Assignment #5 -- Parkinson's Freezing of Gait Prediction

題目要求

使用Kaggle競賽的資料集：Parkinson's Freezing of Gait Prediction，並調整範例程式的DNN模型，如下:

1. 更改model的block數，比較調整前、調整後的模型準確率。你可以自由的增加block數，並提供模型訓練的Loss vs Iteration(或Epoch)圖，觀察模型的訓練是否過度擬合抑或訓練成功。

2. 更改model內block的層數，試著增加或減少nn.Linear 或nn.BatchNorm1d，比較調整前、調整後的模型準確率，並提供並提供模型訓練的Loss vs Iteration(或Epoch)圖。

※model的block數可在Config內的model_nblocks更改，block內層數在Model內更改
※Kaggle繳交方式在PPT內有教學

我的答案

如何執行 (Execution Description)

在 Kaggle 上按 Run All 。

實驗結果 (Experimental Results)

根據題目要求整體上來說是要更改 model 的 block 數量以及 model block 裡面的層的內容。

這邊我定義 2 種 `_block` 實作方式：

- $Block_1$ ：

```
def _block(in_features, out_features, drop_rate):
    return nn.Sequential(
        nn.Linear(in_features, out_features),
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Dropout(drop_rate)
    )
```

- $Block_2$

```
    def _block(in_features, out_features, drop_rate):
        return nn.Sequential(
            nn.Linear(in_features, out_features),
            nn.BatchNorm1d(out_features),
            nn.ReLU(),
            nn.Linear(out_features, out_features),
            nn.BatchNorm1d(out_features),
            nn.ReLU(),
            nn.Linear(out_features, out_features),
            nn.BatchNorm1d(out_features),
            nn.ReLU(),
            nn.Dropout(drop_rate)
        )
```

這邊是我用來整理實驗結果的 code block ：

```
print("<<< RESULT >>>")

print("model_nblocks:", cfg.model_nblocks)
print("_block info:", block_info_str)
print("max_score:", max_score)
print("train_loss_list:", train_loss_list)
print("val_loss_list:", val_loss_list)

epoch_list = [i for i in range(cfg.num_epochs)]
print("epoch_list:", epoch_list)

import matplotlib.pyplot as plt

plt.plot(epoch_list, train_loss_list, label="Train Loss")
plt.plot(epoch_list, val_loss_list, label="Validation Loss")
plt.title("Training Loss and Validation Loss curves")
plt.ylabel("Loss")
plt.xlabel("Epochs")
plt.legend()
```

我把我所做的調整與相對應的實驗數據截圖整理如下：

- Block 數量為 1 且使用 $Block_1$ ：

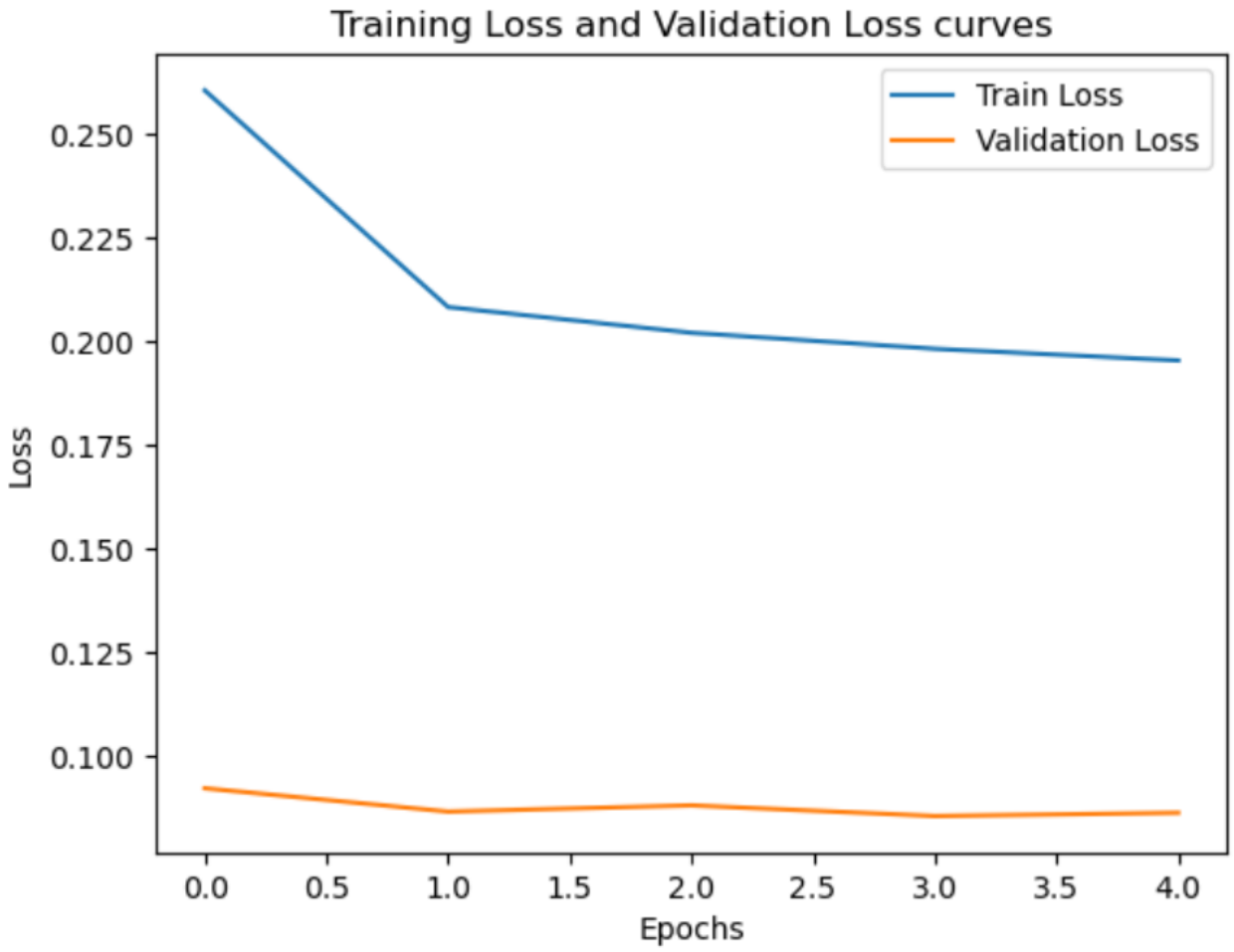
```
<<< RESULT >>>
model_nblocks: 1
_block info:
def _block(in_features, out_features, drop_rate):
    return nn.Sequential(
        nn.Linear(in_features, out_features),
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Dropout(drop_rate)
    )

max_score: 0.20068357677572557
train_loss_list: [0.2450706131013231, 0.2077493015955199, 0.20367738543817945, 0.20126657730520395, 0.20027466415870684]
val_loss_list: [0.10169156809603748, 0.09652705182399426, 0.09675354148070317, 0.09499888078536017, 0.09618783227490438]
epoch_list: [0, 1, 2, 3, 4]
```

- Block 數量為 8 且使用 $Block_1$ ：

```
<<< RESULT >>>
model_nblocks: 8
_block info:
def _block(in_features, out_features, drop_rate):
    return nn.Sequential(
        nn.Linear(in_features, out_features),
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Dropout(drop_rate)
    )

max_score: 0.2564702498144204
train_loss_list: [0.26051066445383947, 0.20822957945106801, 0.20203398732030195, 0.19816846890135084, 0.19535354367843935]
val_loss_list: [0.09214978422128749, 0.08652418836812648, 0.08802334248538154, 0.08545473541392147, 0.08623198132232647]
epoch_list: [0, 1, 2, 3, 4]
```



- Block 數量為 1 且使用 $Block_2$ ：

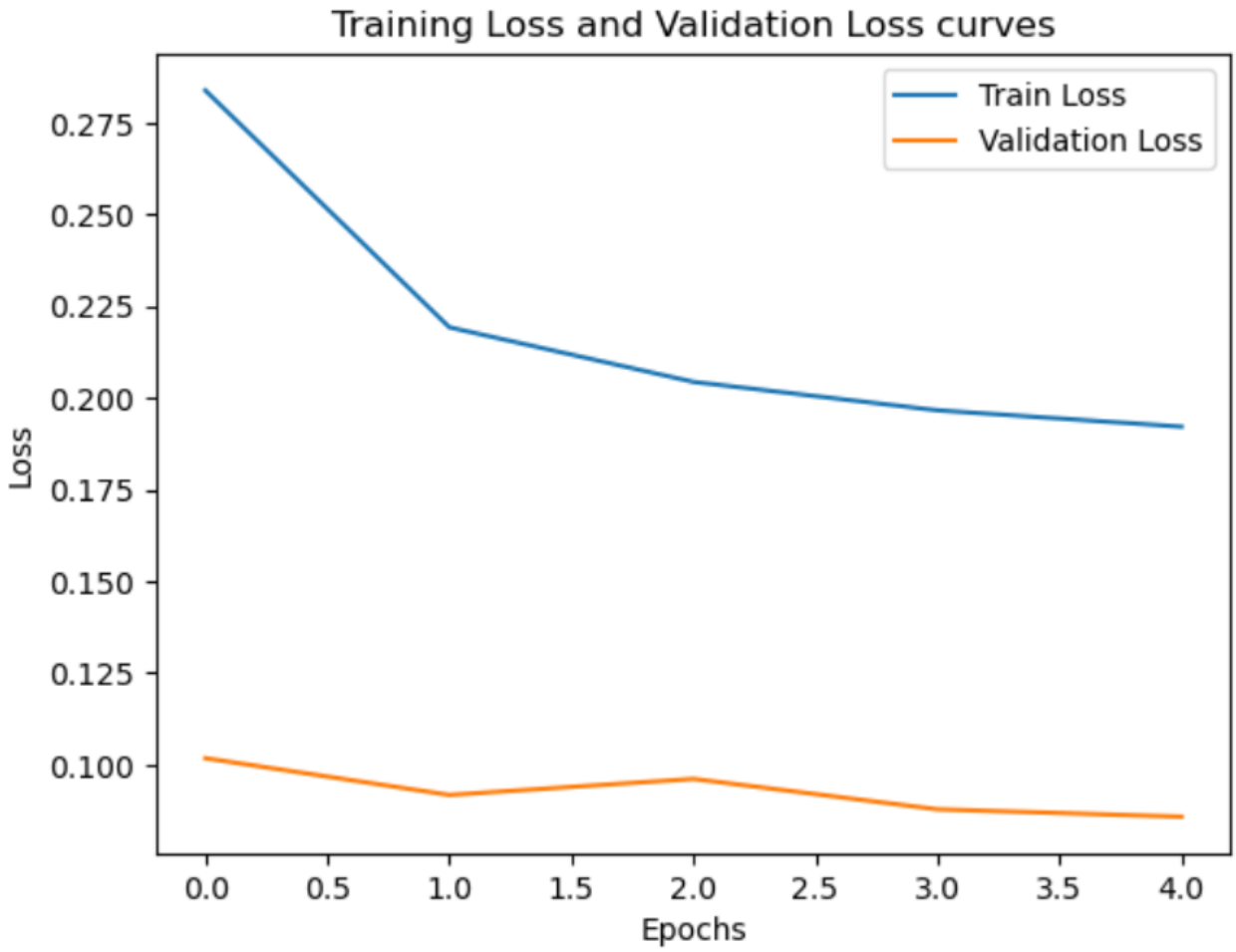
```
<<< RESULT >>>
model_nblocks: 1
_block info:
def _block(in_features, out_features, drop_rate):
    return nn.Sequential(
        nn.Linear(in_features, out_features),
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Linear(out_features, out_features),
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Linear(out_features, out_features),
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Dropout(drop_rate)
    )

max_score: 0.2181009382479144
train_loss_list: [0.24175928857161474, 0.1852530268263724, 0.18040393606612556, 0.17731899990120945, 0.1755070377788948]
val_loss_list: [0.09011935705566967, 0.09582560887875696, 0.09126174518636412, 0.08932103100364044, 0.0921487122684283]
epoch_list: [0, 1, 2, 3, 4]
```

- Block 數量為 8 且使用 $Block_2$ ：

```
<<< RESULT >>>
model_nblocks: 8
_block info:
def _block(in_features, out_features, drop_rate):
    return nn.Sequential(
        nn.Linear(in_features, out_features),
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Linear(out_features, out_features),
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Linear(out_features, out_features),
        nn.BatchNorm1d(out_features),
        nn.ReLU(),
        nn.Dropout(drop_rate)
    )

max_score: 0.21332948325255377
train_loss_list: [0.28376879970448293, 0.21919887990961792, 0.2043304743945977, 0.19657932878297396, 0.19212588938855255]
val_loss_list: [0.10176075138641413, 0.09174282136023115, 0.09611517121701026, 0.08782632168667083, 0.08580882545396928]
epoch_list: [0, 1, 2, 3, 4]
```



結果觀察 (Conclusion)

- 使用 $Block_1$ 的情況來看， block 數量為 8 的結果 (`max_score`) 會比 block 數量為 1 的結果還要好。
- 使用 $Block_2$ 的情況來看， block 數量為 8 的結果會比 block 數量為 1 的結果還要差。
- 從實驗數據可以看出 block 數量為 8 且使用 $Block_1$ 時的結果最好。

相關討論 (Disscussion)

- 因為不確定 block 數量與 block 實作的方式哪個對結果的影響比較大，所以我常試計算各種可能的解果，以這次實驗來看也就是 4 種。
- 以我的實驗結果來看 block 多一點，實作方式採 $Block_1$ 不要過於複雜的可以達到比較好的結果。
- 在實作 block 時，要注意前後層參數的對應。

程式碼在 `ml-parkinson-s-freezing-of-gait-prediction.ipynb` 檔案中