第一輪實驗:

Run 45次: 5次做成(finalavg\_result, finalZ\_result)

ZonePBDVR+\_1cost

run main.py ->產生 txt\_result

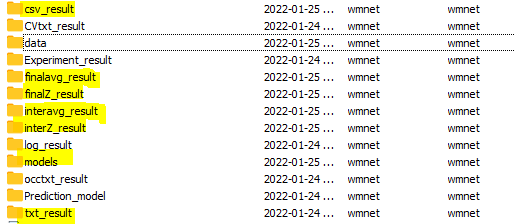
run txt\_to\_csv.py ->產生 csv\_result

run vehavg\_inst.py ->產生 interavg\_result, interZ\_result

run finalavg.py ->產生 finalavg\_result, finalZ\_result

run train.py ->產生 models 資料夾

其它是一些 log檔可忽略



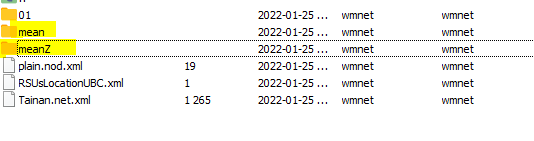
第二輪實驗以後使用此份code:

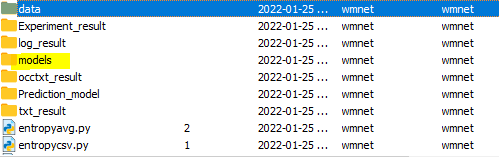
ZonePBDVR+\_1cost2

將ZonePBDVR+\_1cost產生的finalavg\_result 重新命名為 mean , finalZ\_result 重新命名為 meanZ 放入data 資料夾

將ZonePBDVR+\_1cost產生的models 資料夾，放在與main.py 同一層

即可執行





file 用途:

main.py ->產生 txt\_result

txt\_result: 紀錄: 車輛ID、車輛碰觸前面sensor的時間、車輛碰觸中間sensor的時間、車輛碰觸中間sensor的速度、車輛碰觸後方sensor的時間（记录以车辆为对象的真实数据）

txt\_to\_csv.py ->產生 csv\_result

csv\_result: 紀錄:  車輛碰觸前面sensor的時間(entry)、 車輛碰觸中間sensor的時間(mid)、 車輛碰觸中間sensor的速度(instspeed)、 車輛碰觸後方sensor的時間(exit)、道路長度(length)（记录以车辆为对象的真实数据，但是没有了车辆ID）

run vehavg\_inst.py ->產生 interavg\_result, interZ\_result

interavg\_result: 紀錄每次實驗各個道路每5分鐘平均/瞬時速度: interval(5分鐘一個間隔)、instspeed(每5分鐘的平均瞬時速度)、avgspeed(每5分鐘的平均速度)

,interZ\_result:  紀錄每次實驗各個道路每5分鐘的Zmax、Zmin、qvalue

run finalavg.py ->產生 finalavg\_result, finalZ\_result

finalavg\_result: 平均前五次實驗的interavg\_result/out0~4的file，當作 mean

finalZ\_result: 平均前五次實驗的interavg\_result/out0~4的file，當作 meanZ（看来可能不需要final这一层）

run train.py ->產生 models 資料夾， 使用interavg\_result/out5~outx的所有file 減去finalavg\_result (mean)來訓練模型（这一层那么就不用再递减, 可以直接训练了）

思路：已经有了每辆车通过的速度统计，则可以知道平均的瞬时速度；间隔速度，瞬时速度，平均速度确定下来；

数据总归能统计，但是需要看在LSTM调用处是如何使用的；

LSTM先预测平均速度，

DB如果高 truncated normal distribution: speedFactor="normc(mean,deviation,lowerCutOff,upperCutOff)" speedfactor;

关键是LSTM如何预测

应该把Model集中起来，考虑将文档集中程度；

每走一步都能够预测到model的数据

每五分钟的数据必须要记录一下。

作为预测的基准，如果没有车辆则按照车辆最大速度开启；

大概逻辑已经确定好，要做能够支撑商业化的prod

真实数据还在跑，3预测1 绝对还是不行（(28, 3, 1)），

还是12预测1比较稳妥；

要想活得高回報數據，必須順著模擬器的特征來操作；

數據問題：

1 在轉到CSV的時候用退出時間做了過濾

data = data.sort\_values(by=["exit"])

2

A：當前修改內容盡量不要觸碰現有數據，先從現有數據中進行合併

B: 現有數據可能需要擴充，看看能不能就當前的數據進行填充。先不破壞現有數據結構。 中間某一行在內存中插入，然後再寫回。

C：數據準備好以後，就可以來做預測了；

數據特征：*veh\_mid是loop的瞬時平均速度*

*line = str(veh\_id)+","+str(Lane\_dict[laneid][veh\_id]["veh\_entry"])+","+str(Lane\_dict[laneid][veh\_id]["veh\_midentry"])+","+\  
str(Lane\_dict[laneid][veh\_id]["veh\_mid"])+","+str(Lane\_dict[laneid][veh\_id]["veh\_exit"])+"\n"*

**插入数值**的原理： 进入时间与退出时间任意一个大于5分钟，则插入数值；

中间点如果没有则添加一个

收尾都要看 与最终的simtime，需要保存到一个文件summary中；

entry,mid,instspeed,exit,length

215.80886759292068,219.25947617233433,13.04531031785959,265.44282864616326,131.93

221.72157200377262,225.3736857581853,11.828418946488181,268.2037201090481,131.93

240.15389518832143,244.20205426899187,11.551188598274893,272.8680856409443,131.93

241.93040120071123,246.03531990077295,11.949596307329893,274.8081325185905,131.93

最主要是插入數值以後，一定要能有數值存在；

所以時根據退出時間來制定規則的，所以只需要在退出時間階段進行插值即可。

看能否從配置文件中獲取到咨詢；node節點等信息；

把实验部分做完，

同时尽快积累项目经验，能够把C++的开源代码搞定，

能够debug并且调整C++写的开源代码；

由于下图原因，将瞬时速度的统计采用时间再排序一次

