计算机视觉实验1

曾世鹏 _ U202115574 _CS2108

一、实验需求分析

1. 读取csv文件
2. 按9:1随机划分train和test，构建train_loader和test_loader
3. 构建fnn，需求为至少有一层隐藏层

二、实验环境介绍

1. 使用pytorch框架
2. 使用pandas和numpy包

三、参数介绍

- 超参数设置

```
# setting
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
batch_size = 32
epochs = 100
learning_rate = 0.0005
```

- 网络、损失函数和优化器设置

```
# using softmax
network_type = "softmax"
# using which optimizer
optimizer_type = "Adam"
# using which loss function
loss_fn_type = "CrossEntropyLoss"
```

- 测试用print控制

```
# print result
noisy = True
# print train info
print_info = True
```

四、实现思路

1. 用pd的read_csv读取数据集
2. 用pd的sample随机取90%的训练集，剩下的作为测试集
3. 将训练集和测试集转为numpy格式后用torch的dataloader构建train_loader和test_loader
4. 构建fnn
    - 网络设计：数据集有2个维度，输出结果为四种可能，将输出结果用独热向量表示，则输入维度为2，输出维度为4，中间隐藏层测试多种方案

- 激活函数：使前面使用relu激活函数加快训练速度，最后输出层使用softmax便于归一化。

5. 实例化网络，优化器和损失函数
6. 训练并保存模型
7. 读取模型并测试准确度

五、具体实现

5.1 构建数据集

1. 读取csv文件

```
# read dataset
dataset = pd.read_csv("./dataset.csv")
```

2. 划分数据集

```
# split dataset
train_set = dataset.sample(frac=0.9, random_state=0)
test_set = dataset.drop(train_set.index
```

3. 构建dataloader

```
# convert dataset to numpy
train_set = train_set.to_numpy()
test_set = test_set.to_numpy()

# construct dataloader
train_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size,
shuffle=True)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=400,
shuffle=True)
```

5.2 网络选择与结果

5.2.1 非输出层激活函数使用relu，输出层使用softmax,输入维度为2，输出维度为4

1.两层隐藏层，神经元数均为10

网络定义：

```
def __init__(self):
    super(FNN1, self).__init__()
    self.fc1 = torch.nn.Linear(2, 10)
    self.fc2 = torch.nn.Linear(10, 10)
    self.fc3 = torch.nn.Linear(10, 4)
```

结果：

```
epoch: 99/100,loss: 0.7584573030471802
accuracy: 0.94
```

2.两层隐藏层，神经元数为8,16

网络定义：

```
def __init__(self):
    super(FNN1, self).__init__()
    self.fc1 = torch.nn.Linear(2, 8)
    self.fc2 = torch.nn.Linear(8, 16)
    self.fc3 = torch.nn.Linear(16, 4)
```

结果：

```
epoch: 99/100,loss: 0.8710744380950928
accuracy: 0.9375
```

3.三层隐藏层，神经元数分别为8,16,10

网络定义：

```
def __init__(self):
    super(FNN1, self).__init__()
    self.fc1 = torch.nn.Linear(2, 8)
    self.fc2 = torch.nn.Linear(8, 16)
    self.fc3 = torch.nn.Linear(16, 10)
    self.fc4 = torch.nn.Linear(10, 4)
```

结果：

```
epoch: 99/100,loss: 0.7527700662612915
accuracy: 0.935
```

5.3 实例化

构建模型，并放入GPU，根据超参数，选择不同的优化器和损失函数

```
# construct model
model = FNN1().to(device)
print("model construct done!")

# construct optimizer
if optimizer_type == "Adam":
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
else:
    optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
print("optimizer construct done!")
# construct loss function ,try more loss function
if loss_fn_type == "MSELoss":
    loss_fn = torch.nn.MSELoss(reduction="mean")
else:
    loss_fn = torch.nn.CrossEntropyLoss(reduction="mean")
print("loss function construct done!")

# get train size and test size
```

```
train_size = len(train_set)
test_size = len(test_set)
```

5.4 构造训练函数

1.分离x，y，并放入GPU

```
def train():
    for epoch in range(epochs):
        for i, data in enumerate(train_loader):
            x= data[:, :-1]
            y = data[:, -1]
            x = x.to(torch.float32)
            x = x.to(device)
            y_pred = model(x)
```

2. 将y转为独热向量，用于匹配y_pred的形式，因为要计算损失，所以不能把y_pred转为y的形式

```
# convert y to one-hot probablity to fit softmax
if network_type == "softmax":
    y = torch.nn.functional.one_hot(y.to(torch.int64)-1,
num_classes=4).to(torch.float32)
    y=y.to(device)
```

3.计算损失，梯度清空，反向传播，优化

```
loss=loss_fn(y_pred,y)
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

4.保存模型

```
# save model
torch.save(model, "./model_{}_{}_{}.pth".format(network_type, optimizer_type,
loss_fn_type))
```

5.5 构建测试函数

1.方便debug测试阶段，直接读取保存的模型（可选）

```
def test():
    # load model
    #model = torch.load("./model_{}_{}_{}.pth".format(network_type,
optimizer_type, loss_fn_type))
```

2.分离x，y，放入GPU，预测

```
with torch.no_grad():
    correct = 0
    for i, data in enumerate(test_loader):
        x = data[:, :-1]
        y = data[:, -1]
        x = x.to(torch.float32)
        x = x.to(device)
```

3.需要注意的是，因为这里是获取预测结果，所以和前面训练时不同，应该把y_pred转为y的格式，同时，因为独热向量是从0开始，所以读取最大值位置后要+1才是真实预测的y的值！！！！

```
# convert y_pred from tensor to label and each +1
y_pred = torch.argmax(y_pred, dim=-1) + 1
y = y.to(device)
```

4.输出y_pred和y（调试用，超参noisy可选），计算正确个数，输出最终准确率

```
if noisy:
        print("y_pred: {}, y: {}".format(y_pred, y))
    # calculate accuracy
    correct += (y_pred == y).sum().item()

print("accuracy: {}".format(correct / test_size))
```

5.6 运行训练与测试

```
# train and test
if __name__ == "__main__":
    train()
    test()
```

六、注意点与总结

本以为本次实验内容比较基础，可以较快完成，但以前没有这样从头开始手搓网络过，有很多细节方面的问题没有注意到，耗费了较多时间。

下面的是本次实验的一些总结

1. csv读取后要先to_numpy才能放入torch的dataloader
2. 如用cuda，model也要to(device)
3. x分离之后需要转为float32才能放入GPU
4. 多用print观察各tensor格式进行匹配，使用torch.nn.functional.one_hot(y.to(torch.int64)-1, num_classes=4).to(torch.float32)将单标签转为独热向量，前参数为第几个为1，由label的值决定，记得要-1！！！！！！后一个参数为独热向量的长度
5. 好像只要tensor格式匹配，可以直接把y和y_pred放入pytorch的损失函数中
6. 使用torch.argmax(y_pred, dim=-1)将独热向量转为对应的预测单值，记得+1！！！！！！！！
7. test_loader如果batch开满，最终是全部值都放在一个tensor里面，所以最终比较y_pred和y是tensor之间的比较，使用correct += (y_pred == y).sum().item()计算正确的个数。

七、附录

总体代码如下：

```
import pandas as pd
import numpy as np
import torch

# setting
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
batch_size = 32
epochs = 100
learning_rate = 0.0005
# using softmax
network_type = "softmax"
# using which optimizer
optimizer_type = "Adam"
# using which loss function
loss_fn_type = "CrossEntropyLoss"
# print result
noisy = False
# print train info
print_info = True

# read dataset
dataset = pd.read_csv("./dataset.csv")

# split dataset
train_set = dataset.sample(frac=0.9, random_state=0)
test_set = dataset.drop(train_set.index)

# convert dataset to numpy
train_set = train_set.to_numpy()
test_set = test_set.to_numpy()

# construct dataloader
train_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size,
shuffle=True)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=400, shuffle=True)


# print dataloader


# construct network, this dataset has 2 features and 1 label
class FNN1(torch.nn.Module):
    def __init__(self):
        super(FNN1, self).__init__()
        self.fc1 = torch.nn.Linear(2, 8)
        self.fc2 = torch.nn.Linear(8, 16)
        self.fc3 = torch.nn.Linear(16, 4)


    def forward(self, x):
        x = torch.nn.functional.relu(self.fc1(x))
```

```
        x = torch.nn.functional.relu(self.fc2(x))
        x = self.fc3(x)

        if network_type == "softmax":
            return torch.nn.functional.softmax(x, dim=-1)
        else:
            return x


# construct model
model = FNN1().to(device)
print("model construct done!")

# construct optimizer
if optimizer_type == "Adam":
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
else:
    optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
print("optimizer construct done!")
# construct loss function ,try more loss function
if loss_fn_type == "MSELoss":
    loss_fn = torch.nn.MSELoss(reduction="mean")
else:
    loss_fn = torch.nn.CrossEntropyLoss(reduction="mean")
print("loss function construct done!")

# get train size and test size
train_size = len(train_set)
test_size = len(test_set)


# train function
def train():
    for epoch in range(epochs):
        for i, data in enumerate(train_loader):
            # here x is from 0 to -2, y is the last column
            # my label have 4 classes, so y should be a tensor with 4 elements,
each element is 0 or 1,if label is 3,the third element is 1,others are 0
            x= data[:, :-1]
            #y should be a tensor with 4 elements, each element is 0 or 1,if
label is 3,the third element is 1,others are 0
            y = data[:, -1]
            x = x.to(torch.float32)
            x = x.to(device)
            y_pred = model(x)
            # convert y to one-hot probablity to fit softmax
            if network_type == "softmax":
                y = torch.nn.functional.one_hot(y.to(torch.int64)-1,
num_classes=4).to(torch.float32)
            y=y.to(device)
            #print(y_pred)
            #print(y)
            loss=loss_fn(y_pred,y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

```
        # output loss each epoch
        if print_info:
            print("epoch: {}/{},loss: {}".format(epoch,epochs,  loss.item()))
    # save model
    torch.save(model, "./model_{}_{}_{}.pth".format(network_type, optimizer_type,
loss_fn_type))


# test function
def test():
    # load model
    #model = torch.load("./model_{}_{}_{}.pth".format(network_type,
optimizer_type, loss_fn_type))
    with torch.no_grad():
        correct = 0
        for i, data in enumerate(test_loader):
            x = data[:, :-1]
            y = data[:, -1]
            x = x.to(torch.float32)
            x = x.to(device)
            y_pred = model(x)
            # convert y_pred from tensor to label and each +1
            y_pred = torch.argmax(y_pred, dim=-1) + 1
            y = y.to(device)
            if noisy:
                print("y_pred: {}, y: {}".format(y_pred, y))
            # calculate accuracy
            correct += (y_pred == y).sum().item()

        print("accuracy: {}".format(correct / test_size))


# train and test
if __name__ == "__main__":
    train()
    test()
```