國立陽明交通大學

暑期課程　深度學習實驗　LAB2

313553036 多工所 1 葉軒宇

1. Overview

基於 Motor Imagery 的 BCI 在實現大腦與外部設備的直接通訊上有相當大的潛力，這次的 LAB 主要在實現並使用 SCCNet 來分類 EEG 的訊號。

在實作內容中，主要的目的是要自己實現 SCCNet 的每步流程，至於其中細微的調整像是則是看個人。另外一個重點則是看各種不同的訓練方案對於準確度的影響

2. Implementation Details

a. Details of training and testing code

```python
def train_model(model, train_loader, test_loader, num_epochs, learning_rate, device, model_save_path):
    criterion = nn.CrossEntropyLoss()
    #optimizer = optim.Adam(model.parameters(), lr=learning_rate*0.01, weight_decay=0.0001, betas=(0.9, 0.99))
    optimizer = optim.Adam(model.parameters(), lr=learning_rate, weight_decay=0.0001)

    history = {'train_loss': [], 'train_acc': [], 'test_loss': [], 'test_acc': []}
    best_test_acc = 0

    for epoch in range(num_epochs):
        model.train()
        train_loss = 0
        train_correct = 0
        train_total = 0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            train_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            train_total += labels.size(0)
            train_correct += (predicted == labels).sum().item()

        train_loss /= len(train_loader)
        train_acc = 100 * train_correct / train_total
```

一開始先定義了損失函數以及優化器，然後進入訓練循環。

```
        model.eval()
        test_loss = 0
        test_correct = 0
        test_total = 0
        with torch.no_grad():
            for inputs, labels in test_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                test_loss += loss.item()
                _, predicted = torch.max(outputs.data, 1)
                test_total += labels.size(0)
                test_correct += (predicted == labels).sum().item()

        test_loss /= len(test_loader)
        test_acc = 100 * test_correct / test_total

        history['train_loss'].append(train_loss)
        history['train_acc'].append(train_acc)
        history['test_loss'].append(test_loss)
        history['test_acc'].append(test_acc)

        print(f'Epoch [{epoch + 1}/{num_epochs}], Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%, '
              f'Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.2f}%')

        if test_acc > best_test_acc:
            best_test_acc = test_acc
            save_model(model, model_save_path)

    plot_training_history(history)
    return model
```

之後將訓練的流程存入 history，方便之後繪製圖形。

```
def train_SD(num_epochs=200, learning_rate=0.0005, batch_size=32):
    model = SCCNet(Nu=22, Nc=20, Nt=1).to(device)
    train_loader = get_dataloader( mode: 'SD_train', batch_size=batch_size)
    test_loader = get_dataloader( mode: 'SD_test', batch_size=batch_size)
    return train_model(model, train_loader, test_loader, num_epochs, learning_rate, device, model_save_path: 'SD_model.pth')


1 usage
def train_LOSO(num_epochs=200, learning_rate=0.001, batch_size=32):
    model = SCCNet(Nu=22, Nc=20, Nt=1).to(device)
    train_loader = get_dataloader( mode: 'LOSO_train', batch_size=batch_size)
    test_loader = get_dataloader( mode: 'LOSO_test', batch_size=batch_size)
    return train_model(model, train_loader, test_loader, num_epochs, learning_rate, device, model_save_path: 'LOSO_model.pth')


def train_LOSO_FT(num_epochs_initial=200, num_epochs_ft=20, learning_rate=0.001, batch_size=32):
    model = SCCNet(Nu=22, Nc=20, Nt=1).to(device)
    train_loader = get_dataloader( mode: 'LOSO_train', batch_size=batch_size)
    test_loader = get_dataloader( mode: 'LOSO_test', batch_size=batch_size)
    model = train_model(model, train_loader, test_loader, num_epochs_initial, learning_rate, device,
                        model_save_path: 'LOSO_FT_initial_model.pth')

    # Fine-tuning
    ft_loader = get_dataloader( mode: 'FT', batch_size=batch_size)
    return train_model(model, ft_loader, test_loader, num_epochs_ft, learning_rate * 0.1, device,
                       model_save_path: 'LOSO_FT_final_model.pth')
```

最後根據不同訓練方式定義其訓練資料以及測試資料，以及最重要的——把 path 存起來。

接下來是 testing 的部分

```python
def test(model, test_loader, device):
    model.eval()
    correct = 0
    total = 0
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            all_preds.extend(predicted.cpu().tolist())
            all_labels.extend(labels.cpu().tolist())

    accuracy = 100 * correct / total
    cm = confusion_matrix_torch(all_labels, all_preds)

    return accuracy, cm
```

Test 函數主要用於預測結果並與真實結果去做比較。

```python
def confusion_matrix_torch(y_true, y_pred, num_classes=4):
    cm = torch.zeros(num_classes, num_classes, dtype=torch.int64)
    for t, p in zip(y_true, y_pred):
        cm[t, p] += 1
    return cm


3 usages
def test_model(model_path, test_mode, batch_size=32):
    model = SCCNet(Nu=22, Nc=20, Nt=1).to(device)
    model.load_state_dict(torch.load(model_path, map_location=device))
    test_loader = get_dataloader(test_mode, batch_size=batch_size)
    accuracy, cm = test(model, test_loader, device)

    print(f'Test Accuracy: {accuracy:.2f}%')
    print("Confusion Matrix:")
    print(cm)

    return accuracy, cm
```

Confusion matrix 可以幫助分析真實以及預測的結果。

Test model 則是家仔已經訓練好的模型，並輸出訓練結果。

```python
#for SD
print("Results for SD model:")
sd_accuracy, sd_cm = test_model( model_path: 'SD_model.pth', test_mode: 'SD_test', batch_size=32)
print(f"SD Model Accuracy: {sd_accuracy:.2f}%")
print("Confusion Matrix:")
print(sd_cm)

#for LOSO
print("\nResults for LOSO model:")
loso_accuracy, loso_cm = test_model( model_path: 'LOSO_model.pth', test_mode: 'LOSO_test', batch_size=32)
print(f"LOSO Model Accuracy: {loso_accuracy:.2f}%")
print("Confusion Matrix:")
print(loso_cm)

#for LOSO_FT
print("\nResults for LOSO with Fine-tuning model:")
loso_ft_accuracy, loso_ft_cm = test_model( model_path: 'LOSO_FT_final_model.pth', test_mode: 'LOSO_test', batch_size=32)
print(f"LOSO with Fine-tuning Model Accuracy: {loso_ft_accuracy:.2f}%")
print("Confusion Matrix:")
print(loso_ft_cm)
```

最後就是根據之前不同訓練方式的結果去 output 出相對應的準確度。

b. Details of the SCCNet

```python
import torch
import torch.nn as nn
import math
import torch.nn.functional as F


7 usages
class SCCNet(nn.Module):
    def __init__(self, C=22, T=438, Nu=22, Nt=1, Nc=20, dropoutRate=0.5, numClasses=4):
        super(SCCNet, self).__init__()
        self.spatialConv = nn.Conv2d( in_channels: 1, Nu, kernel_size=(C, Nt), bias=False)
        self.batchNorm1 = nn.BatchNorm2d(Nu)
        self.temporalConv = nn.Conv2d(Nt, Nc, kernel_size=(Nu, 12), padding=(0, 11//2))
        self.batchNorm2 = nn.BatchNorm2d(Nc)
        self.dropout = nn.Dropout(dropoutRate)
        self.avgPool = nn.AvgPool2d(kernel_size=(1, 62), stride=(1, 12))
        self.fc_input_size = (math.floor((T - 62) / 12) + 1) * Nc
        self.fc = nn.Linear(self.fc_input_size, numClasses)
```

```python
def forward(self, x):
    x = x.permute(0, 2, 1, 3)

    x = self.spatialConv(x)
    x = self.batchNorm1(x)
    x = x.permute(0, 2, 1, 3)
    x = self.temporalConv(x)
    x = self.batchNorm2(x)
    x = F.elu(x)
    x = torch.square(x)
    x = self.dropout(x)
    x = self.avgPool(x)

    x = x.view(x.size(0), -1)

    # 驗證 fc 輸入大小
    assert x.size(1) == self.fc_input_size, f"FC input size mismatch. Expected {self.fc_input_size}, got {x.size(1)}"

    x = self.fc(x)
    x = F.softmax(x, dim=1)
    return x
```

根據參考的 paper 以及 data 的 shape 可以知道 C, T, Nu, Nt, Nc

這些參數的數值。設定好之後便可將其導入 forward 中。

3. Analyze on the experiment results

a. Discover during the training process

在一開始訓練、調整參數時，並沒有辦法達到作業要求的準確率，在試著在 forward 中加入 activation function(elu)以及調整 optimizer 的 betas 之後發現有 elu 以及沒有 betas 的情況下可以有最好的準確率。

	A	B	C	D	E	F
1		with elu	with elu		no elu	no elu
2		with betas	no betas		with beatas	no betas
3	LOSO					
4		51.74%	52.82%		59.03%	52.78%
5		52.43%	52.78%		51.39%	52.43%
6		51.74%	55.98%		52.17%	59.72%
7	SD					
8		59.64%	61.19%		60.37%	59.94%
9		60.42%	62.28%		59.38%	60.29%
10		60.11%	61.89%		59.07%	61.07%
11	LOSO_FT					
12		65.62%	63.89%		65.28%	62.50%
13		63.54%	67.71%		67.71%	65.28%
14		64.62%	71.18%		63.89%	63.54%

但在調整後還是都不是每次均可以達到要求的準確度，尤其是 LOSO_FT，於是就對 epochs 進行調整。

一開始 initial, ft 分別是 200 以及 20，想說調高一點應該會比較準確，於是調成(1000,50)，learning rate = 0.0006。

```
print("Training LOSO with Fine-tuning model...")
train_LOSO_FT(num_epochs_initial=1000, num_epochs_ft=50, learning_rate=0.0006, batch_size=32)
```

但可以發現在前面用 LOSO 的 data 訓練的階段，在超過 100 之後其實 loss 就不會再降低了，而且這樣也導致準確率比原本(200, 20)還要差，於是就改成(50, 100), learning rate = 0.0006。

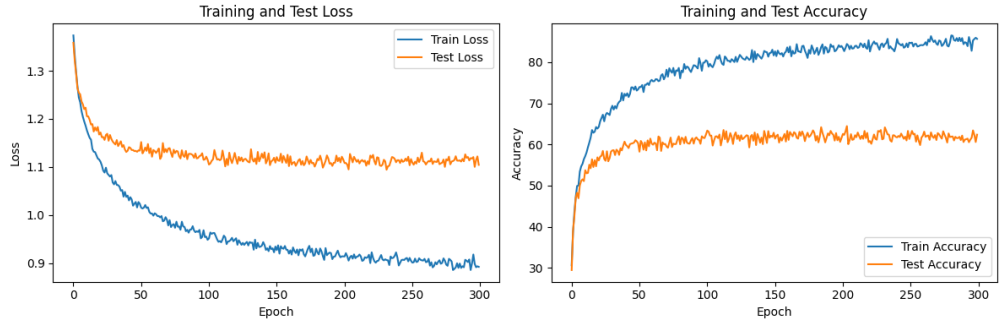
```
print("Training LOSO with Fine-tuning model...")
train_LOSO_FT(num_epochs_initial=50, num_epochs_ft=100, learning_rate=0.0006, batch_size=32)
```

這樣反而得到了比較好的結果。

b. Comparison between the three training method

除了前面提及的 LOSO_FT 的部分外，縱使在評分標準上看似 LOSO 應該較 SD 難得到比較好的準確率。但在同樣的架構、類似的參數情況下，LOSO 以及 SD 的準確率卻沒有相差太大，而 LOSO_FT 擁有最高的準確度。

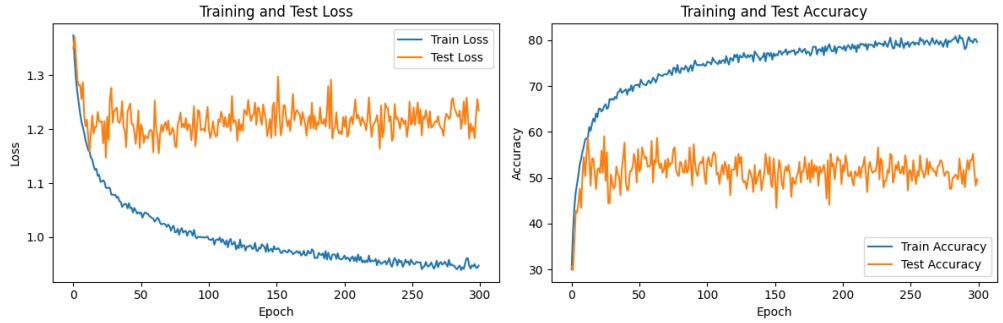
SD:




```
print("Training SD model...")
train_SD(num_epochs=300, learning_rate=0.0007, batch_size=32)
```

LOSO:

```
print("Training LOSO model...")
train_LOSO(num_epochs=300, learning_rate=0.0006, batch_size=32)
```



LOSO_FT:

```
print("Training LOSO with Fine-tuning model...")
train_LOSO_FT(num_epochs_initial=50, num_epochs_ft=100, learning_rate=0.0006, batch_size=32)
```

4. Discussion

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

EEG 的信號本來就是由人腦所發出，所以根據不同人的腦袋

發出的信號，會有一定的差距存在。這個訊號又容易受到外部電極的干擾，導致樣本數不高。且 EEG 具有強烈的時間依賴性，在提取 feature 及 label 時就會相當困難。除此之外，因為四種 class(左手、右手、雙腳以及舌頭)之間的區別可能並不是非常明顯，像是左手以及右手的訊號可能就會很相似。

在評分標準上，LOSO 可能要求最低的原因是，它試著泛化未見過的受試者，而不同受試者間的 EEG 模式可能就會有很大的差異。

b. Try What can you do to improve the accuracy of this task

如果想要增加準確率，可能就是在 Nu, learning rate 以及 batch 數值上再進行調整。或者是對提供的 data 進行再處理，像是混和數據或是添加高斯噪聲 EEG 信號中。

但在我的實驗過程中發現，前兩層的 convolution 是有明確意義的。若貿然加入第三層的 convolution，在不確定是要提取什麼特徵的情況下，準確率不只不會提高，甚至還會大幅度下降。