Experiment 2:

Image Classification

Yanming Guo

College of Systems Engineering

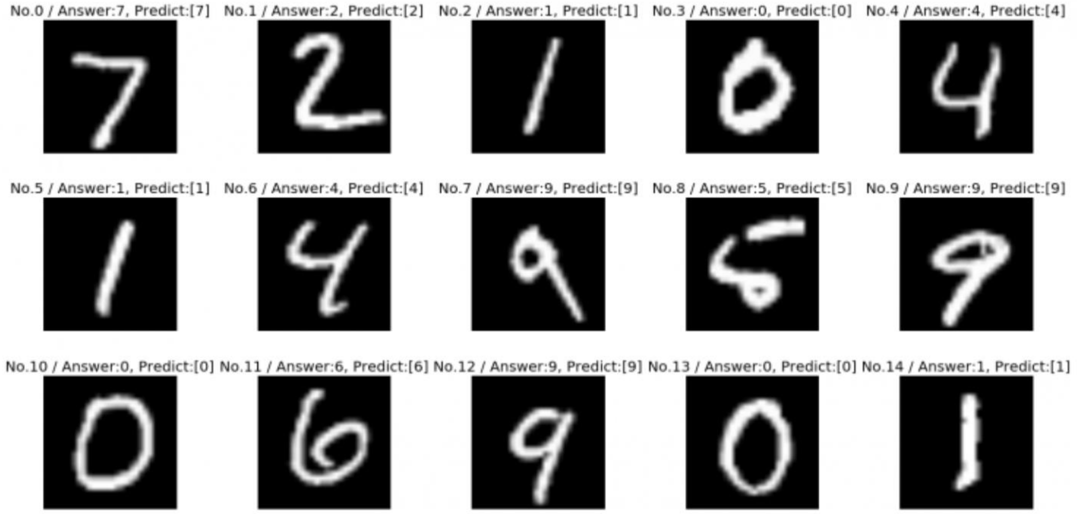
实验一回顾

- Pytorch初识（Tensor的构建和运算）
- 用Numpy实现两层神经网络（重点是对BP算法的理解）
- 用Tensor替换Numpy构建该神经网络 (Tensor)
- 用Tensor的autograd直接计算梯度（Tensor＋autograd）
- 利用nn库来构建网络（Tensor＋autograd＋nn）
- 利用optim来更新参数（Tensor＋autograd＋nn＋optim）
- 定义类来构建神经网络（Tensor＋autograd＋nn＋optim＋class）

实验内容

- 内容一：手写数字识别
  - ✓ torchvision自带数据集（MNIST）
  - ✓ 自己构建网络并进行训练

手写数字识别



No.0 / Answer:7, Predict:[7] No.1 / Answer:2, Predict:[2] No.2 / Answer:1, Predict:[1] No.3 / Answer:0, Predict:[0] No.4 / Answer:4, Predict:[4]

No.5 / Answer:1, Predict:[1] No.6 / Answer:4, Predict:[4] No.7 / Answer:9, Predict:[9] No.8 / Answer:5, Predict:[5] No.9 / Answer:9, Predict:[9]

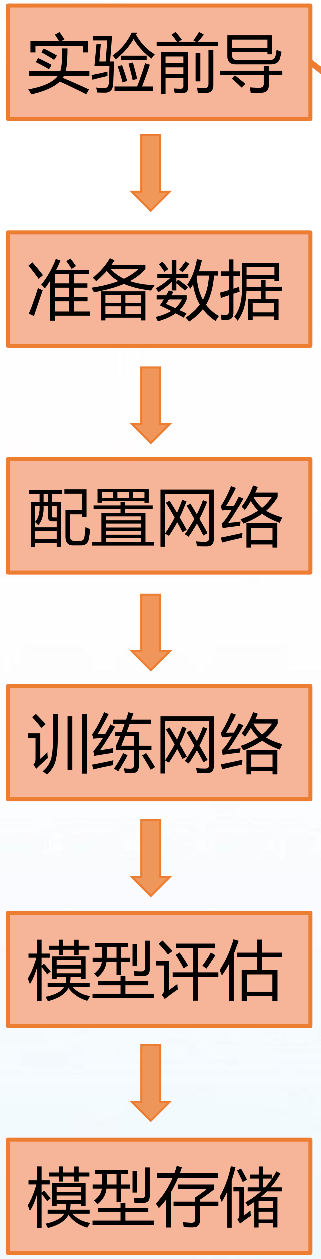
No.10 / Answer:0, Predict:[0] No.11 / Answer:6, Predict:[6] No.12 / Answer:9, Predict:[9] No.13 / Answer:0, Predict:[0] No.14 / Answer:1, Predict:[1]

- 内容二：蚂蚁(ant)和蜜蜂(bee)分类
  - ✓ 自定义数据集
  - ✓ 在已有模型基础上进行finetune

蚂蚁蜜蜂识别

实验目的：奠定利用Pytorch深度学习平台解决实际问题的基础

实验步骤



实验前导

准备数据

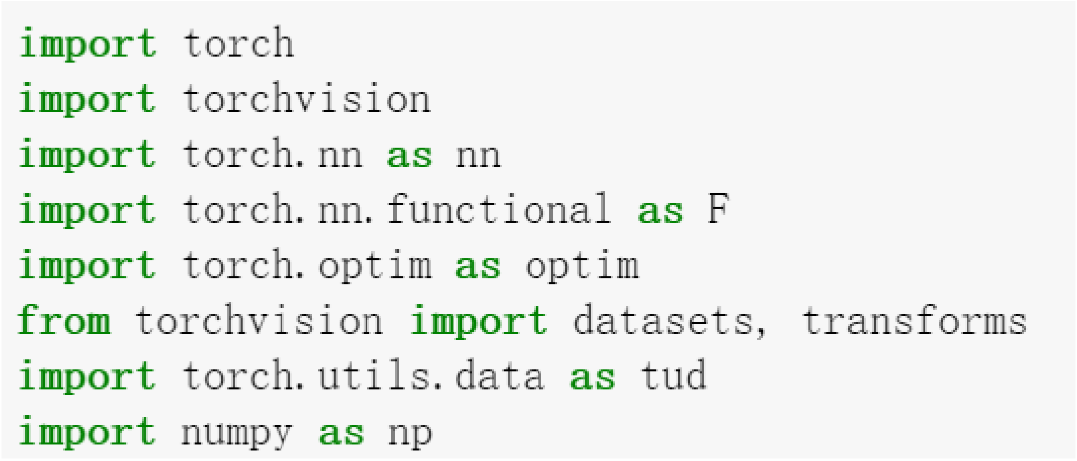
配置网络

训练网络

模型评估

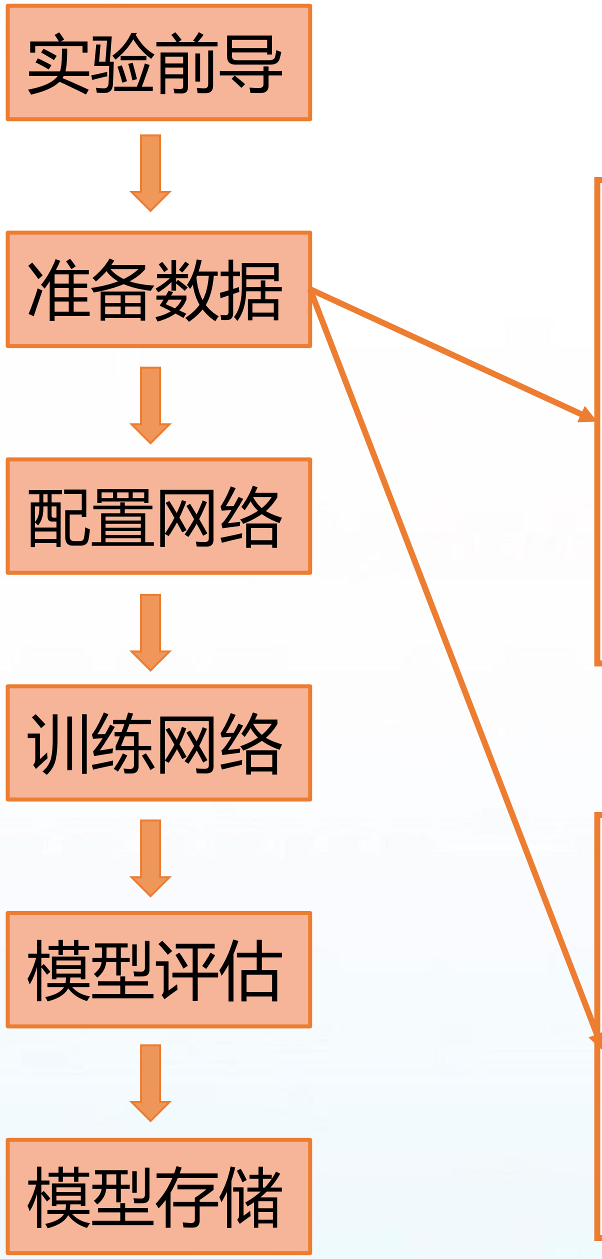
模型存储

导入必要的包



```python
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import torch.utils.data as tud
import numpy as np
```

实验步骤



实验前导

准备数据

配置网络

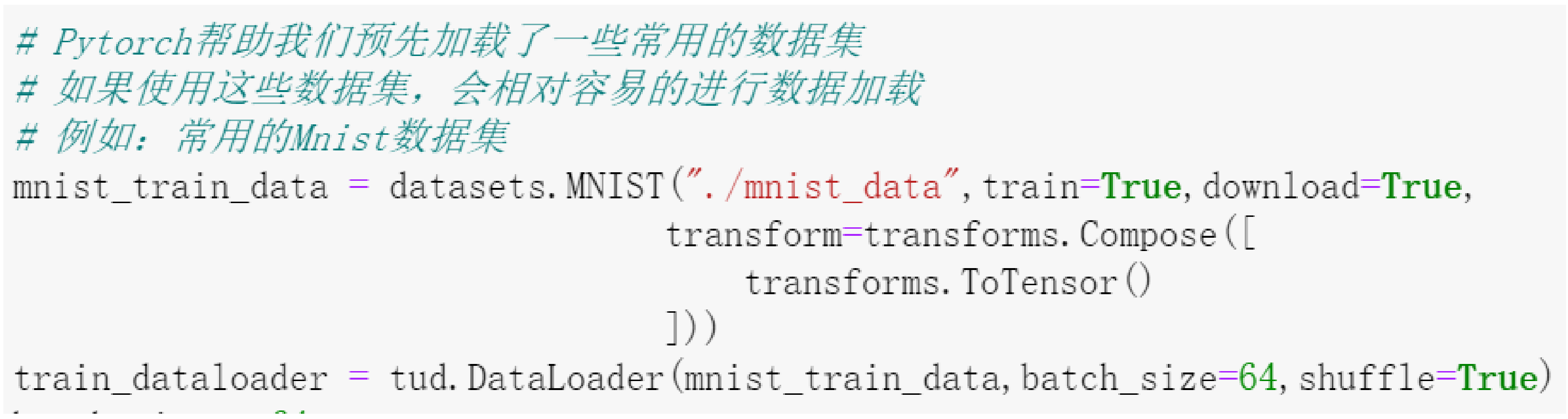
训练网络

模型评估

模型存储

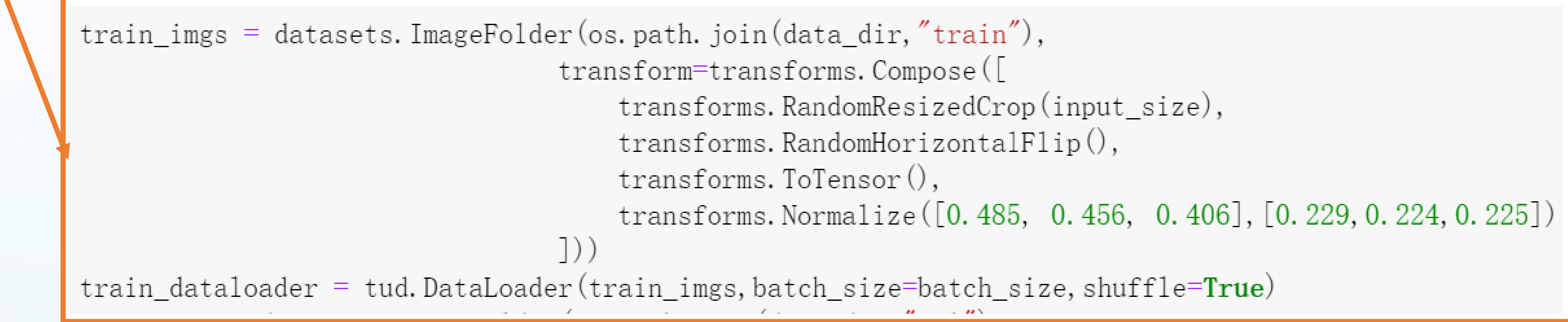
定义Dataset和Dataloader

torchvision数据集



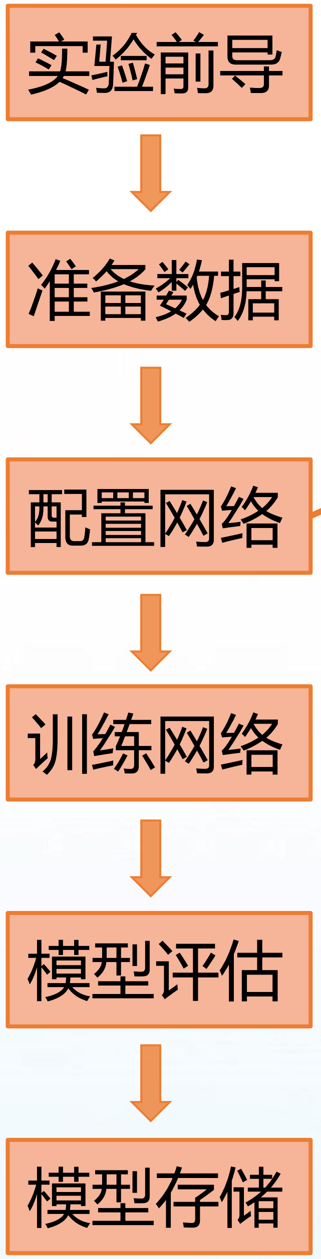
```
# Pytorch帮助我们预先加载了一些常用的数据集
# 如果使用这些数据集，会相对容易的进行数据加载
# 例如：常用的Mnist数据集
mnist_train_data = datasets.MNIST("./mnist_data",train=True,download=True,
                                  transform=transforms.Compose([
                                      transforms.ToTensor()
                                  ]))
train_dataloader = tud.DataLoader(mnist_train_data,batch_size=64,shuffle=True)
```

自定义数据集

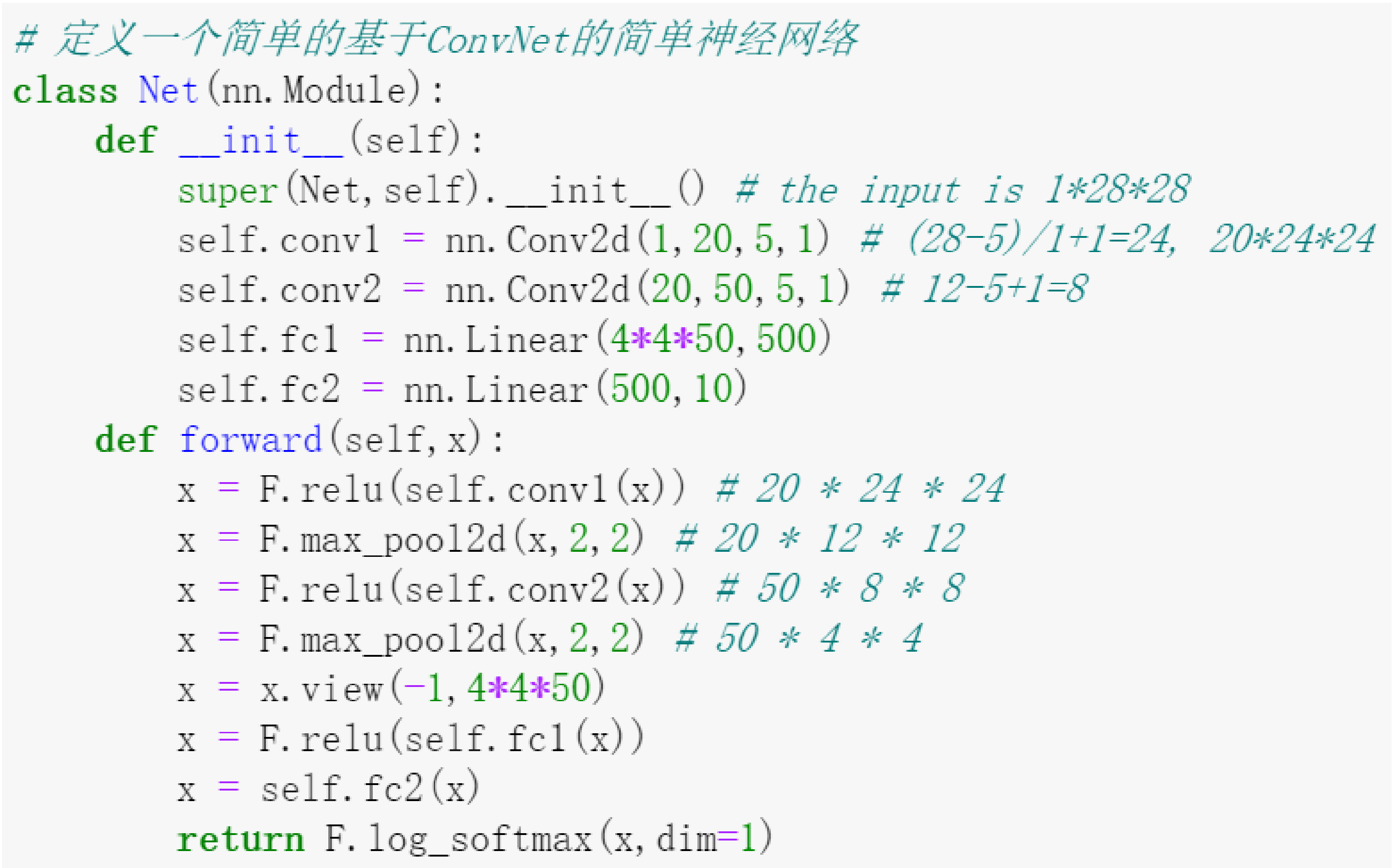


```
train_imgs = datasets.ImageFolder(os.path.join(data_dir,"train"),
                                  transform=transforms.Compose([
                                      transforms.RandomResizedCrop(input_size),
                                      transforms.RandomHorizontalFlip(),
                                      transforms.ToTensor(),
                                      transforms.Normalize([0.485, 0.456, 0.406],[0.229,0.224,0.225])
                                  ]))
train_dataloader = tud.DataLoader(train_imgs,batch_size=batch_size,shuffle=True)
```

实验步骤

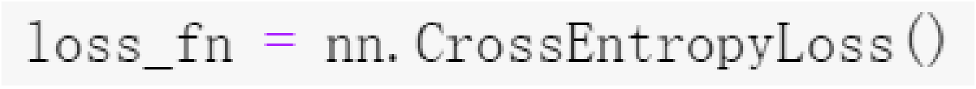


1. 定义网络



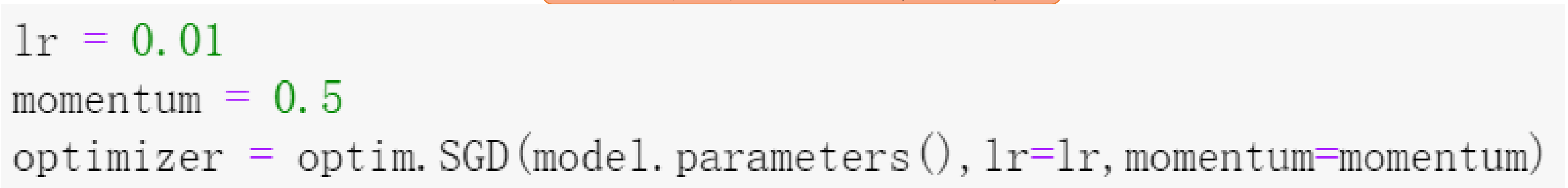
```
# 定义一个简单的基于ConvNet的简单神经网络
class Net(nn.Module):
    def __init__(self):
        super(Net,self).__init__() # the input is 1*28*28
        self.conv1 = nn.Conv2d(1,20,5,1) # (28-5)/1+1=24,  20*24*24
        self.conv2 = nn.Conv2d(20,50,5,1) # 12-5+1=8
        self.fc1 = nn.Linear(4*4*50,500)
        self.fc2 = nn.Linear(500,10)
    def forward(self,x):
        x = F.relu(self.conv1(x)) # 20 * 24 * 24
        x = F.max_pool2d(x,2,2) # 20 * 12 * 12
        x = F.relu(self.conv2(x)) # 50 * 8 * 8
        x = F.max_pool2d(x,2,2) # 50 * 4 * 4
        x = x.view(-1,4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x,dim=1)
```

2. 定义损失函数



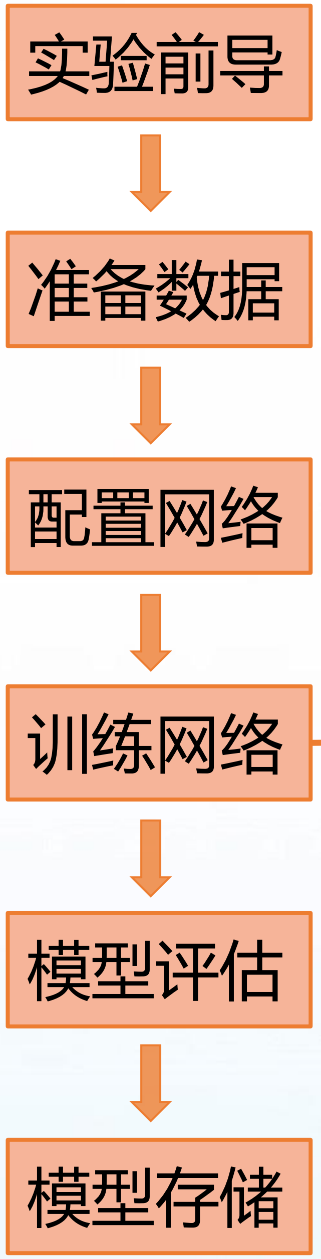
```
loss_fn = nn.CrossEntropyLoss()
```

3. 定义优化算法

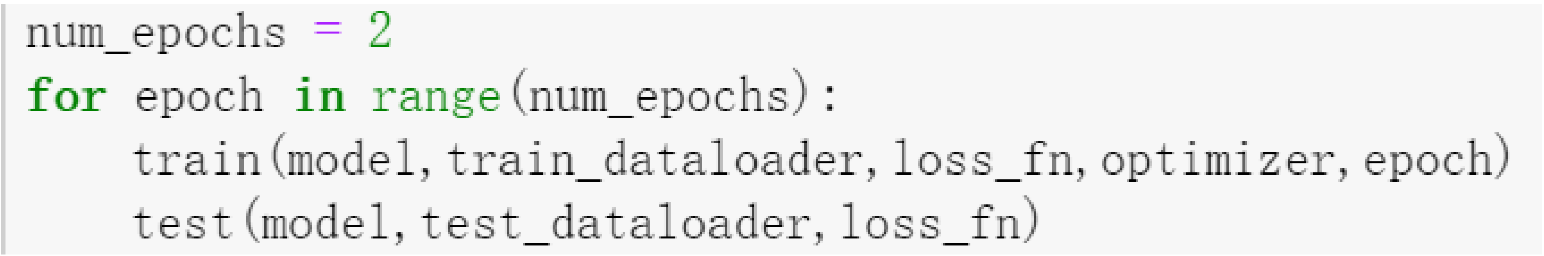


```
lr = 0.01
momentum = 0.5
optimizer = optim.SGD(model.parameters(),lr=lr,momentum=momentum)
```

实验步骤

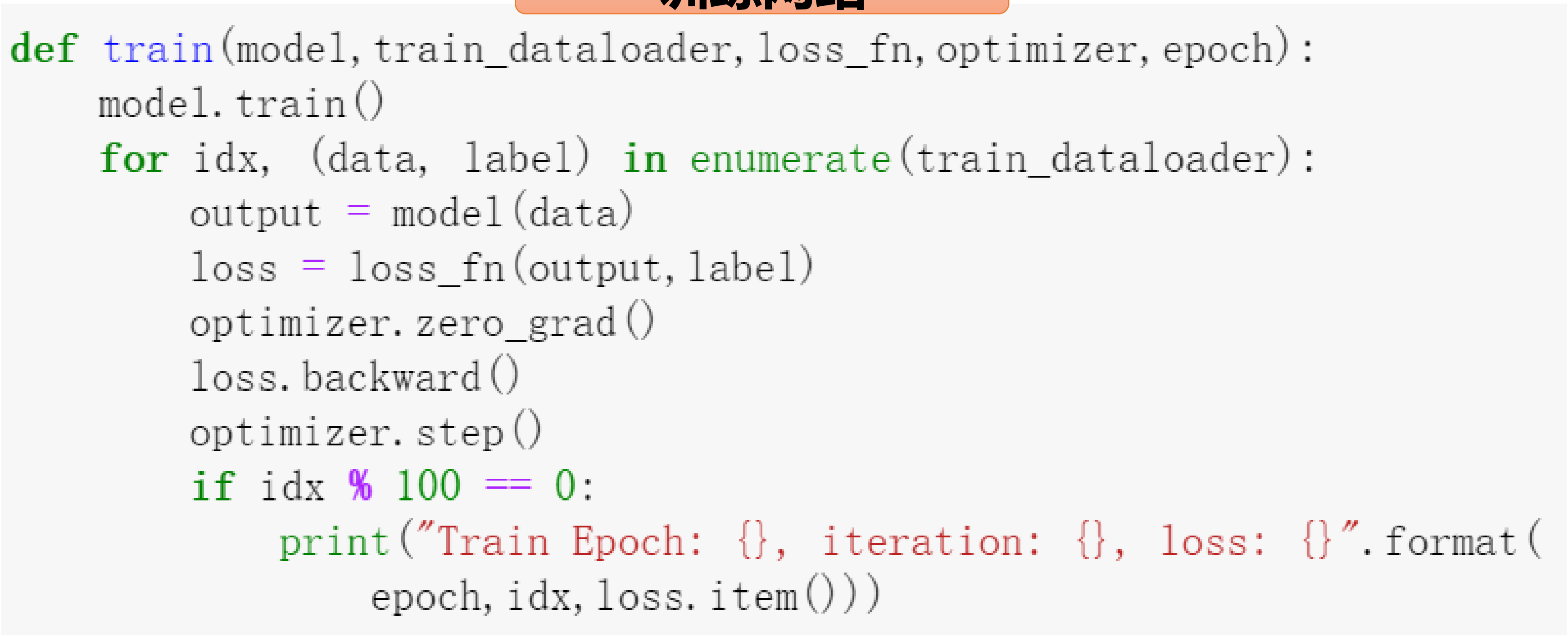


主函数



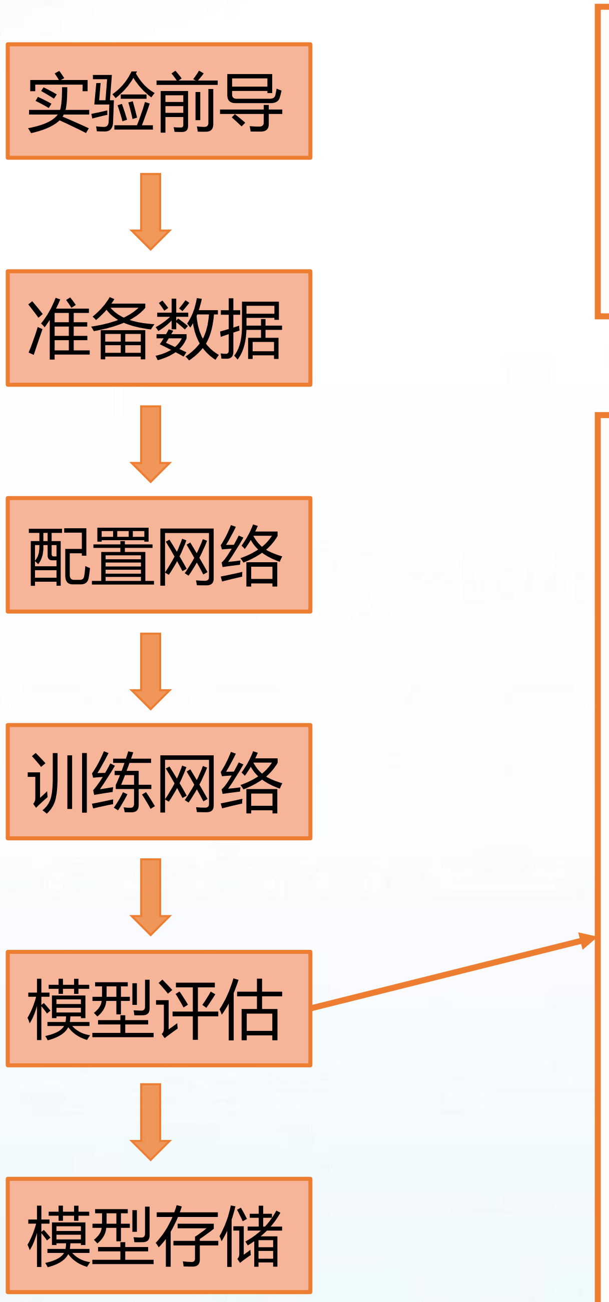
```python
num_epochs = 2
for epoch in range(num_epochs):
    train(model, train_dataloader, loss_fn, optimizer, epoch)
    test(model, test_dataloader, loss_fn)
```

训练网络

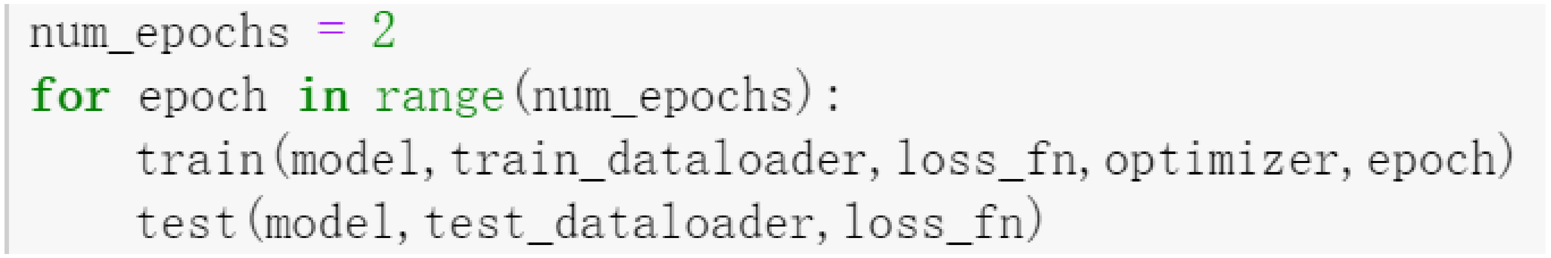


```python
def train(model, train_dataloader, loss_fn, optimizer, epoch):
    model.train()
    for idx, (data, label) in enumerate(train_dataloader):
        output = model(data)
        loss = loss_fn(output, label)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if idx % 100 == 0:
            print("Train Epoch: {}, iteration: {}, loss: {}".format(
                epoch, idx, loss.item()))
```

实验步骤

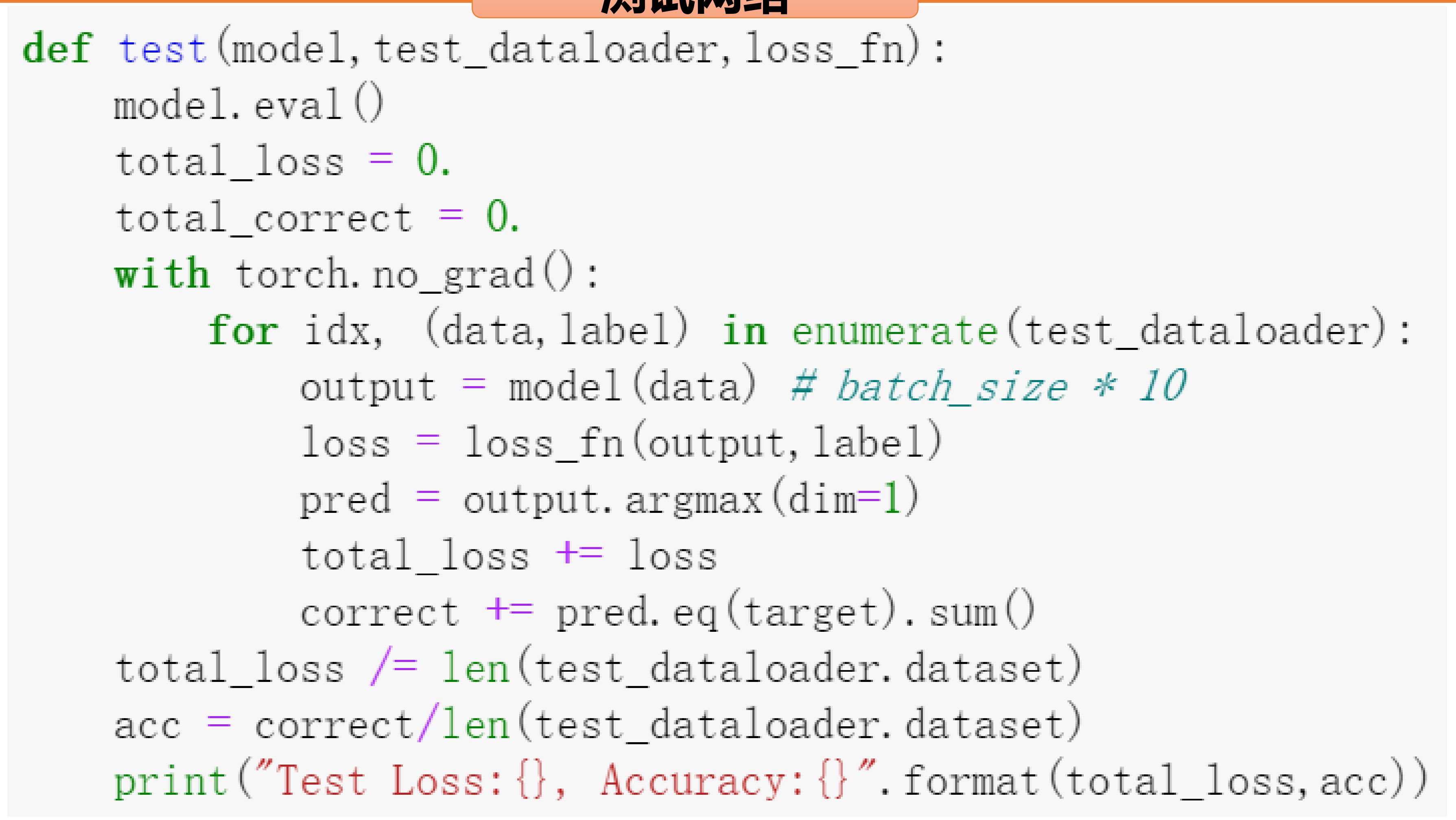


主函数



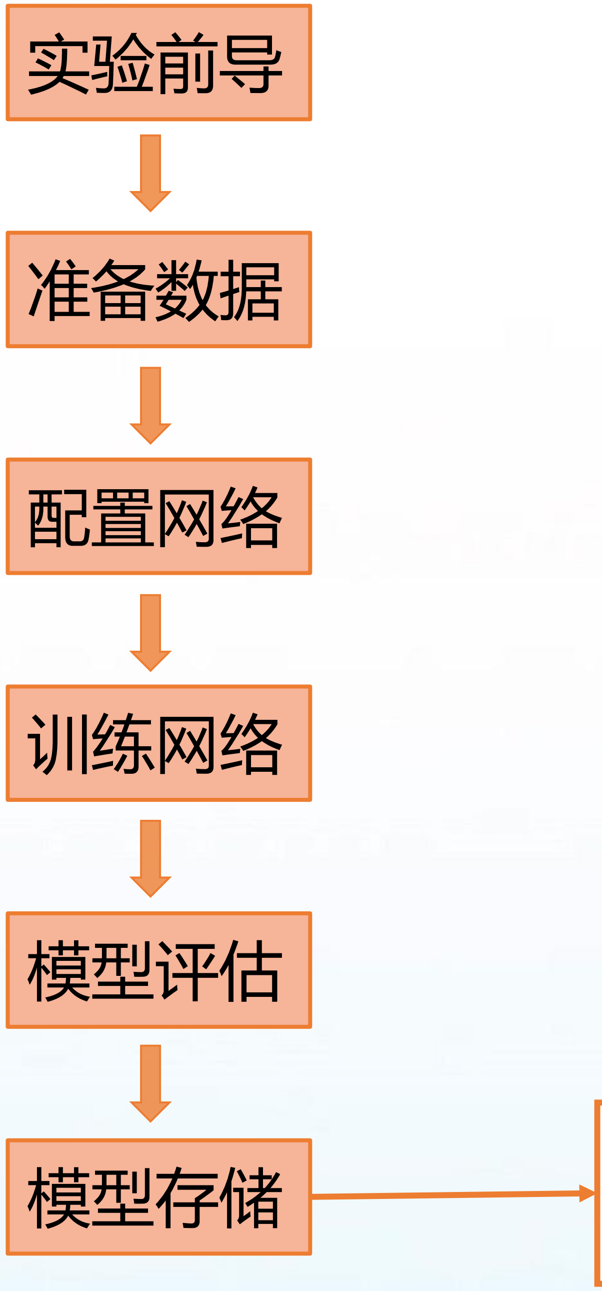
```
num_epochs = 2
for epoch in range(num_epochs):
    train(model, train_dataloader, loss_fn, optimizer, epoch)
    test(model, test_dataloader, loss_fn)
```

测试网络



```
def test(model, test_dataloader, loss_fn):
    model.eval()
    total_loss = 0.
    total_correct = 0.
    with torch.no_grad():
        for idx, (data, label) in enumerate(test_dataloader):
            output = model(data) # batch_size * 10
            loss = loss_fn(output, label)
            pred = output.argmax(dim=1)
            total_loss += loss
            correct += pred.eq(target).sum()
    total_loss /= len(test_dataloader.dataset)
    acc = correct/len(test_dataloader.dataset)
    print("Test Loss:{}, Accuracy:{}".format(total_loss, acc))
```

实验步骤



实验前导

准备数据

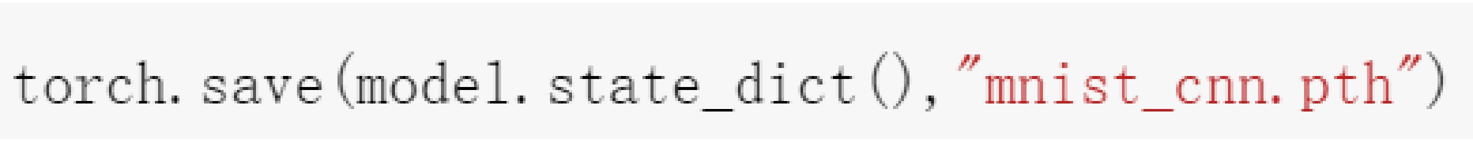
配置网络

训练网络

模型评估

模型存储

存储模型



```
torch.save(model.state_dict(),"mnist_cnn.pth")
```