HW3

612415013 蕭宥羽

1. How to execute codes.

- `Config` 主要是做超參數的設定，而這次的做也我主要調整這幾個參數

```
model_dropout = 0.2                                                    #dropout機率
model_hidden = 64                                                      #每層linear layer的神經元數
model_nblocks = 3                                                      #model內的block數

lr = 0.00015                                                           #learning rate
num_epochs = 20                                                         #訓練的epochs數
```

`model_dropout` 選擇 dropout 的比例，dropout 會隨機丟棄神經元 避免過度依賴某些神經元而造成 overfitting

`model_hidden` 設定每層中的神經元數量

`model_nblocks` 設定 model 中 block 的數量，也就是說中間有幾層的意思

`lr、num_epochs` 學習率與迭代次數

- 接著做資料及的載入及預處理，使用"tdcsfog" 和 "defog" 兩個資料集，並將使用 K Fold 方式做資料及處理

- Model 這塊設定模型的架構

```
def _block(in_features, out_features, drop_rate):
    return nn.Sequential(
        nn.Linear(in_features, out_features),      # 全連接層 in_features：輸入的特徵維度  out_features:輸出的特徵維度
        nn.BatchNorm1d(out_features),              # 做標準化的處理 對每個特徵的值進行調整，使其服從標準正態分佈
        nn.ReLU(),                                 # activate function 使用非線性的函數 讓神經網路可以學習更複雜的關係
        nn.Dropout(drop_rate)                      # 隨機將部份神經元的輸出設為0 使神經網路不會過度依賴特定的神經元 避免overfitting
    )

class FOGModel(nn.Module):
    # 初始化
    def __init__(self, p=cfg.model_dropout, dim=cfg.model_hidden, nblocks=cfg.model_nblocks):
        super(FOGModel, self).__init__()
        self.dropout = nn.Dropout(p)                                                    # 定義dropout
        self.in_layer = nn.Linear(cfg.window_size*3, dim)                               # 定義模型的輸入層 輸入為原始資料的維度 輸出為dim
        self.blocks = nn.Sequential(*[_block(dim, dim, p) for _ in range(nblocks)])     # 使用上面寫的 _block 共使用nblocks個_block
        self.out_layer = nn.Linear(dim, 3)                                              # 輸出層的維度 輸出3個結果

    # 做前向傳播                                  # dim=32
    def forward(self, x):                         # x.Size([1024,32,3])
        x = x.view(-1, cfg.window_size*3)         # x.Size([1024,96])  將x重塑成二維張量
        x = self.in_layer(x)                      # x.Size([1024,32])  將x通過in_layer進行線性映射，將特徵維度變成dim
        for block in self.blocks:
            x = block(x)                          # x.Size([1024,32])  將x通過中間的隱藏層
        x = self.out_layer(x)                     # x.Size([1024,3])   最終的輸出
        return x
```

會傳入 Config 中所設定的一些參數，可以調整中間的層數及神經元數量

- Train

```
def train_one_epoch(model, loader, optimizer, criterion):
def validation_one_epoch(model, loader, criterion):
```

設定每一次的 training 的過程，包括 Loss 的計算等等

而這兩個的差別在於有沒有做反向傳播，train 有做 validation 沒有做，因為我們是用訓練資料及去反向傳播更新我們的參數。

- Submission

進行模型的測試和生成提交文件

2. Experimental results

- Original V1

  model_dropout = 0.2

  model_hidden = 32

  model_nblocks = 1

  num_epochs = 5

ML_Parkinson's Freezing of Gait Prediction - Version 1 **0.200009** **0.240621**

Succeeded (after deadline) · 3d ago · Notebook ML_Parkinson's Freezing ...

- V4

  model_dropout = 0.2

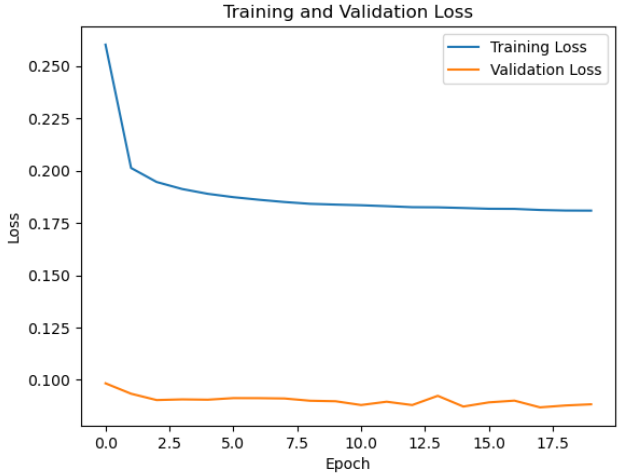
  model_hidden = 32

  model_nblocks = 3

  num_epochs = 20

ML_Parkinson's Freezing of Gait Prediction - Version 4 **0.253830** **0.240297**

Succeeded (after deadline) · 3d ago



- V6

  model_dropout = 0.5

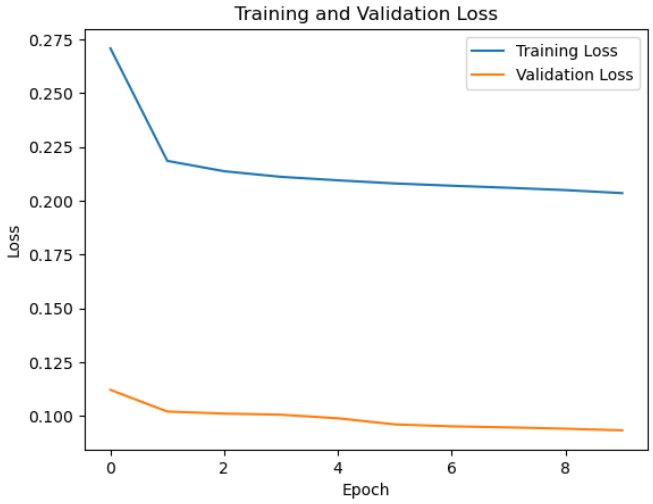
  model_hidden = 64

  model_nblocks = 5

  num_epochs = 10

ML_Parkinson's Freezing of Gait Prediction - Version 6 **0.168369** **0.232902**

Succeeded (after deadline) · 3d ago · Notebook ML_Parkinson's Freezing ...



- V9

  model_dropout = 0.5

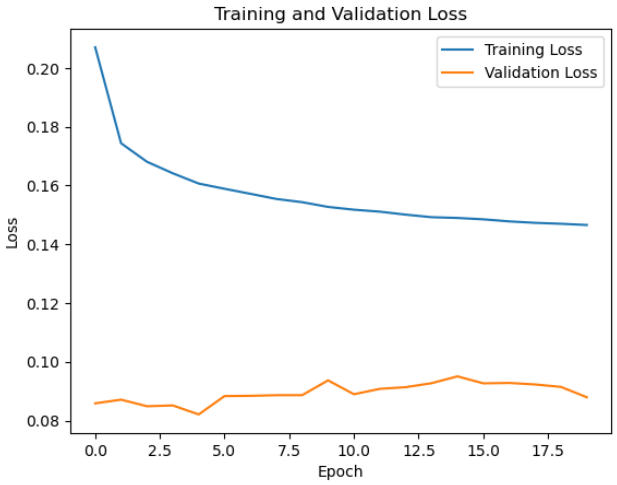
  model_hidden = 128

  model_nblocks = 3

  num_epochs = 20

ML_Parkinson's Freezing of Gait Prediction - Version 9 **0.278701** **0.250984**

Succeeded (after deadline) · 1d ago · Notebook ML_Parkinson's Freezing ...



V15 without nn.BatchNorm1d(out_features)

model_dropout = 0.2

model_hidden = 32

model_nblocks = 1

num_epochs = 5

ML_Parkinson's Freezing of Gait Prediction - Version 15 0.190200 0.224547

Succeeded (after deadline) · 23m ago · Notebook ML_Parkinson's Freezing of Gait Prediction | Version 15

3. Conclusion

I. 根據 v1 v4 比較，我們可以看出將模型的深度加深可以提升正確率，因為更深的模型可以更好地擬合複雜的數據分佈，從而改善了模型的性能。

II. v6 的結果，將 dropout 提高後正確率反而降低，有兩種可能導致這個原因

1. dropout 比例過高時，模型可能會失去過多重要信息，導致模型的表徵能力下降，從而影響了模型的性能。
2. 當 dropout 比例過高時，模型可能會過度正則化，導致模型學習不足，無法充分利用訓練數據，從而影響了模型的泛化能力。

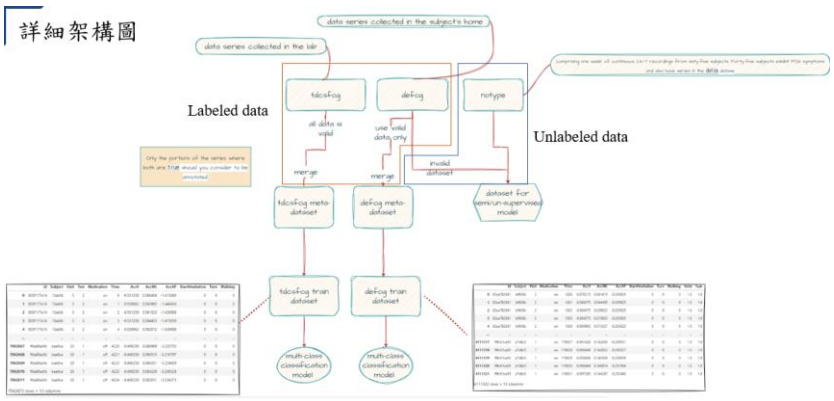
III. v9 的的方法將模型的中間層的神經元數增加，得到了最好的結果，我覺得有可能是因為增加中間層的神經元數可以增加模型的表示能力和學習能力，更好地擬合複雜的數據分佈，從而提高模型的性能。

IV. v9 取得了最佳的分數，但根據所繪製出來的摺線圖 validation loss 有稍微往上升，可能有 overfitting 的發生，但也有可能是正常的波動值，這點可能需要更多的迭代次數來觀察

V. v1 與 v15 差載有沒有做正規化，而結果顯示沒做正規化的結果比較差，以下是沒有做 normalization 可能會造成的影響

1. 收斂速度變慢： Batch Normalization 有助於加速模型的收斂速度。它通過對每個批次的輸入進行標準化，使得輸入的分佈更加穩定，從而有助於減少梯度消失和爆炸問題，加速收斂。
2. 訓練不穩定： Batch Normalization 有助於減少內部協變量偏移（Internal Covariate Shift），從而使得模型更加穩定。如果沒有 Batch Normalization，模型可能會更容易受到訓練數據中的一些特定統計特徵的影響，導致訓練過程變得不穩定。
3. overfitting 風險增加： Batch Normalization 有助於防止過擬合，因為它增加了模型的泛化能力。如果沒有 Batch Normalization，模型可能會更容易過擬合訓練數據。

VI. 我們所用的資料集是使用"tdcsfog" 和 "defog"，根據 PPT 中所給的介紹，分別為 lable data 與 unlable date 所組成的訓練資料，所以我認為這次的作業是使用半監督式學習的方法。



4. Discussion

我花了一些時間去了解程式碼的每個部份的作用，對我學習有時分大的幫助。而我所遇到的困難點是因為我的程式是用 kaggle 的雲端資源去運算，在訓練的時候需要花一段時間，而繳交結果時也還要花一段時間跑，而且跑得比 train 更久，所以如果要是各種不同超參數並得到最終分數的話就要花許多時間去嘗試，我認為這對我是這項作業的一項困難點。而由於我對於數據的理解程度有限，因此調整模型的過程變得更加困難。缺乏對數據特徵和分佈的深入理解可能會導致模型性能的提升空間受到限制，需要更多的實驗和試錯來找到最佳的模型配置。