



もう恋はしないと誓ったばかりが、
二度目の恋に気づいたのは、
高校二年生の終わりだった。
それまで彼女は、いつもぼくの側にいて、
すみれの花のように優しく微笑んでくれた。
あの夕暮れのまばゆい金色の光に満ちた小さな部屋で、
いつも幸せそうに本のページをめくり、
ぼくにたくさんの物語を聞かせてくれた。

小学少女で恋する神話集 1110

Exercise Thinking

卷三 动手学深度学习 Pytorch 习题解

作者：latalealice

日期：2025/05/09

目 录

序言	1
第一章 引言	2
第二章 预备知识	3
2.1 数据操作	3
2.1.1 运行代码,将条件语句 $X = Y$ 更改为 $X < Y$ 或 $X > Y$,看看可以得到什么样的张量	3
2.1.2 用其他形状(例如三维张量)替换广播机制中按元素操作的两个张量,结果如何	3
2.2 数据预处理	4
2.2.1 创建包含更多行和列的原始数据集	4
2.3 线性代数	4
2.3.1 证明一个矩阵 A 的转置的转置是 A	4
2.3.2 给出两个矩阵 A 和 B ,证明“它们转置的和”等于“它们和的转置”	5
2.3.3 给定任意方阵 A , $A + A^T$ 总是对称的吗	5
2.3.4 定义形状 $(2,3,4)$ 的张量 X , $\text{len}(X)$ 的输出结果是什么	5
2.3.5 对于任意形状的张量 X , $\text{len}(X)$ 是否总是对应于 X 特定轴的长度?这个轴是什么	5
2.3.6 运行 $A/A.\text{sum}(\text{axis}=1)$,看看会发生什么.请分析一下原因	5
2.3.7 考虑一个具有形状 $(2,3,4)$ 的张量,在轴 0、1、2 上的求和输出是什么形状	6
2.3.8 为 linalg.norm 函数提供 3 个或更多轴的张量,并观察其输出.对于任意形状的张量这个函数计算得到什么	6
2.4 微积分	6
2.4.1 绘制函数 $y = f(x) = x^3 - \frac{1}{x}$ 和其在 $x = 1$ 处切线的图像	6
2.4.2 求函数 $f(x) = 3x_1^2 + 5e^{x_2}$ 的梯度	7
2.4.3 函数 $f(x) = \ x\ _2$ 的梯度是什么	7
2.4.4 尝试写出函数 $u = f(x, y, z)$,其中 $x = x(a, b)$, $y = y(a, b)$, $z = z(a, b)$ 的链式法则	7
2.5 自动微分	8
2.5.1 为什么计算二阶导数比一阶导数的开销要更大	8
2.5.2 在运行反向传播函数之后,立即再次运行它,会发生什么	8
2.5.3 在控制流的例子中,计算 d 关于 a 的导数,如果将变量 a 更改为随机向量或矩阵,会发生什 么	8
2.5.4 重新设计一个求控制流梯度的例子,运行并分析结果	9
2.5.5 使 $f(x) = \sin x$,绘制 $f(x)$ 和 $\frac{df(x)}{dx}$ 的图像,其中后者不使用 $f'(x) = \cos x$	9
2.6 概率	9
2.6.1 进行 $m = 500$ 组实验,每组抽取 $n = 10$ 个样本.改变 m 和 n ,观察和分析实验结果	10
2.6.2 给定两个概率为 $P(A)$ 和 $P(B)$ 的事件,计算 $P(A \cup B)$ 和 $P(A \cap B)$ 的上限和下限(使用韦恩图 来展示这些情况)	10
2.6.3 假设有一系列随机变量,例如 A, B 和 C ,其中 B 只依赖于 A ,而 C 只依赖于 B ,能简化联合概率吗 (这是一个马尔可夫链)	11
2.6.4 艾滋病测试问题	11
第三章 线性回归网络	12
3.1 线性回归网络	12
3.1.1 假设有一些数据 $x_1, x_2, \dots, x_n \in R$,目标是找到一个常数 b ,使得 $\sum_i (x_i - b)^2$ 最小化	12
3.1.2 推导出使用平方误差的线性回归优化问题的解析解,忽略偏置 b 简化问题(可以通过向 X 添加 所有值为 1 的一列来做到这一点)	12
3.1.3 假定控制附加噪声的 ϵ 噪声模型是指数分布, $p(\epsilon) = \frac{1}{2} \exp(- \epsilon)$	12
3.2 从零开始实现	13
3.2.1 如果将权重 w 初始化为零,会发生什么?算法仍然有效吗	13
3.2.2 为电压和电流的关系建立一个模型,自动微分可以用来学习模型的参数吗	13
3.2.3 基于普朗克定律使用光谱能量密度来确定物体的温度	14

3.2.4	计算二阶导数时可能会遇到什么问题?如何解决	14
3.2.5	为什么在 <code>squared_loss</code> 函数中需要使用 <code>reshape</code> 函数	15
3.2.6	尝试使用不同的学习率,观察损失函数值下降的快慢	15
3.2.7	如果样本个数不能被批量大小整除, <code>data_iter</code> 函数的行为会有什么变化	16
3.3	简洁实现	16
3.3.1	如果将小批量的总损失替换为小批量损失的平均值,需要如何更改学习率	16
3.3.2	查看深度学习框架文档,它们提供了哪些损失函数和初始化方法	17
3.3.3	访问线性回归中的梯度	17

序言

此 PDF 为习题解答,需求的库如下

```
1 from d2l import torch as d2l
2 from matplotlib_inline import backend_inline
3 from matplotlib_venn import venn2
4 from torch.distributions import multinomial
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import os
8 import pandas as pd
9 import seaborn as sns
10 import torch
11 import torch.nn as nn
```

py

第一章 引言

1. 你当前正在编写的代码的哪些部分可以“学习”,即通过学习和自动确定代码中所做的设计选择来改进?你的代码是否包含启发式设计选择?
2. 你遇到的哪些问题有许多解决它们的样本,但没有具体的自动化方法?这些可能是使用深度学习的主要候选者.
3. 如果把人工智能的发展看作一场新的工业革命,那么算法和数据之间的关系是什么?它类似于蒸汽机和煤吗?根本区别是什么?
4. 你还可以在哪里应用端到端的训练方法,比如图 1.1.2 、物理、工程和计量经济学?

第二章 预备知识

2.1 数据操作

2.1.1 运行代码,将条件语句 $X == Y$ 更改为 $X < Y$ 或 $X > Y$,看看可以得到什么样的张量

代码

```
1 X = torch.arange(12,  
dtype=torch.float32).reshape((3,4))  
2 Y = torch.tensor([[2.0, 1, 4, 3], [1, 2,  
3, 4], [4, 3, 2, 1]])  
3 X > Y, X == Y, X < Y
```

结果

```
1 tensor([[False, False, False, False],  
2         [ True,  True,  True,  True],  
3         [ True,  True,  True,  True]]),  
4 tensor([[False,  True, False,  True],  
5         [False, False, False, False],  
6         [False, False, False, False]]),  
7 tensor([[ True, False,  True, False],  
8         [False, False, False, False],  
9         [False, False, False, False]])
```

2.1.2 用其他形状(例如三维张量)替换广播机制中按元素操作的两个张量,结果如何

代码

```
1 a =  
torch.arange(25).reshape((-1,5,1))  
2 b = torch.arange(10).reshape((-1,1,2))  
3 a+b  
4 # 广播矩阵,复制a的列,b的行
```

结果

```
1 tensor([[[[ 0,  1],  
2          [ 1,  2],  
3          [ 2,  3],  
4          [ 3,  4],  
5          [ 4,  5]],  
6         [[ 7,  8],  
7          [ 8,  9],  
8          [ 9, 10],  
9          [10, 11],  
10         [11, 12]],  
11        [[14, 15],  
12         [15, 16],  
13         [16, 17],  
14         [17, 18],  
15         [18, 19]],  
16        [[21, 22],  
17         [22, 23],  
18         [23, 24],  
19         [24, 25],  
20         [25, 26]],  
21        [[28, 29],  
22         [29, 30],  
23         [30, 31],  
24         [31, 32],  
25         [32, 33]]]])
```

2.2 数据预处理

2.2.1 创建包含更多行和列的原始数据集

- 删除缺失值最多的列
- 将预处理后的数据集转换为张量格式

代码

```
1 os.makedirs(os.path.join '..',  
2 'data'), exist_ok=True)  
3 data_file = os.path.join '..', 'data',  
4 'house_tiny.csv')  
5 with open(data_file, 'w') as f:  
6     f.write('NumRooms,Alley,Price\n') # 列  
7     # 名  
8     f.write('NA,Pave,127500\n') # 每行表示  
9     # 一个数据样本  
10    f.write('2,NA,106000\n')  
11    f.write('4,NA,178100\n')  
12    f.write('NA,NA,140000\n')  
13    f.write('3,Pave,122000\n')  
14    # 1. 删除缺失值最多的列  
15    # 计算每列的缺失值数量  
16    missing_values = data.isnull().sum()  
17    # 找出缺失值最多的列名  
18    column_to_drop =  
19    missing_values.idxmax()  
20    # 删除该列  
21    data_preprocessed =  
22    data.drop(column_to_drop, axis=1)  
23    print(data_preprocessed)  
24    # 2. 将预处理后的数据集转换为张量格式  
25    # 首先处理剩余的缺失值(用均值填充数值列)  
26    data_preprocessed['NumRooms'] =  
27    data_preprocessed['NumRooms'].  
28    fillna(data_preprocessed['NumRooms'].mean())  
29    # 转换为张量  
30    tensor_data =  
31    torch.tensor(data_preprocessed.values,  
32    dtype=torch.float32)
```

结果

1		NumRooms	Price
2	0	NaN	127500
3	1	2.0	106000
4	2	4.0	178100
5	3	NaN	140000
6	4	3.0	122000
7	tensor([[3.0000e+00, 1.2750e+05],		
8	[2.0000e+00, 1.0600e+05],		
9	[4.0000e+00, 1.7810e+05],		
10	[3.0000e+00, 1.4000e+05],		
11	[3.0000e+00, 1.2200e+05]])		

2.3 线性代数

2.3.1 证明一个矩阵 A 的转置的转置是 A

代码

结果

```
1 A = torch.arange(20).reshape(5, -1)
2 A.T.T == A
```

```
1 tensor([[True, True, True, True],
2         [True, True, True, True],
3         [True, True, True, True],
4         [True, True, True, True],
5         [True, True, True, True]])
```

2.3.2 给出两个矩阵 A 和 B , 证明“它们转置的和”等于“它们和的转置”

代码

```
1 A = torch.randn(3,4)
2 B = torch.randn(3,4)
3 (A+B).T == A.T+B.T
```

结果

```
1 tensor([[True, True, True],
2         [True, True, True],
3         [True, True, True],
4         [True, True, True]])
```

2.3.3 给定任意方阵 A , $A + A^T$ 总是对称的吗

代码

```
1 A=torch.randn(4,4)
2 A+A.T == (A.T+A).T
```

结果

```
1 tensor([[True, True, True, True],
2         [True, True, True, True],
3         [True, True, True, True],
4         [True, True, True, True]])
```

2.3.4 定义形状(2,3,4)的张量 X , $\text{len}(X)$ 的输出结果是什么

代码

```
1 X = torch.arange(24).reshape(2,3,4)
2 len(X)
```

结果

```
1 2
```

2.3.5 对于任意形状的张量 X , $\text{len}(X)$ 是否总是对应于 X 特定轴的长度? 这个轴是什么

代码

```
1 # 1维张量
2 X = torch.tensor([1, 2, 3])
3 len(X)
4 # 2维张量
5 X = torch.tensor([[1, 2], [3, 4], [5, 6]])
6 len(X)
7 # 3维张量
8 X = torch.rand(2, 3, 4)
9 len(X)
```

结果

```
1 # len(X)总是返回张量在第一个轴axis=0的
   长度
2 3
3 3
4 2
```

2.3.6 运行 $A/A.\text{sum}(\text{axis}=1)$, 看看会发生什么. 请分析一下原因

代码

结果


```

1 A=torch.arange(25).reshape(5,-1)
2 A
3 A/A.sum(axis=1)
4 tensor([[ 0,  1,  2,  3,  4],
5         [ 5,  6,  7,  8,  9],
6         [10, 11, 12, 13, 14],
7         [15, 16, 17, 18, 19],
8         [20, 21, 22, 23, 24]])

```

```

1 tensor([
2   [0.0000, 0.0286, 0.0333, 0.0353,
3     0.0364],
4   [0.5000, 0.1714, 0.1167, 0.0941,
5     0.0818],
6   [1.0000, 0.3143, 0.2000, 0.1529,
7     0.1273],
8   [1.5000, 0.4571, 0.2833, 0.2118,
9     0.1727],
10  [2.0000, 0.6000, 0.3667, 0.2706,
11    0.2182]])

```

2.3.7 考虑一个具有形状(2,3,4)的张量,在轴 0、1、2 上的求和输出是什么形状

代码

```

1 A = torch.arange(24).reshape(2,3,4)
2 A
3 A.sum(axis=0)
4 A.sum(axis=1)
5 A.sum(axis=2)

```

结果

```

1 tensor([[12, 14, 16, 18],
2         [20, 22, 24, 26],
3         [28, 30, 32, 34]])
4 tensor([[12, 15, 18, 21],
5         [48, 51, 54, 57]])
6 tensor([[ 6, 22, 38],
7         [54, 70, 86]])

```

2.3.8 为 linalg.norm 函数提供 3 个或更多轴的张量,并观察其输出.对于任意形状的张量这个函数计算得到什么

代码

```

1 # 创建一个3x4x5的张量
2 tensor_3d = torch.randn(3, 4, 5)
3 norm_default = np.linalg.norm(tensor_3d)
4 norm_default
5 norm_axis0 = np.linalg.norm(tensor_3d,
6 axis=0)
7 norm_axis01 = np.linalg.norm(tensor_3d,
8 axis=(0, 1))
9 norm_axis01

```

结果

```

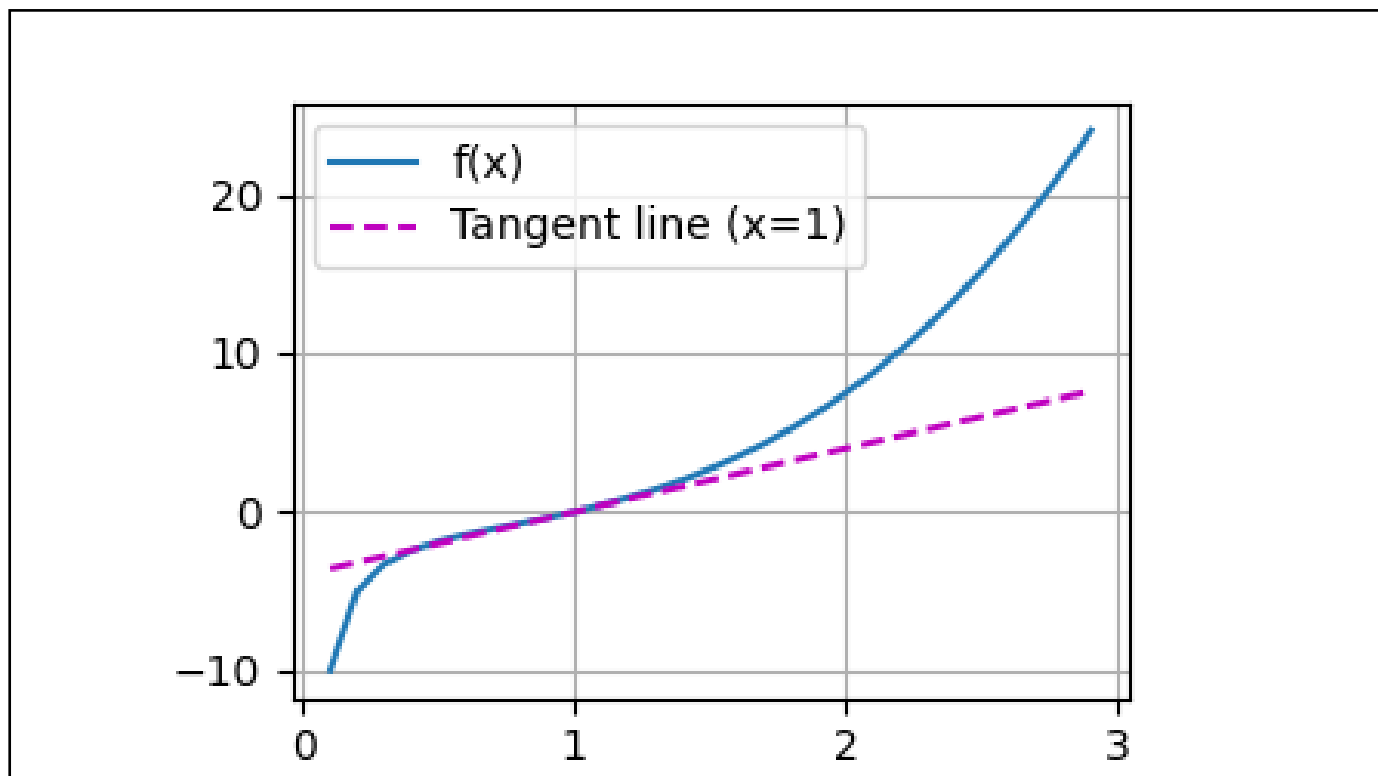
1 7.6230054
2 [[1.5946476 1.4472774 2.368117
3   2.1636908 2.045201 ],[1.8514549
4   1.5507743 2.2650828 0.7746272
5   1.3306752 ],[1.4223679 0.55377847
6   2.2253275 1.3022163 1.2005248 ],
7   [1.6133649 2.9035609 1.4079779
8   1.3459872 0.64774853]]
9 [4.924984 , 2.7686076, 3.7564898,
10  2.058105 , 2.5080800]

```

2.4 微积分

2.4.1 绘制函数 $y = f(x) = x^3 - \frac{1}{x}$ 和其在 $x = 1$ 处切线的图像

图像



2.4.2 求函数 $f(x) = 3x_1^2 + 5e^{x_2}$ 的梯度

代码

```
1 x1, x2 = sp.symbols('x1 x2')
2 f = 3 * x1**2 + 5 * sp.exp(x2)
3 gradient = [sp.diff(f, var) for var in
4 (x1, x2)]
4 print(gradient)
```

结果

```
1 [6*x1, 5*exp(x2)]
```

2.4.3 函数 $f(x) = \|x\|_2$ 的梯度是什么

代码

```
1 n = 3
2 x = sp.Matrix([sp.symbols('x{i}',
3 real=True) for i in range(1, n+1)])
4 f = sp.sqrt(sum(x[i]**2 for i in
5 range(n))) # L2 范数
6 gradient = [f.diff(x[i]) for i in
7 range(n)]
8 print(gradient)
```

结果

```
[x1/sqrt(x1**2 + x2**2 + x3**2), x2/
1 sqrt(x1**2 + x2**2 + x3**2), x3/
sqrt(x1**2 + x2**2 + x3**2)]
```

2.4.4 尝试写出函数 $u = f(x, y, z)$, 其中 $x = x(a, b)$, $y = y(a, b)$, $z = z(a, b)$ 的链式法则

代码

结果

```

1 a, b = sp.symbols('a b')
  x, y, z = sp.symbols('x y z',
2 cls=sp.Function) # x(a,b), y(a,b),
  z(a,b)
3 f = sp.Function('f')(x(a, b), y(a, b),
  z(a, b)) # u = f(x(a,b), y(a,b), z(a,b))
4 # 计算 u 对 a 的偏导数
5 du_da = sp.diff(f, a)
6 print("∂u/∂a =", du_da)
7 # 计算 u 对 b 的偏导数
8 du_db = sp.diff(f, b)
9 print("∂u/∂b =", du_db)

```

```

∂u/∂a = Derivative(f(x, y, z),
x)*Derivative(x, a) + Derivative(f(x,
1 y, z), y)*Derivative(y, a) +
  Derivative(f(x, y, z),
z)*Derivative(z, a)
∂u/∂b = Derivative(f(x(a, b), y(a, b),
z(a, b)), x(a, b))*Derivative(x(a, b), b)
+ Derivative(f(x(a, b), y(a, b), z(a,
2 b)), y(a, b))*Derivative(y(a, b), b) +
  Derivative(f(x(a, b), y(a, b), z(a, b)),
z(a, b))*Derivative(z(a, b), b)

```

2.5 自动微分

2.5.1 为什么计算二阶导数比一阶导数的开销要更大

二阶导数的计算需要在一阶导数的计算图上再叠加一层微分操作,导致计算图更复杂

2.5.2 在运行反向传播函数之后,立即再次运行它,会发生什么

在 PyTorch 中,默认情况下计算图(computational graph)在调用.backward()后会被自动释放以节省内存.当尝试第二次调用.backward()时,计算图已经不存在了,因此会报错.可以通过设置 retain_graph=True 来保留计算图

```
y.backward(retain_graph=True)
```

```
y.backward()
```

如果多次调用.backward(),梯度会累加到 x.grad 中(而不是覆盖).如果需要重新计算梯度,需要在每次反向传播前手动清零梯度:

```
x.grad.zero_()
```

```
y.backward()
```

2.5.3 在控制流的例子中,计算 d 关于 a 的导数,如果将变量 a 更改为随机向量或矩阵,会发生什么

代码

结果

```

x =
1 torch.arange(40.,requires_grad=True).
  reshape(5,-1)
2 x.retain_grad() # 保留梯度
3 y = 2 * torch.sum(x**2)
4 y.backward()
5 print(x.grad)

```

```

1 tensor([
2      [ 0.,  4.,  8., 12., 16.,
3        20., 24., 28.],
4      [ 32., 36., 40., 44., 48.,
5        52., 56., 60.],
6      [ 64., 68., 72., 76., 80.,
7        84., 88., 92.],
8      [ 96., 100., 104., 108., 112.,
9        116., 120., 124.],
10     [128., 132., 136., 140., 144.,
11      148., 152., 156.]
12 ])

```

2.5.4 重新设计一个求控制流梯度的例子,运行并分析结果

代码

```

1 def f(x):
2     y = x ** 2
3     if y.norm() > 0:
4         y = y ** 2
5         z = y
6     else:
7         z = 100 * y
8     return z
9 x = torch.randn(size=(),
10 requires_grad=True)
11 a = f(x)
12 a.backward()
13 print(x.grad)

```

结果

```

1 tensor(-24.2866)

```

2.5.5 使 $f(x) = \sin x$, 绘制 $f(x)$ 和 $\frac{df(x)}{dx}$ 的图像, 其中后者不使用 $f'(x) = \cos x$

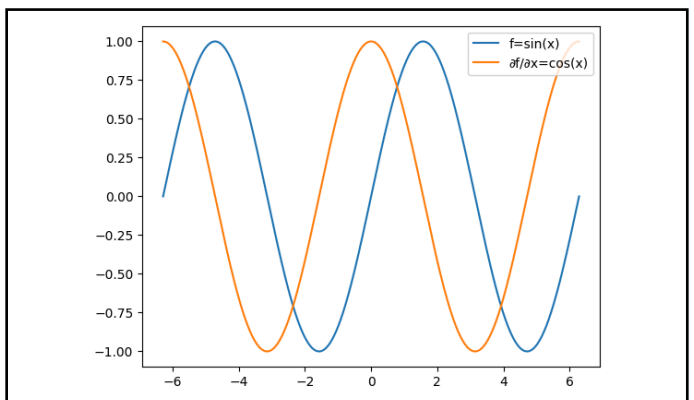
代码

```

1 x = torch.linspace(-2 * np.pi,
2 2*np.pi, 1000)
3 x.requires_grad_(True)
4 f = torch.sin(x)
5 f.sum().backward()
6 plt.plot(x.detach(), f.detach(),
7 label='f=sin(x)')
8 plt.plot(x.detach(), x.grad, label='∂f/
9 ∂x=cos(x)')
10 plt.legend(loc='upper right')
11 plt.show()

```

结果



2.6 概率

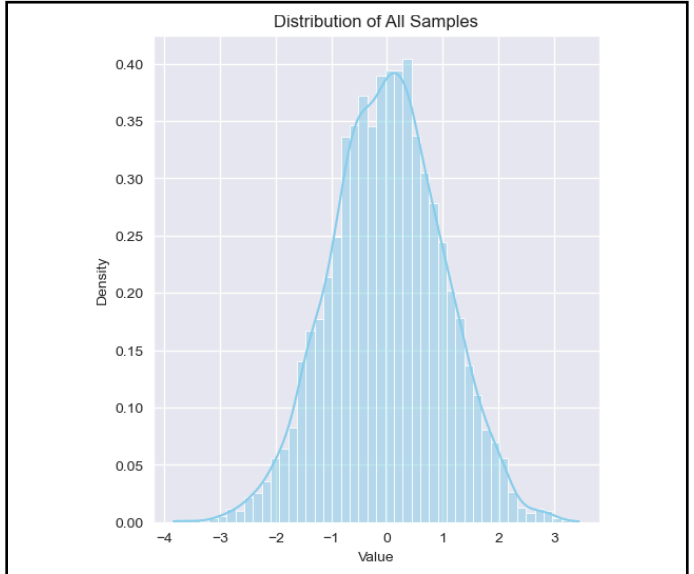
2.6.1 进行 $m = 500$ 组实验, 每组抽取 $n = 10$ 个样本. 改变 m 和 n , 观察和分析实验结果

随着 m 和 n 的增大, 总样本的分布越来越接近标准高斯分布

代码

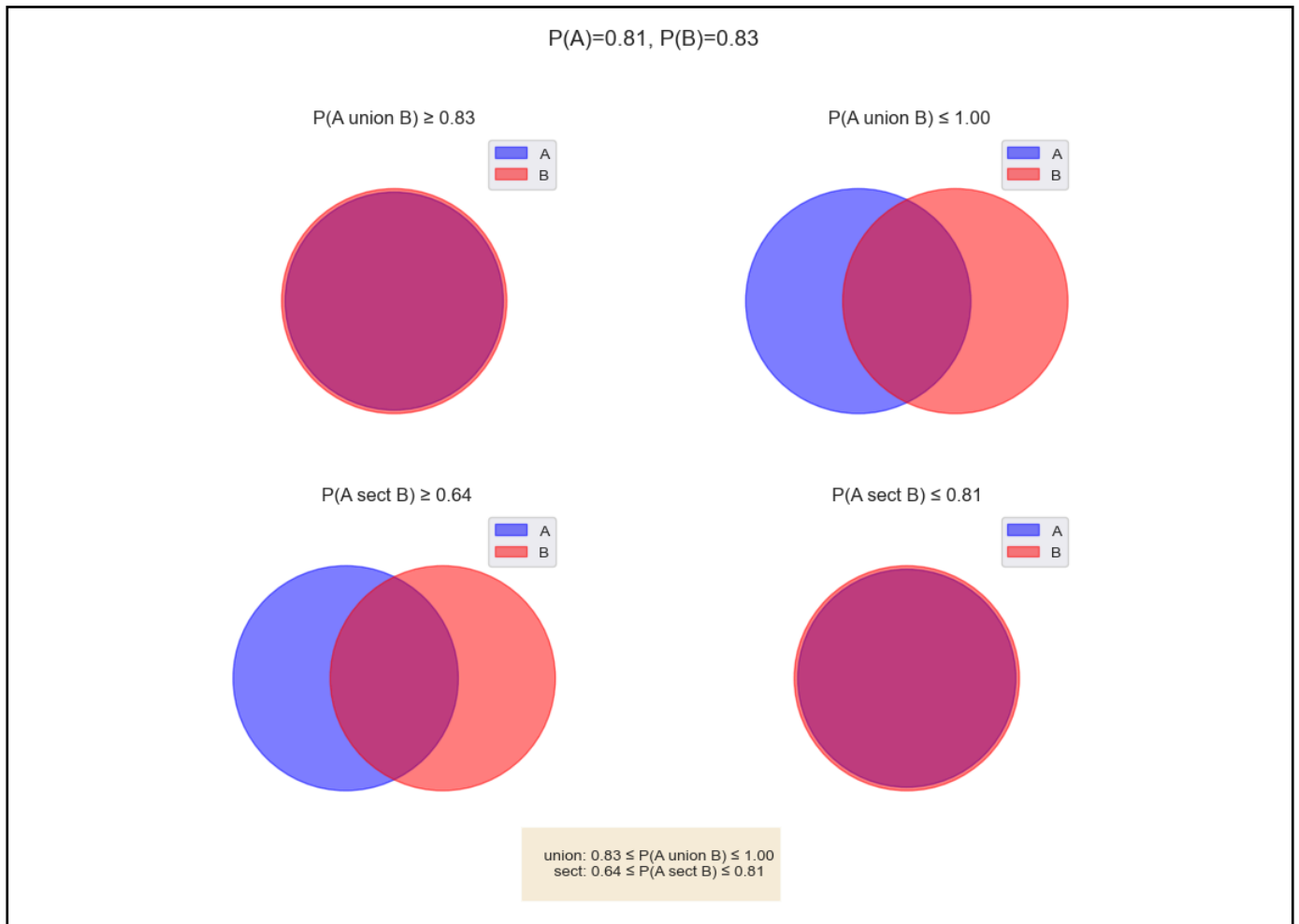
```
1 torch.manual_seed(42)
2 n = 10
3 m = 500
4 samples = torch.randn(m, n)
5 samples_np = samples.numpy()
6 plt.figure(figsize=(12, 6))
7 plt.subplot(1, 2, 1)
8 sns.histplot(samples_np.flatten(),
9 kde=True, stat="density",
10 color='skyblue')
11 plt.title('Distribution of All Samples')
12 plt.xlabel('Value')
13 plt.ylabel('Density')
```

结果



2.6.2 给定两个概率为 $P(A)$ 和 $P(B)$ 的事件, 计算 $P(A \cup B)$ 和 $P(A \cap B)$ 的上限和下限(使用韦恩图来展示这些情况)

图像



2.6.3 假设有一系列随机变量,例如 A, B 和 C ,其中 B 只依赖于 A ,而 C 只依赖于 B ,能简化联合概率吗(这是一个马尔可夫链)

$$P(A, B, C) = P(A)P(B|A)P(C|B)$$

2.6.4 艾滋病测试问题

- 假设一个医生对患者进行艾滋病病毒测试.这个测试是相当准确的,如果患者健康但测试显示他患病,这个概率只有 1%;如果患者真正感染 HIV,它永远不会检测不出.用 D_1 来表示诊断结果(如果阳性,则为 1,如果阴性,则为 0), H 来表示感染艾滋病病毒的状态(如果阳性,则为 1,如果阴性,则为 0).人口总体是相当健康的, $P(H=1)=0.0015$,现在测试显示他患病,那么患者真正患病的概率是多少

$$\text{先验概率: } P(H = 1) = 0.0015, P(H = 0) = 0.9985$$

$$\text{条件概率: } P(D_1 = 1|H = 1) = 1, P(D_1 = 1|H = 0) = 0.01$$

$$\text{全概率: } P(D_1 = 1) = P(D_1 = 1|H = 1)P(H = 1) + P(D_1 = 1|H = 0)P(H = 0) = 0.011485$$

$$\text{后验概率: } P(H = 1|D_1 = 1) = \frac{P(D_1=1|H=1)P(H=1)}{P(D_1=1)} \approx 13.05\%$$

- 患者得知测试阳性后要求医生进行另一次测试来确定病情.第二个测试具有不同的特性,它不如第一个测试那么精确,如果患者健康但测试显示他患病,这个概率有 3%;如果患者真正感染 HIV 但测试显示他没病,这个概率有 2%.经过第二次测试,依然显示患病,请问现在患者患病的概率是多少

$$\text{先验概率: } P(H = 1) = 0.1305, P(H = 0) = 0.8695$$

$$\text{条件概率: } P(D_2 = 1|H = 1) = 0.98, P(D_2 = 1|H = 0) = 0.03$$

$$\text{全概率: } P(D_2 = 1) = P(D_2 = 1|H = 1)P(H = 1) + P(D_2 = 1|H = 0)P(H = 0) \approx 0.153975$$

$$\text{后验概率: } P(H = 1|D_2 = 1) = \frac{P(D_2=1|H=1)P(H=1)}{P(D_2=1)} \approx 83.06\%$$

- 虽然第一个测试在准确性方面表现更好,但在临床实践中,有时会采用两个不同的测试而不是重复使用同一个测试
 - 避免系统性错误
 - 提高独立性和确认性
 - 分层更新贝叶斯概率

第三章 线性回归网络

3.1 线性回归网络

3.1.1 假设有一些数据 $x_1, x_2, \dots, x_n \in \mathbb{R}$, 目标是找到一个常数 b , 使得 $\sum_i (x_i - b)^2$ 最小化

- 如何找到最优值的解析解
- 这个问题及其解与正态分布有什么关系

常数模型 $y_i = b + \varepsilon_i$

设计矩阵

$$\mathbf{X} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^{n \times 1}$$

线性模型

$$\mathbf{y} = \mathbf{X}b + \varepsilon$$

解析解

$$b = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} = \frac{1}{n} \sum_i x_i$$

假设数据 $x_1, x_2, \dots, x_n \in \mathbb{R}$ 从正态分布 $\mathcal{N}(\mu, \sigma^2)$ 中独立采样得到, 其概率密度与似然函数

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$
$$L(\mu) = \prod_i f(x_i) \propto \exp\left(-\frac{1}{2\sigma^2} \sum_i (x_i - \mu)^2\right)$$

最大化 $\log L(\mu)$ 等价于最小化 $\sum_i (x_i - b)^2$, 正态分布的最大似然估计 $\hat{\mu}$ 就是样本均值

3.1.2 推导出使用平方误差的线性回归优化问题的解析解, 忽略偏置 b 简化问题(可以通过向 \mathbf{x} 添加所有值为 1 的一列来做到这一点)

- 用矩阵和向量表示法写出优化问题(将所有数据视为单个矩阵, 将所有目标值视为单个向量)
- 计算损失对 w 的梯度
- 将梯度设为 0, 求解矩阵方程来找到解析解
- 什么时候可能比使用随机梯度下降更好? 这种方法何时会失效

给定数据矩阵 $\mathbf{X} \in \mathbb{R}^{n \times d}$ 和目标向量 $\mathbf{y} \in \mathbb{R}^n$ 线性回归的优化目标是最小化平方误差

$$\min L(w) = \|\mathbf{X}w - \mathbf{y}\|^2 = (\mathbf{X}w - \mathbf{y})^T (\mathbf{X}w - \mathbf{y})$$
$$L(w) = w^T \mathbf{X}^T \mathbf{X} w - 2\mathbf{y}^T \mathbf{X} w + \mathbf{y}^T \mathbf{y}$$
$$\nabla_w L = 2\mathbf{X}^T \mathbf{X} w - 2\mathbf{X}^T \mathbf{y}$$
$$w = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

解析解的优势

- 精确解: 直接得到最优参数, 无需调参(如学习率)
 - 小规模数据高效: 当特征数 d 较小, 计算 $(\mathbf{X}^T \mathbf{X})^{-1}$ 的时间复杂度 $O(d^3)$ 可接受
- 失效情况
- 矩阵不可逆: 当 \mathbf{X} 列线性相关或 $n < d$, $\mathbf{X}^T \mathbf{X}$ 不可逆
 - 高维数据: 特征数 d 很大, 计算逆矩阵成本过高
 - 内存限制: 数据量 n 极大时, 存储 \mathbf{X} 并计算 $\mathbf{X}^T \mathbf{X}$ 不可行

3.1.3 假定控制附加噪声的 ϵ 噪声模型是指数分布, $p(\epsilon) = \frac{1}{2} \exp(-|\epsilon|)$

- 写出模型 $-\log P(\mathbf{y}|\mathbf{X})$ 下数据的负对数似然
- 写出解析解

- 提出一种随机梯度下降算法来解决这个问题

$$\mathbf{y} = \mathbf{X}\beta + \epsilon_i$$

$$P(\mathbf{y}|\mathbf{X}) = \prod_i \frac{1}{2} \exp(-|\mathbf{y} - \mathbf{x}_i\beta|)$$

$$-\log P(\mathbf{y}|\mathbf{X}) = n \log 2 + \sum_i \exp(-|\mathbf{y} - \mathbf{x}_i\beta|)$$

带指数分布噪声的线性回归模型等价于最小化 L1 损失函数,也称为中位数回归(Median Regression).与普通最小二乘法(OLS)不同,由于 L1 损失函数在残差为零处不可导,L1 回归通常没有封闭形式的解析解.对于一维情况,中位数回归的解析解就是将回归线穿过数据点的中位数.对于多维情况,通常需要使用数值优化方法

1. 线性规划
2. 迭代加权最小二乘法
3. 梯度下降类算法(需要处理不可导点)

由于 L1 损失在残差为零处不可导,需要使用次梯度(subgradient)方法

3.2 从零开始实现

3.2.1 如果将权重 w 初始化为零,会发生什么?算法仍然有效吗

线性回归在权重初始化为零时仍然有效,主要是因为

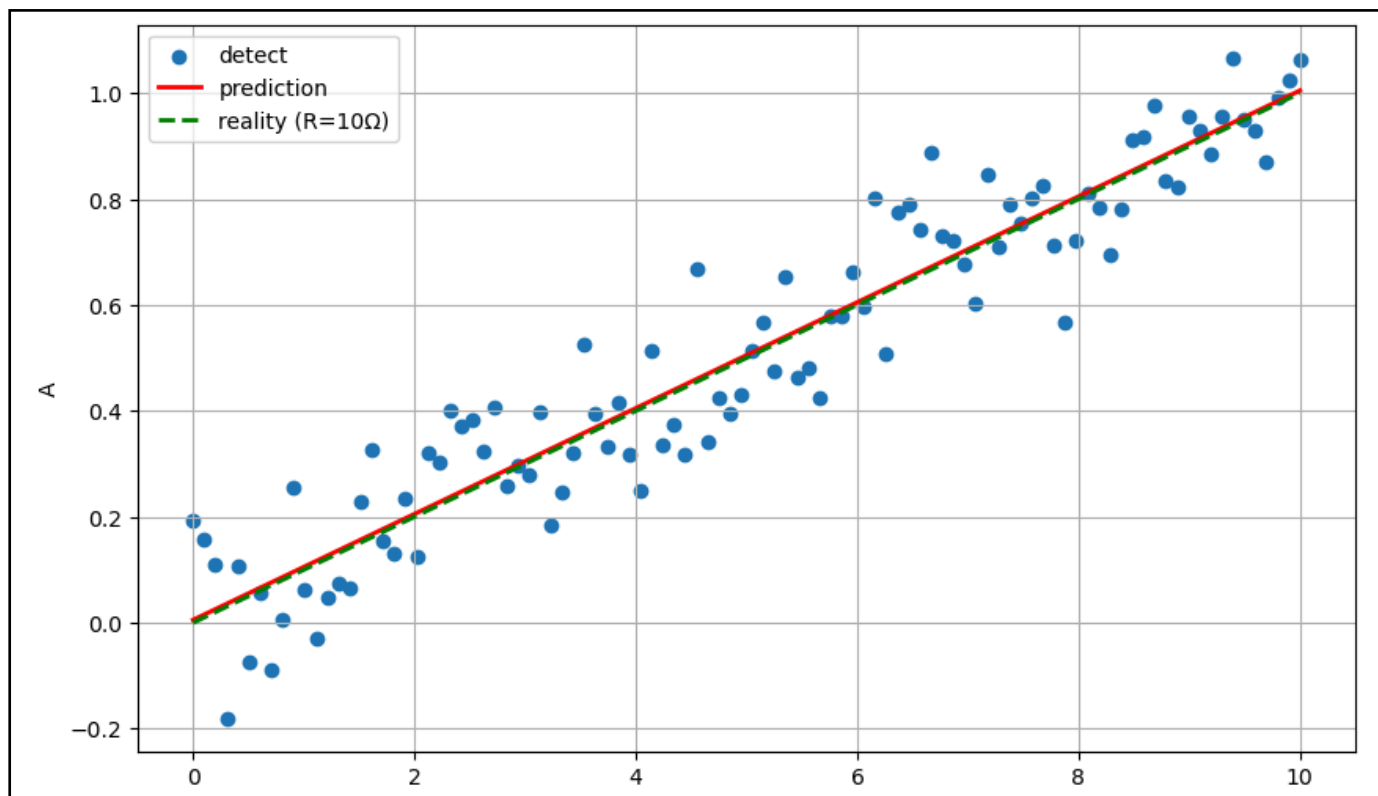
1. 梯度取决于输入特征:即使权重相同,但由于输入特征 x_i 各不相同,因此计算得到的梯度也各不相同
2. 无隐藏层结构:线性回归是直接从输入到输出的映射,没有中间层的复杂结构,因此不会遇到神经网络中的表达能力受限问题.
3. 单一全局最优解:线性回归的损失函数是凸函数,总是有一个全局最优解,梯度下降算法最终会收敛到这个解.

3.2.2 为电压和电流的关系建立一个模型,自动微分可以用来学习模型的参数吗

自动微分在以下几个关键步骤中起到了重要作用

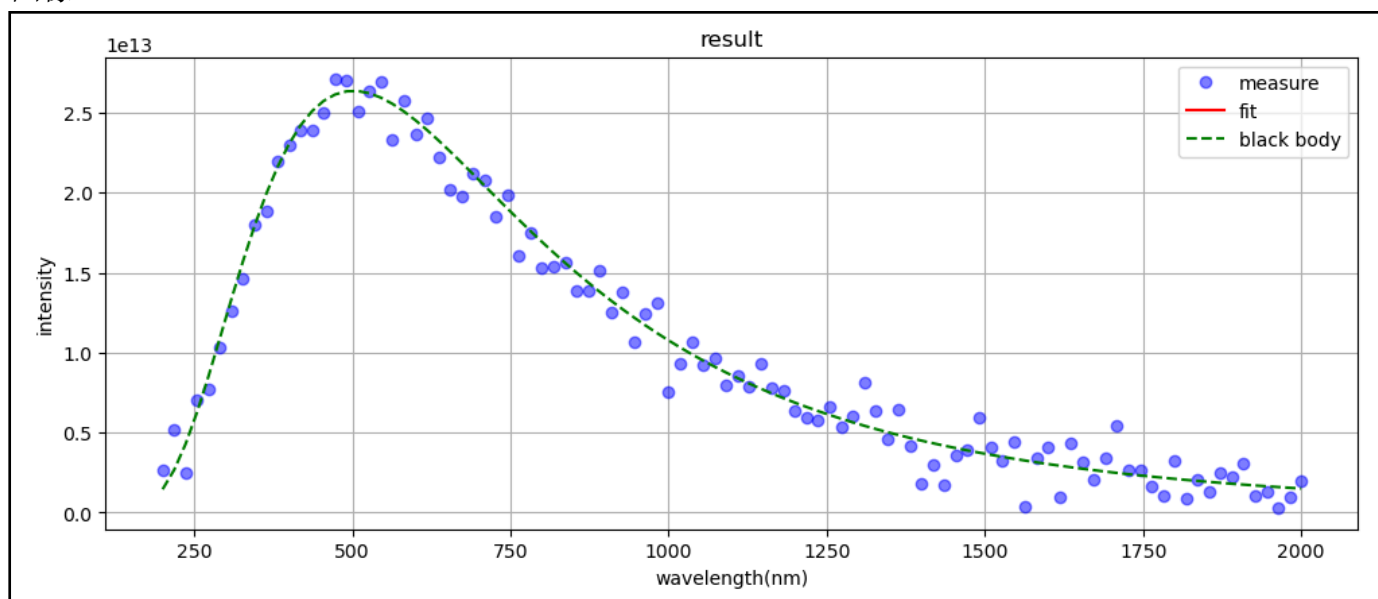
1. 参数跟踪:创建 `nn.Linear` 层时,PyTorch 自动为权重和偏置参数设置 `requires_grad=True`,参数的梯度将被计算和存储
2. 损失函数计算:当计算模型输出与实际电流之间的损失时,PyTorch 构建了一个计算图,跟踪损失是如何依赖于模型参数的
3. 梯度计算:当调用 `loss.backward()` 时,PyTorch 自动计算损失函数相对于每个参数的梯度.这些梯度展示如何调整参数以减小损失
4. 参数更新:`optimizer.step()` 使用计算出的梯度来更新模型参数,通常沿着梯度的负方向移动(梯度下降)

图像



3.2.3 基于普朗克定律使用光谱能量密度来确定物体的温度

图像



3.2.4 计算二阶导数时可能会遇到什么问题?如何解决

主要问题

1. 梯度消失问题:当计算深层网络的二阶导数时,由于链式法则的连乘效应,可能导致梯度变得非常小
2. 内存占用过大:二阶导数计算需要储存更多的中间计算结果,特别是对于大型模型,可能导致 GPU 内存不足
3. 计算效率低下:直接计算 Hessian 矩阵计算量很大,尤其是参数众多的模型
4. 数值不稳定性:浮点数精度限制可能导致数值计算不稳定

解决方案

1. 使用 Hessian 向量积(Hessian-vector product, HVP)
2. 使用自动微分库
3. 有限差分法近似
4. 低阶优化算法:如果只是为了优化而计算二阶导数,可以考虑使用近似二阶方法如 L-BFGS、KFAC 等
5. 分批计算 Hessian 矩阵

3.2.5 为什么在 squared_loss 函数中需要使用 reshape 函数

1. 维度匹配

在计算均方误差(MSE)时,预测值和真实值的形状必须一致或能够进行广播.reshape 确保这两个张量可以直接进行数学运算

2. 应对不同的输出格式

线性回归模型的输出形状可能是:

- [batch_size]:一维张量
- [batch_size, 1]:二维张量列向量

而标签数据可能是不同形状:

- 可能是普通 Python 列表:[value1, value2, ...]
- 可能是 NumPy 数组或 Pandas Series
- 可能是不同形状的 PyTorch 张量

reshape 函数确保无论输入格式如何,都能统一处理

3. 处理模型输出

许多 PyTorch 模型的输出是形状为[batch_size, num_features]的张量,即使 num_features=1

4. 防止隐式广播导致的错误

如果不使用 reshape,PyTorch 的广播机制可能会导致意外的计算结果

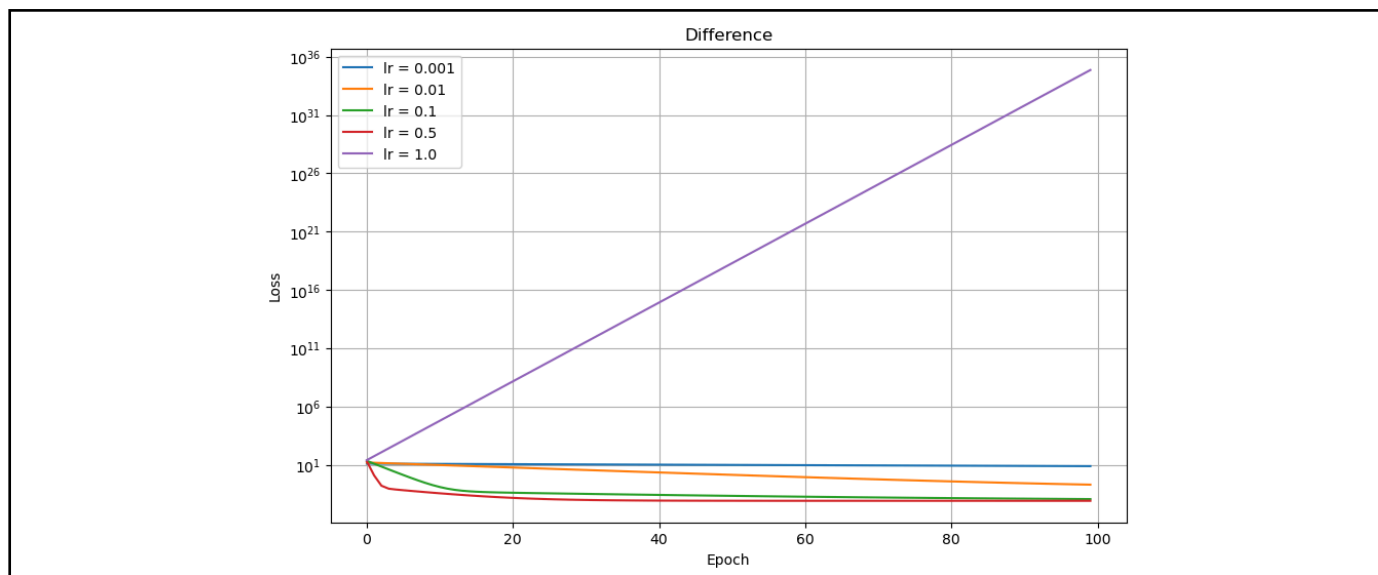
5. 批量处理的一致性

在批量处理数据时,确保每个样本的损失计算方式一致是很重要的

3.2.6 尝试使用不同的学习率,观察损失函数值下降的快慢

- 从小学习率开始(0.001)
- 逐渐增加(乘以 10)
- 当观察到损失函数开始震荡或发散时停止
- 选择震荡前的最大学习率作为最优值

图像



3.2.7 如果样本个数不能被批量大小整除,data_iter 函数的行为会有什么变化

当样本总数不能被批量大小整除时,会出现以下几种情况:

1. 最后一个批次的大小不足

这是最常见的处理方式.如果总样本数为 n ,批量大小为 $batch_size$,当 $n \% batch_size \neq 0$ 时

- 前面的批次都是完整的 $batch_size$ 大小
- 最后一个批次的大小为 $n \% batch_size$,小于标准批量大小

例如:有 103 个样本,批量大小为 32,则会产生 4 个批次.前 3 个批次各包含 32 个样本,最后 1 个批次只包含 7 个样本($103-3 \times 32=7$)

2. 对最后一个不完整批次的不同处理策略

在实际应用中,对于最后一个不完整批次,有几种常见的处理策略:

- 保留不完整批次(默认行为)

如上所述,就是保留最后一个较小的批次.这是最直接的方法,确保所有样本都被使用

- 丢弃不完整批次

某些情况下,特别是当批次归一化(Batch Normalization)层对批次大小敏感时,可能会选择丢弃最后一个不完整的批次

- 填充(padding)不完整批次

在 NLP 或计算机视觉任务中,常用填充方法使最后一个批次达到完整大小

3.3 简洁实现

3.3.1 如果将小批量的总损失替换为小批量损失的平均值,需要如何更改学习率

总损失

$$L_{\text{sum}} = \sum_{i=1}^n l(x_i; w)$$

对参数 w 的梯度为

$$\nabla_w L_{\text{sum}} = \sum_{i=1}^n \nabla_w l(x_i; w)$$

一个随机梯度下降步长(Stochastic Gradient Descent)

$$\Delta w = -\eta \nabla_w L_{\text{sum}}$$

平均损失

$$L_{\text{mean}} = \frac{1}{n} \sum_{i=1}^n l(x_i; w)$$

$$\nabla_w L_{\text{mean}} = \frac{1}{n} \sum_{i=1}^n \nabla_w l(x_i; w)$$

$$\Delta w = -\frac{\eta}{n} \nabla_w L_{\text{mean}}$$

为了让平均损失的每步更新幅度与总损失版本一致,需要将学习率放大 n 倍

3.3.2 查看深度学习框架文档,它们提供了哪些损失函数和初始化方法

PyTorch 中的损失函数和初始化方法

- 回归任务
 - nn.MSELoss 均方误差 Mean Squared Error Loss
 - nn.L1Loss 平均绝对误差 Mean Absolute Error
 - nn.HuberLoss Huber 损失 Huber Loss
 - nn.SmoothL1Loss 平滑 L1 损失 Smooth L1 Loss
- 分类任务
 - BinaryCrossentropy() 二元交叉熵
 - CategoricalCrossentropy() 多类交叉熵
 - SparseCategoricalCrossentropy() 稀疏多类交叉熵
- 其他任务
 - KLDivergence() KL 散度
 - CosineSimilarity() 余弦相似度

权重初始化方法

- xavier_uniform_ / xavier_normal_ 适用于 Sigmoid/Tanh 激活函数
- kaiming_uniform_ / kaiming_normal_ 适用于 ReLU 激活函数
- orthogonal_ 正交初始化
- constant_ 常数初始化
- zeros_ / ones_ 全零或全一初始化

3.3.3 访问线性回归中的梯度

代码

```

1  x = torch.tensor([[1.0], [2.0],
2  [3.0]])
3  y = torch.tensor([[2.0], [4.0], [6.0]])
4  model = nn.Linear(in_features=1,
5  out_features=1)
6  criterion = nn.MSELoss()
7  optimizer =
8  torch.optim.SGD(model.parameters(),
9  lr=0.01)
10 pred = model(x)
11 loss = criterion(pred, y)
12 optimizer.zero_grad() # 清除旧的梯度
13 loss.backward() # 自动计算梯度
14 print("Weight gradient:",
15       model.weight.grad)
16 print("Bias gradient:", model.bias.grad)
```

结果

```

1 Weight gradient: tensor([[ -10.5606]])
2 Bias gradient: tensor([-4.3515])
```

