(loss)
    #对参数求偏导
    e = Z_hat - Z
    da = np.mean(2 * e * x**3)
    db = np.mean(2 * e * y**4)
    dc = np.mean(2 * e * np.exp(y))
    dd = np.mean(2 * e)
    #更新参数
    #为使后面的循环梯度下降不要太小，我用了权重衰退的公式，测试时效果比a =
    a = (1-k*lr)*a- lr * da
    b = (1-k*lr)*b- lr * db
    c = (1-k*lr)*c- lr * dc
    d = (1-k*lr)*d- lr * dd
    #每20次迭代打印损失
    if epoch % 20 == 0:
        print(f'epoch {epoch }, loss: {loss:.5f}')
```

```
epoch 0, loss: 97.91282
epoch 20, loss: 927044139335399060077798379290355521111432192.00000
epoch 40, loss: 9329135706712406776954240189733888035237333248426306466097405107565782643843910264034426 88.00000
epoch 60, loss: 9388201633705434868238452499350839148428431542942451
1088128422738727793823233276701458290032017832119560215356324217172070
39411748864.00000
epoch 80, loss: 944764152714522318298769243788782022962565387960193054844764808682615936478184368078143949094526776091817310231845899782230547072110413171916210474203723760400812981724379 9257088.00000
epoch 100, loss: 950745775474037005999942762516381574041097186403473
```

```
928144853421757992302077571565668147616088722715136796694689799259180
737400792458197064000720492674900596653894644190466754050325138721338
08364751898938579699245948862464.00000
epoch 120, loss: 956765269919023637448323333723456950347753369712843 7
0586763902011481506881387447267610074034647087725706206382860940383 84
6309503905141013654891947646175909921112168678234121688681600846286 78
0712268950775950542610773359722428758380393749029846429092495359890 67
```

5. 绘制损失曲线。用plt.plot(losses)然后设置xy轴即可

```python
plt.figure(figsize=(10, 6))
plt.plot(losses)
plt.xlabel('epoch', fontsize=12)
plt.ylabel('Loss', fontsize=12)
plt.grid(True, alpha=0.3)
plt.show()
```



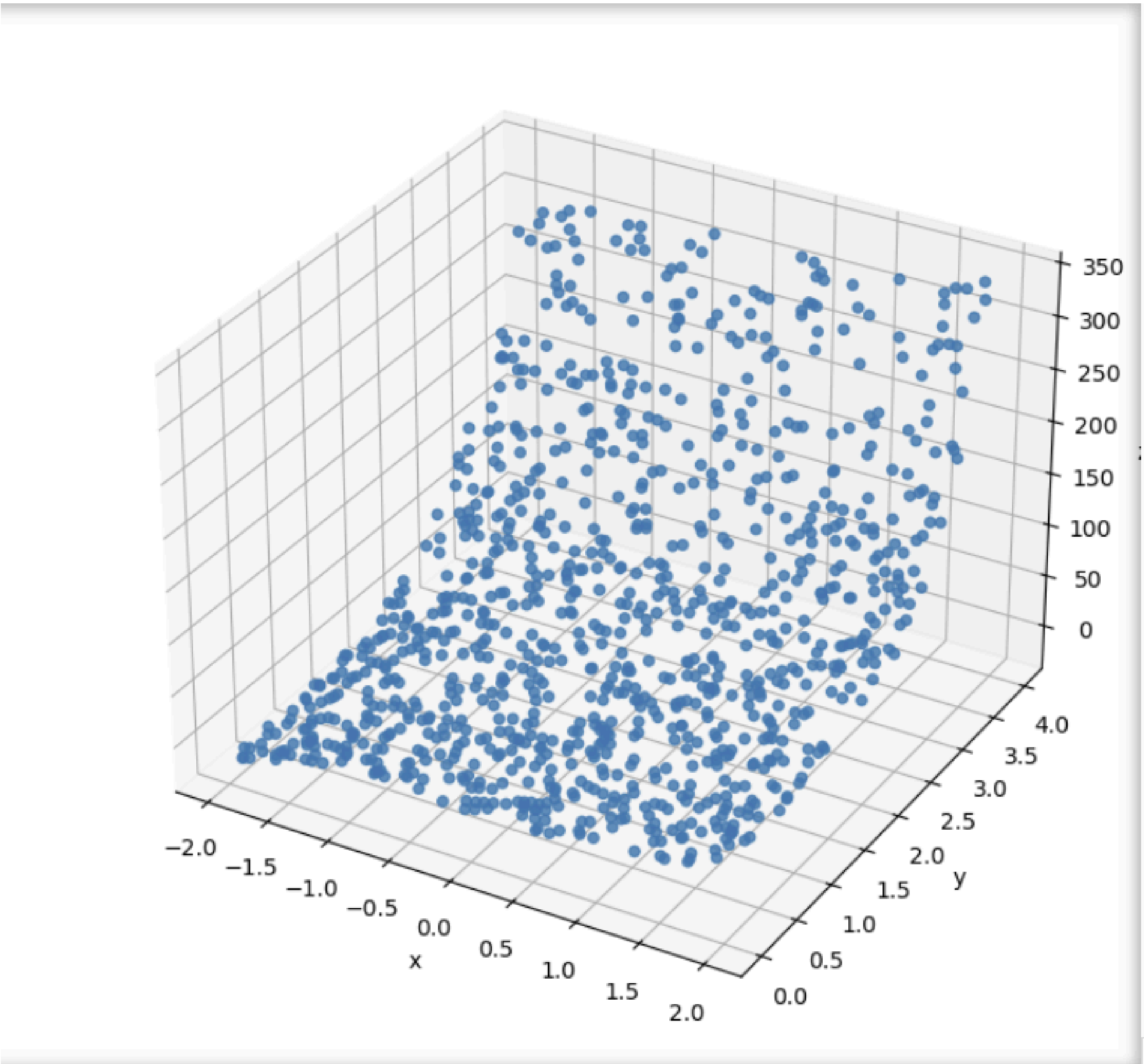
- 损失值随迭代次数的变化曲线总能画出，但是损失函数L(a,b,c,d)有五维，人脑只能理解3维图像，不可能画出

6. 调整y范围

- 问题：散点图的形状大大改变，毕竟自变量范围变化使取到的点完全不同了；出现了梯度爆炸，原因可能是y定义域大小没变但正负不对称且绝对值增大，又经过y**4和exp(y)产生了指数级的增大效

果,求偏导时使梯度值急剧增大

- 解决方案：大大减小lr；可以使用权重衰减法（已用在代码中）或暂退法



```python
    if epoch % 20 == 0:
        print(f'epoch {epoch }, loss: {loss:.5f}')
```

```
poch 0, loss: 25188.26795
poch 20, loss: 371151377841178514737975037983454540297782422946
8760521616281928952514330848410891241210031045563198208.00000
poch 40, loss: 546982918976127232007907860221110112572146067600
581831880723359642046646535121487322021859754831541616925447925
101574025038611913743850623885921468027356255832228698585538104
4487009214534693749721661440.00000
poch 60, loss: 806113978053648430350863883886937838807612289736
904840459694070215989045700190173226115290330698348298436607138
076112968230412302970367298364391696129499285926686519738378598
961854572661046755107299611037167164797081603042717764784951397
574954695771888458868787292617833286624345299013176070588753510
.00000
poch 80, loss: inf
poch 100, loss: inf
poch 120, loss: inf
poch 140, loss: nan
poch 160, loss: nan
poch 180, loss: nan
```

7. 小总结

(1)过程 生成数据-定义拟合函数-初始化参数-设置超参数-定义loss公式-求梯度并更新参数-定义循环-训练！
简单推导

对 $l(X,y,w,b)=\frac{1}{2n}\sum_{i=1}^{n}[y_i-(Xw+b)]$

此题中 $w=a, b, c, d$. 即 $w=\begin{bmatrix}a\\b\\c\\d\end{bmatrix}$

求梯度方向.

只对自变量求导
其他视为常量

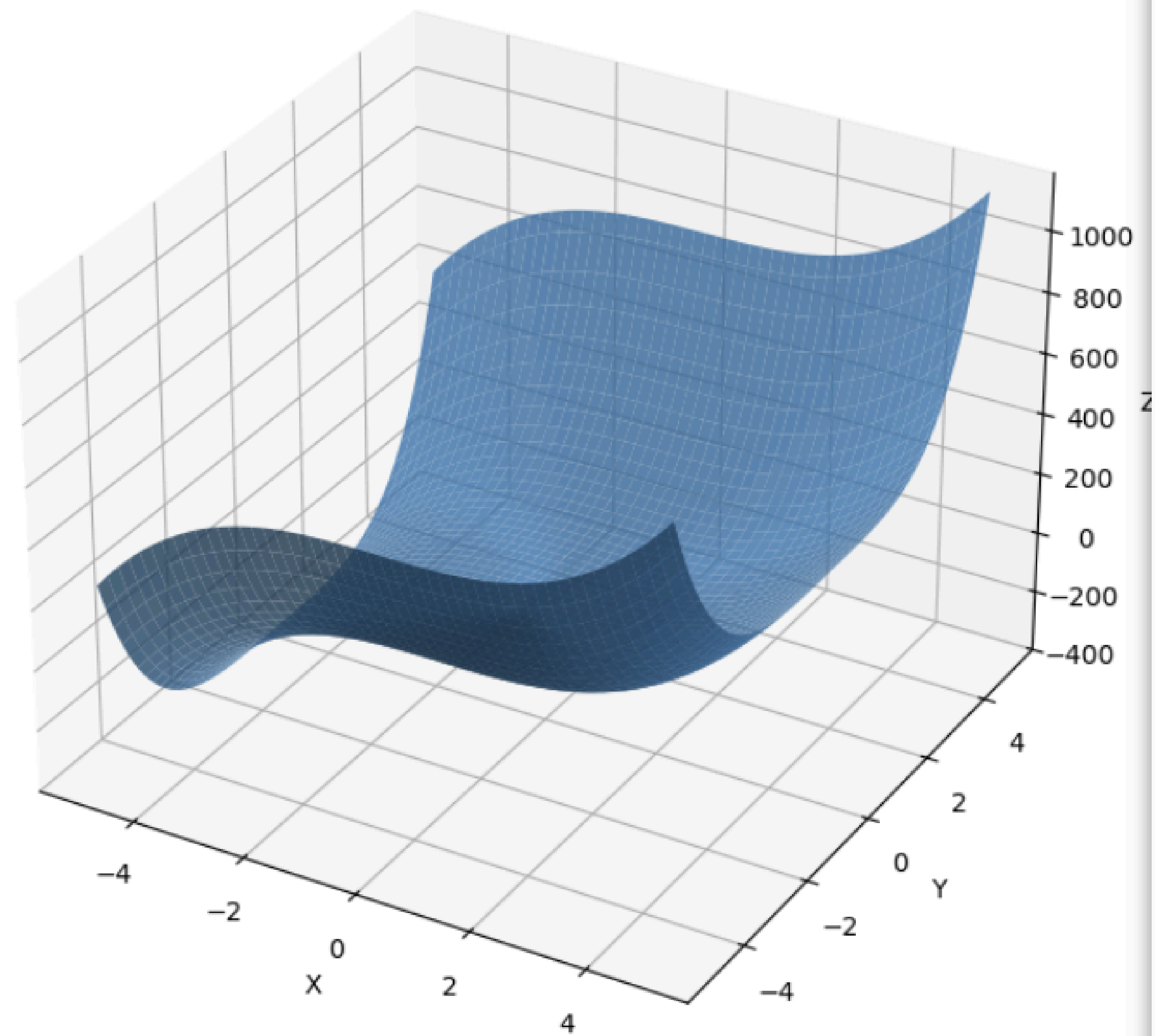
$$\frac{dl}{dw_{t-1}} \text{ 即 } = \begin{bmatrix}\frac{dl}{da_{t-1}}\\ \frac{dl}{db_{t-1}}\\ \frac{dl}{dc_{t-1}}\\ \frac{dl}{dd_{t-1}}\end{bmatrix} \text{ 在当前参数 } \begin{bmatrix}a_{t-1}\\ b_{t-1}\\ c_{t-1}\\ d_{t-1}\end{bmatrix} \text{ 处的偏导} \rightarrow \text{梯度}$$

∴ 参数更新方向确定. lr 设置好.

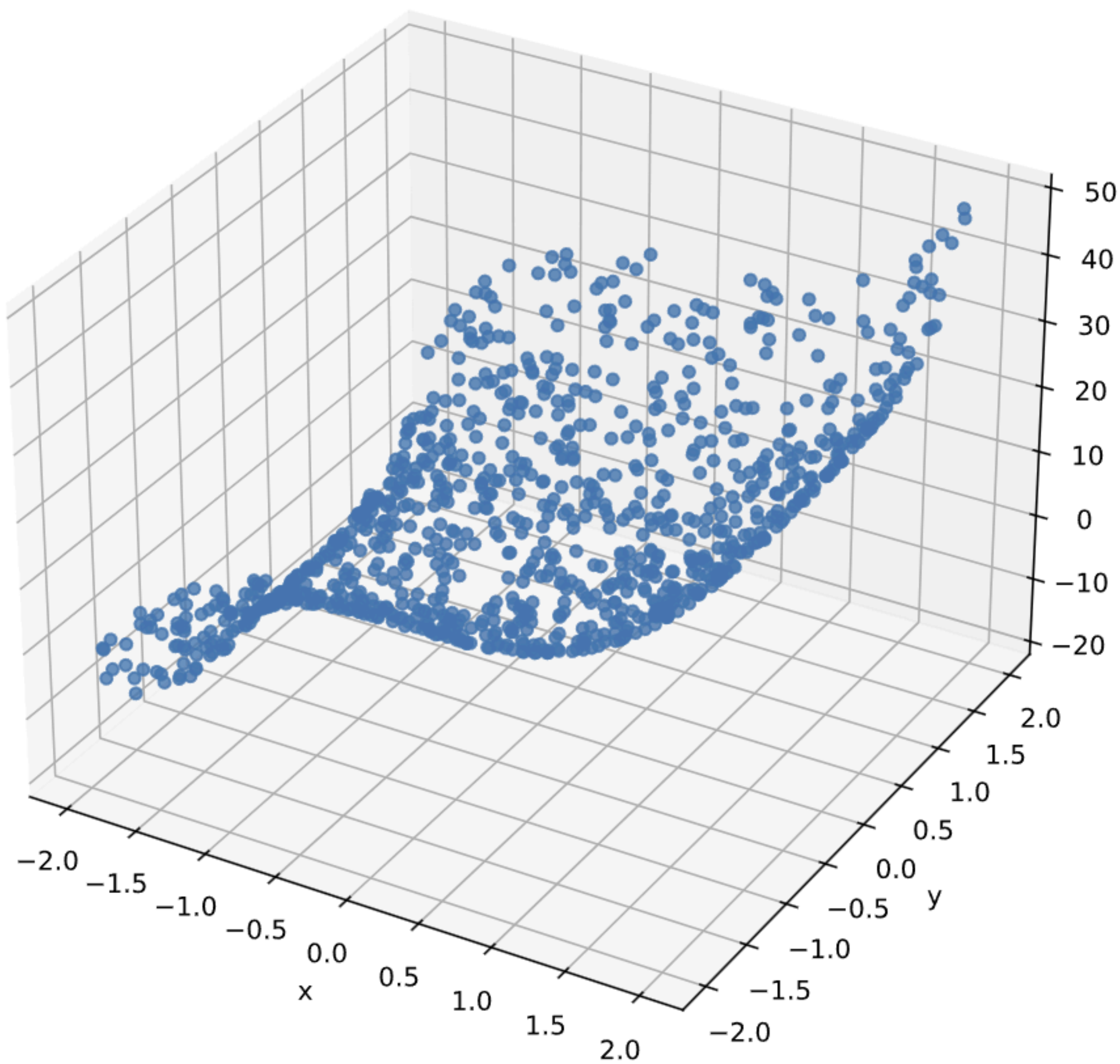
有 $w = w_{t-1} - lr \cdot \frac{dl}{dw_{t-1}}$

(2)解决思路 先读书看视频了解公式的数学推导和基本过程，跟着书中<3.2. 线性回归的从零开始实现>跑了一遍代码，用AI了解了每一个函数/工具的用处，试着用其大框架做招新题，换了很多函数工具，一直error一直改（真的每天除了上数学就在学机器学习，没招了）

(3) 学习收获 了解了很多函数工具，最重要的是有了解决线性回归问题（甚至是机器学习问题）的基本思路，认识到ml的目标就是最小化损失（后面就是不同方法的尝试）；算提前预习了点高数，将课本上的知识用起来很有满足感，对偏导，范数，方向向量，矩阵乘法到底有什么用有了理解；熟练使用AI了；被折磨的过程中锻炼了意志，现已人淡如菊



1000
800
600
400
200
0
−200
−400
Z
−4
−2
0
2
4
X
4
2
0
−2
−4
Y



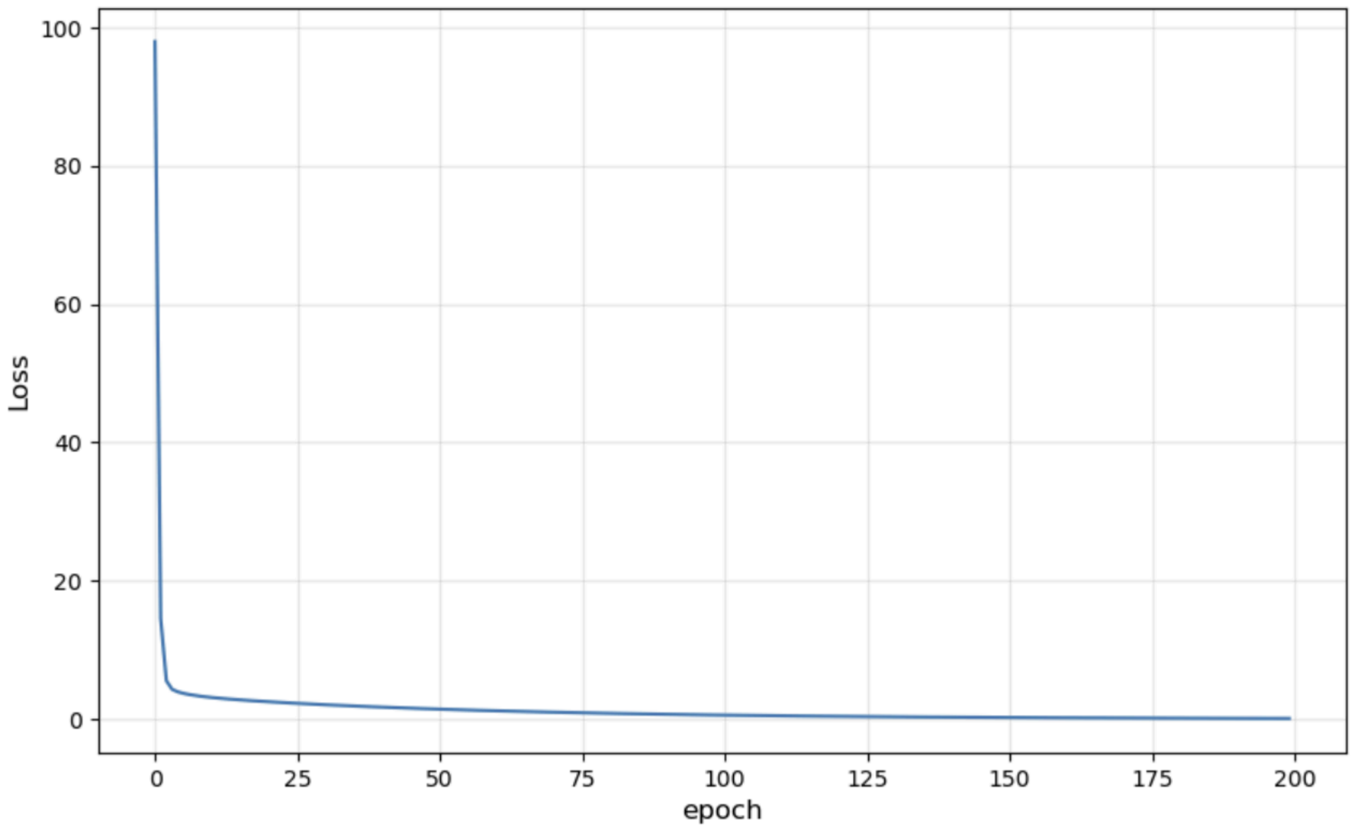
x
y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
−20
−10
0
10
20
30
40
50

```
epoch 0, loss: 0.09492
epoch 20, loss: 0.06620
epoch 40, loss: 0.04622
epoch 60, loss: 0.03230
epoch 80, loss: 0.02260
epoch 100, loss: 0.01583
epoch 120, loss: 0.01112
epoch 140, loss: 0.00782
epoch 160, loss: 0.00551
epoch 180, loss: 0.00390
```

```python
print(f"a = {a.item():.6f}
print(f"b = {b.item():.6f}
print(f"c = {c.item():.6f}
print(f"d = {d.item():.6f}
```

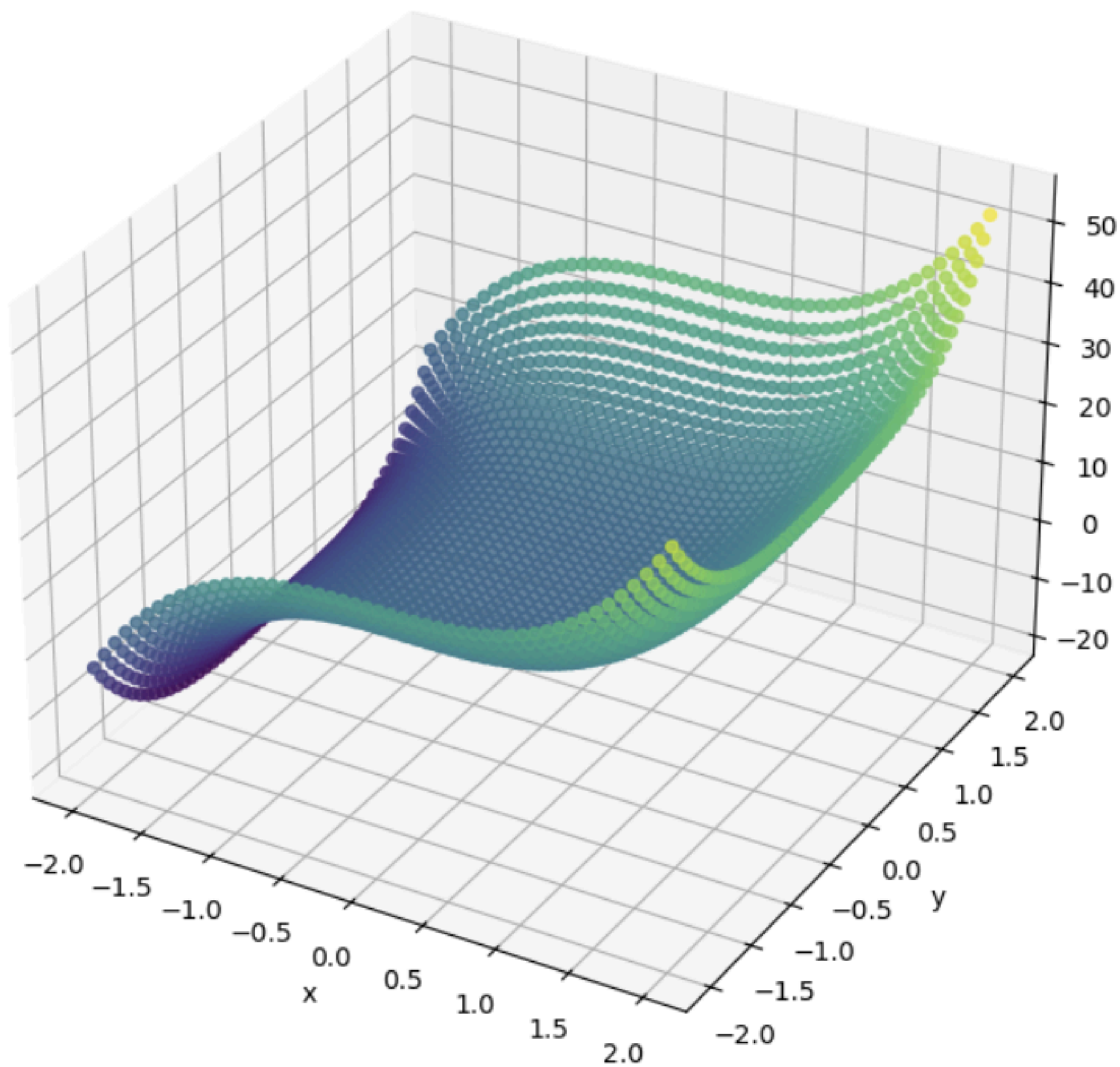
```
a = 2.999322 (true: 3.0)
b = 1.003022 (true: 1.0)
c = 1.016801 (true: 1.0)
d = 4.914977 (true: 5.0)
```

```
plt.figure(figsize=(10, 6))
plt.plot(losses)
plt.xlabel('epoch', fontsize=12)
plt.ylabel('Loss', fontsize=12)
plt.grid(True, alpha=0.3)
plt.show()
```



用np.linspace

```
plt.show()
```



50
40
30
20
10
0
−10
−20
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x