# 实验 2：利用 PyTorch 实现简单的线性回归算法

## 一、　实验内容

实验内容：使用PyTorch构建一个简单的线性回归网络。
实验环境：Python-3.11;PyTorch-cu121;CUDA-12.1;VS code/JupyterNotebook

## 二、　实验过程和结果

1. 准备数据

```
import torch
from torch.autograd import Variable
import numpy as np

x=Variable(torch.linspace(0,100,100).type(torch.FloatTensor))
rand=Variable(torch.randn(100))*10
y=x+rand
```

该段代码引入 torch 库和自动微分变量构造函数 Variable，numpy 库。

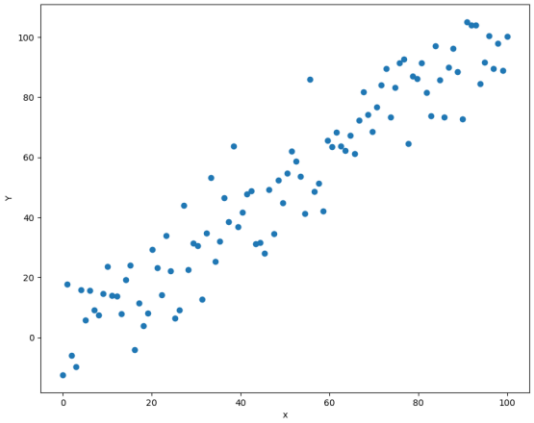
其中 x 为线性插值生成 0-100 的自然数一维张量。torch.FloatTensor 是一种基本的数据格式，用于存储浮点数值，可以用于各种数学运算。

而 rand 为使用 torch.randn 生成产生标准正态分布-1~1 的随机数，乘 10，使其范围为-10~10。将 x 与 rand 相加得到 y，为近似线性分布具有随机偏置的一组数据。

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.figure(figsize=(10,8))
plt.plot(x.data.numpy(),y.data.numpy(),'o')
plt.xlabel('x')
plt.ylabel('Y')
plt.show()
```

该段代码引入了 matplotlib 绘图库，%matplotlib inline 可以将绘图显示在代码行中。设置图像长宽，绘制‘o’形的散点图，plt 库绘制需要将 numpy 数组作为参数传入，因此使用 .data.numpy()取出张量后并转为 numpy 数组。绘图结果如右图所示，能够看出散点大致分布趋势为线性。




2. 构造模型计算损失函数

```
a=Variable(torch.rand(1),requires_grad=True)
b=Variable(torch.rand(1),requires_grad=True)
print('Initial parameters',[a,b])
```

```
Initial parameters [tensor([0.9810], requires_grad=True), tensor([0.6445], requires_grad=True)]
```

该段代码设置了线性回归模型的两个参数 a，b，采用随机生成，需要求梯度。输出初始的参数值。

```python
learning_rate=0.0001
for i in range(1000):
    if(a.grad is not None) and (b.grad is not None):
        a.grad.data.zero_()
        b.grad.data.zero_()
    predictions=a.expand_as(x)*x+b.expand_as(x)
    loss=torch.mean((predictions-y)**2)
    print('loss:',loss.data.numpy())
    loss.backward()
    a.data.add_(-learning_rate*a.grad.data)
    b.data.add_(-learning_rate*a.grad.data)
```

Python

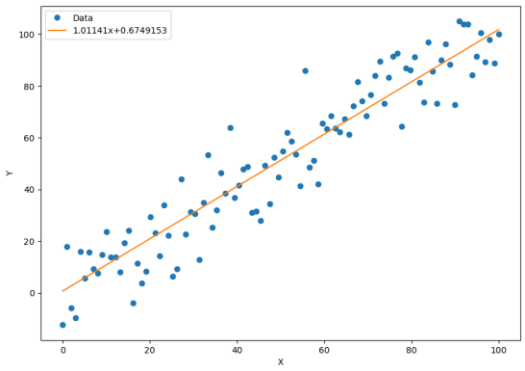
```
loss: 109.6102
loss: 106.75015
loss: 106.46035
loss: 106.43162
loss: 106.429
loss: 106.42885
loss: 106.42885
loss: 106.428856
loss: 106.42885
```

这段代码设置学习率为 0.0001。在循环中先将 a，b 的梯度数据清零，建立线性的预测模型 ax+b，由于 a 和 b 是一维的张量，先将 a，b 扩展为与 x 相同维度和大小后进行对应位置相乘相加。损失函数为均方差函数，对损失函数进行反向传播，求出对 a，b 的梯度，并根据梯度对 a，b 进行调整。该段步骤循环 1000 次，足够对 ab 进行充分训练。

```python
x_data=x.data.numpy()
plt.figure(figsize=(10,7))
xplot,=plt.plot(x_data,y.data.numpy(),'o')
yplot,=plt.plot(x_data,a.data.numpy()*x.data.numpy()+b.data.numpy())
plt.xlabel('X')
plt.ylabel('Y')
strl=str(a.data.numpy()[0])+'x+'+str(b.data.numpy()[0])
plt.legend([xplot,yplot],['Data',strl])
plt.show()
```

Python

这段代码对训练结果进行了展示。首先将 x 包裹的张量转换为 numpy 数组，绘制训练用得的数据点，再用 ax+b 生成训练得到的拟合直线上的点，绘制出经过线性拟合的直线。并设置横纵坐标标签和图例。




3. 测试模型

```
x_test=Variable(torch.FloatTensor([1,2,10,100,1000]))
predictions_test=a.expand_as(x_test)*x_test+b.expand_as(x_test)
predictions_test
```

Python

```
tensor([   1.6863,    2.6977,   10.7890,  101.8159, 1012.0849],
       grad_fn=<AddBackward0>)
```

这段代码对得到的模型进行测试，选取 x=1，2，10，100，1000 五个输入值，输出值为 1.6863,2.6977，10.7890,101.8159,1012.0849。

## 三、心得体会

本次实验采用线性回归问题练习了 pytorch 手动建模，机器学习时我们要考虑四个要素：数据、模型、目标函数、最优化算法。在编程实现时，我们首先要将数据集处理，考虑哪些特征需要被输入到模型中训练，本次实验的数据集是编程生成的，建模时我们先要考虑模型的结构，哪些参数是可变的需要对其进行反向传播进行优化，这些参数变量需要设置为 requires_grad。目标函数是指要达到什么效果，一般来说我们想要使训练结果和标准结果尽可能相近，对于连续值输出结果可以采用损失函数为均方根误差函数，使损失函数取最小值的参数值认为是该模型下的最优解。最后是最优化算法，是指采用什么样的方法改变的参数的值使模型向真实值靠近使损失函数最小，一般我们采用梯度下降算法，在大量循环中对输出结果向参数求偏导，找到其梯度，就是能使损失函数下降最快的方向，改变参数的值，并不断循环就能找到局部最优的解。

本次实验对深度学习的方法，尤其是如何建立模型，如何训练，步骤和常用的方法函数有了更深的了解。遇到的问题有：torch 版本变化导致 linspace 传入的参数个数和使用方法有变化，通过报错提示查阅了最新的 linspace 使用方法，改变传入的参数就能正常运行了。