1.加载数据

In [ ]:
```
import os
import sys
import numpy as np
import torch
import pandas as pd

data_path = 'train_data'
train_data = []
labels = []
for file in os.listdir(data_path):
    # 获取文件的绝对路径
    file_path = os.path.join(data_path, file)
    # 截取文件名
    label = file.split('_')[-1]
    # 检查文件是否是目录
    if os.path.isdir(file_path):
        for data in os.listdir(file_path):
            # 读取文件内容
            a = np.load(os.path.join(file_path, data))
            train_data.append(a)
            labels.append(int(label))
```

2.数据预处理

In [ ]:
```
length = [i.shape[0] for i in train_data]
width = [i.shape[1] for i in train_data]
max_length = max(length)
width = max(width)

# 将所有数据填充到最大长度
for i in range(len(train_data)):
    if train_data[i].shape[0] < max_length:
        train_data[i] = np.vstack((train_data[i], np.zeros((max_length - train_data[i].shape[0], width))))

train_data = np.array(train_data)
labels = np.array(labels)

# 将数据和标签转换为张量
train_data_tensor = torch.tensor(train_data).unsqueeze(1).float()
print(train_data_tensor.shape)
labels_tensor = torch.tensor(labels).long()
```

```
torch.Size([4000, 1, 734, 80])
```

上述代码获取了数据的最大长度，并将所有数据扩充至最大长度。

3.建立模型

In [ ]:
```
# 构建神经网络
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# 定义网络
# 该网络输入为一个data序列，输出为该序列的labels分类
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, (3, 3))
        self.conv2 = nn.Conv2d(32, 64, (3, 3))
        self.conv3 = nn.Conv2d(64, 128, (3, 3))
        self.fc1 = nn.Linear(90 * 8 * 128, 128)
        self.fc2 = nn.Linear(128, 2)
        self.pool = nn.MaxPool2d((2, 2))
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        # print('-' * 20)
        x = self.pool(self.relu(self.conv1(x)))
        # print(x.shape)
        x = self.pool(self.relu(self.conv2(x)))
        # print(x.shape)
        x = self.pool(self.relu(self.conv3(x)))
        # print(x.shape)
        x = x.view(-1, 90 * 8 * 128)
        x = self.dropout(x)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

4.训练数据并输出准确率

In [ ]:
```
# 利用torch.utils.data构建数据集
import torch
from torch.utils.data import Dataset, DataLoader, TensorDataset

print("Data shape:", train_data_tensor.shape)
print("Labels shape:", labels_tensor.shape)
# 创建 TensorDataset 对象
dataset = TensorDataset(train_data_tensor, labels_tensor)

train_dataset, test_dataset = torch.utils.data.random_split(dataset, [int(len(dataset) * 0.8), len(dataset) - int(len(dataset) * 0.8)])

# 创建 DataLoader 对象
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=True)

# 定义损失函数和优化器
net = Model()
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net.parameters(), lr=0.001)

# 训练网络
for epoch in range(10):
    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data
        optimizer.zero_grad()
        # print(inputs.shape)
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if i % 10 == 9:
            print('[%d, %5d] loss: %.3f' % (epoch + 1, i + 1, running_loss / 10))
            running_loss = 0.0

# 保存模型
torch.save(net.state_dict(), 'model.pth')
```

```
Data shape: torch.Size([4000, 1, 734, 80])
Labels shape: torch.Size([4000])
[1,    10] loss: 1.740
[1,    20] loss: 0.677
[1,    30] loss: 0.672
[1,    40] loss: 0.667
[1,    50] loss: 0.655
[1,    60] loss: 0.619
[1,    70] loss: 0.659
[1,    80] loss: 0.634
[1,    90] loss: 0.627
[1,   100] loss: 0.624
[2,    10] loss: 0.623
[2,    20] loss: 0.598
[2,    30] loss: 0.616
[2,    40] loss: 0.594
[2,    50] loss: 0.564
[2,    60] loss: 0.578
[2,    70] loss: 0.543
[2,    80] loss: 0.521
[2,    90] loss: 0.529
[2,   100] loss: 0.544
[3,    10] loss: 0.405
[3,    20] loss: 0.413
[3,    30] loss: 0.395
[3,    40] loss: 0.355
[3,    50] loss: 0.337
[3,    60] loss: 0.358
[3,    70] loss: 0.326
[3,    80] loss: 0.345
[3,    90] loss: 0.351
[3,   100] loss: 0.350
[4,    10] loss: 0.345
[4,    20] loss: 0.280
[4,    30] loss: 0.292
[4,    40] loss: 0.285
[4,    50] loss: 0.267
[4,    60] loss: 0.300
[4,    70] loss: 0.274
[4,    80] loss: 0.301
[4,    90] loss: 0.299
[4,   100] loss: 0.260
[5,    10] loss: 0.239
[5,    20] loss: 0.281
[5,    30] loss: 0.275
[5,    40] loss: 0.227
[5,    50] loss: 0.226
[5,    60] loss: 0.330
[5,    70] loss: 0.246
[5,    80] loss: 0.259
[5,    90] loss: 0.256
[5,   100] loss: 0.285
[6,    10] loss: 0.219
[6,    20] loss: 0.276
[6,    30] loss: 0.209
[6,    40] loss: 0.259
[6,    50] loss: 0.295
[6,    60] loss: 0.250
[6,    70] loss: 0.263
[6,    80] loss: 0.255
[6,    90] loss: 0.273
[6,   100] loss: 0.266
[7,    10] loss: 0.216
[7,    20] loss: 0.211
[7,    30] loss: 0.188
[7,    40] loss: 0.243
[7,    50] loss: 0.216
[7,    60] loss: 0.253
[7,    70] loss: 0.218
[7,    80] loss: 0.291
[7,    90] loss: 0.257
[7,   100] loss: 0.236
[8,    10] loss: 0.164
[8,    20] loss: 0.253
[8,    30] loss: 0.193
[8,    40] loss: 0.213
[8,    50] loss: 0.174
[8,    60] loss: 0.282
[8,    70] loss: 0.253
[8,    80] loss: 0.242
[8,    90] loss: 0.180
[8,   100] loss: 0.203
[9,    10] loss: 0.200
[9,    20] loss: 0.180
[9,    30] loss: 0.206
[9,    40] loss: 0.223
[9,    50] loss: 0.230
[9,    60] loss: 0.270
[9,    70] loss: 0.230
[9,    80] loss: 0.211
[9,    90] loss: 0.227
[9,   100] loss: 0.184
[10,    10] loss: 0.202
[10,    20] loss: 0.230
[10,    30] loss: 0.216
[10,    40] loss: 0.177
[10,    50] loss: 0.259
[10,    60] loss: 0.218
[10,    70] loss: 0.229
[10,    80] loss: 0.162
[10,    90] loss: 0.185
[10,   100] loss: 0.205
```

In [ ]:
```
# 加载模型
net = Model()
net.load_state_dict(torch.load('model.pth'))
```

5.利用测试集验证网络的精度

In [ ]:
```
# 测试模型
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        inputs, labels = data
        outputs = net(inputs)
        _, predicted = torch.max(outputs.data, dim=1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
print('Accuracy: %d %%' % (100 * correct / total))
```

```
Accuracy: 90 %
```

6.预测数据

In [ ]:
```
# 构建测试集
result_data_path = 'test_data'
result_data = []
result_labels = []
for file in os.listdir(result_data_path):
    a = np.load(os.path.join(result_data_path, file))
    result_data.append(a)
    result_labels.append(-1)

# 截断或补全到 max_length x width
for i in range(len(result_data)):
    if result_data[i].shape[0] < max_length:
        result_data[i] = np.vstack((result_data[i], np.zeros((max_length - result_data[i].shape[0], width))))
    elif result_data[i].shape[0] > max_length:
        result_data[i] = result_data[i][:max_length, :]

result_data = np.array(result_data)
result_labels = np.array(result_labels)

# 将数据和标签转换为张量
result_data_tensor = torch.tensor(result_data).unsqueeze(1).float()
print(result_data_tensor.shape)
result_labels_tensor = torch.tensor(result_labels).long()
```

```
torch.Size([2000, 1, 734, 80])
```

In [ ]:
```
# 构建预测集
result_dataset = TensorDataset(result_data_tensor, result_labels_tensor)
result_data_loader = DataLoader(result_dataset, batch_size=32, shuffle=False)

# 测试网络
predictions = []
with torch.no_grad():
    for inputs,_ in result_data_loader:
        outputs = net(inputs)
        _, predicted = torch.max(outputs, 1)
        predictions.extend(predicted.numpy())
```

7.输出结果并保存

In [ ]:
```
import pandas as pd
result = pd.read_csv('test.csv')


# 将预测结果保存
result['label'] = predictions
result.to_csv('2321090049.csv', index=False)
```