

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	2
2	教学要求	2
3	实验要求	2
3.1	使用 Python 编程构建手动实现单隐层全连接神经网络	2
3.2	使用 PyTorch 库简洁实现全连接神经网络	3
3.3	提交要求	4
4	实验部分	4
4.1	实验环境	4
4.2	使用 Python 编程构建手动实现单隐层全连接神经网络	4
4.2.1	模型训练参数与可视化	4
4.2.2	模型评估与实验结果	5
4.2.3	参数分析实验	6
4.3	使用 PyTorch 库简洁实现全连接神经网络	7
4.3.1	模型训练参数与可视化	7
4.3.2	模型评估与实验结果	7
4.3.3	参数分析实验	9
5	实验总结	10
5.1	实验结果	10
5.2	实验感悟	10
5.3	实验的不足以及改进方向	10
	附录	11
A	使用 Python 编程构建手动实现单隐层全连接神经网络 Code	11
B	使用 PyTorch 库简洁实现全连接神经网络 Code	15
C	config.py	19
D	predict.py	20

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 $28 \times 28 = 784$ 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）

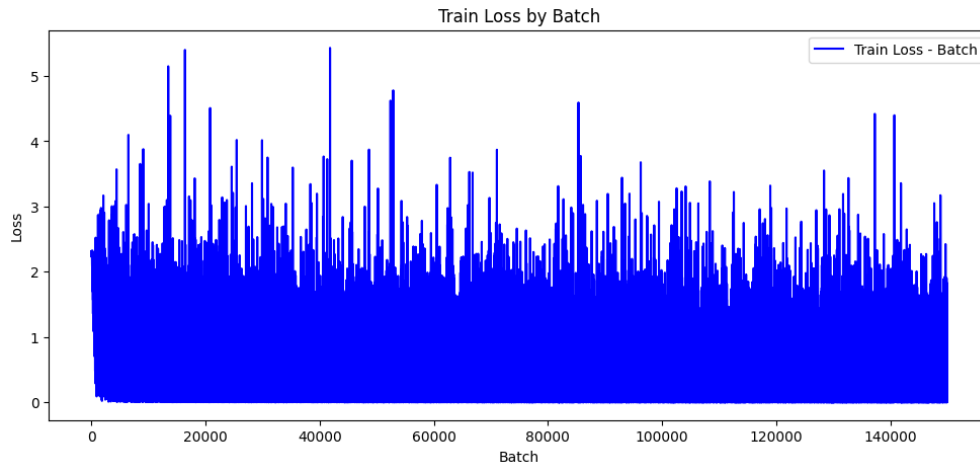


图 1: 训练集误差随 batch 变化的曲线图

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

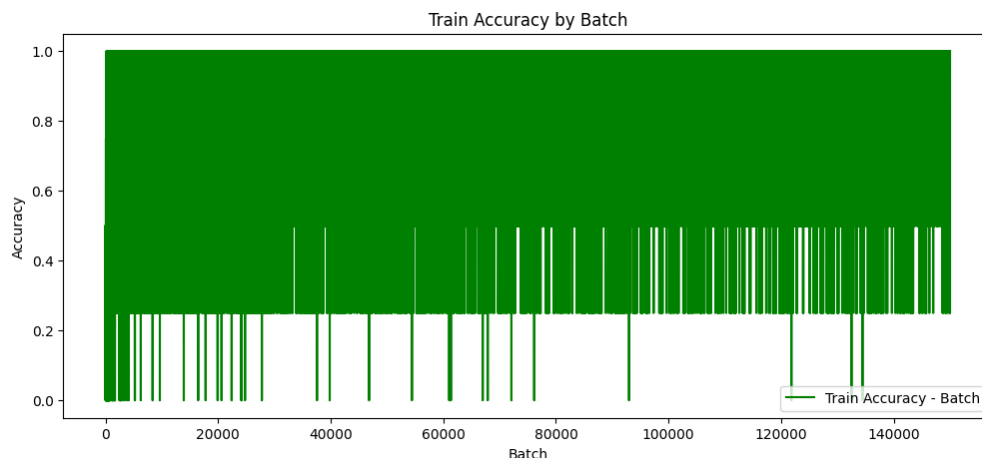


图 2: 训练集准确率随 batch 变化的曲线图

训练集和测试集上的准确率均高于 87%, 模型在训练集和测试集上的 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, 模型在测试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。

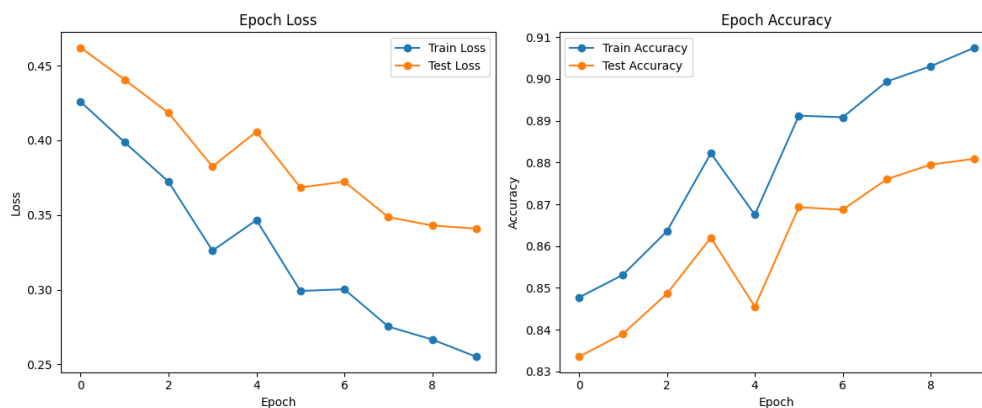


图 3: 训练集与验证集误差随 epoch 变化的曲线图

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。

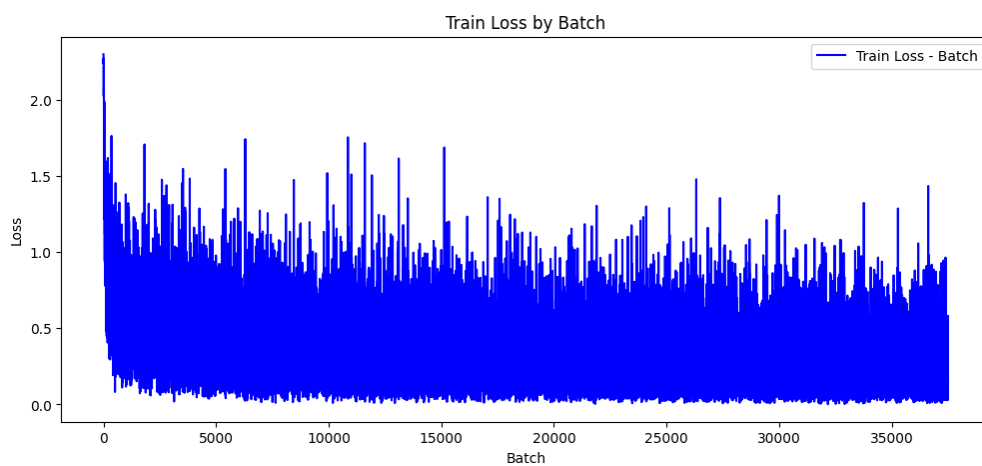


图 4: 训练集误差随 batch 变化的曲线图

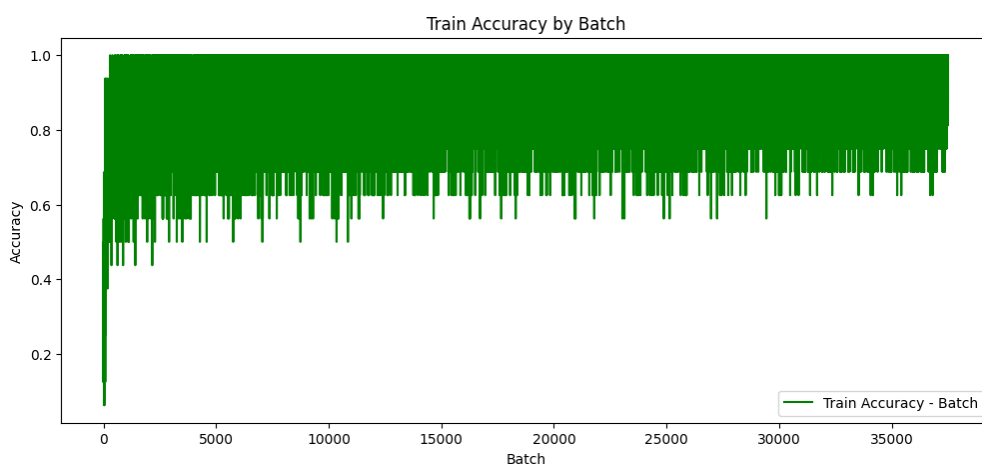


图 5: 训练集准确率随 batch 变化的曲线图

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。

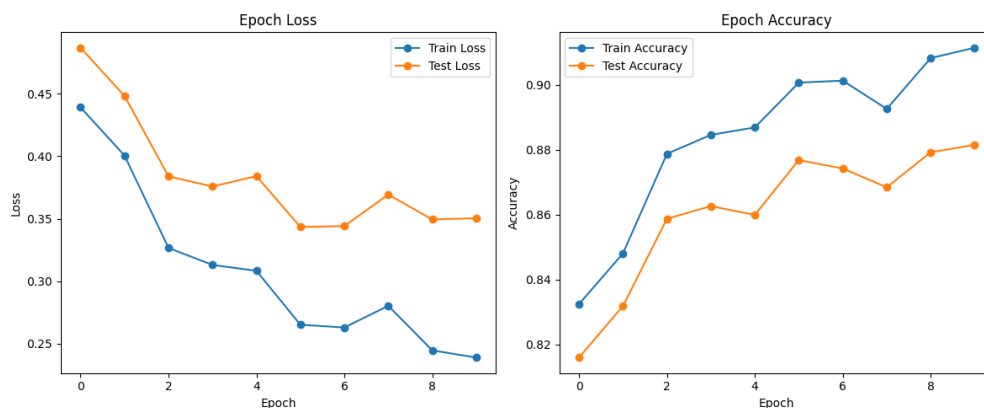


图 6: 训练集与验证集误差随 epoch 变化的曲线图

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。

```
1 # Initialize parameters
2 nn.init.normal_(self.fc1.weight, mean=0, std=0.1)
3 nn.init.constant_(self.fc1.bias, 0)
4 nn.init.normal_(self.fc2.weight, mean=0, std=0.1)
5 nn.init.constant_(self.fc2.bias, 0)
6
```

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

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from torchvision import datasets, transforms
4  from torch.utils.data import DataLoader
5  import config
6  args = config.args
7
8
9  # 数据加载
10 trans = transforms.ToTensor()
11 mnist_train = datasets.FashionMNIST(root="./data", train=True, transform=
    trans, download=True)
12 mnist_test = datasets.FashionMNIST(root="./data", train=False, transform=
    trans, download=True)
13
14 num_inputs, num_outputs, num_hiddens = 784, 10, 256
15
16
17 train_iter = DataLoader(mnist_train, batch_size=args.batch_size, shuffle=
    True, num_workers=0)
18 test_iter = DataLoader(mnist_test, batch_size=10000, shuffle=False,
    num_workers=0)
19
20 # 激活函数和softmax函数
21 def relu(x):
22     return np.maximum(x, 0)
23
24 def sigmoid(x):
25     return 1 / (1 + np.exp(-x))
26
27 def softmax(x):
28     exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
29     return exp_x / np.sum(exp_x, axis=1, keepdims=True)
30
31 # 损失函数
32 def cross_entropy(y_hat, y):
33     m = y_hat.shape[0]
```

```

34     y_one_hot = np.eye(num_outputs)[y]
35     return -np.sum(np.log(y_hat[range(m), y])) / m
36
37 # 前向传播
38 def model(X, w1, b1, w2, b2):
39     H = relu(np.dot(X, w1) + b1)
40     O = np.dot(H, w2) + b2
41     return softmax(O), H
42
43 # 梯度计算
44 def calc_gradient(X, y, y_hat, H, w2):
45     m = y.shape[0]
46     y_one_hot = np.eye(num_outputs)[y]
47
48     dO = (y_hat - y_one_hot) / m
49     grad_w2 = np.dot(H.T, dO)
50     grad_b2 = np.sum(dO, axis=0)
51
52     dH = np.dot(dO, w2.T) * (H > 0).astype(float)
53     grad_w1 = np.dot(X.T, dH)
54     grad_b1 = np.sum(dH, axis=0)
55
56     return grad_w1, grad_b1, grad_w2, grad_b2
57
58 # 参数更新
59 def update_params(params, grads, lr):
60     for param, grad in zip(params, grads):
61         param -= lr * grad
62
63 # 参数初始化
64 w1 = np.random.randn(num_inputs, num_hiddens) * 0.01
65 b1 = np.zeros(num_hiddens)
66 w2 = np.random.randn(num_hiddens, num_outputs) * 0.01
67 b2 = np.zeros(num_outputs)
68 params = [w1, b1, w2, b2]
69
70 # Metrics storage
71 batch_metrics = {
72     'train_losses': [],
73     'train_accs': [],
74     'test_losses': [],

```

```

75     'test_accs': []
76 }
77 epoch_metrics = {
78     'train_losses': [],
79     'train_accs': [],
80     'test_losses': [],
81     'test_accs': []
82 }
83 def accuracy(loader, w1, b1, w2, b2):
84     total_loss = 0
85     total_correct = 0
86     total_samples = 0
87
88     # 设置数据加载器加载的是numpy数组而非torch.Tensor
89     for images, labels in loader:
90         X = images.view(-1, num_inputs).numpy() # 转换为适当的形状和类型
91         y = labels.numpy()
92
93         # 前向传播
94         y_hat, _ = model(X, w1, b1, w2, b2)
95
96         # 计算损失
97         loss = cross_entropy(y_hat, y)
98         total_loss += loss * len(y) # 累计总损失
99         total_correct += (np.argmax(y_hat, axis=1) == y).sum() # 计算正确预
100         测的数量
101         total_samples += len(y)
102
103     average_loss = total_loss / total_samples
104     accuracy = total_correct / total_samples
105     return average_loss, accuracy
106
107 # Training and evaluation loop
108
109 for epoch in range(args.epochs):
110     for images, labels in train_iter:
111         X = images.view(-1, num_inputs).numpy()
112         y = labels.numpy()
113         y_hat, H = model(X, *params)
114         loss = cross_entropy(y_hat, y)
115         grads = calc_gradient(X, y, y_hat, H, w2)

```

```

115         update_params(params, grads, lr=args.lr)
116
117         batch_metrics['train_losses'].append(loss)
118         batch_acc = (np.argmax(y_hat, axis=1) == y).mean()
119         batch_metrics['train_accs'].append(batch_acc)
120
121     # Evaluate at the end of each epoch
122     epoch_train_loss, epoch_train_acc = accuracy(train_iter, *params)
123     epoch_test_loss, epoch_test_acc = accuracy(test_iter, *params)
124
125     epoch_metrics['train_losses'].append(epoch_train_loss)
126     epoch_metrics['train_accs'].append(epoch_train_acc)
127     epoch_metrics['test_losses'].append(epoch_test_loss)
128     epoch_metrics['test_accs'].append(epoch_test_acc)
129
130     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}')
131
132
133
134
135     # 绘制训练和测试损失与准确率的曲线图
136
137
138     # 绘制训练损失的曲线图
139     plt.figure(figsize=(12, 5))
140     plt.plot(batch_metrics['train_losses'], label='Train Loss - Batch', color='
blue')
141     plt.title('Train Loss by Batch')
142     plt.xlabel('Batch')
143     plt.ylabel('Loss')
144     plt.legend()
145     plt.show()
146
147
148     # 绘制训练准确率的曲线图
149     plt.figure(figsize=(12, 5))
150     plt.plot(batch_metrics['train_accs'], label='Train Accuracy - Batch', color=
'green')
151     plt.title('Train Accuracy by Batch')

```

```

152 plt.xlabel('Batch')
153 plt.ylabel('Accuracy')
154 plt.legend()
155 plt.show()
156
157
158 # 绘制训练和测试损失的曲线图
159 plt.figure(figsize=(12, 5))
160 plt.subplot(1, 2, 1)
161 plt.plot(epoch_metrics['train_losses'], label='Train Loss', marker='o')
162 plt.plot(epoch_metrics['test_losses'], label='Test Loss', marker='o')
163 plt.title('Epoch Loss')
164 plt.xlabel('Epoch')
165 plt.ylabel('Loss')
166 plt.legend()
167
168 # 绘制训练和测试准确率的曲线图
169 plt.subplot(1, 2, 2)
170 plt.plot(epoch_metrics['train_accs'], label='Train Accuracy', marker='o')
171 plt.plot(epoch_metrics['test_accs'], label='Test Accuracy', marker='o')
172 plt.title('Epoch Accuracy')
173 plt.xlabel('Epoch')
174 plt.ylabel('Accuracy')
175 plt.legend()
176
177 plt.tight_layout()
178 plt.show()

```

B 使用 PyTorch 库简洁实现全连接神经网络 Code

```

1  import torch
2  from torch import nn, optim
3  from torchvision import datasets, transforms
4  from torch.utils.data import DataLoader
5  import matplotlib.pyplot as plt
6  import config
7
8  args = config.args
9  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:

```

```

48         images = images.view(-1, 784).to(device)
49         labels = labels.to(device)
50         output = model(images)
51         loss = criterion(output, labels)
52         total_loss += loss.item()
53         total_correct += (output.argmax(1) == labels).sum().item()
54         total += labels.size(0)
55     model.train()
56     return total_loss / len(loader), total_correct / total
57
58 # Metrics storage
59 batch_metrics = {
60     'train_losses': [],
61     'train_accs': [],
62     'test_losses': [],
63     'test_accs': []
64 }
65 epoch_metrics = {
66     'train_losses': [],
67     'train_accs': [],
68     'test_losses': [],
69     'test_accs': []
70 }
71
72 # Training and evaluation loop
73 for epoch in range(args.epochs):
74     for images, labels in train_loader:
75         images = images.view(-1, 784).to(device)
76         labels = labels.to(device)
77         output = model(images)
78         loss = criterion(output, labels)
79         optimizer.zero_grad()
80         loss.backward()
81         optimizer.step()
82
83     # Record batch loss and accuracy
84     batch_loss = loss.item()
85     batch_accuracy = (output.argmax(1) == labels).float().mean().item()
86     batch_metrics['train_losses'].append(batch_loss)
87     batch_metrics['train_accs'].append(batch_accuracy)
88

```

```

89     # Evaluate on test data after each epoch
90     test_loss, test_accuracy = accuracy(test_loader)
91     batch_metrics['test_losses'].append(test_loss)
92     batch_metrics['test_accs'].append(test_accuracy)
93
94     # Record epoch metrics
95     epoch_train_loss, epoch_train_acc = accuracy(train_loader)
96     epoch_metrics['train_losses'].append(epoch_train_loss)
97     epoch_metrics['train_accs'].append(epoch_train_acc)
98     epoch_metrics['test_losses'].append(test_loss)
99     epoch_metrics['test_accs'].append(test_accuracy)
100
101     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}')
102
103     # Plotting code
104
105     # Plotting
106     # 绘制训练损失的曲线图
107     plt.figure(figsize=(6, 5))
108     plt.plot(batch_metrics['train_losses'], label='Train Loss - Batch', color='
blue')
109     plt.title('Train Loss by Batch')
110     plt.xlabel('Batch')
111     plt.ylabel('Loss')
112     plt.legend()
113     plt.show()
114
115
116     # 绘制训练准确率的曲线图
117     plt.figure(figsize=(6, 5))
118     plt.plot(batch_metrics['train_accs'], label='Train Accuracy - Batch', color=
'green')
119     plt.title('Train Accuracy by Batch')
120     plt.xlabel('Batch')
121     plt.ylabel('Accuracy')
122     plt.legend()
123     plt.show()
124
125

```

```

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

```

```

13 parser.add_argument('--dataset_path', default = './data/', help='fixed trainset
    root path')
14
15 parser.add_argument('--save', default='model', help='save path of trained model'
    )
16
17 parser.add_argument('--predict', default='model', help='save path of predict
    model')
18
19 parser.add_argument('--batch_size', type=list, default=256, help='batch size of
    trainset')
20
21 # train
22 parser.add_argument('--epochs', type=int, default=10, metavar='N', help='number
    of epochs to train (default: 10)')
23
24 parser.add_argument('--lr', type=float, default=0.1, metavar='LR', help='
    learning rate (default: 0.01)')
25
26 parser.add_argument('--momentum', type=float, default=0.99, metavar='M', help='
    SGD momentum (default: 0.5)')
27
28 parser.add_argument('--weight_decay', type=float, default=3e-5, metavar='W',
    help='SGD weight_decay (default: 3e-5)')
29
30 parser.add_argument('--nesterov', type=bool, default=True, help='SGD nesterov
    (default: True)')
31
32 parser.add_argument('--early-stop', default=20, type=int, help='early stopping
    (default: 20)')
33 #args = parser.parse_args()
34 args, unknown = parser.parse_known_args()

```

D predict.py