Attention LSTM : Application in Stock Price Prediction on Single Companies

Mingshi Wang

Abstract

According to the development of data-related techniques, aimed at exploring the largest value of data, price prediction has been seen as more vital for quantifying and pricing stock. To solve this problem, the learning based algorithm became popular during modern computing techniques development. LSTM (Long Short-Term Memory) techniques attract most within the new techniques for their interesting architecture and excellent performance. However, this leads to a new idea that maybe it can achieve better performance and also avoid recurrent message-passing steps with Attention LSTM, a multi-head Attention model. The main topic of this paper is the performance of Attention LSTM and LSTM models in the stock price prediction of single companies. To evaluate the performance of Attention LSTM in stock price prediction, the researcher designed several tests with different input lengths and two ways of splitting data: sliding window and feed-forward input. Based on the comparison, the result could be seen clearly that Attention LSTM could have better performance than LSTM with suitable test procedure 'sliding window'. LSTM could performed better with 'feed-forward input' approach. However, Attention LSTM could still run faster in this approach so it could help in tasks with large sequence length required high speed.

1 Introduction

Time series have been playing an important role in the financial world for a long time for its wide ranging applications, e.g. price prediction, and thus it is of great importance for things like portfolio management, trading of stock and bonds, etc.

In recent years, learning-based methods, like recurrent neural network(RNN)[1], gated recurrent unit(GRU)[2] and other variants, have demonstrated their power in time series analysis. Two most attractive frameworks were noticed, namely Long Short Term Memory (LSTM)[3] and Attention mechanism[4], which have great potential in analyzing time series. LSTM is widely used for time-series prediction based on its good performance. However, it has to go over all previous time steps to predict the next one based on its underlying logic. And thus, LSTM's application to longer time series will be restricted from its inability to run concurrently.

Differently, it is noticed that Multi-headed self-attention layers can also be seen as a variant of LSTM under some assumptions, while it can run all time steps concurrently. Attention mechanism's application in time series will be useful in tight time budget circumstances. And thus, it might be a more interesting and powerful architecture if these two could be compared.

Attention LSTM should have the ability to predict time series in various domains, including the financial world. It ignores forget gates but respects causality by multiplying the weights with the lower triangular matrix, details were explained in Section 2.

And thus, these were summarized into a research question: If Attention LSTM could perform better than the original LSTM model in stock price prediction or not. To test the new network architecture: Attention LSTM (ALSTM) on stock price prediction, the research questions were listed below to be more specific.

- Given the price series of a specific stock with all features(including open, high, low, traded volume), can the network predict the close price of this stock with a chosen input time length?
- How does the prediction quality change due to the adjustment of the chosen input time length?

In advance, the report is structured as follows: Section 2 introduces research methodology. Then, the setting of the data splitting method: sliding window and feed-forward input, and also experiments procedure comparing Attention LSTM and the original LSTM's performance were explained in Section 3. Details about related results would be organized in Section 4, which contains the comparison of these two models with different input time length: one day, 5 days, and 10 days, and also two methods: sliding window and feed-forward input. Responsible Research would be discussed in Section 5. Conclusions and future work were summarized in Section 6.

2 Methodology

The main idea of the Attention LSTM architecture is indicated as lemma shown below, with 'H' stands for Hidden state, 'W' stands for Weights and 'S' stands for Similarity.

$$H = \sum_{k=1}^{m'} W^k \sigma_C(X\theta^k), \text{ with } W^k \quad = S^k \oslash (S^k 11^T) \text{ and } S^k = exp(\frac{1}{\sqrt{d}} XQ^k K^k X^T) \odot L$$

Instead of employing forget gates, Attention LSTM solely determined the feature-wise weights by the input and output gates. As the inner product's dimension grows, there could be a chance that it will outperform LSTM in terms of performance between time steps. By multiplying the weights with the bottom triangular matrix, the outcome is Attention LSTM (ALSTM), which ignores forget gates but respects causality.

More preparation procedures were required for the prior logic's implementation, such as exploring the appropriate data-set. This project will use a shared data collection from 'Kaggle'[1] that contains companies' stock prices from 2006 to 2018. One data point at each timestamp from a company contains 5 features: open price, highest price, lowest price, closing price, and volume for a single day.

Prior to the Attention LSTM model's implementation, the researcher would conduct a feasibility investigation. The goal of the feasibility analysis is to check if the data set is appropriate for learning based algorithms in stock price prediction, as well as if the learning based algorithms can solve stock price prediction problems in general. As a result, the stock price was predicted using the original LSTM architecture. Because the Attention LSTM is presumed to perform better, if it worked for the original LSTM, it should also work for Attention LSTM. The outcome is positive due to the predicted price from LSTM is pretty close to the actual value and has a low loss value, which is the mean square error computed from the predicted price and actual price. This might be utilized as the foundation testing for the research issue established in Section 3.

All of the preceding processes have been achieved, and the new architecture Attention LSTM has been implemented using the formula previously given, which offers the probability for tests

[1]https://www.kaggle.com/szrlee/stock-time-series-20050101-to-20171231

comparing the performance of Attention LSTM and original LSTM by mean square error. As a result, the research questions were satisfactorily answered as Attention LSTM could perform better in Section 4.

3 Experiments

Data Set Split

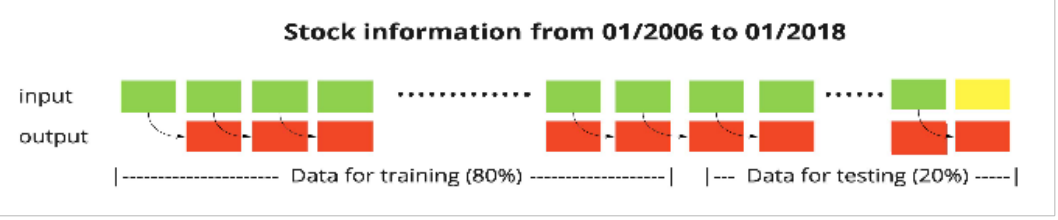

Figure 1: One day as input for prediction

As introduced previously in Section 2, the experimental data set contains company stock prices from 2006 to 2018. The whole data set of one single company's stock price was split into two parts: 80% as training data set and the rest 20% as testing data set as shown in Figure 1.

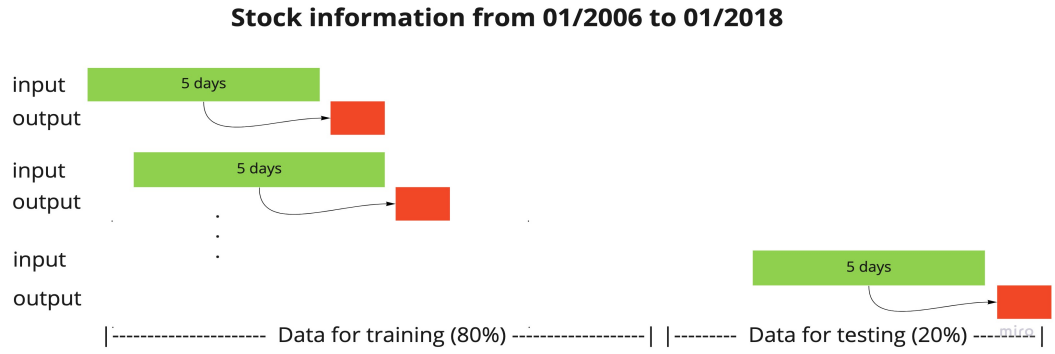

Figure 2: Five day as input for prediction

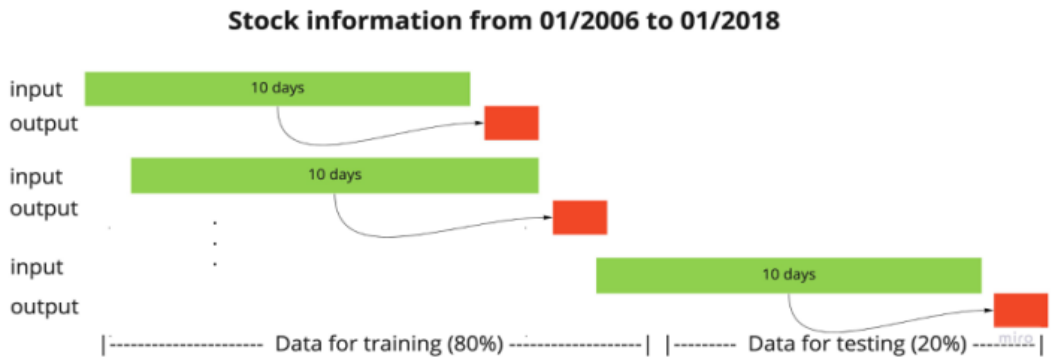

Figure 3: Ten day as input for prediction

Moreover, the input period length is set to one day or ten days and keep rolling as shown in Figure 1, Figure 2 and Figure 3, with green stands for input and red stands for expected output. To be more specific, as shown in Figure 1, if the input period length is one day, then the first day's data, with five features, would be imported as input, and the close price for the second day is the expected output. And then, the original, real second day's data would also be used as input to predict the close price of the third day. The date will keep moving until it achieves the last day in the training or testing data set.

This approach was called 'Sliding Window' in this paper due to the way that data is split looks like a window sliding from left to right. The reason to get various input length of 'sliding window' way in splitting data is that the first way shown in Figure 1 is known as the standard method to test a model's performance in the real quantitative industry, however, this is not an exact time series prediction. Details will be discussed in Section 4. And thus, input with 5 days and 10 days, as shown in Figure 2 and 3 separately, were settled to compare the performance difference of time series with different input length. In advance, using feed-forward days as input, as illustrated in Figure 4, could be more interesting in comparison.

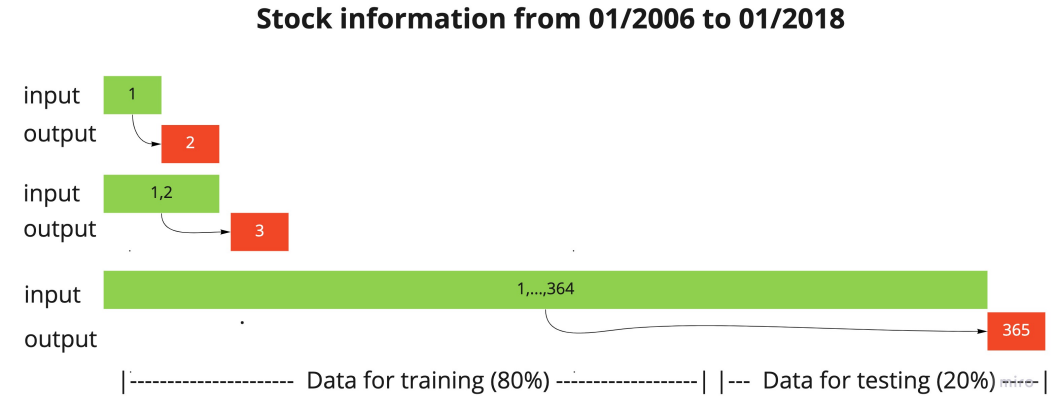

Figure 4: feed-forward input for prediction

To be more specific, the model would get the first day's data as input and expect to predict the second day's close price. And, in a compounding way, use the first day and second day's data to

predict the close price of the third day and keep increasing. In this way, the learning algorithm based on time series was expected to perform best. This approach will be called 'feed-forward input' in the rest of the paper.

Basic Test for one single company comparing performance with input length 1 day, 5 days and 10 days

The code structure, as shown below, comprises four steps:

1. Data preparation and model generation
2. Attention LSTM training and testing with random noise and learning rate decay
3. LSTM training and testing with random noise and learning rate decay
4. illustrating final findings.

Besides, there are also wrappers for both models which a) allows for customized model configurations and also b) keep both LSTM and Attention LSTM have the same programming interface. Both wrappers are derived from torch.nn.Module and overwrites an initialize function and a forward function, with customized LSTM models initialized in the former and called in the latter.

```
def allCompany():
    for i in range(company_amounts):
        one_company_test(...)

def one_company_test():
    #preparation
    read_file()
    normalize_price()
    split_data()
    generate_model_criterion_and_optimizer()
    # test for Attention LSTM
    for j in range(num_epochs):
       randomize_noise()
       model_training()
       learning_rate_decay()
       gradient_clipping()
       model_testing()
    # test for LSTM
    ...
    #print
    print_result()
```

To begin, the stock price is adjusted between -1 and 1 to improve training results. And, as previously stated in section 3, data is split as expected. The mean square error is used to assess the model's loss.

```
def randomize_noise():
    noise = (torch.rand(1) - 0.5) * 0.5
```

For the stock price of a corporation tends to rise each year in reality, statistics might change dramatically over time. This might mean that a model trained on data from the last ten years won't be suitable for testing on data from recent years because it never learns such a high price. As a result, in order to solve the problem of price fluctuation, more noise was added to the training step. The noise, as shown above, is a random number allowing the model to adjust to the large price changes, which could be helpful in providing an over-fitting problem.

The model might then be trained by forwarding data and prediction results to change parameters backward and optimize parameters. However, the loss may continue to fluctuate rather than converging steadily due to observation, which might be caused by the learning rate is too large that the result

would fluctuate between the optimal point, implying that something has to be altered to remedy this issue.

```
def learning_rate_decay():
    lr_new = learning_rate * (
                decay_rate ** (i / 100)
        )
    for param_group in optimizer_lstm.param_groups:
            param_group["lr"] = lr_new
```

As indicated above, learning rate decay is the solution to the loss fluctuation problem. It is observed that in the last stage of training, the loss graph does not converge, if not diverging. By lowering the learning rate, parameter can be changed by only a small bit in each step as the minimum are approached, instead of moving around it using a large learning rate. The learning rate would drop with each iteration. This approach greatly aids in preserving performance and achieving rapid convergence.

```
def gradient_clipping:
    torch.nn.utils.clip_grad_norm_(model.parameters(),10)
```

To train the network more stably, the technique of gradient clipping is noticed, which will clip all gradient norms greater than a predefined threshold, which is set to 10 in the above code snippet. During the early stage of training, the gradient value tends to be too large to allow for stable training of the model. It is observed better performance after applying gradient clipping, please refer to the result section for detailed comparisons.

With all previous steps, this general test procedure could compare performance of Attention LSTM and LSTM with input length of 1 day, 5 days and 10 days.

Comparison for sliding window and feed-forward input

Sliding window and feed-forward input, are two data split methods defined in previous subsection 3. To compare the performance difference with these two methods, somewhere of the basic test procedure introduced in the last subsection would be changed.

```
def allCompany():
    for i in range(company_amounts):
        one_company_test(...)

def one_company_test():
    #preparation
    read_file()
    select_last_year_data()
    normalize_price()
    split_data_sliding_window()
    split_data_feed_forward()
    generate_four_models()
    #for every model run test_for_models function
    test_for_models()
    print_result()

def test_for_models():
    generate_criterion_and_optimizer()
    for j in range(num_epochs):
       #similar to previous procedure
```

Firstly, due to the limited hardware resources and time issues, the whole 12 year data set is reduced to one year by selecting the latest year as the whole data set for model training and testing. All

models would still use the last 20% days as expected test data, which means the test ratio is still 0.2. Moreover, for there were two ways of splitting data, the data set is also split with two different splitting functions implemented by researcher, and the split data sets were stored separately to ensure independence.

Secondly, to confirm that each test is independent of others, four models were generated in the generation step before training, playing a different role for Attention LSTM sliding window, LSTM sliding window, Attention LSTM feed-forward input, and LSTM feed-forward input. The main training and testing procedure was encapsulated into one single function and would generate criterion and optimizer each time using a new model.

Furthermore, in the training and testing procedure function, there is a parameter which is controlling if this model is trained and tested with feed-forward input or not. This parameter enables a special trick for the feed-forward type of input, which is that training and testing can be done at the same time. It is noticed that in testing the feed-forward network, all training and testing data need to be feed into the network, and the output consists of both predictions of training and testing days. And thus the network can be trained and tested in one single pass. All data were fed into the network and only select the training predictions for loss calculation and the test predictions for testing.

One final note is that in feed-forward type of input, the input sequence length of Attention LSTM varies from training to testing, instead of being always the same as in sliding windows approach. This was solved by maintaining a dictionary of lower triangular matrix L with each sequence length mapped to a corresponding matrix. This will same a lot of time compared to calculate a matrix L during running time.

4 Results

The default parameters' value is listed below in Table 1 and all tests were using the same value if it is not specifically indicated.

Table 1: Default Parameter Settings

Parameter	Value	Description
test_ratio	0.2	percentages accounts for testing data
num_epochs	1500	epochs amount
input_dim	5	input dimension amount stands for feature size
output_dim	1	output dimension amount stands for output time length
hidden_dim	32	hidden layer size
learning_rate	5e-4	starting point of learning rate
decay_rate	0.7	decay rate of learning rate decay function

Basic Test Result for One Single Company comparing performance with/without time series input using sliding window method

To compare LSTM and Attention LSTM to a trivial baseline of predicting stock price using ten days' data represents for time series approach and only one day's trading data, where no time series are

involved, two companies were chose for the experiments below, namely Amazon and Apple. The graph and numerical result of Apple's test were stated in Appendix A

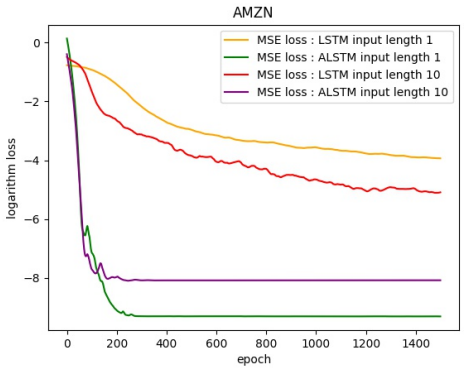

Figure 5: Amazon MSE graph, input : one day and ten days

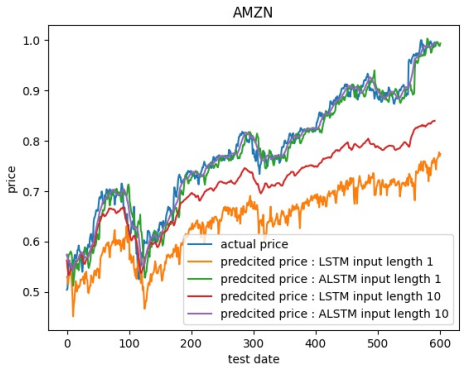

Figure 6: Amazon test graph, input : one day and ten days

Table 2: Numerical result of test with different input time length: one day and ten days

Input Length	Model	Mean Square Error	Training & Testing Time
1 day	LSTM	0.019516	6.20
1 day	ALSTM	9.0e-05	10.3
10 days	LSTM	0.006141	92.8
10 days	ALSTM	0.000308	83.6

Both model's performance was discussed for non-time series data and time series data using 10 days as input time length, As illustrated in Figure 5 and Appendix A, Attention LSTM model is shown to be converged faster than LSTM at around 100 epochs for both company. However, LSTM tends to spend less time than Attention LSTM as shown in Table 2.

At convergence, Attention LSTM had a smaller loss than LSTM, showing its larger model capacity. And hence, Attention LSTM could be more competitive with enough hardware computing resources. As shown in Figure 6, the testing result graph of both models compared to the actual price, it is easy to conclude that Attention LSTM is closer to the actual price.

Hence, the comparison with input length of one day and ten days concludes that Attention LSTM does have better performance than LSTM although LSTM could take less running time. Moreover, to test the performance of models, use one day as input could be a sufficient way for it would run faster and get the same result as a longer period.

Basic Test Result for One Single Company comparing performance of time series with different input length 5 days and 10 days using sliding window method

This test is aimed at finding how sliding window size affects both models' performance. As shown in Figure 7 and Figure 8, Attention LSTM stably outperforms LSTM in different window sizes, while

a longer window size will have little influence on both methods' performance based on selected data from Google company. The fitting results of testing are omitted because of no visible difference between the two settings, instead, a table of quantitative results are provided at Table 3, where Attention LSTM outperforms LSTM in both settings with slightly less time.

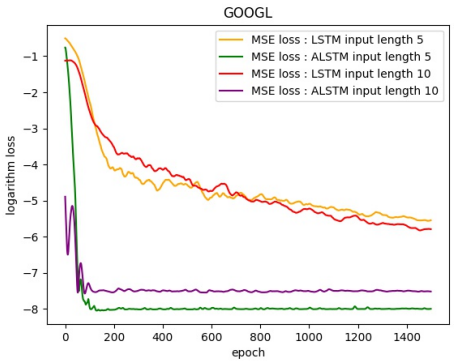

Figure 7: Google Test graph, input: 5 days and 10 days

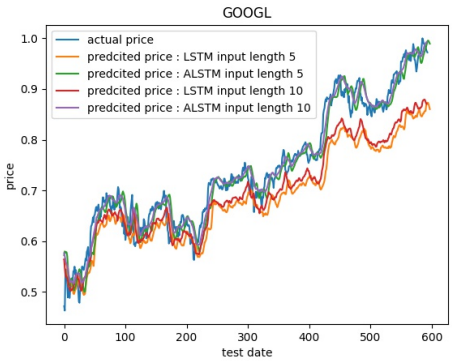

Figure 8: Google Test graph, input: 5 days and 10 days

Table 3: Numerical result of test with different input time length: 5 days and 10 days

Input Time Length	Model	Mean Square Error	Training & Testing Time
5	LSTM	0.003913	44.2
5	ALSTM	0.000336	43.0
10	LSTM	0.003058	91.8
10	ALSTM	0.000544	88.5

Focusing on changes due to different input length, larger input length is observed to require more time for processing. However, the loss is not changing consistently as running time. According to Table 3, LSTM's loss became smaller with the increase of input length, but Attention LSTM's loss was increased under the same situation. And thus, it can be concluded that changes of input length would absolutely affect the length of running time, and it will not affect the comparison result of the model's performance.

Comparison result for sliding window and feed-forward input

As introduced in Section 3, there are two ways of splitting data: sliding window and feed-forward input. The charts below are the results of comparison using sliding window, which input length is 10, and feed-forward input, with Apple and Google's data.

Some parameters were changed to speed up the test due to feed-forward required data to be entered as input in a compounding way. The whole data set length is changed to one year and the aim of two splitting data ways is to predict the close price for the last 54 days in this year. The rest days of this year would be settled to be training data set, which leads to the adjustment of the test_ratio.

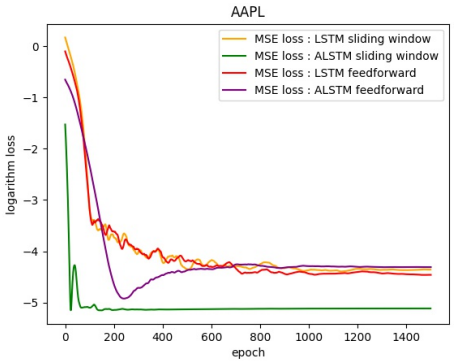

Figure 9: Apple MSE graph comparing sliding window with input length 10 and feed-ward input

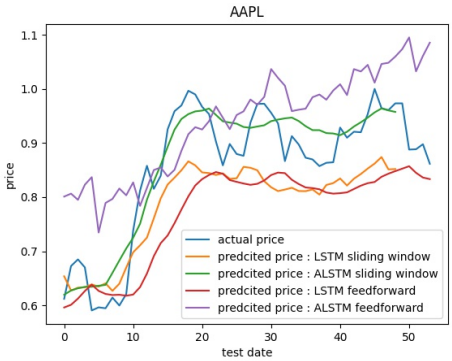

Figure 10: Apple Test graph comparing sliding window with input length 10 and feed-ward input

Table 4: Numerical result of Apple's test comparing sliding window with input length 10 days and feed-ward input

Splitting Data Method	Model	Mean Square Error	Training & Testing Time
Sliding Window	LSTM	0.012815	13.3
Sliding Window	ALSTM	0.005998	11.4
Feed-forward	LSTM	0.011551	89.6
Feed-forward	ALSTM	0.013435	13.5

As shown in Table 4 and Figure 9, LSTM seems to be robust against data splitting methods whereas Attention LSTM struggles with feed-forward way of feeding data. This might because of the removal of forget gate in Attention LSTM which disables its ability to forget history data and focus on recent trading data. Qualitatively, this is also shown in Figure 10, where Attention LSTM fails to model the stock completely using the 'feed-forward' method. This is also shown in Figure 11 and Figure 12, representing results of Google's test.

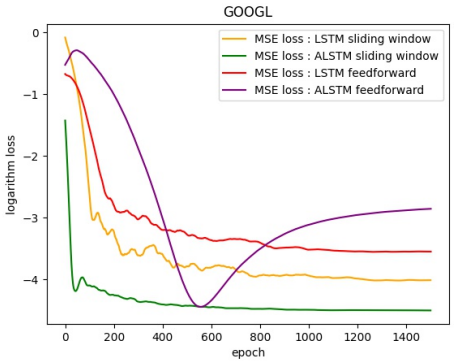

Figure 11: Google MSE graph comparing sliding window with input length 10 and feed-ward input

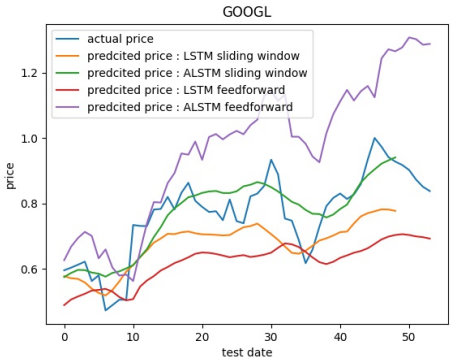

Figure 12: Google Test graph comparing sliding window with input length 10 and feed-ward input

Table 5: Numerical result of Google's test comparing sliding window with input length 10 days and feed-ward input

Splitting Data Method	Model	Mean Square Error	Training & Testing Time
Sliding Window	LSTM	0.018160	13.6
Sliding Window	ALSTM	0.011127	11.9
Feed-forward	LSTM	0.028851	81.7
Feed-forward	ALSTM	0.057653	14.1

What is more interesting is that comparing the mean square error from Apple and Google's test, indicated in Table 4 and Table 5, it is clear to see that Google's test has a larger mean square error. This might be caused by the company's hidden property, which might be related to noise or decay rate in the code. More will be discussed in Section 6.

More adjustment comparison

Comparison of test procedure with and without gradient clipping

As introduced in Section 3, gradient clipping was applied to stabilize Attention LSTM's training with feed-forward input. In the early stage of training, it is observed that Attention LSTM's gradient being so huge to be stably trained, and thus gradient clipping will effectively lower each step's change to parameters.

Compare Table 6 and Table 4, gradient clipping merely affects all other methods, but improves Attention LSTM's performance by a large margin. The more interesting thing is, comparing Figure 13 and Figure 14 to Figure 9 and Figure 11, it is observed that without gradient clipping, the prediction price from Attention LSTM with feed-forward input tends to smaller than the actual price for both company. However, this does not occur in the other approaches.

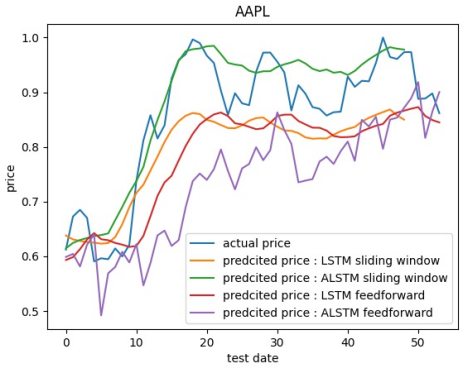

Figure 13: Apple Test graph without gradient clipping

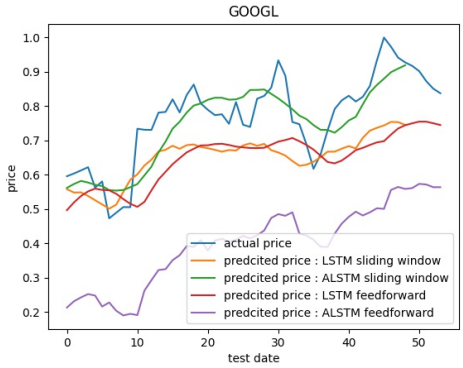

Figure 14: Google Test graph without gradient clipping

Table 6: Numerical result of test without gradient clipping

Company	Splitting Data Method	Model	Mean Square Error	Training & Testing Time
Apple	Sliding Window	LSTM	0.011087	13.2
Apple	Sliding Window	ALSTM	0.005894	10.9
Apple	Feed-forward	LSTM	0.009143	81.7
Apple	Feed-forward	ALSTM	0.022127	13.2
Google	Sliding Window	LSTM	0.024071	13.4
Google	Sliding Window	ALSTM	0.013271	11.4
Google	Feed-forward	LSTM	0.020925	82.5
Google	Feed-forward	ALSTM	0.140109	13.8

Comparison of Attention LSTM model with and without 'relu' function

Relu function, or shown as σ_C, is an identity function used to calculate hidden size based on weight as indicated in Section 2.

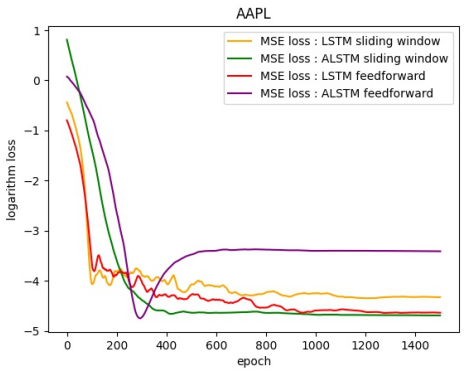

Figure 15: Apple MSE graph with relu method

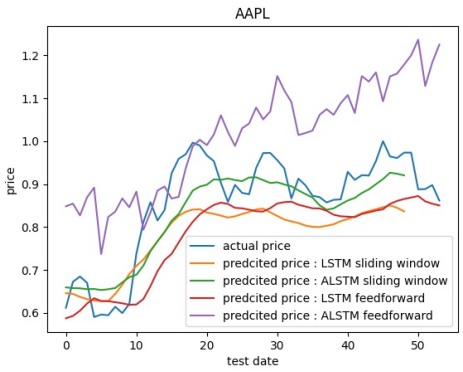

Figure 16: Apple Test graph with relu method

Table 7: Numerical result of Apple's test using 'relu' method

Splitting Data Method	Model	Mean Square Error	Training & Testing Time
Sliding Window	LSTM	0.013233	13.6
Sliding Window	ALSTM	0.009192	11.9
Feed-forward	LSTM	0.009691	89.5
Feed-forward	ALSTM	0.033053	15.4

Compared to Table 4, it is observed a performance drop if not almost the same in all settings in Table 7. Using the ReLU function is argued to prohibit the network to learn an identity function

which is highly useful in the setting, because stock price tends to be around the price of the day before, while ReLU will change (normalized) price below zero to zero.

This could also be approved by comparing Figure 9 and Figure 10 to Figure 15 and Figure 16. It is obvious that without ReLU, models with sliding window approach will not be affected much, and models with feed-forward approach could have a better and more steady result, especially for Attention LSTM. The predicted price of Attention LSTM became more closed to the actual price without ReLU.

5 Responsible Research

The responsibility of this research is discussed in this section. Special attention is paid to the fair comparison and reproducibility of the experiments.

First of all, an open-source data set ,available on 'Kaggle' website according to the original license, was used in the research progress. It ensures that there are no security and privacy related concerns in the project. As for fair comparison and design of experiments, out of the hundred companies in SPX, experiments were run on the selected ones: Amazon, Apple, and Google for they are well-known and more representative.

As for reproducibility, the code will be publicized upon the finishing of the project in addition to model parameters to reproduce the quantitative results, a demo Jupiter notebook will be provided for fast reproduction of the results. As for hyperparameter settings, the code uses Adam optimizer with default hyperparameters as defined in PyTorch.

As shown at the beginning of Section 4, the detailed default value of the used parameter was explicitly listed in Table 1 and every change of parameter's value were also indicated in the section. Furthermore, the detailed function which could help in improving performance was also introduced clearly in Section 3, as well as the logic introduced in Section 3. These comments, it is expected to help other researchers to reproduce the experiments.

6 Conclusions and Future Discussion

In this work, Attention LSTM was introduced and implemented , and run detailed experiments comparing LSTM and Attention LSTM, with a special focus on predicting time series in the financial world.

In general, the research question in Section 1 is answered that Attention LSTM outperforms LSTM with 'sliding window' method defined in Section 3. It has a higher chance of getting a lower mean square error after more iterations, implying that it could match the expectations of high-accuracy prediction outcomes if powerful computing equipment is available. And it is even slightly faster than LSTM. Moreover, use one day as input length could represent the performance evaluating result using less time though it is not an actual time series.

Furthermore, to answer the question which is 'if changing the input time series' length would affect the performance', it is observed that input length wouldn't affect much in models' performance comparison. There might be a slight difference in loss which is not stable, but a larger input length is confirmed to require more time for running.

In advance, it is interesting to compare the performance with two methods defined in Section 3: 'sliding window' and 'feed-forward input'. First of all, it is observed that Attention LSTM can hardly handle 'feed-forward' input. Although Attention LSTM takes less running time compared to LSTM with feed-forward input, it always has a larger loss which means more uncertainty. With feed-forward input, LSTM has better performance, which might due to its architecture is more suitable for large

input time length. However, compared to 'feed-forward' method, both LSTM and Attention LSTM could have similar or even much better performance with 'sliding window' approach with less time. Hence, 'sliding window' is observed to be a better approach.

What's more, during the experimental process, gradient clipping was found out to be helpful and the 'relu' function is observed to be unnecessary.

Future works

Future works were discussed that may enhance the method's performance in this section, namely improvements with respect to data, model, and training procedure.

- Data: The input feature of stock in the current implementation only consists of the open/close price, trading volume, while in traditional stock technical analysis, it is prevalent to use (exponential) moving average, beta value, related indices, etc. By introducing such features, the network is believed to be able to have a more explicit and broad view of the stock itself and the market it is in. Such feature engineering methods are also quite popular in the data analysis world.

- Model: The current implementation of Attention LSTM only consists of a few message passing steps, which only allows message passing from pair of time steps, whereas LSTM, by maintaining a hidden state evolving across time steps, can pass messages from any previous time step to current time step, which might render LSTM more advantageous against the Attention LSTM. More message passing steps will make Attention LSTM's representation power about the same as LSTM, which could fully unleash the power of Attention LSTM.

- Training procedure: In the experiments, two kinds of training procedures are introduced: a sliding window way and a feed-forward way. The detailed comparison shows how differently those training procedures behave. However, to allow stable training of Attention LSTM, new training procedures should be proposed, for example, progressive training with variable sequence length, a residual training to allow the network to learn an identity function. More experiments can be done to discover more about Attention LSTM. As most of the paper focuses on how different hyperparameter settings affect the performance of Attention LSTM, how hyperparameter settings affect an individual stock are also interested, i.e. a specific stock may have its own set of hyperparameter. Such per-stock finetuning of parameters may allow for better performance.

A Apple test result of sliding window with input length one day and ten days

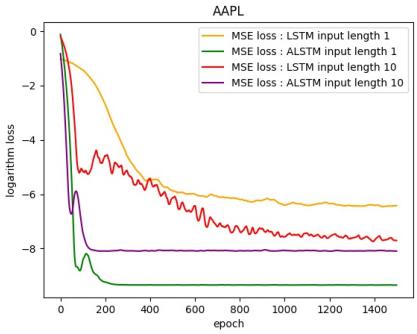

Figure 17: Apple MSE graph, input : one day and ten days

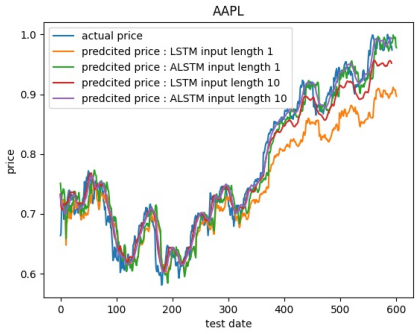

Figure 18: Apple test graph, input : one day and ten days

Table 8: Numerical result of Apple test with different input time length: one day and ten days

Input Length	Model	Mean Square Error	Training & Testing Time
1 day	LSTM	0.001639	6.14
1 day	ALSTM	8.8e-05	11.0
10 days	LSTM	0.0000452	95.4
10 days	ALSTM	0.000306	86.0

References

Alex Sherstinsky (2018). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. CoRR, abs/1808.03314.

Junyoung Chung and Caglar GÃŒlcehre and KyungHyun Cho and Yoshua Bengio (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. CoRR, abs/1412.3555.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Comput., 9(8), 1735â1780.

Ashish Vaswani and Noam Shazeer and Niki Parmar and Jakob Uszkoreit and Llion Jones and Aidan N. Gomez and Lukasz Kaiser and Illia Polosukhin (2017). Attention Is All You Need. CoRR, abs/1706.03762.