Deep Learning for ETF Price Prediction

Dickson Chan, Michael Hsieh, Sophie Pan
kingsan@stanford.edu, mhsieh33@cs.stanford.edu, yuepan@stanford.edu

Motivation

We are addressing the problem of predicting midterm asset price movements by looking at historical prices of ETFs. The premise of our study is that information reflecting the rise and fall of these different market segments can predict future price movements in other market segments. For example, the increase in the price of a US oil ETF may precede a fall in the price of a consumer spending ETF. We aim to apply deep learning models to model these predictive relationships between different market segments.

Data and Preprocessing

We have obtained historical price time series information from a universe of ETFs in 9 different categories:

Categories		
Industry	Equity Sector	Volatility
Fixed Income	Commodity	US State
Currency	Country	Market Cap

The raw data is in the form of closing prices of the day of 133 ETFs from 20141124 to 20191010. Instead of working with the raw data, we have transform the data by computing 1-day backward percentage change. Then, we have decided to use a time period of 60 days to try to predict the 1-week forward percentage change of the ETFs. The model is being trained by the first 85% of the data and tested on the last 15%. The inputs and outputs of our model can be summarized in the table below:

Table 1: Summary of the inputs and outputs of the models

Item	Matrix size
test N	125
train N	984
feature time period	60
# of ETF	133
feature size	$60 \times 133 = 7980$
X_test shape	(125, 60, 133)
Y_test shape	(125, 133)
X_train shape	(984, 60, 133)
Y_train shape	(984, 133)

Price Change Prediction Results By Category

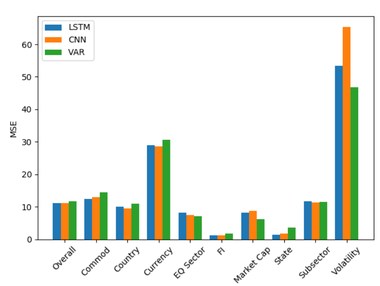

Figure 1: Mean Standard Error By Category. We see that our two neural networks models perform very similarly. We compare versus a VAR time series model.

Models

Table 2: LSTM Implementation. We first trained a 3 layer deep 50 neuron LSTM network. We added two Dropout layers to prevent overfitting.

Layer	Neurons/Rate
LSTM	50
Dropout	0.2
LSTM	50
LSTM	50
Dropout	0.2
Dense	133

Table 3: CNN Implementation. Next, we trained a convolutional neural network. Each Conv1D layer had 3 kernels, each MaxPool1D layer had 2 units, and finally the output was flattened and fed into two dense layers.

Layer	Kernel Size	Units
Conv1D	3	10
MaxPool1D		2
Conv1D	3	10
MaxPool1D		2
Flatten		
Dense		500
Dense		133

Hyperparameter Tuning Results

Table 4: MSE for LSTM and CNN models across different configurations. We had 984 samples in the training set and 125 samples in the test set.

Type	Neurons/Kernels	Training MSE	Test MSE
VAR	N/A	8.62834	11.76532
LSTM	50	5.16845	11.170467
LSTM	200	3.422018	12.114844
LSTM	500	2.379474	13.745019
CNN	3	5.16845	11.170467
CNN	6	3.422018	12.114844
CNN	9	2.379474	13.745019

Price Change Prediction Results By Date

We also examined the MSE and MPA across the dates in the test set.

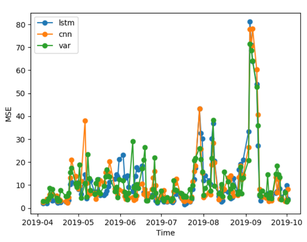

Figure 2: Mean Standard Error By Date

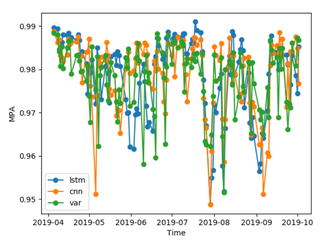

Figure 3: Mean Prediction Accuracy By Date

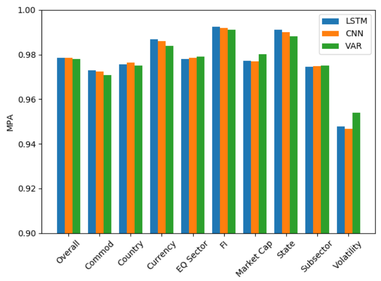

Figure 4: Mean Prediction Accuracy By Category.

$$MPA(T,L) = 1 - \frac{1}{T}\sum_{t=1}^{T}\frac{1}{L}\sum_{l=1}^{L}\frac{|X_{t,l} - \hat{X}_{t,l}|}{100},$$

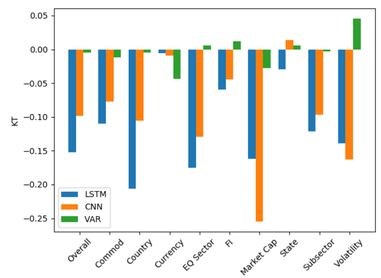

Figure 5: Kendall Tau By Category.

$$\tau = \frac{(P-Q)}{\sqrt{(P+Q+T)*(P+Q+U)}},$$

where P is the number of concordant pairs, Q the number of discordant pairs, T the number of ties only in the actual percent price changes, and U the number of ties only in the predicted percent price changes.

Reinforcement Learning Architecture

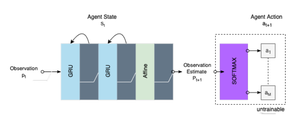

Figure 6: Model-based RNN Architecture. Two layer gated recurrent unit recurrent neural network; receives log returns ρ_t as input, builds internal state s_t and estimates future log returns. Regularized mean squared error is used as the loss function, optimized with the ADAM adaptive optimizer.

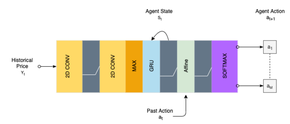

Figure 7: Model-free Policy Gradient (PG) architecture. $\rho_{T-t}t \in R^{M\times T}$ are passed throw two 2D-convolution layers, which generate a feature map, which is, in turn, processed by the GRU state manager. The agent state produced is combined (via matrix flattening and vector concatenation) with the past action to estimate action-values $q_1, \ldots, q_M$

References

Li, et al. (2019) Risk Management via Anomaly Circumvent: Mnemonic Deep Learning for Midterm Stock Prediction

Jiang, et al. (2019) A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem

Reinforcement Learning Portfolio Management Agent

Part of the project aim is to investigate the effectiveness of context-independent Reinforcement Learning agents on **continuous sequential asset allocation**.

Trading Agents Comparison Matrix: Commodity ETF				
	Reward Generating Functions	Cumulative Returns (%)	Sharpe Ratio (%)	Max Drawdown (%)
Model-based	VAR	7.64	0.502	13.85
	RNN	7.81	0.515	13.76
	LSTM	5.15	0.373	14.67
Model-free	PG	17.7	0.717	17.9
Benchmark	Quadratic	16.3	0.440	26.8
	Momentum	4.91	0.217	15.1
	Random	7.83	0.299	13.7

Table 5: Comprehensive comparison of evaluation metrics of reinforcement learning trading algorithms and their variants on test data. Improvement 9.2% in annualized cumulative returns and 13.4% in annualized annualized Sharpe Ratio, compared to the model-based reinforcement learning models. Qudratic agent is a mean-variance optimizing agent.

Experimentation: Built classes compatible with OpenAI Gym. Conduct 30 experiments across 3 ETF categories (*commodity*, *currency*, *industry* and hyper-parameters combinations.

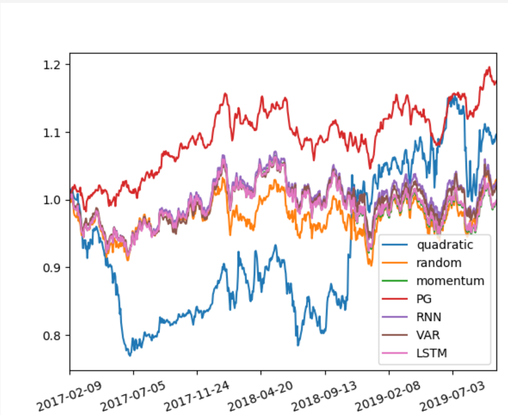

Figure 8: Comparison of portfolio value for model-based learning algorithms and model-free algorithm by PG. The agent is trained on market data for **14 commodity ETFs**. Training period: **2014-07-18 to 2016-12-31**. Testing period: **2017-01-01 to 2019-10-10**.

Conclusions

- When model-free **PG algorithm** converges, it performs better to model-based algorithms. However, the algorithm requires **stationary** transition, thus can lead to failure to of finding globally optimal strategy.
- DDPG agent is **degenerate** in most of our experiments, (not included in the chart) which often tends to buy only one asset at a time. This indicates more modifications are needed for designing promising algorithms.