國立清華大學
工業工程與工程管理學系

智慧化企業整合 Final Project

指導教授： 邱銘傳 教授
學　　號： 110034535
學　　生: 黃于瑞

中華民國　一百十一　年　一　月

摘要

本次期末報告主題是「利用深度學習的方法來預測交通流量」，準確的交通流預測在智能交通系統（ITS）中很重要，尤其是對於交通流量的控制。過往的方法ARMA、ARIMA等模型主要是線性模型，無法描述交通流的隨機性和非線性性質。近年來，基於深度學習的方法已被用作交通流預測的新替代方案。然而，哪種深度神經網絡最適合用於交通流預測的模型仍未解決。在本次期末報告中，使用長短期記憶（LSTM）和門控循環單元（GRU）神經網絡方法以及堆疊自編碼器神經網路來預測短期交通流量，並且實驗證明基於循環神經網絡（RNN）的深度學習方法如LSTM 和 GRU 以及堆疊自編碼器神經網路可以得到高效能的預測結果。

一、 緒論

近年來短期交通量預測的應用日益受到重視，許多先進交通旅行者資訊系統及先進交通管理系統(ATMS)的應用都需要估計及預測路網之交通狀況，主要其目的在於提供有用之旅行資訊給旅行者及提升整體路網的效率。透過道路上各種不同偵測器所蒐集到之交通狀況歷史資料，我們可以掌握道路上之即時資訊並用來估計目前的交通狀況及預測短期內可能發生的交通狀況。目前大部分交通流量預測的文獻都著重在高速公路流量的預測，事實上市區道路的交通流量預測由於還要考慮機車、紅綠燈的影響，且市區路網也較高速公路路網複雜許多，因此精準且有效的預測交通流量是不可或缺的一環。

1. 問題定義

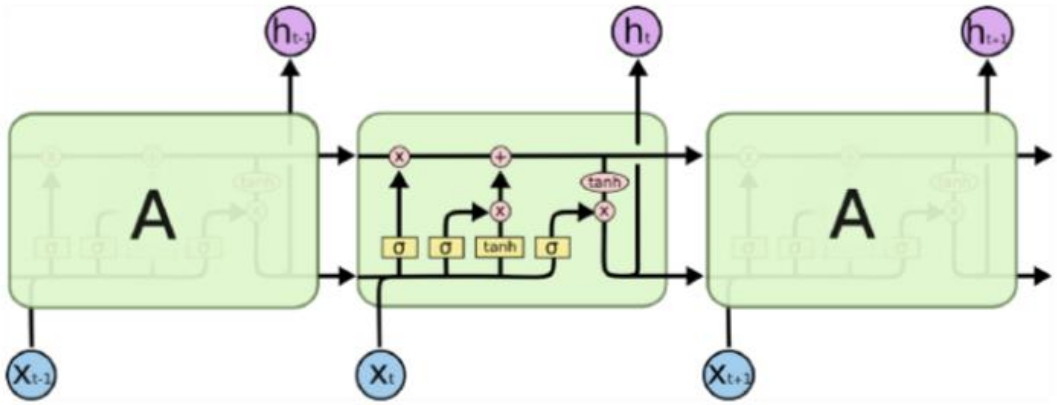
以下透過 5W1H 手法進行問題定義分析：

What?	精準的預測交通流量
When?	提供政府交通部門系統有效的資料量
Who?	政府部門、交通旅遊的民眾
Where?	一般道路以及高速公路
Why?	可以根據預測狀況來調整交通規劃
How?	深度學習

二、 方法

1 長短期記憶（英語：Long Short-Term Memory LSTM）

循環神經網絡（Recurrent Neural Networks，RNN)是一種擅於處理序列數據的模型，例如文本、時間序列、股票市場等。但是傳統的 RNN 架構存在所謂的梯度消失問題。為了克服這種缺點，提出了某些 RNN 結構，例如 LSTM，其設計目的是讓記憶細胞能夠確定何時忘記某些信息，從而確定時間序列問題的最佳時間滯後。由於其長期存在的記憶能力，這些特徵對於交通領域的短期交通流預測特別有效，尤其像是這種有關時間序列的問題，畢竟前一段時間的交通流量一定會影響到下一段時間的交通流量。典型的 LSTM 由一個輸入層、一個以記憶塊為基本單元的循環隱藏層和一個輸出層組成。存儲塊包含具有自連接的存儲單元，用於存儲時間狀態，以及三個自適應的乘法門控單元：輸入、輸出和遺忘門，用於控制塊中的信息流。三個額外的門提供對塊的寫入、讀取和重置操作的連續模擬。乘法門可以學習打開和關閉，因此 LSTM 內存單元可以長時間存儲和訪問信息，從而緩解梯度消失問題。LSTM架構如下圖所示。



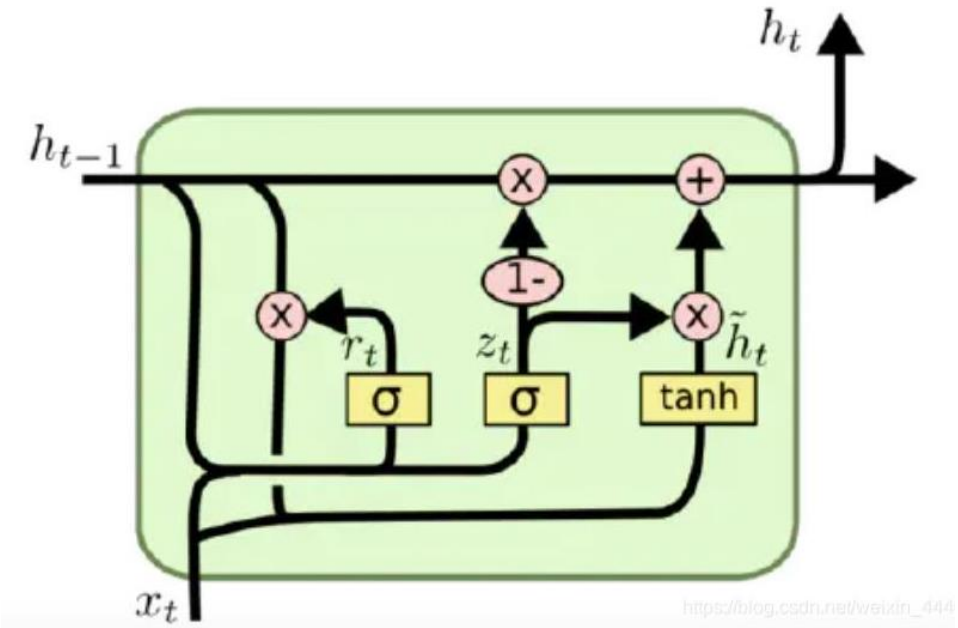
另外程式碼的部分從keras recurrent 去import LSTM，並且建構網路，程式碼參考如下。

```
from keras.layers import Dense, Dropout, Activation
from keras.layers.recurrent import LSTM, GRU
from keras.models import Sequential
```

```
def get_lstm(units):
    """LSTM(Long Short-Term Memory)
    Build LSTM Model.

    # Arguments
        units: List(int), number of input, output and hidden units.
    # Returns
        model: Model, nn model.
    """

    model = Sequential()
    model.add(LSTM(units[1], input_shape=(units[0], 1), return_sequences=True))
    model.add(LSTM(units[2]))
    model.add(Dropout(0.2))
    model.add(Dense(units[3], activation='sigmoid'))

    return model
```

2. **GRU** (Gate Recurrent Unit)

它與 LSTM 同樣為 RNN（循環神經網路）的變體，也與 LSTM 同樣旨在解決 RNN 當中存在著的梯度問題。但 LSTM 也有個問題就是執行速度較慢，因此有學者在 2014 年提出了 Gated Recurrent Unit，用來加快執行速度及減少記憶體的耗用。GRU 則將LSTM 中的遺忘閥（forget gate）與輸入閥（input gate）用一個更新閥（update gate）取代，並把單元狀態（cell state）和隱藏狀態（ht）進行合併，計算新資訊的方式和 LSTM 也有所不同。關於 GRU 模型的架構與數學我這邊就簡單帶過，來談談他與 LSTM 的比較。 GRU 了一個 gate ，使用的參數較少，因此計算上比 LSTM 快上許多，而且表現並沒有因此下降多少（LSTM 還是比較強的）。GRU架構如下圖所示。



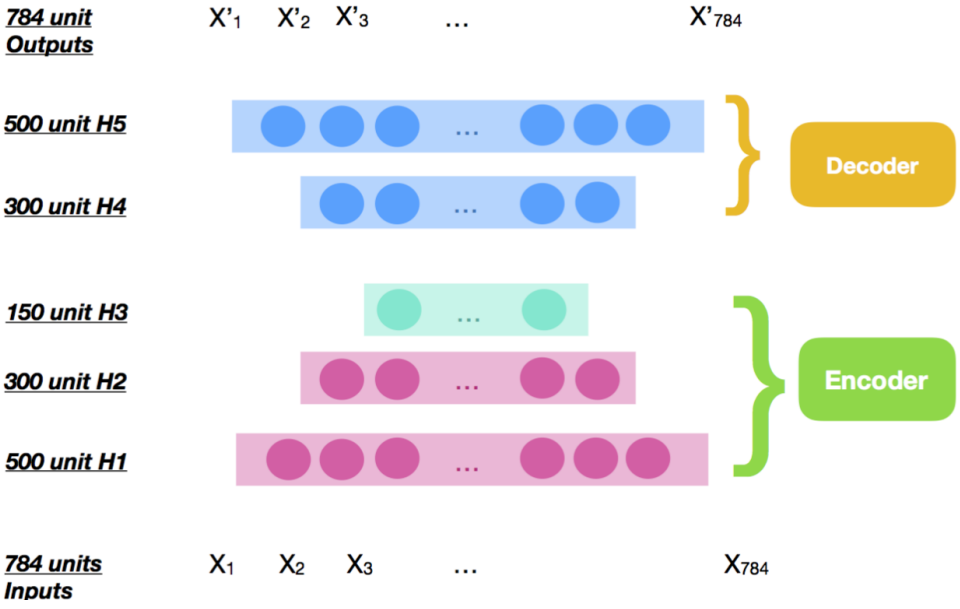
另外程式碼的部分從keras recurrent 去import GRU，並且建構網路，程式碼參考如下。

```
from keras.layers import Dense, Dropout, Activation
from keras.layers.recurrent import LSTM, GRU
from keras.models import Sequential
```

```
def get_gru(units):
    """GRU(Gated Recurrent Unit)
    Build GRU Model.

    # Arguments
        units: List(int), number of input, output and hidden units.
    # Returns
        model: Model, nn model.
    """

    model = Sequential()
    model.add(GRU(units[1], input_shape=(units[0], 1), return_sequences=True))
    model.add(GRU(units[2]))
    model.add(Dropout(0.2))
    model.add(Dense(units[3], activation='sigmoid'))

    return model
```

3. 堆疊自編碼器（Stacked Auto-Encoder, SAE）

AutoEncoder 做一件事，就是把輸入資料，複製一份到輸出。聽起來就是複製而已，好像很簡單。不過，典型的 AE 還會有所謂的內部表示層（Internal Representation），透過對內部表示層的一些維度限制、或加入雜訊到輸入資料，讓整個AE的結構更加複雜，整個流程就不是只有「複製輸入到輸出」，這樣單純而已了。AE 的結構跟 MLP（Multilayer perceptron）可說是如出一徹，但是有幾點限制：輸出的數量(神經元)要跟輸入一樣。結構上就是兩個部分，其一：一定會有個 Encoder 把輸入轉譯成內部表示層。其二：再來就會有 Decoder，把內部表示層轉譯成輸出。堆疊自編碼器是一種神經網絡，由多層稀疏自編碼器組成，其中每個隱藏層的輸出連接到連續隱藏層的輸入。如上圖所示，隱藏層通過無監督算法進行訓練，然後通過監督方法進行微調。堆疊式自編碼器主要包括三個步驟。第一步使用輸入數據訓練自動編碼器並獲取學習數據。第二步上一層的學習數據作為下一層的輸入，一直持續到訓練完成。第三部一旦所有隱藏層都訓練完畢，使用反向傳播算法來最小化成本函數，並使用訓練集更新權重以實現微調。Stacked Autoendocer 的最新進展是它提供了一個原始數據版本，其中包含非常詳細和有前景的特徵信息，用於訓練具有特定上下文的分類器，並找到比使用原始數據進行訓練更好的準確性。堆疊式自動編碼器通過嵌入層中的自動編碼器提高深度學習的準確性。SAES示意圖如下：



程式碼參考如下圖：

```
def get_saes(layers):
    """SAEs(Stacked Auto-Encoders)
    Build SAEs Model.

    # Arguments
        layers: List(int), number of input, output and hidden units.
    # Returns
        models: List(Model), List of SAE and SAEs.
    """
    sae1 = _get_sae(layers[0], layers[1], layers[-1])
    sae2 = _get_sae(layers[1], layers[2], layers[-1])
    sae3 = _get_sae(layers[2], layers[3], layers[-1])

    saes = Sequential()
    saes.add(Dense(layers[1], input_dim=layers[0], name='hidden1'))
    saes.add(Activation('sigmoid'))
    saes.add(Dense(layers[2], name='hidden2'))
    saes.add(Activation('sigmoid'))
    saes.add(Dense(layers[3], name='hidden3'))
    saes.add(Activation('sigmoid'))
    saes.add(Dropout(0.2))
    saes.add(Dense(layers[4], activation='sigmoid'))

    models = [sae1, sae2, sae3, saes]
```

三、實驗

1. 資料及以及資料前處理

本次實驗所採用的Datasets是數據來自 Caltrans Performance Measurement System (PeMS)。數據是從跨越加利福尼亞州所有主要大都市區的高速公路系統的各個探測器實時收集的。資料處理的部分使用了sklearn的preprocessing去切割資料，得到了7千多筆的訓練資料以及3千多的測試資料

7769	29/02/2016	13	1	100
7770	29/02/2016	19	1	100
7771	29/02/2016	17	1	100
7772	29/02/2016	13	1	100
7773	29/02/2016	19	1	100
7774	29/02/2016	6	1	100
7775	29/02/2016	14	1	100
7776	29/02/2016	11	1	100
7777	29/02/2016	10	1	100

3324	21/03/2016	105	1	100
3325	21/03/2016	107	1	100
3326	21/03/2016	92	1	100
3327	21/03/2016	96	1	100
3328	21/03/2016	90	1	100
3329	21/03/2016	110	1	100
3330	21/03/2016	98	1	100
3331	21/03/2016	90	1	100
3332	21/03/2016	103	1	100
3333				

資料錢處理的代碼如下:

```
Processing the data
"""
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler


def process_data(train, test, lags):
    """Process data
    Reshape and split train\test data.

    # Arguments
            train: String, name of .csv train file.
            test: String, name of .csv test file.
            lags: integer, time lag.
    # Returns
            X_train: ndarray.
            y_train: ndarray.
            X_test: ndarray.
            y_test: ndarray.
            scaler: StandardScaler.
    """
    attr = 'Lane 1 Flow (Veh/5 Minutes)'
    df1 = pd.read_csv(train, encoding='utf-8').fillna(0)
    df2 = pd.read_csv(test, encoding='utf-8').fillna(0)

    # scaler = StandardScaler().fit(df1[attr].values)
    scaler = MinMaxScaler(feature_range=(0, 1)).fit(df1[attr].values.r
    flow1 = scaler.transform(df1[attr].values.reshape(-1, 1)).reshape(
    flow2 = scaler.transform(df2[attr].values.reshape(-1, 1)).reshape(

    train, test = [], []
    for i in range(lags, len(flow1)):
            train.append(flow1[i - lags: i + 1])
    for i in range(lags, len(flow2)):
            test.append(flow2[i - lags: i + 1])
```

2. 實驗架構

再來我這次的實驗架構是拿這些訓練資料分別去三個模型進行訓練，所得到的權重來進行預測，另外將測試集去進行測試之後，再以各種指標去衡量這個模型。實驗架構如下圖所示。

3. 參數優化

三種不同網路架構除了預訓練時所獲得的最佳模型參數外，其餘的參數皆是使用相同的數據，還有超參數優化的空間。希望可以在有限的實驗次數中找到最佳模型配置來達到最高的準確率，節省時間以及減少人為調整的因素。我們利用了實驗設計中的田口方法，有效減少調整參數的總次數，並獲得相同的結果。我們選擇了四個因子分別是dropout rate、batch size、activate function、epcoh，並使用三個水準，應用 L9 實驗設計參數組合來幫助參數優化。

	Dropout	Batch size	Activate function	Epoch
LEVEL 1	0.2	128	tanh	500
LEVEL 2	0.3	256	Sigmoid	600
LEVEL 3	0.4	512	Relu	700

利用上述四因子三水準的minitab軟體跑出L9直交表。

Experiment	Dropout	Batch size	Activate function	Epoch
1	0.2	128	tanh	500
2	0.2	256	Sigmoid	600
3	0.2	512	Relu	700
4	0.3	128	Relu	700
5	0.3	256	Sigmiod	600
6	0.3	512	Tanh	500
7	0.4	128	Sigmoid	600
8	0.4	256	Relu	700
9	0.4	512	Tanh	500

最後得到LSTM最佳結果如下圖所示

Experiment	Dropout	Batch size	Activate function	Epoch	R-square	Mse
1	0.2	128	tanh	500	0.9218	100.562
2	0.2	256	Sigmoid	600	0.9399	98.929
3	0.2	512	Relu	700	0.9152	99.151
4	0.3	128	Relu	700	0.9341	101.256
5	0.3	256	Sigmiod	600	0.9298	99.076
6	0.3	512	Tanh	500	0.9054	102.364
7	0.4	128	Sigmoid	600	0.9263	103.456
8	0.4	256	Relu	700	0.9312	100.421
9	0.4	512	Tanh	500	0.9255	98.994

依照相同的方法去執行GRU以及SAES的實驗設計

	Dropout	Batch size	Activate function	Epoch
LEVEL 1	0.2	128	tanh	500
LEVEL 2	0.3	256	Sigmoid	600
LEVEL 3	0.4	512	Relu	700

Experiment	Dropout	Batch size	Activate function	Epoch
1	0.2	128	tanh	500
2	0.2	256	Sigmoid	600
3	0.2	512	Relu	700
4	0.3	128	Relu	700
5	0.3	256	Sigmiod	600
6	0.3	512	Tanh	500
7	0.4	128	Sigmoid	600
8	0.4	256	Relu	700
9	0.4	512	Tanh	500

Experiment	Dropout	Batch size	Activate function	Epoch	R-square	Mse
1	0.2	128	tanh	500	0.9185	106.738
2	0.2	256	Sigmoid	600	0.9214	105.047
3	0.2	512	Relu	700	0.9296	108.269
4	0.3	128	Relu	700	0.9136	104.902
5	0.3	256	Sigmiod	600	0.9363	104.857
6	0.3	512	Tanh	500	0.8917	106.216
7	0.4	128	Sigmoid	600	0.9343	110.956
8	0.4	256	Relu	700	0.9211	105.238
9	0.4	512	Tanh	500	0.9193	112.483

	Dropout	Batch size	Activate function	Epoch
LEVEL 1	0.2	128	tanh	500
LEVEL 2	0.3	256	Sigmoid	600
LEVEL 3	0.4	512	Relu	700

Experiment	Dropout	Batch size	Activate function	Epoch
1	0.2	128	tanh	500
2	0.2	256	Sigmoid	600
3	0.2	512	Relu	700
4	0.3	128	Relu	700
5	0.3	256	Sigmiod	600
6	0.3	512	Tanh	500
7	0.4	128	Sigmoid	600
8	0.4	256	Relu	700
9	0.4	512	Tanh	500

Experiment	Dropout	Batch size	Activate function	Epoch	R-square	Mse
1	0.2	128	tanh	500	0.9137	102.46
2	0.2	256	Sigmoid	600	0.9402	100.128
3	0.2	512	Relu	700	0.921	103.574
4	0.3	128	Relu	700	0.8954	96.481
5	0.3	256	Sigmiod	600	0.9264	99.962
6	0.3	512	Tanh	500	0.9357	101.548
7	0.4	128	Sigmoid	600	0.9394	97.146
8	0.4	256	Relu	700	0.9422	95.088
9	0.4	512	Tanh	500	0.9069	98.452

四、結果

1. 最佳配置及結果

本結果的部分可以看到三種不同的模型最佳配置如下

	Dropout	Batch size	Activate function	Epoch
LSTM	0.2	256	Sigmoid	600
GRU	0.3	256	Sigmiod	600
SAES	0.4	256	Relu	700

另外我們可以發現三種模型跑出來的結果都相差不大，不管是MAEMSE這些數據，又或是模型解釋力R-Square值，可以說是表現都差不多，通常r-square值達到0.8以上我們就可以說這個模型的解釋例是足夠的，那這次跑出來的結果大約落在0.930.94，表示這是三個都是成功的預測模型，真的要比的話SAES堆疊自編碼器會好一些

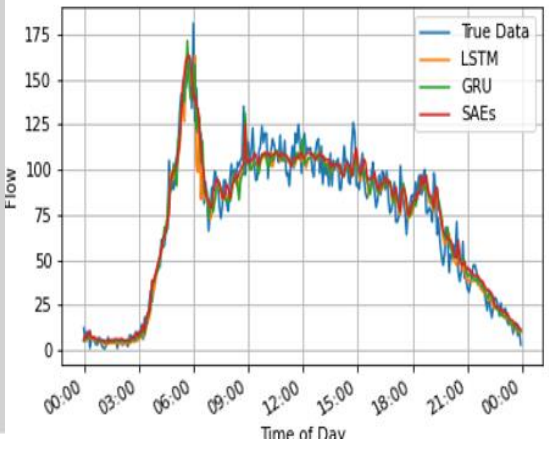
Metrics	MAE	MSE	RMSE	MAPE	R-squre	Explaind variance score
LSTM	7.17	98.92	9.94	16.91%	0.9399	0.94
GRU	7.31	104.85	10.24	18.05%	0.9363	0.9363
SAEs	7.14	95.08	9.75	20.26%	0.9422	0.9439

。

再來是執行時間的部分，我有特別去計算執行時間來驗證說前面所提到的GRU所花的執行時間的確會比另外兩個模型還少，我們可以看到SAES執行時間花的是最多的。另外我也有將預測結果圖表化，可以看到三種模型的預測結果是差不多的。

```
LSTM
execution time: 1.3946423530578613
explained_variance_score:0.940020
mape:16.913820%
mae:7.172698
mse:98.929231
rmse:9.946317
r2:0.939904
GRU
execution time: 0.8498928546905518
explained_variance_score:0.936315
mape:18.054205%
mae:7.319282
mse:104.857864
rmse:10.240013
r2:0.936303
SAEs
execution time: 1.4183828830718994
explained_variance_score:0.943919
mape:20.260020%
mae:7.147675
mse:95.088501
rmse:9.751333
r2:0.942237
```

```
end = time.time()
sec = end - start
print(name)
print('execution time:',sec)
eva_regress(y_test, predicted)
```



五、結論及貢獻

貢獻的部分我認為精準的交通流量預測對於交通系統是很重要的，那這次實驗結果也展示結果是不錯的，另外也證明GRU這個RNN變體的方法可以得到與LSTM預測模型差不多的結果，但是可以省下一些時間成本。

六、反思

雖然只有單獨使用交通流作為輸入變量可以產生可接受的交通流預測性能，但是如果多加一個速度這個變量一起用輸入也許會比單獨使用交通流產生更好的結果。再來就是我看到許多論文都在推崇LSTM作為交通流量預測的模型，是因為他們認為LSTM比SAES預測來的準，但這次做完project後得到的結果SAES也不比其他兩個方法差，那我自己是推測說以前可能沒有去作參數優化以及加入dropout這個防止過度擬和的技術，另外本次project所加的網路層數也比較多，也有加入dropout機制所以可以達到差不多的效果。

七、參考資料

- https://ieeexplore.ieee.org/abstract/document/8317872
- https://github.com/xiaochus/TrafficFlowPrediction
- https://ieeexplore.ieee.org/document/6894591
- https://www.twblogs.net/a/5b8e34a32b717718834380e5
- https://ir.nctu.edu.tw/handle/11536/45372?locale=zh_TW
- https://kknews.cc/zh-tw/code/rnja46v.html