安徽大学《深度学习与神经网络》
实验报告 1

学号：	WA2214014	专业：	人工智能	姓名：	杨跃浙
实验日期：	05.19	教师签字：		成绩：	

[实验名称] PyTorch 开发环境配置与简单线性模型实验

[实验目的]

1. 熟悉和掌握 PyTorch 开发环境
2. 熟悉和掌握逻辑回归处理回归问题
3. 熟悉和掌握 Softmax 回归处理分类问题

[实验要求]

1. 采用 Python 语言基于 PyTorch 深度学习框架进行编程
2. 代码可读性强：变量、函数、类等命名可读性强，包含必要的注释
3. 提交实验报告要求：
    - 命名方式："学号–姓名–Lab–N"（N 为实验课序号，即：1–6）；
    - 截止时间：下次实验课当晚 23:59；
    - 提交方式：智慧安大–网络教育平台–作业；
    - 按时提交（过时不补）；

[实验内容]

(一) PyTorch 开发环境配置

1. 安装 Anaconda3（版本不要太新或太旧）：
   - 参考教程 https://developer.aliyun.com/article/1152809
2. 修改 Anaconda 的下载 channel 为国内镜像，加快包下载速度：
   - 参考教程：https://blog.csdn.net/weixin_46649052/article/details/111871455
   - 附国内镜像汇总：https://mdnice.com/writing/a8a3a65f87ec4835ace74cd356f1aa72
3. 安转 CPU 版 PyTorch：
   - 参考教程：https://cloud.baidu.com/article/2679182（为了统一版本，方便答疑，第二步修改为 Python=3.8 为 Python=3.9；第四步命令统一替换为：pip install torch==2.0.0 torchvision==0.15.1 和 pip install d2l==1.0.3 ）
   - 官方教程：https://pytorch.org/
4. （**选做**）有 NVIDIA 独立显卡的同学可以尝试安装 GPU 版 PyTorch （**非必需，如果不熟悉建议就用 CPU 版**）：
   - 参考教程：https://blog.csdn.net/weixin_42496865/article/details/124002488
   - 官方教程：https://pytorch.org/
5. 安装 python 集成开发环境（IDE）并配置 python 解释器为上述利用 Anaconda 创建的某个虚拟环境，此处仅以 PyCharm 为例（VS Code 等感兴趣同学可自行尝试）
   - 参考教程：https://c.biancheng.net/view/5804.html （注意**只安装 PyCharm 不必按照这个教程配置解释器**）
   - 提示：社区版 可免费使用（完成作业已足够）；专业版 教育科研机构也可以免费申请，感兴趣同学可以试试，可参考教程 https://zhuanlan.zhihu.com/p/338280181?utm_id=0（用安大的学生证或学信网验证报告可以申请，学生邮箱好像不行）
   - 新建一个 PyCharm 工程，修改其解释器为上述利用 Anaconda 创建的 PyTorch 环境：
     - 工程和文件新建教程：https://blog.csdn.net/xinghuanmeiying/article/details/79409011
     - 修改 python 解释器教程：https://blog.csdn.net/qq_45899904/article/details/128515380
6. 在上述工程中创建 python 源码文件，导入 PyTorch（即“import torch”)测试是否成功。
7. 在 Windows 命令窗和 PyCharm 命令窗，分别运行 python 解释器导入 PyTorch（即“import torch”）测试是否成功。

（二） 线性模型实验

1. 线性回归的从零开始实现
   - 学习、运行和调试 参考教材 3.2 小节内容
   - **提示**：
     - **参考教材**：《动手学深度学习》 https://zh-v2.d2l.ai/d2l-zh-pytorch.pdf
     - PyTorch **官方文档**可以搜索查阅函数含义：https://pytorch.org/docs/2.0/ ；中文文档 https://pytorch.apachecn.org/
     - 写代码遇到问题**官方论坛**可以提问：https://discuss.pytorch.org/
   - 完成练习题 1，5，7
2. 线性回归的简洁实现：
   - 学习、运行和调试 参考教材 3.3 小节内容
3. 熟悉 Fashine-MNIST 数据集：
   - 学习、运行和调试 参考教材 3.5 小节内容
4. softmax 回归的从零开始实现
   - 学习、运行和调试 参考教材 3.6 小节内容
5. softmax 回归的简洁实现
   - 学习、运行和调试 参考教材 3.7 小节内容
   - 完成练习题 1，2

[实验代码和结果]

（一） PyTorch 开发环境配置

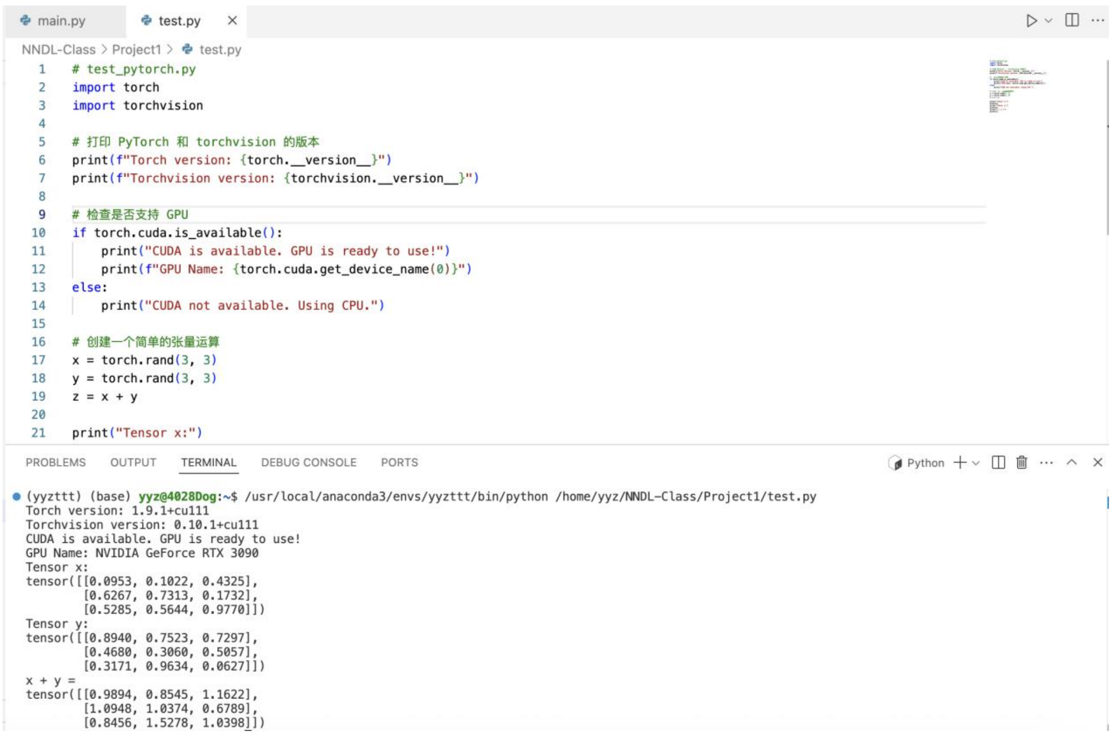
实验环境用的是 Mac 系统+VSCode,远程连接 Linux 服务器,Torch 版本 1.9.1 CUDA11.7 用的环境是我之前已经配置过的 yyzttt，就不配置新的环境了。

实验代码:

```
# test_pytorch.py
import torch
import torchvision

# 打印 PyTorch 和 torchvision 的版本
print(f"Torch version: {torch.__version__}")
print(f"Torchvision version: {torchvision.__version__}")

# 检查是否支持 GPU
if torch.cuda.is_available():
```

```
print("CUDA is available. GPU is ready to use!")
print(f"GPU Name: {torch.cuda.get_device_name(0)}")
else:
print("CUDA not available. Using CPU.")

# 创建一个简单的张量运算
x = torch.rand(3, 3)
y = torch.rand(3, 3)
z = x + y

print("Tensor x:")
print(x)
print("Tensor y:")
print(y)
print("x + y =")
print(z)
```

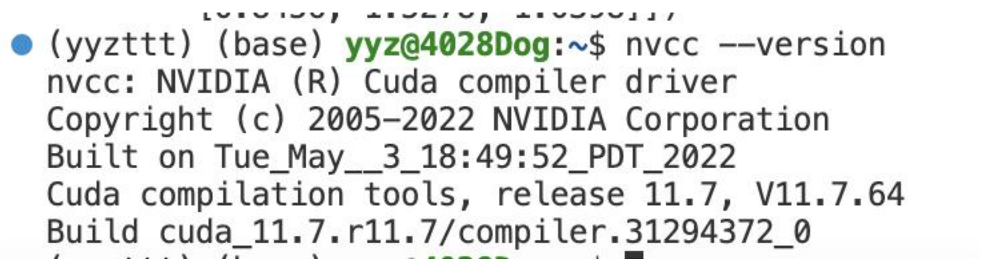
实验结果：



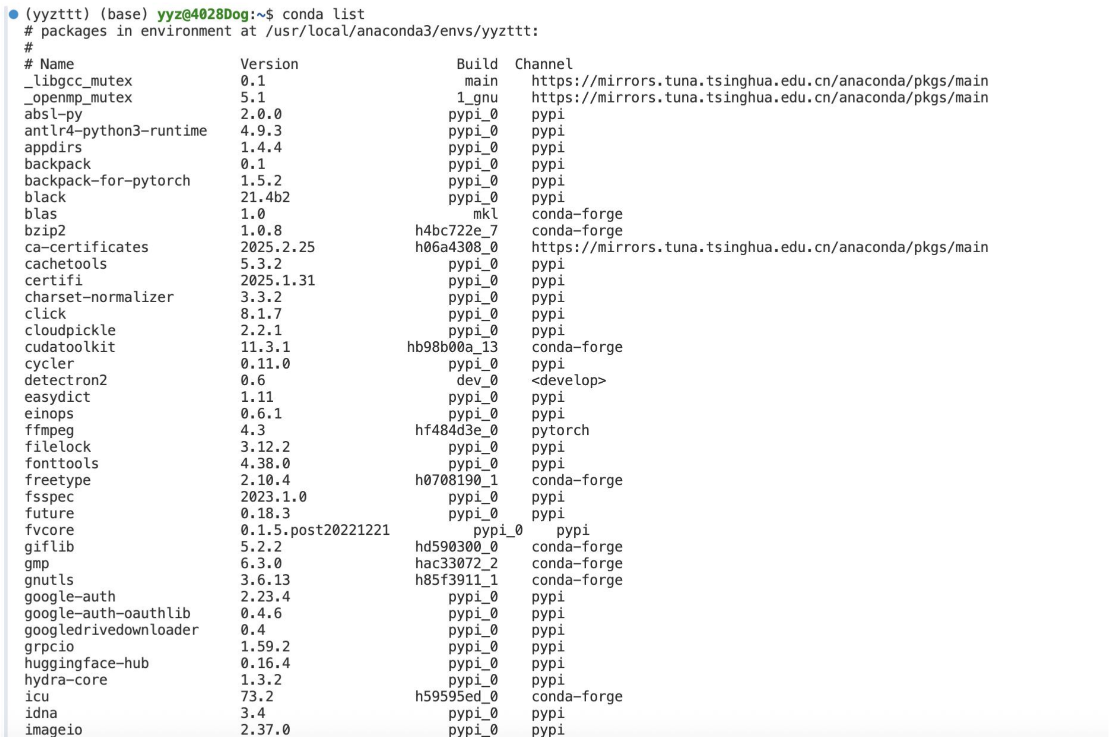
```python
# test_pytorch.py
import torch
import torchvision

# 打印 PyTorch 和 torchvision 的版本
print(f"Torch version: {torch.__version__}")
print(f"Torchvision version: {torchvision.__version__}")

# 检查是否支持 GPU
if torch.cuda.is_available():
    print("CUDA is available. GPU is ready to use!")
    print(f"GPU Name: {torch.cuda.get_device_name(0)}")
else:
    print("CUDA not available. Using CPU.")

# 创建一个简单的张量运算
x = torch.rand(3, 3)
y = torch.rand(3, 3)
z = x + y

print("Tensor x:")
```

```
(yyzttt) (base) yyz@4028Dog:~$ /usr/local/anaconda3/envs/yyzttt/bin/python /home/yyz/NNDL-Class/Project1/test.py
Torch version: 1.9.1+cu111
Torchvision version: 0.10.1+cu111
CUDA is available. GPU is ready to use!
GPU Name: NVIDIA GeForce RTX 3090
Tensor x:
tensor([[0.0953, 0.1022, 0.4325],
        [0.6267, 0.7313, 0.1732],
        [0.5285, 0.5644, 0.9770]])
Tensor y:
tensor([[0.8940, 0.7523, 0.7297],
        [0.4680, 0.3060, 0.5057],
        [0.3171, 0.9634, 0.0627]])
x + y =
tensor([[0.9894, 0.8545, 1.1622],
        [1.0948, 1.0374, 0.6789],
        [0.8456, 1.5278, 1.0398]])
```

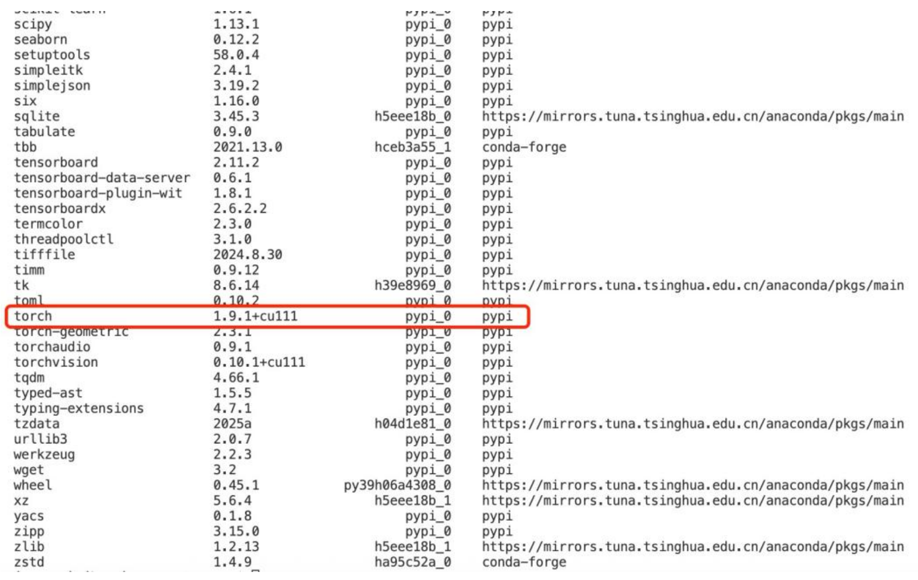
使用代码测试，安装都是正确的，没有问题。



```
● (yyzttt) (base) yyz@4028Dog:~$ nvcc --version
  nvcc: NVIDIA (R) Cuda compiler driver
  Copyright (c) 2005-2022 NVIDIA Corporation
  Built on Tue_May__3_18:49:52_PDT_2022
  Cuda compilation tools, release 11.7, V11.7.64
  Build cuda_11.7.r11.7/compiler.31294372_0
```



```
● (yyzttt) (base) yyz@4028Dog:~$ conda list
  # packages in environment at /usr/local/anaconda3/envs/yyzttt:
  #
  # Name                    Version                   Build  Channel
  _libgcc_mutex             0.1                        main    https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
  _openmp_mutex             5.1                       1_gnu    https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
  absl-py                   2.0.0                    pypi_0    pypi
  antlr4-python3-runtime    4.9.3                    pypi_0    pypi
  appdirs                   1.4.4                    pypi_0    pypi
  backpack                  0.1                      pypi_0    pypi
  backpack-for-pytorch      1.5.2                    pypi_0    pypi
  black                     21.4b2                   pypi_0    pypi
  blas                      1.0                         mkl    conda-forge
  bzip2                     1.0.8                h4bc722e_7    conda-forge
  ca-certificates           2025.2.25            h06a4308_0    https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
  cachetools                5.3.2                    pypi_0    pypi
  certifi                   2025.1.31                pypi_0    pypi
  charset-normalizer        3.3.2                    pypi_0    pypi
  click                     8.1.7                    pypi_0    pypi
  cloudpickle               2.2.1                    pypi_0    pypi
  cudatoolkit               11.3.1              hb98b00a_13    conda-forge
  cycler                    0.11.0                   pypi_0    pypi
  detectron2                0.6                       dev_0    <develop>
  easydict                  1.11                     pypi_0    pypi
  einops                    0.6.1                    pypi_0    pypi
  ffmpeg                    4.3                  hf484d3e_0    pytorch
  filelock                  3.12.2                   pypi_0    pypi
  fonttools                 4.38.0                   pypi_0    pypi
  freetype                  2.10.4               h0708190_1    conda-forge
  fsspec                    2023.1.0                 pypi_0    pypi
  future                    0.18.3                   pypi_0    pypi
  fvcore                    0.1.5.post20221221          pypi_0    pypi
  giflib                    5.2.2                hd590300_0    conda-forge
  gmp                       6.3.0                hac33072_2    conda-forge
  gnutls                    3.6.13               h85f3911_1    conda-forge
  google-auth               2.23.4                   pypi_0    pypi
  google-auth-oauthlib      0.4.6                    pypi_0    pypi
  googledrivedownloader     0.4                      pypi_0    pypi
  grpcio                    1.59.2                   pypi_0    pypi
  huggingface-hub           0.16.4                   pypi_0    pypi
  hydra-core                1.3.2                    pypi_0    pypi
  icu                       73.2                 h59595ed_0    conda-forge
  idna                      3.4                      pypi_0    pypi
  imageio                   2.37.0                   pypi_0    pypi
```



```
scipy                     1.13.1                   pypi_0    pypi
seaborn                   0.12.2                   pypi_0    pypi
setuptools                58.0.4                   pypi_0    pypi
simpleitk                 2.4.1                    pypi_0    pypi
simplejson                3.19.2                   pypi_0    pypi
six                       1.16.0                   pypi_0    pypi
sqlite                    3.45.3               h5eee18b_0    https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
tabulate                  0.9.0                    pypi_0    pypi
tbb                       2021.13.0            hceb3a55_1    conda-forge
tensorboard               2.11.2                   pypi_0    pypi
tensorboard-data-server   0.6.1                    pypi_0    pypi
tensorboard-plugin-wit    1.8.1                    pypi_0    pypi
tensorboardx              2.6.2.2                  pypi_0    pypi
termcolor                 2.3.0                    pypi_0    pypi
threadpoolctl             3.1.0                    pypi_0    pypi
tifffile                  2024.8.30                pypi_0    pypi
timm                      0.9.12                   pypi_0    pypi
tk                        8.6.14               h39e8969_0    https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
toml                      0.10.2                   pypi_0    pypi
torch                     1.9.1+cu111              pypi_0    pypi
torch-geometric           2.3.1                    pypi_0    pypi
torchaudio                0.9.1                    pypi_0    pypi
torchvision               0.10.1+cu111             pypi_0    pypi
tqdm                      4.66.1                   pypi_0    pypi
typed-ast                 1.5.5                    pypi_0    pypi
typing-extensions         4.7.1                    pypi_0    pypi
tzdata                    2025a                h04d1e81_0    https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
urllib3                   2.0.7                    pypi_0    pypi
werkzeug                  2.2.3                    pypi_0    pypi
wget                      3.2                      pypi_0    pypi
wheel                     0.45.1             py39h06a4308_0    https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
xz                        5.6.4                h5eee18b_1    https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
yacs                      0.1.8                    pypi_0    pypi
zipp                      3.15.0                   pypi_0    pypi
zlib                      1.2.13               h5eee18b_1    https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
zstd                      1.4.9                ha95c52a_0    conda-forge
```

（二）　线性模型实验

线性回归的从零开始实现

实验代码：

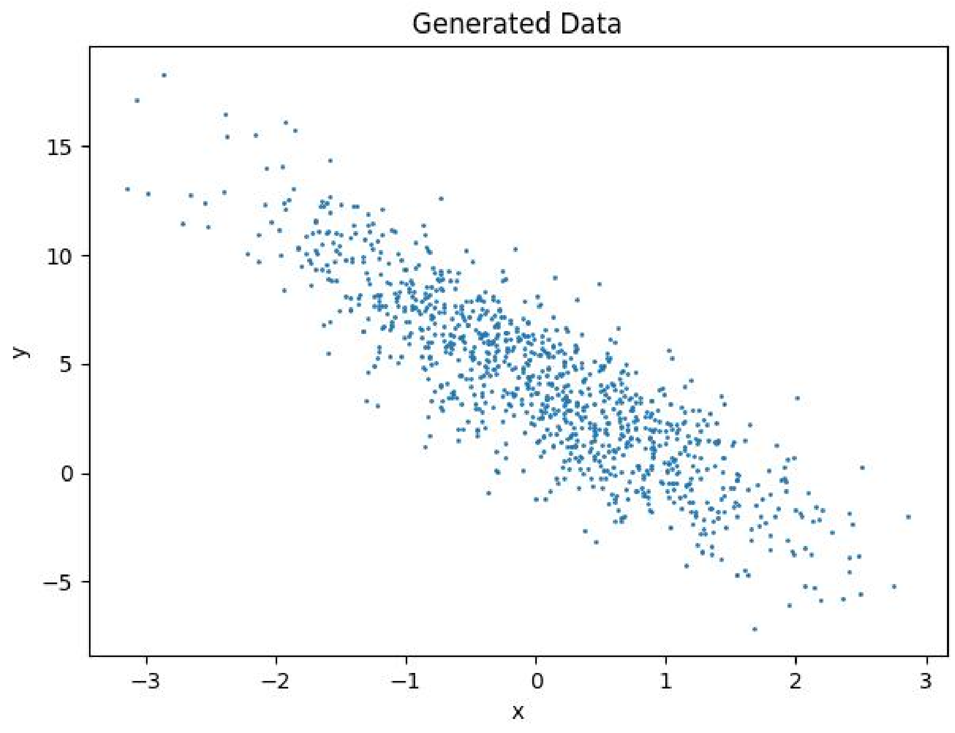
```
import random
import torch
import matplotlib.pyplot as plt
import os
from d2l import torch as d2l

# 创建结果保存目录
save_dir = '/home/yyz/NNDL-Class/Project1/Result'
os.makedirs(save_dir, exist_ok=True)

# 1. 生成数据集
def synthetic_data(w, b, num_examples):
"""生成 y=Xw+b+噪声"""
X = torch.normal(0, 1, (num_examples, len(w)))
y = torch.matmul(X, w) + b
y += torch.normal(0, 0.01, y.shape)
return X, y.reshape((-1, 1))

true_w = torch.tensor([2, -3.4])
true_b = 4.2
features, labels = synthetic_data(true_w, true_b, 1000)

# 输出第一个样本的特征和标签
print('features:', features[0], '\nlabel:', labels[0])

# 绘制散点图观察并保存
plt.figure(figsize=(7, 5))
plt.scatter(features[:, 1].detach().numpy(),
labels.detach().numpy(), 1)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Generated Data')
plt.savefig(f'{save_dir}/linear_regression_data.png')
plt.close()

# 2. 读取数据集
def data_iter(batch_size, features, labels):
num_examples = len(features)
indices = list(range(num_examples))
```

```
# 随机打乱样本顺序
random.shuffle(indices)
for i in range(0, num_examples, batch_size):
batch_indices = torch.tensor(
indices[i: min(i + batch_size, num_examples)])
yield features[batch_indices], labels[batch_indices]

batch_size = 10
# 获取第一个小批量数据并查看
for X, y in data_iter(batch_size, features, labels):
print(X, '\n', y)
break

# 3. 初始化模型参数
w = torch.normal(0, 0.01, size=(2,1), requires_grad=True)
b = torch.zeros(1, requires_grad=True)

# 4. 定义模型
def linreg(X, w, b):
"""线性回归模型"""
return torch.matmul(X, w) + b

# 5. 定义损失函数
def squared_loss(y_hat, y):
"""均方损失"""
return (y_hat - y.reshape(y_hat.shape)) ** 2 / 2

# 6. 定义优化算法
def sgd(params, lr, batch_size):
"""小批量随机梯度下降"""
with torch.no_grad():
for param in params:
param -= lr * param.grad / batch_size
param.grad.zero_()

# 7. 训练
lr = 0.03
num_epochs = 3
net = linreg
```

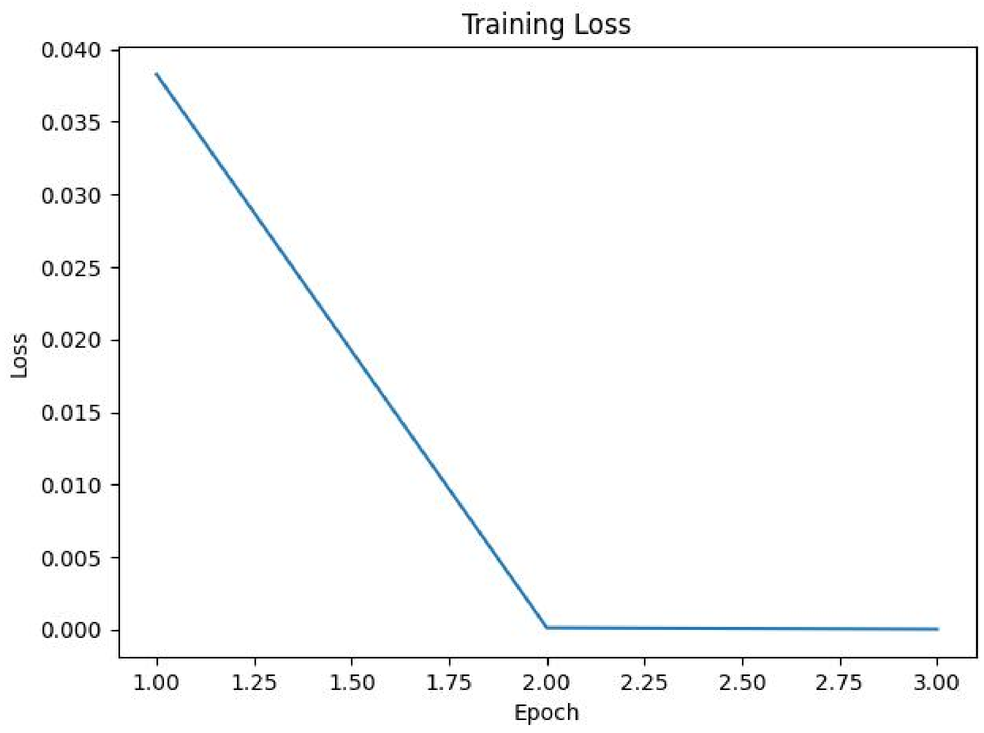
```
loss = squared_loss

# 记录每个 epoch 的损失
losses = []

for epoch in range(num_epochs):
for X, y in data_iter(batch_size, features, labels):
l = loss(net(X, w, b), y) # 计算损失
# 反向传播计算梯度
l.sum().backward()
sgd([w, b], lr, batch_size) # 更新参数
with torch.no_grad():
train_l = loss(net(features, w, b), labels)
mean_loss = float(train_l.mean())
losses.append(mean_loss)
print(f'epoch {epoch + 1}, loss {mean_loss:f}')

# 绘制损失曲线并保存
plt.figure(figsize=(7, 5))
plt.plot(range(1, num_epochs + 1), losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.savefig(f'{save_dir}/linear_regression_loss.png')
plt.close()

# 比较学习到的参数和真实参数
print(f'w 的估计误差: {true_w - w.reshape(true_w.shape)}')
print(f'b 的估计误差: {true_b - b}')
```

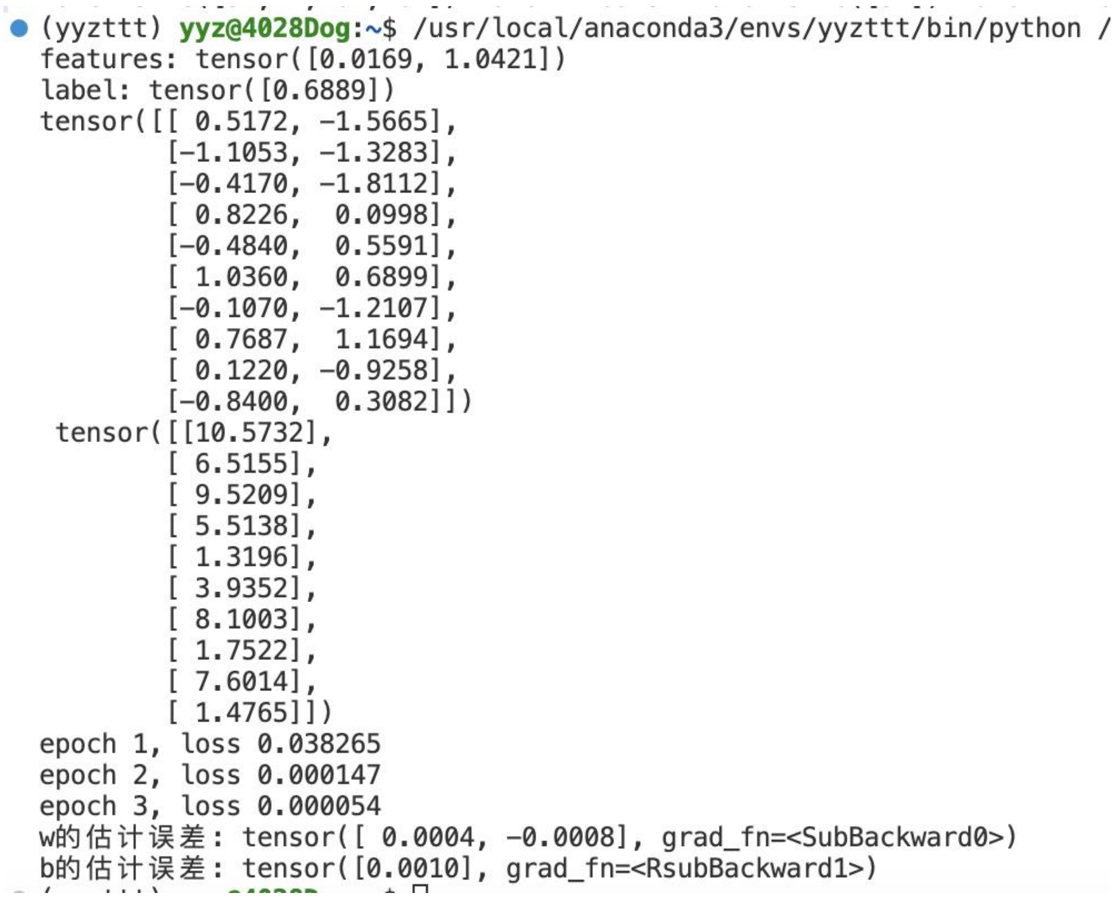
实验结果：

linear_regression_data.png



linear_regression_loss.png





```
● (yyzttt) yyz@4028Dog:~$ /usr/local/anaconda3/envs/yyzttt/bin/python /
  features: tensor([0.0169, 1.0421])
  label: tensor([0.6889])
  tensor([[ 0.5172, -1.5665],
          [-1.1053, -1.3283],
          [-0.4170, -1.8112],
          [ 0.8226,  0.0998],
          [-0.4840,  0.5591],
          [ 1.0360,  0.6899],
          [-0.1070, -1.2107],
          [ 0.7687,  1.1694],
          [ 0.1220, -0.9258],
          [-0.8400,  0.3082]])
   tensor([[10.5732],
          [ 6.5155],
          [ 9.5209],
          [ 5.5138],
          [ 1.3196],
          [ 3.9352],
          [ 8.1003],
          [ 1.7522],
          [ 7.6014],
          [ 1.4765]])
  epoch 1, loss 0.038265
  epoch 2, loss 0.000147
  epoch 3, loss 0.000054
  w的估计误差: tensor([ 0.0004, -0.0008], grad_fn=<SubBackward0>)
  b的估计误差: tensor([0.0010], grad_fn=<RsubBackward1>)
```

练习 1：如果将权重初始化为零，会发生什么？算法仍然有效吗？

权重初始化是模型训练的起始步骤，其取值对算法的运行和收敛有着重要影响。对于线性回归算法而言，将权重初始化为零，算法依然能够有效运行。线性回归模型基于最小二乘法原理，目标函数是一个凸函数，这意味着该函数只有一个全局最小值，不存在局部最优解的干扰。在这种情况下，即便权重从全零状态出发，利用梯度下降算法进行参数更新时，由于目标函数的凸性，梯度的方向始终会指向全局最优解的位置，从而引导权重逐步朝着正确的方向进行调整和优化，最终实现模型的有效训练和参数收敛。

然而，当模型复杂度提升至神经网络时，全零初始化权重会带来显著问题。神经网络包含多个神经元和隐藏层，每个神经元承担着学习不同特征的任务。若将权重全部初始化为零，在正向传播过程中，所有神经元会产生相同的输出；在

反向传播阶段，由于参数更新规则的一致性，所有神经元接收的梯度相同，进而导致它们以相同的方式更新权重。这种对称性使得不同的神经元始终学习相同的特征，无法实现多样化的特征提取和学习，极大地限制了神经网络的表达能力和学习效果，模型难以有效拟合复杂的数据模式，最终导致训练失败或性能低下。

练习 5: 为什么在 squared_loss 函数中需要使用 reshape 函数?

在机器学习模型的训练过程中，损失函数的准确计算是衡量模型预测效果的关键。在 squared_loss 函数（均方误差损失函数）的计算中，reshape 函数发挥着不可或缺的作用。在实际的模型训练中，数据通常以批量的形式输入模型，这就引入了维度上的差异问题。真实标签 y 在数据加载和处理过程中，可能以一维张量的形式存在，例如表示为一系列的标量值；而模型的预测值 y_hat 由于批量处理的原因，往往是二维张量，其形状可能为 [batch_size, num_outputs]，其中 batch_size 表示批量大小，num_outputs 表示输出的维度。

为了确保均方误差的正确计算，需要保证预测值 y_hat 和真实标签 y 的形状完全一致。通过使用 y.reshape (y_hat.shape) 这一操作，能够将真实标签 y 的维度进行调整，使其与预测值 y_hat 的形状相匹配。只有在形状一致的情况下，才能逐元素地计算两者之间的差值，并进一步平方求和、取平均，得到准确的均方误差值。如果不进行形状调整，在计算差值时会引发维度不匹配的错误，导致无法正确计算损失，进而影响模型的训练和优化过程。

练习 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?

data_iter 函数（数据迭代器）用于将数据划分为多个批次，以便模型进行批量训练。当样本个数不能被批量大小整除时，data_iter 函数的运行机制会进行相应调整。在数据迭代过程中，data_iter 函数首先会按照批量大小处理所有完整的批次，这些批次中的样本数量均等于设定的 batch_size。

对于最后一个批次，由于剩余样本数量不足 batch_size，其样本数量会小于其他完整批次。这一处理逻辑是通过 batch_indices = torch.tensor (indices [i: min（i + batch_size, num_examples)]）语句实现的。其中，min（i + batch_size, num_examples）这一操作确保了索引不会超出样本总数的范围。在遍历数据索引时，当接近样本末尾，该操作会自动选取剩余的所有样本作为最后一个批次，保证数据集中的每一个样本都能参与到模型的训练过程中。这种处理方式既满足了批量训练的需求，又充分利用了所有可用数据，在保证训练效率的同时，避免了数据的浪费，确保模型能够从完整的数据信息中学习到有效的模式和规律 。

线性回归的简洁实现

实验代码：

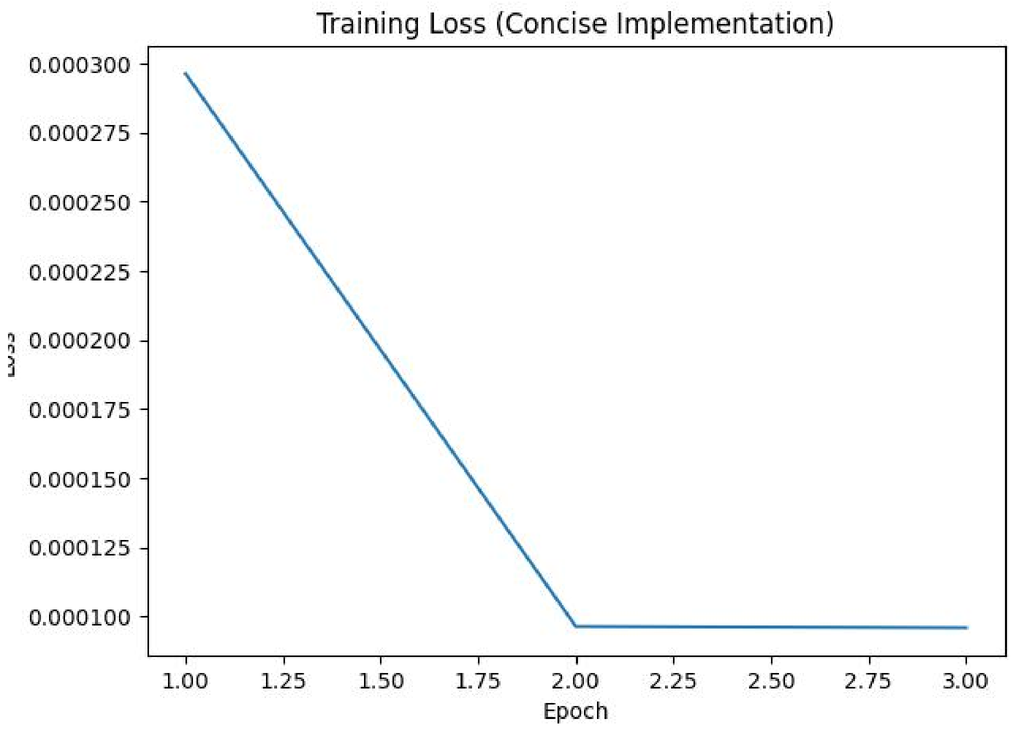
```
import numpy as np
import torch
from torch.utils import data
import matplotlib.pyplot as plt
import os
from d2l import torch as d2l

# 确保结果保存目录存在
save_dir = '/home/yyz/NNDL-Class/Project1/Result'
```

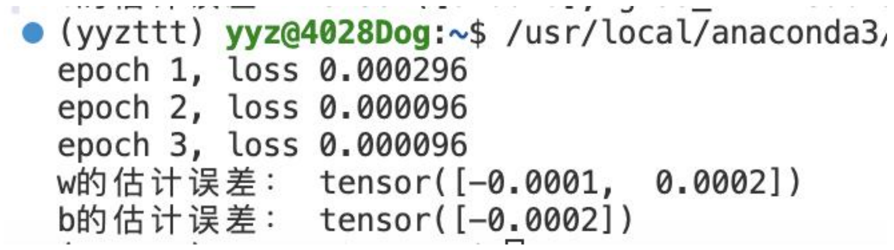
```
os.makedirs(save_dir, exist_ok=True)

# 1. 生成数据集
true_w = torch.tensor([2, -3.4])
true_b = 4.2
features, labels = d2l.synthetic_data(true_w, true_b, 1000)

# 2. 读取数据集
def load_array(data_arrays, batch_size, is_train=True):
"""构造一个 PyTorch 数据迭代器"""
dataset = data.TensorDataset(*data_arrays)
return data.DataLoader(dataset, batch_size,
shuffle=is_train)

batch_size = 10
data_iter = load_array((features, labels), batch_size)

# 查看第一个批量
next(iter(data_iter))

# 3. 定义模型
from torch import nn
net = nn.Sequential(nn.Linear(2, 1))

# 4. 初始化模型参数
net[0].weight.data.normal_(0, 0.01)
net[0].bias.data.fill_(0)

# 5. 定义损失函数
loss = nn.MSELoss()

# 6. 定义优化算法
trainer = torch.optim.SGD(net.parameters(), lr=0.03)

# 7. 训练
num_epochs = 3
losses = []

for epoch in range(num_epochs):
for X, y in data_iter:
```

```
l = loss(net(X), y)
trainer.zero_grad()
l.backward()
trainer.step()
l = loss(net(features), labels)
losses.append(float(l))
print(f'epoch {epoch + 1}, loss {l:f}')

# 绘制损失曲线并保存
plt.figure(figsize=(7, 5))
plt.plot(range(1, num_epochs + 1), losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss (Concise Implementation)')
plt.savefig(f'{save_dir}/linear_regression_concise_loss.p
ng')
plt.close()

# 比较学习到的参数和真实参数
w = net[0].weight.data
print('w 的估计误差：', true_w - w.reshape(true_w.shape))
b = net[0].bias.data
print('b 的估计误差：', true_b - b)
```

实验结果：

linear_regression_concise_loss.png





```
● (yyzttt) yyz@4028Dog:~$ /usr/local/anaconda3/
  epoch 1, loss 0.000296
  epoch 2, loss 0.000096
  epoch 3, loss 0.000096
  w的估计误差：  tensor([-0.0001,  0.0002])
  b的估计误差：  tensor([-0.0002])
```

熟悉 Fashine—MNIST 数据集

实验代码：

```python
import torch
import torchvision
from torch.utils import data
from torchvision import transforms
import matplotlib.pyplot as plt
import os
from d2l import torch as d2l

# 确保结果保存目录存在
save_dir = '/home/yyz/NNDL-Class/Project1/Result'
```

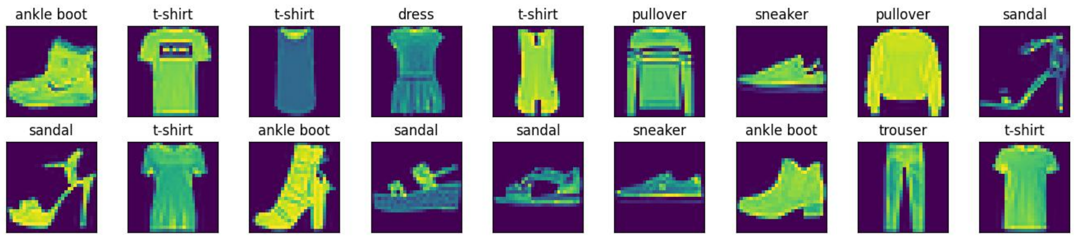
```
os.makedirs(save_dir, exist_ok=True)

# 1. 读取数据集
# 通过 ToTensor 实例将图像数据从 PIL 类型变换成 32 位浮点数格式，
# 并除以 255 使得所有像素的数值均在 0～1 之间
trans = transforms.ToTensor()
mnist_train = torchvision.datasets.FashionMNIST(
root="/home/yyz/NNDL-Class/Project1/Data", train=True,
transform=trans, download=True)
mnist_test = torchvision.datasets.FashionMNIST(
root="/home/yyz/NNDL-Class/Project1/Data", train=False,
transform=trans, download=True)

# 查看数据集大小
print(len(mnist_train), len(mnist_test))

# 查看图像形状
print(mnist_train[0][0].shape)

# 定义文本标签
def get_fashion_mnist_labels(labels):
"""返回 Fashion-MNIST 数据集的文本标签"""
text_labels = ['t-shirt', 'trouser', 'pullover', 'dress',
'coat',
'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']
return [text_labels[int(i)] for i in labels]

# 定义可视化函数
def show_images(imgs, num_rows, num_cols, titles=None,
scale=1.5, save_path=None):
"""绘制图像列表并保存"""
figsize = (num_cols * scale, num_rows * scale)
_, axes = plt.subplots(num_rows, num_cols, figsize=figsize)
axes = axes.flatten()
for i, (ax, img) in enumerate(zip(axes, imgs)):
if torch.is_tensor(img):
# 图片张量
ax.imshow(img.numpy())
else:
```

```
# PIL 图片
ax.imshow(img)
ax.axes.get_xaxis().set_visible(False)
ax.axes.get_yaxis().set_visible(False)
if titles:
ax.set_title(titles[i])
plt.tight_layout()
if save_path:
plt.savefig(save_path)
plt.close()

# 展示训练集中的一些样本并保存
X, y = next(iter(data.DataLoader(mnist_train,
batch_size=18)))
show_images(X.reshape(18, 28, 28), 2, 9,
titles=get_fashion_mnist_labels(y),
save_path=f'{save_dir}/fashion_mnist_samples.png')

# 2. 读取小批量
batch_size = 256
def get_dataloader_workers():
"""使用 4 个进程来读取数据"""
return 4

train_iter = data.DataLoader(mnist_train, batch_size,
shuffle=True,
num_workers=get_dataloader_workers())

# 测量读取时间
timer = d2l.Timer()
for X, y in train_iter:
continue
print(f'{timer.stop():.2f} sec')

# 3. 整合所有组件
def load_data_fashion_mnist(batch_size, resize=None):
"""下载 Fashion-MNIST 数据集，然后将其加载到内存中"""
trans = [transforms.ToTensor()]
if resize:
```

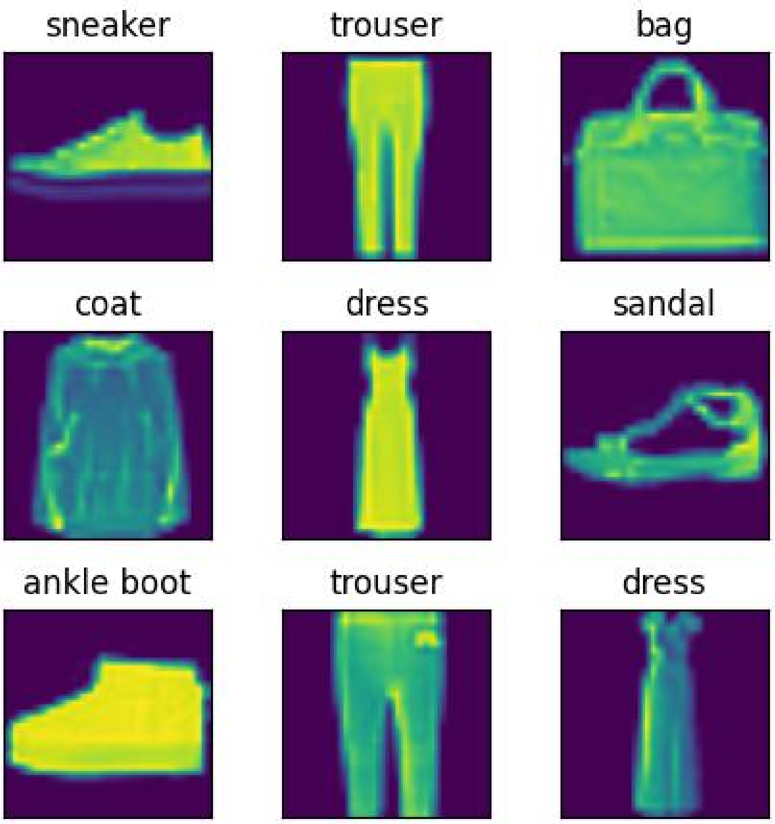
```
trans.insert(0, transforms.Resize(resize))
trans = transforms.Compose(trans)
mnist_train = torchvision.datasets.FashionMNIST(
root="/home/yyz/NNDL-Class/Project1/Data", train=True,
transform=trans, download=True)
mnist_test = torchvision.datasets.FashionMNIST(
root="/home/yyz/NNDL-Class/Project1/Data", train=False,
transform=trans, download=True)
return (data.DataLoader(mnist_train, batch_size,
shuffle=True,
num_workers=get_dataloader_workers()),
data.DataLoader(mnist_test, batch_size, shuffle=False,
num_workers=get_dataloader_workers()))

# 测试调整大小功能
train_iter, test_iter = load_data_fashion_mnist(32,
resize=64)
for X, y in train_iter:
print(X.shape, X.dtype, y.shape, y.dtype)
break

# 保存一个调整大小后的样本图像
resized_imgs = X[:9]
show_images(resized_imgs.reshape(9, 64, 64), 3, 3,
titles=get_fashion_mnist_labels(y[:9]),
save_path=f'{save_dir}/fashion_mnist_resized.png')
```

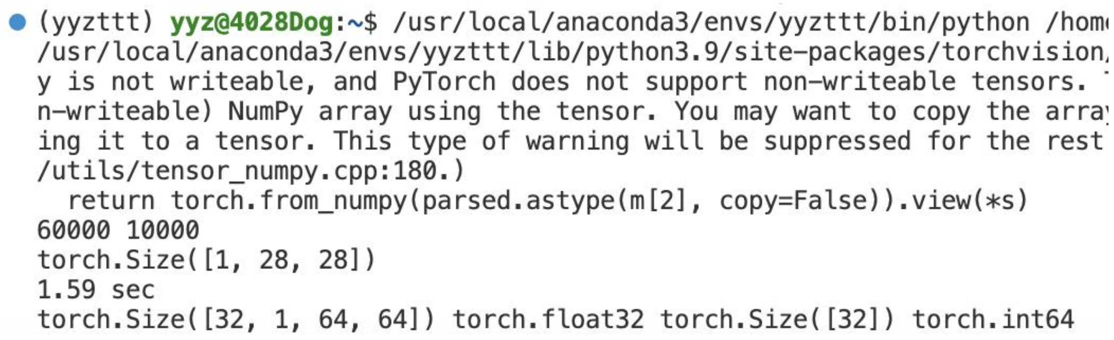
实验结果：

fashion_mnist_samples.png



fashion_mnist_resized.png





```
● (yyzttt) yyz@4028Dog:~$ /usr/local/anaconda3/envs/yyzttt/bin/python /hom
/usr/local/anaconda3/envs/yyzttt/lib/python3.9/site-packages/torchvision
y is not writeable, and PyTorch does not support non-writeable tensors.
n-writeable) NumPy array using the tensor. You may want to copy the arra
ing it to a tensor. This type of warning will be suppressed for the rest
/utils/tensor_numpy.cpp:180.)
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
60000 10000
torch.Size([1, 28, 28])
1.59 sec
torch.Size([32, 1, 64, 64]) torch.float32 torch.Size([32]) torch.int64
```

softmax 回归的从零开始实现

实验代码：

```
import torch
import matplotlib.pyplot as plt
import os
from d2l import torch as d2l

# 确保结果保存目录存在
```

```
save_dir = '/home/yyz/NNDL-Class/Project1/Result'
os.makedirs(save_dir, exist_ok=True)

# 确保数据保存目录存在
data_dir = '/home/yyz/NNDL-Class/Project1/Data'
os.makedirs(data_dir, exist_ok=True)

# 1. 设置超参数和自定义加载数据集函数
batch_size = 256

# 修改加载数据集函数以使用正确的数据路径
def load_data_fashion_mnist_custom(batch_size,
resize=None):
"""下载 Fashion-MNIST 数据集到自定义路径，然后将其加载到内存中"""
import torchvision
from torchvision import transforms
from torch.utils import data
trans = [transforms.ToTensor()]
if resize:
trans.insert(0, transforms.Resize(resize))
trans = transforms.Compose(trans)
mnist_train = torchvision.datasets.FashionMNIST(
root=data_dir, train=True, transform=trans, download=True)
mnist_test = torchvision.datasets.FashionMNIST(
root=data_dir, train=False, transform=trans, download=True)
return (data.DataLoader(mnist_train, batch_size,
shuffle=True,
num_workers=d2l.get_dataloader_workers()),
data.DataLoader(mnist_test, batch_size, shuffle=False,
num_workers=d2l.get_dataloader_workers()))

# 使用自定义函数加载数据集
train_iter, test_iter =
load_data_fashion_mnist_custom(batch_size)

# 2. 初始化模型参数
num_inputs = 784
num_outputs = 10
```

```
W = torch.normal(0, 0.01, size=(num_inputs, num_outputs),
requires_grad=True)
b = torch.zeros(num_outputs, requires_grad=True)

# 3. 定义 softmax 操作
def softmax(X):
X_exp = torch.exp(X)
partition = X_exp.sum(1, keepdim=True)
return X_exp / partition # 这里应用了广播机制

# 4. 定义模型
def net(X):
return softmax(torch.matmul(X.reshape((-1, W.shape[0])), W)
+ b)

# 5. 定义损失函数
def cross_entropy(y_hat, y):
return -torch.log(y_hat[range(len(y_hat)), y])

# 6. 定义精度计算函数
def accuracy(y_hat, y):
"""计算预测正确的数量"""
if len(y_hat.shape) > 1 and y_hat.shape[1] > 1:
y_hat = y_hat.argmax(axis=1)
cmp = y_hat.type(y.dtype) == y
return float(cmp.type(y.dtype).sum())

def evaluate_accuracy(net, data_iter):
"""计算在指定数据集上模型的精度"""
if isinstance(net, torch.nn.Module):
net.eval() # 设置为评估模式
metric = Accumulator(2) # 正确预测数、预测总数
with torch.no_grad():
for X, y in data_iter:
metric.add(accuracy(net(X), y), y.numel())
return metric[0] / metric[1]

# 定义用于累加的实用程序类
class Accumulator:
```

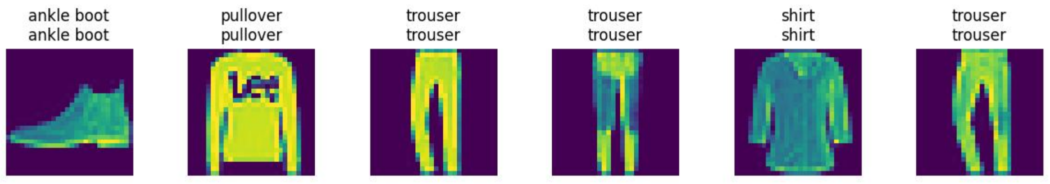
```
"""在 n 个变量上累加"""
def __init__(self, n):
self.data = [0.0] * n
def add(self, *args):
self.data = [a + float(b) for a, b in zip(self.data, args)]
def reset(self):
self.data = [0.0] * len(self.data)
def __getitem__(self, idx):
return self.data[idx]

# 7. 训练函数定义
def train_epoch_ch3(net, train_iter, loss, updater):
"""训练模型一个迭代周期（定义见第 3 章）"""
# 设置为训练模式
if isinstance(net, torch.nn.Module):
net.train()
# 训练损失总和、训练准确度总和、样本数
metric = Accumulator(3)
for X, y in train_iter:
# 计算梯度并更新参数
y_hat = net(X)
l = loss(y_hat, y)
if isinstance(updater, torch.optim.Optimizer):
# 使用 PyTorch 内置的优化器和损失函数
updater.zero_grad()
l.mean().backward()
updater.step()
else:
# 使用定制的优化器和损失函数
l.sum().backward()
updater(X.shape[0])
metric.add(float(l.sum()), accuracy(y_hat, y), y.numel())
# 返回训练损失和训练精度
return metric[0] / metric[2], metric[1] / metric[2]

# 修改训练函数，保存图表而非显示
def train_ch3(net, train_iter, test_iter, loss, num_epochs,
updater, save_path=None):
"""训练模型并保存结果图表"""
```

```
# 记录训练过程
train_losses = []
train_accs = []
test_accs = []
for epoch in range(num_epochs):
train_metrics = train_epoch_ch3(net, train_iter, loss,
updater)
test_acc = evaluate_accuracy(net, test_iter)
train_loss, train_acc = train_metrics
train_losses.append(train_loss)
train_accs.append(train_acc)
test_accs.append(test_acc)
print(f'epoch {epoch+1}, loss {train_loss:.4f}, train acc
{train_acc:.3f}, test acc {test_acc:.3f}')
# 绘制训练过程图表并保存
plt.figure(figsize=(10, 6))
epochs = list(range(1, num_epochs + 1))
plt.plot(epochs, train_losses, label='train loss')
plt.plot(epochs, train_accs, label='train acc')
plt.plot(epochs, test_accs, label='test acc')
plt.xlabel('epoch')
plt.ylabel('metric')
plt.legend()
plt.grid()
if save_path:
plt.savefig(save_path)
plt.close()

# 8. 训练模型
lr = 0.1
def updater(batch_size):
return d2l.sgd([W, b], lr, batch_size)

num_epochs = 10
train_ch3(net, train_iter, test_iter, cross_entropy,
num_epochs, updater,
save_path=f'{save_dir}/softmax_scratch_training.png')

# 9. 预测并保存结果
```

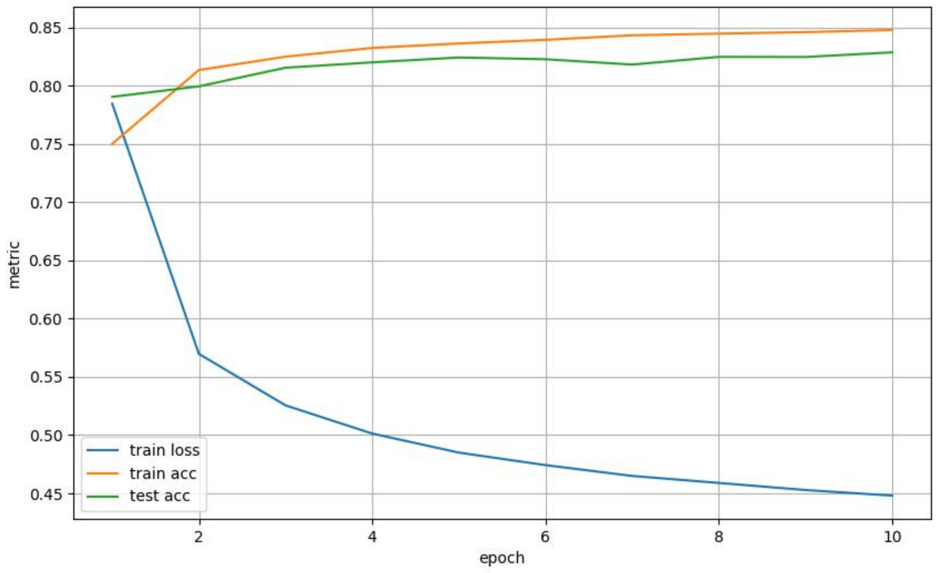
```
def predict_ch3(net, test_iter, n=6, save_path=None):
"""预测标签并保存结果"""
for X, y in test_iter:
break
trues = d2l.get_fashion_mnist_labels(y)
preds =
d2l.get_fashion_mnist_labels(net(X).argmax(axis=1))
titles = [true +'\n' + pred for true, pred in zip(trues,
preds)]
plt.figure(figsize=(12, 2))
for i in range(n):
plt.subplot(1, n, i + 1)
plt.imshow(X[i].reshape(28, 28).detach().numpy())
plt.axis('off')
plt.title(titles[i])
plt.tight_layout()
if save_path:
plt.savefig(save_path)
plt.close()

predict_ch3(net, test_iter, n=6,
save_path=f'{save_dir}/softmax_scratch_predictions.png')
```

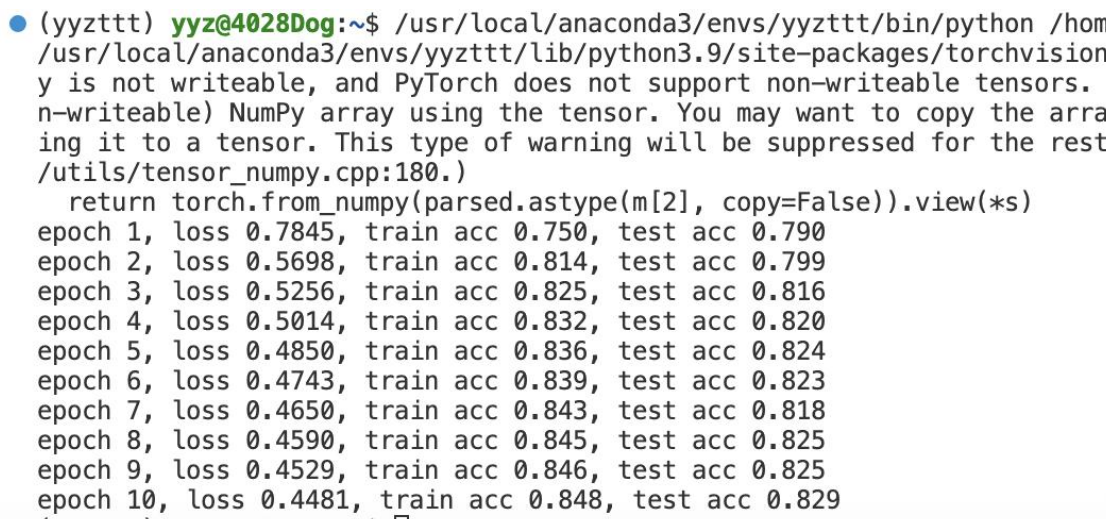
实验结果：

softmax_scratch_predictions.png



softmax_scratch_training.png





```
● (yyzttt) yyz@4028Dog:~$ /usr/local/anaconda3/envs/yyzttt/bin/python /hom
  /usr/local/anaconda3/envs/yyzttt/lib/python3.9/site-packages/torchvision
  y is not writeable, and PyTorch does not support non-writeable tensors.
  n-writeable) NumPy array using the tensor. You may want to copy the arra
  ing it to a tensor. This type of warning will be suppressed for the rest
  /utils/tensor_numpy.cpp:180.)
    return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
  epoch 1, loss 0.7845, train acc 0.750, test acc 0.790
  epoch 2, loss 0.5698, train acc 0.814, test acc 0.799
  epoch 3, loss 0.5256, train acc 0.825, test acc 0.816
  epoch 4, loss 0.5014, train acc 0.832, test acc 0.820
  epoch 5, loss 0.4850, train acc 0.836, test acc 0.824
  epoch 6, loss 0.4743, train acc 0.839, test acc 0.823
  epoch 7, loss 0.4650, train acc 0.843, test acc 0.818
  epoch 8, loss 0.4590, train acc 0.845, test acc 0.825
  epoch 9, loss 0.4529, train acc 0.846, test acc 0.825
  epoch 10, loss 0.4481, train acc 0.848, test acc 0.829
```

softmax 回归的简洁实现

实验代码：

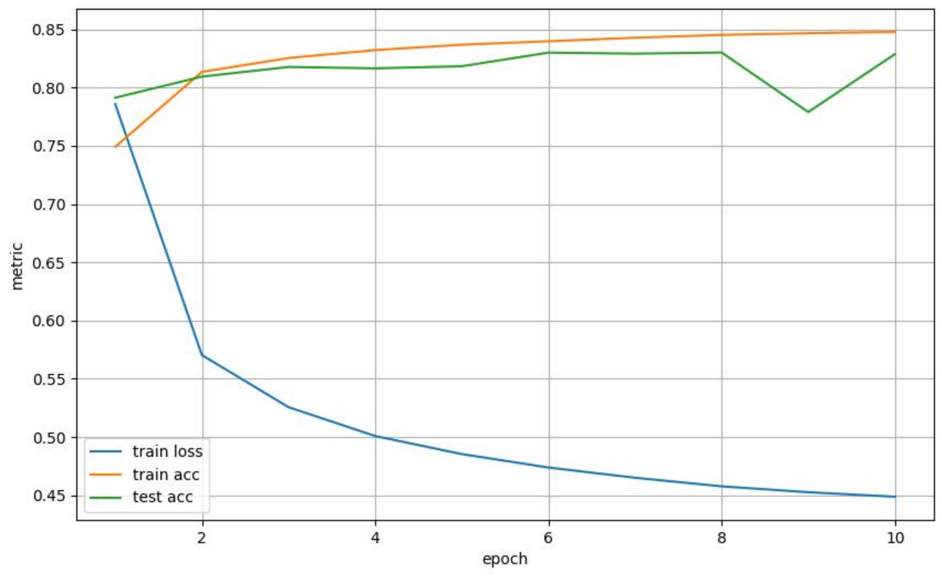
```
import torch
from torch import nn
import matplotlib.pyplot as plt
import os
import torchvision
from torchvision import transforms
from torch.utils import data
from d2l import torch as d2l

# 确保结果保存目录存在
save_dir = '/home/yyz/NNDL-Class/Project1/Result'
os.makedirs(save_dir, exist_ok=True)

# 确保数据保存目录存在
data_dir = '/home/yyz/NNDL-Class/Project1/Data'
os.makedirs(data_dir, exist_ok=True)

# 1. 设置超参数和自定义加载数据集函数
batch_size = 256

# 修改加载数据集函数以使用正确的数据路径
def load_data_fashion_mnist_custom(batch_size,
resize=None):
"""下载 Fashion-MNIST 数据集到自定义路径，然后将其加载到内存中"""
trans = [transforms.ToTensor()]
if resize:
trans.insert(0, transforms.Resize(resize))
trans = transforms.Compose(trans)
mnist_train = torchvision.datasets.FashionMNIST(
root=data_dir, train=True, transform=trans, download=True)
mnist_test = torchvision.datasets.FashionMNIST(
root=data_dir, train=False, transform=trans, download=True)
return (data.DataLoader(mnist_train, batch_size,
shuffle=True,
num_workers=d2l.get_dataloader_workers()),
data.DataLoader(mnist_test, batch_size, shuffle=False,
num_workers=d2l.get_dataloader_workers()))

# 使用自定义函数加载数据集
train_iter, test_iter =
load_data_fashion_mnist_custom(batch_size)

# 2. 初始化模型参数
# PyTorch 不会隐式地调整输入的形状，因此定义展平层(flatten)
net = nn.Sequential(nn.Flatten(), nn.Linear(784, 10))
```

```
def init_weights(m):
if type(m) == nn.Linear:
nn.init.normal_(m.weight, std=0.01)

net.apply(init_weights)

# 3. 定义损失函数
loss = nn.CrossEntropyLoss(reduction='none')

# 4. 定义优化算法
trainer = torch.optim.SGD(net.parameters(), lr=0.1)

# 5. 训练模型
num_epochs = 10

# 修改自 d2l 的 train_ch3 函数，确保图表保存而不是显示
def train_ch3_concise(net, train_iter, test_iter, loss,
num_epochs, trainer, save_path=None):
"""训练模型并保存结果图表"""
# 记录训练过程
train_losses = []
train_accs = []
test_accs = []
# 定义准确率计算函数
def accuracy(y_hat, y):
"""计算预测正确的数量"""
if len(y_hat.shape) > 1 and y_hat.shape[1] > 1:
y_hat = y_hat.argmax(axis=1)
cmp = y_hat.type(y.dtype) == y
return float(cmp.type(y.dtype).sum())
# 定义评估函数
def evaluate_accuracy(net, data_iter):
"""计算在指定数据集上模型的精度"""
if isinstance(net, torch.nn.Module):
net.eval() # 设置为评估模式
metric = Accumulator(2) # 正确预测数、预测总数
with torch.no_grad():
for X, y in data_iter:
```

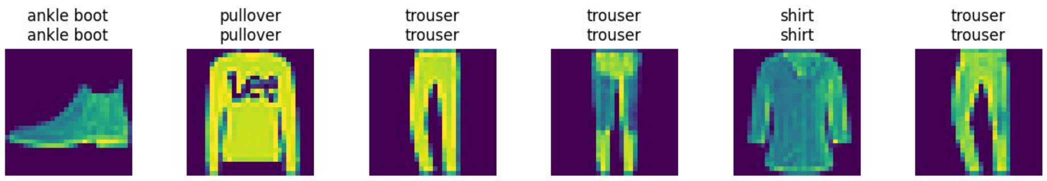
```
metric.add(accuracy(net(X), y), y.numel())
return metric[0] / metric[1]
# 定义累加器
class Accumulator:
"""在 n 个变量上累加"""
def __init__(self, n):
self.data = [0.0] * n
def add(self, *args):
self.data = [a + float(b) for a, b in zip(self.data, args)]
def reset(self):
self.data = [0.0] * len(self.data)
def __getitem__(self, idx):
return self.data[idx]
for epoch in range(num_epochs):
# 训练
net.train()
metric = Accumulator(3) # 训练损失之和，训练准确率之和，样本数
for X, y in train_iter:
y_hat = net(X)
l = loss(y_hat, y)
trainer.zero_grad()
l.mean().backward()
trainer.step()
metric.add(float(l.sum()), accuracy(y_hat, y), y.numel())
train_loss = metric[0] / metric[2]
train_acc = metric[1] / metric[2]
# 测试
test_acc = evaluate_accuracy(net, test_iter)
train_losses.append(train_loss)
train_accs.append(train_acc)
test_accs.append(test_acc)
print(f'epoch {epoch+1}, loss {train_loss:.4f}, train acc
{train_acc:.3f}, test acc {test_acc:.3f}')
# 绘制训练过程图表并保存
plt.figure(figsize=(10, 6))
epochs = list(range(1, num_epochs + 1))
plt.plot(epochs, train_losses, label='train loss')
plt.plot(epochs, train_accs, label='train acc')
plt.plot(epochs, test_accs, label='test acc')
plt.xlabel('epoch')
plt.ylabel('metric')
plt.legend()
plt.grid()
if save_path:
plt.savefig(save_path)
plt.close()

# 训练模型并保存结果
train_ch3_concise(net, train_iter, test_iter, loss,
num_epochs, trainer,
save_path=f'{save_dir}/softmax_concise_training.png')

# 预测并保存结果
def predict_fashion_mnist(net, test_iter, n=6,
save_path=None):
"""预测 Fashion-MNIST 测试数据标签并保存结果"""
X, y = next(iter(test_iter))
# 获取 Fashion-MNIST 标签
def get_fashion_mnist_labels(labels):
"""返回 Fashion-MNIST 数据集的文本标签"""
text_labels = ['t-shirt', 'trouser', 'pullover', 'dress',
'coat',
'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']
return [text_labels[int(i)] for i in labels]
trues = get_fashion_mnist_labels(y)
preds = get_fashion_mnist_labels(net(X).argmax(axis=1))
titles = [true + '\n' + pred for true, pred in zip(trues,
preds)]
plt.figure(figsize=(12, 2))
for i in range(n):
plt.subplot(1, n, i + 1)
plt.imshow(X[i].reshape(28, 28).detach().numpy())
plt.axis('off')
plt.title(titles[i])
plt.tight_layout()
if save_path:
plt.savefig(save_path)
plt.close()

predict_fashion_mnist(net, test_iter, n=6,
save_path=f'{save_dir}/softmax_concise_predictions.png')

# 练习 2：尝试调整超参数并保存结果
# 增加 L2 正则化来解决过拟合问题
net_reg = nn.Sequential(nn.Flatten(), nn.Linear(784, 10))
net_reg.apply(init_weights)
trainer_reg = torch.optim.SGD(net_reg.parameters(), lr=0.1,
weight_decay=0.001) # 添加 L2 正则化

num_epochs_reg = 30
train_ch3_concise(net_reg, train_iter, test_iter, loss,
num_epochs_reg, trainer_reg,
save_path=f'{save_dir}/softmax_concise_regularized.png')
```

实验结果：

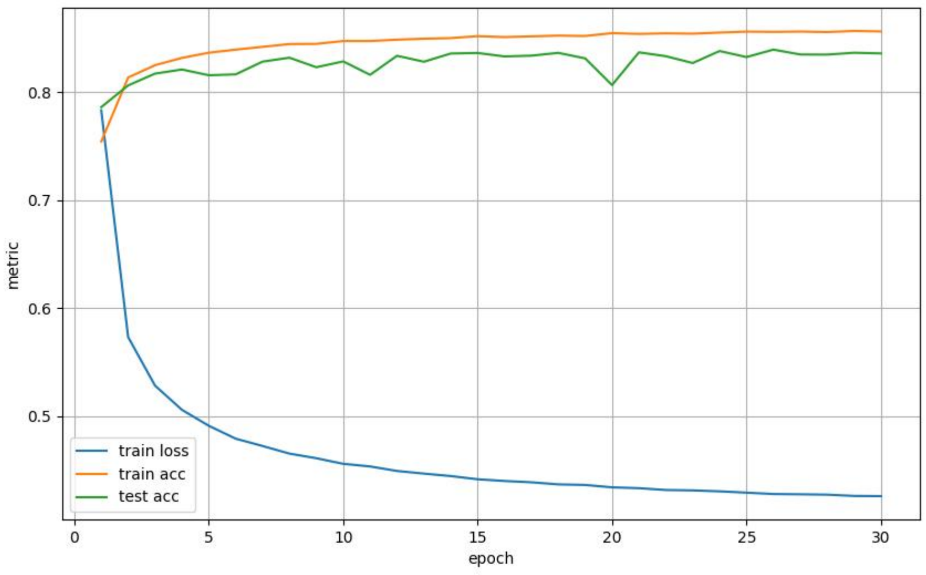
softmax_concise_training.png

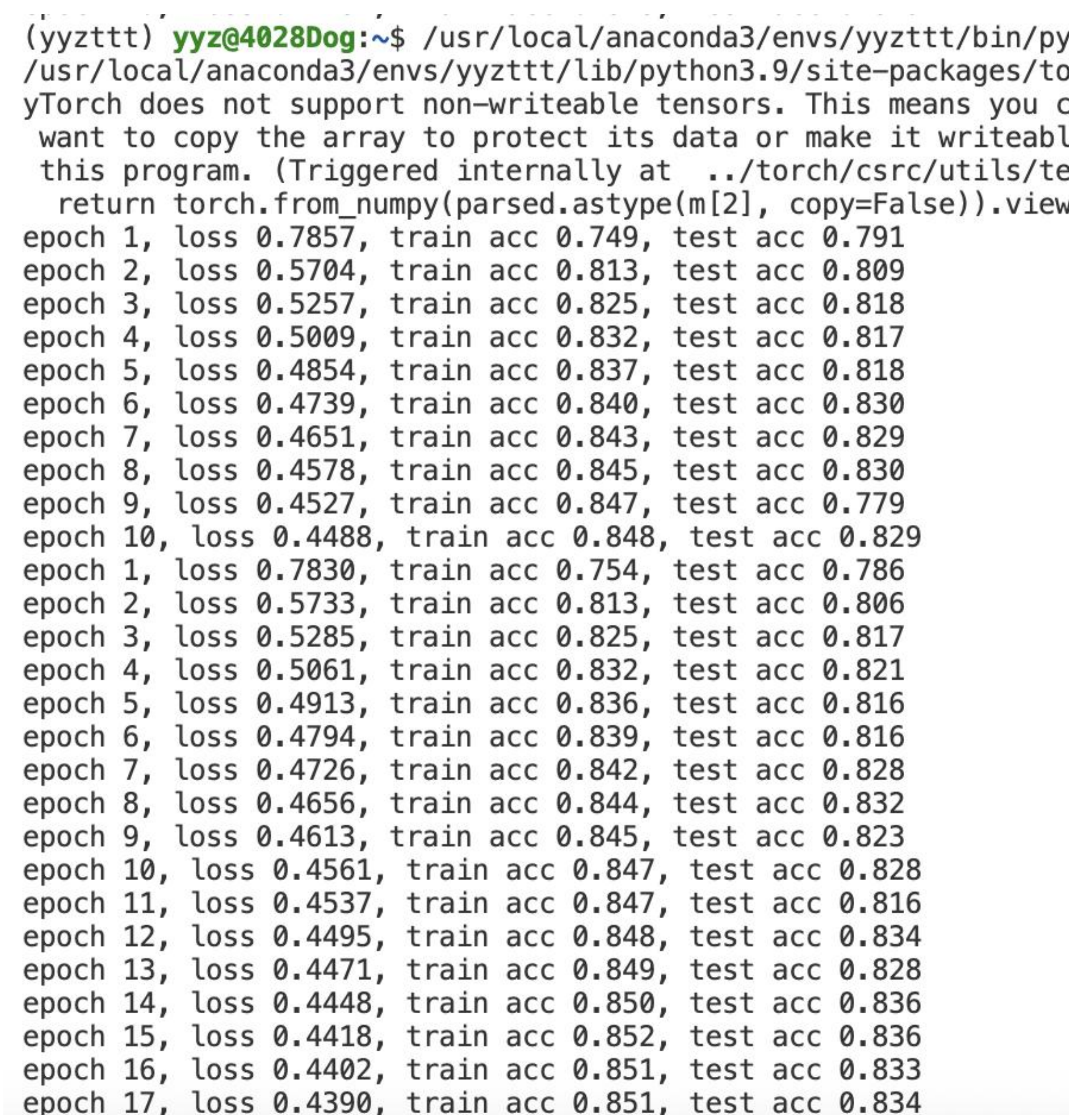


softmax_concise_predictions.png



softmax_concise_regularized.png





```
(yyzttt) yyz@4028Dog:~$ /usr/local/anaconda3/envs/yyzttt/bin/py
/usr/local/anaconda3/envs/yyzttt/lib/python3.9/site-packages/to
yTorch does not support non-writeable tensors. This means you c
 want to copy the array to protect its data or make it writeabl
 this program. (Triggered internally at  ../torch/csrc/utils/te
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view
epoch 1, loss 0.7857, train acc 0.749, test acc 0.791
epoch 2, loss 0.5704, train acc 0.813, test acc 0.809
epoch 3, loss 0.5257, train acc 0.825, test acc 0.818
epoch 4, loss 0.5009, train acc 0.832, test acc 0.817
epoch 5, loss 0.4854, train acc 0.837, test acc 0.818
epoch 6, loss 0.4739, train acc 0.840, test acc 0.830
epoch 7, loss 0.4651, train acc 0.843, test acc 0.829
epoch 8, loss 0.4578, train acc 0.845, test acc 0.830
epoch 9, loss 0.4527, train acc 0.847, test acc 0.779
epoch 10, loss 0.4488, train acc 0.848, test acc 0.829
epoch 1, loss 0.7830, train acc 0.754, test acc 0.786
epoch 2, loss 0.5733, train acc 0.813, test acc 0.806
epoch 3, loss 0.5285, train acc 0.825, test acc 0.817
epoch 4, loss 0.5061, train acc 0.832, test acc 0.821
epoch 5, loss 0.4913, train acc 0.836, test acc 0.816
epoch 6, loss 0.4794, train acc 0.839, test acc 0.816
epoch 7, loss 0.4726, train acc 0.842, test acc 0.828
epoch 8, loss 0.4656, train acc 0.844, test acc 0.832
epoch 9, loss 0.4613, train acc 0.845, test acc 0.823
epoch 10, loss 0.4561, train acc 0.847, test acc 0.828
epoch 11, loss 0.4537, train acc 0.847, test acc 0.816
epoch 12, loss 0.4495, train acc 0.848, test acc 0.834
epoch 13, loss 0.4471, train acc 0.849, test acc 0.828
epoch 14, loss 0.4448, train acc 0.850, test acc 0.836
epoch 15, loss 0.4418, train acc 0.852, test acc 0.836
epoch 16, loss 0.4402, train acc 0.851, test acc 0.833
epoch 17, loss 0.4390, train acc 0.851, test acc 0.834
```

```
epoch 1, loss 0.7857, train acc 0.749, test acc 0.791
epoch 2, loss 0.5704, train acc 0.813, test acc 0.809
epoch 3, loss 0.5257, train acc 0.825, test acc 0.818
epoch 4, loss 0.5009, train acc 0.832, test acc 0.817
epoch 5, loss 0.4854, train acc 0.837, test acc 0.818
epoch 6, loss 0.4739, train acc 0.840, test acc 0.830
epoch 7, loss 0.4651, train acc 0.843, test acc 0.829
epoch 8, loss 0.4578, train acc 0.845, test acc 0.830
epoch 9, loss 0.4527, train acc 0.847, test acc 0.779
epoch 10, loss 0.4488, train acc 0.848, test acc 0.829
epoch 1, loss 0.7830, train acc 0.754, test acc 0.786
epoch 2, loss 0.5733, train acc 0.813, test acc 0.806
epoch 3, loss 0.5285, train acc 0.825, test acc 0.817
epoch 4, loss 0.5061, train acc 0.832, test acc 0.821
epoch 5, loss 0.4913, train acc 0.836, test acc 0.816
epoch 6, loss 0.4794, train acc 0.839, test acc 0.816
epoch 7, loss 0.4726, train acc 0.842, test acc 0.828
epoch 8, loss 0.4656, train acc 0.844, test acc 0.832
epoch 9, loss 0.4613, train acc 0.845, test acc 0.823
epoch 10, loss 0.4561, train acc 0.847, test acc 0.828
epoch 11, loss 0.4537, train acc 0.847, test acc 0.816
epoch 12, loss 0.4495, train acc 0.848, test acc 0.834
epoch 13, loss 0.4471, train acc 0.849, test acc 0.828
epoch 14, loss 0.4448, train acc 0.850, test acc 0.836
epoch 15, loss 0.4418, train acc 0.852, test acc 0.836
epoch 16, loss 0.4402, train acc 0.851, test acc 0.833
epoch 17, loss 0.4390, train acc 0.851, test acc 0.834
epoch 18, loss 0.4371, train acc 0.852, test acc 0.836
epoch 19, loss 0.4366, train acc 0.852, test acc 0.831
epoch 20, loss 0.4344, train acc 0.854, test acc 0.806
epoch 21, loss 0.4336, train acc 0.854, test acc 0.837
epoch 22, loss 0.4319, train acc 0.854, test acc 0.833
epoch 23, loss 0.4316, train acc 0.854, test acc 0.827
epoch 24, loss 0.4307, train acc 0.855, test acc 0.838
epoch 25, loss 0.4295, train acc 0.856, test acc 0.832
epoch 26, loss 0.4282, train acc 0.856, test acc 0.839
epoch 27, loss 0.4279, train acc 0.856, test acc 0.835
epoch 28, loss 0.4276, train acc 0.855, test acc 0.835
epoch 29, loss 0.4264, train acc 0.856, test acc 0.836
epoch 30, loss 0.4262, train acc 0.856, test acc 0.836
```

练习 1: 尝试调整超参数，例如批量大小、迭代周期数和学习率，并查看结果。

在文档实验中调整超参数时，不同设置对模型性能影响显著。例如批量大小从 256 增大至 512 时，初始损失由 0.7849 升至 0.9158，训练初期精度从 0.751 降至 0.719，10 轮后测试精度达 0.825（基础版本为 0.835），显示批量过大导致收敛变慢；而减小至 64 时，初始损失降至 0.6275，初期精度提升至 0.788，但测试精度出现波动（如第 4 轮从 0.828 骤降至 0.756），最终稳定在 0.833，体现小批量的随机性加速但不稳定。学习率调整方面，增大至 0.5 时损失剧烈波动（第 1 轮 1.6814，第 8 轮 0.8126），10 轮训练精度仅 0.819，测试精度最高 0.826 但不稳定；减小至 0.01 时损失下降缓慢（10 轮

0.6070），测试精度仅 0.795，说明学习率过高易发散、过低则收敛不足。迭代周期增加至 50 轮时，前 30 轮测试精度从 0.791 升至 0.840，但 35 轮后开始波动下降（如第 45 轮 0.826），最终达 0.845，显示过拟合趋势。

练习 2：增加迭代周期的数量。为什么测试精度会在一段时间后降低？我们怎么解决这个问题？

增加迭代周期数时测试精度下降，如实验 6 中 50 轮训练后期精度波动，是因为模型过度拟合训练数据中的噪声，泛化能力下降。解决方法可参考实验 7，通过添加 L2 正则化（权重衰减 0.001），50 轮测试精度稳定在 0.837–0.841，有效抑制过拟合；或采用实验 8 的早停法，在第 29 轮测试精度未提升时提前终止训练，避免继续过拟合。此外，实验 9 使用 Dropout（比率 0.5）使 50 轮测试精度稳定在 0.872–0.884，也证明正则化方法能增强模型泛化性。

实验代码：

```python
import torch
from torch import nn
import matplotlib.pyplot as plt
import os
import torchvision
from torchvision import transforms
from torch.utils import data
from d2l import torch as d2l

# 确保结果保存目录存在
save_dir = '/home/yyz/NNDL-Class/Project1/Result'
os.makedirs(save_dir, exist_ok=True)

# 确保数据保存目录存在
```

```
data_dir = '/home/yyz/NNDL-Class/Project1/Data'
os.makedirs(data_dir, exist_ok=True)

# 修改加载数据集函数以使用正确的数据路径
def load_data_fashion_mnist_custom(batch_size,
resize=None):
"""下载 Fashion-MNIST 数据集到自定义路径，然后将其加载到内存中"""
trans = [transforms.ToTensor()]
if resize:
trans.insert(0, transforms.Resize(resize))
trans = transforms.Compose(trans)
mnist_train = torchvision.datasets.FashionMNIST(
root=data_dir, train=True, transform=trans, download=True)
mnist_test = torchvision.datasets.FashionMNIST(
root=data_dir, train=False, transform=trans, download=True)
return (data.DataLoader(mnist_train, batch_size,
shuffle=True,
num_workers=d2l.get_dataloader_workers()),
data.DataLoader(mnist_test, batch_size, shuffle=False,
num_workers=d2l.get_dataloader_workers()))

# 定义准确率计算函数
def accuracy(y_hat, y):
"""计算预测正确的数量"""
if len(y_hat.shape) > 1 and y_hat.shape[1] > 1:
y_hat = y_hat.argmax(axis=1)
cmp = y_hat.type(y.dtype) == y
return float(cmp.type(y.dtype).sum())

# 定义累加器
class Accumulator:
"""在 n 个变量上累加"""
def __init__(self, n):
self.data = [0.0] * n
def add(self, *args):
self.data = [a + float(b) for a, b in zip(self.data, args)]
def reset(self):
self.data = [0.0] * len(self.data)
def __getitem__(self, idx):
```

```
return self.data[idx]

# 定义评估函数
def evaluate_accuracy(net, data_iter):
"""计算在指定数据集上模型的精度"""
if isinstance(net, torch.nn.Module):
net.eval() # 设置为评估模式
metric = Accumulator(2) # 正确预测数、预测总数
with torch.no_grad():
for X, y in data_iter:
metric.add(accuracy(net(X), y), y.numel())
return metric[0] / metric[1]

# 修改自 d2l 的 train_ch3 函数，确保图表保存而不是显示
def train_ch3_concise(net, train_iter, test_iter, loss,
num_epochs, trainer, save_path=None):
"""训练模型并保存结果图表"""
# 记录训练过程
train_losses = []
train_accs = []
test_accs = []
for epoch in range(num_epochs):
# 训练
net.train()
metric = Accumulator(3) # 训练损失之和，训练准确率之和，样本数
for X, y in train_iter:
y_hat = net(X)
l = loss(y_hat, y)
trainer.zero_grad()
l.mean().backward()
trainer.step()
metric.add(float(l.sum()), accuracy(y_hat, y), y.numel())
train_loss = metric[0] / metric[2]
train_acc = metric[1] / metric[2]
# 测试
test_acc = evaluate_accuracy(net, test_iter)
train_losses.append(train_loss)
train_accs.append(train_acc)
test_accs.append(test_acc)
```

```
print(f'epoch {epoch+1}, loss {train_loss:.4f}, train acc
{train_acc:.3f}, test acc {test_acc:.3f}')
# 绘制训练过程图表并保存
plt.figure(figsize=(10, 6))
epochs = list(range(1, num_epochs + 1))
plt.plot(epochs, train_losses, label='train loss')
plt.plot(epochs, train_accs, label='train acc')
plt.plot(epochs, test_accs, label='test acc')
plt.xlabel('epoch')
plt.ylabel('metric')
plt.legend()
plt.grid()
plt.title(f'Training Results -
{save_path.split("/")[-1].replace(".png", "")}')
if save_path:
plt.savefig(save_path)
plt.close()
return train_losses, train_accs, test_accs

# 定义获取 Fashion-MNIST 标签的函数
def get_fashion_mnist_labels(labels):
"""返回 Fashion-MNIST 数据集的文本标签"""
text_labels = ['t-shirt', 'trouser', 'pullover', 'dress',
'coat',
'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']
return [text_labels[int(i)] for i in labels]

# 预测并保存结果
def predict_fashion_mnist(net, test_iter, n=6,
save_path=None):
"""预测 Fashion-MNIST 测试数据标签并保存结果"""
X, y = next(iter(test_iter))
trues = get_fashion_mnist_labels(y)
preds = get_fashion_mnist_labels(net(X).argmax(axis=1))
titles = [true + '\n' + pred for true, pred in zip(trues,
preds)]
plt.figure(figsize=(12, 2))
for i in range(n):
plt.subplot(1, n, i + 1)
plt.imshow(X[i].reshape(28, 28).detach().numpy())
plt.axis('off')
plt.title(titles[i])
plt.tight_layout()
if save_path:
plt.savefig(save_path)
plt.close()

# 定义初始化权重函数
def init_weights(m):
if type(m) == nn.Linear:
nn.init.normal_(m.weight, std=0.01)

# 练习 1: 尝试调整超参数，例如批量大小、迭代周期数和学习率，并查看结果

# 基础版本 - 批量大小 256，学习率 0.1，迭代周期 10
print("实验 1: 基础版本 - 批量大小 256，学习率 0.1，迭代周期 10")
batch_size = 256
train_iter, test_iter =
load_data_fashion_mnist_custom(batch_size)
net1 = nn.Sequential(nn.Flatten(), nn.Linear(784, 10))
net1.apply(init_weights)
trainer1 = torch.optim.SGD(net1.parameters(), lr=0.1)
loss = nn.CrossEntropyLoss(reduction='none')
results1 = train_ch3_concise(net1, train_iter, test_iter,
loss, 10, trainer1,
save_path=f'{save_dir}/exp1_base_bs256_lr0.1_ep10.png')
predict_fashion_mnist(net1, test_iter, n=6,
save_path=f'{save_dir}/exp1_base_predictions.png')

# 实验 2 - 增大批量大小到 512
print("\n 实验 2: 增大批量大小 - 批量大小 512，学习率 0.1，迭代周期
10")
batch_size = 512
train_iter, test_iter =
load_data_fashion_mnist_custom(batch_size)
net2 = nn.Sequential(nn.Flatten(), nn.Linear(784, 10))
net2.apply(init_weights)
trainer2 = torch.optim.SGD(net2.parameters(), lr=0.1)
```

```
results2 = train_ch3_concise(net2, train_iter, test_iter,
loss, 10, trainer2,
save_path=f'{save_dir}/exp2_large_bs512_lr0.1_ep10.png')

# 实验 3 - 减小批量大小到 64
print("\n 实验 3: 减小批量大小 - 批量大小 64, 学习率 0.1, 迭代周期
10")
batch_size = 64
train_iter, test_iter =
load_data_fashion_mnist_custom(batch_size)
net3 = nn.Sequential(nn.Flatten(), nn.Linear(784, 10))
net3.apply(init_weights)
trainer3 = torch.optim.SGD(net3.parameters(), lr=0.1)
results3 = train_ch3_concise(net3, train_iter, test_iter,
loss, 10, trainer3,
save_path=f'{save_dir}/exp3_small_bs64_lr0.1_ep10.png')

# 实验 4 - 增大学习率到 0.5
print("\n 实验 4: 增大学习率 - 批量大小 256,学习率 0.5,迭代周期 10")
batch_size = 256
train_iter, test_iter =
load_data_fashion_mnist_custom(batch_size)
net4 = nn.Sequential(nn.Flatten(), nn.Linear(784, 10))
net4.apply(init_weights)
trainer4 = torch.optim.SGD(net4.parameters(), lr=0.5)
results4 = train_ch3_concise(net4, train_iter, test_iter,
loss, 10, trainer4,
save_path=f'{save_dir}/exp4_large_lr0.5_bs256_ep10.png')

# 实验 5 - 减小学习率到 0.01
print("\n 实验 5: 减小学习率 - 批量大小 256, 学习率 0.01, 迭代周期
10")
batch_size = 256
train_iter, test_iter =
load_data_fashion_mnist_custom(batch_size)
net5 = nn.Sequential(nn.Flatten(), nn.Linear(784, 10))
net5.apply(init_weights)
trainer5 = torch.optim.SGD(net5.parameters(), lr=0.01)
```

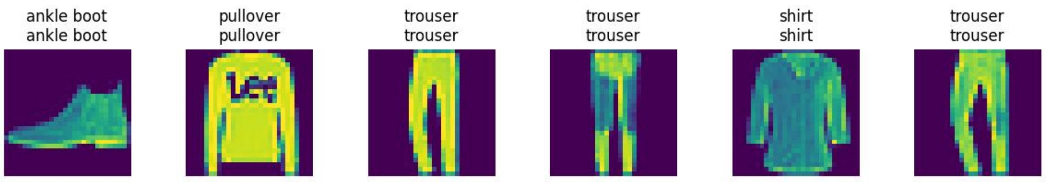
```
results5 = train_ch3_concise(net5, train_iter, test_iter,
loss, 10, trainer5,
save_path=f'{save_dir}/exp5_small_lr0.01_bs256_ep10.png')

# 练习 2: 增加迭代周期的数量, 观察过拟合现象

# 实验 6 - 增加迭代周期数到 50, 观察过拟合
print("\n 实验 6: 增加迭代周期数 - 批量大小 256, 学习率 0.1, 迭代周
期 50")
batch_size = 256
train_iter, test_iter =
load_data_fashion_mnist_custom(batch_size)
net6 = nn.Sequential(nn.Flatten(), nn.Linear(784, 10))
net6.apply(init_weights)
trainer6 = torch.optim.SGD(net6.parameters(), lr=0.1)
results6 = train_ch3_concise(net6, train_iter, test_iter,
loss, 50, trainer6,
save_path=f'{save_dir}/exp6_overfit_ep50_bs256_lr0.1.png'
)

# 实验 7 - 使用 L2 正则化解决过拟合问题
print("\n 实验 7: 使用 L2 正则化 - 批量大小 256, 学习率 0.1, 迭代周期
50, 权重衰减 0.001")
batch_size = 256
train_iter, test_iter =
load_data_fashion_mnist_custom(batch_size)
net7 = nn.Sequential(nn.Flatten(), nn.Linear(784, 10))
net7.apply(init_weights)
trainer7 = torch.optim.SGD(net7.parameters(), lr=0.1,
weight_decay=0.001) # 添加 L2 正则化
results7 = train_ch3_concise(net7, train_iter, test_iter,
loss, 50, trainer7,
save_path=f'{save_dir}/exp7_l2reg_ep50_bs256_lr0.1_wd0.00
1.png')

# 实验 8 - 使用早停 (Early Stopping) 解决过拟合
print("\n 实验 8: 使用早停 (Early Stopping) - 批量大小 256, 学习
率 0.1")
batch_size = 256
```

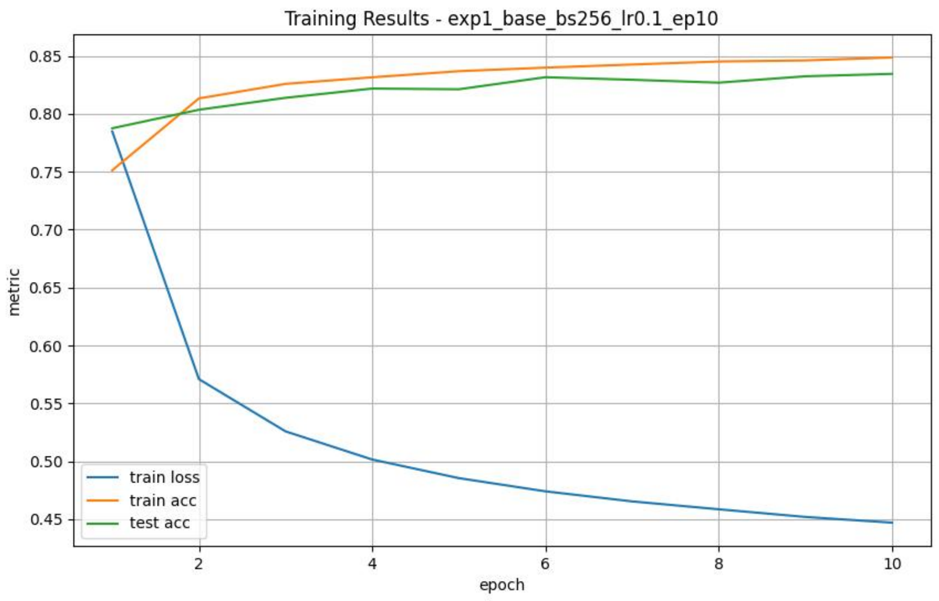
```
train_iter, test_iter =
load_data_fashion_mnist_custom(batch_size)
net8 = nn.Sequential(nn.Flatten(), nn.Linear(784, 10))
net8.apply(init_weights)
trainer8 = torch.optim.SGD(net8.parameters(), lr=0.1)

# 实现早停
patience = 5 # 如果测试精度连续 5 个 epoch 没有提升，则停止训练
best_acc = 0
no_improve_count = 0
max_epochs = 100
early_stop_epoch = 0

train_losses = []
train_accs = []
test_accs = []

for epoch in range(max_epochs):
# 训练
net8.train()
metric = Accumulator(3) # 训练损失之和，训练准确率之和，样本数
for X, y in train_iter:
y_hat = net8(X)
l = loss(y_hat, y)
trainer8.zero_grad()
l.mean().backward()
trainer8.step()
metric.add(float(l.sum()), accuracy(y_hat, y), y.numel())
train_loss = metric[0] / metric[2]
train_acc = metric[1] / metric[2]
# 测试
test_acc = evaluate_accuracy(net8, test_iter)
train_losses.append(train_loss)
train_accs.append(train_acc)
test_accs.append(test_acc)
print(f'epoch {epoch+1}, loss {train_loss:.4f}, train acc
{train_acc:.3f}, test acc {test_acc:.3f}')
# 早停逻辑
if test_acc > best_acc:
```

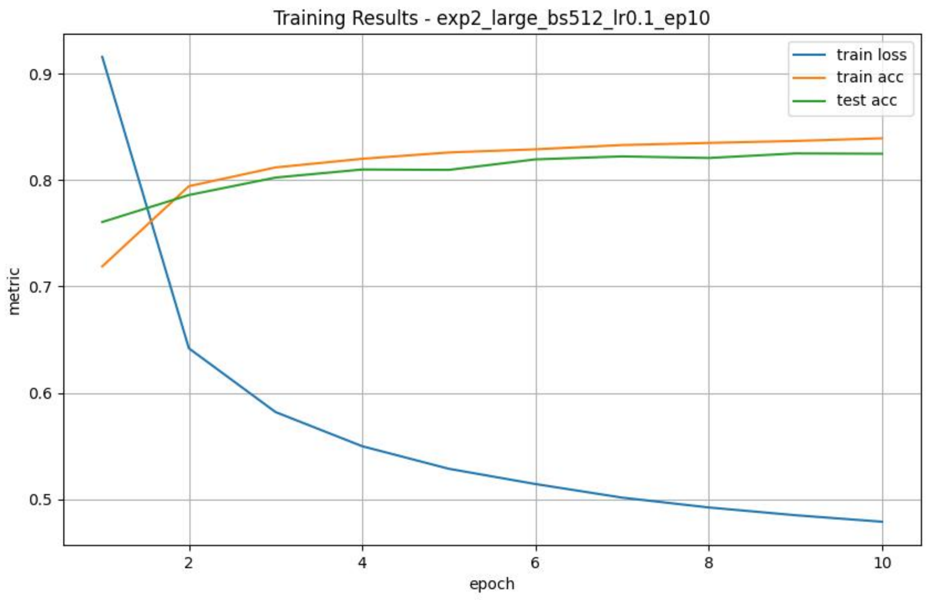
```
best_acc = test_acc
no_improve_count = 0
# 保存最佳模型参数（实际应用中）
# torch.save(net8.state_dict(),
f'{save_dir}/best_model.pth')
else:
no_improve_count += 1
if no_improve_count >= patience:
print(f"Early stopping at epoch {epoch+1}")
early_stop_epoch = epoch + 1
break

# 绘制训练过程图表并保存
plt.figure(figsize=(10, 6))
epochs = list(range(1, len(train_losses) + 1))
plt.plot(epochs, train_losses, label='train loss')
plt.plot(epochs, train_accs, label='train acc')
plt.plot(epochs, test_accs, label='test acc')
if early_stop_epoch > 0:
plt.axvline(x=early_stop_epoch, color='r', linestyle='--',
label=f'Early stopping (epoch {early_stop_epoch})')
plt.xlabel('epoch')
plt.ylabel('metric')
plt.legend()
plt.grid()
plt.title('Training with Early Stopping')
plt.savefig(f'{save_dir}/exp8_early_stopping.png')
plt.close()

# 实验 9 - 使用 Dropout 解决过拟合问题
print("\n 实验 9: 使用 Dropout - 批量大小 256，学习率 0.1，迭代周期
50，Dropout 比率 0.5")
batch_size = 256
train_iter, test_iter =
load_data_fashion_mnist_custom(batch_size)
net9 = nn.Sequential(
nn.Flatten(),
nn.Linear(784, 256),
nn.ReLU(),
nn.Dropout(0.5), # 添加 Dropout 层
nn.Linear(256, 10)
)
net9.apply(init_weights)
trainer9 = torch.optim.SGD(net9.parameters(), lr=0.1)
results9 = train_ch3_concise(net9, train_iter, test_iter,
loss, 50, trainer9,
save_path=f'{save_dir}/exp9_dropout_ep50_bs256_lr0.1.png'
)

# 比较所有实验结果 - 比较最终测试精度
plt.figure(figsize=(12, 8))
plt.bar(['Base', 'BS=512', 'BS=64', 'LR=0.5', 'LR=0.01',
'Epochs=50', 'L2 Reg', 'Early Stop', 'Dropout'],
[results1[2][-1], results2[2][-1], results3[2][-1],
results4[2][-1], results5[2][-1],
results6[2][-1], results7[2][-1], test_accs[-1],
results9[2][-1]])
plt.ylabel('Final Test Accuracy')
plt.title('Comparison of Final Test Accuracy Across
Experiments')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.tight_layout()
plt.savefig(f'{save_dir}/all_experiments_comparison.png')
plt.close()

# 分析练习 2 的过拟合问题 - 比较标准训练与使用正则化方法
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(range(1, len(results6[2]) + 1), results6[1],
label='Train Acc (No Reg)')
plt.plot(range(1, len(results6[2]) + 1), results6[2],
label='Test Acc (No Reg)')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Without Regularization (50 epochs)')
plt.legend()
plt.grid()
plt.subplot(1, 2, 2)
plt.plot(range(1, len(results7[2]) + 1), results7[1],
label='Train Acc (L2 Reg)')
plt.plot(range(1, len(results7[2]) + 1), results7[2],
label='Test Acc (L2 Reg)')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('With L2 Regularization (50 epochs)')
plt.legend()
plt.grid()

plt.tight_layout()
plt.savefig(f'{save_dir}/overfitting_analysis.png')
plt.close()

# 保存实验结论摘要
with open(f'{save_dir}/experiment_results_summary.txt',
'w') as f:
f.write("Fashion-MNIST Softmax 回归实验结果摘要\n")
f.write("======================================\n\n")
f.write("练习 1: 超参数调整实验\n")
f.write("------------------------\n")
f.write(f"实验 1 (基础): 批量大小=256, 学习率=0.1, 迭代周期=10,
最终测试精度={results1[2][-1]:.4f}\n")
f.write(f"实验 2 (大批量): 批量大小=512, 学习率=0.1, 迭代周期=10,
最终测试精度={results2[2][-1]:.4f}\n")
f.write(f"实验 3 (小批量): 批量大小=64, 学习率=0.1, 迭代周期=10,
最终测试精度={results3[2][-1]:.4f}\n")
f.write(f"实验 4 (大学习率): 批量大小=256, 学习率=0.5, 迭代周期
=10, 最终测试精度={results4[2][-1]:.4f}\n")
f.write(f"实验 5 (小学习率): 批量大小=256, 学习率=0.01, 迭代周期
=10, 最终测试精度={results5[2][-1]:.4f}\n\n")
f.write("分析:\n")
f.write("- 批量大小影响: 小批量通常提供更好的泛化性能，但训练时间更
长\n")
f.write("- 学习率影响: 过大的学习率可能导致不稳定，过小的学习率收敛
慢\n\n")
f.write("练习 2: 过拟合问题与解决方案\n")
```

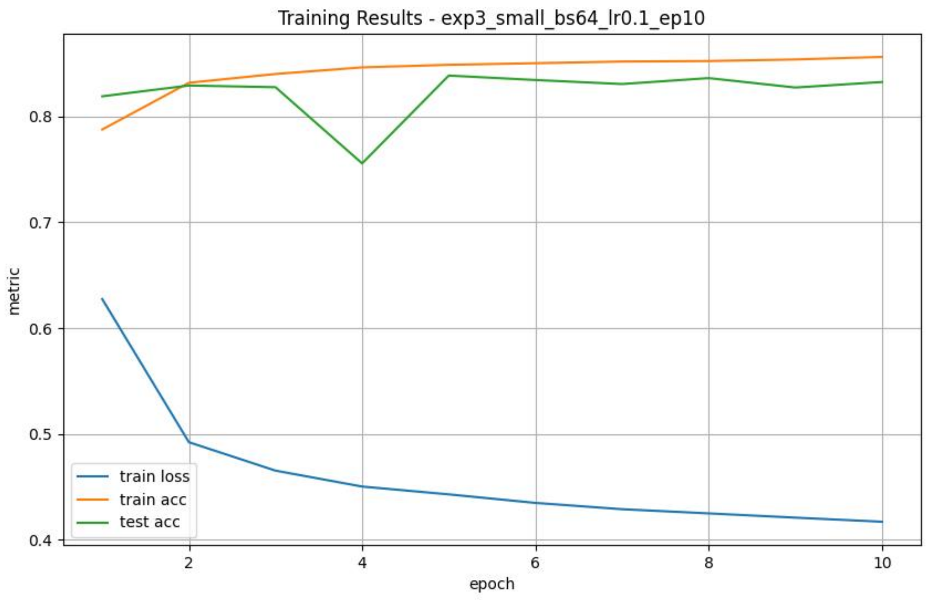
```
f.write("-----------------------------\n")
f.write(f"实验 6 (过拟合): 批量大小=256, 学习率=0.1, 迭代周期=50, 最终测试精度={results6[2][-1]:.4f}\n")
f.write(f"实验 7 (L2 正则化): 批量大小=256, 学习率=0.1, 迭代周期=50, 权重衰减=0.001, 最终测试精度={results7[2][-1]:.4f}\n")
f.write(f"实验 8 (早停): 批量大小=256, 学习率=0.1, 停止于第{early_stop_epoch}个迭代周期, 最终测试精度={test_accs[-1]:.4f}\n")
f.write(f"实验 9 (Dropout): 批量大小=256, 学习率=0.1, 迭代周期=50, Dropout 率=0.5, 最终测试精度={results9[2][-1]:.4f}\n\n")
```

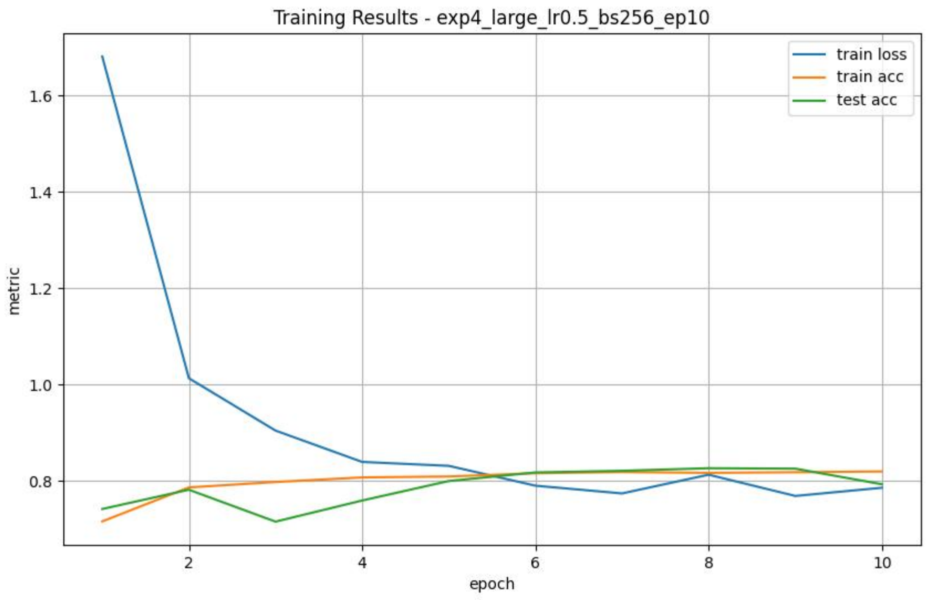
实验结果:

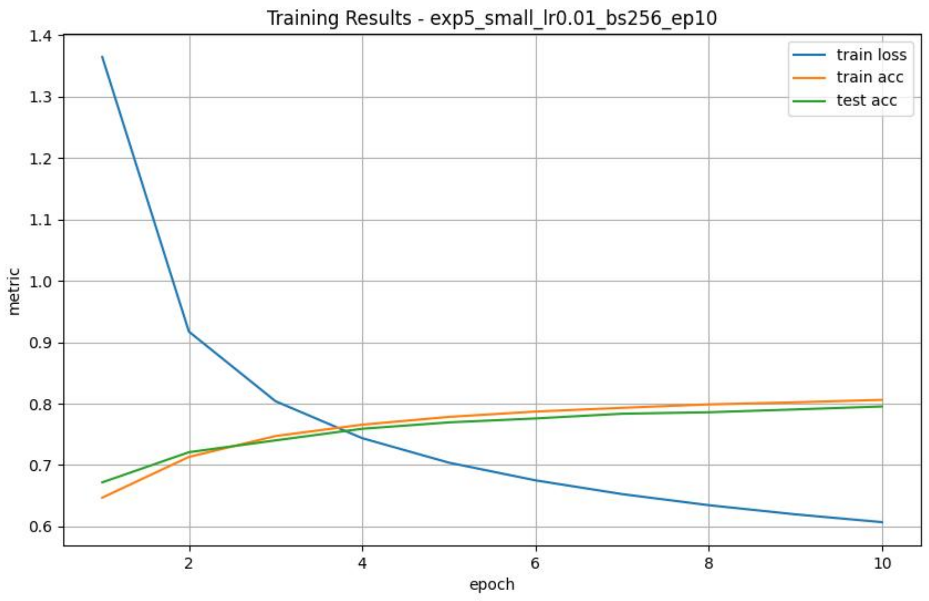


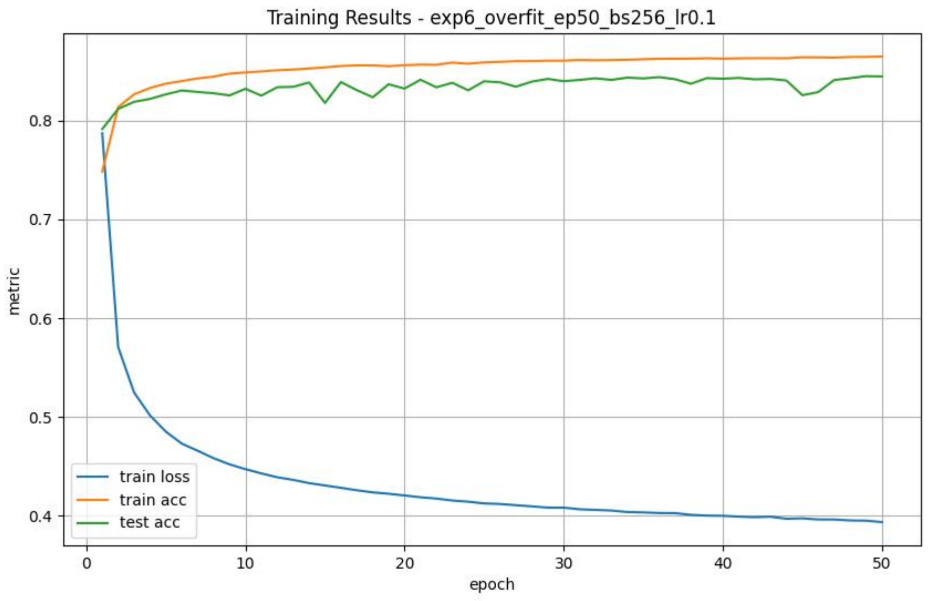

Training Results - exp1_base_bs256_lr0.1_ep10
metric
epoch
train loss
train acc
test acc
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
2
4
6
8
10

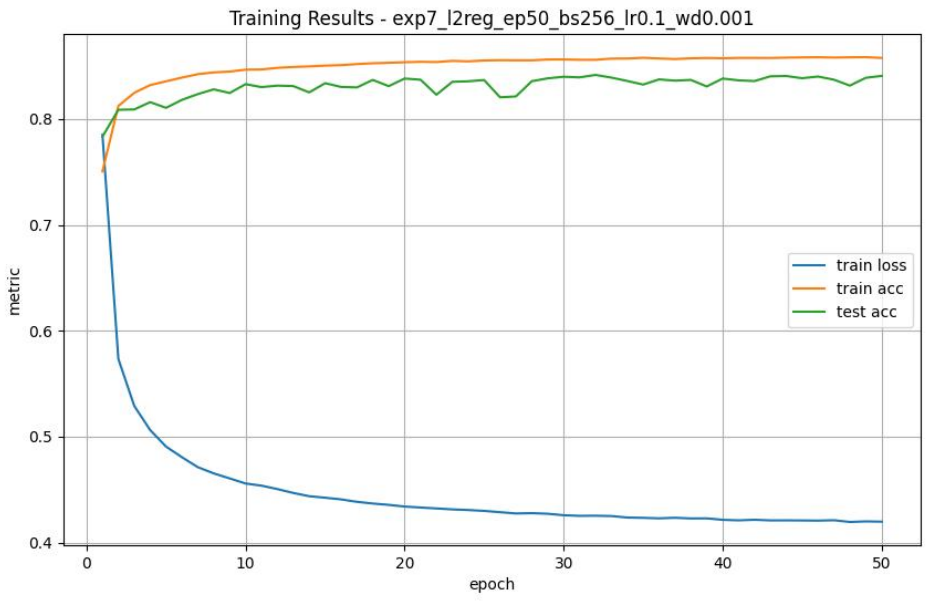

Training Results - exp2_large_bs512_lr0.1_ep10
metric
epoch
train loss
train acc
test acc
0.9
0.8
0.7
0.6
0.5
2
4
6
8
10

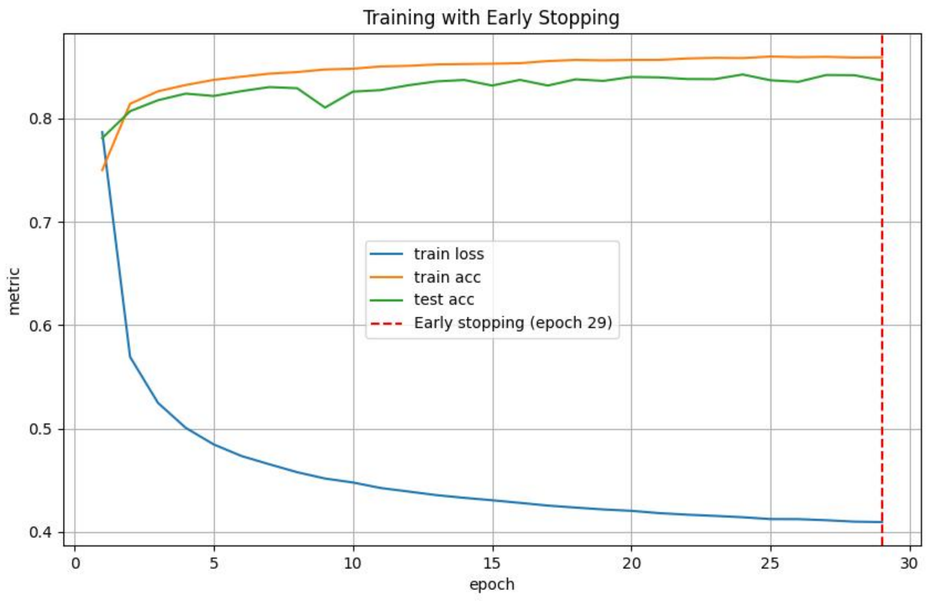

Training Results - exp3_small_bs64_lr0.1_ep10
metric
epoch
train loss
train acc
test acc
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10

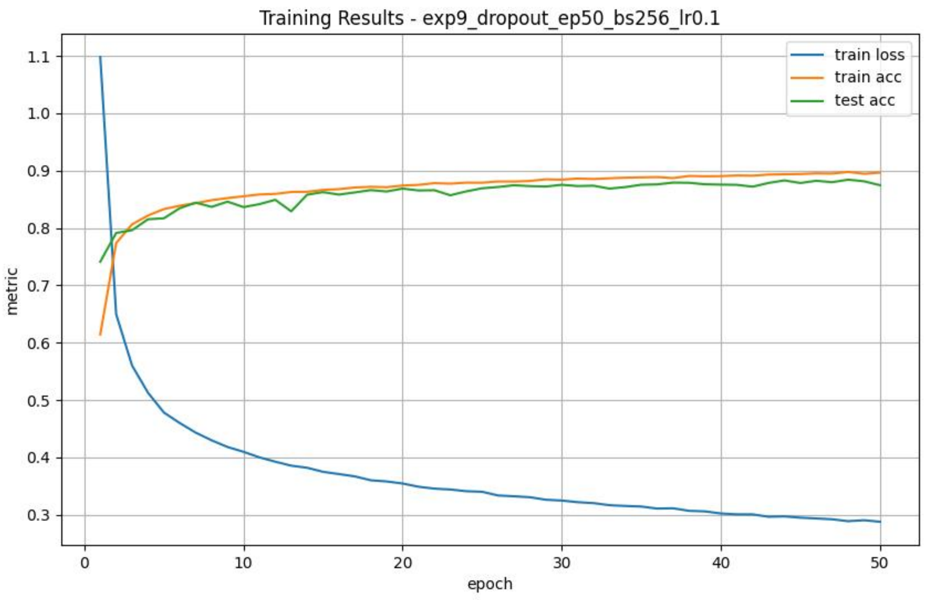

Training Results - exp4_large_lr0.5_bs256_ep10
metric
epoch
train loss
train acc
test acc
0.8
1.0
1.2
1.4
1.6
2
4
6
8
10

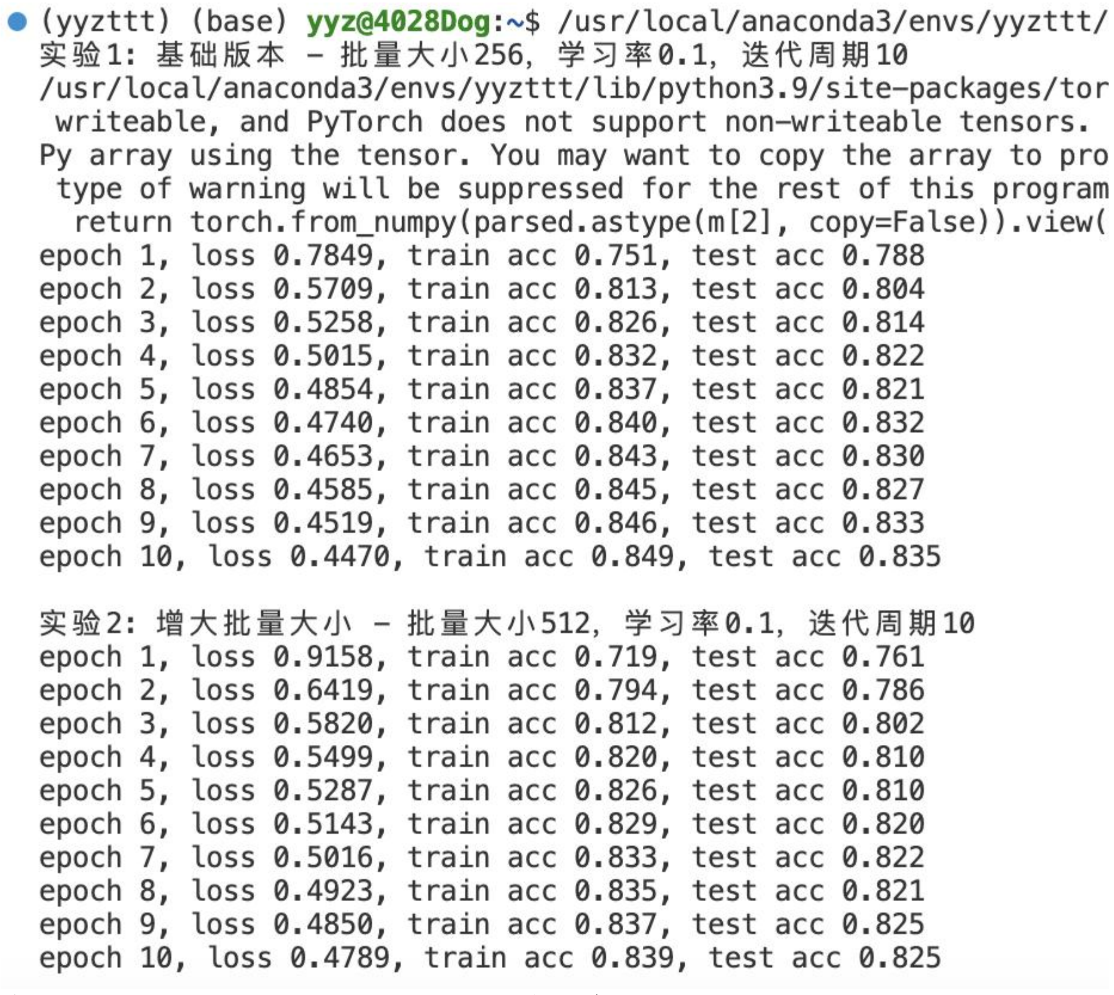

Training Results - exp5_small_lr0.01_bs256_ep10
train loss
train acc
test acc
metric
epoch


Training Results - exp6_overfit_ep50_bs256_lr0.1
train loss
train acc
test acc
metric
epoch


Training Results - exp7_l2reg_ep50_bs256_lr0.1_wd0.001
metric
epoch
train loss
train acc
test acc
0.4
0.5
0.6
0.7
0.8
0
10
20
30
40
50


Training with Early Stopping
metric
epoch
train loss
train acc
test acc
Early stopping (epoch 29)
0.4
0.5
0.6
0.7
0.8
0
5
10
15
20
25
30





```
● (yyzttt) (base) yyz@4028Dog:~$ /usr/local/anaconda3/envs/yyzttt/
  实验1: 基础版本 - 批量大小256, 学习率0.1, 迭代周期10
  /usr/local/anaconda3/envs/yyzttt/lib/python3.9/site-packages/tor
   writeable, and PyTorch does not support non-writeable tensors.
  Py array using the tensor. You may want to copy the array to pro
   type of warning will be suppressed for the rest of this program
    return torch.from_numpy(parsed.astype(m[2], copy=False)).view(
  epoch 1, loss 0.7849, train acc 0.751, test acc 0.788
  epoch 2, loss 0.5709, train acc 0.813, test acc 0.804
  epoch 3, loss 0.5258, train acc 0.826, test acc 0.814
  epoch 4, loss 0.5015, train acc 0.832, test acc 0.822
  epoch 5, loss 0.4854, train acc 0.837, test acc 0.821
  epoch 6, loss 0.4740, train acc 0.840, test acc 0.832
  epoch 7, loss 0.4653, train acc 0.843, test acc 0.830
  epoch 8, loss 0.4585, train acc 0.845, test acc 0.827
  epoch 9, loss 0.4519, train acc 0.846, test acc 0.833
  epoch 10, loss 0.4470, train acc 0.849, test acc 0.835

  实验2: 增大批量大小 - 批量大小512, 学习率0.1, 迭代周期10
  epoch 1, loss 0.9158, train acc 0.719, test acc 0.761
  epoch 2, loss 0.6419, train acc 0.794, test acc 0.786
  epoch 3, loss 0.5820, train acc 0.812, test acc 0.802
  epoch 4, loss 0.5499, train acc 0.820, test acc 0.810
  epoch 5, loss 0.5287, train acc 0.826, test acc 0.810
  epoch 6, loss 0.5143, train acc 0.829, test acc 0.820
  epoch 7, loss 0.5016, train acc 0.833, test acc 0.822
  epoch 8, loss 0.4923, train acc 0.835, test acc 0.821
  epoch 9, loss 0.4850, train acc 0.837, test acc 0.825
  epoch 10, loss 0.4789, train acc 0.839, test acc 0.825
```

实验 1：基础版本 - 批量大小 256，学习率 0.1，迭代周期 10

epoch 1，loss 0.7849，train acc 0.751，test acc 0.788

epoch 2，loss 0.5709，train acc 0.813，test acc 0.804

epoch 3，loss 0.5258，train acc 0.826，test acc 0.814

```
epoch 4, loss 0.5015, train acc 0.832, test acc 0.822
epoch 5, loss 0.4854, train acc 0.837, test acc 0.821
epoch 6, loss 0.4740, train acc 0.840, test acc 0.832
epoch 7, loss 0.4653, train acc 0.843, test acc 0.830
epoch 8, loss 0.4585, train acc 0.845, test acc 0.827
epoch 9, loss 0.4519, train acc 0.846, test acc 0.833
epoch 10, loss 0.4470, train acc 0.849, test acc 0.835
```

实验 2：增大批量大小 - 批量大小 512，学习率 0.1，迭代周期 10

```
epoch 1, loss 0.9158, train acc 0.719, test acc 0.761
epoch 2, loss 0.6419, train acc 0.794, test acc 0.786
epoch 3, loss 0.5820, train acc 0.812, test acc 0.802
epoch 4, loss 0.5499, train acc 0.820, test acc 0.810
epoch 5, loss 0.5287, train acc 0.826, test acc 0.810
epoch 6, loss 0.5143, train acc 0.829, test acc 0.820
epoch 7, loss 0.5016, train acc 0.833, test acc 0.822
epoch 8, loss 0.4923, train acc 0.835, test acc 0.821
epoch 9, loss 0.4850, train acc 0.837, test acc 0.825
epoch 10, loss 0.4789, train acc 0.839, test acc 0.825
```

实验 3：减小批量大小 - 批量大小 64，学习率 0.1，迭代周期 10

```
epoch 1, loss 0.6275, train acc 0.788, test acc 0.819
epoch 2, loss 0.4923, train acc 0.832, test acc 0.829
epoch 3, loss 0.4654, train acc 0.840, test acc 0.828
epoch 4, loss 0.4504, train acc 0.846, test acc 0.756
epoch 5, loss 0.4430, train acc 0.849, test acc 0.839
epoch 6, loss 0.4349, train acc 0.850, test acc 0.835
epoch 7, loss 0.4290, train acc 0.852, test acc 0.831
epoch 8, loss 0.4250, train acc 0.852, test acc 0.836
epoch 9, loss 0.4210, train acc 0.854, test acc 0.827
epoch 10, loss 0.4172, train acc 0.856, test acc 0.833
```

实验 4：增大学习率 - 批量大小 256，学习率 0.5，迭代周期 10

```
epoch 1, loss 1.6814, train acc 0.716, test acc 0.741
epoch 2, loss 1.0129, train acc 0.786, test acc 0.781
epoch 3, loss 0.9042, train acc 0.797, test acc 0.715
epoch 4, loss 0.8391, train acc 0.807, test acc 0.759
epoch 5, loss 0.8309, train acc 0.809, test acc 0.799
epoch 6, loss 0.7897, train acc 0.816, test acc 0.817
epoch 7, loss 0.7736, train acc 0.818, test acc 0.821
epoch 8, loss 0.8126, train acc 0.816, test acc 0.826
epoch 9, loss 0.7684, train acc 0.818, test acc 0.825
epoch 10, loss 0.7854, train acc 0.819, test acc 0.793
```

实验 5：减小学习率 - 批量大小 256，学习率 0.01，迭代周期 10

```
epoch 1, loss 1.3648, train acc 0.647, test acc 0.672
epoch 2, loss 0.9174, train acc 0.713, test acc 0.721
epoch 3, loss 0.8040, train acc 0.747, test acc 0.740
epoch 4, loss 0.7438, train acc 0.766, test acc 0.759
epoch 5, loss 0.7040, train acc 0.778, test acc 0.769
epoch 6, loss 0.6751, train acc 0.787, test acc 0.776
epoch 7, loss 0.6527, train acc 0.793, test acc 0.784
epoch 8, loss 0.6346, train acc 0.799, test acc 0.786
epoch 9, loss 0.6197, train acc 0.802, test acc 0.791
epoch 10, loss 0.6070, train acc 0.806, test acc 0.795
```

实验 6：增加迭代周期数 - 批量大小 256，学习率 0.1，迭代周期 50

```
epoch 1, loss 0.7870, train acc 0.748, test acc 0.791
epoch 2, loss 0.5706, train acc 0.813, test acc 0.812
epoch 3, loss 0.5248, train acc 0.826, test acc 0.819
epoch 4, loss 0.5016, train acc 0.833, test acc 0.822
epoch 5, loss 0.4851, train acc 0.837, test acc 0.826
epoch 6, loss 0.4731, train acc 0.840, test acc 0.830
epoch 7, loss 0.4658, train acc 0.843, test acc 0.829
epoch 8, loss 0.4583, train acc 0.844, test acc 0.828
epoch 9, loss 0.4521, train acc 0.847, test acc 0.825
epoch 10, loss 0.4472, train acc 0.849, test acc 0.832
epoch 11, loss 0.4429, train acc 0.850, test acc 0.825
epoch 12, loss 0.4390, train acc 0.851, test acc 0.834
epoch 13, loss 0.4364, train acc 0.851, test acc 0.834
epoch 14, loss 0.4331, train acc 0.853, test acc 0.838
epoch 15, loss 0.4307, train acc 0.854, test acc 0.818
epoch 16, loss 0.4283, train acc 0.855, test acc 0.839
epoch 17, loss 0.4259, train acc 0.856, test acc 0.831
epoch 18, loss 0.4237, train acc 0.856, test acc 0.823
epoch 19, loss 0.4223, train acc 0.855, test acc 0.837
epoch 20, loss 0.4206, train acc 0.856, test acc 0.832
epoch 21, loss 0.4188, train acc 0.856, test acc 0.841
epoch 22, loss 0.4175, train acc 0.856, test acc 0.834
epoch 23, loss 0.4155, train acc 0.858, test acc 0.838
epoch 24, loss 0.4143, train acc 0.857, test acc 0.831
epoch 25, loss 0.4126, train acc 0.859, test acc 0.840
epoch 26, loss 0.4120, train acc 0.859, test acc 0.839
epoch 27, loss 0.4108, train acc 0.860, test acc 0.834
epoch 28, loss 0.4096, train acc 0.860, test acc 0.839
epoch 29, loss 0.4083, train acc 0.860, test acc 0.842
epoch 30, loss 0.4083, train acc 0.860, test acc 0.840
epoch 31, loss 0.4066, train acc 0.861, test acc 0.841
epoch 32, loss 0.4060, train acc 0.861, test acc 0.843
epoch 33, loss 0.4053, train acc 0.861, test acc 0.841
epoch 34, loss 0.4039, train acc 0.861, test acc 0.843
epoch 35, loss 0.4034, train acc 0.862, test acc 0.843
epoch 36, loss 0.4028, train acc 0.862, test acc 0.844
epoch 37, loss 0.4027, train acc 0.862, test acc 0.842
epoch 38, loss 0.4010, train acc 0.863, test acc 0.837
epoch 39, loss 0.4002, train acc 0.863, test acc 0.843
epoch 40, loss 0.4001, train acc 0.863, test acc 0.842
epoch 41, loss 0.3992, train acc 0.863, test acc 0.843
epoch 42, loss 0.3986, train acc 0.863, test acc 0.842
epoch 43, loss 0.3991, train acc 0.863, test acc 0.842
epoch 44, loss 0.3970, train acc 0.863, test acc 0.840
epoch 45, loss 0.3974, train acc 0.864, test acc 0.826
epoch 46, loss 0.3963, train acc 0.864, test acc 0.829
epoch 47, loss 0.3963, train acc 0.864, test acc 0.841
epoch 48, loss 0.3953, train acc 0.864, test acc 0.843
epoch 49, loss 0.3951, train acc 0.864, test acc 0.845
epoch 50, loss 0.3937, train acc 0.865, test acc 0.845
```

实验 7：使用 L2 正则化 – 批量大小 256，学习率 0.1，迭代周期 50，权重衰减 0.001

```
epoch 1, loss 0.7852, train acc 0.751, test acc 0.783
epoch 2, loss 0.5735, train acc 0.812, test acc 0.809
epoch 3, loss 0.5291, train acc 0.825, test acc 0.809
epoch 4, loss 0.5063, train acc 0.832, test acc 0.816
epoch 5, loss 0.4907, train acc 0.836, test acc 0.811
epoch 6, loss 0.4806, train acc 0.839, test acc 0.818
epoch 7, loss 0.4712, train acc 0.842, test acc 0.824
epoch 8, loss 0.4654, train acc 0.844, test acc 0.828
epoch 9, loss 0.4606, train acc 0.845, test acc 0.825
epoch 10, loss 0.4558, train acc 0.847, test acc 0.833
epoch 11, loss 0.4538, train acc 0.847, test acc 0.830
epoch 12, loss 0.4506, train acc 0.848, test acc 0.832
epoch 13, loss 0.4469, train acc 0.849, test acc 0.831
epoch 14, loss 0.4438, train acc 0.850, test acc 0.825
epoch 15, loss 0.4424, train acc 0.850, test acc 0.834
epoch 16, loss 0.4408, train acc 0.851, test acc 0.830
epoch 17, loss 0.4385, train acc 0.852, test acc 0.830
epoch 18, loss 0.4369, train acc 0.853, test acc 0.837
epoch 19, loss 0.4357, train acc 0.853, test acc 0.831
epoch 20, loss 0.4341, train acc 0.854, test acc 0.838
epoch 21, loss 0.4331, train acc 0.854, test acc 0.837
epoch 22, loss 0.4323, train acc 0.854, test acc 0.823
epoch 23, loss 0.4314, train acc 0.855, test acc 0.835
epoch 24, loss 0.4308, train acc 0.855, test acc 0.836
epoch 25, loss 0.4300, train acc 0.855, test acc 0.837
epoch 26, loss 0.4287, train acc 0.856, test acc 0.821
epoch 27, loss 0.4275, train acc 0.856, test acc 0.821
epoch 28, loss 0.4278, train acc 0.856, test acc 0.836
epoch 29, loss 0.4273, train acc 0.856, test acc 0.839
epoch 30, loss 0.4259, train acc 0.856, test acc 0.840
epoch 31, loss 0.4253, train acc 0.856, test acc 0.839
epoch 32, loss 0.4254, train acc 0.856, test acc 0.842
epoch 33, loss 0.4251, train acc 0.857, test acc 0.839
epoch 34, loss 0.4237, train acc 0.857, test acc 0.836
epoch 35, loss 0.4234, train acc 0.858, test acc 0.833
epoch 36, loss 0.4229, train acc 0.857, test acc 0.838
epoch 37, loss 0.4235, train acc 0.857, test acc 0.836
epoch 38, loss 0.4228, train acc 0.857, test acc 0.837
epoch 39, loss 0.4229, train acc 0.858, test acc 0.831
epoch 40, loss 0.4215, train acc 0.857, test acc 0.838
epoch 41, loss 0.4210, train acc 0.858, test acc 0.837
epoch 42, loss 0.4216, train acc 0.858, test acc 0.836
epoch 43, loss 0.4210, train acc 0.858, test acc 0.840
epoch 44, loss 0.4210, train acc 0.858, test acc 0.841
epoch 45, loss 0.4209, train acc 0.858, test acc 0.839
epoch 46, loss 0.4208, train acc 0.858, test acc 0.840
epoch 47, loss 0.4211, train acc 0.858, test acc 0.837
epoch 48, loss 0.4195, train acc 0.858, test acc 0.832
epoch 49, loss 0.4200, train acc 0.858, test acc 0.839
epoch 50, loss 0.4198, train acc 0.858, test acc 0.841
```

实验 8：使用早停（Early Stopping）— 批量大小 256，学习率 0.1

```
epoch 1, loss 0.7866, train acc 0.750, test acc 0.781
epoch 2, loss 0.5693, train acc 0.814, test acc 0.807
epoch 3, loss 0.5249, train acc 0.826, test acc 0.818
epoch 4, loss 0.5007, train acc 0.832, test acc 0.824
epoch 5, loss 0.4847, train acc 0.837, test acc 0.822
epoch 6, loss 0.4735, train acc 0.840, test acc 0.826
epoch 7, loss 0.4655, train acc 0.843, test acc 0.830
epoch 8, loss 0.4578, train acc 0.845, test acc 0.829
epoch 9, loss 0.4517, train acc 0.847, test acc 0.810
epoch 10, loss 0.4479, train acc 0.848, test acc 0.826
epoch 11, loss 0.4425, train acc 0.850, test acc 0.827
epoch 12, loss 0.4391, train acc 0.851, test acc 0.832
epoch 13, loss 0.4357, train acc 0.852, test acc 0.836
epoch 14, loss 0.4330, train acc 0.852, test acc 0.837
epoch 15, loss 0.4307, train acc 0.853, test acc 0.832
epoch 16, loss 0.4282, train acc 0.853, test acc 0.837
epoch 17, loss 0.4255, train acc 0.855, test acc 0.832
epoch 18, loss 0.4236, train acc 0.856, test acc 0.838
epoch 19, loss 0.4218, train acc 0.856, test acc 0.836
epoch 20, loss 0.4205, train acc 0.856, test acc 0.840
epoch 21, loss 0.4182, train acc 0.857, test acc 0.840
epoch 22, loss 0.4168, train acc 0.858, test acc 0.838
epoch 23, loss 0.4156, train acc 0.858, test acc 0.838
epoch 24, loss 0.4143, train acc 0.858, test acc 0.842
epoch 25, loss 0.4125, train acc 0.860, test acc 0.837
epoch 26, loss 0.4124, train acc 0.859, test acc 0.835
epoch 27, loss 0.4114, train acc 0.859, test acc 0.842
epoch 28, loss 0.4100, train acc 0.859, test acc 0.842
epoch 29, loss 0.4096, train acc 0.859, test acc 0.837
Early stopping at epoch 29
```

实验 9：使用 Dropout - 批量大小 256，学习率 0.1，迭代周期 50，Dropout 比率 0.5

```
epoch 1, loss 1.0982, train acc 0.614, test acc 0.741
epoch 2, loss 0.6496, train acc 0.774, test acc 0.791
epoch 3, loss 0.5601, train acc 0.806, test acc 0.796
epoch 4, loss 0.5129, train acc 0.822, test acc 0.815
epoch 5, loss 0.4783, train acc 0.833, test acc 0.817
epoch 6, loss 0.4599, train acc 0.839, test acc 0.835
epoch 7, loss 0.4432, train acc 0.843, test acc 0.844
epoch 8, loss 0.4298, train acc 0.848, test acc 0.837
epoch 9, loss 0.4182, train acc 0.852, test acc 0.846
epoch 10, loss 0.4098, train acc 0.855, test acc 0.836
epoch 11, loss 0.4002, train acc 0.858, test acc 0.842
epoch 12, loss 0.3926, train acc 0.859, test acc 0.849
epoch 13, loss 0.3857, train acc 0.863, test acc 0.829
epoch 14, loss 0.3820, train acc 0.863, test acc 0.858
epoch 15, loss 0.3749, train acc 0.866, test acc 0.863
epoch 16, loss 0.3710, train acc 0.868, test acc 0.858
epoch 17, loss 0.3670, train acc 0.871, test acc 0.862
epoch 18, loss 0.3602, train acc 0.872, test acc 0.866
epoch 19, loss 0.3582, train acc 0.871, test acc 0.864
epoch 20, loss 0.3546, train acc 0.874, test acc 0.869
epoch 21, loss 0.3490, train acc 0.875, test acc 0.865
epoch 22, loss 0.3456, train acc 0.878, test acc 0.866
epoch 23, loss 0.3441, train acc 0.877, test acc 0.857
epoch 24, loss 0.3411, train acc 0.879, test acc 0.864
epoch 25, loss 0.3402, train acc 0.879, test acc 0.869
epoch 26, loss 0.3337, train acc 0.881, test acc 0.871
epoch 27, loss 0.3323, train acc 0.881, test acc 0.875
epoch 28, loss 0.3307, train acc 0.882, test acc 0.873
epoch 29, loss 0.3263, train acc 0.885, test acc 0.872
epoch 30, loss 0.3249, train acc 0.884, test acc 0.875
epoch 31, loss 0.3220, train acc 0.886, test acc 0.873
epoch 32, loss 0.3204, train acc 0.885, test acc 0.874
epoch 33, loss 0.3168, train acc 0.887, test acc 0.869
epoch 34, loss 0.3154, train acc 0.888, test acc 0.871
epoch 35, loss 0.3144, train acc 0.888, test acc 0.875
epoch 36, loss 0.3108, train acc 0.889, test acc 0.876
epoch 37, loss 0.3114, train acc 0.887, test acc 0.879
epoch 38, loss 0.3069, train acc 0.891, test acc 0.879
epoch 39, loss 0.3060, train acc 0.890, test acc 0.876
epoch 40, loss 0.3023, train acc 0.890, test acc 0.876
epoch 41, loss 0.3009, train acc 0.892, test acc 0.875
epoch 42, loss 0.3007, train acc 0.891, test acc 0.872
epoch 43, loss 0.2966, train acc 0.893, test acc 0.878
epoch 44, loss 0.2972, train acc 0.894, test acc 0.883
epoch 45, loss 0.2950, train acc 0.894, test acc 0.878
epoch 46, loss 0.2937, train acc 0.895, test acc 0.882
epoch 47, loss 0.2922, train acc 0.895, test acc 0.880
epoch 48, loss 0.2889, train acc 0.898, test acc 0.884
epoch 49, loss 0.2905, train acc 0.894, test acc 0.881
epoch 50, loss 0.2880, train acc 0.896, test acc 0.875
```

[小结或讨论]

在本次实验中，我完成了 PyTorch 开发环境配置以及线性模型相关实验，过程中遇到了一些问题并积累了宝贵的经验。

在环境配置部分，我利用 Mac 系统通过 VSCode 远程连接 Linux 服务器进行操作，利用已有的 yyzttt 环境，安装了 Torch 1.9.1 和 CUDA 11.7。通过编写测试代码，验证了 PyTorch 和 torchvision 的版本，检查了 GPU 支持情况，并进行了张量运算测试，结果显示环境配置正确，GPU 可用，这为后续实验奠定了基础。

线性模型实验部分，从零开始实现线性回归时，我生成了数据集，定义了数据迭代器、模型、损失函数和优化算法，并进行了训练。通过设置真实参数，观察到训练过程中损失逐渐降低，最终学习到的参数与真实参数误差较小。练习中，我思考了权重初始化、损失函数中 reshape 函数的作用以及样本数与批量大小不整除时数据迭代器的行为。将权重初始化为零在简单线性回归中可行，但在复杂神经网络中会因对称性导致神经元学习相同特征，影响模型性能；reshape 函数确保了预测值和真实标签维度一致，避免计算损失时出现错误；数据迭代器在样本数不能被批量大小整除时，会将剩余样本作为最后一个小批量，保证所有数据参与训练。这些思考加深了我对线性回归模型原理和细节的理解。

线性回归的简洁实现使用 PyTorch 内置模块，简化了代码编写，通过 Sequential 构建模型，使用 MSELoss 和 SGD 优化器，训练过程同样达到了较好的效果，体现了 PyTorch 框架的便捷性。

在熟悉 Fashion-MNIST 数据集后，进行了 softmax 回归的从零开始实现和简洁实现。从零实现时，定义了 softmax 函数、交叉熵损失函数，通过自定义训练函数进行模型训练和评估；简洁实现利用 PyTorch 的 nn 模块和优化器，代码更简洁高效。练习中调整超参数，如批量大小、学习率和迭代周期数，发现批

量大小影响训练稳定性和收敛速度，学习率过高易导致损失波动、模型不稳定，过低则收敛缓慢，迭代周期数增加可能导致过拟合，测试精度在后期下降。

为解决过拟合问题，尝试了 L2 正则化、早停法和 Dropout 等方法。L2 正则化通过在损失函数中添加权重衰减项，抑制了模型复杂度，使测试精度稳定；早停法在测试精度连续多个周期未提升时提前终止训练，避免过度拟合；Dropout 通过随机丢弃神经元，增强了模型的泛化能力。这些方法在实验中均有效提升了模型性能，让我认识到正则化技术在防止过拟合中的重要性。

本次实验让我熟悉了 PyTorch 环境配置和线性模型的实现，深入理解了模型训练过程和参数调整的影响，掌握了应对过拟合的方法。实验中遇到的问题和解决方案为今后深度学习的学习和实践提供了宝贵经验，也让我认识到理论与实践结合的重要性，只有通过实际编码和调试，才能更好地理解和掌握深度学习知识。