Machine learning for index price prediction and portfolio weight optimization

Xin Gu, Dingtian Zhu

Aug 28, 2020

Abstract

Relative to traditional statistical methods in asset pricing, machine learning accommodates a far more expansive list of potential predictor variables, as well as richer specifications of functional forms. Machine learning methods can be successfully applied to the canonical problem of empirical asset pricing: predicting returns in cross-sectional way.

The literature has accumulated a long list of predictors that various researchers have argued possessing forecasting power for returns. The volume of stock-level predictive characteristics reported in the literature numbers in hundreds and macroeconomic predictors of the aggregate market number in dozens. However, compared to the enthusiasm researchers devote to factor development, there is less literature focusing on portfolio construction. Machine learning can help with most portfolio construction tasks like price prediction, asset allocation, weight optimization and the back-testing of strategies.

Here we want to use machine learning methods, mainly to build an Recurrent Neural Network(RNN) model, to realize an automated prediction for index price using historical daily price data. The model would mainly help investors in asset allocation and portfolio construction. Stock-level predictive characteristics and macro-economical factors could be added on the base of our model to build more complicated ones for stock price prediction or to improve performance.

Key Words

Portfolio construction, cross-section of expected returns, Recurrent Neural Networks, LSTM.

Contents

1 Introduction

Asset allocation is a must-made decision faced by both individual and institutional investors when building portfolios. One needs to choose how to allocate the funds across a basket of asset classes. For instance, a globally invested pension funds must decide how much to invest in each major country or region.

In principle Modern Portfolio Theory (the mean-variance approach of Markowitz), it offers a solution to this problem once the expected returns and covariances of the assets are known. While Modern Portfolio Theory is an important theoretical advance, its application has universally encountered a problem: although the covariances of a few assets can be adequately estimated, it is difficult to come up with reasonable estimates of expected returns.

The existing forecasting methods make use of both linear (AR, MA, ARIMA) and non-linear algorithms (ARCH, GARCH, Neural Networks). Here we want to make progress from a data processing perspective, to give expected returns using machine learning methods. In particular, we show how to build a Recurrent Neural Network (RNN) with Long-Short-Term-Memory units to predict index price using historical daily price data.

Our methodology can be thought of as a data processing step before weight optimization, and we benchmark it against naive historical price time series. Also we try statistical models like ARIMA and use predicted price as input to decide allocation, and to compare performance of the constructed portfolio.

2 Literature review

Long-Short-Term-Memory (LSTM) networks are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter Schmidhuber (1997), and got refined and popularized by researchers. LSTM is widely used in sequence modelling as it solves the problem of gradient vanishing/explosion of RNN.

LSTM is also used in financial time series processing recent years. Zhou et al. (2015) model and predict China stock returns using LSTM. The historical data of China stock market are transformed into 30-days-long sequences with 10 learning features and 3-day earning rate labeling.

Selvin et al. (2017) focus on identifying the latent dynamics existing in the data using deep learning architectures for NSE listed companies and compared the prediction results between RNN, LSTM and CNN.

Chen et al. (2019) states macroeconomic information dynamics are summarized by macroeconomic state variables which are obtained by a Recurrent Neural Network (RNN) with Long-Short-Term-Memory units.

Our paper follows some of their research directions and provides a deep learning framework of index price prediction.Our deep learning framework would have three-fold innovations.

1) We use LSTM model for index price prediction, focusing on helping investors make better asset allocation decisions.

2) Compared to traditional models like ARIMA or just using historical price to give expected returns, our portfolio constructed by LSTM model performs well in long-term return. Also it shows better results than some naive investment strategies like buy-and-hold.

3) Our model uses daily price data, which is easily accessible. One can fine-tune our model following our framework for stock selection.

3 Methods

In our study, we first use both traditional statistical methods and machine learning methods to predict selected index prices. We then use mean-variance framework and max_sharpe method provided by PyPortfolioOpt package for portfolio construction and performance comparison.

Since our research is based on price time series, we would use time series cross-validation and sliding-window validation. For cross-validation, instead of using one test and training set, we set a series of training-test sets. Each training set consists of price data prior to a specific predetermined date, which is also the observation that forms the test set. And the corresponding test set consists of the data of the same time range after the training period. For instance, in our study, we would use the first sixteen years' daily data as our base training set. Each time after analyzing training and test sets, the test set would be combined to the training set, and the next month's data would work as a new test set. This procedure would be repeated until the end of the whole data set.

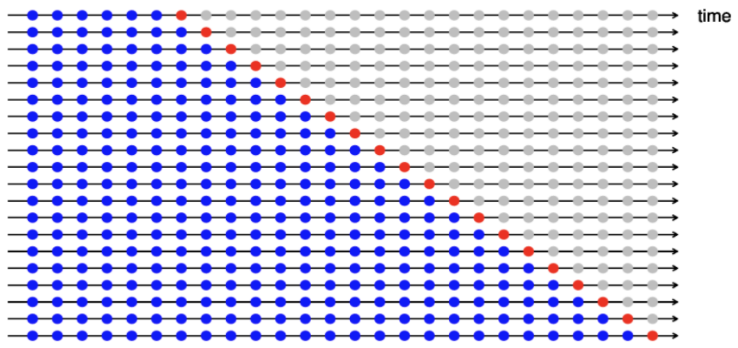

Figure 1: Time Series Cross-Validation

Next, we push predicted price time series as inputs into PyPortfolioOpt package to get monthly expected returns and expected sharpe ratio of the constructed portfolios. We can compare the results to see whether the machine learning methods would help to construct investment portfolios with better performance.

3.1 Traditional methods

In this part, we would try two different approaches. One is to use historical price information directly; the other is the Auto Regressive Integrated Moving Average (ARIMA) model, which is a classic statistical model for analyzing and forecasting time-series data.

3.1.1 Using historical data

First, we would use the historical price data directly as inputs to optimize allocation and get actual returns of the constructed portfolio.

The portfolio is allocated based on maximize Sharpe Ratio. Each time, we first calculate the weight of each index in the combination with the highest Sharpe Ratio using the training set. These weights would be used to construct the portfolio in the test set. Since we have already got real historical price information, here we just use these real price data to calculate returns. In this method, what we get is a series of portfolio returns computed using actual prices data, which can be treated as actual returns using historical data naively.

Source Code: Getting weights by PyPortfolioOpt

```
    mu = expected_returns.mean_historical_return(historical_time_seires)
    S = risk_models.sample_cov(historical_time_seires)
    ef = EfficientFrontier(mu, S)
    raw_weights = ef.max_sharpe()
    cleaned_weights = ef.clean_weights()
```

3.1.2 Using ARIMA model

In addition to using the original historical data, we also try the statistical ARIMA model, which is often used to analyze time-series data.

ARIMA model is a generalization of an Auto Regressive Moving Average (ARMA) model. It combines differencing with autoregression and a moving-average model. The Auto Regressive (AR) part suggests that the time series is based on its own lagged values. While the Moving Average (MA) part indicates the interested variable is also regressed on its lagged forecasted errors.

Therefore, the ARIMA model can be expressed as:

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \ldots + \beta_p Y_{t-p} \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + \ldots + \phi_q \epsilon_{t-q}$$

Where Y_{t-1} is the lag_1 for the series, $beta_1$ is the coefficient of lag_1, α is the intercept term and errors ϵ_t and ϵ_{t-1} are the errors from the following equations:

$$Y_t = \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \ldots + \beta_0 Y_0 + \epsilon_t$$

$$Y_{t-1} = \beta_1 Y_{t-2} + \beta_2 Y_{t-3} + \ldots + \beta_0 Y_0 + \epsilon_{t-1}$$

3.2 Machine learning method

3.2.1 RNN and LSTM

Long-short-term-memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feed-forward neural networks, LSTM has feedback connections. The advantage of LSTM is that it keeps long-term information and drops it when necessary.

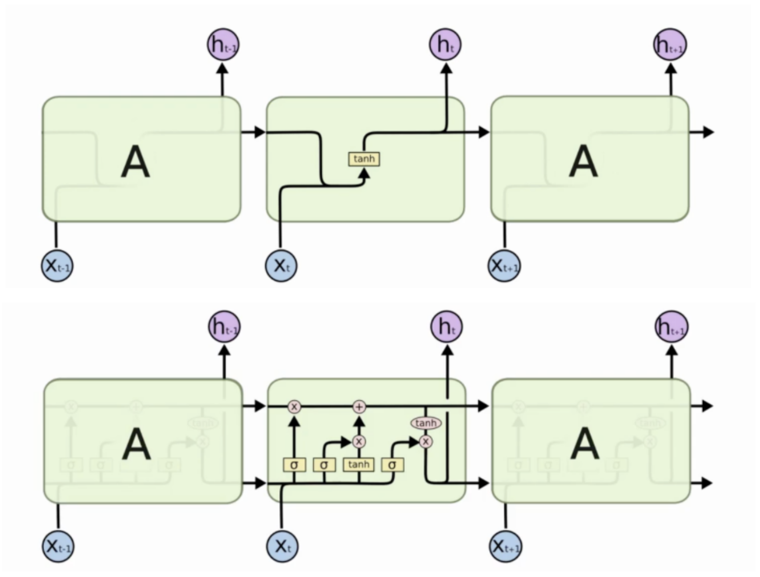

Figure 2: Schema of LSTM cell VS RNN cell

In a traditional RNN cell, input X_t is processed by activation function. Combined with previous information h_{t-1}, the cell gives out new output h_t.

In a LSTM cell, in addition to the dense layer of tanh activation function, there are three gates to take control of the information flow. Forget gate and Input gate work together to get updated cell state C_t from C_{t-1}, h_{t-1} and X_t. Output gate gives h_t according to input X_t and cell state C_t. In that way, time series information is passed by both C_t and h_t, which contains long-term and short-term information.

3.2.2 Modelling process

For this part, we talk about the general design of our model. The detailed process and schema would be shown in the empirical study part.

The RNN model we are to build has LSTM cells as basic hidden units. We use values from the very beginning in the first sliding window W_0 to the window W_t at time t: W

$$W_1 = (p_w, p_{w+1}, \dots, p_{2w-1})$$

$$\dots$$

$$W_t = \left(p_{tw}, p_{tw+1}, \dots, p_{(t+1)w-1}\right)$$

to predict the prices in the following window W_{t+1}:

$$W_{t+1} = \left(p_{(t+1)w}, p_{(t+1)w+1}, \dots, p_{(t+2)w-1}\right)$$

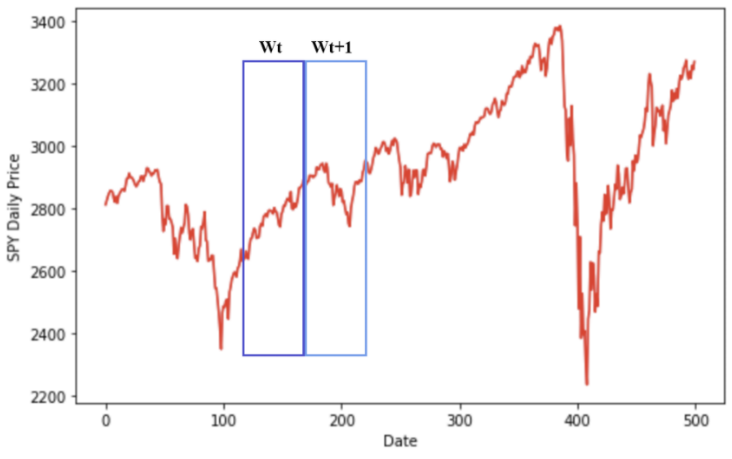

Figure 3: Sliding-window training

Here we use sliding-window validation as the model should take fixed-dimensional data as input.

As our target is to predict index price on a 30-day scale based on daily price data, we have two alternatives. The first way is to use the sliding-window method to predict 1-day price, and tune the model by minimizing prediction loss. Then we use the trained model to test performance in test set by iteratively predicting 1-day price for 30 times to give a 30-day prediction window.

The second way is to predict the 30-day price series directly. In this way, it's much more time-consuming for model training as the output would be 30-dimensional for a single index, while easier for test prediction because there's no iterative work.

To realize comparison between them, here we follow a popular design pattern by setting two parameters in our model named *input_size* and *num_steps*. The sequence of prices are first split into non-overlapped small windows. Each contains *input_size* numbers and considered as one independent input element. Then any *num_steps* consecutive input elements are grouped into one training input, forming an unrolled version of RNN for training on Tensorflow.

For instance, if we set *input_size* = 3 and *num_steps* = 2, our first few training examples would look like:

$$\text{Input}_1 = [[p_0, p_1, p_2], [p_3, p_4, p_5]], \ \text{Label}_1 = [p_6, p_7, p_8]$$
$$\text{Input}_2 = [[p_3, p_4, p_5], [p_6, p_7, p_8]], \ \text{Label}_2 = [p_9, p_{10}, p_{11}]$$
$$\text{Input}_3 = [[p_6, p_7, p_8], [p_9, p_{10}, p_{11}]], \ \text{Label}_3 = [p_{12}, p_{13}, p_{14}]$$

In that way, we should just choose between parameter set: *input_size* = 1, *num_steps* = 252 and *input_size* = 30, *num_steps* = 12. We set *num_steps* according to real-world trading, as there are approximately 252 trading days in a year and we try to use approximately one-year's previous data as input to learn the pattern contained in historical price. Also, literature shows LSTM doesn't often show great work with long time series with thousands of data points as input.

After some training work, we choose to apply the first setting in later empirical study.

4 Metrics for assessing forecast performance

This part will introduce the optimizing metrics we will use in our study.

4.1 Performance measures for point forecasts

For LSTM prediction error, we would use the average, referred to as the overall weighted average (OWA), of two of the most popular accuracy measures:

1) mean squared error MSE

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

2) mean absolute percentage error MAPE

$$MAPE = \frac{100\%}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

Here we only use MSE as the data is normalized beforehand and prediction target actually becomes daily return, which means MAPE or SMAPE will not work as percentage shows no meaning.

For the predicted price series, we construct portfolios and compare their performance, MSE_MAPE comparison is made between ARIMA and LSTM.

4.2 Performance measures for constructed portfolio

For portfolio performance, we choose:

1) Cumulative return

2) Sharpe ratio

3) Max drawdown

All of the three indicators are widely used to measure portfolio performance over time. Applying a combination of these measurements would take care of both investment return and risk.

5 Empirical study

In this section, we will introduce the empirical study of our research, including the datasets and software we used, some preliminary analysis we referred to, the results of our study and the performance comparison of the built portfolio.

5.1 Data used

Since our study focuses on individual-accessible daily price data, we choose 49 widely-known indexes with available daily price data, which are listed in Appendix. Considering missing data, the time span of these 49 index's daily price data is from January 2000 to July 2020.

Our work is combined with preliminary work like investment pool selection. Once the investment pool is determined, one can follow the framework to use our prediction model.

5.2 Preliminary part: analysis of traditional methods

5.2.1 Data Preparation

Here in this part of traditional methods, we simplify the analysis by choosing 6 out of the 49 indexes to form a investment pool, which are:

1) Russell 2000 Total Return
2) iShares Russell Mid-Cap ETF
3) Bloomberg Commodity Index Total Return
4) Bloomberg Barclays US Aggregate Bond Index
5) MSCI World ex USA total net return
6) Bloomberg Barclays Global High Yield Total Return Index Value Unhedged.

These 6 indexes are selected because they are widely picked by funds and diversified. So in our empirical study, we aim to predict the price of these 6 indexes in a month and use these data to optimize the weight of each index to construct portfolios.

5.2.2 Train-Test set split and cross-validation

As whole dataset covers a time range of nearly 20 years, we decide to follow the 80/20 split rule to set training-set. For each round, we would use the next month's data as the test-set.

In the part where only historical data was used, we directly add one month's data into the training set, and use the month after as the test set again until July 2020. While in ARIMA part, we use the predicted prices as the new data and add them into the previous training dataset. So now the new training set consists of both real and forecasted price information. Similarly, we repeat this process until the last month of the whole dataset.

5.2.3 Model Construction

In order to make analysis of these indexes' price data and to help find the appropriate parameter of the ARIMA model, we first plotted the autocorrelation of differencing for the base training dataset.

From the above plots, we can see that the time series is stationary at lag = 1, since for almost all these indexes, the first lag is above the significance level. Therefore, we would try d = 1 in our model.

For parameter p and q, it's not easy to find out the best value directly from the plots. Thus we tried a few different values. However, changing these parameters didn't improve the performance and no matter what values we chose, the model gave a bad result. In the final ARIMA, we used p = 5 and q = 1.

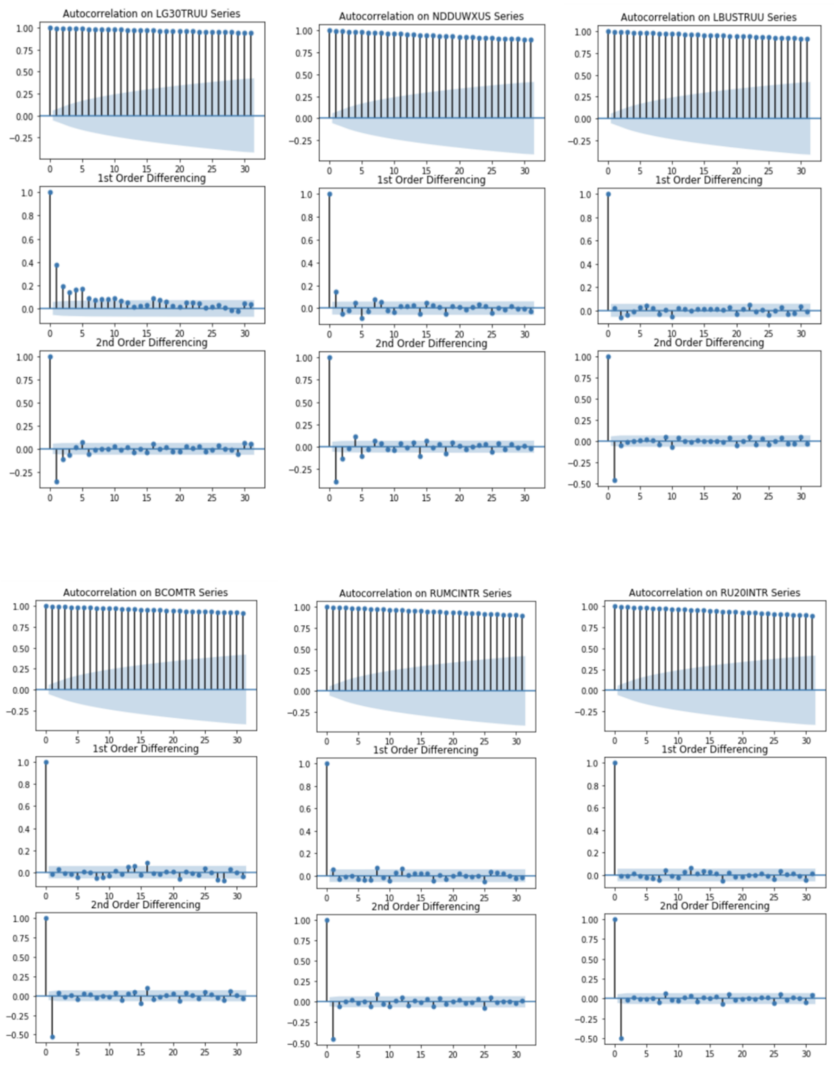

Figure 4: Autocorrelation function

5.2.4 Prediction results

The plot below shows the comparison between the actual return calculated using historical data and the ARIMA predicted return. We transformed all the returns to annual ones for comparison. The ARIMA predicted return shows very slight volatility and changes almost around the mean while the actual returns show more obvious fluctuations. MSE = 7.15 and MAPE = 4.88% for the price series in this case.

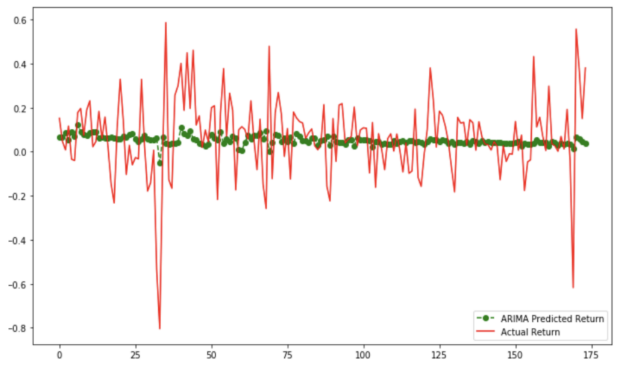

Figure 5: Predicted_Actual return

5.3 Machine learning part

5.3.1 Data preparation

We mainly do three things to the raw daily price data of 49 indexes.

The first thing is to normalize the price data to return, so that the task becomes predicting percent change instead of prices. As the input_size = 1, actually we turn price into daily return. This step is important because price would easily move out-of-scale in the test set when price reaches historical high or low. In that way, a model predicting price would show poor work.

Source code: normalization of price data

```
        if self.normalized:
            seq = [seq[0] / seq[0][0] - 1.0] + [
                curr / seq[i][-1] - 1.0 for i, curr in enumerate(seq[1:])]
```

Secondly, we set the data into the size of input and output, so that it can be fed into our LSTM model.

The last thing that we do is to encode the symbol of indexes into integer labels, and put them into an embedding vector. An alternative way is to use one-hot encoding. But here we find the embedding vector would be more compressed compared to the one-hot encoding sparse matrix, so we take use of embedding vector.

5.3.2 Model construction

We use TensorFlow to build LSTM cells and construct the network. Some of the API may get deprecated in the future and keras may be used instead.

Source code: LSTM network

```
        def _create_one_cell():
            lstm_cell = tf.contrib.rnn.LSTMCell(self.lstm_size, state_is_tuple=True)
            lstm_cell = tf.contrib.rnn.DropoutWrapper(lstm_cell,
                output_keep_prob=self.keep_prob)
            return lstm_cell

        cell = tf.contrib.rnn.MultiRNNCell(
            [_create_one_cell() for _ in range(self.num_layers)],
            state_is_tuple=True
        ) if self.num_layers > 1 else _create_one_cell()
```

And we use TensorBoard to visualize our design and training process.

5.3.3 Tuning Parameter

We mainly work on tuning *learning_rate*, *number_of_layer*, *lstm_size*, *batch_size* in our model. When tuning parameters, we use BCOMTR(Bloomberg Commodity Index Total Return) as our prediction target to see how prediction error changes.

For *learning_rate*, we first use an adaptive algorithm to make *learning_rate* change with epoch trained. Then we set *learning_rate* as fixed value.

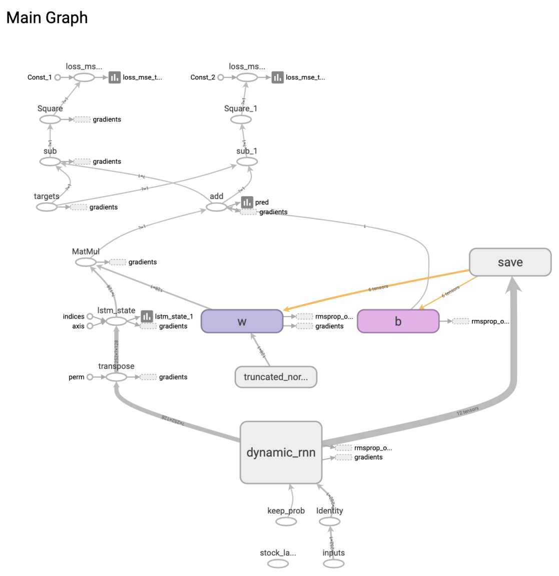

Figure 6: TensorBoard information flow

Source code: Learning_rate calculation

```
def _compute_learning_rates(config=DEFAULT_CONFIG):
    learning_rates_to_use = [
        config.init_learning_rate * (
        config.learning_rate_decay ** max(float(i + 1 - config.init_epoch), 0.0))
        for i in range(config.max_epoch)
                    ]
```

For *number_of_layer*, we try the set [1, 2] as three or more layers would not be necessary for LSTM. For *lstm_size*, we try the set[32, 64, 128]. Larger *lstm_size* may help to reduce prediction error, here we take 128 as the maximum amount according to our training time limit.

Table 1: MSE result

MSE		lstm_size		
		32	64	128
number_of_layer	1	0.00169	0.00122	0.00068
	2	0.00202	0.00137	0.00074

According to the training result, we would choose *number_of_layer* = 1 and *lstm_size* = 128.

For *batch_size*, we choose *batch_size* = 64 considering our GPU capacity and efficiency.

5.3.4 Prediction results

Here we list the test-set plot. From the plot we can see that model predicted return shows less volatility for most of the indexes, but it's better than ARIMA in plain sense. For now it's hard to say whether it's a good thing to give predictions that indicates less volatility. And we must view these six predictions as an integrity as we need to construct a portfolio by the predictions. We'll later put the data into PyPortfolioOpt to compare the performance of constructed portfolio.

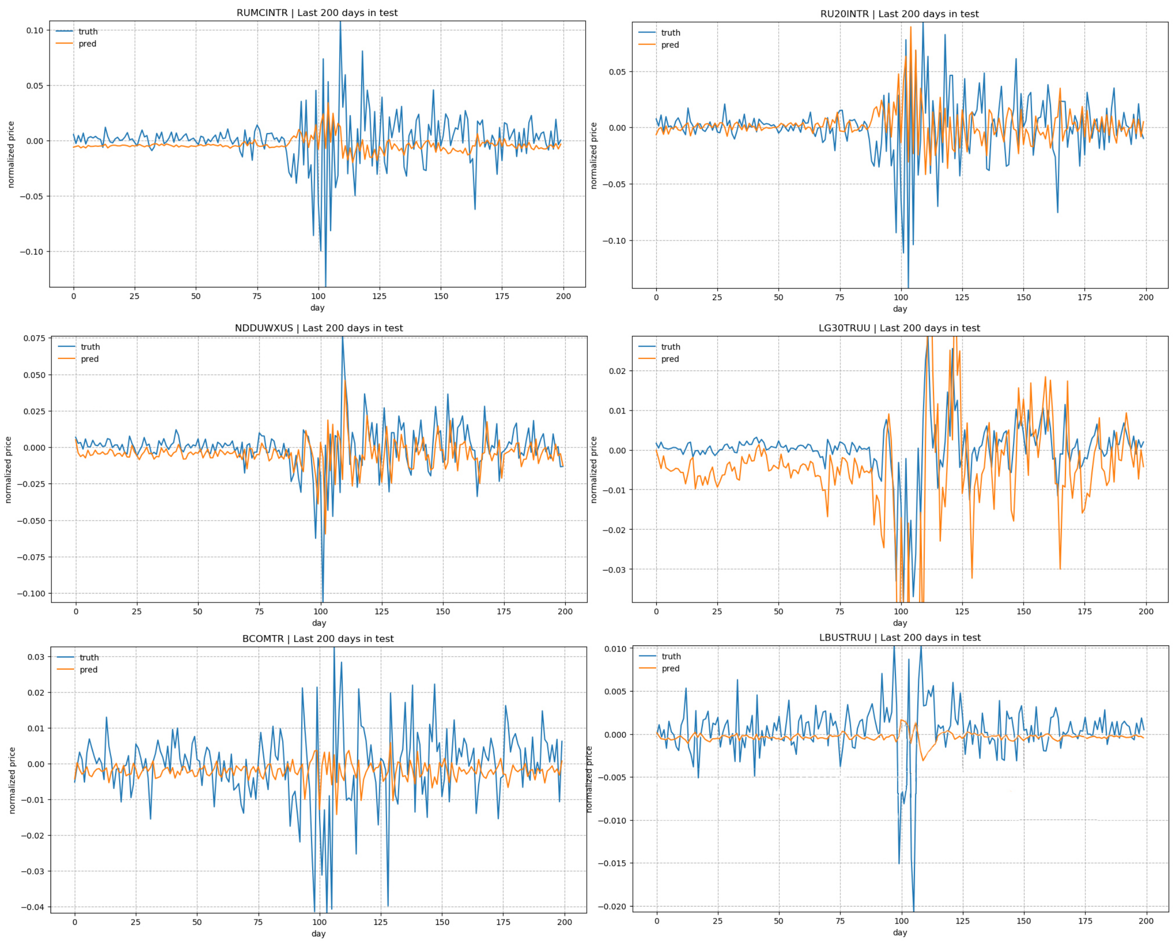

Figure 7: Predicted return

5.4 Comparison results

In this part, we first transfer return back to price data, then use PyPortfolioOpt package to decide weight allocation, and show the cumulative return, sharpe ratio and max drawdown of the tree strategy.

Before comparison, we must point out that when using traditional methods, we predict one calendar month's return. While in the RNN part, we predict index return on a 30-day scale, which is not quite the same. The difference between position adjustment may lead to some difference in cumulative return.

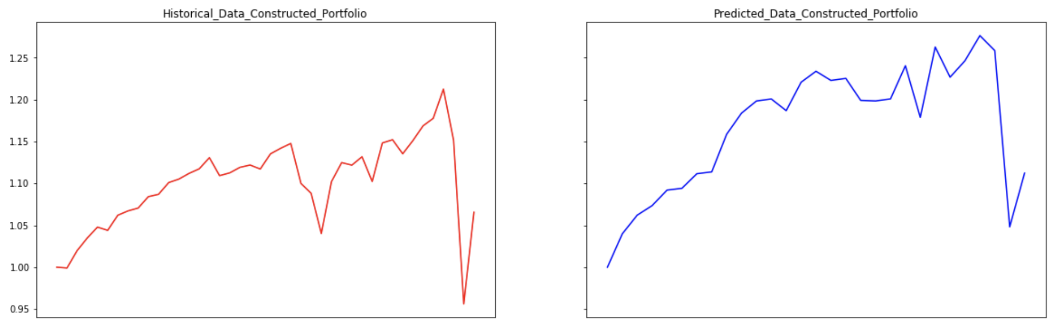

Figure 8: Cumulative return_1

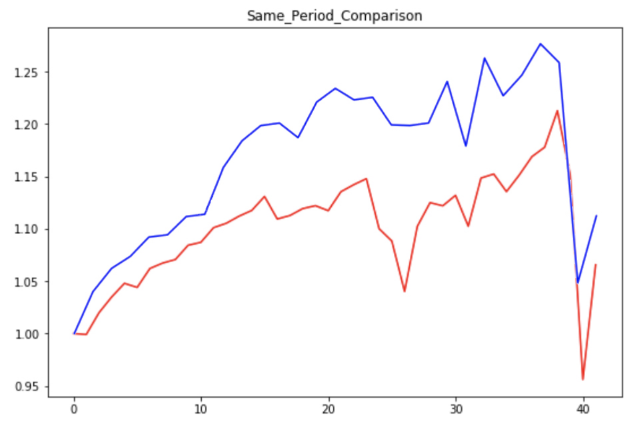

Figure 9: Cumulative return_2

We can see that for the 40-month test period, LSTM_portfolio shows better performance than historical_data_built_portfolio. If we set rf = 0.02, Sharpe Ratio and max drawdown 0.27 for LSTM_portfolio are and -16.7%, while for historical_data_built_portfolio the values are -0.2 and -17.0%. If we don't consider 2020's stock disaster and truncate the test set to March 2020, then Sharpe Ratio would be 0.87 and 0.45 correspondingly.

Our model surpass traditional method from two aspects.

Firstly, our model give predictions for future return more prudently at some time, as it would give negative return predictions for all of the indexes listed. In that way, the best investment should be to choose risk-free assets, and we manually set return as 0.002 for that 30-day period. We can see for that reason the cumulative return plot is more smooth for LSTM_portfolio and it suffers less loss until 2020 spring, when stock market crashed and the net value of both portfolios tumble down. While traditional method would give positive expected returns for most indexes, portfolio may suffer loss more often.

Secondly, LSTM-built-portfolio also performs better when the trends are up. This should also own to the relatively precise prediction of expected return. If simply taking historical return as expected return, past information takes same weight as newer information, which may not be the case in real-world trading. LSTM makes full use of both long-term and short-term information, and tends to give better prediction.

6 Conclusions

6.1 Main conclusions of our study

We use LSTM cells to build a RNN model to predict index price in monthly scale. We construct portfolio based on predicted price and back_test its performance on test set. LSTM_portfolio shows better performance than historical_data_built_portfolio from 2016 to 2020.

If we set rf = 0.02, Sharpe Ratio and max drawdown 0.27 for LSTM_portfolio are and -16.7%, while for historical_data_built_portfolio the values are -0.2 and -17.0%. If we get rid of the effect of 2020's stock disaster, then Sharpe Ratio would be 0.87 and 0.45 correspondingly.

6.2 Discussions of further improvement

Firstly, choosing between parameter set: $input\_size = 1$, $num\_steps = 252$ and $input\_size = 30$, $num\_steps = 12$ is a hard step to make. We take the first in our empirical work as the second setting would be much time-consuming. A compromising solution is to use monthly data and use LSTM on a monthly-scale. In that way, LSTM's ability to capture long-term information would also work well as monthly price data tends to be more smooth and the pattern may be easier to get.

Secondly, when using Max_Sharpe method in PyPortfolioOpt, the weights often tend to look like[1,0,0,0,0,0], which means according to the method, we should allocate funds on single index, which is not diversified and potentially risky. We could make some progress in adding constraints on weight, use L1 or L2 regularization, which may lift performance of constructed portfolio.

Thirdly, in our paper we simply compare portfolios' performance between LSTM and histocial baseline model. If one wants to step further in fine-tuning the model and apply the algorithm in real-world trading, more statistical methods like hybrid ARIMA-GARCH and neural network methods like CNN should be used as benchmarks.

Lastly, our model could be further modified to predict daily stock return. The framework could also apply to prediction use monthly data like accounting information and some widely-used factor data.

7 Appendix

7.1 Dataset list

Name	Description	Name	Description
BCOMTR	Bloomberg Commodity Index Total Return	RU20VATR	iShares Russell 2000 Value ETF
LBUSTRUU	Bloomberg Barclays US Aggregate Bond Index	RUMCINTR	iShares Russell Mid-Cap ETF
LG30TRUU	Bloomberg Barclays Global High Yield Total Return Index Value Unhedge	RUMRINTR	iShares Micro-Cap ETF
LMBITR	Bloomberg Barclays Municipal Bond Index Total Return Index Value Unhedged USD	RUTPINTR	iShares Russell Top 200 ETF
NDDUE15X	Amundi MSCI Europe Ex UK Ucits ETF Dr	S5COND	S&P 500 Consumer Discretionary Index
NDDUJN	MSCI Japan Index	S5CONS	S&P 500 Consumer Staples Index
NDDUNA	iShares MSCI North America UCITS ETF	S5ENRS	S&P 500 Energy Index
NDDUPXJ	MSCI Pacific ex Japan UCITS ETF	S5FINL	S&P 500 Financials Sector GICS Level 1 Index
NDDUUK	iShares MSCI UK ETF	S5HLTH	S&P 500 Health Care Index
NDDUWXUS	MSCI World ex USA total net return	S5INDU	S&P 500 Industrials Index
NDUEEGF	SPDR MSCI Emerging Markets UCITS ETF	S5INFT	S&P 500 Information Technology Index
RU10GRTR	iShares Russell 1000 Growth ETF	S5MATR	S&P 500 Materials Index
RU10VATR	iShares Russell 1000 Value ETF	S5RLST	S&P 500 Real Estate Index
RU20GRTR	iShares Russell 2000 Growth ETF	S5TELS	S&P 500 Communication Services Index
RU20INTR	Russell 2000 Total Return	S5UTIL	S&P 500 Utilities Index
LUICTRUU	Bloomberg Barclays U.S. Intermediate Credit Total Return Index	MXEA	MSCI EAFE Index
LULCTRUU	Bloomberg Barclays U.S. Long Credit Index	MXEF	MSCI Emerging Markets Index
M1CXBRU	iShares Core MSCI International Developed Markets ETF	MXUSMVOL	MSCI USA Minimum Volatility Index
M1USMVOL	MSCI USA Minimum Volatility (USD) Index	MXWD	MSCI All Countries World Index
M2US000$	iShares Edge MSCI USA Momentum Factor ETF	MXWOUIM	MSCI All Countries World Index
M2USEV	MSCI USA Enhanced Value Index	NDDUUS	MSCI Daily Total Return Net USA USD Index
M2USRWGT	MSCI USA Risk Weighted Index	SPX	S&P 500 Index

References

[1] A. V. Devadoss and T. A. A. Ligori, ”Forecasting of stock prices using multi layer perceptron”, Int J Comput Algorithm, vol. 2, pp. 440-449, 2013.

[2] Adamantios Ntakaris, Giorgio Mirone, Juho Kanniainen, Moncef Gabbouj, Alexandros Iosifidis, ”Feature Engineering for Mid-Price Prediction With Deep Learning”, Access IEEE, vol. 7, pp. 82390-82412, 2019.

[3] Kai Chen ; Yi Zhou ; Fangyan Dai, “A LSTM-based method for stock returns prediction: A case study of China stock market”, Access IEEE, DOI: 10.1109/BigData.2015.7364089

[4] S.; Vinayakumar, R.; Gopalakrishnan, E.A.; Menon, V.K.; Soman, K.P. Stock price prediction using LSTM, RNN and CNN-sliding window model. In Proceedings of the 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, India, 13–16 September 2017; pp. 1643–1647

[5] Derek Snow, “Machine Learning in Asset Management”, The Journal of Financial Data Science Winter 2020, 2 (1) 10-23; DOI: https://doi.org/10.3905/jfds.2019.1.021

[6] Gu, Shihao and Kelly, Bryan T. and Xiu, Dacheng, Empirical Asset Pricing via Machine Learning (September 13, 2019). Chicago Booth Research Paper No. 18-04, Available at SSRN: https://ssrn.com/abstract=3159577

[7] Holmberg, D. (2020, June 11). ARIMA Forecasting in Python. Retrieved August 26, 2020, from https://towardsdatascience.com/arima-forecasting-in-python-90d36c2246d3

[8] Zachary C. Lipton, ”A Critical Review of Recurrent Neural Networks for Sequence Learning”, arXiv preprint arXiv:1506.00019, 2015

[9] Rather Akhter Mohiuddin, Agarwal Arun and V. N. Sastry, ”Recurrent neural network and a hybrid model for prediction of stock returns”, Expert Systems with Applications, vol. 42, no. 6, pp. 3234-3241, 2015.

[10] Sepp Hochreiter and Schmidhuber Jürgen, ”Long short-term memory”, Neural computation, vol. 9, no. 8, pp. 1735-1780, 1997

[11] S. Siami-Namini and A. S. Namin, ”Forecasting economics and financial time series: ARIMA vs. LSTM”, arXiv:1803.06386, 2018, [online] Available: https://arxiv.org/abs/1803.06386.

[12] S. Siami-Namini and A. S. Namin, ”Forecasting economics and financial time series: ARIMA vs. LSTM”, arXiv:1803.06386, 2018, [online] Available: https://arxiv.org/abs/1803.06386.