机器学习实验报告

实验名称：建立全连接神经网络

学生：谢兴

学号：58122204

日期：2024/4/24

指导老师：刘胥影

助教：田秋雨

目录

1 任务描述

通过两种方式实现全连接神经网络，并对图片分类任务行进行测试与实验。

1. 手动实现简单的全连接神经网络
2. 使用 Pytorch 库简洁实现全连接神经网络

Fashion-MNIST 图片分类数据集包含 10 个类别的时装图像，训练集有 60,000 张图片，测试集中有 10,000 张图片。图片为灰度图片，高度（h）和宽度（w）均为 28 像素，通道数（channel）为 1。10 个类别分别为：t-shirt(T 恤), trouser（裤子）, pullover（套衫）, dress（连衣裙）, coat（外套），sandal（凉鞋），shirt（衬衫），sneaker（运动鞋），bag（包），ankle boot（短靴）。使用训练集数据进行训练，测试集数据进行测试。

2 教学要求

1. 掌握多层前馈神经网络及 BP 算法的原理与构建
2. 了解 PyTorch 库，掌握本实验涉及的相关部分
3. 进行参数分析实验，理解学习率等参数的影响

3 实验要求

3.1 使用 Python 编程构建手动实现单隐层全连接神经网络

模型架构

- 输入层 $28 \times 28 = 784$ 个节点，输出层 10 个节点，隐藏层 256 个节点。注意，可以将这两个变量都视为超参数。通常选择 2 的若干次幂作为层的宽度。因为内存在硬件中的分配和寻址方式，这么做往往可以在计算上更高效。
- 激活函数：ReLU 函数
- 损失函数：Cross entropy
- 性能指标：准确率
- 优化算法：实现标准 BP 或小批量梯度下降算法均可

实现内容

1. 初始化模型参数：对于每一层都要记录一个权重矩阵和一个偏置向量。

2. 设置激活函数：使用 ReLU 函数作为激活函数，要求手动实现该函数。

3. 前向计算：实现该函数。注意：需要将每个二维图像转化为向量进行操作。

4. 设置损失函数：使用 cross entropy 作为损失函数。可以自己手动实现，也可以直接调用 `nn.CrossEntropyLoss` 函数。

5. 训练模型：

   (a) 实现训练函数：该训练函数将会运行多个迭代周期（由 *num_epochs* 指定）。在每个迭代周期结束时，利用 *test_iter* 访问到的测试数据集对模型进行评估。利用后面给出的 Animator 类来可视化训练进度。

   (b) 可以使用 PyTorch 内置的优化器（`torch.optim.SGD`），也可以使用自己定制的优化器。

   (c) 可以调用 `torch.optim.SGD` 函数进行参数更新。

   (d) 迭代周期数 *epoch* 设置为 10，学习率设置为 0.1，训练模型。

6. 设置性能函数：使用准确率 *accuracy* 作为性能指标。实现该函数。

7. 模型评估：

   (a) 对测试集数据进行测试。

   (b) 进行性能评估。

8. 参数分析实验：

   (a) 在所有其他参数保持不变的情况下，更改超参数 *num_hiddens* 的值，并查看此超参数的变化对结果有何影响。确定此超参数的最佳值。

   (b) 改变学习速率会如何影响结果？保持模型架构和其他超参数（包括轮数）不变，学习率设置为多少会带来最好的结果？

3.2 使用 PyTorch 库简洁实现全连接神经网络

手动实现一个简单的多层神经网络是很容易的。然而如果网络有很多层，从零开始实现会变得很麻烦。可以使用高级 API 如 PyTorch 库简洁实现。

1. 请使用 PyTorch 库简洁实现前述的全连接神经网络，并进行模型评估。

(a) 优化器：使用 `torch.optim.SGD`

(b) 小批量数据载入函数参见提供的代码。

2. **参数分析实验**

(a) 尝试添加不同数量的隐藏层（也可以修改学习率），怎么样设置效果最好？

(b) 尝试不同的激活函数，哪个效果最好？

(c) 尝试不同的方案来初始化权重，什么方法效果最好？

3.3 提交要求

其中报告内容包括以下几个部分：

1. 手动实现单隐层全连接神经网络

(a) 训练过程中，训练集与验证集误差随 epoch 变化的曲线图

(b) 性能评估结果

(c) 参数分析实验：包括实验设置与结果分析

2. 使用 PyTorch 库简洁实现全连接神经网络

(a) 性能评估结果

(b) 参数分析实验：包括实验设置与结果分析

4 实验部分

4.1 实验环境

实验环境见表 1。

4.2 使用 Python 编程构建手动实现单隐层全连接神经网络

4.2.1 模型训练参数与可视化

1. 输入层 28*28=784 个节点，输出层 10 个节点，隐藏层 256 个节点

2. 激活函数：ReLU 函数

3. 损失函数：Cross entropy

表 1: Experiment Environment

Items	Version
CPU	Intel Core i5-1135G7
RAM	16 GB
Python	3.11.5
Pytorch	2.2.1
Operating system	Windows11

4. 性能指标：准确率

5. 小批量梯度下降算法

训练过程可视

如图 1 和图 2 所示。（囿于实验环境算力有限，训练过程可视化没有实现测验集误差和准确率随 batch 的变化曲线）

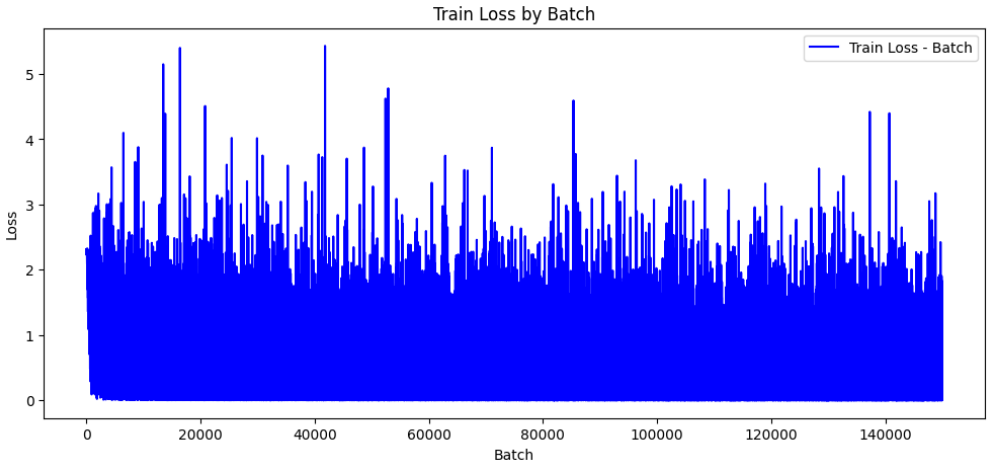

图 1: 训练集误差随 batch 变化的曲线图

4.2.2 模型评估与实验结果

训练过程中，训练集与验证集误差随 epoch 变化的曲线图如图 3 所示。从图中可以看到模型经过 10 个 epoch 的迭代，从图中可以看到模型经过 10 个 epoch 的迭代，模型在训

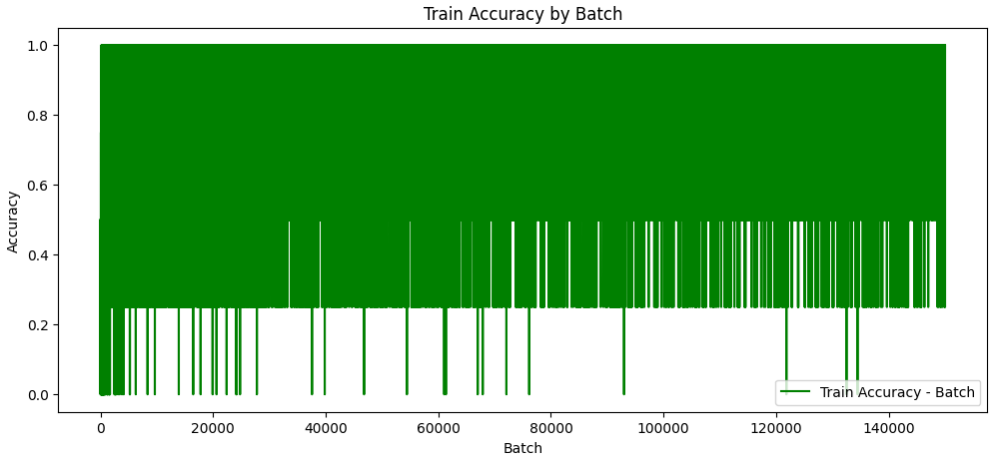

图 2: 训练集准确率随 batch 变化的曲线图

练集和测验集上的准确率均高于 87%, 模型在训练集和测验集上的 loss 均低于 0.4，说明模型具有较强的分类预测能力。

但与此同时，模型在测验集上的准确率低于在训练集上的准确率，说明模型存在一定的过拟合现象。

这两者评估结果表明多层感知机的模型性能较为强大。

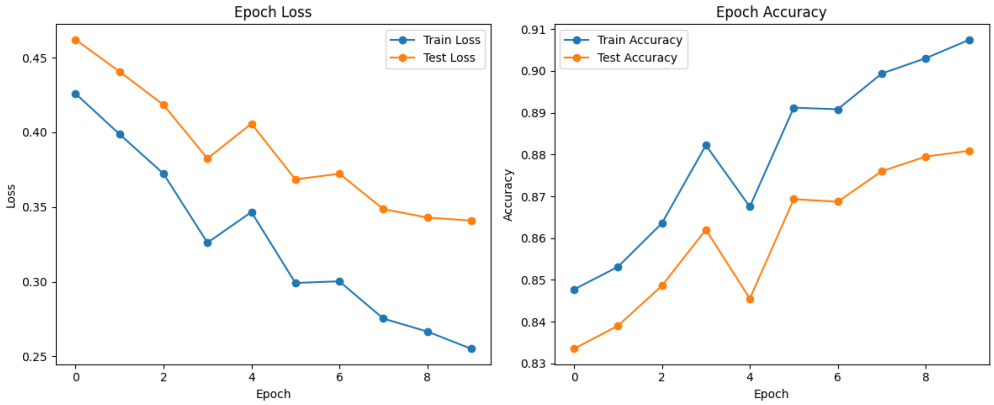

图 3: 训练集与验证集误差随 epoch 变化的曲线图

4.2.3 参数分析实验

1. **修改隐藏层数**

   分别测试了 num_hiddens 为 64, 128, 256, 512 时的模型性能，发现 num_hiddens 为 256 时，模型性能最好。同时我也观察到 num_hiddens 越大，模型训练所需时间越长。

2. **修改学习率**

分别测试了 lr 为 0.01, 0.05，0.1, 时的模型性能，发现 lr 为 0.01 时，模型性能最好。调参过程中注意到 lr 越小，模型训练所需时间越长。

3. **修改 epoch**

进过调参的经验和实际测试发现，epoch 增大，模型预测的准确率会有一定的提高，但是增大到一定程度后，模型的准确率会趋于稳定，不再有明显的提高。并且增大 epoch 会增加训练时间，所以在实际应用中，需要根据实际情况来选择 epoch 的大小。

4.3 使用 PyTorch 库简洁实现全连接神经网络

4.3.1 模型训练参数与可视化

使用 PyTorch 库简洁实现全连接神经网络的训练参数如表 2 所示。

表 2: 训练参数

参数	值
learning rate	0.1
epochs	10
optimizer	SGD
loss function	Cross Entropy
performance	accuracy
batch size	16
num_hiddens	256

训练过程可视化

如图 4 和图 5 所示。（囿于实验环境算力有限，训练过程可视化没有实现测验集误差和准确率随 batch 的变化曲线）

4.3.2 模型评估与实验结果

训练过程中，训练集与验证集误差随 epoch 变化的曲线图如图 6 所示。从图中可以看到模型经过 10 个 epoch 的迭代，模型在训练集和测验集上的准确率均高于 87%, 模型在训练集和测验集上的 loss 均低于 0.4，说明模型具有较强的分类预测能力。

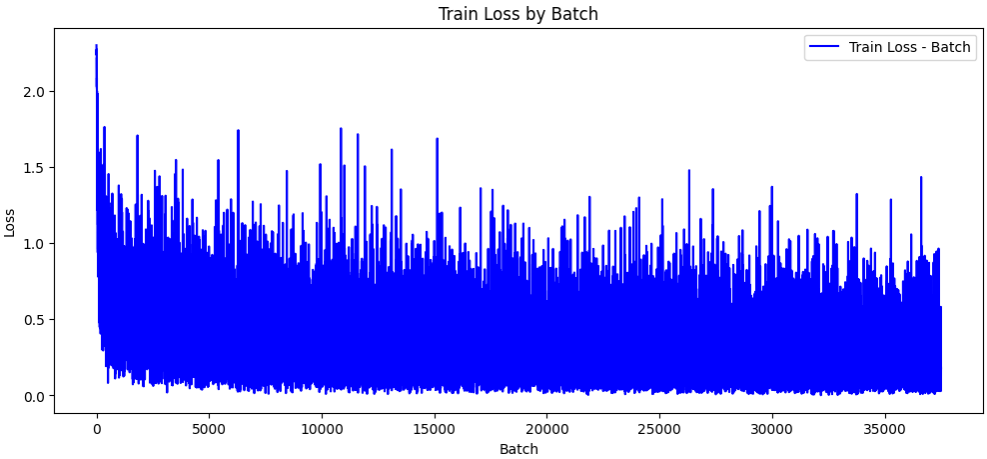

图 4: 训练集误差随 batch 变化的曲线图

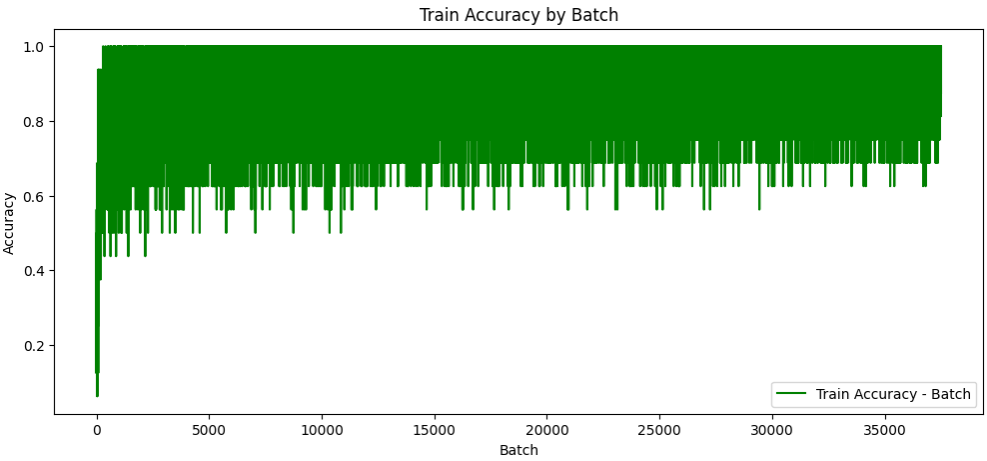

图 5: 训练集准确率随 batch 变化的曲线图

但与此同时，模型在测验集上的准确率低于在训练集上的准确率，说明模型存在一定的过拟合现象。

这两者评估结果表明多层感知机的模型性能较为强大。

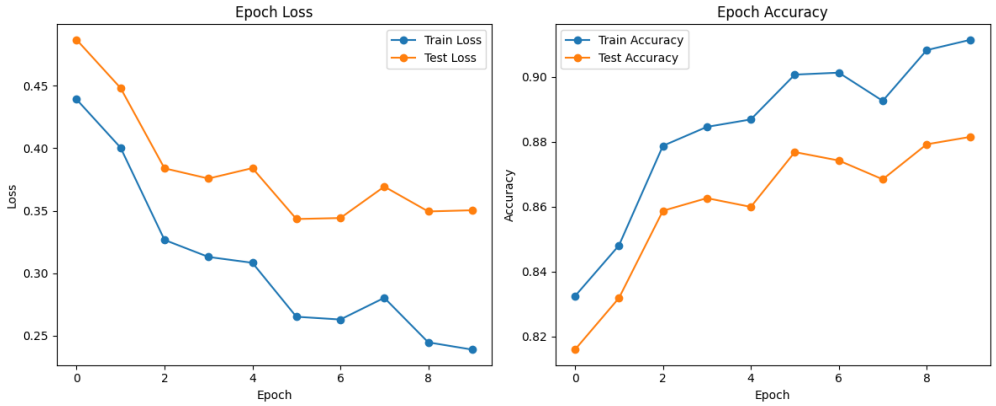

图 6: 训练集与验证集误差随 epoch 变化的曲线图

4.3.3 参数分析实验

1. **修改隐藏层数和学习率**

   分别测试了 lr 为 0.01, 0.05，0.1, 时的模型性能，发现 lr 为 0.01 时，模型性能最好。分别测试了 num_hiddens 为 64, 128, 256, 512 时的模型性能，发现 num_hiddens 为 256 时，模型性能最好。

2. **使用不同的激活函数**

   分别使用了 relu 和 sigmoid 作为激活函数，发现 relu 的效果更好。

3. **使用不同的方案来初始化权重**

   分别使用了不同的方案初始化权重，发现以下的参数初始化方法效果最好。

```
    # Initialize parameters
    nn.init.normal_(self.fc1.weight, mean=0, std=0.1)
    nn.init.constant_(self.fc1.bias, 0)
    nn.init.normal_(self.fc2.weight, mean=0, std=0.1)
    nn.init.constant_(self.fc2.bias, 0)

```

5 实验总结

5.1 实验结果

本次实验实现了两种方式的全连接神经网络，一种是手动实现的简单全连接神经网络，另一种是使用 PyTorch 库简洁实现的全连接神经网络。通过实验，我们发现使用 PyTorch 库简洁实现的全连接神经网络在训练过程中的代码量更少，更加简洁，同时也更加高效。在参数分析实验中，我们发现在手动实现的全连接神经网络中，隐藏层节点数为 256 时，效果最好；在 PyTorch 库简洁实现的全连接神经网络中，隐藏层节点数为 256 时，效果最好。

5.2 实验感悟

通过这次实验，我对全连接神经网络有了更深入的了解，掌握了多层前馈神经网络及 BP 算法的原理与构建，了解了 PyTorch 库，掌握了本实验涉及的相关部分。

5.3 实验的不足以及改进方向

模型性能方面:这次实验是通过两种方式实现全连接神经网络,并对图片数据集 Fashion-MNIST 中的图片进行分类, 我们可以看到，本次实验的准确率并不是很高（仅有 85% 左右），一方面与实验的 epochs 数过小有关，另一方面也与模型本身的性能有关，在以后的学习中可以使用更加复杂的模型（如 CNN 等），以提高模型的准确率等性能。

训练过程可视化方面: 未来我们可以采用像 YOLOv5 那样在训练过程中直接在终端实时动态更新输出每一个 batch 后参数更新后的各种 loss，和准确率，召回率，置信度等相关参数等。

A 使用 Python 编程构建手动实现单隐层全连接神经网络 Code

```python
import numpy as np
import matplotlib.pyplot as plt
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import config
args = config.args


# 数据加载
trans = transforms.ToTensor()
mnist_train = datasets.FashionMNIST(root="./data", train=True, transform=
 trans, download=True)
mnist_test = datasets.FashionMNIST(root="./data", train=False, transform=
 trans, download=True)

num_inputs, num_outputs, num_hiddens = 784, 10, 256


train_iter = DataLoader(mnist_train, batch_size=args.batch_size, shuffle=
 True, num_workers=0)
test_iter = DataLoader(mnist_test, batch_size=10000, shuffle=False,
 num_workers=0)

# 激活函数和softmax函数
def relu(x):
    return np.maximum(x, 0)

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
    return exp_x / np.sum(exp_x, axis=1, keepdims=True)

# 损失函数
def cross_entropy(y_hat, y):
    m = y_hat.shape[0]
```

```
    y_one_hot = np.eye(num_outputs)[y]
    return -np.sum(np.log(y_hat[range(m), y])) / m

# 前向传播
def model(X, w1, b1, w2, b2):
    H = relu(np.dot(X, w1) + b1)
    O = np.dot(H, w2) + b2
    return softmax(O), H

# 梯度计算
def calc_gradient(X, y, y_hat, H, w2):
    m = y.shape[0]
    y_one_hot = np.eye(num_outputs)[y]

    dO = (y_hat - y_one_hot) / m
    grad_w2 = np.dot(H.T, dO)
    grad_b2 = np.sum(dO, axis=0)

    dH = np.dot(dO, w2.T) * (H > 0).astype(float)
    grad_w1 = np.dot(X.T, dH)
    grad_b1 = np.sum(dH, axis=0)

    return grad_w1, grad_b1, grad_w2, grad_b2

# 参数更新
def update_params(params, grads, lr):
    for param, grad in zip(params, grads):
        param -= lr * grad

# 参数初始化
w1 = np.random.randn(num_inputs, num_hiddens) * 0.01
b1 = np.zeros(num_hiddens)
w2 = np.random.randn(num_hiddens, num_outputs) * 0.01
b2 = np.zeros(num_outputs)
params = [w1, b1, w2, b2]

# Metrics storage
batch_metrics = {
    'train_losses': [],
    'train_accs': [],
    'test_losses': [],
```

```
    'test_accs': []
}
epoch_metrics = {
    'train_losses': [],
    'train_accs': [],
    'test_losses': [],
    'test_accs': []
}
def accuracy(loader, w1, b1, w2, b2):
    total_loss = 0
    total_correct = 0
    total_samples = 0

    # 设置数据加载器加载的是numpy数组而非torch.Tensor
    for images, labels in loader:
        X = images.view(-1, num_inputs).numpy()  # 转换为适当的形状和类型
        y = labels.numpy()

        # 前向传播
        y_hat, _ = model(X, w1, b1, w2, b2)

        # 计算损失
        loss = cross_entropy(y_hat, y)
        total_loss += loss * len(y)  # 累计总损失
        total_correct += (np.argmax(y_hat, axis=1) == y).sum()  # 计算正确预测的数量
        total_samples += len(y)

    average_loss = total_loss / total_samples
    accuracy = total_correct / total_samples
    return average_loss, accuracy

# Training and evaluation loop

for epoch in range(args.epochs):
    for images, labels in train_iter:
        X = images.view(-1, num_inputs).numpy()
        y = labels.numpy()
        y_hat, H = model(X, *params)
        loss = cross_entropy(y_hat, y)
        grads = calc_gradient(X, y, y_hat, H, w2)
```

```
            update_params(params, grads, lr=args.lr)

            batch_metrics['train_losses'].append(loss)
            batch_acc = (np.argmax(y_hat, axis=1) == y).mean()
            batch_metrics['train_accs'].append(batch_acc)

        # Evaluate at the end of each epoch
        epoch_train_loss, epoch_train_acc = accuracy(train_iter, *params)
        epoch_test_loss, epoch_test_acc = accuracy(test_iter, *params)

        epoch_metrics['train_losses'].append(epoch_train_loss)
        epoch_metrics['train_accs'].append(epoch_train_acc)
        epoch_metrics['test_losses'].append(epoch_test_loss)
        epoch_metrics['test_accs'].append(epoch_test_acc)

        print(f'Epoch {epoch + 1}, Train Loss: {epoch_train_loss:.6f}, Train Acc
    : {epoch_train_acc:.6f}, Test Loss: {epoch_test_loss:.6f}, Test Acc: {
    epoch_test_acc:.6f}')




# 绘制训练和测试损失与准确率的曲线图


# 绘制训练损失的曲线图
plt.figure(figsize=(12, 5))
plt.plot(batch_metrics['train_losses'], label='Train Loss - Batch', color='
    blue')
plt.title('Train Loss by Batch')
plt.xlabel('Batch')
plt.ylabel('Loss')
plt.legend()
plt.show()


# 绘制训练准确率的曲线图
plt.figure(figsize=(12, 5))
plt.plot(batch_metrics['train_accs'], label='Train Accuracy - Batch', color=
    'green')
plt.title('Train Accuracy by Batch')
```

```
plt.xlabel('Batch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()


# 绘制训练和测试损失的曲线图
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epoch_metrics['train_losses'], label='Train Loss', marker='o')
plt.plot(epoch_metrics['test_losses'], label='Test Loss', marker='o')
plt.title('Epoch Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# 绘制训练和测试准确率的曲线图
plt.subplot(1, 2, 2)
plt.plot(epoch_metrics['train_accs'], label='Train Accuracy', marker='o')
plt.plot(epoch_metrics['test_accs'], label='Test Accuracy', marker='o')
plt.title('Epoch Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

B 使用 PyTorch 库简洁实现全连接神经网络 Code

```
import torch
from torch import nn, optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import config

args = config.args
device = torch.device('cuda' if not args.cpu and torch.cuda.is_available()
 else 'cpu')
```

```
10
11  # Data loading
12  trans = transforms.ToTensor()
13  mnist_train = datasets.FashionMNIST(root="./data", train=True, transform=
     trans, download=True)
14  mnist_test = datasets.FashionMNIST(root="./data", train=False, transform=
     trans, download=True)
15
16  # Model definition
17  class CustomNetwork(nn.Module):
18      def __init__(self, num_inputs, num_hiddens, num_outputs):
19          super(CustomNetwork, self).__init__()
20          self.fc1 = nn.Linear(num_inputs, num_hiddens)
21          self.fc2 = nn.Linear(num_hiddens, num_outputs)
22          # Initialize parameters
23          nn.init.normal_(self.fc1.weight, mean=0, std=0.01)
24          nn.init.constant_(self.fc1.bias, 0)
25          nn.init.normal_(self.fc2.weight, mean=0, std=0.01)
26          nn.init.constant_(self.fc2.bias, 0)
27
28      def forward(self, x):
29          x = torch.relu(self.fc1(x))
30          x = self.fc2(x)
31          return x
32
33  # Model instantiation, loss, and optimizer
34  model = CustomNetwork(784, 256, 10).to(device)
35  criterion = nn.CrossEntropyLoss()
36  optimizer = optim.SGD(model.parameters(), lr=args.lr)
37
38  # Data loading
39  train_loader = DataLoader(mnist_train, batch_size=args.batch_size, shuffle=
     True)
40  test_loader = DataLoader(mnist_test, batch_size=10000, shuffle=False)
41
42  # Function to evaluate the network
43  def accuracy(loader):
44      total_loss, total_correct, total = 0, 0, 0
45      model.eval()
46      with torch.no_grad():
47          for images, labels in loader:
```

```
            images = images.view(-1, 784).to(device)
            labels = labels.to(device)
            output = model(images)
            loss = criterion(output, labels)
            total_loss += loss.item()
            total_correct += (output.argmax(1) == labels).sum().item()
            total += labels.size(0)
    model.train()
    return total_loss / len(loader), total_correct / total

# Metrics storage
batch_metrics = {
    'train_losses': [],
    'train_accs': [],
    'test_losses': [],
    'test_accs': []
}
epoch_metrics = {
    'train_losses': [],
    'train_accs': [],
    'test_losses': [],
    'test_accs': []
}

# Training and evaluation loop
for epoch in range(args.epochs):
    for images, labels in train_loader:
        images = images.view(-1, 784).to(device)
        labels = labels.to(device)
        output = model(images)
        loss = criterion(output, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Record batch loss and accuracy
        batch_loss = loss.item()
        batch_accuracy = (output.argmax(1) == labels).float().mean().item()
        batch_metrics['train_losses'].append(batch_loss)
        batch_metrics['train_accs'].append(batch_accuracy)

```

```
        # Evaluate on test data after each epoch
        test_loss, test_accuracy = accuracy(test_loader)
        batch_metrics['test_losses'].append(test_loss)
        batch_metrics['test_accs'].append(test_accuracy)

        # Record epoch metrics
        epoch_train_loss, epoch_train_acc = accuracy(train_loader)
        epoch_metrics['train_losses'].append(epoch_train_loss)
        epoch_metrics['train_accs'].append(epoch_train_acc)
        epoch_metrics['test_losses'].append(test_loss)
        epoch_metrics['test_accs'].append(test_accuracy)

        print(f'Epoch {epoch + 1}, Train Loss: {epoch_train_loss:.6f}, Train Acc
    : {epoch_train_acc:.6f}, Test Loss: {test_loss:.6f}, Test Acc: {
    test_accuracy:.6f}')

# Plotting code

# Plotting
# 绘制训练损失的曲线图
plt.figure(figsize=(6, 5))
plt.plot(batch_metrics['train_losses'], label='Train Loss - Batch', color='
    blue')
plt.title('Train Loss by Batch')
plt.xlabel('Batch')
plt.ylabel('Loss')
plt.legend()
plt.show()


# 绘制训练准确率的曲线图
plt.figure(figsize=(6, 5))
plt.plot(batch_metrics['train_accs'], label='Train Accuracy - Batch', color=
    'green')
plt.title('Train Accuracy by Batch')
plt.xlabel('Batch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()


```

```
126
127   # 绘制训练和测试损失的曲线图
128   plt.figure(figsize=(12, 5))
129   plt.subplot(1, 2, 1)
130   plt.plot(epoch_metrics['train_losses'], label='Train Loss', marker='o')
131   plt.plot(epoch_metrics['test_losses'], label='Test Loss', marker='o')
132   plt.title('Epoch Loss')
133   plt.xlabel('Epoch')
134   plt.ylabel('Loss')
135   plt.legend()
136
137   # 绘制训练和测试准确率的曲线图
138   plt.subplot(1, 2, 2)
139   plt.plot(epoch_metrics['train_accs'], label='Train Accuracy', marker='o')
140   plt.plot(epoch_metrics['test_accs'], label='Test Accuracy', marker='o')
141   plt.title('Epoch Accuracy')
142   plt.xlabel('Epoch')
143   plt.ylabel('Accuracy')
144   plt.legend()
145
146   plt.tight_layout()
147   plt.show()
148
149
150   args.save(model.state_dict(), 'fashion_mnist.pt')
```

C config.py

```
1
2   import argparse
3 parser = argparse.ArgumentParser(description='Hyper-parameters management')
4
5 # Hardware options
6 parser.add_argument('--n_threads', type=int, default=6,
7                     help='number of threads for data loading')
8 parser.add_argument('--cpu', action='store_true',
9                     help='use cpu only')
10 parser.add_argument('--seed', type=int, default=1, help='random seed')
11
12 # data in/out and dataset
```

```
parser.add_argument('--dataset_path',default = './data/',help='fixed trainset root path')

parser.add_argument('--save',default='model',help='save path of trained model')

parser.add_argument('--predict',default='model',help='save path of predict model')

parser.add_argument('--batch_size', type=list, default=256,help='batch size of trainset')

# train
parser.add_argument('--epochs', type=int, default=10, metavar='N',help='number of epochs to train (default: 10)')

parser.add_argument('--lr', type=float, default=0.1, metavar='LR',help='learning rate (default: 0.01)')

parser.add_argument('--momentum', type=float, default=0.99, metavar='M',help='SGD momentum (default: 0.5)')

parser.add_argument('--weight_decay', type=float, default=3e-5, metavar='W', help='SGD weight_decay (default: 3e-5)')

parser.add_argument('--nesterov', type=bool, default=True, help='SGD nesterov (default: True)')

parser.add_argument('--early-stop', default=20, type=int, help='early stopping (default: 20)')
#args = parser.parse_args()
args, unknown = parser.parse_known_args()
```

D predict.py

```
# predict.py
import torch
from torchvision import transforms
from PIL import Image

```

```
def load_model(model_path):
    """
    Load the PyTorch model from the specified path.
    """
    model = torch.load(model_path)
    #model.eval()  # Set the model to evaluation mode
    return model

def predict(model, image):
    """
    Make a prediction for the given image using the provided model.
    """
    # Assuming image is a PIL image, we must transform it first.
    transform = transforms.Compose([
        transforms.ToTensor(),  # Convert the PIL Image to a tensor.
        transforms.Normalize((0.5,), (0.5,))  # Normalize the image
    ])

    # Apply the transformations and add batch dimension
    image = transform(image).unsqueeze(0)

    # No need to track gradients for prediction
    with torch.no_grad():
        output = model(image)

    # Get the predicted class with the highest score
    _, predicted_class = torch.max(output, 1)
    return predicted_class.item()  # Return the index as a Python int

# Load the model
model = load_model('fashion_mnist.pt')

# Load an image (ensure the image is in the correct format for the model)
image = Image.open('image.png')

# Make a prediction
predicted_class = predict(model, image)
print('Predicted class:', predicted_class)
```