系級：大氣二　　學號：B08209006　　姓名：謝晉維

所有程式及資料都可以在以下網址找到
https://github.com/Aaron-Hsieh-0129/ESOE-Programming

壹、這學期收集的程式以及整理方法

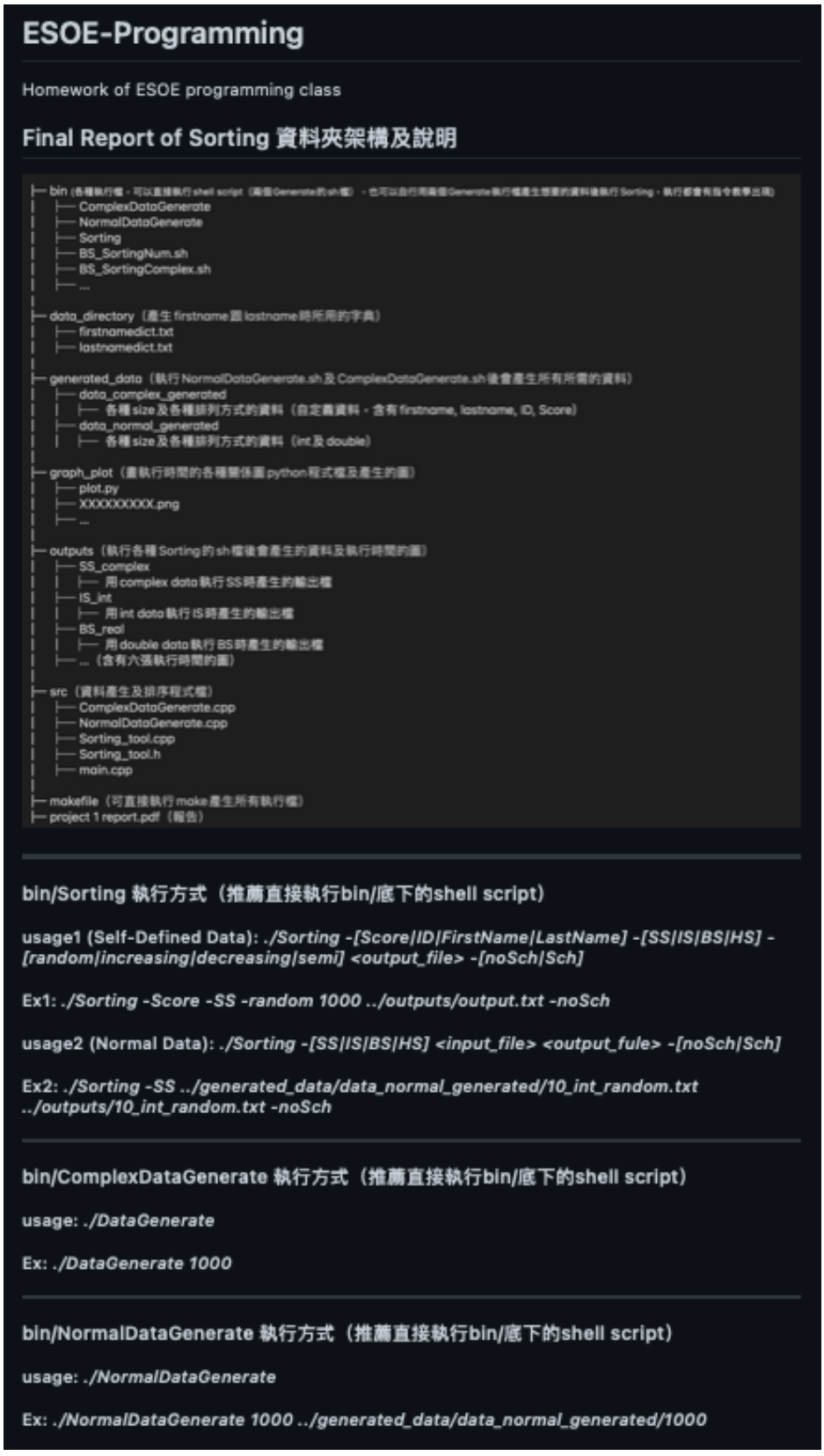
各種資料夾整理形式
bin/底下放的是執行檔以及shell script
src/底下放的是.cpp檔以及.h檔
outputs/底下放的是程式輸出的結果
generated_data/或是data_generated/底下放的是利用已知字典產生的資料
data_dictionary/底下放的是要用到的字典
graph_plot/底下放的是畫圖的python檔案以及產生的圖

I. 作業一、作業二、作業三、Final Project of Sorting
Sorting：
- Selection Sort
- Insertion Sort
- Bubble Sort
- Merge Sort
- Heap Sort
- Quick Sort（有實作但電腦記憶體不夠註解掉了，若是電腦夠好可以去main.cpp把註解拿掉）

整理方法：
a. 在src/底下，有四個.cpp file即一個.h file
- Sorting_tool.cpp是寫各種排序的方法以及需要運用的comparison function，Sorting_tool.h裡面是宣告在編譯執行檔時會共同用到的函式以及變數。
- NormalDataGenerate.cpp及ComplexDataGenerate.cpp則分別是用來產生純數字的資料（有random、increasing、decreasing、semi ordered data）以及Complex Data（含有ID、Score、FirstName、LastName，皆有random、increasing、decreasing、semi ordered data）。
- main.cpp是在寫各種I/O的檔案操作，以及各種函數利用整合的程式

b. 在資料夾一開始有完成一個makefile，此檔案是用來直接編譯所有寫好的程式

c. graph_plot資料夾底下存了一個畫所有圖的python程式



ESOE-Programming

Homework of ESOE programming class

Final Report of Sorting 資料夾架構及說明

```
├─ bin (各種執行檔，可以直接執行 shell script（兩個Generate的.sh檔），也可以自行用兩個Generate執行檔產生想要的資料後執行 Sorting，執行都會有指令教學出現)
|    ├── ComplexDataGenerate
|    ├── NormalDataGenerate
|    ├── Sorting
|    ├── BS_SortingNum.sh
|    ├── BS_SortingComplex.sh
|    ├── ...
|
├─ data_directory（產生 firstname 跟 lastname 時所用的字典）
|    ├── firstnamedict.txt
|    ├── lastnamedict.txt
|
├─ generated_data（執行 NormalDataGenerate.sh 及 ComplexDataGenerate.sh 後會產生所有所需的資料）
|    ├── data_complex_generated
|    |    ├── 各種size及各種排列方式的資料（自定義資料，含有 firstname, lastname, ID, Score）
|    ├── data_normal_generated
|    |    ├── 各種size及各種排列方式的資料（int及double）
|
├─ graph_plot（畫執行時間的各種關係圖 python 程式檔及產生的圖）
|    ├── plot.py
|    ├── XXXXXXXXX.png
|    ├── ...
|
├─ outputs（執行各種 Sorting 的 sh 檔後會產生的資料及執行時間的圖）
|    ├── SS_complex
|    |    ├── 用 complex data 執行 SS 時產生的輸出檔
|    ├── IS_int
|    |    ├── 用 int data 執行 IS 時產生的輸出檔
|    ├── BS_real
|    |    ├── 用 double data 執行 BS 時產生的輸出檔
|    ├── ...（含有六張執行時間的圖）
|
├─ src（資料產生及排序程式檔）
|    ├── ComplexDataGenerate.cpp
|    ├── NormalDataGenerate.cpp
|    ├── Sorting_tool.cpp
|    ├── Sorting_tool.h
|    ├── main.cpp
|
├─ makefile（可直接執行 make 產生所有執行檔）
├─ project 1 report.pdf（報告）
```

---

bin/Sorting 執行方式（推薦直接執行bin/底下的shell script）

usage1 (Self-Defined Data): *./Sorting -[Score|ID|FirstName|LastName] -[SS|IS|BS|HS] -[random|increasing|decreasing|semi] <output_file> -[noSch|Sch]*

Ex1: *./Sorting -Score -SS -random 1000 ../outputs/output.txt -noSch*

usage2 (Normal Data): *./Sorting -[SS|IS|BS|HS] <input_file> <output_fule> -[noSch|Sch]*

Ex2: *./Sorting -SS ../generated_data/data_normal_generated/10_int_random.txt ../outputs/10_int_random.txt -noSch*

---

bin/ComplexDataGenerate 執行方式（推薦直接執行bin/底下的shell script）

usage: *./DataGenerate*

Ex: *./DataGenerate 1000*

---

bin/NormalDataGenerate 執行方式（推薦直接執行bin/底下的shell script）

usage: *./NormalDataGenerate*

Ex: *./NormalDataGenerate 1000 ../generated_data/data_normal_generated/1000*

II. Hw4、Ex5、Ex6

Combination & Permutation

Eight Queen Problem

Sudoku

整理方法：

因為這幾個程式都不是大的project所以就只分成bin/、src/，使用makefile就可以直接在bin/底下產生執行檔，到bin/底下即可執行程式。

貳、說明你這學期有做了什麼特別的程式設計工作（python機器學習）

Coworker: B08209032沈世明（系上同學）

程式碼及各種資料可在以下連結找到

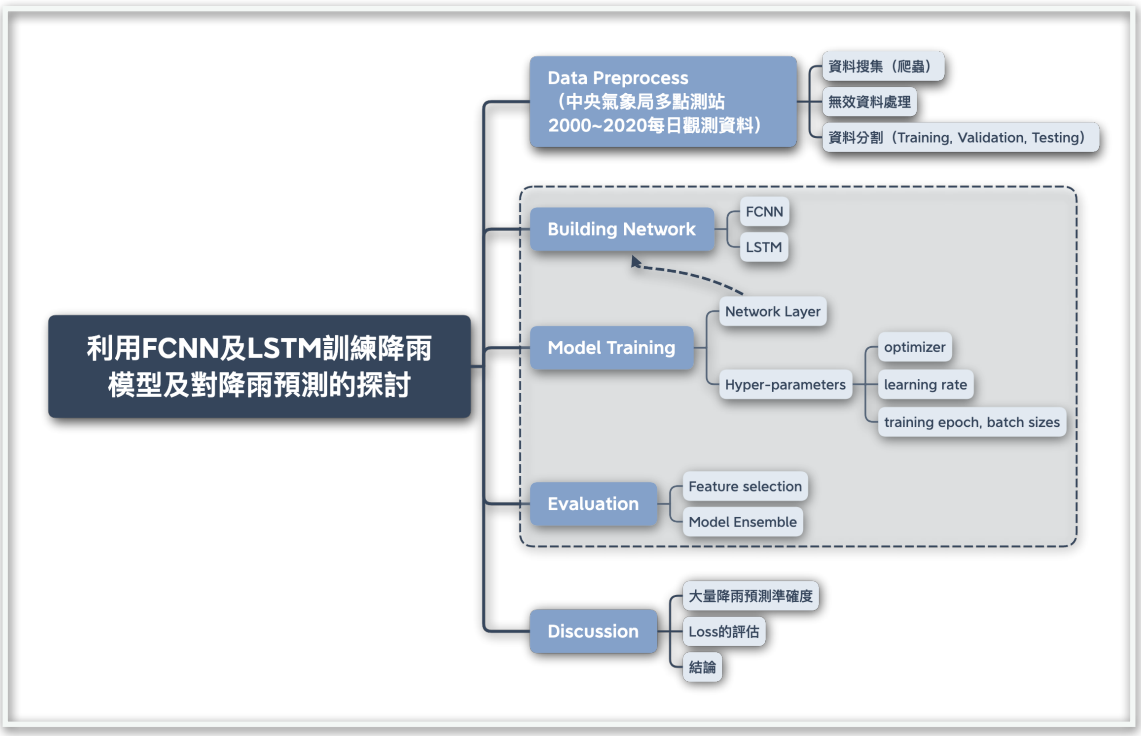
https://github.com/Aaron-Hsieh-0129/ESOE-Programming/tree/main/DNN%20%26%20LSTM

主題：

利用FCNN及LSTM訓練降雨模型及對降雨量預測的探討

若要執行程式請先執行爬蟲.ipynb抓資料再跑模型

- 模型架構



FCNN network

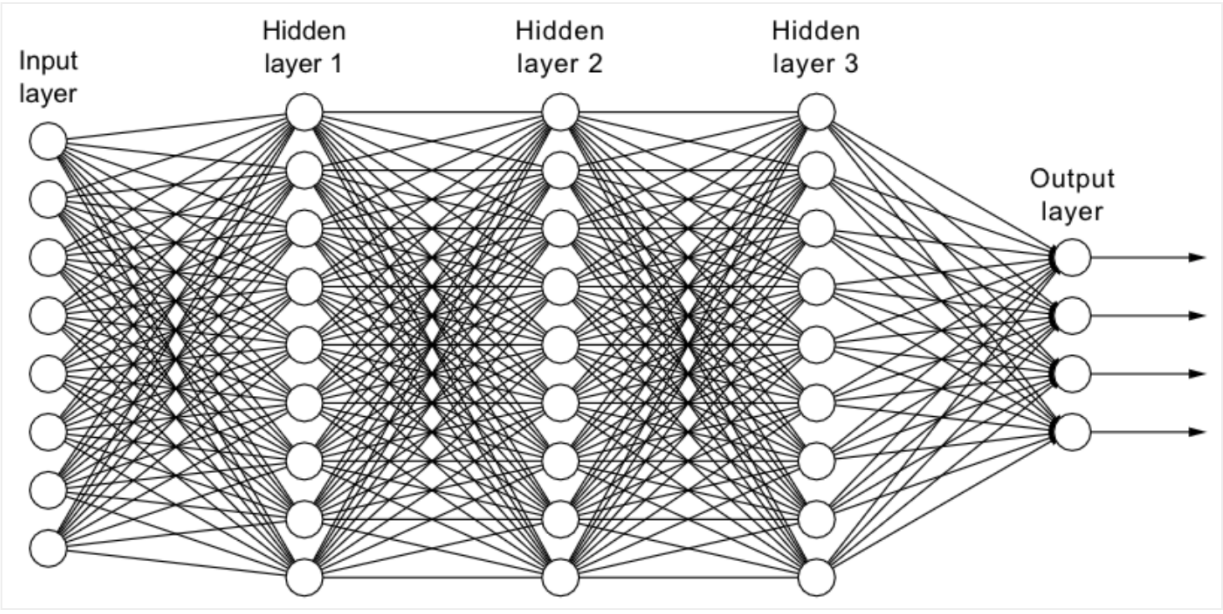


Figure 1 An example of a fully connected neural network (fcNN) model with three hidden layers

https://freecontent.manning.com/neural-network-architectures/

LSTM（長短型記憶模型）

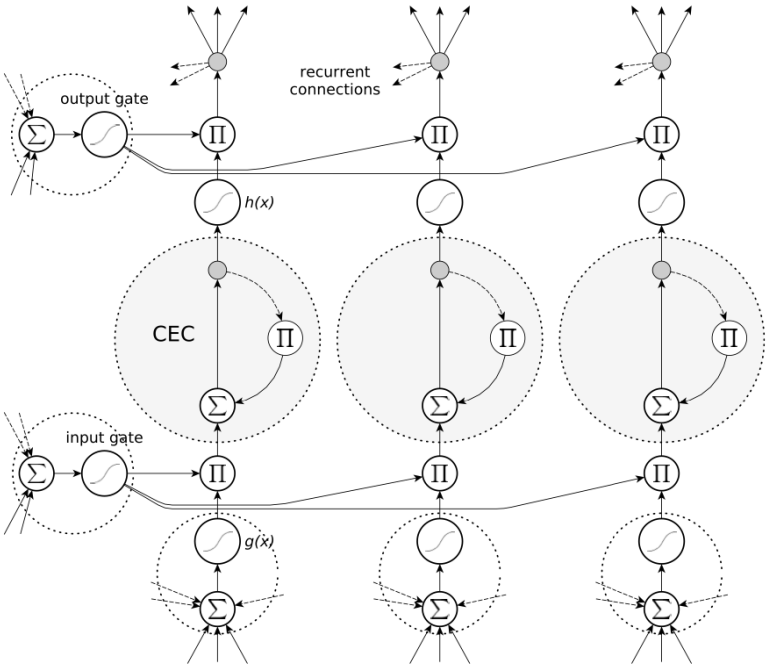
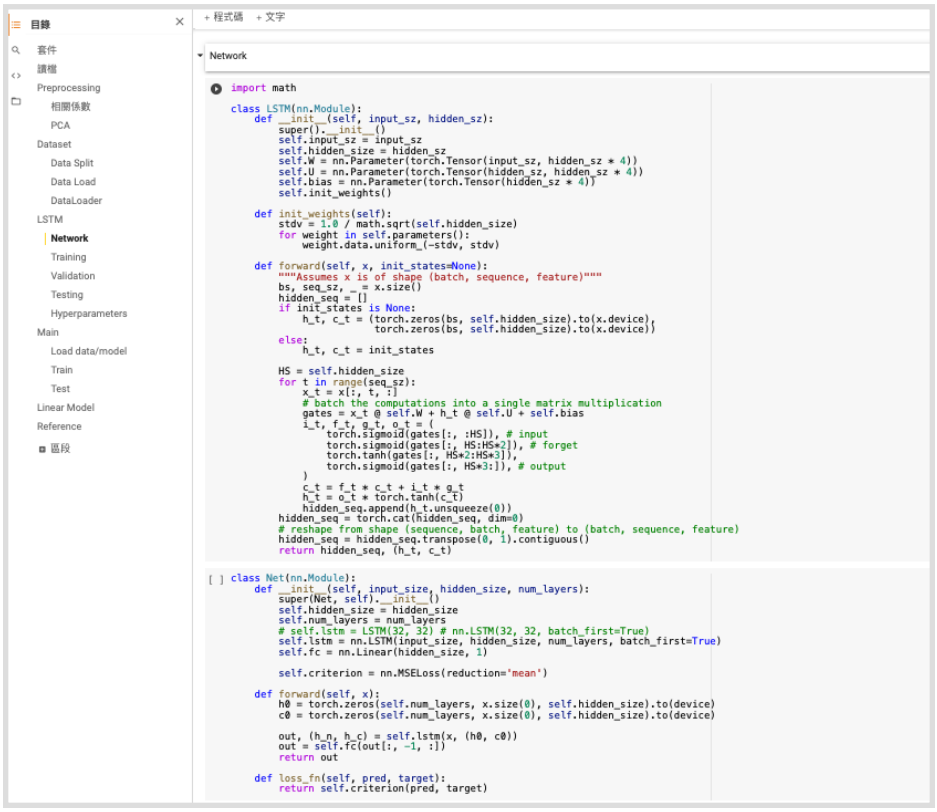


Figure 12: A three cell LSTM memory block with recurrent self-connections

https://arxiv.org/abs/1909.09586

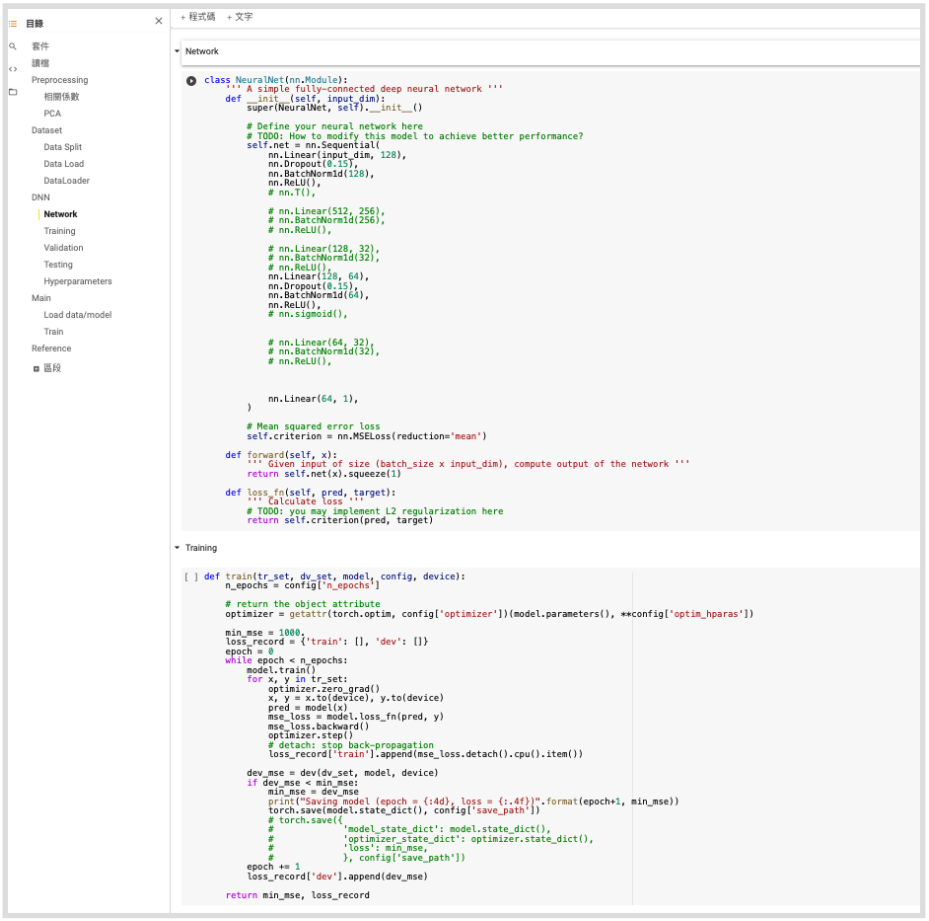
- 此處僅提供模型架構的code，其他各種資料處理請到github上查看

LSTM



```
目錄
套件
讀檔
Preprocessing
相關係數
PCA
Dataset
Data Split
Data Load
DataLoader
LSTM
Network
Training
Validation
Testing
Hyperparameters
Main
Load data/model
Train
Test
Linear Model
Reference
區段

+ 程式碼  + 文字

Network

import math

class LSTM(nn.Module):
    def __init__(self, input_sz, hidden_sz):
        super().__init__()
        self.input_sz = input_sz
        self.hidden_size = hidden_sz
        self.W = nn.Parameter(torch.Tensor(input_sz, hidden_sz * 4))
        self.U = nn.Parameter(torch.Tensor(hidden_sz, hidden_sz * 4))
        self.bias = nn.Parameter(torch.Tensor(hidden_sz * 4))
        self.init_weights()

    def init_weights(self):
        stdv = 1.0 / math.sqrt(self.hidden_size)
        for weight in self.parameters():
            weight.data.uniform_(-stdv, stdv)

    def forward(self, x, init_states=None):
        """Assumes x is of shape (batch, sequence, feature)"""
        bs, seq_sz, _ = x.size()
        hidden_seq = []
        if init_states is None:
            h_t, c_t = (torch.zeros(bs, self.hidden_size).to(x.device),
                        torch.zeros(bs, self.hidden_size).to(x.device))
        else:
            h_t, c_t = init_states

        HS = self.hidden_size
        for t in range(seq_sz):
            x_t = x[:, t, :]
            # batch the computations into a single matrix multiplication
            gates = x_t @ self.W + h_t @ self.U + self.bias
            i_t, f_t, g_t, o_t = (
                torch.sigmoid(gates[:, :HS]), # input
                torch.sigmoid(gates[:, HS:HS*2]), # forget
                torch.tanh(gates[:, HS*2:HS*3]),
                torch.sigmoid(gates[:, HS*3:]), # output
            )
            c_t = f_t * c_t + i_t * g_t
            h_t = o_t * torch.tanh(c_t)
            hidden_seq.append(h_t.unsqueeze(0))
        hidden_seq = torch.cat(hidden_seq, dim=0)
        # reshape from shape (sequence, batch, feature) to (batch, sequence, feature)
        hidden_seq = hidden_seq.transpose(0, 1).contiguous()
        return hidden_seq, (h_t, c_t)

[ ] class Net(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers):
        super(Net, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        # self.lstm = LSTM(32, 32) # nn.LSTM(32, 32, batch_first=True)
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, 1)

        self.criterion = nn.MSELoss(reduction='mean')

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)

        out, (h_n, h_c) = self.lstm(x, (h0, c0))
        out = self.fc(out[:, -1, :])
        return out

    def loss_fn(self, pred, target):
        return self.criterion(pred, target)
```

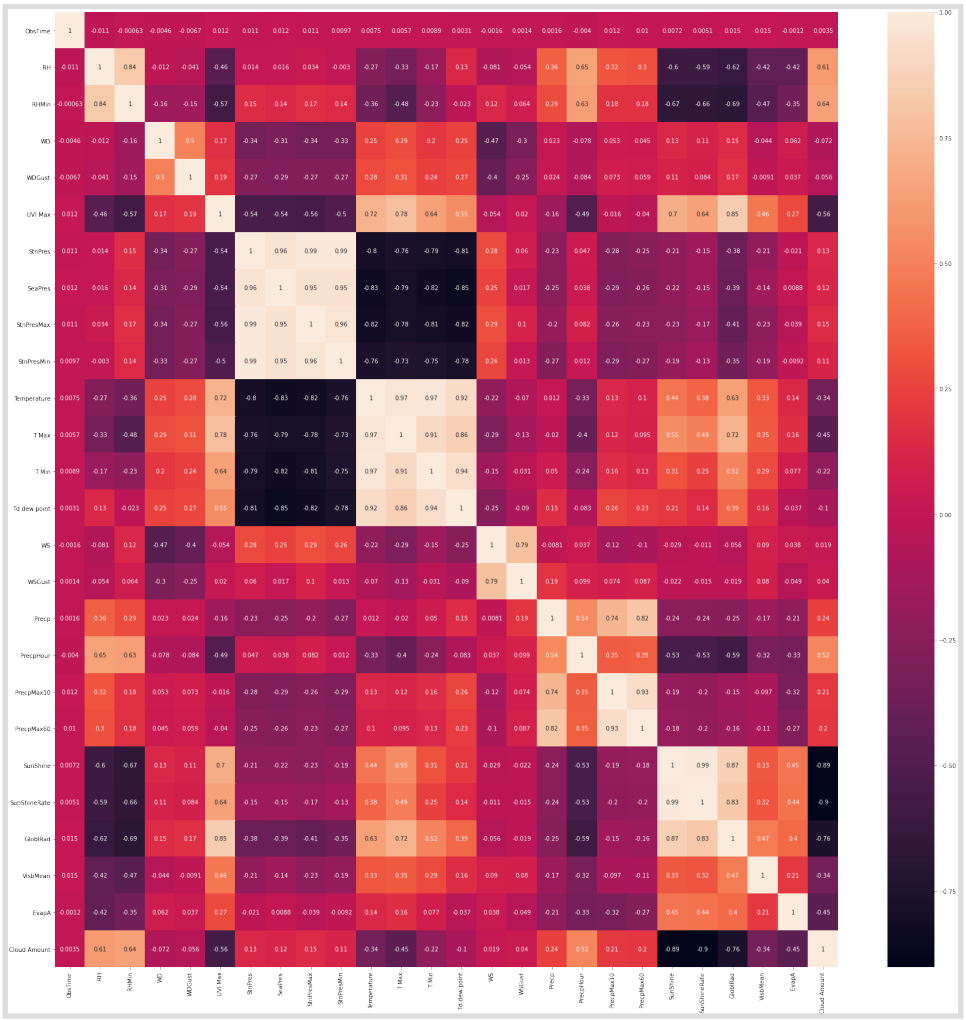
FCNN



```
目錄
套件
讀檔
Preprocessing
相關係數
PCA
Dataset
Data Split
Data Load
DataLoader
DNN
Network
Training
Validation
Testing
Hyperparameters
Main
Load data/model
Train
Reference
區段

+ 程式碼  + 文字

Network

class NeuralNet(nn.Module):
    ''' A simple fully-connected deep neural network '''
    def __init__(self, input_dim):
        super(NeuralNet, self).__init__()

        # Define your neural network here
        # TODO: How to modify this model to achieve better performance?
        self.net = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.Dropout(0.15),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            # nn.T(),

            # nn.Linear(512, 256),
            # nn.BatchNorm1d(256),
            # nn.ReLU(),

            # nn.Linear(128, 32),
            # nn.BatchNorm1d(32),
            # nn.ReLU(),
            nn.Linear(128, 64),
            nn.Dropout(0.15),
            nn.BatchNorm1d(64),
            nn.ReLU(),
            # nn.sigmoid(),


            # nn.Linear(64, 32),
            # nn.BatchNorm1d(32),
            # nn.ReLU(),


            nn.Linear(64, 1),
        )

        # Mean squared error loss
        self.criterion = nn.MSELoss(reduction='mean')

    def forward(self, x):
        ''' Given input of size (batch_size x input_dim), compute output of the network '''
        return self.net(x).squeeze(1)

    def loss_fn(self, pred, target):
        ''' Calculate loss '''
        # TODO: you may implement L2 regularization here
        return self.criterion(pred, target)

Training

[ ] def train(tr_set, dv_set, model, config, device):
    n_epochs = config['n_epochs']

    # return the object attribute
    optimizer = getattr(torch.optim, config['optimizer'])(model.parameters(), **config['optim_hparas'])

    min_mse = 1000.
    loss_record = {'train': [], 'dev': []}
    epoch = 0
    while epoch < n_epochs:
        model.train()
        for x, y in tr_set:
            optimizer.zero_grad()
            x, y = x.to(device), y.to(device)
            pred = model(x)
            mse_loss = model.loss_fn(pred, y)
            mse_loss.backward()
            optimizer.step()
            # detach: stop back-propagation
            loss_record['train'].append(mse_loss.detach().cpu().item())

        dev_mse = dev(dv_set, model, device)
        if dev_mse < min_mse:
            min_mse = dev_mse
            print("Saving model (epoch = {:4d}, loss = {:.4f})".format(epoch+1, min_mse))
            torch.save(model.state_dict(), config['save_path'])
            # torch.save({
            #             'model_state_dict': model.state_dict(),
            #             'optimizer_state_dict': optimizer.state_dict(),
            #             'loss': min_mse,
            #             }, config['save_path'])
        epoch += 1
        loss_record['dev'].append(dev_mse)

    return min_mse, loss_record
```

- Feature Selection（各個參數對降雨的相關係數）

在調整模型的時候可以挑features進行訓練，讓model更能夠fit降雨



- 結論

- 大雨預測準確度

1. 全部Data訓練不做feature selection

Linear Model: 0.28, FCNN: 0.68, LSTM: 0.57

2. 全部Data訓練有做feature selection

Linear Model: 0.17, FCNN: 0.69, LSTM: 0.75

3. 有降雨Data訓練不做feature selection

Linear Model: 0.17, FCNN: 0.63, LSTM: 0.33

4. 有降雨Data訓練有做feature selection

Linear Model: 0.15, FCNN: 0.73, LSTM: 0.35

討論：

1. 對於預測強降雨這種問題，機器學習非常有用，在實務上可以預先訓練好模型之後只要用這個模型就可以快速發布警報
2. 輸入所有資料做feature selection及model ensemble的確可以讓模型表現變比沒有做還要好

- 預測降雨量在Testing set上Loss（MSE：殘差平方和）的評估（越小越好）

1. 全部Data訓練不做feature selection
Linear Model: 144.8, FCNN: 121.4, LSTM: 109.6
2. 全部Data訓練有做feature selection
Linear Model: 164.8, FCNN: 183.1, LSTM: 122.5
3. 有降雨Data訓練不做feature selection
Linear Model: 305.3, FCNN: 364.6, LSTM: 122.5
4. 有降雨Data訓練有做feature selection
Linear Model: 270.9, FCNN: 406.9, LSTM: 283.8

討論：

1. 對於預測實際降雨量這個問題太大了，雖然模型表現不錯但誤差平均來說還有10幾mm左右（部分原因可能是training data不夠）
2. 只有降雨才訓練loss很難降下來（資料量不足）
3. 有做feature selection及model ensemble都會變好

總結：

1. 機器學習在大氣中的應用層面非常廣且表現也非常不錯
2. 機器學習在大氣科學上的限制比較多是在我們很難詮釋機器為什麼會學到這樣的結論，若要更深入研究因此需要一些explainable AI的技術。