DLP_LAB2

313551133 陳軒宇

Overview

這次實驗需要使用 Deep Learning 技術對 Brain-Computer Interface(BCI) 中的運動想像(Motor Imagery)任務進行分類，我們需要實現並訓練一個名為 SCCNet(Spatial Component-wise Convolutional Network) 的深度學習模型，用於分類四種不同的運動想像任務：左手、右手、雙腳和舌頭。

使用的 dataset 是 `BCI Competition IV 2a` ，該資料集包含多個受試者的腦電圖(EEG)信號。每個受試者有兩個 session 的資料，每個 session 包含 288 次試驗，其中每種運動想像類型各有72個試驗。數據已經過預處理，包括去除眼電圖(EOG)信號、帶通濾波(bandpass filter)、歸一化(normalization)和降低採樣率(downsampling)。

本實驗需要實現三種不同的訓練方法:

1. 受試者相關(Subject Dependent)
    - 此部分的訓練方法和論文中的方法有所不同，論文中是除了其中一個受試者的第 2 個 session 作為 validation set，其餘的所有 session 都做為 training set；而本次實驗中，需要使用所有受試者的第 1 個 session 作為 training set，第 2 個 session 作為 validation set。
2. 留一受試者交叉驗證(Leave-One-Subject-Out, LOSO)
    - 這種方式需要將其中一個受試者的第 2 個 session 作為 validation set ，並使用其餘受試者的所有 session 作為 training set，被選中受試者的第一個 session 將不予使用。
3. LOSO加微調(Fine-tuning)
    - 在 LOSO 的結果上，使用被選中受試者的第一個 session 作為 training set 來做 fine-tuning，並同樣使用該受試者的第 2 個 session 作為 validation set。

我們需要分別實現這三種方法，訓練對應的模型，並比較它們的性能。

Implementation Details

Details of training and testing code

訓練部分 (trainer.py)

1. 定義訓練函數 `train` ，用於訓練 SCCNet 模型。其傳入參數如下：

    - `epochs`
    - `batch_size`
    - `learning_rate`
    - `dropout_rate`
    - `target`：目標準確率，用於判斷模型是否已經超過目標準確率
    - `optimizer`：最佳化器，預設為 Adam
    - `scheduler`：學習率調度器，預設為 StepLR
    - `train_dataset_mode`：使用的訓練數據集。
    - `test_dataset_mode`：使用的測試數據集。
    - `base_model_path`：微調模型的路徑，在 finetune 訓練方法中使用，預設為 None
    - `model_path`：保存訓練過程中模型的路徑
    - `final_model_path`：最終模型的路徑

```
def train(epochs=1000,
          batch_size=32,
          learning_rate=0.01,
          dropout_rate=0.5,
          target=70,
          optimizer=optim.Adam,
          scheduler=optim.lr_scheduler.StepLR,
          train_dataset_mode=None, test_dataset_mode=None,
          base_model_path=None, # for finetune
          model_path='./model_weight/sd.pt',
          final_model_path='./model_weight/sd_final.pt'):
```

2. 載入資料集：

    - 使用在 Dataloader.py 中定義的 `MIBCI2aDataset` 類別加載訓練數據。
    - 通過 DataLoader 將數據批量化，並在訓練時進行 shuffle。

```
train_dataset = MIBCI2aDataset(mode=train_dataset_mode)
train_loader = DataLoader(train_dataset,
                          batch_size=batch_size,
                          shuffle=True)
```

3. 模型初始化:

    - 創建 SCCNet 模型的 instance，並將其移動到可用的設備(GPU 或 CPU)上。

```
device = torch.device('cuda' if torch.cuda.is_available(
model = SCCNet(dropoutRate=dropout_rate).to(device)
```

4. 定義 loss function, optimizer, scheduler ：

   - 使用 `CrossEntropyLoss` 作為 loss function。
   - 使用 Adam 優化器，並設置學習率為傳入的 `learning_rate` 參數，並根據論文，將 `weight_decay` 設置為 0.0001 (l_2 regularization)
   - 使用 `StepLR` 學習率調度器，每 100 個 epoch 將學習率降低 50%。

```
criterion = nn.CrossEntropyLoss()
optimizer = optimizer(model.parameters(), lr=learning_ra
scheduler = scheduler(optimizer, step_size=100, gamma=0.
```

5. 訓練過程：

   - 將模型設置為訓練模式。
   - 在每個 epoch 中，遍歷所有批次數據，進行前向傳播、反向傳播和參數更新。
   - 計算並記錄每個 epoch 的平均損失，並更新學習率。

```
model.train() # set model to train mode

for epoch in range(1, epochs + 1):
    running_loss = 0.0
    for _, (inputs, labels) in enumerate(train_loader):
        inputs, labels = inputs.to(device), labels.to(de

        # clear the gradients of all optimized variables
        optimizer.zero_grad()
        # forward pass
        outputs = model(inputs)
        # compute the loss
        loss = criterion(outputs, labels)
        # backpropagation
        loss.backward()
        # update the parameters
        optimizer.step()
        # accumulate the loss
        running_loss += loss.item()

    # compute the average loss
    avg_loss = running_loss / len(train_loader)
    losses.append(avg_loss)
    current_lr = optimizer.param_groups[0]['lr']
    # update the learning rate
    scheduler.step()
```

6. 模型保存和驗證:
    - 保存每個 epoch 的模型，並使用 tester.py 中的函數在 validation set 上評估模型性能。
        - 註：為了節省空間且避免發生意外，這裡不保存所有 epoch 的模型，而是在評估模型性能後達到目標準確率或超過目前最高的準確率時，才保存作為最終模型。否則僅保存最後一個 epoch 的模型。
    - 如果達到目標準確率，則提前終止訓練。

```
# save the model
torch.save(model.state_dict(), model_path)
# save the loss history
with open(f"{model_path}.pkl", 'wb') as f:
    pickle.dump(losses, f)

# test the model
acc = sccnet_test(batch_size, test_dataset_mode, model_p
if acc > max_acc:
    # update the max accuracy
    max_acc = acc
    # save the final model
    torch.save(model.state_dict(), final_model_path)
# if the max accuracy is reached, stop training
if acc > target:
    break
```

7. 微調(finetune)：
    - 提供了加載預訓練模型進行 finetune 的功能，用於 LOSO+FT 訓練方法。

```
if train_dataset_mode == 'finetune':
    # if the base model path does not exist
    if not os.path.exists(base_model_path):
        # raise an error
        raise ValueError(f'Base model does not exist')
    # load the base model
    model.load_state_dict(torch.load(base_model_path))
```

測試部分 (tester.py)

1. 定義測試函數 `sccnet_test` ，用於測試 SCCNet 模型。其傳入參數如下：
    - `batch_size` ：測試批次大小
    - `mode` ：使用的測試資料集，可以是 `sd_test` 或 `loso_test`
    - `model_path`：使用的模型檔案路徑
    - `model`：使用的模型，為了應對參數可能有變化的情況。

```
def sccnet_test(batch_size=32,
                mode="sd_test",
                model_path='./model_weight/model.pt',
                model=SCCNet(Nu=22, dropoutRate=0.5)):
```

2. 載入資料集：

   - 使用在 Dataloader.py 中定義的 `MIBCI2aDataset` 類別加載訓練數據。
   - 通過 DataLoader 將數據批量化，但在測試時不需要隨機打亂數據。

```
test_dataset = MIBCI2aDataset(mode=mode)
test_loader = DataLoader(test_dataset,
                         batch_size=batch_size,
                         shuffle=False)
```

3. 載入模型參數：

   - 檢查模型檔案是否存在，如果不存在則 raise 一個錯誤。
   - 由於在函數的傳入參數中已經包含了模型的 instance，所以不需要再次定義模型，只需要將模型參數加載到模型中即可。
   - 將模型設置為評估模式。

```
if not os.path.exists(model_path):
    raise FileNotFoundError(f"Model path not found")

device = torch.device('cuda' if torch.cuda.is_available(
model.to(device)
model.load_state_dict(torch.load(model_path))
model.eval() # set model to evaluation mode
```

4. 評估模型：

   - 在測試資料集上運行模型，累加資料數量以及正確數量，並保存預測結果與正確結果。
   - 返回正確率。

```
    tot = cnt = 0
    all_preds = []
    all_labels = []

    with torch.no_grad():  # disable gradient calculation
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(de

            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)

            tot += labels.size()[0]
            cnt += (predicted == labels).sum().item()

            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    acc = cnt / tot * 100

    return acc
```

Details of the SCCNet

```
SCCNet(
  (conv1): Sequential(
    (0): Conv2d(1, 44, kernel_size=(22, 2), stride=(1, 1))
    (1): Permute2d()
    (2): BatchNorm2d(1, eps=1e-05, momentum=0.1, affine=True,
  )
  (conv2): Sequential(
    (0): Conv2d(1, 20, kernel_size=(44, 12), stride=(1, 1), p
    (1): BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=True
  )
  (square): SquareLayer()
  (dropout): Dropout(p=0.5, inplace=False)
  (avgpool): AvgPool2d(kernel_size=(1, 62), stride=(1, 12), p
  (fc): Linear(in_features=640, out_features=4, bias=True)
  (softmax): Softmax(dim=1)
)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param
================================================================
            Conv2d-1          [-1, 44, 1, 437]           1,9
          Permute2d-2          [-1, 1, 44, 437]
       BatchNorm2d-3          [-1, 1, 44, 437]
            Conv2d-4          [-1, 20, 1, 438]          10,5
       BatchNorm2d-5          [-1, 20, 1, 438]
       SquareLayer-6          [-1, 20, 1, 438]
           Dropout-7          [-1, 20, 1, 438]
         AvgPool2d-8           [-1, 20, 1, 32]
            Linear-9                  [-1, 4]           2,5
          Softmax-10                  [-1, 4]
================================================================
Total params: 15,166
Trainable params: 15,166
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.04
Forward/backward pass size (MB): 0.71
Params size (MB): 0.06
Estimated Total Size (MB): 0.81
----------------------------------------------------------------
```

SCCNet (Spatial Component-wise Convolutional Network) 是一個專門為運動想像EEG分類設計的卷積神經網絡模型。其主要特點在於初始卷積層的設計，模仿了傳統EEG分析中常用的空間濾波技術。SCCNet的架構主要包含以下幾個部分：

1. 空間成分分析層 (Spatial Component Analysis Layer)：
    - 使用2D卷積層，輸入通道數為 1，輸出通道數為 Nu (預設為44)
    - 卷積核大小為 (C, N_t)，其中 C 為 EEG 通道數(22)，N_t 為時間卷積核大小(預設為1)
    - 此層的作用類似於傳統EEG分析中的空間濾波，將原始EEG信號轉換為空間成分。
    - 為了適應後續的 BatchNorm2d 操作，在 Conv2d 層之後加入 Permute2d 層，以保持輸入形狀為 $(batch_size, 1, C, time_sample)$。

```
self.conv1 = nn.Sequential(
    nn.Conv2d(in_channels=1, out_channels=Nu, kernel_siz
    Permute2d((0, 2, 1, 3)),
    nn.BatchNorm2d(1)
)
```

2. 時空卷積層 (Spatio-Temporal Convolution Layer)：

- 使用2D卷積層，輸入通道數為 1，輸出通道數為 N_c (預設為 20)
- 卷積核大小為 $(N_u, 12)$，其中 12 對應於 0.1 秒的時間窗口
- 此層對空間成分進行進一步的時空特徵提取

```
self.conv2 = nn.Sequential(
    nn.Conv2d(in_channels=1, out_channels=Nc, kernel_siz
    nn.BatchNorm2d(Nc),
)
```

3. 平方層 (Square Layer):

- 對前一層的輸出進行平方運算
- 這一步驟目的是提取信號的能量特徵，因為頻譜能量變化是運動想像EEG的主要特徵

```
self.square = SquareLayer()
```

4. Dropout層:

- 使用 dropout 率為 0.5 (可調整)，用於防止 overfitting

```
self.dropout = nn.Dropout(dropoutRate)
```

5. 平均池化層 (Average Pooling Layer):

- kernel_size 為 $(1, 62)$，stride 為 $(1, 12)$ ，其中 62 對應於 0.5 秒的時間窗口，12 對應於 0.1 秒的時間窗口
- 此層用於在時間維度上進行平滑處理和降維

```
self.avgpool = nn.AvgPool2d(kernel_size=(1, 62), stride=
```

6. 全連接層 (Fully Connected Layer):

- 輸入特徵數為 $N_c * (\lfloor \frac{\text{timeSample}-62}{12} \rfloor + 1)$
- 輸出特徵數 4 (對應左手、右手、雙腳和舌頭的運動想像)

```
fc_inSize = Nc * ((timeSample - 62) // 12 + 1)
self.fc = nn.Linear(in_features=fc_inSize, out_features=
```

7. Softmax層:
    - 用於將全連接層的輸出轉換為機率分佈

```
self.softmax = nn.Softmax(dim=1)
```

(Optional) Anything you want to mention

Analyze on the experiment results

Discover during the training process

Comparison between the three training methods

(Optional) Anything you want to mention

Discussion

What is the reason to make the task hard to achieve high accuracy?

What can you do to improve the accuracy of this task?

(Optional) Anything you want to mention