Deep learning for portfolio management

Long Shen[*]
longshen@stanford.edu

Abstract

Everyone is eager to make money from stock market. But the trend of stock price is full of randomness and noise. Who want to predict it will finally get a low accuracy results when they apply their work in the real world. In this paper, I will use LSTM to predict portfolio directly. With this end to end design, we can lower risk by a more diverse investing and make better use of correlation among stocks.

1 Introduction

Our model is to predict the portfolio we should buy. For convenience, we assume there is no slippage and taxes. We will sell all stocks every night and buy the stocks every morning depends on the prediction.

The main input to our algorithm is the stock data, including date,volume,open,close,high,low,adjclose). Also it has fixed input to get contextual information. We use 1DConv to process input. Because when the input dimension is large, the 1DConv can reduce the dimension and let us train model fast. Then we stacked LSTM and some denses to process output. We have two outputs, one is used to predict price(linear regression) and another is used to predict portfolio(softamx). The two output share the same lstm and some denses. We want to the portfolio prediction task can learn some information form price prediction task. Also, we add extra fixed input and combined it with the LSTM output. Because we want to the model can learn some contextual features.

2 Related work

LSTM is a state of art model to handle the problem of time series. There are a lot of work using various kinds of LSTM[1] to predict stock price. For example, Li, Hao, Yanyan Shen, and Yanmin Zhu they used multiple input[2]. Bao, Wei, Jun Yue, and Yulei Rao used stacked LSTM. Also, there are a lot of variations of LSTM we can apply them on stock prediction. For example, DARNN[4], It introduces a novel input attention mechanism that can adaptively select the relevant driving series. In the decoder, a temporal attention mechanism is used to automatically select relevant encoder hidden states across all time steps. Graphical RNN[5] It used the same RNN for time series in the same class. Also, it encoded all hidden status of nodes in the same graph and concatenate the output of every graph then feed them as input.

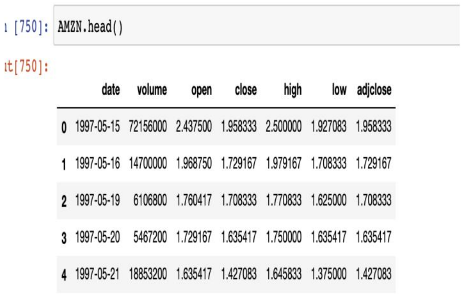
However, I find these papers focus on price prediction. But what do we really care in real world is the total earring by investing. Depends on intuition sense, we can know although we have a low accuracy(53-57percentage) for the going up or down of an individual stock, we can still have a positive earning because a proper portfolio can offset these randomness. Especially when the market is going up. If our portfolio's earnings can exceed the SP index, it would be useful.

[*]Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

Also, I find some previous teams are working on long term prediction. But it's meaningless to predict the stock price for a long term. Because major is fed by short term dataset, all information will vanish after a certain time. Moreover, it's unnecessary to do a long term prediction, because we can take action right now. The earning is same if the price increase 1 dollar tomorrow or in the next year.

3 Dataset and Features

For convenience, we select 4 stocks as the original data from SP 500, they are FB, GOOGL, MSFT and AMZN.[6] We have 1085 training examples and 510 testing examples which date range is from 2012-05-18 to 2018-11-02. Every example has 32 timestep(window size) and 3 inputs. The main input is the volume,open,close,high,low,adjclose of 4 stocks. The discrete input is the date, which don't need to predict. Also we have an fixed input to handle contextual features(e.g. earning report). I normalization the main input data and scale the discrete and fixed input under 1. The stock data of amazon.

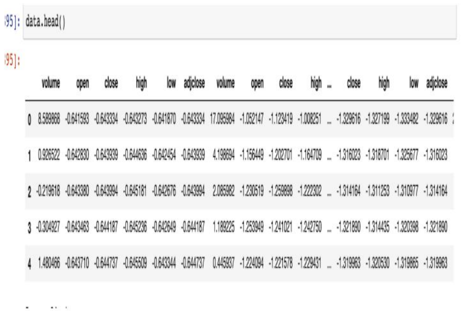


```
n [750]: AMZN.head()
```

```
t[750]:
```

	date	volume	open	close	high	low	adjclose
0	1997-05-15	72156000	2.437500	1.958333	2.500000	1.927083	1.958333
1	1997-05-16	14700000	1.968750	1.729167	1.979167	1.708333	1.729167
2	1997-05-19	6106800	1.760417	1.708333	1.770833	1.625000	1.708333
3	1997-05-20	5467200	1.729167	1.635417	1.750000	1.635417	1.635417
4	1997-05-21	18853200	1.635417	1.427083	1.645833	1.375000	1.427083

Main input after combination and normalization

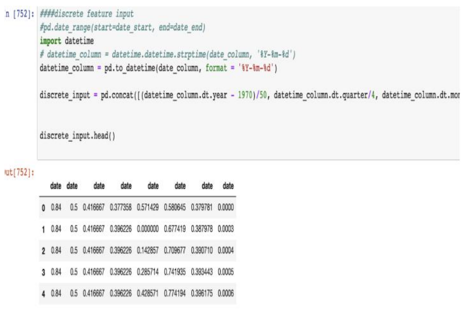


```
95]: data.head()
```

```
95]:
```

	volume	open	close	high	low	adjclose	volume	open	close	high	...	close	high	low	adjclose
0	8.589868	-0.641593	-0.643334	-0.643273	-0.641870	-0.643334	17.095984	-1.052147	-1.123419	-1.008251	...	-1.329616	-1.327199	-1.333482	-1.329616
1	0.926522	-0.642830	-0.643939	-0.644636	-0.642454	-0.643939	4.198694	-1.156449	-1.202701	-1.164709	...	-1.316023	-1.318701	-1.325677	-1.316023
2	-0.219618	-0.643380	-0.643994	-0.645181	-0.642676	-0.643994	2.085982	-1.230519	-1.259898	-1.222302	...	-1.314164	-1.311253	-1.310977	-1.314164
3	-0.304927	-0.643463	-0.644187	-0.645236	-0.642649	-0.644187	1.189225	-1.253949	-1.241021	-1.242750	...	-1.321890	-1.314435	-1.320398	-1.321890
4	1.480466	-0.643710	-0.644737	-0.645509	-0.643344	-0.644737	0.445937	-1.224094	-1.221578	-1.229431	...	-1.319963	-1.320530	-1.319865	-1.319963

Discrete input is year, quarters, months, weeks, dayofweek, dayofmonth, dayofyear, total days since 1970. Which have been scaled.



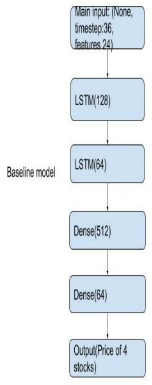
```
n [752]: ####discrete feature input
         #pd.date_range(start=date_start, end=date_end)
         import datetime
         # datetime_column = datetime.datetime.strptime(date_column, '%Y-%m-%d')
         datetime_column = pd.to_datetime(date_column, format = '%Y-%m-%d')

         discrete_input = pd.concat([(datetime_column.dt.year - 1970)/50, datetime_column.dt.quarter/4, datetime_column.dt.mon


         discrete_input.head()
```

```
ut[752]:
```

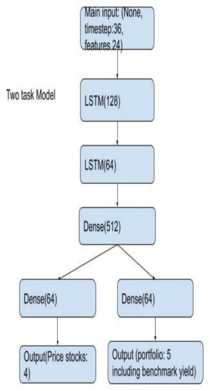
	date	date	date	date	date	date	date	date
0	0.84	0.5	0.416667	0.377358	0.571429	0.580645	0.379781	0.0000
1	0.84	0.5	0.416667	0.396226	0.000000	0.677419	0.387978	0.0003
2	0.84	0.5	0.416667	0.396226	0.142857	0.709677	0.390710	0.0004
3	0.84	0.5	0.416667	0.396226	0.285714	0.741935	0.393443	0.0005
4	0.84	0.5	0.416667	0.396226	0.428571	0.774194	0.396175	0.0006

4 Methods

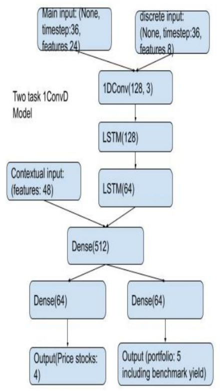
For baseline model, we used the following structure to predict the price. Then, only buy the one which has the largest daily return.


Main input: (None, timestep:36, features 24)
LSTM(128)
LSTM(64)
Baseline model
Dense(512)
Dense(64)
Output(Price of 4 stocks)

For two task model, we used the following structure. We will have two outputs. The key part of our is to define the target of portfolio prediction task. For this task, it's hard to define a true label. The simplest way is to select the one who have a highest increasing ratio as the true label. But we don't want to hard encoded it. For example, if stock A increase 20%, stock B increase 19% and stock C decrease 20%. If we hard encode it we will treat stock B and C same. So we can't learn enough information. The first method is to softmax these daily return(excluding decreasing ones) and set these as true labels and the loss is categorical cross entropy. The second method is to use daily return directly. We find the second ones have a better performance.


Main input: (None, timestep:36, features 24)
Two task Model
LSTM(128)
LSTM(64)
Dense(512)
Dense(64)
Dense(64)
Output(Price stocks: 4)
Output (portfolio: 5 including benchmark yield)

For two task model with 1DConv, we concatenate discrete input and main input, then feed them into 1Dconv. Also we will concatenate the contextual input with the output of LSTM and use this concatenate layer as the input of the next dense layer.


Main input: (None, timestep:36, features 24)
discrete input: (None, timestep:36, features 8)
1DConv(128, 3)
Two task 1ConvD Model
LSTM(128)
Contextual input: (features: 48)
LSTM(64)
Dense(512)
Dense(64)
Dense(64)
Output(Price stocks: 4)
Output (portfolio: 5 including benchmark yield)

5 Experiments/Results/Discussion

The epoches is 500. The timestep for every example is 32. benchmark yield is 1. Batch size is 64. The optimizer is Adam and learning rate 0.001. The filter size of 1dconv is 3. Also, I use drop out when training, the ratio is 0.2 because I'm afraid the information of input is not enough. All these hyperparameters can be adjusted to have a better result. But for me, they are good enough The metrics I used to measure model is the annually return and sharpe ratio. The Sharpe ratio is used to help investors understand the return of an investment compared to its risk.

The Formula for Sharpe Ratio Is

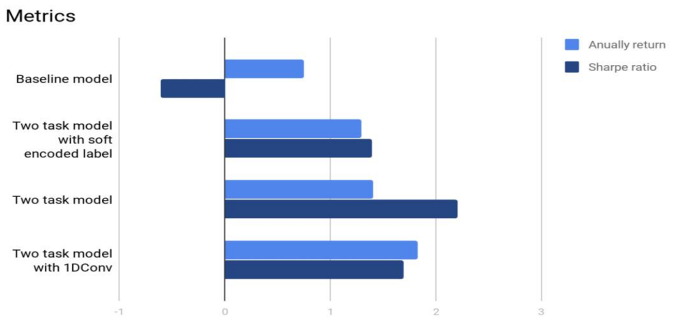
$$\textit{Sharpe Ratio} = \frac{R_p - R_f}{\sigma_p}$$

where:

R_p = return of portfolio

R_f = risk-free rate

σ_p = standard deviation of the portfolio's excess return



6 Conclusion/Future Work

The model can perform very well. The yearly return can achieve 1.83 and sharpe ratio can achieve 1.70, which is much better than baseline model. But we still can do better. First, it's an ideal environment(without slippage and taxes). To apply this model in the future, the next step is to add these restrictions, so we can simulate the real world. Second, we can introduce graphic algorithm to capture relationships among stocks better. So we can speed up our algorithm when input hundreds of stocks. Third, because of the instability of stocks, more recent example are more important. Swe can also introduce weight decay for these examples. Forth, we can have a longer time step for examples, so the model can capture more information from historical data. Fifth, we can introduce other external data other than stock data.

References

[1] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[2] Li, Hao, Yanyan Shen, and Yanmin Zhu. "Stock Price Prediction Using Attention-based Multi-Input LSTM." Asian Conference on Machine Learning. 2018.

[3] Bao, Wei, Jun Yue, and Yulei Rao. "A deep learning framework for financial time series using stacked autoencoders and long-short term memory." PloS one 12.7 (2017): e0180944.

[4] Qin, Yao, et al. "A dual-stage attention-based recurrent neural network for time series prediction." arXiv preprint arXiv:1704.02971 (2017).

[5] Ashish Bora, Sugato Basu, Joydeep Ghosh. “Graphical RNN Models” Neural and Evolutionary Computing (2016).