OPTIMIZATION

一、内容

优化是调整模型参数以减少每个训练步骤中的模型误差的过程。优化算法定义了这个过程是如何进行的（在这个例子中，使用随机梯度下降）。所有的优化逻辑都封装在优化器对象中。在这里使用SGD优化器；PyTorch中还有许多不同的优化器，如ADAM和RMSProp，它们对不同类型的模型和数据更有效。

二、代码

在训练循环中，优化分为三个步骤：

- 调用 optimizer.zero_grad () 来重置模型参数的梯度。梯度默认是累加的；为了防止重复计算，我们在每次迭代时明确地将它们归零。
- 用 loss.backward () 反向传播预测损失。PyTorch会将每个参数的损失梯度存储起来。
- 有了梯度，就调用 optimizer.step () 来根据反向传播过程中收集到的梯度调整参数。

1、加载数据

```
class myDataset(Dataset):
    def __init__(self, *, root, train_data=True, transform=None, target_transform=None):
        self.root = root
        self.path = os.path.join(self.root, "train" if train_data is True else "test")
        # transform和target_transform是对数据和标签的预处理方法
        self.transform = transform
        self.target_transform = target_transform
        # 参过函数load_csv获得数据的路径和标签
        self.X, self.y = self.load_csv('save_path.csv')

    def __len__(self):
        return len(self.X)

    def __getitem__(self, index):
        x, label = self.X[index], self.y[index]
```

```python
        x = Image.open(x)
        # 必须是tensor类型数据才能使用batch
        # x要先转成np才能转成tensor
        x = np.array(x)
        x = torch.tensor(x).float()
        label = torch.tensor(label)
        # 并在这里对数据预处理
        if self.transform is not None:
            x = self.transform(x)
        if self.target_transform is not None:
            label = self.target_transform(label)

        return x, label

    def load_csv(self, savepath):
        if not os.path.exists(os.path.join( self.path, savepath)):
            data_X = []
            data_y = []
            # 如果该路径下的文件不存在就创建一个
            # 因为每类数据都存在各自的文件夹下，所以要对每个数据生成唯一的路径方便索引
            # 在数据集中，分为多个类别，每个类别的数据存在以类别命名的文件夹中,把路径集中起来
            labels = os.listdir(self.path)
            # 遍历每个类
            for label in labels:
                files = os.listdir(os.path.join(self.path, label))
                # 遍历类中的每个文件
                for file in files:
                    data_X.append(os.path.join(self.path, label, file))
                    data_y.append(label)

            # 打乱顺序
            key = np.array(range(len(data_X)))
            np.random.shuffle(key)
            data_X = np.array(data_X)
            data_y = np.array(data_y)
            # 变成np型式才不会报错
            data_X = data_X[key]
            data_y = data_y[key]

            # 保存成文件，方便直接从路径索引
            with open(os.path.join(self.path, savepath), mode='w', newline='') as f:
                # mode='w' 参数指定以写入模式打开文件，如果文件已经存在，则覆盖原有内容。

                # newline='' 参数指定在写入文件时不添加额外的换行符。
                writer = csv.writer(f)
                for X, y in zip(data_X,data_y):
                        writer.writerow([X, y]) #
fashion_mnist_images\train\7\3151.png,7

        X = []
        y = []
        # 从保存文件中读数据
        with open(os.path.join(self.path, savepath)) as f:
            reader = csv.reader(f)
            for row in reader:
                x, label = row
                label = int(label)
                X.append(x)
                y.append(label)

        # 返回数据的路径和标签
        return X, y
```

> 这里使用了Datasets and DataLoaders部所用的载数据类

2、epoch

优化循环来训练和优化模型。优化循环的每次迭代称为一个epoch。

每个epoch包括两个主要部分：

- Train Loop（训练循环） - 遍历训练数据集，试图收敛到最优参数。
- Validation/Test Loop（验证/测试循环） - 遍历测试数据集，检查模型性能是否有改善。

```
def train_loop(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    # 当使用batch normalization and dropout layers，一定要使用model.train()
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()


        if batch % 100 == 0:
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

```
def test_loop(dataloader, model, loss_fn):
    # 当使用batch normalization and dropout layers，一定要使用model.eval()
    # 因为测试集上不希望使用batch normalization累积归一化的值，也不希望使用dropout
layers,
    # 并且这两种神经网络层在前向传播过程中就会进行更新，不在反向传播更新
    # model.eval()将不会行更新这两种神经网络层
    model.eval()
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    test_loss, correct = 0, 0

    # Evaluating过程中使用torch.no_grad()确保不计算梯度
    with torch.no_grad():
        for X, y in dataloader:
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f}
\n")
```

当模型依赖于torch.nn.Dropout和torch.nn.BatchNorm2d等模块，这些模块可能根据训练模式而表现不同，例如，为了避免在验证数据上更新BatchNorm运行统计信息，则调用model.eval()和model.train()。在训练时始终使用model.train()，在评估模型（验证/测试）时使用model.eval()，即使不确定模型是否具有训练模式特定的行为，因为使用的某个模块可能会被更新以在训练和评估模式下表现不同。

3、Loss Function和Optimizer

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
```

> 这里使用交叉熵损失和SGD优化器。

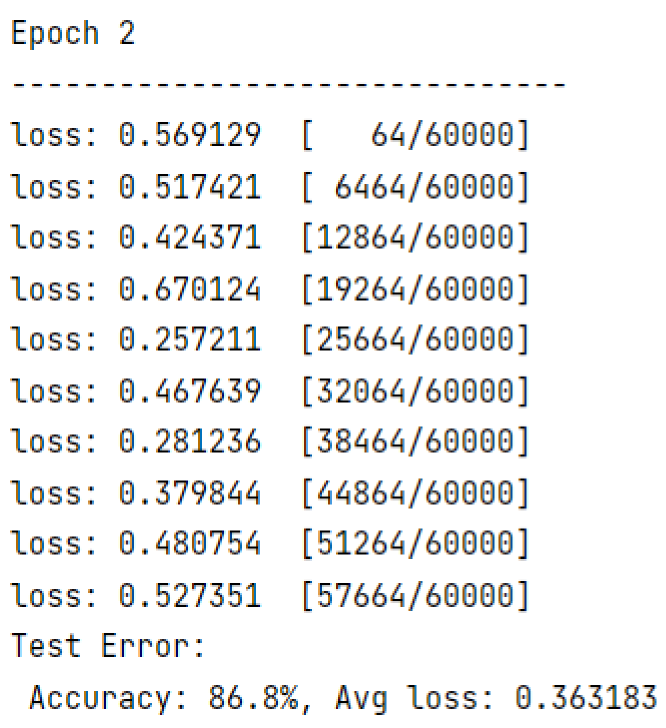
模型依赖于torch.nn.Dropout和torch.nn.BatchNorm2d等模块，这些模块可能根据训练模式而表现不同，例如，为了避免在验证数据上更新BatchNorm运行统计信息，您有责任调用model.eval()和model.train()。

建议您在训练时始终使用model.train()，在评估模型（验证/测试）时使用model.eval()，即使您不确定模型是否具有训练模式特定的行为，因为您使用的某个模块可能会被更新以在训练和评估模式下表现不同。

4、完整代码

```
import csv
import os.path

import numpy as np
import torch
from PIL import Image
from torch import nn
from torch.utils.data import Dataset, DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt


class myDataset(Dataset):
    def __init__(self, *, root, train_data=True, transform=None, target_transform=None):
        self.root = root
        self.path = os.path.join(self.root, "train" if train_data is True else "test")
        # transform和target_transform是对数据和标签的预处理方法
        self.transform = transform
        self.target_transform = target_transform
        # 参过函数load_csv获得数据的路径和标签
        self.X, self.y = self.load_csv('save_path.csv')

    def __len__(self):
        return len(self.X)

    def __getitem__(self, index):
        x, label = self.X[index], self.y[index]

        x = Image.open(x)
        # 必须是tensor类型数据才能使用batch
        # x要先转成np才能转成tensor
        x = np.array(x)
        x = torch.tensor(x).float()
        label = torch.tensor(label)
        # 并在这里对数据预处理
        if self.transform is not None:
            x = self.transform(x)
        if self.target_transform is not None:
            label = self.target_transform(label)

        return x, label

    def load_csv(self, savepath):
        if not os.path.exists(os.path.join( self.path, savepath)):
            data_X = []
            data_y = []
            # 如果该路径下的文件不存在就创建一个
            # 因为每类数据都存在各自的文件夹下，所以要对每个数据生成唯一的路径方便索
引
            # 在数据集中，分为多个类别，每个类别的数据存在以类别命名的文件夹中,把路
径集中起来
            labels = os.listdir(self.path)
            # 遍历每个类
            for label in labels:
                files = os.listdir(os.path.join(self.path, label))
                # 遍历类中的每个文件
                for file in files:
                    data_X.append(os.path.join(self.path, label, file))
                    data_y.append(label)

            # 打乱顺序
            key = np.array(range(len(data_X)))
            np.random.shuffle(key)
            data_X = np.array(data_X)
            data_y = np.array(data_y)
            # 变成np型式才不会报错
            data_X = data_X[key]
            data_y = data_y[key]

            # 保存成文件，方便直接从路径索引
            with open(os.path.join(self.path, savepath), mode='w', newline='') as f:
                # mode='w'参数指定以写入模式打开文件，如果文件已经存在，则覆盖原有
内容。
                # newline=''参数指定在写入文件时不添加额外的换行符。
                writer = csv.writer(f)
                for X, y in zip(data_X,data_y):
                    writer.writerow([X, y]) #
fashion_mnist_images\train\7\3151.png,7

        X = []
        y = []
        # 从保存文件中读数据
        with open(os.path.join(self.path, savepath)) as f:
            reader = csv.reader(f)
            for row in reader:
                x, label = row
                label = int(label)
                X.append(x)
                y.append(label)

        # 返回数据的路径和标签
        return X, y



def train_loop(dataloader, model, loss_fn, optimizer):
```

```python
    size = len(dataloader.dataset)
    # 当使用batch normalization and dropout layers，一定要使用model.train()
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()


        if batch % 100 == 0:
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")


def test_loop(dataloader, model, loss_fn):
    # 当使用batch normalization and dropout layers，一定要使用model.eval()
    # 因为测试集上不希望使用batch normalization累积归一化的值，也不希望使用dropout
layers,
    # 并且这两种神经网络层在前向传播过程中就会进行更新，不在反向传播更新
    # model.eval()将不会行更新这两种神经网络层
    model.eval()
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    test_loss, correct = 0, 0

    # Evaluating过程中使用torch.no_grad()确保不计算梯度
    with torch.no_grad():
        for X, y in dataloader:
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f}
\n")

class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork()

trainData = myDataset(root='fashion_mnist_images', train_data=True)
train_dataloader = DataLoader(trainData, batch_size=64, shuffle=True)

testData = myDataset(root='fashion_mnist_images', train_data=False)
test_dataloader = DataLoader(trainData, batch_size=64, shuffle=True)


loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)

epochs = 2
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader, model, loss_fn, optimizer)
    test_loop(test_dataloader, model, loss_fn)
print("Done!")
```



```
Epoch 2
-------------------------------
loss: 0.569129  [   64/60000]
loss: 0.517421  [ 6464/60000]
loss: 0.424371  [12864/60000]
loss: 0.670124  [19264/60000]
loss: 0.257211  [25664/60000]
loss: 0.467639  [32064/60000]
loss: 0.281236  [38464/60000]
loss: 0.379844  [44864/60000]
loss: 0.480754  [51264/60000]
loss: 0.527351  [57664/60000]
Test Error:
 Accuracy: 86.8%, Avg loss: 0.363183
```