國立高雄科技大學

資訊管理系(所)

碩士論文

雙向注意力機制之深度學習模型應用於石油期貨價格之預測

Oil Futures Prices Prediction Using Dual Attention Deep Learning Models

研究生: 葉新群

指導教授： 黃文楨 博 士

中華民國 109 年 11 月

雙向注意力機制之深度學習模型應用於石油期貨價格之預測

Oil Futures Prices Prediction Using Dual Attention Deep Learning Models

研究生：葉新群

指導教授：黃文楨

國立高雄科技大學

資訊管理系(所)

碩士論文

A Thesis
Presented to
Department of Mechanical Engineering
National Kaohsiung University of Science and Technology
in Partial Fulfillment of the Requirements
for the Degree of
Master of Engineering
in
Mechanical Engineering

Sep 2020
Kaohsiung, Taiwan, Republic of China

National Kaohsiung University of Applied Sciences is the predecessor of
National Kaohsiung University
of Science and Technology (renamed on Feb. 1, 2018)

中華民國 109 年 11 月

國立高雄科技大學（第一校區）研究所學位論文考試審定書

資訊管理系碩士班[第一] 碩士班

研究生 葉新群 所提之論文

論文名稱(中文): 雙向注意力機制之深度學習模型應用於石油期貨價格之預測

論文名稱(英/日/德文): Oil Futures Prices Prediction Using Dual Attention Deep Learning Models

經本委員會評審，符合 碩士 學位論文標準。

學位考試委員會

召 集 人 簽章

委 員

指導教授 簽章

系所主管 簽章

中華民國 109 年 11 月 25 日

保存期限：永久

中文摘要

因石油期貨價格具有非線性且複雜的特性，容易遭到外在因素所影響，準確說想要預測石油期貨價格是深度學習時間序列中非常具有挑戰性的，而現有的類神經網路缺少導入相關特徵的研究，本論文目的在於使用神經網路來預測石油期貨價格與導入黃金價格作為特徵。

資料來源取自 Yahoo 財經提供 2000/02/28 至 2020/06/29 共 5146 筆，並將數據透過深度學習預測石油期貨使用 RMSE 或 MSE 評估其表現，便可找出最優秀的類神經網路模型。論文內使用 CNNBilstm、CNNBiGRU 模型和增加注意力機制的 Attention CNNBilstm、Attention CNNBiGRU、DualAttention CNNBiGRU、DualAttention CNNBiLSTM 與 TPA-LSTM、Prophet、ARIMA 進行實驗找出最佳模型，並探索對於滑動視窗的回看天數處理二維時間序列資料與特徵天數延後影響預測實驗。

實驗結果顯示回看天數的增加對於預測將導致精準度下降。使用二維度時間序列的方式，導入新模型融合注意力機制，從實驗中的資料集區段 80 訓練 20 測試項目中帶有雙重注意力機制的 CNNBiGRU，可以提升預測的精準度，如模型 BiGRU 的 2.79 透過新增雙重注意力機制可改善至 1.85 RMSE，另外發現回歸模型對於預測數據若波動較大，會促使 RMSE 與 MAE 上升尤其在 96%訓練 4%測試的實驗項目上最為明顯。

關鍵字： DualAttention、CNNBiLSTM、CNNBiGRU、RMSE、類神經網路、滑動視窗、Prophet、TPA-LSTM、ARIMA

Abstract

Oil futures prices are non-linear and complex, and easily affected by external factors. To be correct , This is a challenging to study time series in depth learning to predict series prices. The existing neural network introduces related characteristics research.

The study aims to predict the Oil Futrues price with the symbol of golden price. Refer to the 5146 rows from 2nd Fab, 2000 to 29th Jun, 2020 as dataset. Through using the deep learning to predict the Oil Futrues and RMSE or MSE to estimate it, you can see the best Neural Network model. For example, uses TPA-LSTM, ARIMA, Prophet CNNBilstm, CNNBiGRU models and Attention CNNBilstm, Attention CNNBiGRU, DualAttention CNNBiGRU, DualAttention CNNBiLSTM that add the attention fuction to conduct experiments and explore the number of days to look back for the sliding window to process the two-dimensional time series data and feature days delayed influence prediction experiment.

In the end show that the increase the number of look back will lead to a decrease in the accuracy of the prediction. Then a two-dimensional time series, the introduction of a new model fusion attention fuction to create a CNNBiGRU with a dual attention mechanism can improve the accuracy of prediction. from the data set segment in the experiment 80% training, 20% test BiGRU 2.79 to DualAttention CNNBiGRU 1.85rmse .in addition, it is found that the regression model will increase the RMSE and MAE if the prediction data fluctuates greatly, especially in the experimental project of 96% training and 4% testing..

Keyword： DualAttention,CNNBiLSTM,CNNBiGRU,RMSE,Neural Network, sliding window.

目錄

圖目錄

表目錄

壹、緒論

一、研究背景與動機

科技發達與進步，預測成為電腦科學挑戰之一，其以股票、天氣、石油產量、流量紀錄檢測等為大宗，若只是依靠傳統方式進行預測檢測，時常造成數據與真實情況相差甚遠。現在的電腦效能提升、儲存裝置容量擴大價格下降，使得儲存資料的成本下降人們逐漸了解到資料價值，使得過往礙於成本以及設備考量無法普及實作的演算法，現在都一一遭到重視。如沉寂已久的類神經網路逐漸在近幾年成為焦點之一，在這幾年可以使用價格較為便宜的雲端伺服器作為運算，並帶動類神經網路在產業內逐漸熱門了起來，使得眾多歷史資料如股票價格、銷售數量、氣候、車流量等問題，漸漸成為主要處理項目。為了能提前準備未知的結果而提出多種方案作為預備，例如：調整車道、電力調配等，以往改善只能使用經驗法則制定的方法，而如何標準量化制定卻只有會計方法、線性回歸等方式，這些能處理時間序列等問題的方法，對於現今而言已過時將被淘汰。

面對社會的需求，科技進步類神經網路的發展在人臉辨識、圖形辨識以及其它相關應用逐漸成熟外，對於時間序列相關研究也更加重視，但相比之下，它的使用率卻沒有其餘類神經網路來的著重，這也導致類神經網路成為最困難的挑戰。嘗試使用電腦科學來處理巨量資料所帶來的訊息，人們希望電腦可藉由大量且重復的學習，學會了解數據趨勢以及定期所發生之特徵如 Google AlphaGo 學會下棋並運用自如，但在解決未來預測問題上，現今仍然是待解決問題之一，也因此衍生出了監督式學習(Supervised Learning)、非監督式學習(Unsupervised Learing)、強化學習(Reinforcement Learning)等。其中現今多使用並也是主要熱門項目之一的監督式學習，在金融、半導體、服務業、零售業等被廣泛運用，但以往類神經網路除了根據模型深度增加外，並無太多方式。直到近 5 年內 FaceBook 提出 Prophet，將複雜的神經網路透過監督式學習，創新讓模型能短期快速學習根據時

間、月、周、季的方式有效學習趨勢並預測短期，但在於長期預測仍與 VAR 等模型相同，都有遇到長期記憶遺失，無法學習的問題。

二、研究目的

透過使用監督式學習是否能解決存在已久的預測問題，改善類神經網路精準度不足的問題並融入新型機制與金融相關特徵，將本研究所使用的資料集「石油期貨」透過新增的功能、資料集特徵是否能提升精準度，並應用於長期預測，將使用石油期貨與黃金價格兩者單位皆為(美元)，用以石油期貨價格為預測目標黃金價格將設定為特徵，透過滑動式窗處理資料集的前處理，探索 1~20 天的回看天數是否將對神經網路學習成為阻礙又或者是提升精準度。

然而這 3 年內 Google 提出透過 Encode Decode Attention 方式，結果比過往類神經網路學習能力更為優秀，本研究將加入 Attention 機制，透過加權方式與 Google 做法不同，改善類神經網路長短期記憶(LSTM)、GRU 的模型。

以下提出幾點目的：

1. 使用監督式學習，能否解決改善存在已久的預測精準度問題?
2. 二維資料延遲輸入是否會影響模型?
3. 使用二維並帶有注意力機制之模型的精準度是否將有效改善?
4. 滑動視窗天數是否會影響學習?

三、研究貢獻與重要性

研究透過 TPA-LSTM、Prophet、ARIMA 進行模組測試，將記錄執行時間，以及預測精準度用 RMSE 做計算並提出 CNN BiLSTM Attention、CNN BiGRU Attention、CNN BiGRU Dual Attention CNN BiLSTM Dual Attention 模型測試，使用 RMSE、MAE 做計算準確率。

1. RMSE 公式: RMSE=$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}$ (1)

   y 是測試資料集，i 為當前位置，使用真實數值減去預測值接著進行平方求和最後開根號獲得 RMSE。

   MAE 公式: MAE=$\frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i-y_i|$ (2)

   透過$\hat{y}_i$為預測值減去y_i實際值取絕對值後加總所得為 MAE

2. Attention 機制 ：將上層所傳遞進行加權分配最終與上層相乘

本研究貢獻如下：

1. 使用 CNN_BiLSTM、CNN_BiGRU 模型在 90%訓練 10%驗證獲得 2.17 RMSE。
2. 導入二維資料針對黃金價格天數延遲後，提升幅度最大的從 3.33 至 3.07 RMSE。
3. 使用二維並帶有注意力機制之模型的精準度從 2.79 至 1.85 RMSE。
4. 滑動視窗天數影響學習預測精準度隨者天數越大則精準度下降。

CNN BiGRU 加入注意力機制後模型架構圖如下:

圖 1 Attention CNNBiGRU

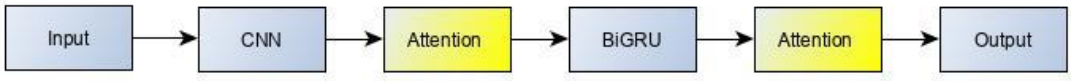

圖 2 Dual Attention with CNNBiGRU

圖 1、圖 2 展示了 CNN BiGRU/CNN BiLSTM 加入兩種注意力機制方式，圖 1 僅使用一組注意力機制位於後方增強最後全輸出，圖 2 則是將注意力機制擺放於 CNN 後，提升 CNN 擷取的短期特徵再次增強，透過雙向 GRU 訓練後再次傳

遞給注意力機制並輸出，後續兩者方式將與 BiGRU/BiLSTM 進行精準度、回看天數、特徵延遲的實驗尋找最佳神經網路模型。

四、研究範圍

使用 Yahoo 所提供資料集其時間序列長度為 2000/02/28 至 2020/06/29 共 5146 筆的石油期貨與黃金價格對於雜訊不做處理，增加模型對於震盪時的反應、學習，應用於 TPA LSTM 與本研究所提出四種神經網路模型做測試。

貳、文獻探討

本章的第一節將簡述回歸模型和這幾年的文獻資料，第二節會介紹改良回歸模型方法技術，第三節會深入探討關於眾多神經網路的方法，第四節則是探討二維資料黃金與石油關聯和相關文獻。

一、回歸模型

預測時間序列數據是經濟學、商業和金融學中的重要主題，線性單變量時間序列預測的最著名模型是自回歸綜合移動平均線（ARIMA），其他則為其他自回歸時間序列模型、自回歸-回歸（AR）、移動平均線（MA）和自回歸移動平均線（ARMA）。Kazem, A., Sharifi, E., Hussain, F. K., Saberi, M., & Hussain, O. K. ,2013 提出新的方向使用線性向量回歸（SVR）最佳化的 SVR 用於預測股市價格，SVR 對引數選擇比較敏感，源自原始的 SVM 擅長處理二分類問題，亦可視為典型的回歸問題，但是這些模型大多僅限線性方式處理。

以下整理出傳統模型預測相關論文:

表格 1　傳統模型預測相關論文

Zhang, G. P. (2003)	自回歸綜合移動平均線(ARIMA)
K. Duan, S. Keerthi, A.N. Poo (2003)	支撐向量機(SVM)
Kazem, A., Sharifi, E., Hussain, F. K., Saberi, M., & Hussain, O. K. (2013).	線性向量回歸(SVR)

Sun, C. S., Wang, Y. N., & Li, X. R. (2008)	向量自回歸(VAR)
Ling, S., & McAleer, M. (2003).	GARCH 模型

二、其它模型

Boyacioglu, M. A., & Avci, D. (2010).提出新的 ANFIS 方法來嘗試預測股票市場價格，並成為之後多數人使用的替代方法之一。根據文獻內的測試結果可知，雖然數據結果相當優秀，但其仍依舊捨棄了許多可以作為數據特徵的相關資料，若加入相關特徵，將有可能影響預測準確度。

三、神經網路

Jia, H. (2016)、Roondiwala, M., Patel, H., & Varma, S. (2017).、Hiransha, M., Gopalakrishnan, E. A., Menon, V. K., & Soman, K. P. (2018)、De Gooijer, Jan G., and Rob J. Hyndman. (2006)提出類神經網路對於股票價格時間序列屬性的預測，可以達到良好的預測效果且優於回歸與其他模型性能表現，採用標準化數據並導入遺傳基因演算法(GA)去調整參數，其結果將獲得高準確率，但測試中捨棄相關特徵且為單一模型，使其在面對不同的資料時，可能無法有效處理。其中衍生 Gilliland, M., Tashman, L., & Sglavo, U. (2016)學者提及神經網路是如何應用在商業預測上，並在實際案例中解決問題，促使神經網路引起大眾關注。接著 Taylor, S. J., & Letham, B. (2018).等學者提出了 Prophet 這種大規模預測的演算法，結合週、月、季等關鍵循環做為週期，在快速學習能力強的神經網路下，實際運用在 Facebook 廣告流量分析預測等。Saldivar, Frank, and Mauricio Ortiz(2019)等學者提出若將 LSTM 和 Prophet 兩者用來預測股價並比較，可以發現 LSTM 的性能明顯優於其

他回歸技術，結果也相對優異。由於 Prophet 受限於本身演算法關係，預測長時間會如回歸模型般預測逐漸失準。

Sagheer, A., & Kotb, M. (2019)學者使用另一種深度學習方法，目的在於解決傳統預測的局限性，屬於深層的長期短期記憶（DLSTM）體系結構，是對遞歸神經網路的擴展增加深度，用以加深網路的學習，結果對於新型的 GRU 變種等，反而效果與單層並無太大區別。

對照 Selvin, S., Vinayakumar, R., Gopalakrishnan, E. A., Menon, V. K., & Soman, K. P. (2017)學者文內使用三種不同的深度學習架構，針對其性能進行比較評估，將滑動視窗方法用於預測未來價值，從結果可以明顯看出，CNN 架構能夠識別趨勢變化，故 CNN 被認為是最佳模型。CNN 模型多使用特定時刻的信息來進行預測，隨著數據的複雜化，該模型仍然有調整的空間，如增加它種神經網路等。

Althelaya, K. A., El-Alfy, E. S. M., & Mohammed, S. (2018)學者將 BLSTM 和 SLSTM 模型的性能與淺層神經網路及單向 LSTM 進行了比較，結果表明 BLSTM 和堆疊 LSTM 網路在預測短期價格方面，均具有更好的性能，故可得知深度學習方法優於淺層神經網路。

除了由傳統神經網路深度發展外，也有部分學者依舊試圖開發新型模型，結合它種模型以提高學習準確率。Kim, H. Y., & Won, C. H. (2018)學者創建了一個模型，該模型將以下三個模型 GARCH，EGARCH 和 EWMA 與 LSTM 組合在一起，並將此模型稱為 GEW-LSTM，提供了我們混合神經網路的想法以及實踐。Pang, X., Zhou, Y., Wang, P., Lin, W., & Chang, V. (2018)學者提及儘管這些模型可以在一定程度上改善預測效果，但歷史數據特徵輸入仍存在一些不足，如股票市場中的文字訊息（例如新聞）。Atsalakis, G. S., & Valavanis, K. P. (2009)使用 LSTM

與 ARIMA 兩者做相比，LSTM 所獲得的平均錯誤率降低了 84-87%，這表明 LSTM 優於 ARIMA 且表現出真正的隨機行為，但卻只有將數據本身導入，並沒有融入相關資料，如(VAR)回歸模型特徵之資料。Sachdeva, A., Jethwani, G., Manjunath, C., Balamurugan, M., & Krishna, A. V. (2019)學者則結合統計方式，對交易量加權平均價格（vwaps）導入 LSTM，而不是使用收盤價，並開發了一種簡化的交易策略，提供一種新用法，但仍然受限於模型性能所限制。Nelson, D. M., Pereira, A. C., & de Oliveira, R. A. (2017)學者提出新想法，將成熟的 LSTM 建構後，使用股票價格做序列，並改變過往不在僅放入收盤價，而是加入技術指標，藉此改善預測問題。結果顯示，在預測特定股票的價格在近期內是否會上漲時，平均準確率達到 55.9%。想獲得更好預測結果的關鍵在於如何改善輸入，或修改模型的新方向，Xingjian, S. H. I., Chen, Z., Wang, H., Yeung, D. Y., Wong, W. K., & Woo, W. C. (2015)學者使用類神經網路 ConvLSTM 進行降雨預報訓練，將 LSTM 導入 CNN 圖形辨識中並且結合 LSTM 的優點，Shih et al (2019)學者們提出選擇將資料切成片段，並透過神經網路取得當前片段與另一片段計算變量其透過注意力選擇變量最大者進行學習，與本研究所提出之模型不同之處在於研究模型將使用 Softmax 函數計算當前時間步長與特徵去計算權重而非擷取片段。

下方表格將整理近期神經網路預測相關論文：

表格 2　相關論文

模型的架構	模型概述
Boyacioglu et al.(2010)	提出新方法適應性類神經模糊推論，常被應用在分類和資料分析裡。
Kazem et al.(2013).	使用線性向量回歸（SVR）最佳化的 SVR 用於預測股市價格改善原始的 SVM 只適合二分類的問題。

Xingjian et al.(2015)	使用類神經網路 ConvLSTM 進行降雨預報訓練，將 LSTM 導入 CNN 圖形辨識中並且結合 LSTM 的優點。
Vaswani et al.(2017)	提出注意力機制，提升神經網路專注。
Taylor et al.(2018)	提出新的方法 Prophet 應用於自家 Facebook 流量預測調整廣告費用等。
Kim et al.(2018)	該模型將所有三個模型 GARCH，EGARCH 和 EWMA 與 LSTM 組合在一起，並將此模型稱為 GEW-LSTM
Althelaya et al.(2018)	BLSTM 和堆疊 LSTM 網路在預測短期價格方面均具有更好的性能，深度學習方法優於淺層神經網路。
Shih et al (2019)	提出擷取片段注意並傳遞給 LSTM 此模型將得到有效率的學習並取得高精準度。

四、石油與黃金之關係

國際原油價格經歷了劇烈的波動，國際原油價格持續上漲達到歷史新高，但隨後遭遇新型冠狀病毒後，導致整體需求降低，全球因疫情影響各產業停擺，總損失難以估計，人們也因此降低旅遊等需求。石油期貨一度經歷「水比油貴」到「買油倒給錢」的狀況，美國油價經歷了歷史性暴跌。由於原油是經濟發展一個重大指標，高波動的原油價格給經濟發展帶來了巨大衝擊，特別是對那些原油進口國。原油定價機制已成為學術界的熱門話題，而在 Narayan, Paresh Kumar, Seema Narayan, and Xinwei Zheng. (2010)學者觀察分析了黃金對通貨膨脹的關聯，

發現到其實黃金與石油是可以互相預測的，兩者會互相影響，油價上漲導致通貨膨脹率上升，這也意味著金價上漲。

Chkili, Walid. (2015)學者使用 GARCH（FIGARCH）模型來分析黃金和原油市場之間的動態關係實驗證明兩個市場之間的動態關聯會隨者時間變化。Ftiti, Z., Fatnassi, I., & Tiwari, A. K. (2016)學者文獻內則將不同市場石油以及黃金做出分析比較，在石油的預防性需求衝擊下，黃金價格反應是滯後於石油價格的波動，但效果是短暫的，僅持續 16 至 24 天，兩個市場之間的共同連動和因果關係更加明顯，先前的研究表明，石油價格會通過多種渠道影響黃金價格。

上面共介紹了眾多模型，接下來我們用表格來整理如下。

表格 3　歷史文獻

模型的架構	模型概述
Narayan et al (2010)	分析了黃金對通貨膨脹的關聯，發現到其實黃金與石油是可以互相預測的，兩者會互相影響。
Chkili et al.(2015)	使用 GARCH（FIGARCH）模型來分析之間的動態關係，實驗證明兩個市場之間的動態關聯會隨著時間產生波動變化。
Ftiti et al (2016)	在石油的預防性需求衝擊下，黃金價格反應是滯後於石油價格的波動，但效果是短暫的，僅持續 16 至 24 天。
Mo, B., Nie, H., & Jiang, Y. (2018)	匯率和石油價格關係成正比，匯率和石油容易遭受到黃金價格的變化影響。
Singhal, S., Choudhary, S., & Biswal, P. C. (2019)	國際金價對墨西哥的股票價格具有影響，而石油價格則產生負面影響且石油價格長期對匯率產生負面影響。

Dutta, A., Das, D., Jana, R. K., & Vo, X. V. (2020)	結果表明在 COVID-19 期間黃金是原油市場的避風港也發現到在此時比特幣僅當作投資者的分散風險投資。

參、研究方法與設計

本研究透過神經網路預測進行實驗，分析二維序列導入多種模型、加入注意力機制、特徵天數延遲、滑動視窗回看天數的影響，是否能解決精準度問題步驟如下:

設計測試方法: 選擇平台、使用多種模型與 TPALSTM、Prophet、ARIMA 進行比較，選定 RMSE、MAE 做基準。

1. 資料集前處理、正規化: 制定資料集，根據實驗產出對應資料。
2. 模型建置: 建立多種模型 BiLSTM/BiGRU，加入注意力機制比較。
3. 研究實驗項目: 訂出五種實驗並檢視其結果。

一、設計測試方法

透過不同模型的融合彌補增強單一模型面對複雜資料集時的問題，將神經網路與 TPA-LSTM 進行測試比較，使用如 Attention CNN BiLSTM、Attention CNN BiGRU、Dual Attention CNN BiLSTM、Dual Attention CNN BiGRU 使用相同標準進行測試，將預測結果使用 RMSE 做出精準度計算，並繪製出圖表，進行比較，找出最優異者神經網路，並且尋找研究資料集前處理回看天數、特徵天數的相關影響，研究建立在 Google colab 平台上下方將列出 Colab 介紹。

Google Colab 全名為 Colaboratory 優點如下:

1. 無須任何設定即可執行操作
2. 免費使用 GPU
3. 支援支援 Python 程式及機器學習 TensorFlow 演算法..等

二、資料集前處理、正規化

在研究中為了獲取資料以及處理龐大的資料集，資料集選用 yahoo 所提供的石油期貨(CL = F)、黃金價格進行處理，並利用其資料集使用在其他類神經網路上，統一將資料進行正規化。

三、模型建置

結合表格內優點將 CNN 與雙向 LSTM/GRU 進行組合設定，CNN 擷取短期特徵，雙向 LSTM/GRU 透過計算將比以往單向獲得更良好學習，並融入注意力機制透過加權相乘的方式增強特徵效果根據模型的不同研究設計模型擁有 1 或 2 的注意力機制。

注意力機制如下圖

h_t = 時間步長

h_s = 特徵

ah_t = 注意力機制後時間步長

ah_s = 注意力機制後特徵

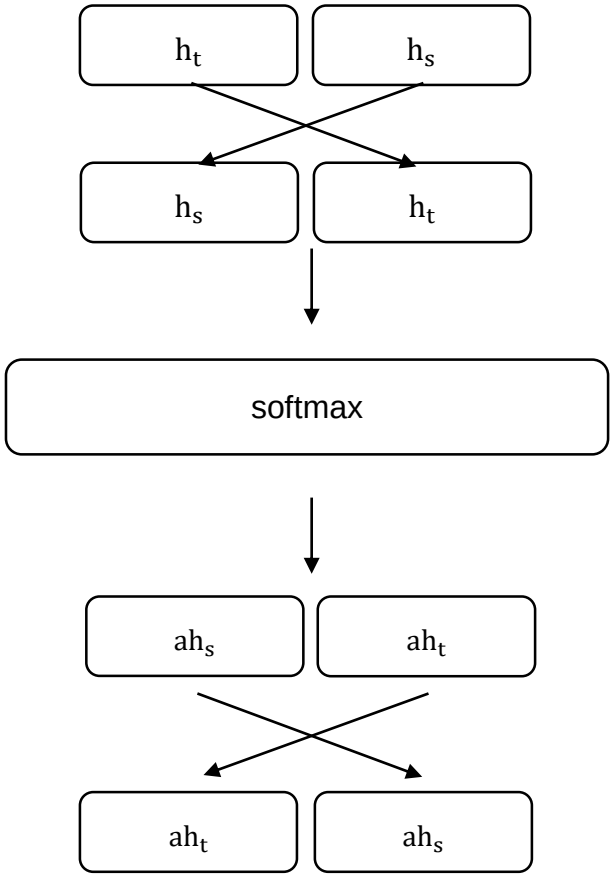

圖 3　注意力機制圖形化

圖 3 透過 CNN 擷取短期特徵神經元，傳遞給下層將當前h_t與h_s進行維度個交換，再透過下層使用 Softmax 函數將輸出數值轉化為相對機率，所有機率的和將等於 1，傳遞給下層，最終矩陣再進行一次維度交換成員使尺寸，結束完此動作後會將原始神經元與經過注意力機制後的神經元進行矩陣逐一相乘，此時將會提升神經元權重，此動作稱之為注意力機制，故能將該注意、學習的部分記住，在後續實驗中嘗試加入雙重注意力機制，一個位於 CNN 之後其次則位於雙向 GRI/LSTM 後，進行測試比較預測精準度。

(一)雙向層(Bidirectional layer)如下:

來自 Schuster, M., & Paliwal, K. K. (1997)，建立反向的神經網路，一種從過去到未來，另外一種從未來到過去，在向後傳遞網路中將保留未來的訊息，並且使用多個隱藏狀態結合，使得此組合能在任何節點上保留過去與未來訊息，此方法廣泛運用於語言學習下方將顯示圖形範例，並加入至後續實驗模型內，雙向模型簡稱為 BiLSTM/BiGRU。

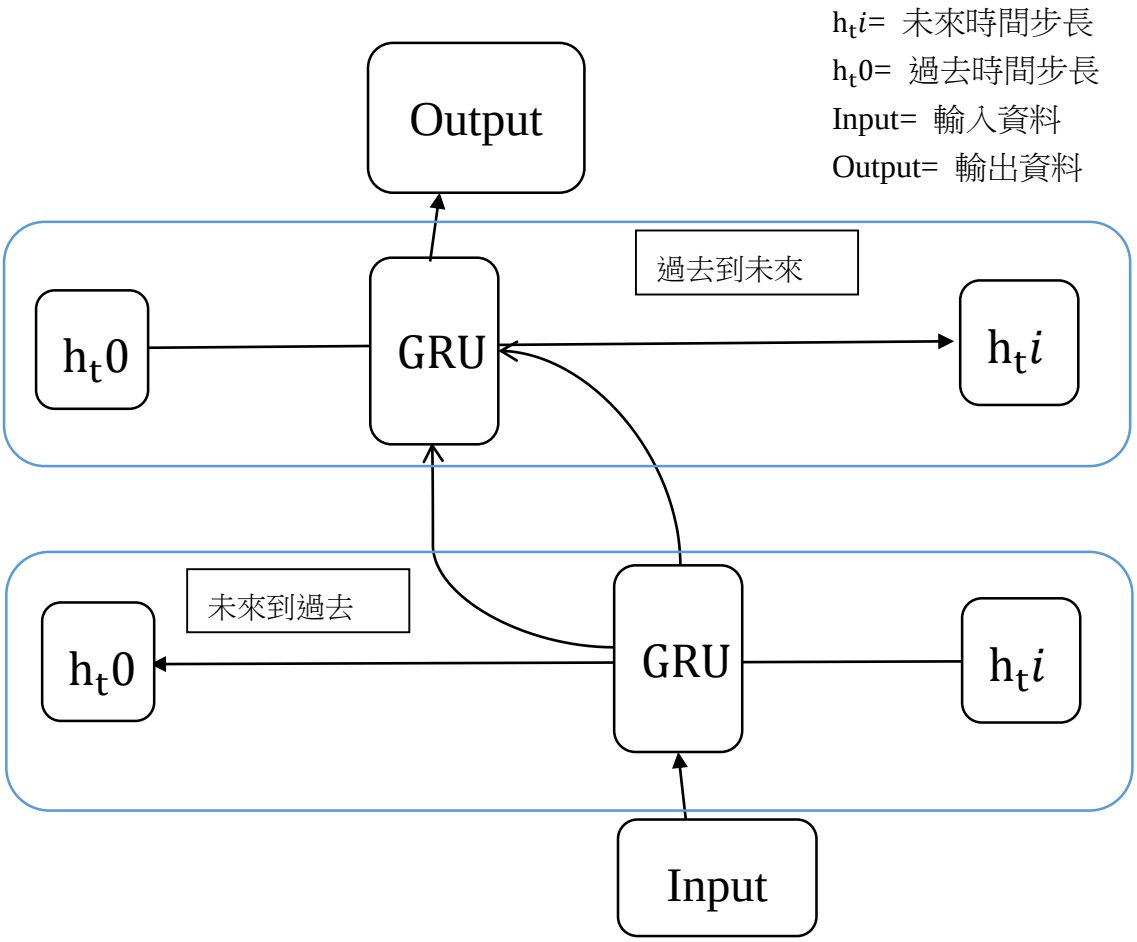

圖 4　雙向 GRU 圖形化

圖 4 介紹了 GRU 層使用雙向機制透過圖形化的顯示，可以清楚明白雙向神經網路的作用，ht_i到ht_0代表反向網路，ht_0到ht_i則是代表正向順序，兩個都連結者一個輸出層者，這種結構組成能夠提供輸出層 Output 過去與未來的完整資訊，故能提高神經網路學習的能力，雙向的機制可以應用在 LSTM、GRU 上，後續實驗將會展示結合雙向的 GRU 與 LSTM 模型。

(二)模型一: 圖內「?」用以代表模型接收第一維度的尺寸

Layer (type)	Output Shape	Param #
bidirectional_6 (Bidirection	(None, 128)	25728
dropout_6 (Dropout)	(None, 128)	0
dense_6 (Dense)	(None, 1)	129

Total params: 25,857
Trainable params: 25,857
Non-trainable params: 0

圖 5　雙向 GRU 摘要

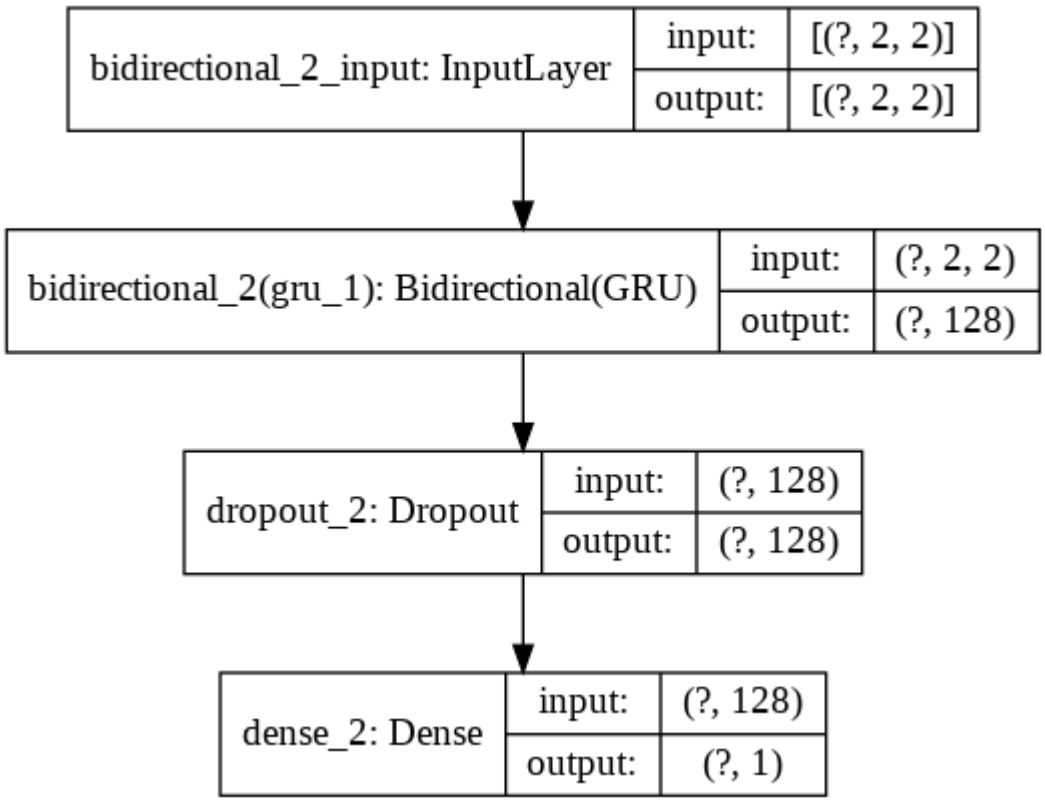

圖 6　雙向 GRU

上圖使用雙向 GRU 共 4 層結構，首先使用 Bidirectional 包裝 GRU 層，使其成為雙向 GRU 簡稱 BiGRU,下層選用 Dropout 層來加速收斂避免耦合，最終則建立輸出層模型步驟如下。

1. 使用 Bidirectional 包裝 GRU 達成雙向 GRU 模型。
2. 增加 Dropout 加速模型收斂。
3. 全輸出層並調用 sigmoid 函數進行訓練，實驗結果調用此函數輕微升精準度。

(三)模型二: 圖內「?」用以代表模型接收第一維度的尺寸

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_10 (Bidirectio (None, 128)               34304
_________________________________________________________________
dropout_10 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 129
=================================================================
Total params: 34,433
Trainable params: 34,433
Non-trainable params: 0
_________________________________________________________________
```

圖 7　雙向 LSTM 摘要

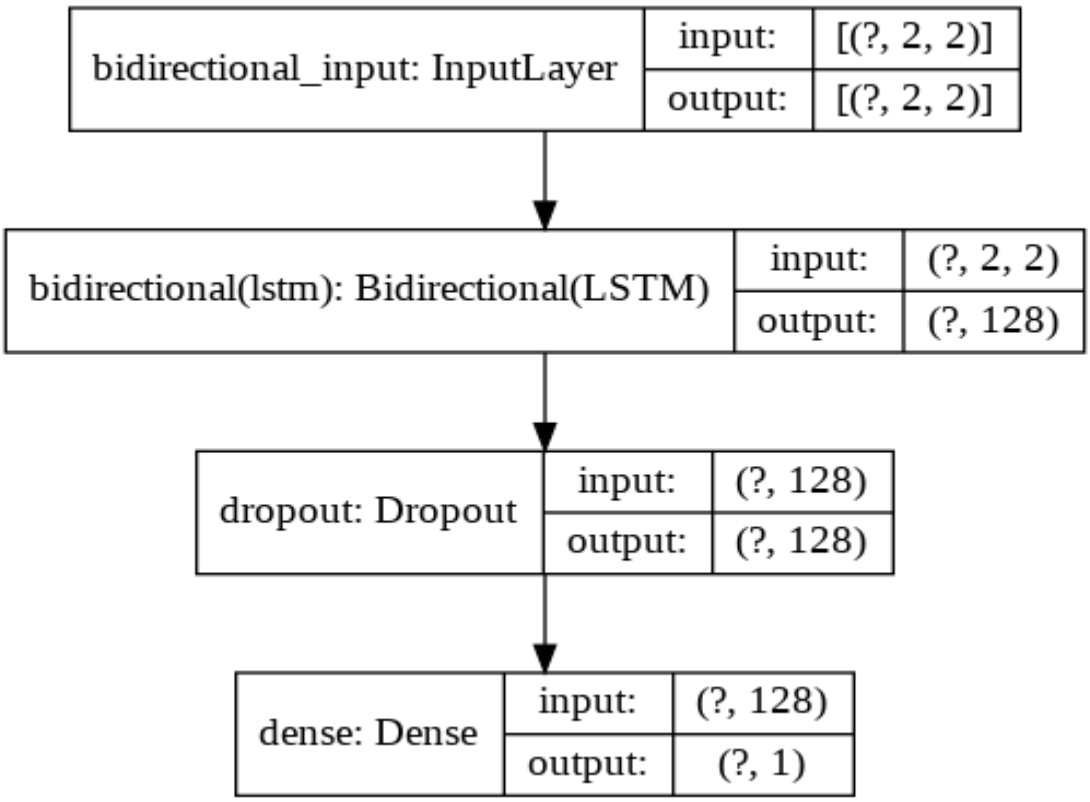

圖 8　雙向 LSTM

上圖使用雙向 LSTM，如圖內所表示，透過 input layer 接收輸入，並傳遞給與雙向 LSTM 進行訓練與反向傳遞最終僅加入一層 Dropout 做加速收斂，並不需使用多餘 Flatten、Batch Normalization 等模型步驟如下。

1. 使用 Bidirectional 包裝 LSTM 達成雙向 LSTM 模型。

2. 增加 Dropout 加速模型收斂。
3. 全輸出層並調用 sigmoid 函數進行訓練，實驗結果調用此函數輕微升精準度。

(四)模型三: 圖內「?」用以代表模型接收第一維度的尺寸

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 2, 2)]	0	
conv1d (Conv1D)	(None, 2, 128)	384	input_1[0][0]
bidirectional (Bidirectional)	(None, 2, 128)	74496	conv1d[0][0]
batch_normalization (BatchNorma	(None, 2, 128)	512	bidirectional[0][0]
dropout (Dropout)	(None, 2, 128)	0	batch_normalization[0][0]
permute (Permute)	(None, 128, 2)	0	dropout[0][0]
dense (Dense)	(None, 128, 2)	6	permute[0][0]
permute_1 (Permute)	(None, 2, 128)	0	dense[0][0]
multiply (Multiply)	(None, 2, 128)	0	dropout[0][0] permute_1[0][0]
batch_normalization_1 (BatchNor	(None, 2, 128)	512	multiply[0][0]
dropout_1 (Dropout)	(None, 2, 128)	0	batch_normalization_1[0][0]
flatten (Flatten)	(None, 256)	0	dropout_1[0][0]
dropout_2 (Dropout)	(None, 256)	0	flatten[0][0]
dense_1 (Dense)	(None, 1)	257	dropout_2[0][0]

Total params: 76,167
Trainable params: 75,655
Non-trainable params: 512

圖 9　Attention with CNNBiLSTM 摘要

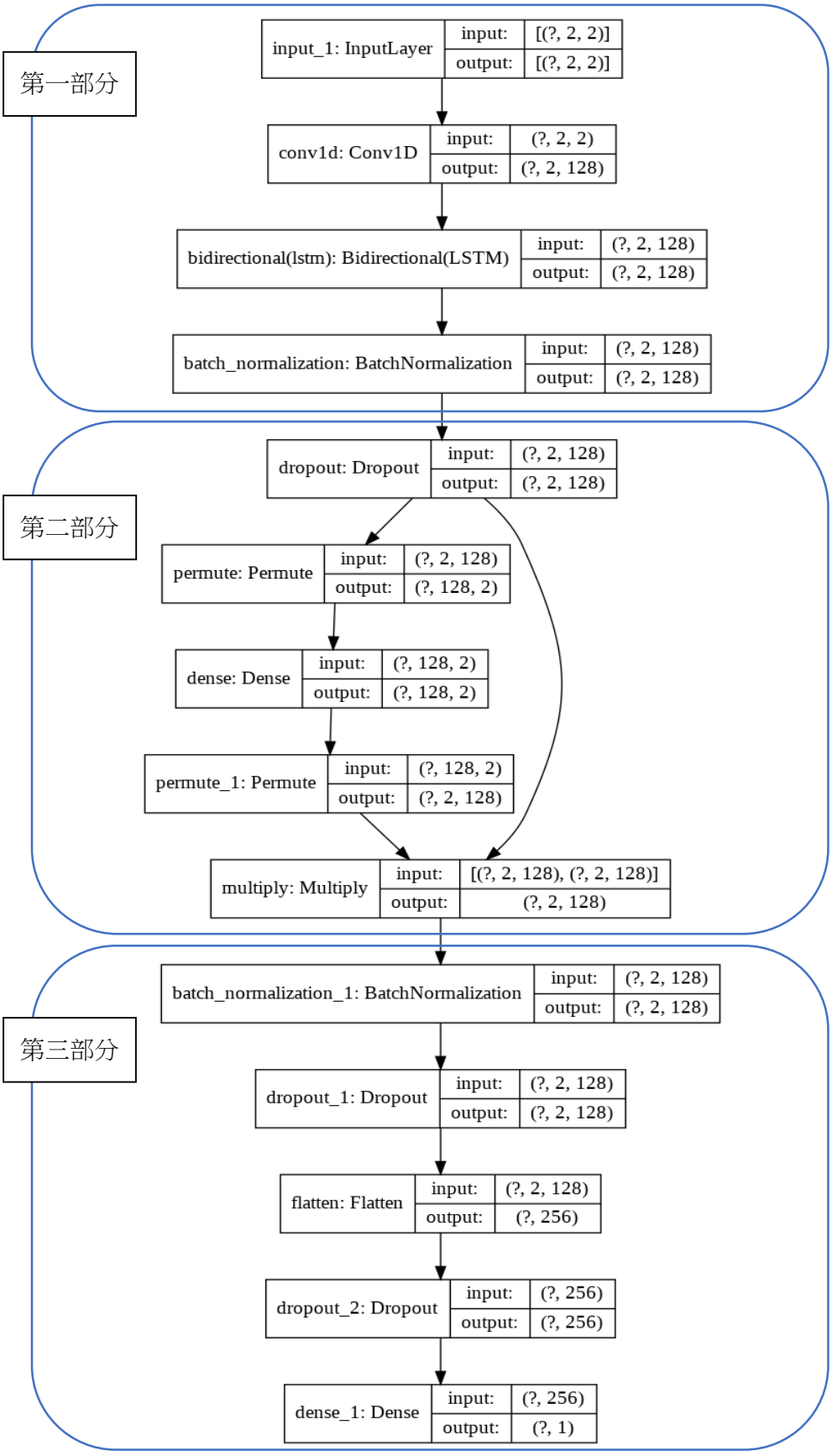

圖 10　Attention with CNNBiLSTM 完整模型圖

將模型分三個部分介紹三個部分如下逐一介紹:

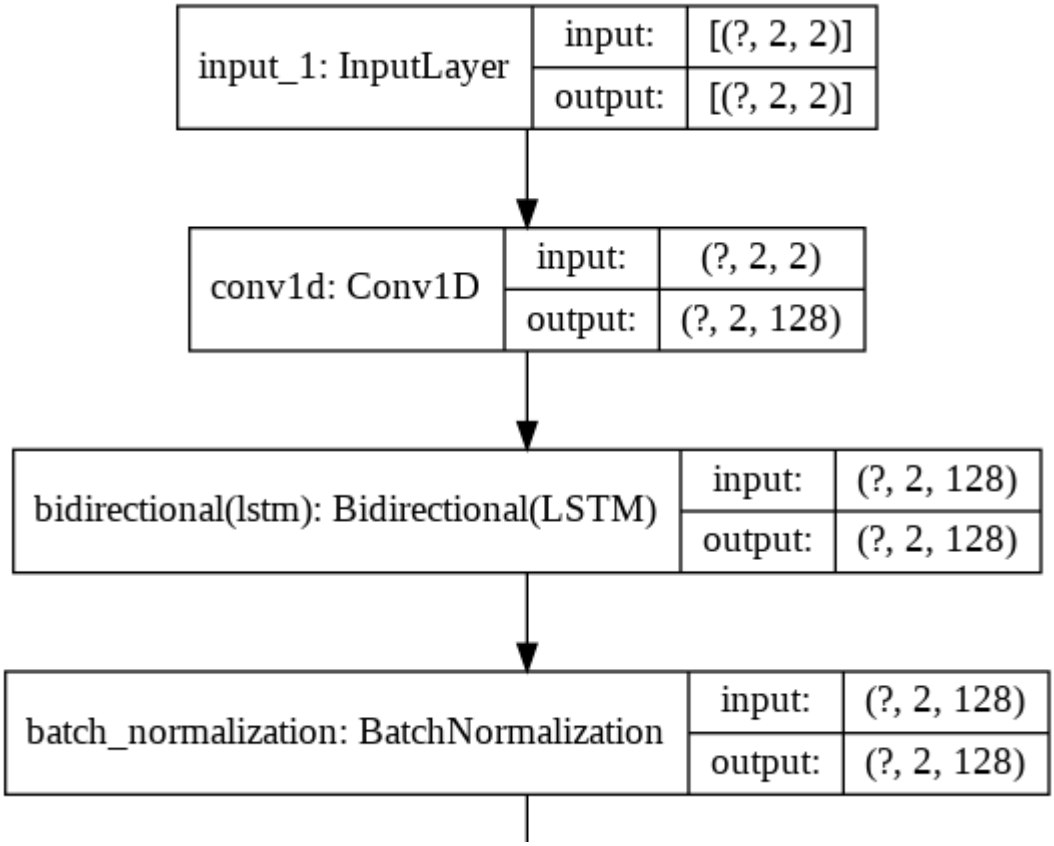

圖 11　Attention CNNBiLSTM 模型圖一

展示 Attention CNNBiLSTM 模型圖式第一部分，第一層建立全連接層接受完整的資料，透過 Conv1D(CNN)獲取短期特徵的神經元後傳遞給下層的雙向 LSTM 進行處理利用雙向特性學習完整訊息的神經元，接者呼叫 Batch 正規化層用以加速訓練防止梯度消失輔助模型收煉。

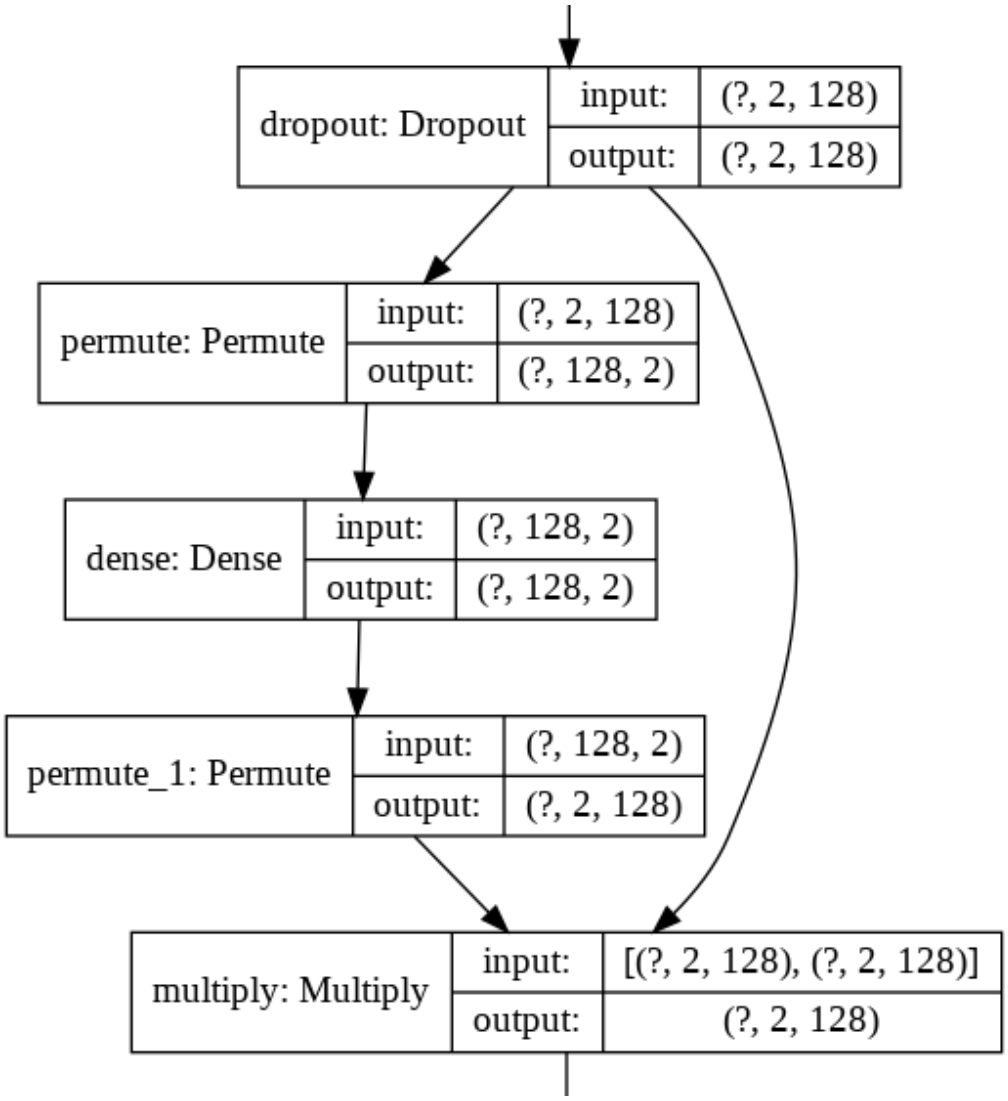

圖 12　Attention with CNNBiLSTM 模型圖二

則是將注意力機制進行建構後，可以從 Permute 層發現到該層進行矩陣維度交換的行為，Dense 層則是呼叫函數進行權重分配，最終呼叫 Multiplu 層來完成分配權重後的神經元與尚未經過注意力機制之神經元進行矩陣相乘，提升神經元權重。

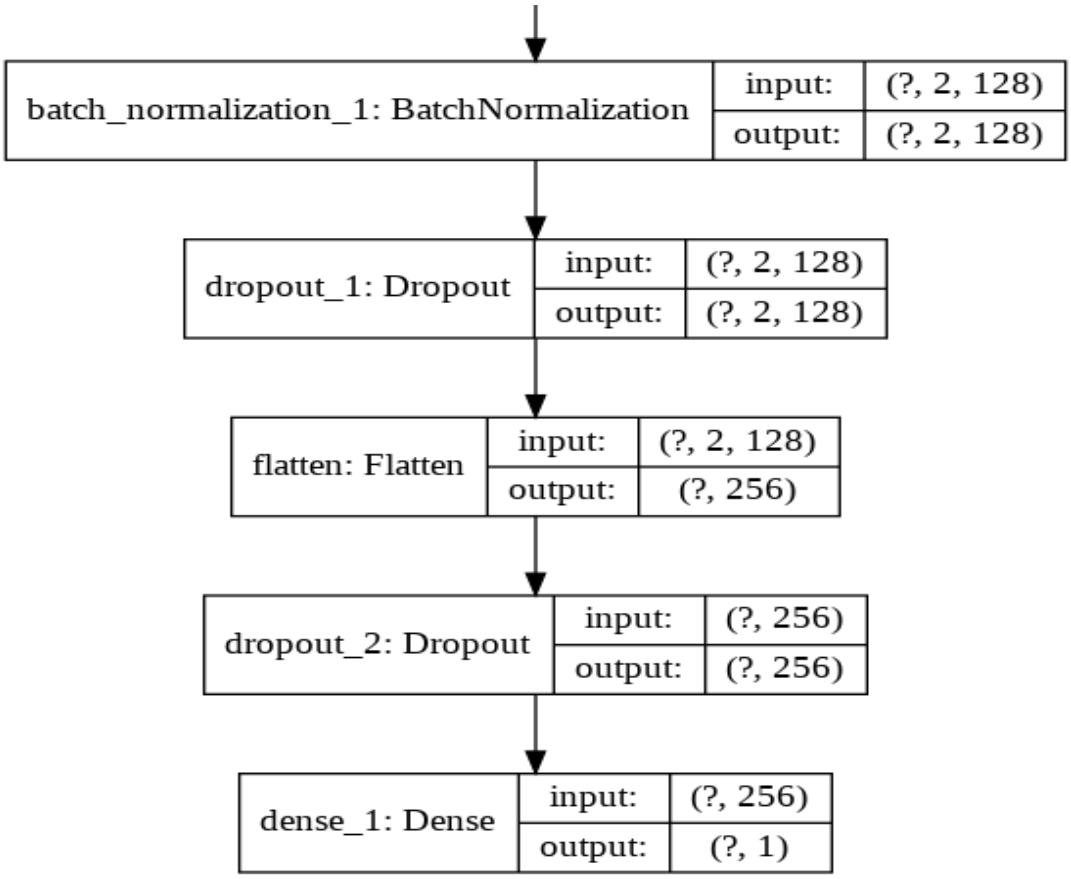

圖 13　Attention with CNNBiLSTM 模型圖三

圖 13 為模型第三部分，使用 Dropout 層與批次正規化層促使度消失與模型無法收斂等問題，接續兩者後使用過濾層，幫助模型進行減少資料維度，

最終將擁有注意的神經元與 BiLSTM 輸出做矩陣相乘，然後增加過濾層促使資料維度下降，模型總結操作如下。

1. 透過 CNN 取得短期特徵。
2. 使用雙向 LSTM 接受 CNN 傳遞的特徵。
3. 增加 Batch Normalization 加速訓練防止梯度消失。
4. 使用注意力機制增強學習。
5. 再次使用 Batch Normalization 。
6. 增加 Flatten 層降低輸出維度。
7. 最終全輸出層，調用 sigmoid 函數可提升精準度。

(五)模型四: 圖內「?」用以代表模型接收第一維度的尺寸

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 2, 2)]	0	
conv1d_2 (Conv1D)	(None, 2, 128)	384	input_3[0][0]
permute_8 (Permute)	(None, 128, 2)	0	conv1d_2[0][0]
dense_6 (Dense)	(None, 128, 2)	6	permute_8[0][0]
permute_9 (Permute)	(None, 2, 128)	0	dense_6[0][0]
multiply_4 (Multiply)	(None, 2, 128)	0	conv1d_2[0][0] permute_9[0][0]
bidirectional_2 (Bidirectional)	(None, 2, 128)	74496	multiply_4[0][0]
batch_normalization_4 (BatchNor	(None, 2, 128)	512	bidirectional_2[0][0]
dropout_6 (Dropout)	(None, 2, 128)	0	batch_normalization_4[0][0]
permute_10 (Permute)	(None, 128, 2)	0	dropout_6[0][0]
dense_7 (Dense)	(None, 128, 2)	6	permute_10[0][0]
permute_11 (Permute)	(None, 2, 128)	0	dense_7[0][0]
multiply_5 (Multiply)	(None, 2, 128)	0	dropout_6[0][0] permute_11[0][0]
batch_normalization_5 (BatchNor	(None, 2, 128)	512	multiply_5[0][0]
dropout_7 (Dropout)	(None, 2, 128)	0	batch_normalization_5[0][0]
flatten_2 (Flatten)	(None, 256)	0	dropout_7[0][0]
dropout_8 (Dropout)	(None, 256)	0	flatten_2[0][0]
dense_8 (Dense)	(None, 1)	257	dropout_8[0][0]

Total params: 76,173
Trainable params: 75,661
Non-trainable params: 512

圖 14　Dual Attention with CNNBiGRU 摘要

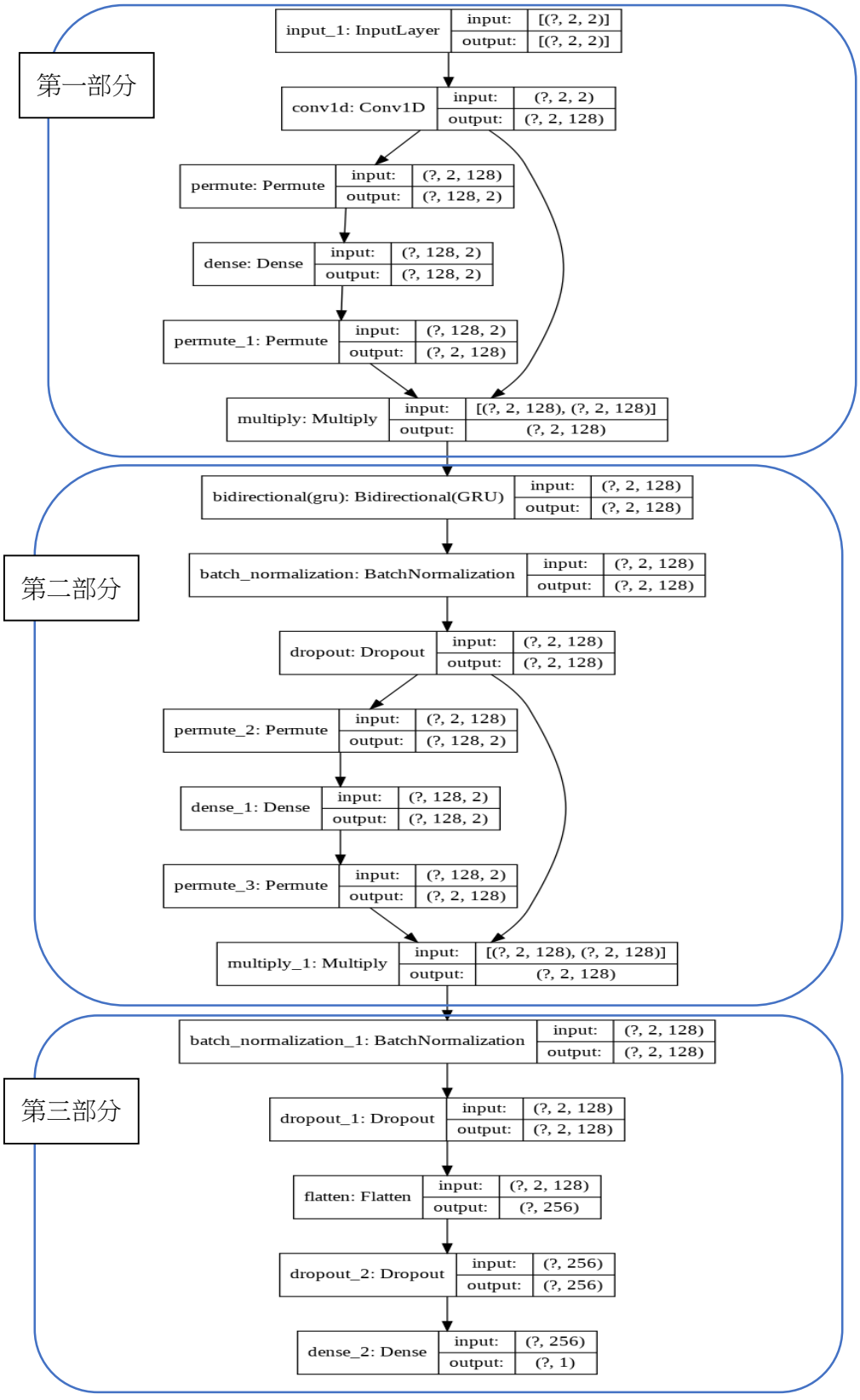

圖 15　Dual Attention CNNBiGRU 完整模型圖

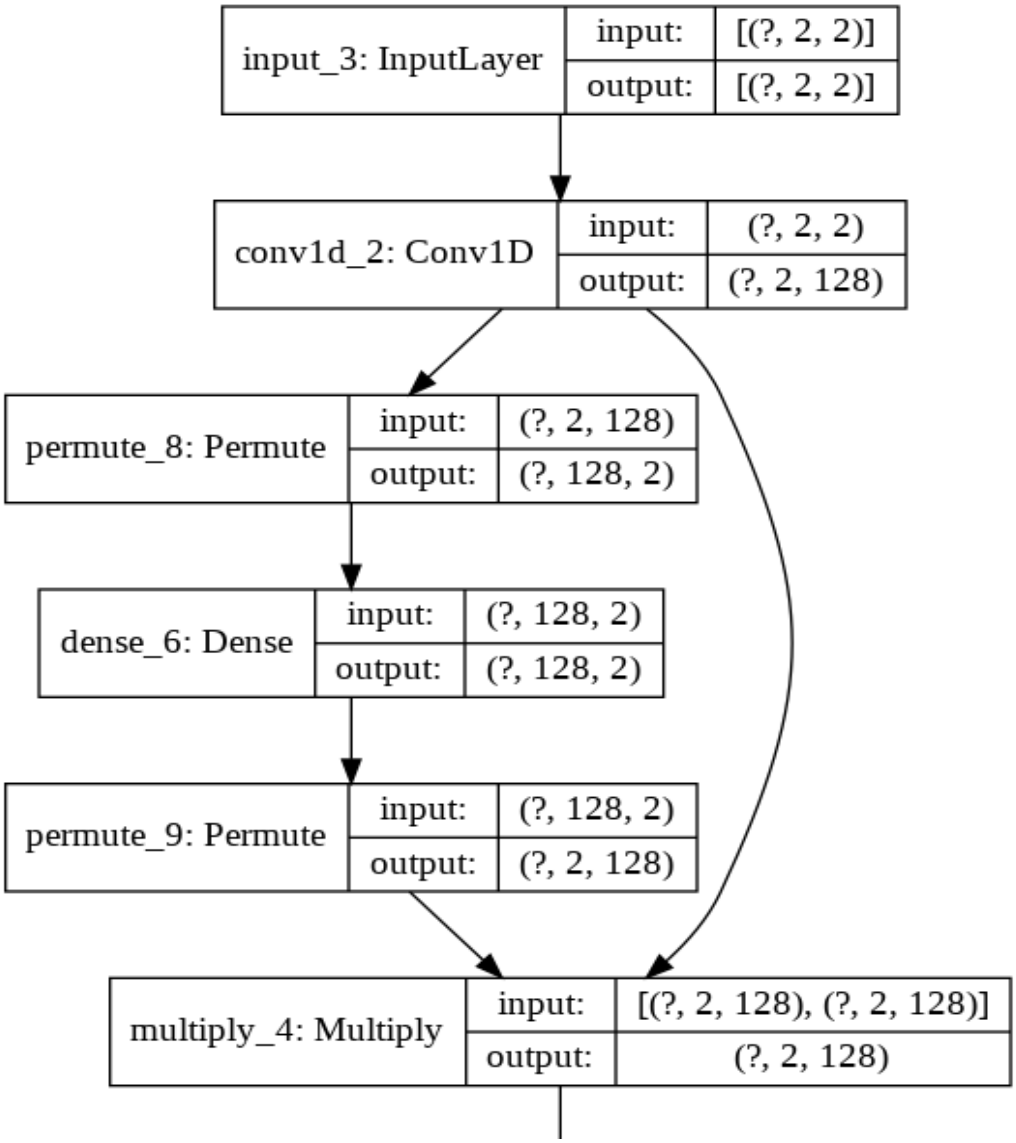

圖 16　Dual attention CNNBiGRU 第一部分

上圖 16 展示 Dual attention CNNBiGRU 第一部分，透過 CNN 擷取短期特徵後，傳入注意力機制進行短期特徵再次增強，透過反轉矩陣並在 Dense 層調用 Softmax 函數分配權重，最終 Multiplu 層進行矩陣相乘達成增強特徵的目的。

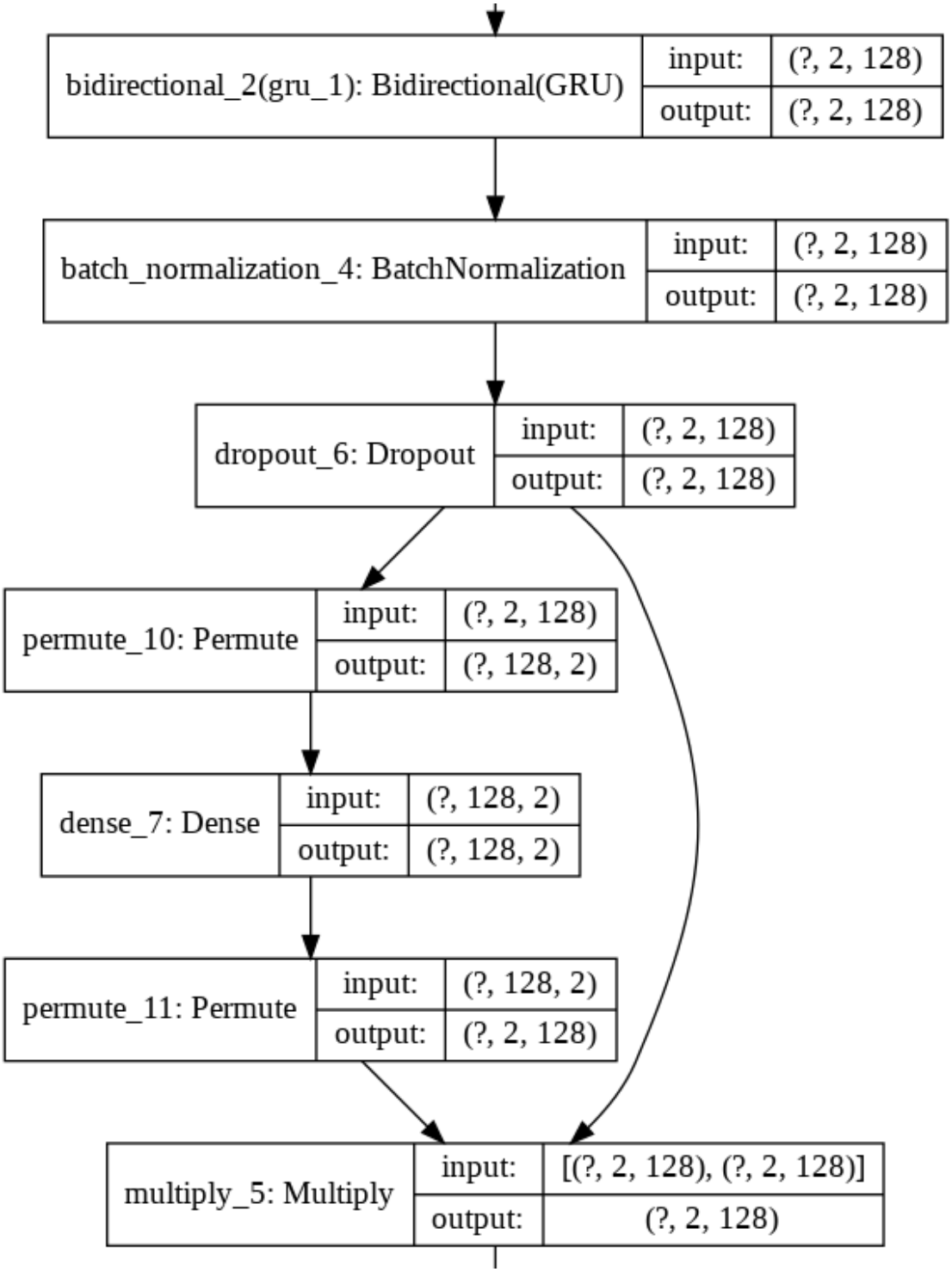

圖 17　Dual attention CNNBiGRU 第二部分

圖 17 接續第一部分圖 16，接收增強後的神經元特徵，導入 BiGRU，BiGRU 擁有過去到未來、未來到過去的特性，取得完整資訊的特性來訓練以增強過的特徵神經元，並在下層增加 Dropout 層來降低偶合度的問題，向下傳遞再次呼叫注意力機制。

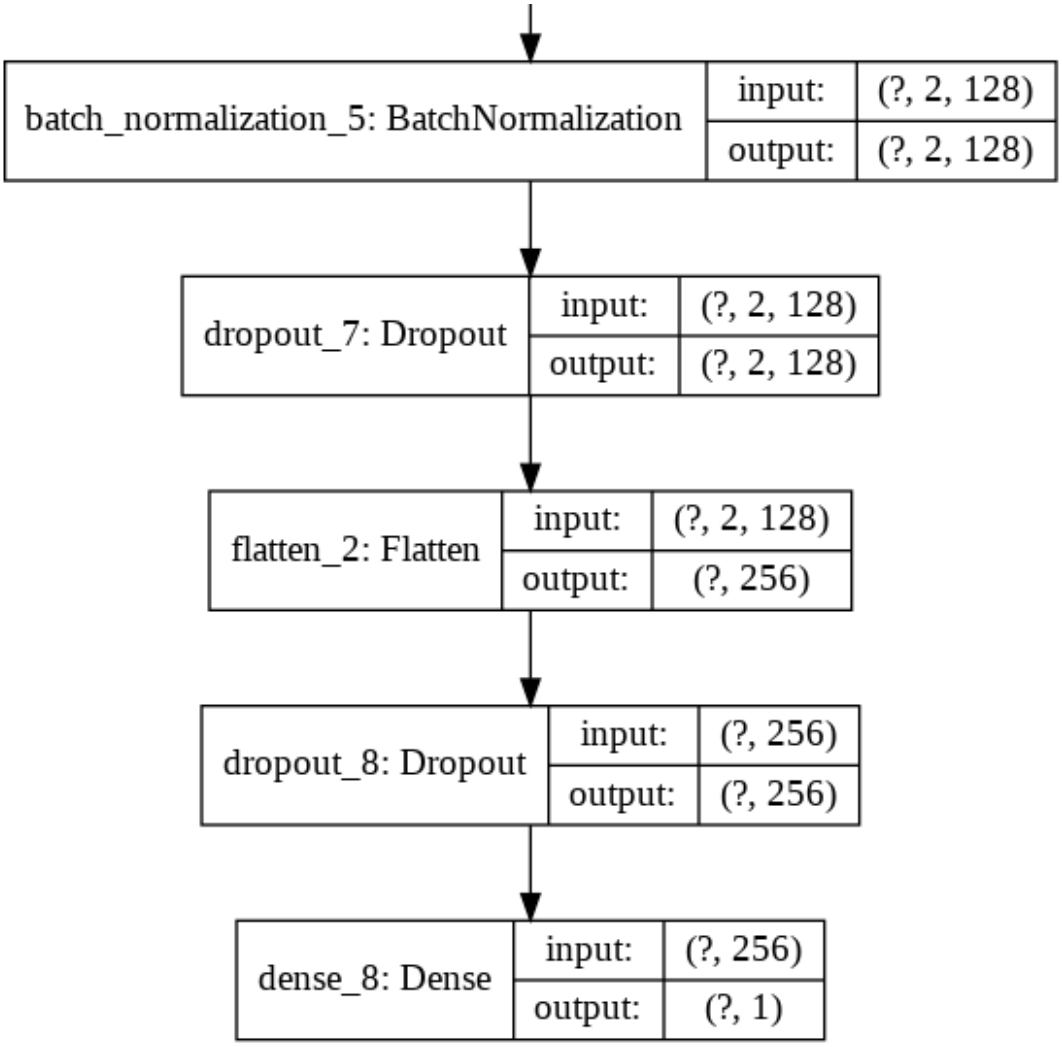

圖 18　Dual attention CNNBiGRU 第三部分

圖 18 接續第二部分，呼叫批次正規化層來使得整體神經網路加速收斂避免耦合過高問題，配合 Dropout 層，最終由於資料尺寸問題使用 Flatten 減少維度，來達成最終輸出為一維矩陣。

模型組節操作如下:

1. 透過 CNN 取得短期特徵。
2. 使用注意力機制增強學習。
3. 使用雙向 GRU 作為主體。
4. 使用 Batch Normalization 層加速收斂減少耦合。
5. 使用 Dropout 層來降低模型耦合過高無法訓練。
6. 再次新增注意力機制。

7. 再次使用 Batch Normalization 層加速訓練。
8. 增加 Flatten 層降低輸出維度。
9. 最終使用 Dense 調用 Sigmod 函數進行運算輸出。

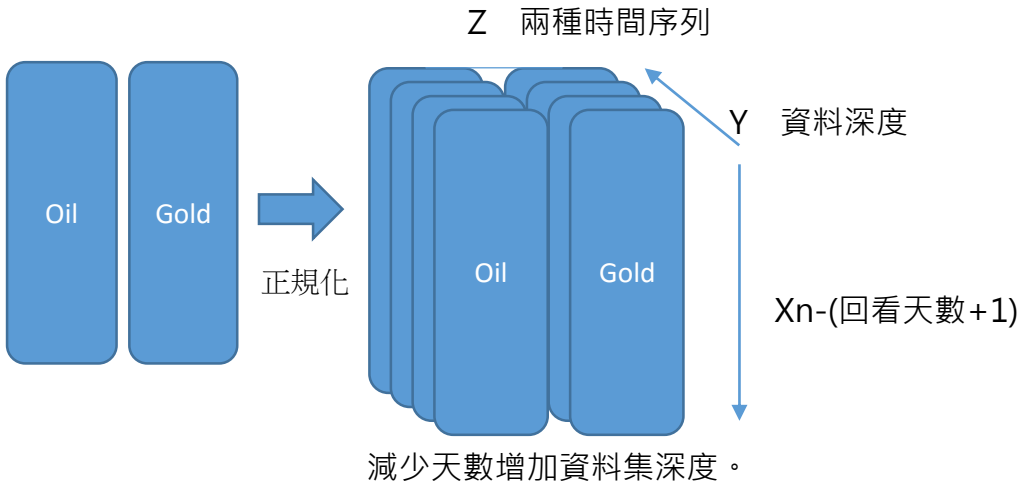

圖 19 滑動視窗圖形化

將整個滑動視窗過程進行圖形化展示敘述整體過程最終產出三維度的矩陣，應用在處理二維時間序列的資料，此方法使神經網路可以更好學習並預測，傳遞當前 i 到 i+回看天數的資料筆數並傳送完整的第二維度意味者，將減少資料筆數使其增加資料矩陣深度，最終總長度不變並產生出三維矩陣資料，附錄將展示該方法虛擬碼如何將二維度的時間序列資料轉換成三維度矩陣。

假設回看天數為 2 則意味者當天與昨天共 2 天，圖形化如下

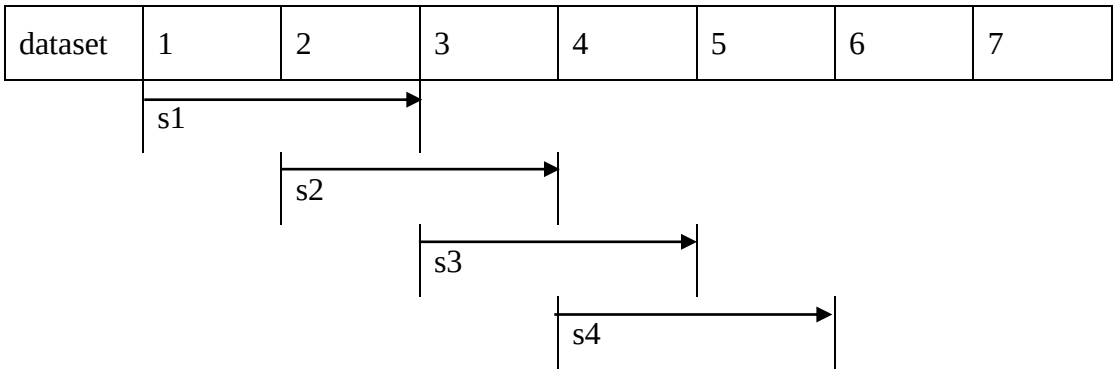

圖 20 滑動視窗圖形化 2

上圖將滑動視窗處理過程中給圖形化每個S_n都新增回看天數的數量，且長度統一為 2，當S_{n+1}時則移除前一筆此時範圍則是S_{n+1}、S_{n+2}。故使其一次學習 2 天的數據，比起單一天數輸入，此方法更能重復學習讓神經網路對於序列式數據擁有更佳的表現。附錄將展示建立好的神經網路該如何建立模型入口，與模型演算法選擇。

肆、實驗結果與分析

(一)、資料集前處理回看天數影響採樣 20 天。

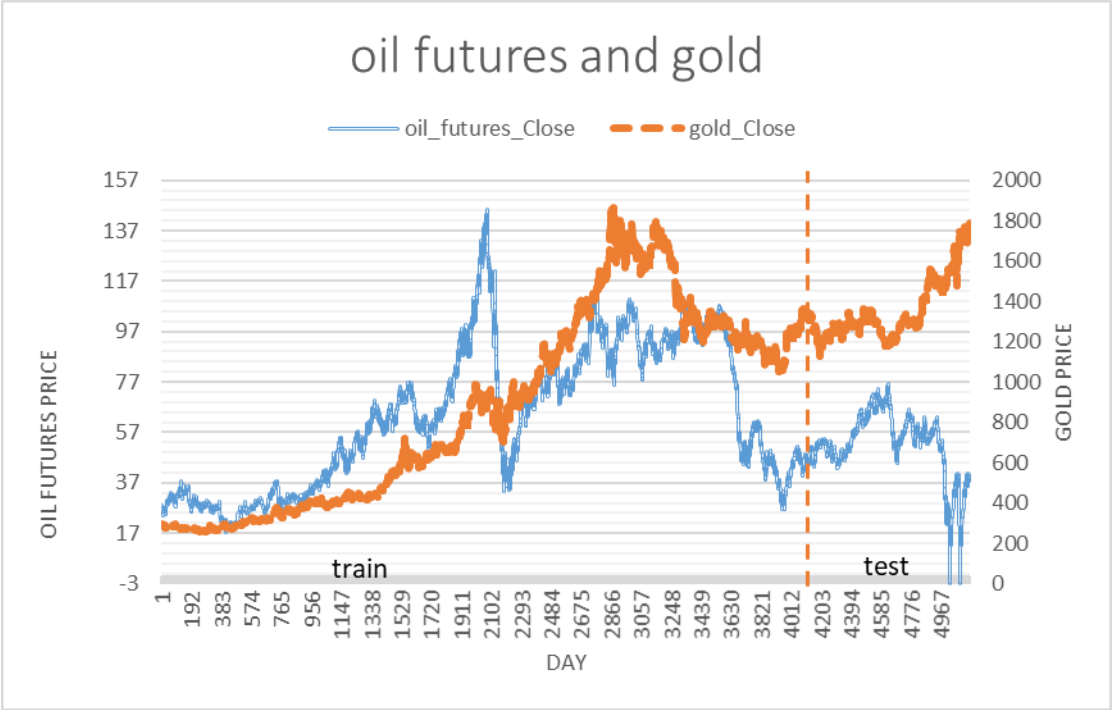

圖 21　石油期貨價格與黃金價格

圖 22 為了觀測石油期貨、黃金價格的原始趨勢，展示了兩者之間重疊後的圖示。使用 Yahoo 所提供資料集其時間序列長度為 2000/02/28 至 2020/06/29 共 5146 筆的石油期貨與黃金價格，導入正規化縮小數據後使用滑動視窗方式研究 1 至 20 天影響神經網路預測 RMSE 比較，尋找最佳結果。

		Add gold price	No gold price	Add gold price	No gold price
		BiLSTM		BiGRU	
"96/4"	RMSE	**2.82**	3.23	**2.12**	2.4
	MAE	1.98	2.01	1.31	1.6
"90/10"	RMSE	**2.17**	2.24	2.17	2.17
	MAE	1.38	1.4	1.38	1.24
"80/20"	RMSE	**2.47**	2.5	2.79	2.79
	MAE	1.52	1.5	1.95	1.83
"70/30"	RMSE	2.54	**1.83**	2.53	**1.99**
	MAE	1.65	1.06	1.64	1.1
"Avg"	RMSE	2.50	2.45	2.40	**2.34**
	MAE	1.633	1.493	1.570	1.443

圖 22　增加黃金價格與為增加黃金價格

為了後續實驗，需先研究模型再增加與尚未增加黃金價格影響 RMSE，從上圖所發現在部分切割增加黃金數據時，RMSE、MAE 精準度高於未加。

(二)、將模型與 TPA-LSTM 進行資料集預測比較如下並展示最佳模型

(1) 96%訓練 4%測試。

(2) 90%訓練 10%測試。

(3) 80%訓練 20%測試。

(4) 70%訓練 30%測試。

(三)、特徵天數延遲對於模型訓練取 8 次採樣。

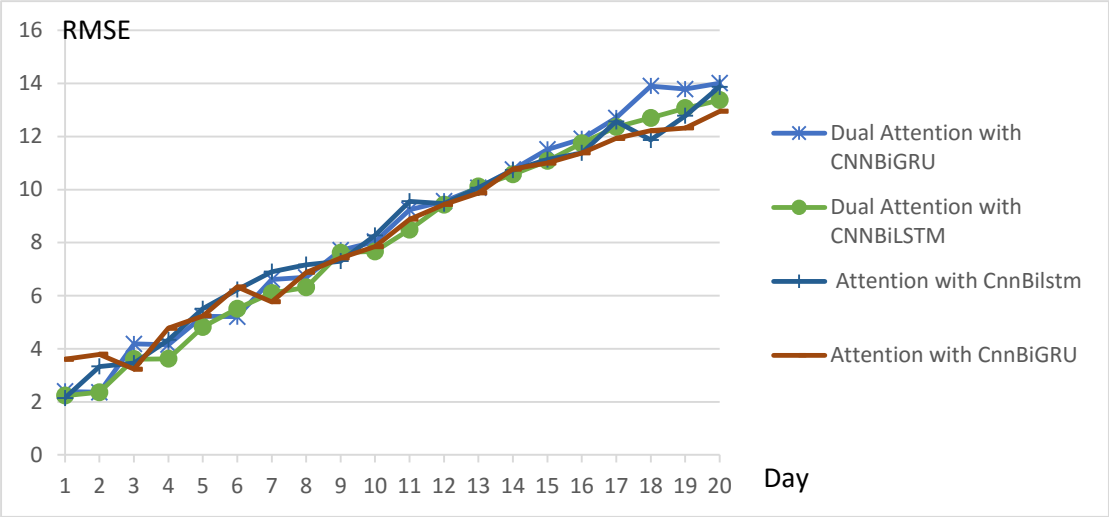

圖 23　回看天數相關圖

圖 23 展示四種模型的測試結果，發現分割資料集的方式對於回看天數越長則將導致 RMSE 越差，研究中使用回看天數 1~20 天，觀測出根據模型不同最佳 RMSE 皆落在 1~2 天內，若超出 2 天，則將導致此分割資料集方式不論哪種模型都將面臨預測精準度逐漸下降 RMSE 上升的問題。

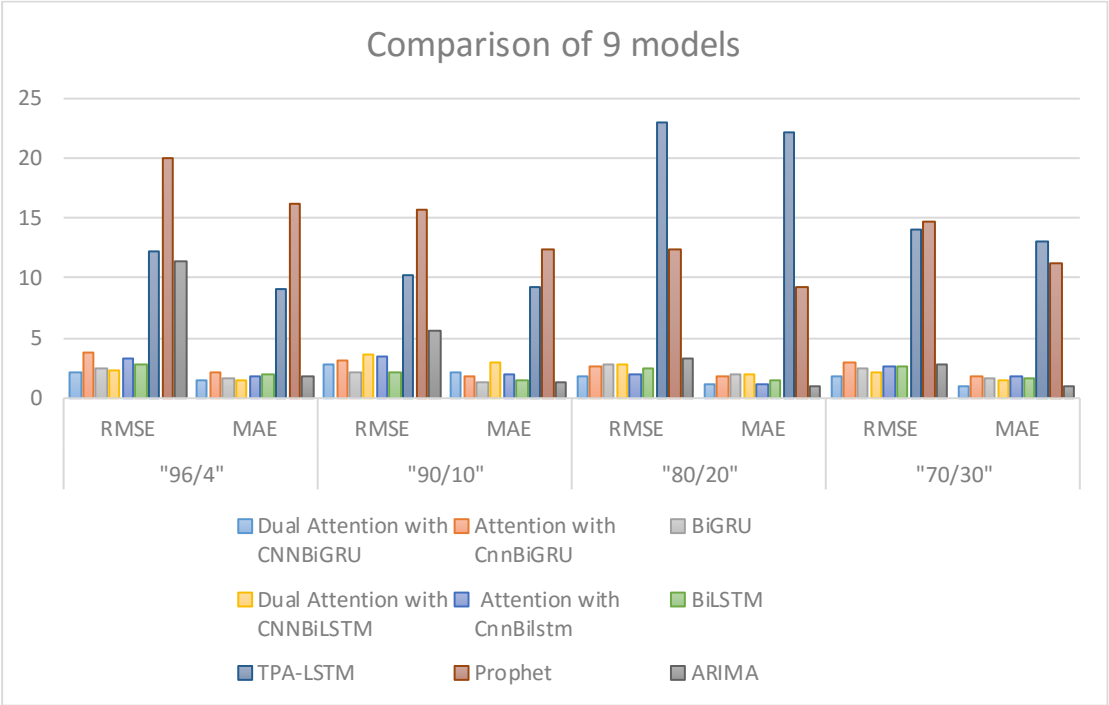

圖 24　9 種模型資料集切割比較

		Dual Attention with CNNBiGRU	Attention with CnnBiGRU	BiGRU
"96/4"	RMSE	**2.12**	3.79	2.42
	MAE	1.5	2.16	1.6
"90/10'	RMSE	2.85	3.11	**2.17**
	MAE	2.08	1.78	1.24
"80/20'	RMSE	**1.85**	2.66	2.79
	MAE	1.06	1.75	1.95
"70/30'	RMSE	**1.76**	3.02	2.53
	MAE	1	1.72	1.64
"Avg"	RMSE	**2.15**	3.15	2.48
	MAE	1.53	1.85	1.61

圖 25 九種模型比較圖一

		Dual Attention with CNNBiLSTM	Attention with CnnBilstm	BiLSTM
"96/4"	RMSE	**2.36**	3.33	2.82
	MAE	1.54	1.86	1.98
"90/10'	RMSE	3.55	3.5	**2.17**
	MAE	2.92	2.01	1.38
"80/20'	RMSE	2.8	**1.91**	2.47
	MAE	1.98	1.16	1.52
"70/30'	RMSE	**2.11**	2.64	2.54
	MAE	1.49	1.76	1.65
"Avg"	RMSE	2.71	2.85	**2.50**
	MAE	1.98	1.70	1.63

圖 26　九種模型比較圖二

		ARIMA	Prophet	TPA-LSTM
"96/4"	RMSE	**11.44**	19.98	12.24
	MAE	1.81	16.2	11.23
"90/10"	RMSE	**5.52**	15.79	10.26
	MAE	1.24	12.34	9.25
"80/20"	RMSE	**3.2**	12.41	23
	MAE	0.98	9.25	22.18
"70/30"	RMSE	**2.756**	14.69	14.06
	MAE	1.014	11.25	13.04
"Avg"	RMSE	**5.73**	15.72	14.89
	MAE	1.26	12.26	13.93

圖 27　九種模型比較圖三

上圖 26、27、28 使用多種模型做比較， Dual Attention with CNNBiGRU 與 Attention with CNN BiLSTM 模型透過資料集的改變提升了精準度，原因來自於預測的數量增多提升 RMSE、MAE 計算平均。另外由於 TPA-LSTM 模型的設計選擇使用關鍵變量，也導致 TPA-LSTM 模型針對預測長時間使用精準度無法下降，其中 Dual Attention with CNNBiGRU 比起 BiGRU 降低了 0.94RMSE。

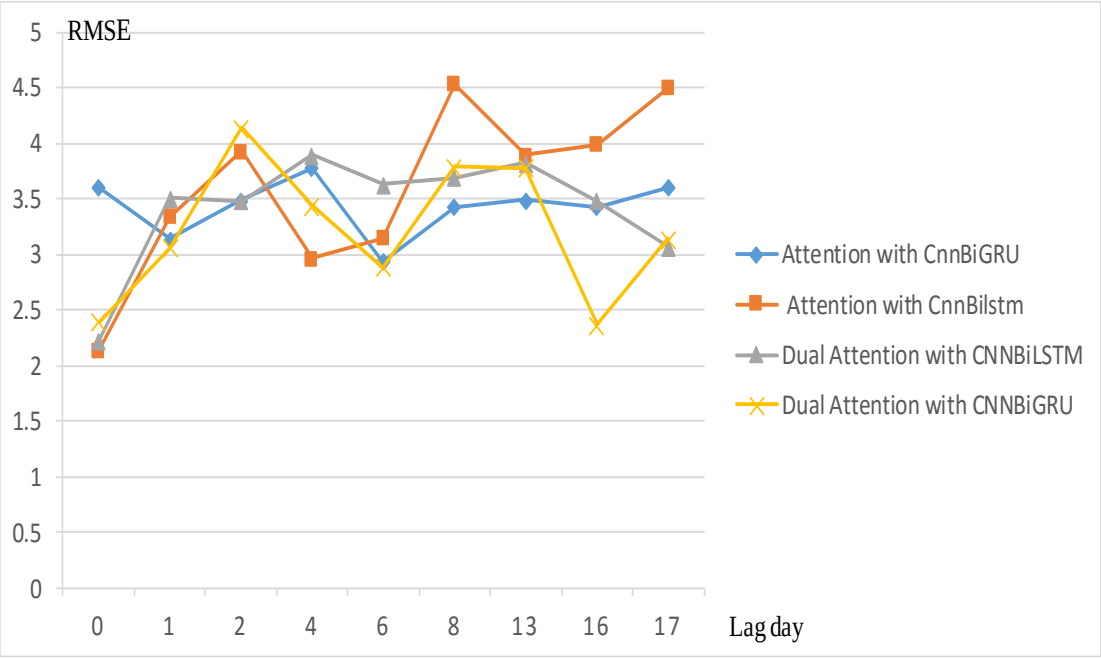

圖 28　特徵天數延後

	Dual Attention with CNNBiGRU	Dual Attention with CNNBiLSTM	Attention with CNNBiLSTM	Attention with CNNBiGRU
Gold lag days	FNormalizeMult Test Score RMSE			
0	**2.39**	**2.23**	**2.14**	3.6
1	3.06	3.51	3.34	3.14
2	4.14	3.48	3.93	3.49
4	3.44	3.89	2.96	3.78
6	2.88	3.63	3.15	**2.94**
8	3.79	3.69	4.53	3.43
13	3.78	3.82	3.89	3.49
16	2.37	3.48	3.99	3.43
17	3.14	3.07	4.5	3.6

圖 29　特徵天數延後表格

透過圖 29，使用四種模型比較使用 RMSE 計算，並取得 8 種不同天數的延遲參數，Dual Attention with CNNBiGRU、Dual Attention with CNNBiLSTM、Attention with CNNBiLSTM 在沒延遲天數獲得最佳，其餘 Attention with CNNBiGRU 則是位於第六天內獲得最佳精準度。證明特徵資料是能提升精準度另外一項實驗將驗證訓練的調整為 Train 80 test 20，下方將顯示結果與圖形繪製。

以下是 Dual Attention with CNNBiGRU 模型 80/20

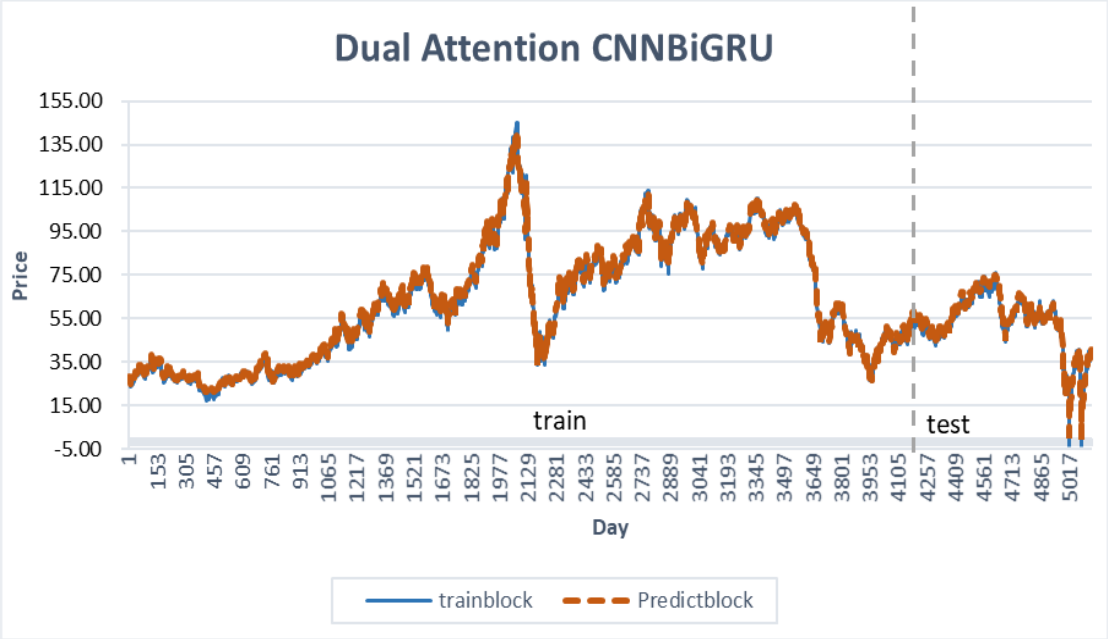

圖 30　訓練與預測圖

1. Train Score： 1.89 RMSE　1.1 MAE
2. Test Score： 1.85RMSE　1.06 MAE

圖 31 在真實的數據下，RMSE 是非常敏感的。預測差距已經得到相當精準的結果，主要影響在於波動劇烈高峰以及石油期貨趨近於 0、負值的雜訊，結果顯示模型對於雜訊處理是有能力反映此波動的並精準的預測。顯示了模型對於20%測試期間的表現即便是面對雜訊也能做出反應，本研究資料包含近期負油價的數據故導致 RMSE 結果較高但模型仍可透過注意機制的增加、CNN 提升精準度，並不會因為雜訊問題導致模型預測精準度下降。

伍、結論

本論文提出使用帶有雙重注意力機制的 CNN 雙向神經網路與滑動式窗處理資料集的方式，可以提升神經網路預測精準度。在使用 GRU 混合的模型實驗中上比起原始單一的模型預測精準度提升許多，證明我們提出的 Dual Attention Cnn BiGRU 模型比起雙向 GRU 模型和傳統 ARIMA 與 Prophet 預測效果更佳且在 94%訓練 6%測試中發現到若預測數據處於大量波動下，回歸模型無法精準的預測，促使 RMSE 與 MAE 逐漸上升，其 Prophet 則適合運用在預測筆數較小的案例上。另外發現 Dual Attention Cnn BiLSTM 模型上則表現比起單純雙向 LSTM 模型預測結果較差，原因在過於增強特徵對於雙向的 LSTM 反而促使模型無法學習長期記憶。

本論文提出一個有效率神經網路模型並進行實作，使用 CNN 擷取短期特徵傳遞給注意力機制增強特徵，從原始雙向 GRU 的 RMSE 2.79 提升 1.85，證明了我們提出的帶有注意力機制的 CNN 雙向神經網路模型比起雙向神經網路更有效率、高精準度。其結果使用 2 天的滑動式窗處理資料集，與 GRU 神經網路融入新增特徵、延遲特徵天數、雙重注意力機制、CNN，提升模型精準度，比起雙向 GRU、LSTM 得到最佳 RMSE。

未來研究除了將已成熟之模型進行實際運用挑戰更多資料外，或許能選擇其它想要預測的時間序列資料，增加對於主要預測資料具有極大相關影響的資料，藉此強化提升精準度並非只針對模型創新、混合、增加模型深度的方式，或許探索更多關於模型資料切割的方式，試圖影響提升模型精準度，改善現有的滑動視窗切割資料的問題。

本論文幾點的貢獻，首先提出帶有注意力機制的神經網路，接下來為針對資料集使用滑動式窗(回看天數)的影響。

1. 使用監督式學習可以解決預測精準度問題誤差達到 1.83。
2. 使用多維資料延遲將會影響模型訓練預測。
3. 實驗中使用 GRU 混合的模型比起 BiGRU 的模型預測精準度從原始雙向 GRU 的 RMSE 2.79 提升 1.85。
4. 滑動式窗回看天數調整越多對於模型訓練預測越困難。

參考文獻

[1]. Althelaya, K. A., El-Alfy, E. S. M., & Mohammed, S. (2018, April). Evaluation of bidirectional lstm for short-and long-term stock market prediction. In 2018 9th International Conference on Information and Communication Systems (ICICS) (pp. 151-156). IEEE.

[2]. Atsalakis, G. S., & Valavanis, K. P. (2009). Surveying stock market forecasting techniques–Part II: Soft computing methods. Expert Systems with Applications, 36(3), 5932-5941.

[3]. Bai, Y. (2019). Study on Attention-based LSTM Model for Multivariate Time-series Prediction (Doctoral dissertation, 서울대학교 대학원).

[4]. Bengio, Y. (2009). Learning deep architectures for AI. Now Publishers Inc. 1-127.

[5]. Boyacioglu, M. A., & Avci, D. (2010). An adaptive network-based fuzzy inference system (ANFIS) for the prediction of stock market return: the case of the Istanbul stock exchange. Expert Systems with Applications, 37(12), 7908-7912.

[6]. Chkili, Walid. (2015) Gold-oil prices co-movements and portfolio diversification implications..

[7]. Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

[8]. De Gooijer, Jan G., and Rob J. Hyndman. (2006) 25 years of time series forecasting. International journal of forecasting 22.3: 443-473.

[9]. Dutta, A., Das, D., Jana, R. K., & Vo, X. V. (2020). COVID-19 and oil market crash: Revisiting the safe haven property of gold and Bitcoin. Resources Policy, 69, 101816..

[10]. Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. European Journal of Operational Research, 270(2), 654-669.

[11]. Ftiti, Z., Fatnassi, I., & Tiwari, A. K. (2016). Neoclassical finance, behavioral finance and noise traders: Assessment of gold–oil markets. Finance Research Letters, 17, 33-40.

[12]. Gharbi, R. B., & Mansoori, G. A. (2005). An introduction to artificial intelligence applications in petroleum exploration and production. Journal of Petroleum Science and Engineering, 49(3-4), 93-96..

[13]. Gilliland, M., Tashman, L., & Sglavo, U. (2016). Business forecasting: Practical problems and solutions. John Wiley & Sons..

[14]. Graves, A., Mohamed, A. R., & Hinton, G. (2013, May). Speech recognition with deep recurrent neural networks. In 2013 IEEE international conference on acoustics, speech and signal processing (pp. 6645-6649). IEEE.

[15]. Hiransha, M., Gopalakrishnan, E. A., Menon, V. K., & Soman, K. P. (2018). NSE stock market prediction using deep-learning models. Procedia computer science, 132, 1351-1362.

[16]. Jia, H. (2016). Investigation into the effectiveness of long short term memory networks for stock price prediction. arXiv preprint arXiv:1603.07893.:

[17]. K. Duan, S. Keerthi, A.N. Poo.(2003) Evaluation of simple performance measures for tuning SVM hyperparameters, Neurocomputing 51 41–59.

[18]. Kazem, A., Sharifi, E., Hussain, F. K., Saberi, M., & Hussain, O. K. (2013). Support vector regression with chaos-based firefly algorithm for stock market price forecasting. Applied soft com puting, 13(2), 947-958.

[19]. Kim, H. Y., & Won, C. H. (2018). Forecasting the volatility of stock price index:

A hybrid model integrating LSTM with multiple GARCH-type models. Expert Systems with Applications, 103, 25-37.

[20]. Kim, T. Y., & Cho, S. B. (2018). Web traffic anomaly detection using C-LSTM neural networks. Expert Systems with Applications, 106, 66-76.

[21]. Ling, S., & McAleer, M. (2003). Asymptotic theory for a vector ARMA-GARCH model. Econometric theory, 280-310.

[22]. Mo, B., Nie, H., & Jiang, Y. (2018). Dynamic linkages among the gold market, US dollar and crude oil market. Physica A: Statistical Mechanics and its Applications, 491, 984-994.

[23]. Narayan, Paresh Kumar, Seema Narayan, and Xinwei Zheng. (2010) Gold and oil futures markets: Are markets efficient. 87.10: 3299-3303.

[24]. Nelson, D. M., Pereira, A. C., & de Oliveira, R. A. (2017, May). Stock market's price movement prediction with LSTM neural networks.

[25]. Omprakash Yadav ,Yash Shah,Saneesha Talim, Brian Britto. (2019). Prediction of Stock Closing Price by applying Long Stock Term Memory Neural Network.

[26]. Pang, X., Zhou, Y., Wang, P., Lin, W., & Chang, V. (2018). An innovative neural network approach for stock market prediction. The Journal of Supercomputing, 1-21.

[27]. Peng, Y., & Jiang, H. (2015). Leverage financial news to predict stock price movements using word embeddings and deep neural networks. arXiv preprint arXiv:1506.07220.:

[28]. Roondiwala, M., Patel, H., & Varma, S. (2017). Predicting stock prices using LSTM. International Journal of Science and Research (IJSR), 6(4), 1754-1756.:

[29]. Sachdeva, A., Jethwani, G., Manjunath, C., Balamurugan, M., & Krishna, A. V. (2019, March). An Effective Time Series Analysis for Equity Market Prediction

Using Deep Learning Model. In 2019 International Conference on Data Science and Communication (IconDSC) (pp. 1-5). IEEE.

[30]. Sagheer, A., & Kotb, M. (2019). Time series forecasting of petroleum production using deep LSTM recurrent networks. Neurocomputing, 323, 203-213.

[31]. Saldivar, Frank, and Mauricio Ortiz(2019). Stock Market Price Prediction Using Various Machine Learning Approaches."

[32]. Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. IEEE transactions on Signal Processing, 45(11), 2673-2681.

[33]. Seabold, S., & Perktold, J. (2010, June). Statsmodels: Econometric and statistical modeling with python. In Proceedings of the 9th Python in Science Conference (Vol. 57, p. 61).

[34]. Selvin, S., Vinayakumar, R., Gopalakrishnan, E. A., Menon, V. K., & Soman, K. P. (2017, September). Stock price prediction using LSTM, RNN and CNN-sliding window model. In 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI) (pp. 1643-1647). IEEE.:

[35]. Shih, S. Y., Sun, F. K., & Lee, H. Y. (2019). Temporal pattern attention for multivariate time series forecasting. Machine Learning, 108(8-9), 1421-1441..

[36]. Singhal, S., Choudhary, S., & Biswal, P. C. (2019). Return and volatility linkages among International crude oil price, gold price, exchange rate and stock markets: Evidence from Mexico. Resources Policy, 60, 255-261.

[37]. Sun, C. S., Wang, Y. N., & Li, X. R. (2008). A vector autoregression model of hourly wind speed and its applications in hourly wind speed forecasting. PROCEEDINGS-CHINESE SOCIETY OF ELECTRICAL ENGINEERING, 28(14), 112.

[38]. Taylor, S. J., & Letham, B. (2018). Forecasting at scale. The American Statistician,

72(1), 37-45..

[39]. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

[40]. Xingjian, S. H. I., Chen, Z., Wang, H., Yeung, D. Y., Wong, W. K., & Woo, W. C. (2015). Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In Advances in neural information processing systems (pp. 802-810).

[41]. Zhang, G. P. (2003). Time series forecasting using a hybrid ARIMA and neural network model. Neurocomputing, 50, 159-175.

附錄一

虛擬碼展示

```
    Input cell state, Output Atention_multiply cell
1   attention_function ( receive cell state from the upper layer cell state)
    #接收上層所傳遞的神經元
2       TimeSteps ← get shape then sent second dimension size from backend
                    keras
        #取得後端第二維度尺寸並賦予 TimeSteps
3       a ← replace the first and second dimensions of the input.
        #將輸入的神經元進行第一維度與第二維度做維度交換
4       a ← use connection layer receive upper layer a and set TimeSteps for
            cell hen call softmax fuction compute
            #使用連接層接收上層 a 並設置 TimeSteps 調用 softmax 函數計
    算
5       a_probs ← replace the first and second dimensions of the a.
        #將上層 a 的輸出進行第一維度與第二維度做維度交換
6       Attention_multiply←call input cell state and a_probs for matrix
    multiplication
        # input cell state  與 a_probs  進行矩陣相乘
7       Output Atention_multiply cell
```

圖 31　注意力機制虛擬碼

展示注意力機制在模型裡實作，使用虛擬碼撰寫解釋注意力機制運作過程。將定義注意力機制。首先行數 2 從後端取得張量矩陣大小分別賦予 Timestep 第二維度尺寸，接者行數 3 接收上層傳遞下來的神經元第一維度與第二維度進行維度交換並把此層命名為 a，行數 4 定義連接層的參數，神經元數量帶入設定為 Timestep 呼叫 softmax 函數計算，行數 5 接收上層輸出進行，第一維度與第二維度做維度交換恢復原始資料的矩陣尺寸與原始輸入使用矩陣相乘來增強注意。

```
1       Input Dataset from oil and gold
        output Normalize data and normalize
        NormalizeMult(data)
        #多元正規化接收來自原始的資料集，石油期貨和黃金
2       data ← input an array form two dimensional data #導入資料集
3       normalize ←create array of data second dimension size × 2
        #創造陣列，大小為資料集第二維度大小並乘 2
4       normalize ← Reshape data to data second dimension size and 2
        #重新調整 normalize，尺寸設定資料集第二維與 2
5       for i (0 to second dimension size of data) do #重復執行 0~第二維度大小
6           list ← full first dimension and second dimension i size of data
            #每次都傳給 list 完整第一維度資料與第二維度 i
7           Listlow , listhigh ← Compute list the 0 and 100 percentile
            #計算 list 中最大最小百分比
8           normalize[i,0] ← listlow#傳遞當次最小百分比
9           normalize[i,1] ← listhigh#傳遞當次最大百分比
10          delta ← listhigh – listlow#最大與最小相減
11          if delta not 0 then
    #若兩者相減不等於零則進行後續運算
12                  for j (0 to first dimension size of data) do
    #重複執行直到 j 走完第一維度的長度
13                      data[j,i]  ←(data[j,i]−listlow)/delta
14                  end for
15      output  Normalize data and normalize
    #最終傳遞出正規化的數據與對應反轉回原始的數據
```

圖 32　正歸化虛擬碼

功能將進行多維度原始資料矩陣正規化處理，以利於後續神經網路的學習。首先行數 1 接收石油與黃金的資料集接者行數 2 則將此資料數值傳給 data 參數，並在行數 3 創造陣列並設定大小為資料集第二維度並乘 2 的尺寸，接續行數 4 則把 normalize 取得資料第二維度大小進行重朔建立成矩陣，行數 5~14 則使用迴

圈將從資料的第 0 位至最後一位走過一次，每次都會進行數值最大與最小差值計算，最終將把原始數據縮小使得類神經網路進行迭代時收斂，最終將產出 Normalize data and normalize 下圖將接續展示反轉正規化虛擬碼。

```
1   Input Normalize data and normalize Output original data
    FNormalizeMult(data and normalize)
    #接收正規化資料與還原正規化
2     data ← input an array form two dimensional data
      #導入資料集
3     for i (0 to second dimension size of data) do
      #重復執行 0~第二維度大小
4         listlow ← i size dimension and second dimension 0 size from
                    normalize
                    #選擇第一維度的 i 位和第二維度位置 0 的給 list low
5         listhigh← i size dimension and second dimension 1 size from
                    normalize
                    #選擇第一維度 i 個和第二維度 1 個的給 list low
6         delta ← listhigh – listlow
          #最大與最小相減
7         if delta not 0 then
          #若兩者相減不等於零則進行後續運算
8           for j (0 to first dimension size of data) do
            #重複執行直到 J 走完第一維度的長度
9               data[j,i] ←data[j, i] × delta + listlow
10          end for
11    return restore original data #還原經過正規化縮小的資料
```

圖 33　反正歸化虛擬碼

圖 34 功能用於還原經由圖 33 正規化後的資料，接收圖 33 產出的 normalize 與資料進行還原處理應用於後續 RMSE 計算時顯示精準度，此功能與 MinMaxScaler 相同皆為縮小放大資料處理，但本次研究使用是二維時間序列，無法應用於 MinMaxScaler 等方法故創建兩個縮小與放大之功能為圖 14、圖 15，後續將展示滑動視窗方法虛擬碼。

```
    Input Normalize data Output Predict data
1   DualAttention_with_CnnBiGRU_model()
    #DualAttention_with_CnnBiGRU 模型創建
2   input layer ← TimeSteps and dataset dims #全輸入層
3   Setting Conv1D layer then call 'relu' #設定 CNN 使用 relu 函數#
4   call attention_function #呼叫注意力機制
5   use Bidirectional to wrap GRU #使用雙向 GRU
6   Then add the BatchNormalization layer to prevent over coupling
    #增加 BatchNormalization  來加速收斂減少耦合
7   add drop layer #丟棄層
8   call attention_function again #再次呼叫注意力機制
9   add BatchNormalization layer again
    #再次使用 BatchNormalization  來加速收斂減少耦合
10  add drop layer
11  Add flatten for down dimension    #呼叫 flatten 層來降低維度
12  add drop layer
13  Dense ← use full dense layer call sigmoid  將全接收上層傳遞的神經元
14  output ←Dense #將 Dense 全連接層輸出一維矩陣
```

圖 34　Dual Attention with CNNBiGRU 虛擬碼

為 keras 提供的格式，建立模型使用 Keras 層分別有 Conv1D、Bidirectional、LSTM、Dropout 和先前定義的功能注意力機制，在某些層別裡使用函數提升模型精準度與下降預測 loss 數，加速收斂，透過修改圖 14 層別增加減少注意力機制可以分別組成多種模型。

模型使用的函數：

1. Relu：又稱修正線性單元，相比 tanh 雙曲函數，更加有效率的梯度下降以及反向傳播，避免了梯度爆炸和梯度消失問題。
2. Sigmoid：輸出實數範圍，範圍為 0 到 1。

```
    Input dataset and lookback Output TrainX and TargetY
1   create_dataset (dataset and look_back)#滑動視窗切割接收資料集與回看天數
2       dataX and dataY both of the list #創造兩個 X 與 Y 的列表
3       For i from 0 ~ dataset length − (look_back + 1) do
4           a ←dataset first dimension form i to i + look_back and full second
                dimension
                #資料集第一維度從 i~i+ look_back 第二維度則完整傳送
5           dataX append a #將 a 加入 dataX 列表尾端
6           dataY append dataset form first dimension i + look_back and full
            second dimension
            #將 dataset 第一維度的 i+look_back 大小與完
            #整第二維度加入 dataY 列表尾端
7       TrainX ← change dataX to array
8       Train_Y ←change dataY to array
9       Output TrainX and TargetY #產出訓練 X 與目標 Y
```

圖 35　滑動視窗虛擬碼

透過此方式分割二維資料集。功能內執行時將傳遞當前 i 到 i+回看天數的資料筆數並傳送完整的第二維度意味者，將減少資料筆數使其增加資料矩陣深度，最終總長度不變並產生出三維矩陣資料，為後續神經網路訓練與測試做調整，達成三維矩陣與單一維度陣列，訓練 X 將在模型迭代時透過目標 Y 去計算每次的收斂數值。

```
1   Setting DualAttention_with_CnnBiGRU_model
        choice compile loss use mean_squared_error and optimizer adam
        #使用 MSE 做收斂損失計算  並選擇亞當演算法
2   Setting Epochs for 44
3   trainPredict ← DualAttention_with_CnnBiGRU_model predict(trainX)
    #輸入 trainX 的訓練資料給模型，結果命名為 trainPredict
4   testPredict ←DualAttention_with_CnnBiGRU_model predict(testX)
    #輸入 testX 的訓練資料給模型，結果命名為 testPredict
```

圖 36　模型訓練預測設定

準備建立定義配置訓練模型，行數 1 進行演算法與收斂計算方式設定。行數 2 則是給予模型的循環次數，行數 3、4 則是分別進行訓練資料集的預測和驗證資料的預測，最終結果分別傳至 trainPredict 與 testPredict。

```
1   trainPredict_FNormalizeMult← FNormalizeMult(trainPredict,tr_normalize)
2   testPredict_FNormalizeMult← FNormalizeMult(testPredict,te_normalize)
```

圖 37　呼叫反轉正規化

將預測出來的數據進行反正規化，把正規化數據根據對應的 normalize 反轉回初始數據以利後續進行 RMSE 計算，例如 trainPredict 對應 tr_normalize。若不將此反轉回原始大小，直接計算 RMSE 會遇到數值過小的問題無法判斷模型預測精準度是否提升，或下降。

```
1   trainScore ← use RMSE calculate train and train Predict
    #計算訓練的 RMSE 並命名為 trainScore
2   testScore ← use RMSE calculate testY and testPredict
    #計算驗證的 RMSE 並命名為 testScore
3   teV  ←  sent testVaild first dimension size of 0~ -(look_back+1)
    #取得原始 testVaild，並只取得石油期貨價格
4   teV  ←  testVaild[:-(look_back+1),0]
    #取得原始 testVaild  數據從 0 到最後  - (look_back+1)
5   trV ←trainVaild[:-(look_back+1),0]
    #取得原始 trainVaild 數據從 0 到最後  - (look_back+1)
6   trainScore  ←  use RMSE calculate trV and trainPredict_FNormalizeMult
    #計算 train 反正規化後的數據 RMSE
7   testScore ← use RMSE calculate teV and testPredict_FNormalizeMult
    #計算 tes 反正規化後的數據 RMSE
```

圖 38　RMSE 計算

將進行 RMSE 計算 1、2 行計算尚未反轉正規的數據，行數 3 則將原始二維度的資料集，只取石油期貨價格並在 4、5 行擷取對應訓練、測試的筆數的長度，使用在行數 6、7 再次進行 RMSE 計算，分別產出反正規化前，與反正規化後的 RMSE 顯示，由於 RMSE 計算的方式對於較小的數值則容易顯示較低的 RMSE 因此容易出現 RMSE 較低但預測圖看起來精準度效果不佳，故須進行反正規化的行為。

附錄二

程式碼來源:

https://colab.research.google.com/drive/10Vzs91QLxOM4iRc9Yhn33yexk0zb3GyR?usp=sharing

以下將展示網址內的程式碼其中以滑動視窗、注意力機制、正規化與反正歸化應用於多個模型故後續模型內，不再重復撰寫。

```
#滑動視窗
def create_dataset2(dataset, look_back):
	dataX, dataY = [], []
	for i in range(len(dataset)-look_back-1):
		a = dataset[i:(i+look_back),:]
		dataX.append(a)
		dataY.append(dataset[i + look_back,:])
	TrainX = numpy.array(dataX)
	Train_Y = numpy.array(dataY)
	return TrainX, Train_Y
```

圖 39　滑動視窗程式碼

```
def attention_function(inputs, single_attention_vector=False):
    #inputs.shape = (batch_size, TimeSteps, Dims)
	TimeSteps = K.int_shape(inputs)[1]
	input_dim = K.int_shape(inputs)[2]
	a = Permute((2, 1))(inputs)
	a = Dense(TimeSteps, activation='softmax')(a)
	if single_attention_vector:
		a = Lambda(lambda x: K.mean(x, axis=1))(a)
		a = RepeatVector(input_dim)(a)
	a_probs = Permute((2, 1))(a)
	# element * wise
	output_attention_mul = Multiply()([inputs, a_probs])
	return output_attention_mul
```

圖 40　注意力機制程式碼

```
#正規化與反正歸化
def NormalizeMult(data):
    #normalize
    data = np.array(data)
    normalize = np.arange(2*data.shape[1],dtype='float64')
    normalize = normalize.reshape(data.shape[1],2)
    # print(normalize.shape)
    for i in range(0,data.shape[1]):
        list = data[:,i]
        listlow,listhigh =  np.percentile(list, [0, 100])
        # print(i)
        normalize[i,0] = listlow
        normalize[i,1] = listhigh
        delta = listhigh - listlow
        if delta != 0:
            for j in range(0,data.shape[0]):
                data[j,i]  =  (data[j,i] - listlow)/delta
    #np.save("./normalize.npy",normalize)
    return  data,normalize
def FNormalizeMult(data,normalize):
    data = np.array(data)
    for i in  range(0,data.shape[1]):
        listlow =  normalize[i,0]
        listhigh = normalize[i,1]
        delta = listhigh - listlow
        if delta != 0:
            #第 j 行
            for j in range(0,data.shape[0]):
                data[j,i]  =  data[j,i]*delta + listlow
    return data
```

圖 41　正規化與反正歸化

一、Dual Attention with CNNBiGRU

```
from keras.layers import Input, Dense, LSTM,
merge ,Conv1D,Dropout,Bidirectional,Multiply,Concatenate,BatchNormalization,
GRU
from keras.models import Model
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from keras.layers import merge
from keras.layers.core import *
from keras.layers.recurrent import LSTM
from keras.models import *
from keras.utils import plot_model
from keras import optimizers
import numpy
import    pandas as pd
import math
import datetime
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM,Dropout
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras import backend as K
```

圖 42　Dual attention with CNNBiGRU 程式碼

```
# fix random seed for reproducibility
numpy.random.seed(1)
# load the dataset
dataframe = read_csv('/content/CNN_BiLSTM_withAttention/OilwithGold.csv')
dataframe = dataframe.drop(['Date'], axis = 1)
dataset = dataframe.values
dataset = dataset.astype('float32')
testVaild=dataset
trainVaild=dataset
# split into train and test sets
train_size = int(len(dataset) * 0.9)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
testVaild = testVaild[train_size:len(dataset),:]
trainVaild = trainVaild[:train_size,:]
# normalize the dataset
dataset,da_normalize = NormalizeMult(dataset)
train,tr_normalize = NormalizeMult(train)
test,te_normalize = NormalizeMult(test)
look_back = 2
TimeSteps=look_back
Dims=2
trainX, trY = create_dataset2(train, look_back)
testX, teY = create_dataset2(test, look_back)
trainY =trY[:,0]
testY =teY[:,0]
```

圖 43　Dual attention with CNNBiGRU 程式碼 2

```
def attention_model():
    inputs = Input(shape=(TimeSteps, Dims))
    x = Conv1D(filters = 128, kernel_size = 1, activation = 'relu')(inputs)
    attention = attention_function(x)
    BiGRU_out = Bidirectional(GRU(64,
return_sequences=True,activation="relu"))(attention)
    Batch_Normalization = BatchNormalization()(BiGRU_out)
    Drop_out = Dropout(0.1)(Batch_Normalization)
    attention = attention_function(Drop_out)
    Batch_Normalization = BatchNormalization()(attention)
    Drop_out = Dropout(0.1)(Batch_Normalization)
    Flatten_ = Flatten()(Drop_out)
    output=Dropout(0.1)(Flatten_)
    output = Dense(1, activation='sigmoid')(output)
    model = Model(inputs=[inputs], outputs=output)
    return model

m = attention_model()
m.summary()
m.compile(loss='mean_squared_error', optimizer='adam')
m.fit(trainX, trainY, epochs=33, batch_size=64,
verbose=0,validation_data=(testX, testY))
# make predictions
trainPredict = m.predict(trainX)
testPredict = m.predict(testX)
#FNormalize
trainPredict_FNormalizeMult= FNormalizeMult(trainPredict,tr_normalize)
testPredict_FNormalizeMult= FNormalizeMult(testPredict,te_normalize)
trainScore = math.sqrt(mean_squared_error(trainY, trainPredict))
print('*    Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY, testPredict))
print('*    Test Score: %.2f RMSE' % (testScore))
teV=testVaild[:-(look_back+1),0]
trV=trainVaild[:-(look_back+1),0]
plt.plot(teV)
plt.plot(testPredict_FNormalizeMult,'r--')
plt.savefig('teV')
```

圖 44　Dual attention with CNNBiGRU 程式碼 3

```
testScore = math.sqrt(mean_squared_error(teV, testPredict_FNormalizeMult))
print('*    FNormalizeMult Test Score: %.2f RMSE' % (testScore))
plot_model(m, show_shapes=True, to_file='DualAttention_with_CnnBiGRU.png')
plt.plot(trV)
plt.plot(trainPredict_FNormalizeMult,'r--')
plt.savefig('trV')
plt.show()
trainScore = math.sqrt(mean_squared_error(trV, trainPredict_FNormalizeMult))
print('*    FNormalizeMult Train Score: %.2f RMSE' % (trainScore))
```

圖 45　Dual attention with CNNBiGRU 程式碼 4

二、Attention with CNNBiGRU

```
# load the dataset
dataframe = read_csv('/content/CNN_BiLSTM_withAttention/OilwithGold.csv')
dataframe = dataframe.drop(['Date'], axis = 1)
dataset = dataframe.values
dataset = dataset.astype('float32')
testVaild, trainVaild =dataset, dataset
# split into train and test sets
train_size = int(len(dataset) * 0.7)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
testVaild = testVaild[train_size:len(dataset),:]
trainVaild = trainVaild[:train_size,:]
print(train.shape)
print(test.shape)
# normalize the dataset
dataset,da_normalize = NormalizeMult(dataset)
train,tr_normalize = NormalizeMult(train)
test,te_normalize = NormalizeMult(test)
look_back = 2
TimeSteps=look_back
Dims=2
trainX, trY = create_dataset2(train, look_back)
testX, teY = create_dataset2(test, look_back)
trainY =trY[:,0]
testY =teY[:,0]
```

圖 46　Attention with CnnBiGRU 程式碼

```
def attention_model():
    inputs = Input(shape=(TimeSteps, Dims))
    x = Conv1D(filters = 128, kernel_size = 1, activation = 'relu')(inputs)
    BiGRU_out = Bidirectional(GRU(64,
return_sequences=True,activation="relu"))(x)
    Batch_Normalization = BatchNormalization()(BiGRU_out)
    Drop_out = Dropout(0.1)(Batch_Normalization)
    attention = attention_function(Drop_out)
    Batch_Normalization = BatchNormalization()(attention)
    Drop_out = Dropout(0.1)(Batch_Normalization)
    Flatten_ = Flatten()(Drop_out)
    output=Dropout(0.1)(Flatten_)
    output = Dense(1, activation='sigmoid')(output)
    model = Model(inputs=[inputs], outputs=output)
    return model
m = attention_model()
m.summary()
m.compile(loss='mean_squared_error', optimizer='adam')
m.fit(trainX, trainY, epochs=30, batch_size=64,
verbose=0,validation_data=(testX, testY))
# make predictions
trainPredict = m.predict(trainX)
testPredict = m.predict(testX)
#FNormalize
trainPredict_FNormalizeMult= FNormalizeMult(trainPredict,tr_normalize)
testPredict_FNormalizeMult= FNormalizeMult(testPredict,te_normalize)
```

圖 47　Attention with CNNBiGRU 程式碼 2

```
#calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY, trainPredict))
print('*    Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY, testPredict))
print('*    Test Score: %.2f RMSE' % (testScore))
teV=testVaild[:-(look_back+1),0]
trV=trainVaild[:-(look_back+1),0]
print("---")
print("*    look_back = ",look_back)
trainScore = math.sqrt(mean_squared_error(trV, trainPredict_FNormalizeMult))
print('*    FNormalizeMult Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(teV, testPredict_FNormalizeMult))
print('*    FNormalizeMult Test Score: %.2f RMSE' % (testScore))
# shift test predictions for plotting
plt.plot(teV)
plt.plot(testPredict_FNormalizeMult,'r')
plt.show()
plot_model(m, show_shapes=True, to_file='AttentionCnnBiGRU.png')
plt.plot(trV)
plt.plot(trainPredict_FNormalizeMult,'r')
plt.show()
```

圖 48　Attention with CNNBiGRU 程式碼 4

三、BiGRU

```
# load the dataset
dataframe = read_csv('/content/drive/My Drive/NewCRAA/OilwithGold.csv')
dataframe = dataframe.drop(['Date'], axis = 1)
# dataframe = dataframe.drop(['gold_Close'], axis = 1)
dataset = dataframe.values
dataset = dataset.astype('float32')
testVaild, trainVaild =dataset,dataset
# split into train and test sets
train_size = int(len(dataset) * 0.8)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
testVaild = testVaild[train_size:len(dataset),:]
trainVaild = trainVaild[:train_size,:]
look_back, Dims = 2,2
trainX, trY = create_dataset2(train, look_back)
testX, teY = create_dataset2(test, look_back)
trainY, testY =trY[:,0], teY[:,0]
model_1 = Sequential()
model_1.add(Bidirectional(GRU(64,
input_shape=(look_back ,Dims),activation="relu")))
model_1.add(Dropout(0.1))
model_1.add(Dense(1 ,activation='sigmoid'))
model_1.compile(loss='mean_squared_error', optimizer='adam')
model_1.fit(trainX, trainY, epochs=35 , batch_size=128,
verbose=0,validation_data=(testX, testY))
model_1.summary()
# make predictions
trainPredict = model_1.predict(trainX)
testPredict = model_1.predict(testX)
#FNormalize
trainPredict_FNormalizeMult= FNormalizeMult(trainPredict,tr_normalize)
testPredict_FNormalizeMult= FNormalizeMult(testPredict,te_normalize)
```

圖 49　BiGRU 程式碼

```
#calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY, trainPredict))
print('*    Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY, testPredict))
print('*    Test Score: %.2f RMSE' % (testScore))
teV=testVaild[:-(look_back+1),0]
trV=trainVaild[:-(look_back+1),0]
trainScore = math.sqrt(mean_squared_error(trV, trainPredict_FNormalizeMult))
print('*    FNormalizeMult Train Score: %.2f RMSE' % (trainScore))
trainScoreMAE = mean_absolute_error(trV, trainPredict_FNormalizeMult)
print('*    Train Score: %.2f MAE' % (trainScoreMAE))
testScore = math.sqrt(mean_squared_error(teV, testPredict_FNormalizeMult))
print('*    FNormalizeMult Test Score: %.2f RMSE' % (testScore))
testScore = mean_absolute_error(teV, testPredict_FNormalizeMult)
print('*    Test Score: %.2f MAE' % (testScore))
# shift test predictions for plotting
plt.plot(teV)
plt.plot(testPredict_FNormalizeMult,'r')
plt.show()
plot_model(model_1, show_shapes=True, to_file='GRU.png')
plt.plot(trV)
plt.plot(trainPredict_FNormalizeMult,'r')
plt.show()
```

圖 50　BiGRU 程式碼 2

四、Dual Attention with CNNBiLSTM

```
dataframe = read_csv('/content/CNN_BiLSTM_withAttention/OilwithGold.csv')
dataframe = dataframe.drop(['Date'], axis = 1)
dataset = dataframe.values
dataset = dataset.astype('float32')
testVaild, trainVaild =dataset, dataset
# split into train and test sets
train_size = int(len(dataset) * 0.9)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
testVaild = testVaild[train_size:len(dataset),:]
trainVaild = trainVaild[:train_size,:]
# normalize the dataset
dataset,da_normalize = NormalizeMult(dataset)
train,tr_normalize = NormalizeMult(train)
test,te_normalize = NormalizeMult(test)
look_back, Dims = 2,2
TimeSteps=look_back
trainX, trY = create_dataset2(train, look_back)
testX, teY = create_dataset2(test, look_back)
trainY,testY =trY[:,0],teY[:,0]
def attention_model():
    inputs = Input(shape=(TimeSteps, Dims))
    x = Conv1D(filters = 128, kernel_size = 1, activation = 'relu')(inputs)
    attention = attention_function(x)
    BiGRU_out = Bidirectional(GRU(64,
return_sequences=True,activation="relu"))(attention)
    Batch_Normalization = BatchNormalization()(BiGRU_out)
    Drop_out = Dropout(0.1)(Batch_Normalization)
    attention = attention_function(Drop_out)
    Batch_Normalization = BatchNormalization()(attention)
    Drop_out = Dropout(0.1)(Batch_Normalization)
    Flatten_ = Flatten()(Drop_out)
    output=Dropout(0.1)(Flatten_)
```

圖 51　Dual Attention with CNNBiLSTM

```
        output=Dropout(0.1)(Flatten_)
        output = Dense(1, activation='sigmoid')(output)
        model = Model(inputs=[inputs], outputs=output)
        return model
m = attention_model()
m.summary()
m.compile(loss='mean_squared_error', optimizer='adam')
m.fit(trainX, trainY, epochs=33, batch_size=64,
verbose=0,validation_data=(testX, testY))
# make predictions
trainPredict = m.predict(trainX)
testPredict = m.predict(testX)
#FNormalize
trainPredict_FNormalizeMult= FNormalizeMult(trainPredict,tr_normalize)
testPredict_FNormalizeMult= FNormalizeMult(testPredict,te_normalize)
#calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY, trainPredict))
print('*    Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY, testPredict))
print('*    Test Score: %.2f RMSE' % (testScore))
teV=testVaild[:-(look_back+1),0]
trV=trainVaild[:-(look_back+1),0]
plt.plot(teV)
plt.plot(testPredict_FNormalizeMult,'r--')
plt.savefig('teV')
plt.show()
testScore = math.sqrt(mean_squared_error(teV, testPredict_FNormalizeMult))
print('*    FNormalizeMult Test Score: %.2f RMSE' % (testScore))
plot_model(m, show_shapes=True, to_file='DualAttention_with_CnnBiGRU.png')
plt.plot(trV)
plt.plot(trainPredict_FNormalizeMult,'r--')
plt.savefig('trV')
plt.show()
trainScore = math.sqrt(mean_squared_error(trV, trainPredict_FNormalizeMult))
print('*    FNormalizeMult Train Score: %.2f RMSE' % (trainScore))
```

圖 52　Dual Attention with CNNBiLSTM 程式碼 2

五、Attention with CNNBiLSTM

```
dataframe = read_csv('/content/CNN_BiLSTM_withAttention/OilwithGold.csv')
dataset = dataframe.values
dataset = dataset.astype('float32')
testVaild, trainVaild =dataset, dataset
# split into train and test sets
train_size = int(len(dataset) * 0.9)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
testVaild = testVaild[train_size:len(dataset),:]
trainVaild = trainVaild[:train_size,:]
# normalize the dataset
dataset,da_normalize = NormalizeMult(dataset)
train,tr_normalize = NormalizeMult(train)
test,te_normalize = NormalizeMult(test)
look_back, Dims = 2,2
TimeSteps=look_back
trainX, trY = create_dataset2(train, look_back)
testX, teY = create_dataset2(test, look_back)
trainY,testY =trY[:,0],teY[:,0]
def attention_model():
    inputs = Input(shape=(TimeSteps, Dims))
    x = Conv1D(filters = 128, kernel_size = 1, activation = 'relu')(inputs)
    lstm_out = Bidirectional(LSTM(64,
return_sequences=True,activation="relu"))(x)
    lstm_out = BatchNormalization()(lstm_out)
    lstm_out = Dropout(0.1)(lstm_out)
    attention = attention_function(lstm_out)
    attention = BatchNormalization()(attention)
    attention = Dropout(0.1)(attention)
    attention = Flatten()(attention)
    output=Dropout(0.1)(attention)
    output = Dense(1, activation='sigmoid')(output)
    model = Model(inputs=[inputs], outputs=output)
    return model
```

圖 53　Attention with CNNBiLSTM 程式碼

```
m = attention_model()
m.summary()
Adam=optimizers.Adam(lr=0.001)
m.compile(loss='mean_squared_error', optimizer=Adam)
m.fit(trainX, trainY, epochs=55, batch_size=72,
verbose=0,validation_data=(testX, testY))
# make predictions
trainPredict = m.predict(trainX)
testPredict = m.predict(testX)
#FNormalize
trainPredict_FNormalizeMult= FNormalizeMult(trainPredict,tr_normalize)
testPredict_FNormalizeMult= FNormalizeMult(testPredict,te_normalize)
#calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY, trainPredict))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY, testPredict))
print('Test Score: %.2f RMSE' % (testScore))
teV=testVaild[:-(look_back+1),0]
trV=trainVaild[:-(look_back+1),0]
print('*    look_back=',look_back)
trainScore = math.sqrt(mean_squared_error(trV, trainPredict_FNormalizeMult))
print('*    FNormalizeMult Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(teV, testPredict_FNormalizeMult))
print('*    FNormalizeMult Test Score: %.2f RMSE' % (testScore))
# shift test predictions for plotting
teV=testVaild[:,0]
plt.plot(teV)
plt.plot(testPredict_FNormalizeMult,'r')
plt.show()
plt.plot(trV)
plt.plot(trainPredict_FNormalizeMult,'r')
plt.show()
plot_model(m, show_shapes=True, to_file='CnnBilstm_with_Attention.png')
```

圖 54　Attention with CNNBiLSTM 程式碼 2

六、BiLSTM

```
dataframe = read_csv('/content/CNN_BiLSTM_withAttention/OilwithGold.csv')
dataset = dataframe.values
dataset = dataset.astype('float32')
testVaild, trainVaild =dataset, dataset
# split into train and test sets
train_size = int(len(dataset) * 0.9)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
testVaild = testVaild[train_size:len(dataset),:]
trainVaild = trainVaild[:train_size,:]
# normalize the dataset
dataset,da_normalize = NormalizeMult(dataset)
train,tr_normalize = NormalizeMult(train)
test,te_normalize = NormalizeMult(test)
look_back, Dims = 2,2
TimeSteps=look_back
trainX, trY = create_dataset2(train, look_back)
testX, teY = create_dataset2(test, look_back)
trainY,testY =trY[:,0],teY[:,0]
model = Sequential()
model.add(Bidirectional(LSTM(64,
input_shape=(look_back ,Dims),activation="relu")))
model.add(Dropout(0.1))
model.add(Dense(1 ,activation='sigmoid'))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=200 , batch_size=128,
verbose=0,validation_data=(testX, testY))
model.summary()
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
#FNormalize
trainPredict_FNormalizeMult= FNormalizeMult(trainPredict,tr_normalize)
testPredict_FNormalizeMult= FNormalizeMult(testPredict,te_normalize)
```

圖 55　BiLSTM 程式碼

```
#calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY, trainPredict))
print('*    Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY, testPredict))
print('*    Test Score: %.2f RMSE' % (testScore))
teV=testVaild[:-(look_back+1),0]
trV=trainVaild[:-(look_back+1),0]
trainScore = math.sqrt(mean_squared_error(trV, trainPredict_FNormalizeMult))
print('*    FNormalizeMult Train Score: %.2f RMSE' % (trainScore))
trainScoreMAE = mean_absolute_error(trV, trainPredict_FNormalizeMult)
print('*    Train Score: %.2f MAE' % (trainScoreMAE))
testScore = math.sqrt(mean_squared_error(teV, testPredict_FNormalizeMult))
print('*    FNormalizeMult Test Score: %.2f RMSE' % (testScore))
testScoreMAE = mean_absolute_error(teV, testPredict_FNormalizeMult)
print('*    Test Score: %.2f MAE' % (testScoreMAE))
# shift test predictions for plotting
plt.plot(teV)
plt.plot(testPredict_FNormalizeMult,'r')
plt.show()
plot_model(model, show_shapes=True, to_file='Bilstm.png')
plt.plot(trV)
plt.plot(trainPredict_FNormalizeMult,'r')
plt.show()
```

圖 56　BiLSTM 程式碼 2

七、TPA LSTM

```
import torch
from torch import nn
import torch.nn.functional as F
import argparse
from progressbar import *
from torch.optim import Adam
import util
import random
import matplotlib.pyplot as plt
from tqdm import tqdm
import numpy as np
import os
import pandas as pd
from datetime import date
# import tensorflow as tf
class TPALSTM(nn.Module):
    def __init__(self, input_size, output_horizon, hidden_size, obs_len,
n_layers):
        super(TPALSTM, self).__init__()
        self.hidden = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.lstm = nn.LSTM(hidden_size, hidden_size, n_layers, \
                    bias=True, batch_first=True) # output (batch_size,
obs_len, hidden_size)
        self.hidden_size = hidden_size
        self.filter_num = 32
        self.filter_size = 1
        self.output_horizon = output_horizon
        self.attention = TemporalPatternAttention(self.filter_size, \
            self.filter_num, obs_len-1, hidden_size)
        self.linear = nn.Linear(hidden_size, output_horizon)
        self.n_layers = n_layers
```

圖 57　TPA-LSTM 程式碼

```
def forward(self, x):
        batch_size, obs_len = x.size()
        x = x.view(batch_size, obs_len, 1)
        xconcat = self.relu(self.hidden(x))
        # x = xconcat[:, :obs_len, :]
        # xf = xconcat[:, obs_len:, :]
        H = torch.zeros(batch_size, obs_len-1, self.hidden_size)
        ht = torch.zeros(self.n_layers, batch_size, self.hidden_size)
        ct = ht.clone()
        for t in range(obs_len):
            xt = xconcat[:, t, :].view(batch_size, 1, -1)
            out, (ht, ct) = self.lstm(xt, (ht, ct))
            htt = ht.permute(1, 0, 2)
            htt = htt[:, -1, :]
            if t != obs_len - 1:
                H[:, t, :] = htt
        H = self.relu(H)

        # reshape hidden states H
        H = H.view(-1, 1, obs_len-1, self.hidden_size)
        new_ht = self.attention(H, htt)
        ypred = self.linear(new_ht)
        return ypred
```

圖 58　TPA-LSTM 程式碼 2

```
class TemporalPatternAttention(nn.Module):


    def __init__(self, filter_size, filter_num, attn_len, attn_size):
        super(TemporalPatternAttention, self).__init__()
        self.filter_size = filter_size
        self.filter_num = filter_num
        self.feat_size = attn_size - self.filter_size + 1
        self.conv = nn.Conv2d(1, filter_num, (attn_len, filter_size))
        self.linear1 = nn.Linear(attn_size, filter_num)
        self.linear2 = nn.Linear(attn_size + self.filter_num, attn_size)
        self.relu = nn.ReLU()


    def forward(self, H, ht):
        _, channels, _, attn_size = H.size()
        new_ht = ht.view(-1, 1, attn_size)
        w = self.linear1(new_ht) # batch_size, 1, filter_num
        conv_vecs = self.conv(H)
        conv_vecs = conv_vecs.view(-1, self.feat_size, self.filter_num)
        conv_vecs = self.relu(conv_vecs)
        # score function
        w = w.expand(-1, self.feat_size, self.filter_num)
        s = torch.mul(conv_vecs, w).sum(dim=2)
        alpha = torch.sigmoid(s)
        new_alpha = alpha.view(-1, self.feat_size, 1).expand(-1, self.feat_size,
self.filter_num)
        v = torch.mul(new_alpha, conv_vecs).sum(dim=1).view(-1,
self.filter_num)
        concat = torch.cat([ht, v], dim=1)
        new_ht = self.linear2(concat)
        return new_ht
```

圖 59　TPA-LSTM 程式碼 3

```
def train(
    X,
    y,
    args
    ):
    '''
    Args:
    - X (array like): shape (num_samples, num_features, num_periods)
    - y (array like): shape (num_samples, num_periods)
    - epoches (int): number of epoches to run
    - step_per_epoch (int): steps per epoch to run
    - seq_len (int): output horizon
    - likelihood (str): what type of likelihood to use, default is gaussian
    - num_skus_to_show (int): how many skus to show in test phase
    - num_results_to_sample (int): how many samples in test phase as prediction
    '''
    num_ts, num_periods, num_features = X.shape
    model = TPALSTM(1, args.seq_len,
        args.hidden_size, args.num_obs_to_train, args.n_layers)
    optimizer = Adam(model.parameters(), lr=args.lr)
    random.seed(2)
    # select sku with most top n quantities
    Xtr, ytr, Xte, yte = util.train_test_split(X, y)
    losses = []
    cnt = 0
```

圖 60　TPA-LSTM 程式碼 4

```
    yscaler = None
    if args.standard_scaler:
        yscaler = util.StandardScaler()
    elif args.log_scaler:
        yscaler = util.LogScaler()
    elif args.mean_scaler:
        yscaler = util.MeanScaler()
    elif args.max_scaler:
        yscaler = util.MaxScaler()
    if yscaler is not None:
        ytr = yscaler.fit_transform(ytr)
    # training
    seq_len = args.seq_len
    obs_len = args.num_obs_to_train
    progress = ProgressBar()
    for epoch in progress(range(args.num_epoches)):
        # print("Epoch {} starts...".format(epoch))
        for step in range(args.step_per_epoch):
            Xtrain, ytrain, Xf, yf = util.batch_generator(Xtr, ytr, obs_len, seq_len,
args.batch_size)
            Xtrain = torch.from_numpy(Xtrain).float()
            ytrain = torch.from_numpy(ytrain).float()
            Xf = torch.from_numpy(Xf).float()
            yf = torch.from_numpy(yf).float()
            ypred = model(ytrain)
            # loss = util.RSE(ypred, yf)
            loss = F.mse_loss(ypred, yf)
            losses.append(loss.item())
            optimizer.zero_grad()
            loss.backward()
            # torch.nn.utils.clip_grad_norm_(model.parameters(), 1)
            optimizer.step()
```

圖 61　TPA-LSTM 程式碼 5

```
# test
    mape_list = []
    # select skus with most top K
    X_test = Xte[:, -seq_len-obs_len:-seq_len, :].reshape((num_ts, -1,
num_features))
    Xf_test = Xte[:, -seq_len:, :].reshape((num_ts, -1, num_features))
    y_test = yte[:, -seq_len-obs_len:-seq_len].reshape((num_ts, -1))
    yf_test = yte[:, -seq_len:].reshape((num_ts, -1))
    yscaler = None
    if args.standard_scaler:
        yscaler = util.StandardScaler()
    elif args.log_scaler:
        yscaler = util.LogScaler()
    elif args.mean_scaler:
        yscaler = util.MeanScaler()
    elif args.max_scaler:
        yscaler = util.MaxScaler()
    if yscaler is not None:
        ytr = yscaler.fit_transform(ytr)
    if yscaler is not None:
        y_test = yscaler.fit_transform(y_test)
    X_test = torch.from_numpy(X_test).float()
    y_test = torch.from_numpy(y_test).float()
    Xf_test = torch.from_numpy(Xf_test).float()
    ypred = model(y_test)
    ypred = ypred.data.numpy()
    if yscaler is not None:
        ypred = yscaler.inverse_transform(ypred)
    ypred = ypred.ravel()
```

圖 62　TPA-LSTM 程式碼 6

```
    loss = np.sqrt(np.sum(np.square(yf_test - ypred)))
    print("losses: ", loss)

    if args.show_plot:
        plt.figure(1, figsize=(20, 5))
        plt.plot([k + seq_len + obs_len - seq_len \
            for k in range(seq_len)], ypred, "r-")
        plt.title('Prediction uncertainty')
        yplot = yte[-1, -seq_len-obs_len:]
        plt.plot(range(len(yplot)), yplot, "k-")
        plt.legend(["prediction", "true", "P10-P90 quantile"], loc="upper left")
        ymin, ymax = plt.ylim()
        plt.vlines(seq_len + obs_len - seq_len, ymin, ymax, color="blue",
linestyles="dashed", linewidth=2)
        plt.ylim(ymin, ymax)
        plt.xlabel("Periods")
        plt.ylabel("Y")
        plt.savefig("TPA_train.jpg")
        plt.show()
    return losses, mape_list
```

圖 63　TPA-LSTM 程式碼 7

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--num_epoches", "-e", type=int, default=1000)
    parser.add_argument("--step_per_epoch", "-spe", type=int, default=2)
    parser.add_argument("-lr", type=float, default=1e-3)
    parser.add_argument("--n_layers", "-nl", type=int, default=3)
    parser.add_argument("--hidden_size", "-hs", type=int, default=24)
    parser.add_argument("--seq_len", "-sl", type=int, default=7)
    parser.add_argument("--num_obs_to_train", "-not", type=int, default=1)
    parser.add_argument("--num_results_to_sample", "-nrs", type=int,
default=10)
    parser.add_argument("--show_plot", "-sp", action="store_true")
    parser.add_argument("--run_test", "-rt", action="store_true")
    parser.add_argument("--standard_scaler", "-ss", action="store_true")
    parser.add_argument("--log_scaler", "-ls", action="store_true")
    parser.add_argument("--mean_scaler", "-ms", action="store_true")
    parser.add_argument("--max_scaler", "-max", action="store_true")
    parser.add_argument("--batch_size", "-b", type=int, default=64)
    # parser.add_argument("--sample_size", type=int, default=100)
    args = parser.parse_args()
```

圖 64　TPA-LSTM 程式碼 8

```
if args.run_test:


        data = pd.read_csv("LD_MT200_hour2.csv", parse_dates=["date"])
        data["year"] = data["date"].apply(lambda x: x.year)
        data["day_of_week"] = data["date"].apply(lambda x: x.dayofweek)
        # data = data.loc[(data["date"] >= date(2014, 1, 1)) & (data["date"] <=
date(2014, 3, 1))]
        features = ["hour", "day_of_week"]
        # hours = pd.get_dummies(data["hour"])
        # dows = pd.get_dummies(data["day_of_week"])
        hours = data["hour"]
        dows = data["day_of_week"]
        X = np.c_[np.asarray(hours), np.asarray(dows)]
        num_features = X.shape[1]
        num_periods = len(data)
        X = np.asarray(X).reshape((-1, num_periods, num_features))
        y = np.asarray(data["MT_200"]).reshape((-1, num_periods))
        # X = np.tile(X, (10, 1, 1))
        # y = np.tile(y, (10, 1))
        losses, mape_list = train(X, y, args)
```

圖 65　TPA-LSTM 程式碼 9

八、ARIMA

```
from pandas import read_csv
from pandas import datetime
from pandas import DataFrame
from statsmodels.tsa.arima_model import ARIMA
from matplotlib import pyplot
series = read_csv('/content/CNN_BiLSTM_withAttention/OilwithGold.csv',
header=0, parse_dates=[0], index_col=0, squeeze=True)
series = series.drop(['gold_Close'], axis = 1)
print(series.head())
series.plot()
pyplot.show()
model = ARIMA(series, order=(5,1,0))
model_fit = model.fit(disp=0)
print(model_fit.summary())
# plot residual errors
residuals = DataFrame(model_fit.resid)
residuals.plot()
pyplot.show()
residuals.plot(kind='kde')
pyplot.show()
print(residuals.describe())             if args.show_plot:
                    plt.plot(range(len(losses)), losses, "k-")
                    plt.xlabel("Period")
                    plt.ylabel("Loss")
                    plt.savefig("TPA.jpg")
                    plt.show()
```

圖 66　ARIMA 程式碼

九、Prophet

```
import pandas as pd
import numpy as np
import datetime
from fbprophet import Prophet
from sklearn.metrics import mean_absolute_error
#to plot within notebook
import matplotlib.pyplot as plt
#setting figure size
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 20,10
#datetime
starttime = datetime.datetime.now()
#for normalizing data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
#read the file
df = pd.read_csv('/content/CNN_BiLSTM_withAttention/OilwithGold.csv')
data=df
new_data = pd.DataFrame(index=range(0,len(df)),columns=['Date', 'oil_Close'])
for i in range(0,len(data)):
      new_data['Date'][i] = data['Date'][i]
      new_data['oil_Close'][i] = data['oil_Close'][i]
new_data.rename(columns={'oil_Close': 'y', 'Date': 'ds'}, inplace=True)
new_data['ds'] = pd.to_datetime(new_data['ds'], format='%Y-%m-%d')
#new_data.index = new_data['ds']
trainA =int(len (new_data) * 0.7)
validA    =len(new_data) - trainA
train = new_data[:trainA]
valid =    new_data[trainA:]
```

圖 67　Prophet 程式碼

```
#fit the model
model = Prophet(changepoint_prior_scale=0.02)
model.fit(new_data)
#predictions
Train_close_prices = model.make_future_dataframe(periods=validA,freq = 'D',
include_history = False)
forecast = model.predict(Train_close_prices)
#print(forecast['yhat'],valid)
test_valid = valid
test_valid.sort_values(by=['ds'])
test_valid.reset_index(inplace=True)
#rmse
forecast_valid = forecast['yhat'].round(decimals=1)
rms=np.sqrt(np.mean(np.power((np.array(test_valid['y'])-
np.array(forecast_valid)),2)))
mae=mean_absolute_error(test_valid['y'],forecast_valid)
rms.round(decimals=2)
print("RMSE: [ ", rms , " ]")
print("MAE: [ ", mae , " ]")
endtime = datetime.datetime.now()
print(endtime - starttime)
#plt
plt.figure(figsize=(8, 5))
plt.plot(test_valid['y'],label='test')
plt.plot(forecast_valid, label='forecast')
plt.legend(loc='upper right')
plt.show()
```

圖 68　Prophet 程式碼 2