# ECSE 484 Computational Intelligence Final Project:

# Recurrent Neural-Network-based in Forecasting S&P 500 index

Linjiang Ke
lxk334

## 1. Introduction

1.1 Abstract
This research mainly forecasts the S&P 500 index in the next few days based on the recurrent neural network. In the early stage of the research, I built a neural network and got good results, but it showed a large data difference at the end of the data, so I updated the neural network again, which means that I increased the hidden layer number, reducing the learning rate and increasing the number of training sessions, and finally got a nice boost.

1.2 Process of My Research:
Regarding this project, firstly, I will get data from the NASDAQ website, then I will design and develop a Recurrent Neural Network (RNN) based on python and PyTorch. Then, I will train the RNN based on the training set. Then test to make sure it works correctly when predicting the next transaction data value. So I will use a sliding window the width of 180 days to test the accuracy of my model, and I used prediction error to show the accuracy of the model. And at the end of the research, I investigated the effect of Input Noise and Uncertainty.

## 2. Background

2.1 Overall Background
For creativity and my own interest, I really love financial data analysis and aspire to work in a Fintech company, so this project is very interesting for me, and predicting the S&P has significant implications for future financial analysis, investing, and many important financial indexes. Especially in quantitative finance, the prediction of stock indexes will greatly improve returns and a stable model will reduce the risk of high-frequency trading.

2.2 Data Collection
I selected S&P500 (Symbol = SPX) Dataset from the official website of Nasdaq. This will get the historical data of S&P from 1960 to now, and the amount of data can be used to make model predictions. And the link is below:
https://www.nasdaq.com/market-activity/index/spx/historical

2.3 Development Environment

Python 3.8 / Pytorch 1.8.0 / jupyter notebook

## 3. Analysis of my RNN Model

As we can see from figure1, this is the architecture of the RNN model I designed. To build our RNN, I defined a class RNN that inherits a base PyTorch class *torch.nn*. Then I defined different parameters and layers for RNN under the constructor RNN, the parameter of *n_layers* means the number of recurrent layers, and the parameter of *hidden_size* means the number of features (different prices) in the hidden state. I should mention that the shape of weights is (4, 16), so the size of the final output is a size 4 vector. The *input_size* will be 4 because it is the number of different prices, and the *output_size* will be the final result. So $W_{hh}$ is a shape of [16,16] and $W_{ih}$ is a shape of [16,4]. And the parameter of *init_hidden* could create a tensor of zeros. And there are 2 important hyperparameters that need to be specifically introduced. Because the model needs to update the weights after each of the times when backpropagation was done, the *learning_rate* will record that.

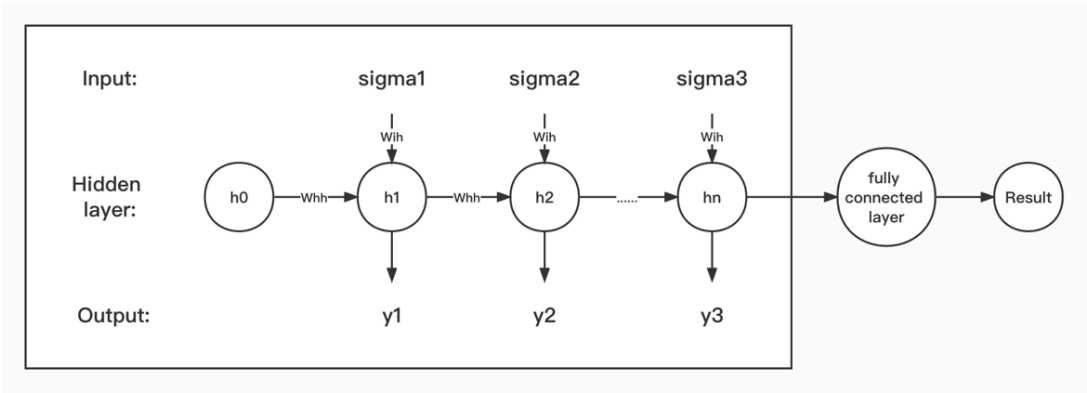


Figure 1: Architecture of RNN

I defined a forward function that belongs to the class method forward and it determines the flow of data and is executed sequentially. So the logic of this model is that let the initial data pass the inputs and the zero-initialized hidden layer first, Then according to the order shown in the figure, passed the RNN outputs to the fully connected layer. And *n_epochs* will be the number of epochs in the RNN model, so I set this epoch at 150 at the first time and after running the training process, I found the epoch will be great. The following figure is the epoch Mean Squared Error in 10 epochs. And the optimizer uses adam to process the results of normalization

<matplotlib.legend.Legend at 0x7fc9522c5af0>

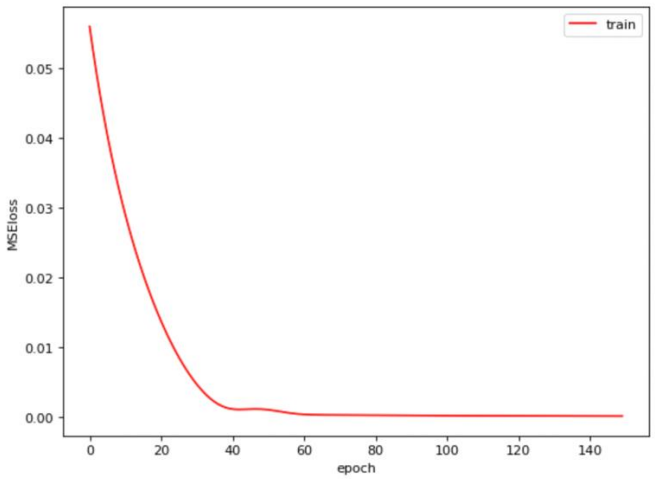


Figure2: Result of MSE when epoch = 150

## 4. Dataset

The original data contains 15682 lines of S&P price data from Jan 4, 1960 to Apr 22, 2022. They all include 4 types of data: open represents the daily opening price, close represents the daily closing price, high represents the highest price, and low represents the lowest price. The following figure is the output of the original data and the line chart of the S&P 500 in almost 60 years.

| | Date | Open | High | Low | Close |
|---|---|---|---|---|---|
| **0** | 1960-01-04 | 59.910000 | 59.910000 | 59.910000 | 59.910000 |
| **1** | 1960-01-05 | 60.389999 | 60.389999 | 60.389999 | 60.389999 |
| **2** | 1960-01-06 | 60.130001 | 60.130001 | 60.130001 | 60.130001 |
| **3** | 1960-01-07 | 59.689999 | 59.689999 | 59.689999 | 59.689999 |
| **4** | 1960-01-08 | 59.500000 | 59.500000 | 59.500000 | 59.500000 |
| **...** | ... | ... | ... | ... | ... |
| **15678** | 2022-04-18 | 4385.630000 | 4410.310000 | 4370.300000 | 4391.690000 |
| **15679** | 2022-04-19 | 4390.630000 | 4471.030000 | 4390.630000 | 4462.210000 |
| **15680** | 2022-04-20 | 4472.260000 | 4488.290000 | 4448.760000 | 4459.450000 |
| **15681** | 2022-04-21 | 4489.170000 | 4512.940000 | 4384.470000 | 4393.660000 |
| **15682** | 2022-04-22 | 4385.830000 | 4385.830000 | 4267.620000 | 4271.780000 |

15683 rows × 5 columns

Figure3: Output of the original data

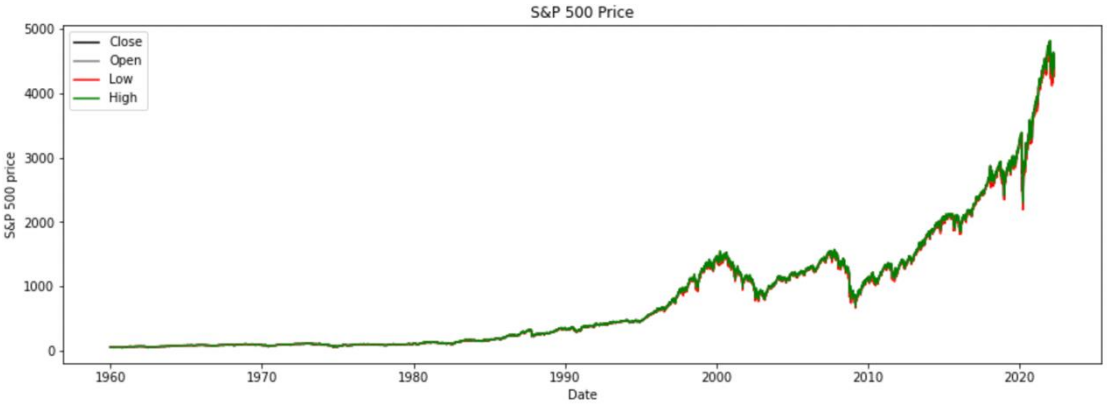


Figure4: S&P 500 Price (1960-2022)

At the begging of the research, I just use the method of splitting year by year, which means that I use the first 80% of the data each year for training and the last 20% for testing, instead of splitting the total data, which can effectively avoid other factors. interference makes the model inaccurate. And the size of the training set is *batch_size*. The following figure shows the value of Close in the training set and test set. Red dots represent the training set and the green represents the test set.

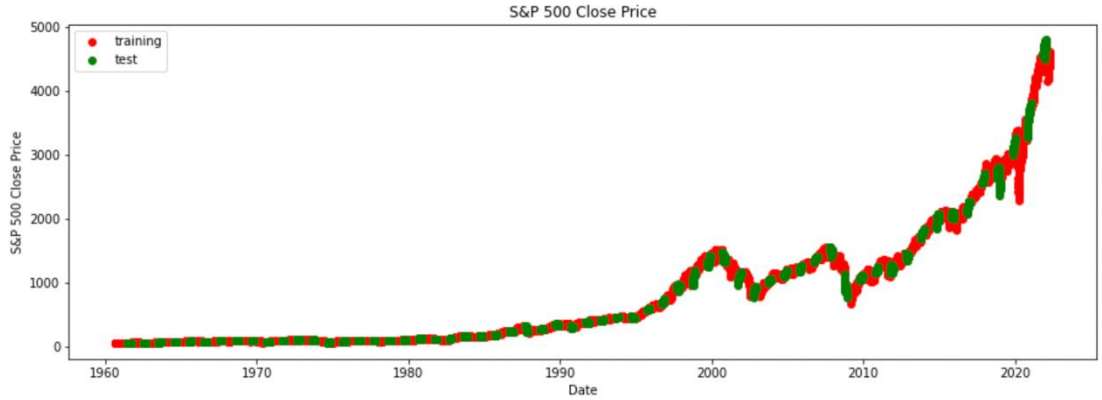


Figure5: S&P 500 Clone Price (1960-2022)

After that, I found out that the result of the end is not very fit, so I improved the model, which split the original dataset into three parts: training, test, and validation. Figure6 is the result of the improved model, the blue point is the test set while the green point is the validation set.

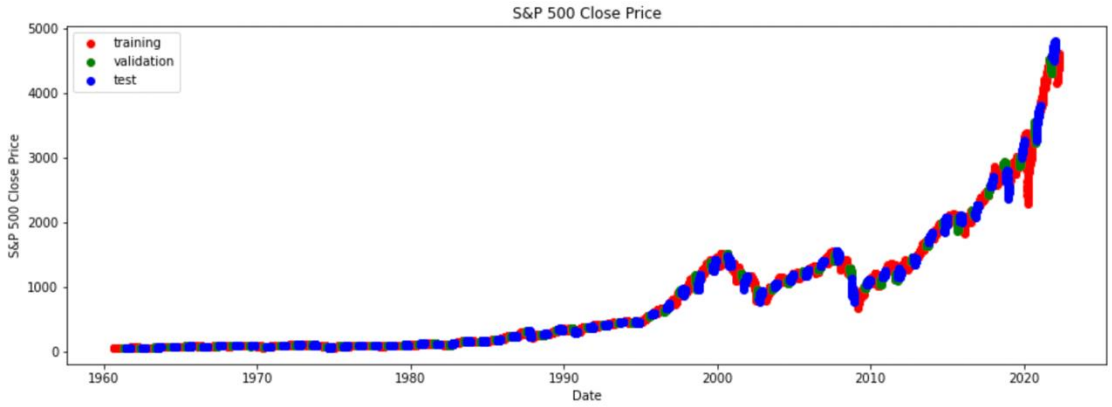


Figure6: S&P 500 Clone Price (1960-2022)

## 5. Result of the RNN model

5.1 Model result before optimization

In the beginning, I didn't use an optimized method, so we can see from the figure, it shows the predicted trend for the next day and 4 days, we can see that at the end of the image, the data of Real and Pred have a large difference, the rest is acceptable.

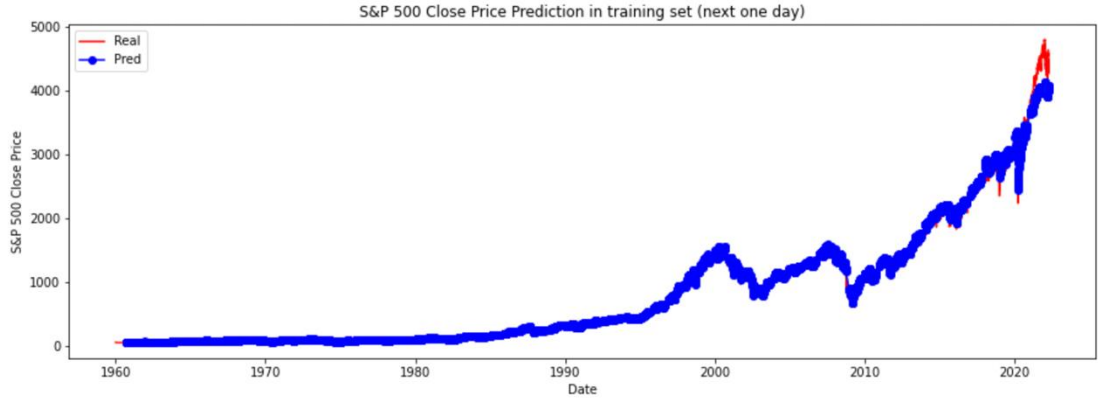


Figure7: S&P 500 Clone Prediction for the next 1-day method

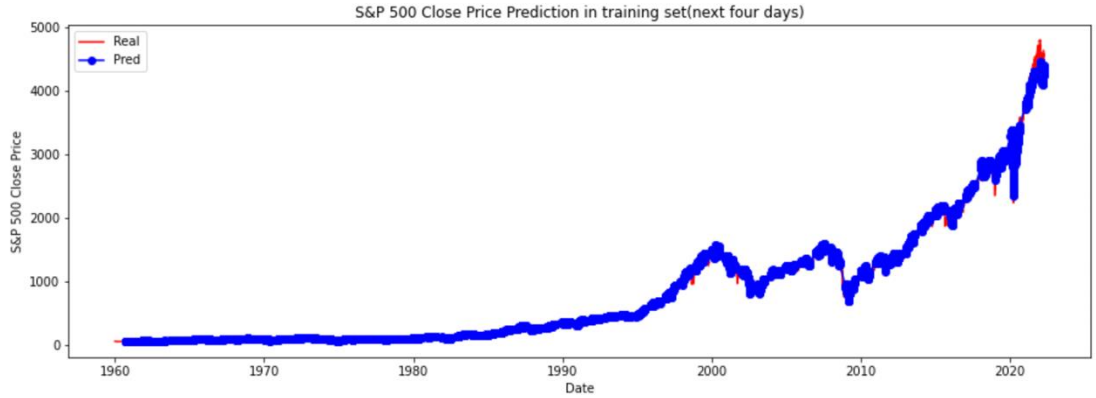


Figure8: S&P 500 Clone Prediction for the next 4 days method

Then I calculate the error. Figure9 and figure10 are the Prediction Error figures.

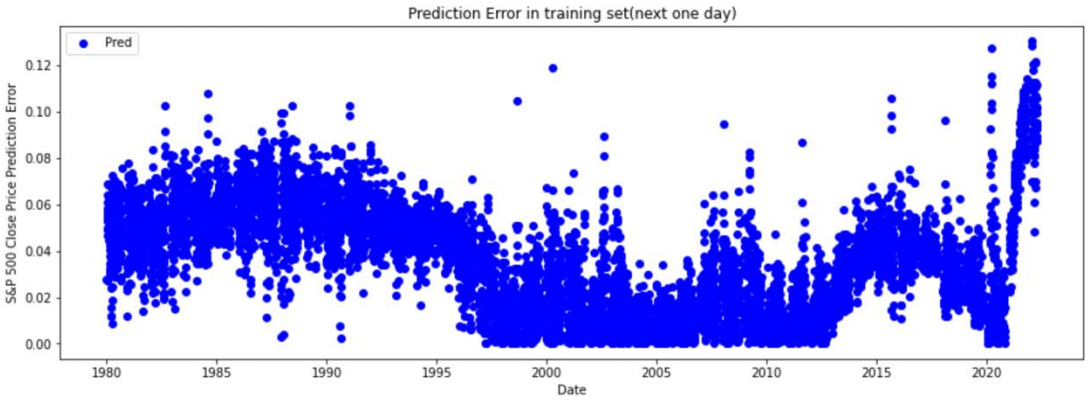


Figure9: Prediction Error for the next 1-day method

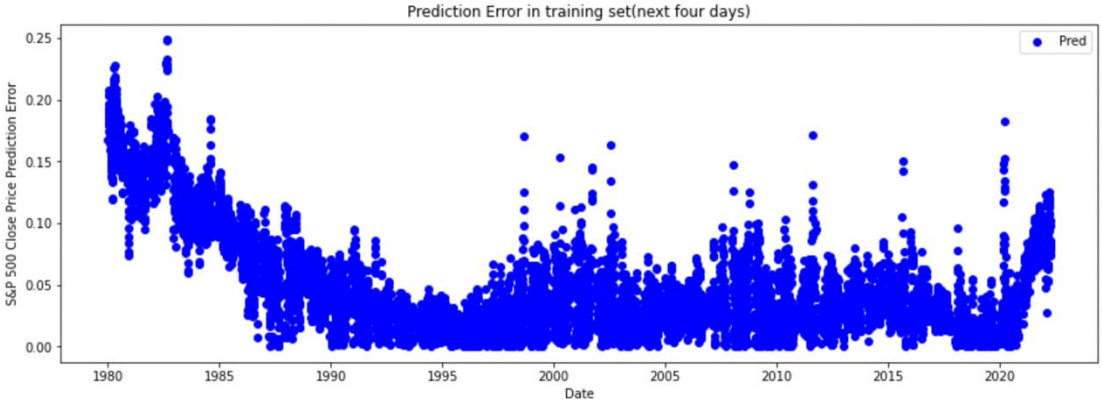


Figure10: Prediction Error for the next 4 days method

And the following figure is the prediction model in the test set. I will just post one figure based on the next 1-day method.

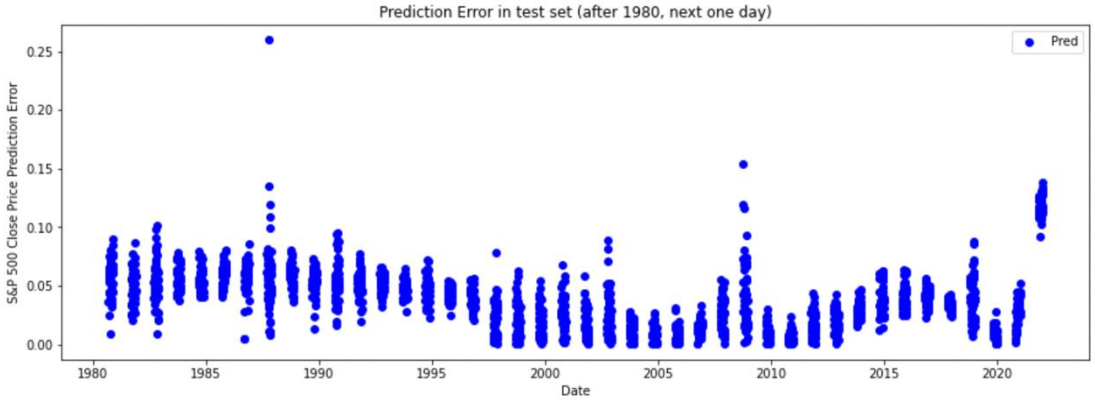


Figure11: Prediction Error (test set) for the next 1-day method

## 5.2 Model result after optimization

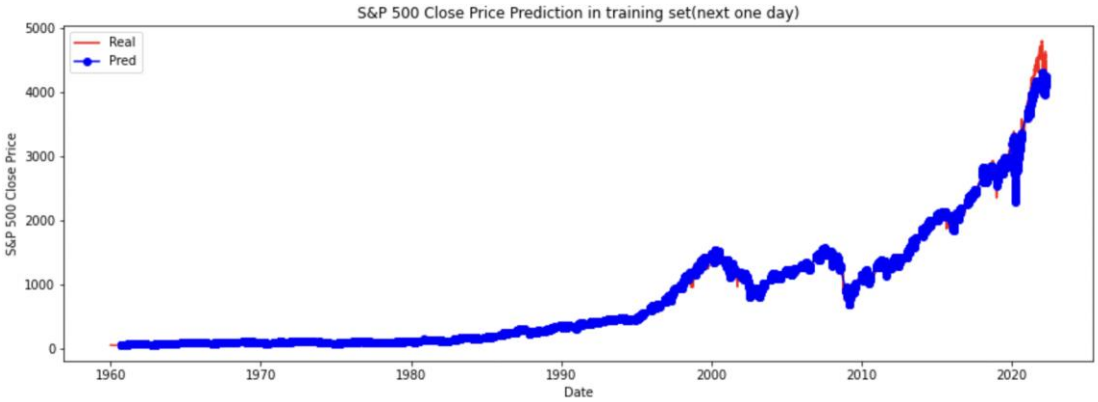


Figure12: S&P 500 Clone Prediction for the next 1-day method (after optimization)

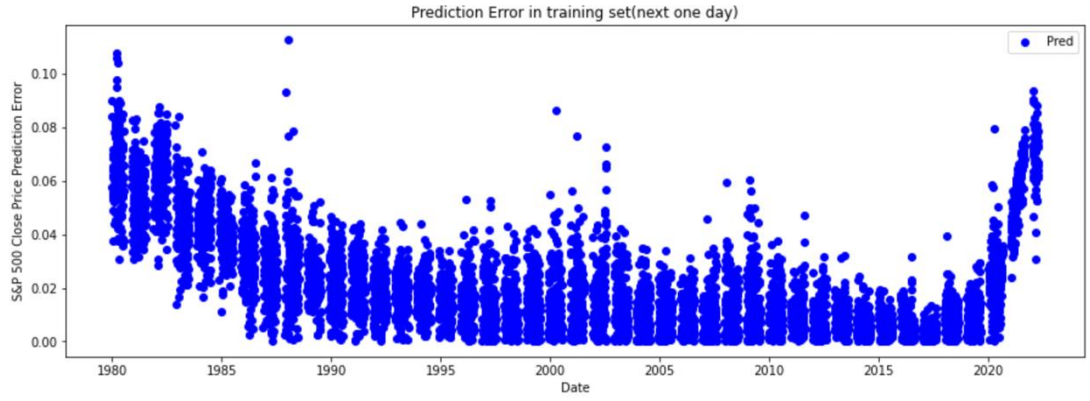


Figure13: Prediction Error for the next 1-day method (after optimization)

We could see clearly that the error is getting smaller if I optimize the model by 3 parts.

## 6. Noise and Uncertainty Prediction Error

As we could know, prediction could be calculated as: $\frac{|\text{predict\_price} - \text{real\_price}|}{\text{reak\_price}}$,

Prediction error metrics include mean, standard, skewness as well as kurtosis. *Mean* is to compute the arithmetic mean of *prediction_error*, *Standard* is to measures the amount of variation or dispersion of *prediction_error*. *Skewness* is to refers to a distortion or asymmetry that deviates from the symmetrical bell curve, and *Kurtos* is describes the degree to which scores cluster in the tails or the peak of a frequency distribution of *prediction_error*.

So following the rules, I designed the model that could show out the prediction error with different noise levels. This is all base on the next 1-day.

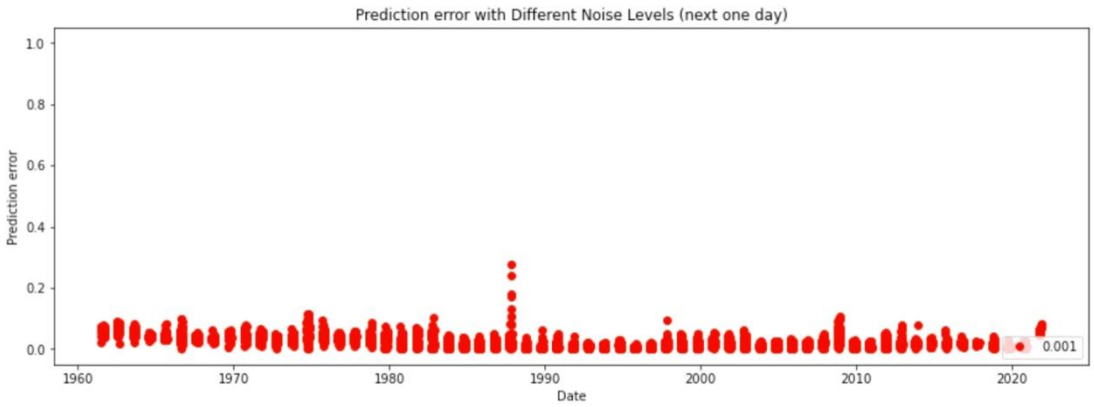


Figure 14: Prediction error with stdv = 0.001 for the next 1-day method

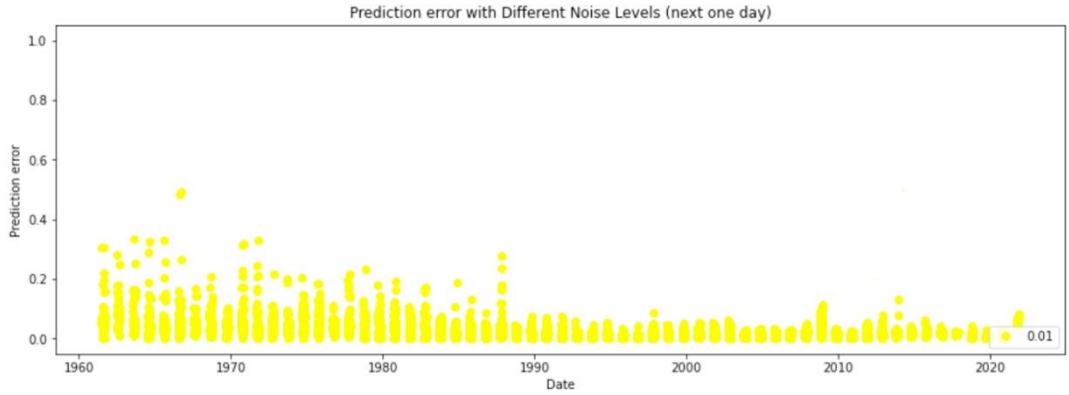


Figure 15: Prediction error with stdv = 0.01 for the next 1-day method

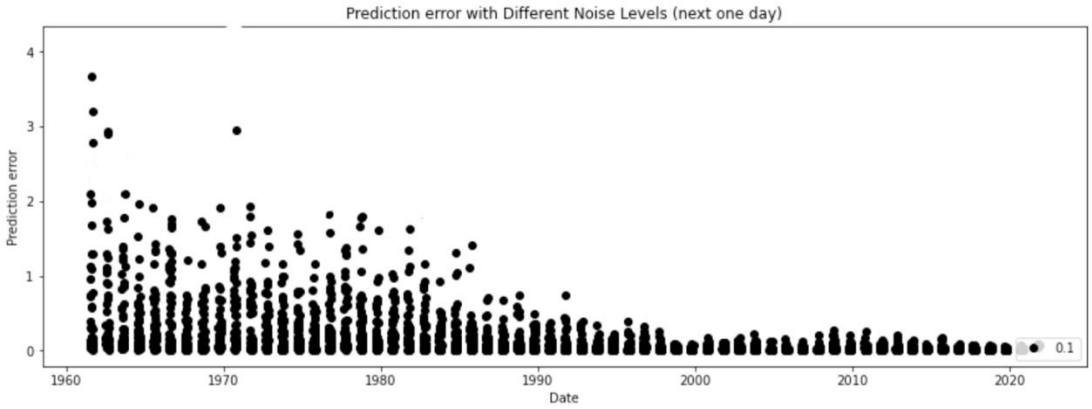


Figure 16: Prediction error with stdv = 0.1 for the next 1-day method

## 7. Improvements and Future

In this final project, I built 3 RNN models, one unoptimized and two optimized models. In the future, I want to import a real P/E index to show a real figure and optimize the prediction model, it will be closer to the real stock market. Finally, Finally, I am very grateful to Professor Wyatt Newman and our TA.