# PyTorch 與機器學習期末報告

# Forecasting WTI Prices Using RNN, BiRNN, LSTM, BiLSTM, GRU model

金融五 **106208096** 蔡易霖

## 1. Abstract

In this report, we compare the performance of different **RNN-based models (simple RNN, LSTM, GRU, Bidirectional RNN, Bidirectional LSTM)** to predict WTI oil price, and we collected many features including Demand & Supply, Inventory, Production capacity, Speculation, Financial Market aspects that relate a lot with oil price.
We found that simple RNN model perform the best among these models, which is aligned with the results of **Kohei et Shigeyuki (2020)**.

## 2. Introduction

Crude oil price can be reckoned as a significant indicator of macroeconomic environment, given that oil is the material of plenty of goods. In this report, we want to forecast WTI crude oil price in order to better understand future macro condition considering the predicted WTI price trails. We take advantaging of oil price sequential properties through using RNN-based models and compare the performance among them.

## 3. Model

### ➢ Features Selection

Based on our macro understanding for WTI oil prices, it can be separated into five aspects — Demand & Supply, Inventory, Production capacity, Speculation, and Financial Market. Noted that all data is obtained from Bloomberg Terminal and frequency is weekly data.

| Classification | Features | Mandarin Name | Explanation |
|---|---|---|---|
| **Target** | WTI | WTI | Dollars per barrel |
| **Demand & Supply** | US oil demand | 美國原油需求 | |
| | US oil supply | 美國原油供給 | |
| | US net export and import | 美國進出口 | |
| **Inventory** | Weekly inventory | 原油每週庫存 | |
| | Cushing stock | 庫欣區庫存 | |
| | Gasoline inventory variety | 美國汽油庫存總變動 | |
| **Production capacity** | Oil Well Drilling | 鑽油井數 | |
| **Speculation** | CFTC Crude Oil speculative net positions | 投機者期貨部位 | |
| **Financial Market** | S&P stock price | S&P | US Stock price |
| | US Treasury 10Y | 美國十年期公債 | US 10-year treasury rate can be viewed as cost of loan |

| | | | |
|---|---|---|---|
| | US 2Y10Y rate spread | 美國長短天期利差 | US 10Y-2Y yield spread can be an indicator as macro environment |
| | Dollars index (UXY) | 美國指數 | Highly connected to oil price |
| | Canadian dollar (CAD) | 加幣 | |
| | Ruble (RUB) | 盧比 | |
| | CRB index | 商品期貨價格指數 | Constructed by several commodity price into an index |
| | Stock volatility (VIX) | 股市波動度 | |

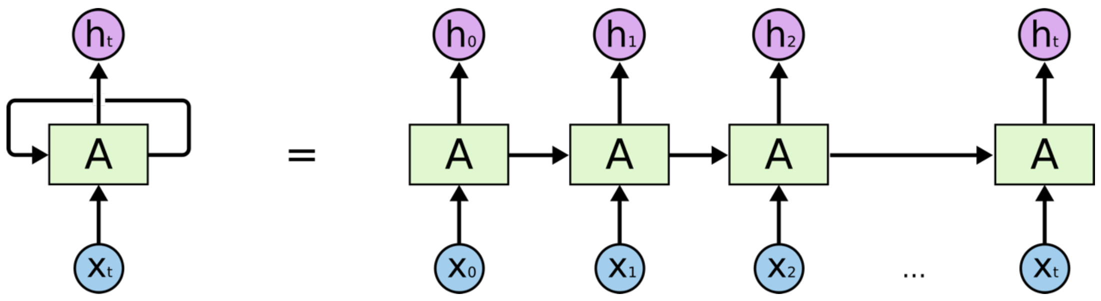
➢ **Models Explanation (5 models)**

♦ **Sequence Model**

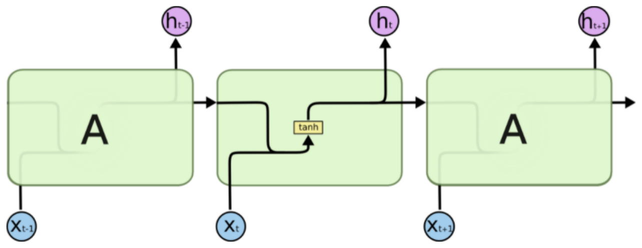
Considering when the output and/or the input data is sequential data, such as speech recognition, sentiment classification, and DNA sequence analysis. All the examples denoted above are under the regime of “supervised learning with label data X”.

♦ **RNN (Recurrent Neutral Network)**

1) RNN solves the problems of neural network that it cannot pass the previous information into the next time term. We could say that RNN has loops in them, allowing information to persist.
Noted the graph below: The left hand side is “a rolled version”, **A** stands for several time steps of information condensed into a small box named **A**, and the right hand side is a “unrolled version.”



An unrolled Recurrent Neural Network



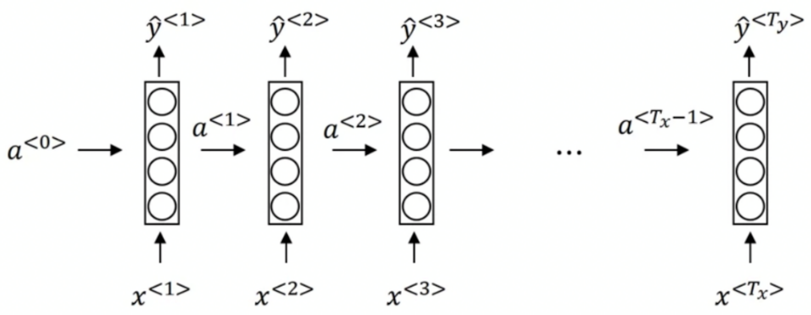
The repeating module in a standard RNN contains a single layer

2) Forward Propagation: Under RNN network, the information is passed forward, namely from the left hand side to the right one showed on the graph above. Noted that **Bidirectional RNN** includes forward and back propagation, allowing the mode forecast to take information from past and future into account.

3) Parameters inside RNN network:

- Hidden Layer → a(1) = g[**W***a(0) + **U***x(0) + bias_a]  ## activation function: tanh ; ReLu
- Output Layer → y(1) = g[**V***a(1) + bias_y]  ## activation function: sigmoid

Three important parameters are: W, U, V

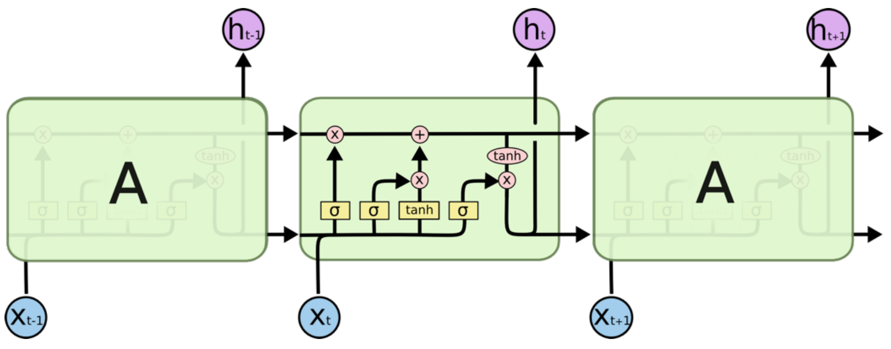


Picture from Coursera – Sequence Models

4) Vanishing gradients with RNNs:
The basic RNN model has many local influences, meaning the output y is mainly influenced by the closest three values. To put it another way, the output $y_t$ is hardly impacted by the information early in the sequences, so we can solve this problem by using the LSTM model.

♦ **LSTM (Long Short Term Memory)**

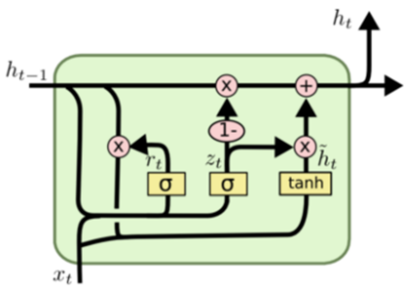
A modification of RNN hidden layer, capturing long-ranged connections and **helping vanishing gradients problem.** Below, we explain the structure of LSTM:
1) **Cell state** flowing throughout whole model: it is for information running through the whole model without changing it.
2) **3 gates** inside hidden layer: **controlling** information to be **deleted or added to the cell gate**, and then to predict the output value.



The Repeating module in an LSTM contains four interesting layer

♦ **GRU (Gated Recurrent Units)**

GRU is a little bit like LSTM model, but it performs faster than LSTM. In LSTM, we have input gate, forget gate and output gate. In GRU, it uses “update gate” to replace the input gate and forget gate, and combine the cell state and ht. From these steps, we can train the model efficiently.



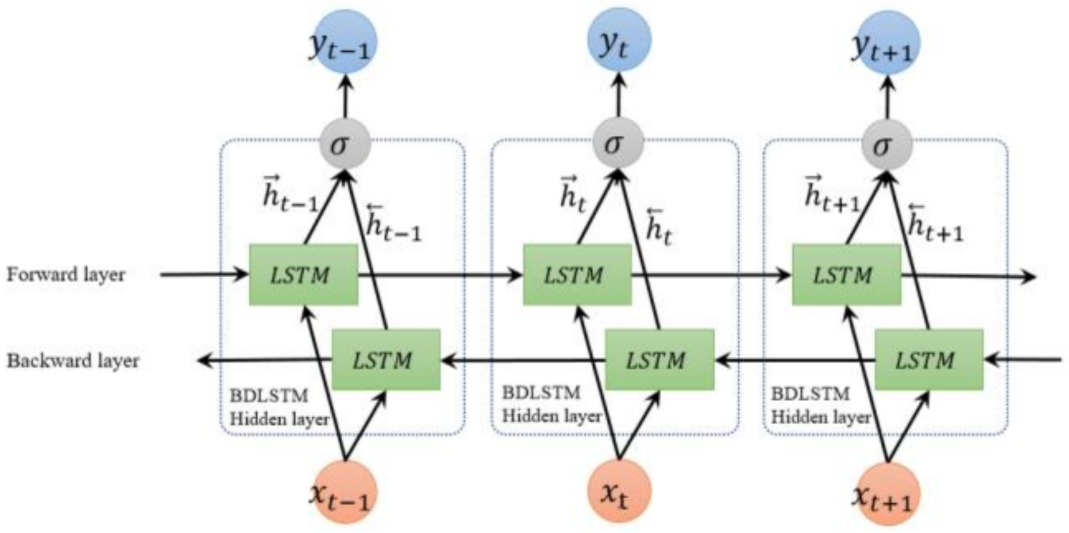
GRU structure is different from LSTM

♦ **BiRNN/BiLSTM (Bidirectional RNN/Bidirectional LSTM)**

Bidirectional RNN is the combination of two unidirectional RNN with forward and backward states.

Bidirectional LSTM is a sequence processing model that consists of two LSTMs, the same as BiRNN, one taking the input in a forward direction, and the other in a backwards direction.

The two models can consider not only before data, also the latter data. Let the prediction be more precise, if the target data also has more related to future information.



➢ **Framework**

**Step 1. Import data**

Before setting our RNN, LSTM and GRU model, also including bidirectional RNN and bidirectional LSTM model. **First**, we needed to **do data collection**. Data was pulled from available sources, we collected our data from Bloomberg. Our target data is WTI (West Texas Intermediate) oil price, and we have sixteen independent variables from different aspects cross Demand & Supply, Inventory, Production capacity, Speculation, and Financial Market. In addition, we dropped 'Dow Jones' and 'NASDAQ' variables from origin dataset, the two variables are higher related to S&P index. But S&P index is more widely related to all industries in US. Then we imported this data into our python file with **Pandas package**, and used **Numpy** to do **data processing.**

**Step 2. Data processing**

**Second**, we did the **data preparation,** that is often referred to as **"pre-processing"** is the stage at which raw data is cleaned up and organized for the following stage of data processing. The purpose of this step is to eliminate redundant, incomplete, or incorrect data, let our prediction become more precise. We checked whether **outliers or extreme values** are including in our variables by dividing every variable into four parts by percentile. We calculate interquartile range by using 75th quantile's data minus 25th quantile. If the data is lower than 25th quantile's data minus sample multiply interquartile range or higher than 75th quantile's data plus sample multiply interquartile range, we need to drop these data. We use for loop in Python to check every variable's data, and it seems like our data don't any outliers or extreme values.

And then we needed to check whether the sixteen variables have **Multi-Collinearity problem.** Multi-Collinearity means is the occurrence of high inter-correlations among two or more independent variables in a multiple regression model. Multi-Collinearity can lead to skewed or misleading results when a researcher or analyst attempts to determine how well each independent variable can be used most effectively to predict or understand the dependent variable in a statistical model. Maybe this variable is positively affect the target data, but its result is negative. That is not reasonable, and the problem is very serious. So we need to calculate the Correlation Coefficients between these independent variables to deal with Multi-Collinearity problem. We let the variables become DataFrame type in Python, and used the Python package to build the correlation coefficient matrix. And then, we found some variables' correlation coefficients are too higher. However, if we dropped the variables with high correlative with other variables, our result will not be more explanatory, R square is lower. In this report, we mainly predicted the target price, so ignore the Multi-Collinearity problem, we can get more accurate model.

**Third**, we wanted to import our data into the machine learning models, so we needed to **let data become suitable type.** So, we divide our data into 80% training data and 20% testing data. Training data are for us to put into machine learning models to train a predicting model, and testing data are used to see whether this model we trained is good at predicting the correct WTI oil price. Training data are from 2005 May to 2018 August, testing data are from 2018 August to 2021 December. We both have dependent and independent variables training and testing data.

Different kind of variables have different data range, such as Demand & Supply variables are very different from Financial Market. Fourth, we will perform min/max scaling on the dataset which **normalizes the data** within a certain range of minimum and maximum values. We will be using the MinMaxScaler class from the sklearn.preprocessing module to scale our data. This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one. This transformation is often used as an alternative to zero mean, unit variance scaling. Our data were been scaled to X_scaled by below formula, X is original data:

**X_std = (X - X.min) / (X.max - X.min)**

**X_scaled = X_std * (X.max - X.min) + X.min**

Finally, Because we will use Pytorch to train our model, we need to transform our data to Tensor type by torch.Tensor function, and use **TensorDataset** to combine our target and independent variables. Typically, time series regression tutorials lessons show how to create features by extracting parts of the timestamps or by lagging features, that is, using past values of each feature as features in their own right. Actually, our data have not only one feature, also not that kind dataset, but we still can use some way to let the data more convenient to iterate in machine learning model. One elegant way to do this is to create a **PyTorch Dataset** class, it represents a Python **iterable over a dataset**, with support for map-style and iterable-style datasets, customizing data loading order, automatic batching, single- and multi-process data loading, automatic memory pinning. And we can set the batch size, if we set batch size is 128, each batch will have 128 rows including target data and features when we iterate this Dataset. If we want to train 866 rows' data, it takes almost 7 iterations to complete an epoch.

After finishing our data processing, we can build the machine learning models and input the data.

## ♦ Step3. RNN, LSTM, GRU, BiRNN and BiLSTM's architecture

We need to create **"Class"** to build our models for us to select which model we want to use. Classes can let an object has **attributes and methods**. The methods like functions but they belong to a particular class. The attributes can be considered as an interface to interact with a class.

### (1) RNN, BiRNN：

First, we need to define the number of **RNN** layers and the nodes in each RNN layer. And then set the RNN layer including some index, such as the number of RNN layers, the number of nodes in each RNN layer and dropout rate etc. The last is fully connected layer convert the output into desired output shape.

We also need to define the **forward propagation** function as a class method. This method is executed sequentially, passing the inputs and the zero-initialized hidden state into this model. Detaching Reshaping the outputs in the shape of fitting into the fully connected layer. Finally converting the final state to our desired output shape.

Talk about the **BiRNN model**, this model just like RNN model. A little different is when set the BiRNN layer, we need to put an index “bidirectional” to let the RNN model convert to BiRNN model. And in final fully connected layer, the number of nodes in each layer need to be twice than RNN model, because we put the forward and backward data into this model.

**(2) LSTM, BiLSTM：**

**LSTM model** just likes RNN model, but the standard version has a little bit different. In addition to the hidden state, LSTMs have the cell state, which works like a conveyor belt that carries the relevant information from the earlier steps to later steps. The new information is added to or removed from the cell state through input and forget gates. So we also need to initialize cell state for first input with zeros.

**BiLSTM model is** based on LSTM model, we need to put an index “bidirectional” to let the LSTM model convert to BiLSTM model, and the number of nodes in each layer need to be twice than standard model.

**(3) GRU：**

Like LSTMs, GRU also capture long-term dependencies, but they do so by using reset and update gates without any cell state. In GRU model, we just need to initialize hidden state for first input with zeros.

Except for setting these models, we need to have a model class, a loss function to calculate the losses, and an optimizer to update the network's weights. In training process, we set it will print the results of training regularly.

- **Step4. Evaluation**

Above we talk about calculate the loss to evaluate accuracy of different models. In Pytorch, there are many methods to calculate the loss.

(1) L1

Loss function L1 is for regression model, also call mean absolute error(MAE). It calculates the loss between each element in the input x and target y. Below is the formula of MAE.

$$l_n = |x_n - y_n|$$

If we set its reduction is mean, this function will return the average loss.

(2) MSE

Loss function MSE is for regression model, that measures the mean squared error (squared L2 norm) between each element in the input x and target y. Below is the formula of MSE.

$$l_n = (x_n - y_n)^2$$

(3) CrossEntropyLoss

This loss function is mainly for classification model, so we didn't use it for our report. This is particularly useful when you have an unbalanced training set.

$$l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \cdot 1\{y_n \neq \text{ignore\_index}\}$$

## 4. Results – RNN Model is the best

### ♦ Accuracy Comparison

We tune our model through **1) increase epoch; 2) increase batch size; 3) change layer dimension; 4) change hidden layer.** We then change the models one after another. For now, we find that RNN model shows more accuracy (the best once can reach 97%) than others. Following that, BiRNN can also reach 80%~90% accuracy on condition that we lower the hidden layer into that of half of RNN best results, which makes sense for its bi-direction traits. But still, RNN performs better on the whole.

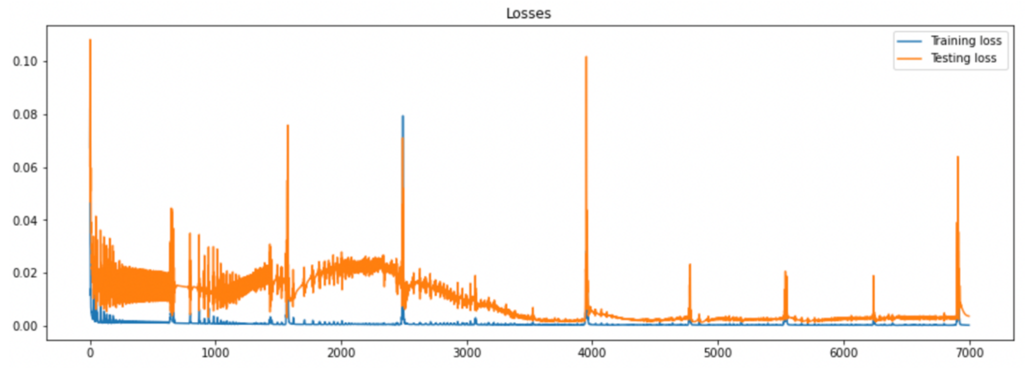
Noted that: We left detailed results at the end of programme file; there, we only show the best result of each model. And our default setting of learning rate is $10^{-3}$ (1e-3); weight decay is $10^{-6}$ (1e-6).

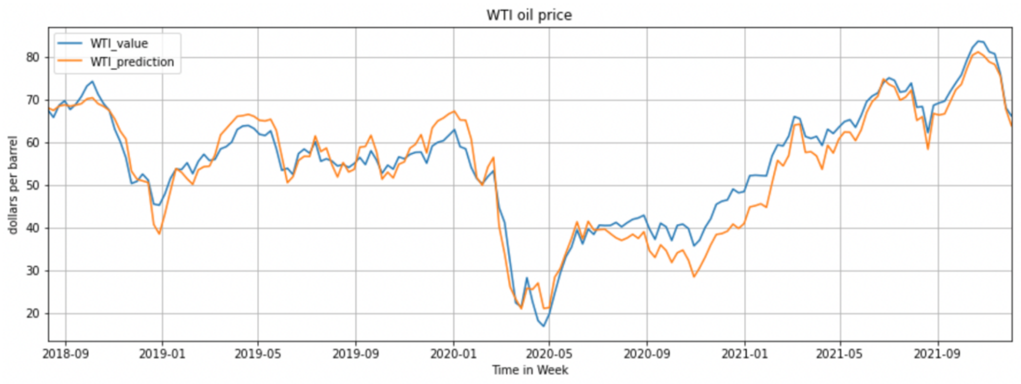
| Model | Hidden Layer | Layer Dimension | Batch size | Epoch | RMSE | $R^2$ |
|---|---|---|---|---|---|---|
| **(1) RNN** | 128 | 5 | 128 | 5500 | 2.52 | 97 % |
| **(2) LSTM** | 64 | 5 | 128 | 4500 | 7.32 | 71 % |
| **(3) GRU** | 64 | 3 | 128 | 5000 | 6.33 | 78 % |
| **(4) BiRNN** | 64 | 5 | 128 | 5000 | 3.76 | 92 % |
| **(5) BiLSTM** | 64 | 5 | 128 | 4500 | 7.32 | 71 |

### ♦ Delve into RNN Model – Changing different loss function

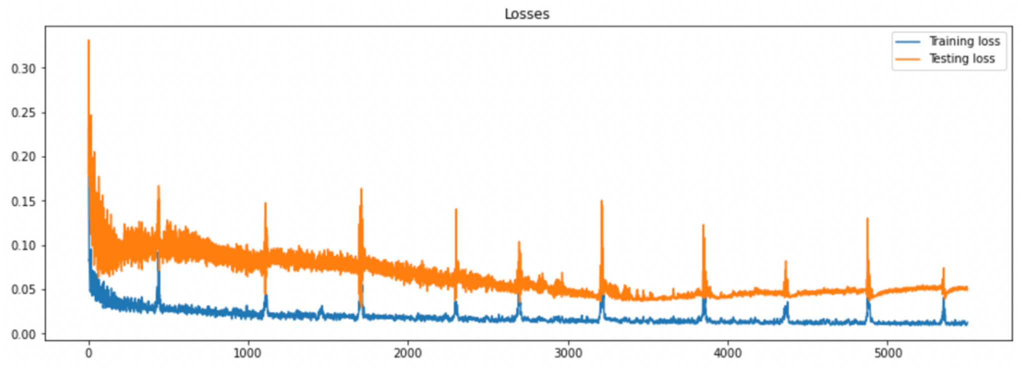
We can see the graph below that MSELoss function converges better than L1Loss; though the random spikes under MSELoss is bigger than that of L1loss. Still, we think MSELoss is better since it performs lower losses on the whole.

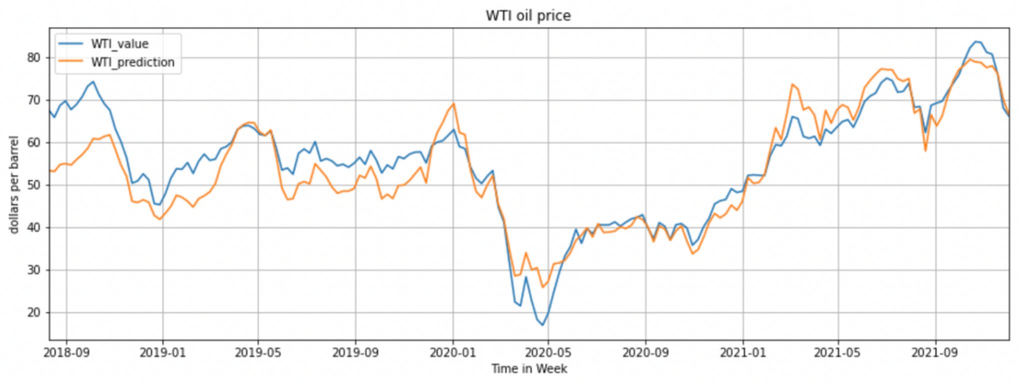
(1) Loss Curves under Loss Function of **MSELoss**





(2) Loss Curves under Loss Function of **L1Loss**

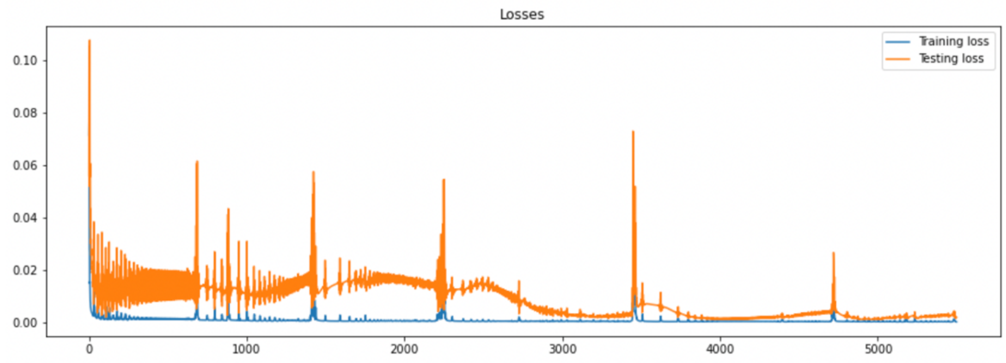




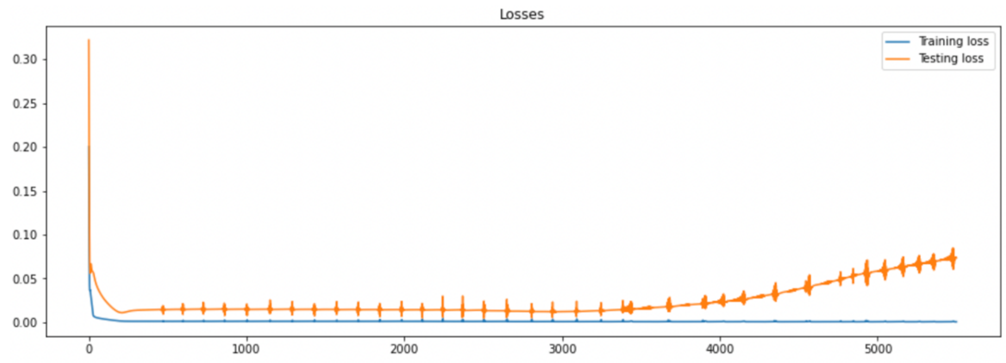
- **Delve into RNN Model – Possible reasons for Spikes inside the loss curve**

(3) Learning rate is too big – this large learning rate may take from one side to the other in the hypothetical valley while descending down to the global minimum "Best case scenario". Hence, we try to decrease our learning rate from 1e-3 into 1e-4 and 1e-5. However, 1e-3 still gets better accuracy, though spikes gets smaller and loss curve gets smoother under 1e-4 and 1e-5.

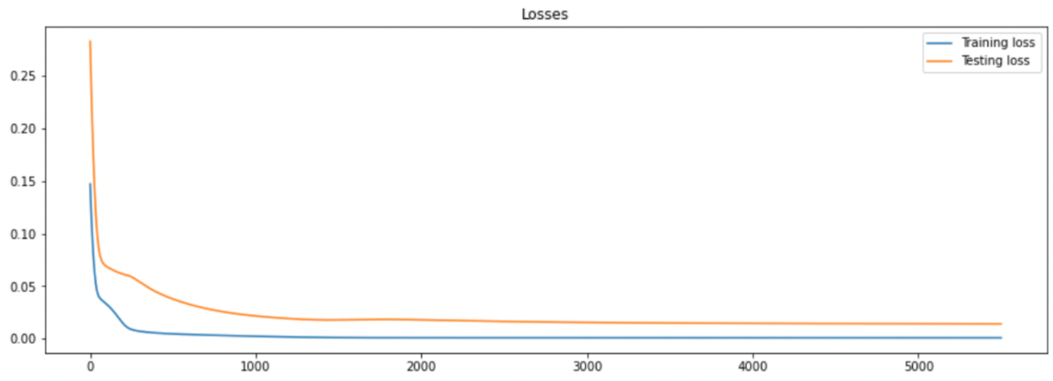
- Learning rate = 1e-3 (R2 = 0.94)



- Learning rate = 1e-4 (spikes get smaller and loss curve stabler; though loss is not converging)



- Learning rate = 1e-5 (spikes get smaller and loss curve stabler; though accuracy (0.66) is not better than the one with 1e-3)



(4) Some data points are noisy under our dataset – since financial asset price is not only just affected by hard data (e.g. demand/supply amount), but it is also influenced by market sentiment and other significant affairs or information.

## 5. Conclusion

In this report, we establish 5 deep learning sequential models (RNN; LSTM; GRU; BiRNN; BiLSTM) to predict WTI crude oil price. We found that RNN model performs the best, which $R^2$ once reaches 97% accuracy. Hence, we conduct some discussion under the model of RNN. We change the loss function model from MSELoss into L1Loss to see whether the loss curve can converge better, which turns out that MSELoss is still the proper one. Besides, we discuss the potential reasons for the emergence of spikes within the loss curve. Large learning rate and noisy financial data in essence are plausible reasons, but still, we choose to ignore spikes but rather side with the one with higher accuracy. In sum, in this report, we compare with different sequential deep learning models and find that RNN performs the best. For future work, we can also add some econometric models, such as ARMA or VAR model, to contrast with the neural network models.

## 6. Reference

♦ Matsuoka, K., & Hamori, S. (2021). Forecasting WTI Futures Prices Using Recurrent Neural Networks. Review of Integrative Business and Economics Research, 10(1), 34-50.

♦ Pyotorch loss function document.
(https://pytorch.org/docs/stable/generated/torch.nn.L1Loss.html#torch.nn.L1Loss)

♦ How to use DataLoader.
(https://blog.csdn.net/zkq_1986/article/details/85249220)

♦ LSTM model documentation

(https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

https://clay-atlas.com/blog/2020/05/12/pytorch-lstm-%E7%9A%84%E5%8E%9F%E7%90%86%E8%88%87%E8%BC%B8%E5%85%A5%E8%BC%B8%E5%87%BA%E6%A0%BC%E5%BC%8F%E7%B4%80%E9%8C%84/

https://stackabuse.com/time-series-prediction-using-lstm-with-pytorch-in-python/

https://www.crosstab.io/articles/time-series-pytorch-lstm)

♦ LSTM & GRU model

(https://towardsdatascience.com/building-rnn-lstm-and-gru-for-time-series-using-pytorch-a46e5b094e7b )