這裡放我碩士論文相關的程式碼

有六個ipynb檔

可以分成四個分類：

1. Rule-based\_PairsTrading.ipynb

主要是在計算Rule-based Pairs Trading之每日獲利，並把他存成.npy

給每一個NN model當作baseline使用

2. RelationNetwork\_Benchmark.ipynb

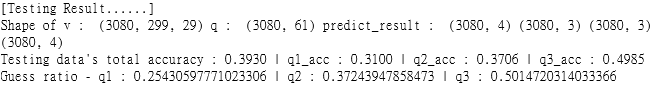
主要用來建立我的「1D Conv RN」、「2D Conv RN」以及基本的「多層MLP」

＊註：

要注意的是

正確的accuracy是我額外用”modelEvaluate” 這個function寫預測出來的才對

（結果如下圖所示，只有在model fit完最後一個epoch才會顯示）



Training過程每一個epoch的model.evaluate只是我用來看loss跟決定earlystopping的

model.evaluate與modelEvaluate的accuracy結果差異在於：

處理multi-output過程的樣本母體不一樣，

前者其他問題類別的答案也會被放到acc計算的分母，

後者是每個問題類別的答案會先區分好後才計算「那個問題類別的準確率」

導致上面的問題是keras的multi-output不像pytorch可以根據每一個output所對應到的label算loss和accuracy，而是要每個問題都用一樣的樣本數量做model.fit

舉個簡單例子：「如果某一個epoch testing data sample出總共1800個vqa pair，然後屬於Q1, Q2, Q3的問答樣本數分別是600, 600, 600」

(1) Pytorch可以對Q1, Q2, Q3個別算loss，然後加總當作total loss

（也就是說Q1\_loss是由600個樣本算出來的，Q2\_loss和Q3\_loss也是）

(2) Keras的做法必須是額外在最後一層layer多一個unit或者把其他問題類別的答案塞到一個不重要的class，然後才一次model.fit "這1800個樣本" 去算每個問題類別的loss，也就是說Q1, Q2, Q3的loss都是由"1800"個樣本算出來的，只是其他1200個問題類別的樣本會額外新增一個class或者不重要的class當作這個output layer的答案

3. AttGGSNN\_1DConv.ipynb

(1) 主要用來建立我的「Attention-based GGSNN」

(2) AttGGSNN\_NoAttention.ipynb和AttGGSNN\_AllConnectGraph\_NoAttention.ipynb其實也都是根據這個AttGGSNN\_1DConv.ipynb做出來的，主要差異如下：

－　AttGGSNN\_AllConnectGraph\_NoAttention.ipynb

不使用我的attention，為基本的GGSNN，而且圖是全連接圖（代表所有的點彼此相鄰、相鄰矩陣所有值為1）

－　AttGGSNN\_NoAttention.ipynb

不使用我的attention，為基本的GGSNN

4. AttGGSNN\_GRUCell.ipynb

(1) 主要用來建立我的GRUCell版本的AttGGSNN，目的在於建立一個可以動態選擇時間點並且更新圖狀態的網路，結果輸Conv版本的GGSNN而有贏RN

(2) 裡面的GraphUpdater有使用到Skip connection的概念，詳見程式碼

(3) Node feature的embedding network分成三個：conv1d, gru, gru\_2

只要修改參數"embedding\_method"即可使用，詳細請見程式碼注解

也就是說這份code裡面的conv1d版本也可以使用，和3. 最大的差別就是在updater加上skip connection的概念



每個function的作用詳見ipynb檔

所有Graph系列的ipynb說明與註釋以AttGGSNN\_1DConv.ipynb為主

(註：AttGGSNN\_GRUCell.ipynb是8月多的版本，有新增一些小function，請以新的function解釋為主)

第一次跑code的時候，generateAndsave請設定成1

把data內除了trade-2017.csv.gz這個檔案以外的檔案放到跟.ipynb同一個目錄下

讀檔路徑要改成自己的server/dir

改完直接按jupyter notebook的”Kernel” 然後選擇"Restart & Run All"即可執行