Deap Learning for ETF Price Prediction

Dickson Chan Michael Hsieh Sophie Pan

Abstract

We applied Long-Short Term Memory (LSTM) and Convolutional Neural Network (CNN) models on exchange traded fund close price data to predict future one week prices. We also applied these models in a reinforcement learning framework to train a portfolio management agent and compared it to a model-free agent. Overall, our price prediction results in terms of mean standard error and mean prediction accuracy for the deep learning models were slightly better than our baseline VAR model. When the model-free reinforcement learning algorithm converged, it performed better than our model-based algorithms.

1. Introduction

We are addressing the problem of predicting midterm asset price movements by looking at historical prices of ETFs. Exchange traded funds (ETFs) are of importance in the investment landscape as they allow investors to obtain exposure to a portfolio of companies or assets in a certain market segment. ETFs are passive investments where their constituent investment allocations are relatively stable. Therefore, prices of ETFs serve as a proxy for the ebb and flow of the market segment that the ETF covers. We have obtained historical price time series information from a universe of ETFs in 9 different categories:

- Industry
- Equity Sector
- Volatility
- Fixed Income
- Commodity
- US State Municipal Bonds
- Currency
- Country
- Market Cap

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. Very few research articles we reviewed apply ML to ETFs or to low-to-mid frequency trading. Our hypothetical trader is a "weekend" investor who is interested in applying ML algorithms to investing with a midterm horizon within more than 5,000 ETFs that are currently available globally. We also extend our modeling to address the question of how to optimize risk and return in a diversified portfolio of ETFs.

Table 1. LSTM Implementation

LAYER	NEURONS/RATE
LSTM	50
DROPOUT	0.2
LSTM	50
LSTM	50
DROPOUT	0.2
DENSE	133

2. Dataset and Features

The dataset we used was the daily close prices between 11/24/2014 and 10/10/2019 for 133 ETFs in the categories listed before. The inputs to our models were 60 prior days of close percent price changes for all 133 ETFs. The output of the models was the forward 1-week (5-day) close percent price change for the ETFs. The dataset was split 85% for the training set and 15% for the test set. We had 984 samples in the training set and 125 samples in the test set.

3. Methods

We decided to explore the use of recurrent neural networks (RNN) and convolutional neural networks (CNN) because they have the ability to extract features without manual feature engineering. They are also capable of using those features to create informative representation of the data.

While RNN is capable of learning the relationship between past data and present data of a time series, it often has difficulties handling long term dependencies. LSTM addresses this problem by storing, forgetting and recalling information it deems more important.

We trained a 3-layer deep 50-neuron LSTM network. We chose the LSTM network because it possesses the potential to resolve difficulties in modeling long sequences. The network was set up to forecast the 1-week return for all 133 ETFs. Table 6 shows the configuration of our LSTM model.

Table 2. CNN Implementation

LAYER	KERNEL SIZE	UNITS
CONV1D	3	10
MAXPOOL1D		2
CONV1D	3	10
MAXPOOL1D		2
FLATTEN		
DENSE		500
DENSE		133

Table 3. MSE for LSTM and CNN models across different configurations

TYPE	NEURONS/KERNELS	TRAIN MSE	TEST MSE
VAR	N/A	8.63	11.77
LSTM	50	5.17	11.17
LSTM	200	3.42	12.11
LSTM	500	2.38	13.75
CNN	3	5.17	11.17
CNN	6	3.42	12.11
CNN	9	2.38	13.75

We also explored using convolutional neural network to forecast the 1-week returns of the ETFs. The configuration of the CNN network is shown in Table 2. Conv1d layers are convolution layers with kernel sizes of 3. MaxPool1d layers have pool sizes of 2. And finally the output is flattened and fed into a fully connected network.

Table 3 shows the overall mean squared error from our models as we tuned our hyperparameters. Our MSE on test data was much higher than our MSE on training data which reflected overfitting of all our models including our baseline VAR model. Overall, the less complexity we had the better the deep learning networks performed.

3.1. Price Change Prediction Results By Category

Figure 1 shows the mean standard error for our entire test set across all 133 ETFs as well as divided into the different categories of ETFs. We see that our two models perform very similarly except for the Volatility ETF category where the CNN performs significantly worse. Overall, the deep learning networks performed slightly better than the baseline VAR model.

Next, we calculate mean prediction accuracy which is defined as

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

where T is the overall time period, L is the number of ETFs included in the group, $X_{t,l}$ is the percent change in price of the l-th ETF from the t-th to the $t+1$-th day, and $\hat{X}_{t,l}$ is the predicted percent change in price on the corresponding time and ETF.

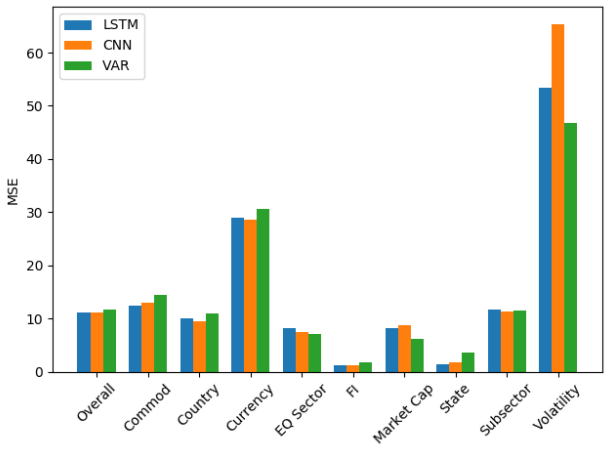

Figure 1. Mean Standard Error By Category

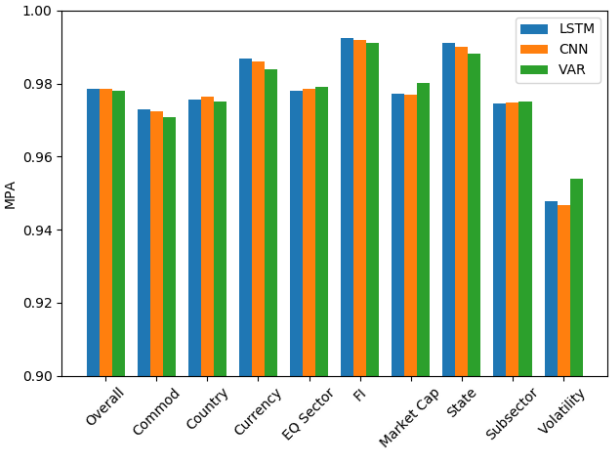

Figure 2. Mean Prediction Accuracy By Category

Figure 2 shows the mean prediction accuracy for all ETFs as well as divided into each category of ETFs as before. The T in the formula for mean prediction accuracy is our entire test period, and the L corresponds to the ETFs which make up each category. We see that both models perform very similarly in all categories in terms of mean prediction accuracy.

Lastly, we calculate the Kendall Tau statistic between the test price change percentages and the predicted price change percentages. The Kendall Tau statistic is defined as

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

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 (Kendall, 1945). Any pair of values $(X_i, \hat{X}_i)$ and $(X_j, \hat{X}_j)$, where $i < j$, are said to be concordant if the ranks of both elements agree: that is, both $X_i > X_j$ and $\hat{X}_i > \hat{X}_j$; or both $X_i < X_j$ and $\hat{X}_i < \hat{X}_j$. There are said to be discordant if $X_i > X_j$ and $\hat{X}_i < \hat{X}_j$; or $X_i < X_j$ and $\hat{X}_i > \hat{X}_j$.

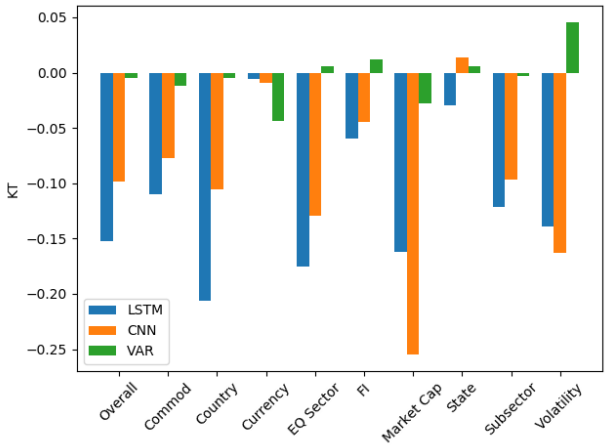

Figure 3. Kendall Tau By Category

Figure 3 shows the Kendall Tau statistics for each ETF category. We see that overall and in most categories the CNN model performs better than LSTM. We note that the CNN model in only the state municipal bond ETF category has a positive Kendall Tau statistic. Overall, the deep learning models underperformed the baseline VAR model in terms of Kendall Tau.

Looking at mean standard error, mean prediction accuracy, and Kendall Tau together we see that the Currency, Fixed Income, and US State Muni categories for ETFs were where our LSTM and CNN models performed the best.

4. Reinforcement Learning

We apply our findings in deep supervised learning to deep reinforcement learning (RL). We compare three classes of models, traditional asset allocation models, model-based (RL) models, and model-free (RL) models. Full model descriptions can be found in Appendix.

Our model-based RL algorithms has a untrainable softmax layer that translates the return predictions directly into portfolio weights. This setup allows us to build an agent with a statistical time series model such as Vector Auto Regressive Model as the reward generating functions. The other two underlying networks GRU-RNN and LSTM are also explored. Among the three modeled-based RL algorithms, RNN performs the best across most asset categories. In contrast, our Policy Gradient model addresses the continuous control problem in portfolio management more directly.

4.1. Reinforcement Learning Architecture

We first set up VAR Reinforcement Agent. The order-p VAR model, VAR(p), assumes that the underlying generating process: 1. Is covariance stationary; 2. Satises the order-p Markov property; and 3. Is linear, conditioned on past samples. Unsurprisingly, most of these assumptions are not realistic for real market data.

The limitations of the vector autoregressive processes are overcome by the recurrent neural network (RNN) environment model. Inspired by the effectiveness of recurrent neural networks in time-series prediction and the encouraging results obtained from the initial one-step predictive LSTM model in earlier Section, we investigate the suitability of GRU/RNN and LSTMs in the context of model-based reinforcement learning, used as environment predictors.

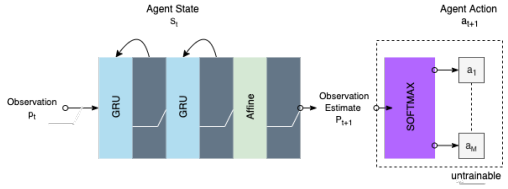

Figure 4. Model-based GRU-RNN Architecture. Two layer gated recurrent unit recurrent neural network; receives log returns ρ_t as input, builds internal state s_t and estimates future log returns. Regularized mean squared error is used as the loss function, optimized with the ADAM adaptive optimizer.

Figure 5. Model-free Policy Gradient (PG) architecture. $\rho_{T-t \to t} \in R^{M \times T}$ 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 $q_1, \ldots, q_M$

4.2. Reinforcement Learning Portfolio Management Agent

We set up our reinforcement learning agents as Algorithm 1 and 2.

Algorithm 1 General setup for model-based trading agents

Input: trading universes of M-assets
initial portfolio vector $w_1 = a_0$
initial asset prices p_0
loss function $\mathcal{L}$
historical dataset $\mathcal{D}$
Output: optimal model parameters θ_*
batch training on $\mathcal{D}$: $\theta \leftarrow argmax_\theta p(\theta|\mathcal{D})$
repeat
  for $t = 1$ **to** T **do**
    predict next state s_t
    observe tuple $< o_t, r_t >$
    get estimate of agent state: $s_t \approx f(\cdot, o_t)$
    calculate gradients: $\nabla_\theta \mathcal{L}(\hat{s}_t, s_t)$
    update model parameters θ {ADAM}
    plan and take action w_t
  end for
until convergence
set $\theta_* \leftarrow \theta$

Algorithm 2 Setup for model-free policy gradient trading agents

Randomly initialize actor $\mu(s|\theta)$
Initialize replay buffer R
for $i = 1$ **to** M **do**
  Receive initial observation state s:
  Add noise into the price data
  for $t = 1$ **to** T **do**
    Select action $\omega_t = \pi(s_t|\theta)$
    Execute action $_t$ and observe r_t , s_{t+1} and $'_t$
    save transition (s_t, w_t, w'_t) in R
  end for
  Update actor policy by policy gradient:
  $\nabla_\theta \mathcal{L} = \nabla_\theta \sum_{t=1}^{T} (\log(\omega_t \cdot p_t))$.
end for

4.3. Experiment Result

We ran pairwise correlations among the indexes and identified sectors where it would be able to easily pick out a portfolio of correlated indexes. We split our data-set into the partition of training and testing: training period: 2014-07-18 to 2016-12-31; testing period: 2017-01-01 to 2019-10-10.

We build trading agent classes compatible with OpenAI Gym and 30 experiments across 3 ETF categories (commodity, currency, and industry) and hyper-parameter combinations. The result of our experimentation for commodity sector is in Table 4.

Trading Agents Comparison Matrix: Commodity ETF				
	Model	Cumul. Returns (%)	Sharpe Ratio (%)	MMD (%)
MB	VAR	7.64	0.502	13.85
	GRU-RNN	7.81	0.515	13.76
	LSTM	5.15	0.373	14.67
MF	PG	17.7	0.717	17.9
BM	Quadratic	16.3	0.440	26.8
	Momentum	4.91	0.217	15.1
	Random	7.83	0.299	13.7

Table 4. Comprehensive comparison of evaluation metrics of reinforcement learning trading algorithms and their variants on test data. MB = Model-Based. MF = Model-free. BM = Benchmark. Improvement of PG model is of 9.9% in annualized cumulative returns and 20.2% in annualized annualized Sharpe Ratio, compared to the best model-based reinforcement learning models, GRU-RNN.

Since ETFs are already diversified in themselves in stock idiosyncratic risk, our goal is to build a further diversified portfolio that would provide maximal risk-adjusted return. For this reason, Sharpe Ratio is probably the most appropriate measure.

In Table 4, we observe that the Policy Gradient model outperformed all the model-based models in Sharpe Ratio. It also outperformed all benchmark trading agents. Although VAR/RNN-based agents do not improve cumulative returns from random agent, they do improve Sharpe Ratio compared to random/uniform agent in our experiments. It is unsurprising to observe that the quadratic agent has a higher Sharpe Ratio than other benchmark models due to its set up to maximize Sharpe Ratio by construction.

Our implementation of Policy Gradient models confirm the superiority of universal model-free reinforcement learning agents over current portfolio management model in asset allocation strategies, with the achieved performance advantage of as 20.2% in annualized Sharpe Ratio.

It is worth mentioning that we presented the best model results for PG. During experimentation, we find the model is very sensitive to many factors:

- Learning Ratio. We experimented learning rate ranging from $10e-1$ to $10e-4$. The convergence of PG is very sensitive to the learning rate. And 0.003 has the best convergence results in our experiments.
- Initiation of w_0. We have experimented with 1) random assignment weights of total weight 1, 2) $[1, 0, \ldots, 0]$ (all money in cash), random assign weight 1 to one

asset 3). The resulting policy differs drastically.

Figure 6 illustrates the performance of a two-layer gated recurrent unit recurrent neural network, which is not outperforming the vector auto-regressive predictor as much as was expected. We notice that GRU-RNN tracks pretty closely to VAR agent. We configure our GRU-RNN Agent with window-size =1 (when we rapidly prototype). This result demonstrates that when only using lag-1 information, linear model like VAR and non-linear model like RNN in this setting do not differ significantly. It need further exploration to discover whether changing window-size will differentiate the behavior of RNN Agent from VAR Agent.

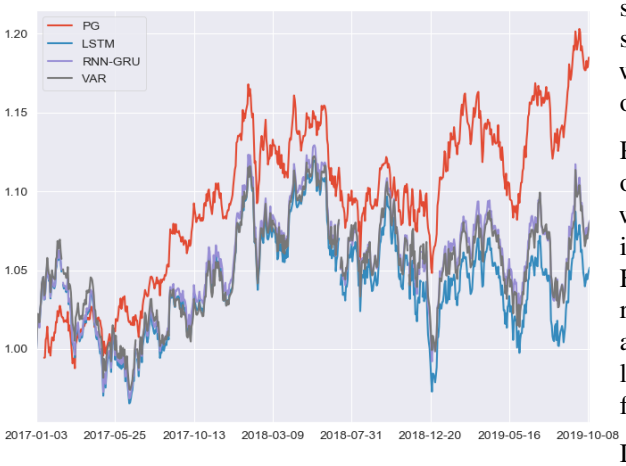

Figure 6. Comparison of portfolio value for model-based agents and model-free PG agent. PG uses two features: adjusted-close price, high price where as model-based agents use only adjusted-close price. PG applies window-size = 10, whereas model-based agents window-size = 1. The network of GRU-RNN and LSTM agents and network of CNN layers in PG are described in Appendix. All agents are trained on market data for 14 commodity ETFs for training period 2014-07-18 to 2016-12-31 and back-tested on testing period 2017-01-01 to 2019-10-10.

4.4. Conclusions

We explore an unified, versatile model-based framework for training agents. We set-up the learning framework in a way that can be used to adapt a variety of reward-generating functions, including traditional statistic models. We explore 3 reward-generating structures, Vector Autoregressive Regression, GRU-RNN, and LSTM. GRU-RNN performs the best in our experiment in 3 sectors. In principle, we expect that LSTM gives us the most control and thus would attain similar if not better results than GRU-RNN. However, it performs worse. We look to explore different network structures and window-sizes in combination with varying network structures to further explore this problem. However, although GRU-RNN and VAR behave similarly in portfolio-rebalancing actions, we find GRU-RNN improved Sharp Ratio by 0.013%. By looking at 1-lag price information, GRU-RNN achieves better results than VAR which looks at lag-15 days price information.

When model-free PG algorithm converges, it performs better than model-based algorithms. However, the algorithm requires stationary transition, thus can lead to failure of finding a globally optimal strategy. In training our models, we also find that model-free algorithms like Deep Deterministic Policy Gradient (DDPG) suffers from convergence issues (which is why we did not include it in most of the results). Due to the requirement of the algorithm, we could only search for optimal policy in the second-order differentiable strategy function set, instead of in the policy function set, which might also lead to the failure of finding a globally optimal strategy.

PG agent is degenerate in some of our experiments, which often tends to buy only one asset at a time or never update weights after initial assignment. This indicates more modifications are needed for designing promising algorithms. Further research on reward generating functions and state representation could improve the convergence properties and overall performance of the agents by exploiting the large number of degrees of freedom given by the framework formulation of financial markets.

Despite the performance gain of the developed strategies, the lack of interpretability and the inability to exhaustively test the deep architectures used pose us great challenges in adopting these models. As a consequence, it is worth investigating and interpreting the learned strategies by opening the deep "black box" and being able to reason for its decisions.

A. Appendix

A.1. Reinforcement Learning Architecture

Overall, the financial market is be modeled as an Innite Partially Observable Markov Decision Process, since:

- The action space is continuous (innite), $A \subseteq R^M$;
- The observations o_t are not sufficient statistics (partially observable) of the environment state;
- The state space is continuous (innite), $\mathbf{S} \subseteq R^K$.

In reinforcement learning framework with regard to portfolio management, we have:

- State(s): one state includes previous open, closing, high, low price, volume or some other financial indexes

in a fixed window.

- Action(a): In order to solve the asset allocation task, the trading agent should be able to determine the portfolio vector w_t at every time step t, therefore the action a_t at time t is the portfolio vector w_{t+1} at time $t+1$:

$$a_t \equiv w_{t+1} = [w_{1,t+1}, \quad w_{1,t+1}, \quad \ldots, \quad w_{M,t+1}]$$

subject to the constraint $\sum_{i=1}^{M} w_{i,t-1} = 1$.

- Reward (r): the uctuation of wealth. We did not consider transaction cost in this exercise. the immediate reward at time t-1 as:

$$r_t\left(s_{t-1}, w_{t-1}\right) = \log\left(\mathbf{w_{t-1}} \cdot \mathbf{y_{t-1}}\right).$$

State representation of reinforcement learning for trading can be expressed as:

$$\hat{s}_t \left\langle w_t, \vec{\rho}_{t-T:t} \right\rangle = \left\langle \begin{bmatrix} w_{1,t} \\ w_{2,t} \\ \vdots \\ w_{M,t} \end{bmatrix}, \begin{bmatrix} \rho_{1,t-T} & \rho_{1,t-T+1} & \cdots & \rho_{1,t} \\ \rho_{2,t-T} & \rho_{2,t-T+1} & \cdots & \rho_{2,t} \\ \vdots & \vdots & \vdots & \ddots \\ \vdots & & & \\ \rho_{M,t-T} & \rho_{M,t-T+1} & \cdots & \rho_{M,t} \end{bmatrix} \right\rangle$$

where $\vec{\rho}_{t-T:t}$ the log cumulative returns of asset i between the time interval $t-T:t$. Overall, the agent state is given by:

$$s_t^a \equiv f\left(s_{t-1}^a, \hat{s}_t; \theta\right)$$

A.2. Benchmark Agent

Quadratic agent is a mean-variance optimizing agent. Please refer for code for details.

A.3. Modeled-based Agent NN Structure

Table 5. LSTM Agent NN Structure

LAYER	NEURONS/RATE
LSTM	64
DROPOUT	0.2
LSTM	64
LSTM	64
DROPOUT	0.2
DENSE	15

Table 6. GRU/RNN Agent NN Structure

LAYER	NEURONS/RATE
GRU-RNN	64
DENSE	14

B. Contributions

Dickson Chan worked on the CNN model training. Michael Hsieh worked on the ETF data collection, LSTM model training/error evaluation, and CNN error evaluation. Sophie Pan worked on the time-series econometric model (VAR) and model-free Q-learning (RL trading agent.)

C. Code

Our code is at https://github.com/mhsieh33/cs229.

References

Filos, A. Reinforcement learning for portfolio management. pp. 127.

Kendall, M. G. The treatment of ties in ranking problems. *Biometrika*, 33(3):239–251, 1945.

M. U. Gudelek, S. A. B. and Ozbayoglu, A. M. A deep learning based stock trading model with 2-d cnn trend detection. pp. 1–8, 2017.

Messmer, M. Deep learning and the cross-section of expected returns. *SSRN Electron. J.*, 2017.

X. Li, Y. Li, X.-Y. L. and Wang, C. D. Risk management via anomaly circumvent: Mnemonic deep learning for midterm stock prediction. *ArXiv190801112 Cs Q-Fin Stat*, Aug., 2019.

Z. Jiang, D. X. and Liang, J. A deep reinforcement learning framework for the financial portfolio management problem. *ArXiv170610059 Cs Q-Fin*, Jun., 2017.