ExerciseThinking

卷三 动手学深度学习 Pytorch 习题解

作者：latalealice

日期：2025/04/24

目 录

序言

此 PDF 为习题解答,**需求的库如下**

```py
from d2l import torch as d2l
from matplotlib_inline import backend_inline
from matplotlib_venn import venn2
from torch.distributions import multinomial
import numpy as np
import matplotlib.pyplot as plt
import os
import pandas as pd
import seaborn as sns
import torch
```

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

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

结果

```py
tensor([[False, False, False, False],
        [ True,  True,  True,  True],
        [ True,  True,  True,  True]]),
tensor([[False,  True, False,  True],
        [False, False, False, False],
        [False, False, False, False]]),
tensor([[ True, False,  True, False],
        [False, False, False, False],
        [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

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

结果

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

```py
os.makedirs(os.path.join('..', 'data'), exist_ok=True)
data_file = os.path.join('..', 'data', 'house_tiny.csv')
with open(data_file, 'w') as f:
    f.write('NumRooms,Alley,Price\n')  # 列名
    f.write('NA,Pave,127500\n')  # 每行表示一个数据样本
    f.write('2,NA,106000\n')
    f.write('4,NA,178100\n')
    f.write('NA,NA,140000\n')
    f.write('3,Pave,122000\n')
# 1. 删除缺失值最多的列
# 计算每列的缺失值数量
missing_values = data.isnull().sum()
# 找出缺失值最多的列名
column_to_drop = missing_values.idxmax()
# 删除该列
data_preprocessed = data.drop(column_to_drop, axis=1)
print(data_preprocessed)
# 2. 将预处理后的数据集转换为张量格式
# 首先处理剩余的缺失值(用均值填充数值列)
data_preprocessed['NumRooms'] = data_preprocessed['NumRooms'].fillna(data_preprocessed['NumRooms'].mean
# 转换为张量
tensor_data = torch.tensor(data_preprocessed.values, dtype=torch.float32)
print(tensor_data)
```

结果

```py
   NumRooms   Price
0       NaN  127500
1       2.0  106000
2       4.0  178100
3       NaN  140000
4       3.0  122000
tensor([[3.0000e+00, 1.2750e+05],
        [2.0000e+00, 1.0600e+05],
        [4.0000e+00, 1.7810e+05],
        [3.0000e+00, 1.4000e+05],
        [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

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

```py
tensor([[True, True, True, True],
        [True, True, True, True],
        [True, True, True, True],
        [True, True, True, True],
        [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

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

结果

```py
tensor([[True, True, True],
        [True, True, True],
        [True, True, True],
        [True, True, True]])
```

2.3.3 给定任意方阵 A, $A+A^T$ 总是对称的吗

代码

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

结果

```py
tensor([[True, True, True, True],
        [True, True, True, True],
        [True, True, True, True],
        [True, True, True, True]])
```

2.3.4 定义形状(2,3,4)的张量 X,len(X)的输出结果是什么

代码

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

结果

```py
2
```

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

代码

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

结果

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

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

代码

结果

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

```py
tensor([
  [0.0000, 0.0286, 0.0333, 0.0353, 0.0364],
  [0.5000, 0.1714, 0.1167, 0.0941, 0.0818],
  [1.0000, 0.3143, 0.2000, 0.1529, 0.1273],
  [1.5000, 0.4571, 0.2833, 0.2118, 0.1727],
  [2.0000, 0.6000, 0.3667, 0.2706, 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

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

结果

```py
tensor([[12, 14, 16, 18],
        [20, 22, 24, 26],
        [28, 30, 32, 34]])
tensor([[12, 15, 18, 21],
        [48, 51, 54, 57]])
tensor([[ 6, 22, 38],
        [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

```py
# 创建一个3×4×5的张量
tensor_3d = torch.randn(3, 4, 5)
norm_default = np.linalg.norm(tensor_3d)
norm_default
norm_axis0 = np.linalg.norm(tensor_3d, axis=0)
norm_axis0
norm_axis01 = np.linalg.norm(tensor_3d, axis=(0, 1))
norm_axis01
```

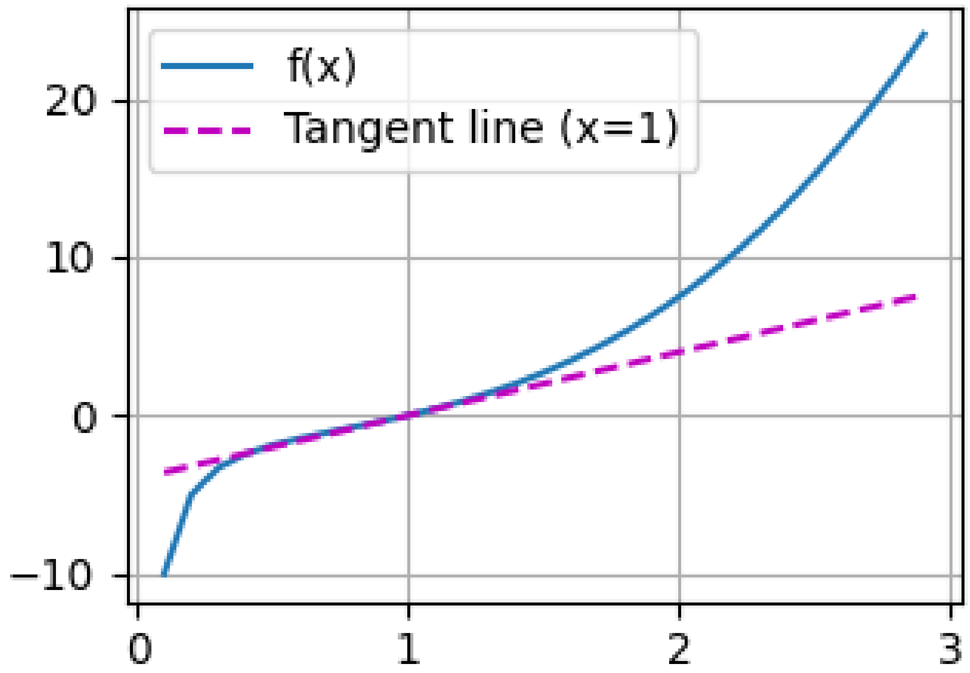
结果

```py
7.6230054
[[1.5946476  1.4472774  2.368117   2.1636908  2.045201 ],[1.8514549  1.5507743  2.2650828  0.7746272  1.3306752 ],[1.4223679  0.55377847  2.2253275  1.3022163  1.2005248 ],[1.6133649  2.9035609  1.4079779  1.3459872  0.64774853]]
[4.924984 , 2.7686076, 3.7564898, 3.958195   2.5980809]
```

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 $f(\mathrm{x}) = 3x_1^2 + 5e^{x_2}$ 的梯度

代码

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

结果

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

2.4.3 函数 $f(\mathrm{x}) = \|\mathrm{x}\|_2$ 的梯度是什么

代码

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

结果

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

```py
a, b = sp.symbols('a b')
x, y, z = sp.symbols('x y z', cls=sp.Function)  # x(a,b), y(a,b), z(a,b)
f = sp.Function('f')(x(a, b), y(a, b), z(a, b))  # u = f(x(a,b), y(a,b), z(a,b))
# 计算 u 对 a 的偏导数
du_da = sp.diff(f, a)
print("∂u/∂a =", du_da)
# 计算 u 对 b 的偏导数
du_db = sp.diff(f, b)
print("∂u/∂b =", du_db)
```

```py
∂u/∂a = Derivative(f(x, y, z), x)*Derivative(x, a) + Derivative(f(x, 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, 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

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

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

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

```python
def f(x):
  y = x ** 2
  if y.norm() >0:
      y = y ** 2
      z = y
  else:
      z = 100 * y
  return z
x = torch.randn(size=(), requires_grad=True)
a = f(x)
a.backward()
print(x.grad)
```

结果

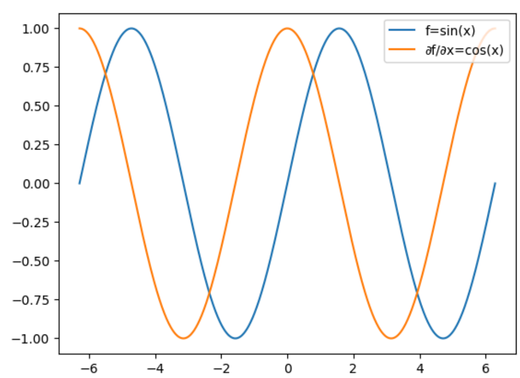
```
tensor(-24.2866)
```

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

代码

```python
x = torch.linspace(-2 * np.pi, 2*np.pi, 1000)
x.requires_grad_(True)
f= torch.sin(x)
f.sum().backward()
plt.plot(x.detach(), f.detach(), label='f=sin(x)')
plt.plot(x.detach(), x.grad, label='∂f/∂x=cos(x)')
plt.legend(loc='upper right')
plt.show()
```

结果



2.6 概率

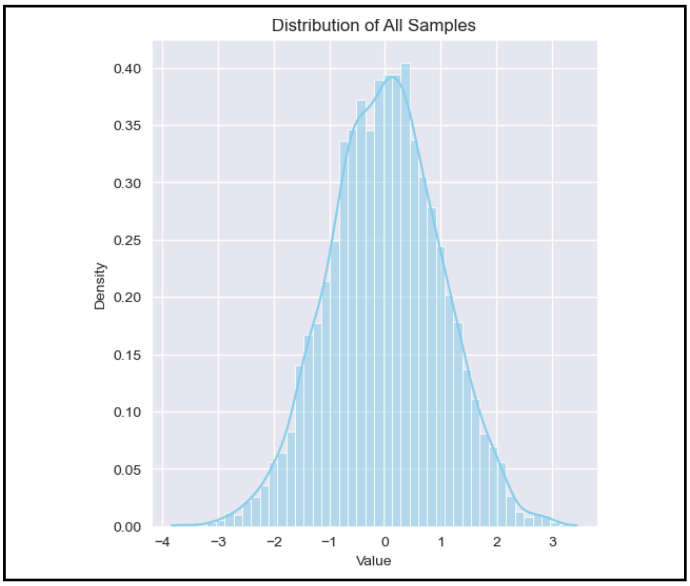
2.6.1 进行 $m = 500$ 组实验,每组抽取 $n = 10$ 个样本.改变 m 和 n,观察和分析实验结果

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

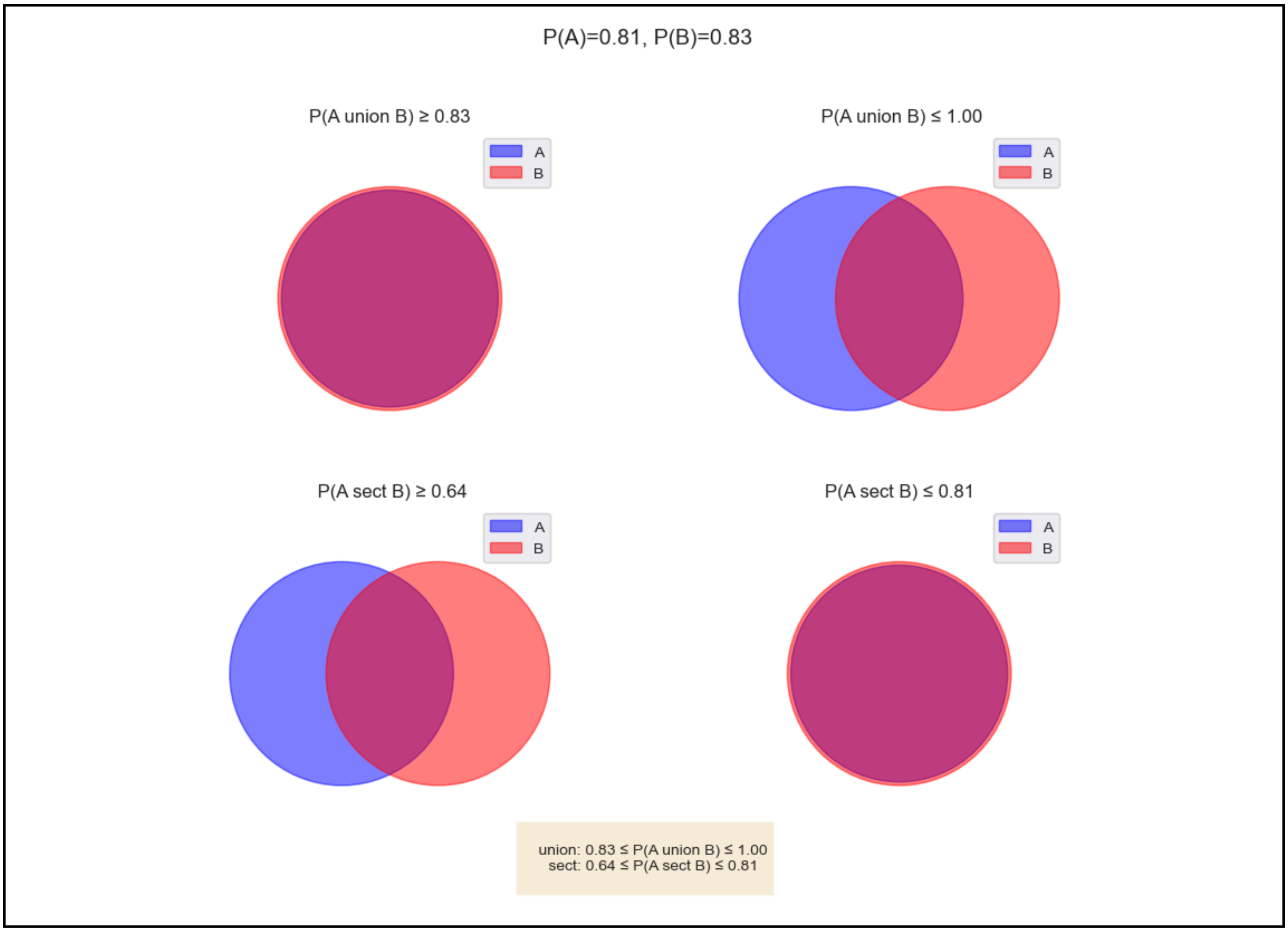
```py
torch.manual_seed(42)
n = 10
m = 500
samples = torch.randn(m, n)
samples_np = samples.numpy()
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.histplot(samples_np.flatten(),
kde=True, stat="density",
color='skyblue')
plt.title('Distribution of All Samples')
plt.xlabel('Value')
plt.ylabel('Density')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

结果



2.6.2 给定两个概率为 $P(\mathcal{A})$ 和 $P(\mathcal{B})$ 的事件,计算 $P(\mathcal{A} \cup \mathcal{B})$ 和 $P(\mathcal{A} \cap \mathcal{B})$ 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)

$$\boxed{P(A,B,C) = P(A)P(B|A)P(C|B)}$$

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

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

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

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

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

- 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

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

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

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

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

- 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 $x_1, x_2, ..., x_n \in R$,目标是找到一个常数 b,使得 $\sum_i (x_i - b)^2$ 最小化

- 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

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

假设数据 $x_1, x_2, ..., x_n \in R$ 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

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

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

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 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

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

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

$$\nabla_w L=2\mathbf{X}^T\mathbf{X}w-2\mathbf{X}^T\mathbf{y}$$

$$w=\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

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 $\left(\mathbf{X}^T\mathbf{X}\right)^{-1}$ 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

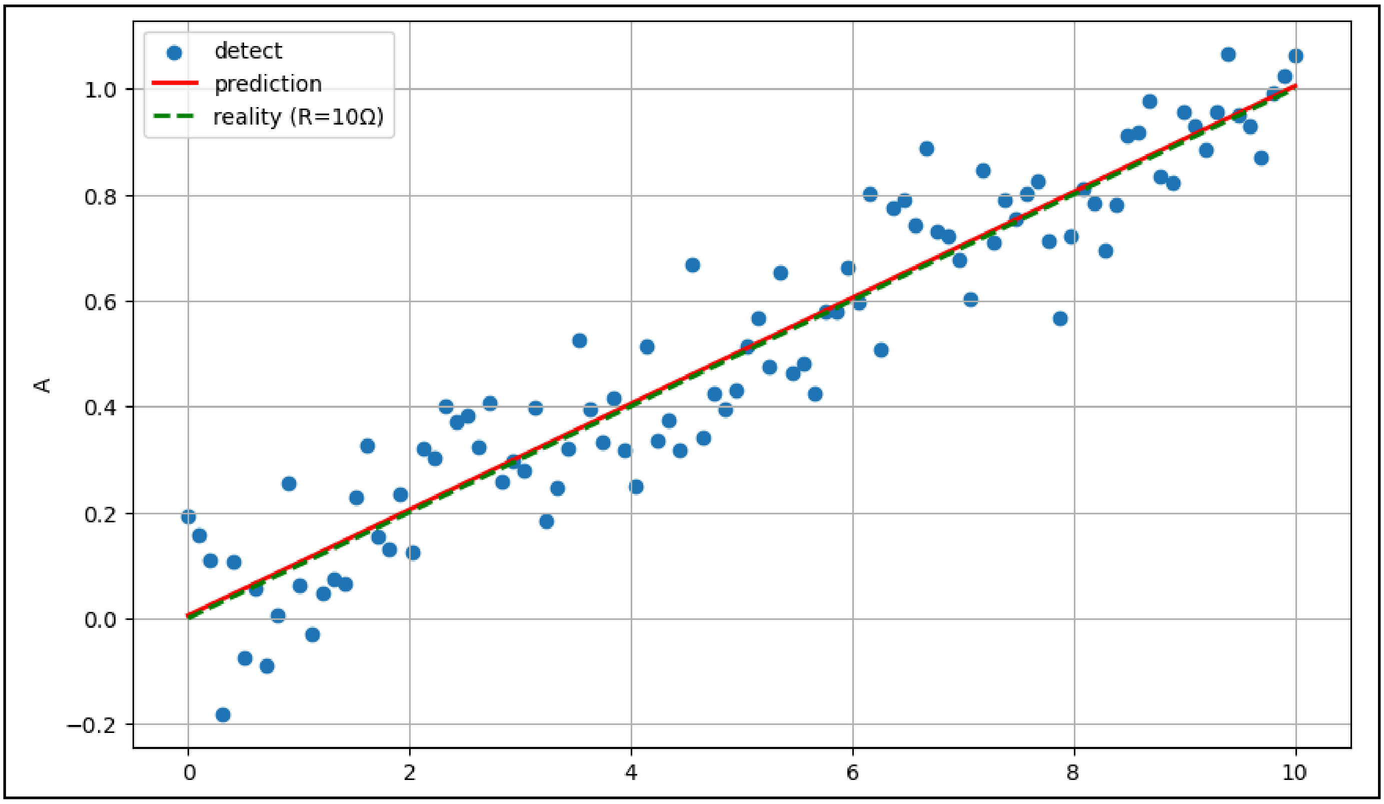
1. 梯度取决于输入特征:即使权重相同,但由于输入特征 x_i 各不相同,因此计算得到的梯度也各不相同
2. 无隐藏层结构:线性回归是直接从输入到输出的映射,没有中间层的复杂结构,因此不会遇到神经网络中的表达能力受限问题.
3. 单一全局最优解:线性回归的损失函数是凸函数,总是有一个全局最优解,梯度下降算法最终会收敛到这个解.

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