```python
# -*- coding: utf-8 -*-
# @Time    : 2024/3/5
# @Author  : quanchenliu
# @Function: 简洁实现线性回归

import numpy as np
import torch
from d2l import torch as d2l
from torch.utils import data
from torch import nn

'''生成数据集'''
def synthetic_data(w, b, num_examples):
    X = torch.normal(0, 1, (num_examples, len(w)))      # [1000, 2]
    y = torch.matmul(X, w) + b                          # [1000, 2] * [2] =
[1000]
    y += torch.normal(0, 0.01, y.shape)
    return X, y.reshape(-1, 1)

'''读取数据集: 调用现有框架中的API来读取数据集'''
def load_array(data_arrays, batch_size, is_train=True):             # @save
    dataset = data.TensorDataset(*data_arrays)                      # 将特
征张量 features 和标签张量 labels 组合成一个数据集对象
    return data.DataLoader(dataset, batch_size, shuffle=is_train)   # 创建
一个数据加载器对象，批量大小为 10，且打乱数据集顺序（shuffle）

def main():
    true_w = torch.tensor([2, -3.4])                                # [2]
    true_b = 4.2
    features, labels = synthetic_data(true_w, true_b, 1000)     # [1000, 2]
[1000, 1]

    batch_size = 10
    data_iter = load_array((features, labels), batch_size)      # 读取数据
集

    ''' nn.Sequential: 一个顺序容器，按序执行其中的每个模块
        nn.Linear: 一个全连接层，第一个参数表征输入特征的维度，第二个参数表征输
出特征的维度'''
    net = nn.Sequential(nn.Linear(2, 1))                            # 定义模型
变量
    net[0].weight.data.normal_(0, 0.01)                             # 初始化模
型参数
    net[0].bias.data.fill_(0)
    loss = nn.MSELoss()                                             # 定义损失
函数
    trainer = torch.optim.SGD(net.parameters(), lr=0.03)        # 定义优化
算法
```

```
    num_epochs = 3
    for epoch in range(num_epochs):
        for X, y in data_iter:
            l = loss(net(X), y)                                    # 计算当前
小批量数据的均分误差
            trainer.zero_grad()                                    # 清除之前
保存的梯度，以避免梯度的累积
            l.backward()                                           # 计算损失
函数关于网络参数的梯度
            trainer.step()                                         # 更新模型
参数
        l = loss(net(features), labels)                            # 计算整个
模型的损失（均方误差）
        print(f'epoch={epoch+1}, loss={l:f}')

    w = net[0].weight.data
    print(f'w的估计误差: {true_w - w.reshape(true_w.shape)}')
    b = net[0].bias.data
    print(f'b的估计误差: {true_b - b}')


if __name__ == "__main__":
    main()
```