Hyperparameter Tuning Using Reinforcement Learning on Deep RL Networks for Stock Market Prediction

Ammar Husain, Maria V. Ruiz-Blondet, David A. Stanley

Abstract

In the present work, we set up a Reinforcement Learning (RL) agent to navigate the hyperparameter space of a deep RL model, in search of the best combination of hyperparameters, as measured by the deep RL model's performance in stock trading. We compare this approach to a more traditional hyperparameter sampling method, and identify whether the final models tuned by both approaches have a similar performance. The results show that our approach performs closely to but not as well as the traditional sampling method (implemented using an open-source library using the contribution of several people across the years). This performance is heavily affected by the formulation of the reward function. A reward that updates every step performs much better than a reward that updates at the end of each episode for this current application. When compared to randomly selecting a hyperparameter set, our approach outperforms random hyperparameter selection for all reward function formulations.

Introduction

Hyperparameter tuning is a challenging problem. Usually the machine learning practitioner will learn through experience the most reasonable values to use for a given set of model characteristics, such as network architecture and training schedule. When one is not an expert using a specific model, looking through the literature for reasonable or previously tested values can be a daunting task. A usual approach in these cases is using a sampling method to randomly try a few values from a fixed search space.

In one study, Zoph and Le (2016) used reinforcement learning to define the architecture of neural networks. Their state space is variable and dependent on the architecture of the network. For example, as the agent decides to add one more node to the network, a state gets added to the state space. Their reward is defined by the performance of the child network on a specific dataset. Given this reward, which requires a heavy calculation (namely, training and testing a neural network), the authors set up a highly parallelizable scheme that allows for the training of ~50 child networks at once.

Jomaa et al., (2019) is another group who tested the possibility of using RL for hyperparameter tuning. Their objective was to develop a network that would be capable of self-tuning its own hyperparameters when facing new types of datasets. In this work, the state space is defined as the hyperparameters chosen for the network at a specific time step, the set of hyperparameter configurations that have previously been tried, and as well as the corresponding reward. This means that the state space grows with each time step. The reward is defined as the cross validation loss of the child model, trained with the specific set of hyperparameters. The authors used the OpenAI framework to train the RL agent and made their code available on GitHub.

A final group we found who has researched this specific topic is Dong et al., (2018) for the area of computer vision, specifically, object tracking in video. Their space state and reward functions are specific to the object tracking problem. Similar to their previous work, the aim is that the network can change its parameters in response to the changes in the dataset that is being classified.

In our work, we aim to explore the same application of RL for hyperparameter tuning. As opposed to the studies mentioned before, we wanted to test if a simple application of RL could optimize hyperparameters better than a sampling method or a basic grid search. We defined the state as the space of all possible hyperparameter configurations we wished to consider for our child model. As opposed to the studies mentioned above, this allows our state space to be fixed across different time steps. The action corresponds to where on the state space to go next, and the reward corresponds to how well the child network performs with a specific set of hyperparameters.

At the risk of sounding recursive, our test network of choice is a deep deterministic policy gradient (DDPG) RL network, and the hyperparameters to optimize correspond to the discount factor, the learning rate, the batch size and the buffer size. The application we are choosing to explore is a stock market investing problem. The child network can decide at each time step whether to sell, buy or hold from a given portfolio. The reward is based on the Sharpe ratio, which is a metric used in finance that takes into consideration not only the absolute monetary wins and losses, but also the volatility (e.g., variability in price) of a specific stock or portfolio of stocks. Similar to the previous work mentioned, we expect that this RL tuned child network can perform successfully across a variety of financial datasets, without having to manually tune the hyperparameters for each specific dataset.

In the following section we will present how we designed, trained and tested the child network tuned with our RL agent. The results section will summarize our model's performance for each test case, which are presented in full as a table in Appendix 1. We will also provide a comparison with the results of the literature. Finally, the discussion will present the applications and limitations of the current study.

Methods

The task we use to validate our hypothesis is for portfolio allocation through stock trading with an initial allocation of $1M. The algorithm used for portfolio allocation was, as mentioned in the introduction, the DDPG Deep Reinforcement learning model. Our approach was to train an RL-based hyperparameter optimization model (called the RL agent) to tune the hyperparameters of this DDPG agent (called the child network). The performance of the RL agent at parameter tuning was compared against two baselines: (i) random sampling and (ii) Optuna - an open source optimization framework. Since our RL agent tasked with tuning hyperparameters of the DDPG model requires prior training to learn how to navigate the high-dimensional hyperparameter manifold, we split the task among 4 datasets all obtained through the Yahoo Finance API:

I. DOW-30 : Price-weighted measurement stock market index of 30 prominent companies listed on stock exchanges in the United States.
II. DAX-40 : Stock market index consisting of the 40 major German blue chip companies trading on the Frankfurt Stock Exchange.
III. HSI-50 : Freefloat-adjusted market-capitalization-weighted stock-market index in Hong Kong. These 50 constituent companies represent about 58% of the capitalisation of the Hong Kong Stock Exchange.
IV. NAS-100 : Stock market index made up of 102 equity securities issued by 100 of the largest non-financial companies listed on the Nasdaq stock market.

While the datasets varied in size given the number of securities that constitute that particular index, the training and evaluation periods for the underlying DDPG stock trading agent remained constant. The stock trading bot was trained with data between Jan 1, 2010 through Dec 31, 2015 and evaluated for performance using the Sharpe ratio metric between Jan 1, 2016 through Dec 31, 2017. The data points were sampled at a per-day granularity. The hypothesis of our approach was that the hyperparameter manifold of the DDPG stock trading agent would be similar between these 4 datasets and therefore a reinforcement learning agent trained to navigate this manifold on one dataset should successfully be able to traverse and optimize on a different unseen dataset.
The DDPG hyperparameters that were posed to the various models for optimization were the following:

- Discount factor : `[0.9, 0.95, 0.98, 0.99, 0.995, 0.999, 0.9999]`
- Learning rate : `[1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1]`
- Batch size : `[16, 32, 64, 100, 128, 256, 512, 1024, 2048]`
- Replay buffer size : `[1e4,1e5,1e6]`

This is a fairly high dimensional grid (1134 possible values) and training + evaluation of the DDPG model for just 5000 timesteps takes around ~120s. Therefore, a simple exhaustive grid search would be infeasible because that would take ~37.8 hours per dataset.

For the set of experiments in this paper, three different models were put to test. Two of these models were baselines to compare our RL-based approach against.

1. Random sampling: This approach simply samples different values on the 4D grid of 1134 cells.
2. Optuna : An open source hyperparameter optimization framework to automate hyperparameter search. While this library is highly customizable with different sampling and pruning algorithms, we used the library defaults to serve as baseline. This was the Tree-structured Parzen Estimator (TPE) for sampling and the Hyperband algorithm for pruning.
3. Ours: We used an Actor Advantage Critic (A2C) model to train our agent to navigate the hyperparameter grid with a simple maximum likelihood policy. Since this agent was tasked with tuning hyperparameters of another stock trading agent, we did not tune its own hyperparameters very much and used the sensible defaults as defined in the Stable-Baselines3 library from OpenAI and DLR. The A2C model was defined as follows:
    a. State: The coordinates of the agent in the 4D hyperparameter grid. The state
    b. Action: [-2,1,0,1,2] for each of the 4 hyperparameter dimensions on whether the agent should increment or decrement up to 2 spaces within that axis of the grid.
    c. Reward: We experimented with several different reward formulations to determine the best learning opportunity for our A2C RL agent. These are as follows:
        i. `best_state_end`: Agent is rewarded only when the episode ends with the value of best Sharpe ratio.
        ii. `best_state_ongoing`: Agent is continuously rewarded with the value of the best Sharpe ratio thus far as it navigates through the grid.
        iii. `current_state_end`: Agent is rewarded only when the episode ends with the value of the Sharpe ratio at the end state.
        iv. `current_state_ongoing`: Agent is continuously rewarded with the value of the current state's Sharpe ratio as it navigates through the grid.
        v. `accumulate_end`: Agent is rewarded only when the episode ends with the aggregate values of all Sharpe ratios that it traversed.
        vi. `accumulate_ongoing`: Agent is continuously rewarded with the aggregate values of all Sharpe ratios that it traversed thus far.

Results

For all approaches, we average the model's output of optimal hyperparameters on 20 different trials with different random initializations to make sure none of them randomly get lucky. Each trial is capped at 30 different hyperparameter runs that it can evaluate before picking the optimal hyperparameters. We train the A2C RL-agent on each of the 4 datasets and evaluate its performance on the remaining 3 datasets (e.g., leave one in training). We repeat this with the 6 different reward formulations described in the previous section. In Appendix 1 we present a table with detailed results for all the datasets and all the reward formulations. Our approach

requires training, so for each of the four datasets, we have trained our agent in one of them and tested it in the remaining three datasets. In contrast, the two baseline approaches do not require any pre-training and receive a cold-start on every single run on every dataset.

The results show that our child model with the hyperparameter set selected from the pool of over 1000 combinations performs better across all reward formulations and training datasets than simply picking the hyperparameters randomly. When compared to Optuna, our approach produces similar results. Across training with 4 different datasets, with 6 different reward formulations, and testing on the remaining 3 datasets, there are 72 possible comparisons. From these, our model outperforms Optuna in 13 instances (see (*) entries in Appendix 1). In all of these instances, with one exception, the reward type is ongoing as opposed to updated at the end of the episode. Additionally, our model performs best when trained on the dataset with the largest number of companies (NAS-100) and trained on the remaining three datasets that are smaller.

Compared to the studies presented in the introduction, here we used a very simple definition of state space, action and tried a few rewards for our RL agent. As opposed to the studies mentioned, our state space does not grow with each time step nor with added nodes to the network. Similar to Jomaa et al. (2019), we found similar results in training with one dataset, and then let the agent decide how to change the child network when facing new datasets.

Discussion

The topic of using reinforcement learning for hyperparameter tuning is extremely limited. However, this short exploration shows that it has as much potential as an open source library with heavy involvement from the community that has been improving itself for several years. For applications where someone works with the same type of data for long periods of time and is faced with new datasets every now and then, it might be worth the investment of training an RL model on a rich dataset to choose hyperparameters, and then use this agent to find new hyperparameters for new datasets that are being explored in the future. If this approach really has potential, it could be added to an open source library like Optuna, pre-trained in classical databases, and it could serve as an additional method for hyperparameter tuning.

This research has really shown the critical nature of defining the rewards in RL. For this particular application, we found that on-going rewards perform much better than rewards at the end of the episode. It might be worth exploring what is driving that difference. Additionally, there is no mechanism right now to end an episode early. It would be valuable to have such a mechanism that detects no change for a given number of steps and ends the episode early. Once that is implemented, we can compare our approach with Optuna again, this time comparing the number of iterations to arrive at the best hyperparameter set. Lastly, we have discretized certain naturally continuous parameters in our model hyperparameter space, such as the learning rate and the discount factor. Future work could re-implement our model with a state space that allows for certain hyperparameters to be continuous.

One of the biggest limitations of the current application is pre-training the reinforcement learning agent, which took several hours. If there is a way to improve this implementation so that pre-training is not necessary, that would expand the number of applications of this RL agent. Since it would not have to be pre-trained using new child networks or datasets, it would become a much more valuable off-the-shelf hyperparameter optimizer. A valuable addition to the table in the appendix, would be then, to add the performance of the child network when it had its hyperparameters selected by an untrained RL agent. This could be a starting point on a research that could allow reinforcement learning to become commonplace in hyperparameter tuning.

References

Deng, Yue, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. "Deep direct reinforcement learning for financial signal representation and trading." *IEEE transactions on neural networks and learning systems* 28, no. 3 (2016): 653-664.

Dong, Xingping, Jianbing Shen, Wenguan Wang, Yu Liu, Ling Shao, and Fatih Porikli. "Hyperparameter optimization for tracking with continuous deep q-learning." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 518-527. 2018. https://ieeexplore.ieee.org/document/8578159

Jomaa, Hadi S., Josif Grabocka, and Lars Schmidt-Thieme. "Hyp-rl: Hyperparameter optimization by reinforcement learning." arXiv preprint arXiv:1906.11527 (2019). https://arxiv.org/pdf/1906.11527.pdf

Liu, Xiao-Yang, Hongyang Yang, Qian Chen, Runjia Zhang, Liuqing Yang, Bowen Xiao, and Christina Dan Wang. "FinRL: A Deep Reinforcement Learning Library for Automated Stock Trading in Quantitative Finance." *arXiv preprint arXiv:2011.09607* (2020).

Xiong, Zhuoran, Xiao-Yang Liu, Shan Zhong, Hongyang Yang, and Anwar Walid. "Practical deep reinforcement learning approach for stock trading." *arXiv preprint arXiv:1811.07522* (2018).

Zoph, Barret, and Quoc V. Le. "Neural architecture search with reinforcement learning." arXiv preprint arXiv:1611.01578 (2016). https://arxiv.org/abs/1611.01578

Appendix 1.

The four columns indicate the criterion used to select the hyperparameter set for the child network. Starting from the left, Top-20 corresponds to the average of the top 20 hyperparameter sets that give the best results. Ours (RL) corresponds to the approach presented in this paper. Optuna corresponds to a well used open-source approach and Random corresponds to selecting the hyperparameter set randomly and observing how the child network performs. The leftmost column presents the dataset our approach was trained on (highlighted in red and presented as a sub-title) as well as the datasets used for testing. This is repeated for each of the six reward formulations.

The datasets highlighted in red below are the ones that the RL agent was trained as well as evaluated on, so it does have an advantage in that it has encountered this grid before. The numbers marked with an asterisk (*) are the ones where our approach outperforms both the baseline algorithms in finding an optimal hyperparameter set that maximizes the model's Sharpe ratio performance. Additionally, highlighted in yellow are those instances that correspond to an outperformance on databases not included during training.

Trained on DAX-30

best_state_end	Top-20	Ours (RL)	Optuna	Random
DAX-30	1.430	1.416*	1.311	0.960
HSI-50	2.671	2.146	2.577	1.966
DOW-30	2.829	2.441	2.705	2.139
NAS-100	1.890	1.630	1.768	1.370
best_state_ongoing	Top-20	Ours (RL)	Optuna	Random
DAX-30	1.430	1.389*	1.353	0.961
HSI-50	2.671	2.538	2.597	1.876
DOW-30	2.829	2.592	2.654	2.186
NAS-100	1.890	1.961*	1.811	1.251
current_state_end	Top-20	Ours (RL)	Optuna	Random
DAX-30	1.430	1.247	1.298	0.990
HSI-50	2.671	2.213	2.603	1.876
DOW-30	2.829	2.336	2.678	2.063
NAS-100	1.890	1.494	1.770	1.378
current_state_ongoing	Top-20	Ours (RL)	Optuna	Random
DAX-30	1.430	1.329*	1.284	0.971
HSI-50	2.671	2.482	2.546	1.999
DOW-30	2.829	2.541	2.759	2.051
NAS-100	1.890	1.625	1.760	1.353
accumulate_end	Top-20	Ours (RL)	Optuna	Random
DAX-30	1.430	1.208	1.281	1.031
HSI-50	2.671	2.520	2.619	1.928
DOW-30	2.829	2.516	2.696	2.119
NAS-100	1.890	1.737	1.801	1.318
accumulate_ongoing	Top-20	Ours (RL)	Optuna	Random

	DAX-30	1.430	1.285	1.312	1.004
	HSI-50	2.671	2.548	2.581	1.893
	DOW-30	2.829	2.576	2.681	2.129
	NAS-100	1.890	1.879*	1.836	1.293

Trained on HSI-50

best_state_end	Top-20	Ours (RL)	Optuna	Random
DAX-30	1.430	1.147	1.315	1.018
HSI-50	2.671	2.467	2.562	1.951
DOW-30	2.829	2.242	2.648	2.150
NAS-100	1.890	1.535	1.782	1.332
best_state_ongoing	**Top-20**	**Ours (RL)**	**Optuna**	**Random**
DAX-30	1.430	1.386*	1.323	0.981
HSI-50	2.671	2.619*	2.591	1.956
DOW-30	2.829	2.577	2.650	2.151
NAS-100	1.890	1.755	1.760	1.268
current_state_end	**Top-20**	**Ours (RL)**	**Optuna**	**Random**
DAX-30	1.430	1.121	1.282	0.971
HSI-50	2.671	2.321	2.632	1.999
DOW-30	2.829	2.255	2.690	2.155
NAS-100	1.890	1.448	1.768	1.398
current_state_ongoing	**Top-20**	**Ours (RL)**	**Optuna**	**Random**
DAX-30	1.430	1.169	1.299	0.992
HSI-50	2.671	2.629*	2.590	2.010
DOW-30	2.829	2.514	2.660	2.121
NAS-100	1.890	1.739	1.816	1.411
accumulate_end	**Top-20**	**Ours (RL)**	**Optuna**	**Random**
DAX-30	1.430	1.286*	1.282	0.969
HSI-50	2.671	2.300	2.565	1.979
DOW-30	2.829	2.424	2.629	2.116
NAS-100	1.890	1.578	1.739	1.366
accumulate_ongoing	**Top-20**	**Ours (RL)**	**Optuna**	**Random**
DAX-30	1.430	1.364*	1.286	0.941
HSI-50	2.671	2.595	2.623	1.967
DOW-30	2.829	2.516	2.713	2.190
NAS-100	1.890	1.661	1.788	1.382

Trained on NAS-100

best_state_end	Top-20	Ours (RL)	Optuna	Random
DAX-30	1.430	1.118	1.327	1.000
HSI-50	2.671	2.264	2.603	1.922
DOW-30	2.829	2.311	2.700	2.251

	NAS-100	1.890	1.578	1.769	1.336
best_state_ongoing	Top-20	Ours (RL)	Optuna	Random	
	DAX-30	1.430	1.388*	1.319	0.971
	HSI-50	2.671	2.649*	2.531	2.004
	DOW-30	2.829	2.554	2.665	2.115
	NAS-100	1.890	1.973*	1.819	1.274
current_state_end	Top-20	Ours (RL)	Optuna	Random	
	DAX-30	1.430	1.161	1.327	1.016
	HSI-50	2.671	2.277	2.577	1.947
	DOW-30	2.829	2.469	2.752	2.241
	NAS-100	1.890	1.551	1.776	1.329
current_state_ongoing	Top-20	Ours (RL)	Optuna	Random	
	DAX-30	1.430	1.227	1.324	1.014
	HSI-50	2.671	2.451	2.582	1.798
	DOW-30	2.829	2.527	2.651	2.148
	NAS-100	1.890	1.973*	1.753	1.370
accumulate_end	Top-20	Ours (RL)	Optuna	Random	
	DAX-30	1.430	1.287	1.318	1.039
	HSI-50	2.671	2.373	2.580	1.890
	DOW-30	2.829	2.384	2.747	2.119
	NAS-100	1.890	1.609	1.804	1.393
accumulate_ongoing	Top-20	Ours (RL)	Optuna	Random	
	DAX-30	1.430	1.310*	1.297	1.020
	HSI-50	2.671	2.533	2.613	2.009
	DOW-30	2.829	2.592	2.733	2.152
	NAS-100	1.890	1.744	1.771	1.311

Trained on DOW-30

best_state_end	Top-20	Ours (RL)	Optuna	Random	
	DAX-30	1.430	1.127	1.334	0.935
	HSI-50	2.671	2.457	2.554	2.176
	DOW-30	2.829	2.843*	2.657	2.207
	NAS-100	1.890	1.567	1.766	1.409
best_state_ongoing	Top-20	Ours (RL)	Optuna	Random	
	DAX-30	1.430	1.283	1.351	0.995
	HSI-50	2.671	2.636*	2.552	2.037
	DOW-30	2.829	2.692*	2.664	2.081
	NAS-100	1.890	1.723	1.762	1.372
current_state_end	Top-20	Ours (RL)	Optuna	Random	
	DAX-30	1.430	1.138	1.366	0.987
	HSI-50	2.671	2.138	2.589	2.015
	DOW-30	2.829	2.451	2.666	2.126
	NAS-100	1.890	1.587	1.716	1.362
current_state_ongoing	Top-20	Ours (RL)	Optuna	Random	

DAX-30	1.430	1.370*	1.279	0.986
HSI-50	2.671	2.607*	2.595	2.030
DOW-30	2.829	2.686*	2.666	2.073
NAS-100	1.890	1.799*	1.768	1.284
accumulate_end	Top-20	Ours (RL)	Optuna	Random
DAX-30	1.430	1.169	1.324	0.998
HSI-50	2.671	2.363	2.569	1.947
DOW-30	2.829	2.342	2.695	2.146
NAS-100	1.890	1.498	1.838	1.316
accumulate_ongoing	Top-20	Ours (RL)	Optuna	Random
DAX-30	1.430	1.227	1.339	0.995
HSI-50	2.671	2.742*	2.566	1.855
DOW-30	2.829	2.753*	2.676	2.111
NAS-100	1.890	1.687	1.851	1.356