homework10

November 3, 2024

[ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader

from torchvision import datasets, transforms

from collections import Counter
```

[ ]:
```
torch.cuda.is_available()
```

[ ]:
```
True
```

[ ]:
```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

fashion_train = datasets.FashionMNIST(root='./data', train=True, download=True,␣
 ↪transform=transform)
fashion_test = datasets.FashionMNIST(root='./data', train=False, download=True,␣
 ↪transform=transform)

train_loader = DataLoader(fashion_train, batch_size=64, shuffle=True)
test_loader = DataLoader(fashion_test, batch_size=64, shuffle=False)
```

[ ]:
```
class CNNModel1(nn.Module):
    def __init__(self, num_classes=10):
        super(CNNModel1, self).__init__()

        self.conv1 = nn.Conv2d(1, 32, 5)
        self.conv2 = nn.Conv2d(32, 64, 5)
        self.pool = nn.MaxPool2d(2)

        self.fc1 = nn.Linear(64 * 4 * 4, 128)
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(128, num_classes)
```

```
    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))

        x = x.view(-1, 64 * 4 * 4)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)

        return x

class CNNModel2(nn.Module):
    def __init__(self, num_classes=10):
        super(CNNModel2, self).__init__()

        self.conv1 = nn.Conv2d(1, 16, 3)
        self.conv2 = nn.Conv2d(16, 32, 3)
        self.pool = nn.MaxPool2d(2)

        self.fc1 = nn.Linear(32 * 5 * 5, 64)
        # self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(64, num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))

        x = x.view(-1, 32 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)

        return x

class CNNModel3(nn.Module):
    def __init__(self, num_classes=10):
        super(CNNModel3, self).__init__()

        self.conv1 = nn.Conv2d(1, 32, 3)
        self.conv2 = nn.Conv2d(32, 64, 3)
        self.pool = nn.MaxPool2d(2)

        self.fc1 = nn.Linear(64 * 5 * 5, 128)
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
```

```
        x = self.pool(F.relu(self.conv2(x)))

        x = x.view(-1, 64 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)

        return x
```

[ ]:
```
def train_model(model, device, train_loader, optimizer, criterion, epochs=5):
    model.to(device)
    model.train()
    for epoch in range(epochs):
        for data, target in train_loader:
            data, target = data.to(device), target.to(device)

            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target)

            loss.backward()
            optimizer.step()

def test_model(model, device, test_loader):
    model.to(device)
    model.eval()
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
    accuracy = 100. * correct / len(test_loader.dataset)
    return accuracy

def majority_vote(models, device, test_loader):
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            predictions = []
            for model in models:
                output = model(data)
                pred = output.argmax(dim=1)
                predictions.append(pred.cpu().numpy())
```

```
            for i in range(len(target)):
                votes = [pred[i] for pred in predictions]
                majority_vote = Counter(votes).most_common(1)[0][0]
                if majority_vote == target[i].item():
                    correct += 1
    accuracy = 100. * correct / len(test_loader.dataset)
    return accuracy
```

[ ]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model1 = CNNModel1().to(device)
model2 = CNNModel2().to(device)
model3 = CNNModel3().to(device)

criterion = nn.CrossEntropyLoss()
optimizer1 = torch.optim.Adam(model1.parameters(), lr=0.001)
optimizer2 = torch.optim.Adam(model2.parameters(), lr=0.001)
optimizer3 = torch.optim.Adam(model3.parameters(), lr=0.001)

train_model(model1, device, train_loader, optimizer1, criterion)
train_model(model2, device, train_loader, optimizer2, criterion)
train_model(model3, device, train_loader, optimizer3, criterion)

accuracy1 = test_model(model1, device, test_loader)
accuracy2 = test_model(model2, device, test_loader)
accuracy3 = test_model(model3, device, test_loader)

print(f'Model 1 Test Accuracy: {accuracy1:.2f}%')
print(f'Model 2 Test Accuracy: {accuracy2:.2f}%')
print(f'Model 3 Test Accuracy: {accuracy3:.2f}%')

ensemble_accuracy = majority_vote([model1, model2, model3], device, test_loader)
print(f'Ensemble Majority Vote Test Accuracy: {ensemble_accuracy:.2f}%')
```

```
Model 1 Test Accuracy: 90.42%
Model 2 Test Accuracy: 89.16%
Model 3 Test Accuracy: 90.33%
Ensemble Majority Vote Test Accuracy: 90.78%
```

[ ]:
```
class CNNModel(nn.Module):
    def __init__(self, num_classes=10):
        super(CNNModel, self).__init__()

        self.conv1 = nn.Conv2d(1, 32, kernel_size=3) # 28x28x1 -> 26x26x32 / 2
 ↪= 13x13x32
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)# 11x11x64 -> 5x5x64
        self.pool = nn.MaxPool2d(2)
```

```
        self.fc1 = nn.Linear(64 * 5 * 5, 128)
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))

        x = x.view(-1, 64 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)

        return x
```

[ ]:
```
baseline_model = CNNModel().to(device)
optimizer_baseline = torch.optim.Adam(baseline_model.parameters(), lr=0.001)
train_model(baseline_model, device, train_loader, optimizer_baseline, criterion)
baseline_accuracy = test_model(baseline_model, device, test_loader)
print(f'Single Model Baseline Test Accuracy: {baseline_accuracy:.2f}%')
```