Adaptive Quantitative Trading: An Imitative Deep Reinforcement Learning Approach

Chia-Hung Liao[1], **Chia-Wei Hsu**[2], **Zhen-Yuan Hsu**[3], **Yu-Lin Lu**[4]
{[1]aiallen.cs07g, [3]anson.cs08g}@nctu.edu.tw
{[2]music87.c, [4]nycu309551038.cs09}@nycu.edu.tw
Department of Computer Science
National Chiao Tung University

1 Problem Overview

Please provide a brief overview of the selected paper. You may want to discuss the following aspects:

- The main research problem tackled by the paper
- High-level description of the proposed method

There are three challenges in this paper.

- Challenge 1: Noisy High-frequent Financial Data

  First, the noisy high-frequent financial data. The market state is hard to be observed since the high-frequent financial data contain much noise. No one can know exactly whether the market is bullish or bearish, and whether it's time to buy or sell the security.

- Challenge 2: Poor Generalization of Technical Analysis

  Second, poor generalization of commonly used technical analysis. Even if the pattern of market reappears, the identical technical analysis may not be able to earn from the similar pattern again. Thousands of factor's interactions cause the market information indistinguishable. To figure out this noisy information and improve generalization ability, machine learning gives the big-scale problem a chance to be solved. This paper seems the market as partially observable Markov decision process (POMDP) [Kaelbling et al., 1998] and solves the process by recurrent deterministic policy gradient (RDPG) [Heess et al., 2015a], an off-policy deep reinforcement learning algorithm while deep learning concentrates on predicting the price trend and reinforcement learning focuses on constructing the best trading strategy.

- Challenge 3: Balancing Exploration and Exploitation

  But the reinforcement learning itself gives the third challenge, balancing exploration and exploitation. Because of the market friction like transaction fee, slippage, and market capacity, randomly explore the market without goals may cause great losses. However, a beneficial trading strategy is hard to get without enough trials and errors. To alleviate the trade-off problem, this paper uses imitative learning techniques, that is, demonstration buffer and behavior cloning while the former learns from the technical index, Dual Thrust and the latter sets a prophetic trading expert which will take a daily sub-optimal action to execute. Overall, the whole three challenges can be tackled by the proposed method from this paper, imitative Recurrent Deterministic Policy Gradient (iRDPG) which contains imitative technique and RDPG algorithm.

Preprint. Under review.

2 Background and The Algorithm

Please present the essential background knowledge and the algorithm in this section. You may also describe the notations and the optimization problem of interest.

2.1 Problem Formulation

- POMDP

  As mentioned in section 1, this paper models the market as partially observable Markov decision process (POMDP) instead of traditional Markov decision process(MDP) due to the noisy trading environment. We can't exactly know what the current market state is, and the only thing we can know is its partial observation.

$$\text{POMDP}\left\{\begin{array}{l}\left.\begin{array}{l} S: \text{a finite set of states}\\ A: \text{a finite set of actions}\\ T: S*A*S\to[0,1]: \text{a state transition function}\\ R: S*A\to R: \text{a reward function}\\ \gamma: \text{a discount factor}\end{array}\right\}\text{MDP}\\ O: \text{the corresponding observations of } S\\ Z: S*A*O\to[0,1]: \text{the corresponding observation transition function of } T\end{array}\right.$$

- Observation

  The observation O we can get involves historical price $\mathbb{P}$, various technical indicators $\mathbb{Q}$, and account profit, $\mathbb{R}$. The transition function of the whole observation set can fit the POMDP framework $Z = Z(o_{t+1}|s_{t+1}, a_t)$. But trading actions from an individual investor have little impact on the entire market which means $Z = Z(o^m_{t+1}|s_{t+1})$, hence the whole observation set of POMDP separates the first two parts as the market observation O^m from the last part as the account observation O^a. Except for the original market observation in this paper, we also add some additional variables into O^m like moving average, margin, stochastic oscillator (KD) to increase the convergence process. Note that the account profit is just a observation, not the reward.

$$O\left\{\begin{array}{l}\left.\begin{array}{l}\mathbb{P}:[p_1,...,p_t,...],\ \text{and } \mathbb{P}_{t-n:t}=[p_{t-n+1},...,p_{t-1},p_t],\\ \quad\text{where } p_t=[p_t^o,p_t^h,p_t^l,p_t^c],\ \text{an open,high,low,close price vector}\\ \mathbb{Q}:[q_1,...,q_t,...],\ \text{where } q_t=\cup_j q_t^j,\ \text{an indicator vector,}\\ \quad q_t^j=f(\mathbb{P}_{t-n:t};\theta^j),\ \text{and } \theta^j \text{ is parameter of technical strategy } j\end{array}\right\}O^m\\ \mathbb{R}:[r_1,...,r_t,...],\ \text{where } r_t \text{ is the account profit}\}\,O^a\end{array}\right.$$

$$\Rightarrow o^a_t \in O^a,\ \text{where } o^a_t \text{ denotes the cumulative account profit, } \sum_{k=1}^{t} r_k \in \mathbb{R}$$

$$o^m_t \in O^m,\ \text{where } o^m_t \text{ is related with } p_t \text{ and } q_t$$

- Action

  The trading action a_t is defined as a continuous probability vector, and the final executed action will be the one with maximal probability. Furthermore, it doesn't mean that the agent should always place an new order to change the position based on the executed action. A new position will be placed or be changed only if the current executed action is empty or alternate from long/short to short/long.

$$a_t = [P_{long}, P_{short}]$$

- Reward

  Account profit $r_t \in \mathbb{R}$ includes two transaction factors, transaction fee δ and slippage ζ.

$$r_t = (p^c_t - p^c_{t-1} - 2\zeta)a_{t-1} - \delta|a_t - a_{t-1}|p^c_t$$

It's not appropriate to use it as reward, because the account profit will often be zero and sparse as the new order not being placed. Instead, this paper uses differential sharp ratio (DSR) D_t as the reward function R. It is the dynamic version of sharp ratio Sr_t that can measure the return while considering the risk.

$$Sr_t = \frac{E[\mathbb{R}_{t-n:t}]}{\sigma[\mathbb{R}_{t-n:t}]},$$

$$D_t \equiv \frac{dSr_t}{d\eta} = \frac{\beta_{t-1}\Delta\alpha_t - \frac{1}{2}\alpha_{t-1}\Delta\beta_t}{(\beta_{t-1} - \alpha_{t-1}^2)^{3/2}},$$

where η is the adaptation rate,

α_t and β_t are exponential moving estimates of first and second moment of r_t,

$$\alpha_t = \alpha_{t-1} + \eta\Delta\alpha_t = \alpha_{t-1} + \eta(r_t - \alpha_{t-1}),$$

$$\beta_t = \beta_{t-1} + \eta\Delta\beta_t = \beta_{t-1} + \eta(r_t^2 - \beta_{t-1}).$$

2.2 Methodology–RDPG

Memory-based control with recurrent neural networks (Heess et al. [2015b]) try to improve DDPG by introducing the recurrent neural networks. The method proposed is called Recurrent Deterministic Policy Gradient abbreviate as RDPG.

DDPG is an actor-critic approach, which bridges the gap between policy gradient methods and value approximation methods for RL. The agent trains the critic network by minimizing the TD error. Simultaneously, the agent learns a deterministic policy actor by directly maximizing the estimated Q function from critic.

This paper use of the GRU to effectively synthesize historical information. The objective function and optimization method for RDPG are almost the same to DDPG. However, because the RDPG will be updated through BPTT, so the unit of an experience in the replay buffer is a sequence data, not a step data .

We show the RDPG algorithm bellow:

```
Initialize critic network $Q^{\omega}\left(a_t, h_t\right)$ and actor $\mu^{\theta}\left(h_t\right)$ with parameters $\omega$ and $\theta$.
Initialize target networks $Q^{\omega'}$ and $\mu^{\theta'}$ with weights $\omega' \leftarrow \omega, \theta' \leftarrow \theta$.
Initialize replay buffer $R$.
for episodes $= 1, \mathrm{M}$
    do initialize empty history $h_0$
    for t $= 1,\ $T do receive observation $o_t$
        $h_t = GRU\left(h_{t-1}, a_{t-1}, o_t\right)$ ( append
        observation and previous action to history )
        select action $a_t = \mu^{\theta}\left(h_t\right) + \epsilon($ with $\epsilon$ : exploration noise )
    end for
    Store the sequence $\left(o_1, a_1, r_1 \ldots o_T, a_T, r_T\right)$ in $R$
    Sample a minibatch of $N$ episodes $\left(o_1^i, a_1^i, r_1^i, \ldots o_T^i, a_T^i, r_T^i\right)_{i=1,\ldots,N}$ from $R$
    Construct histories $h_t^i = \left(o_1^i, a_1^i, \ldots a_{t-1}^i, o_t^i\right)$
    Compute target values for each sample episode $\left(y_1^i, \ldots y_T^i\right)$ using the recurrent target networks
        $y_t^i = r_t^i + \gamma Q^{\omega'}\left(h_{t+1}^i, \mu^{\theta'}\left(h_{t+1}^i\right)\right)$
    Compute critic update (using BPTT)
        loss $L = \mathbb{E}\left[\left(y_t^i - Q^{\omega}\left(h_t^i, a_t^i\right)\right)^2\right]$
    Compute actor update (using BPTT)
        $\nabla_\theta J = \mathbb{E}\left[ \nabla_a Q^\omega(h,a)|_{h=h_t^i, a=\mu^\theta(h_t^i)} \nabla_\theta \mu^\theta(h)\Big|_{h=h_t^i} \right]$
    Update actor and critic using Adam Update the target networks
        $\omega' \leftarrow \tau\omega + (1-\tau)\omega'$
        $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$
end for
```

2.3 Methodology–Demonstration Buffer

In order to enhance the effectiveness of training, Liu et al. [2020] introduced the imitation learning technique. Since the detail is limited in previous paper, we refer to the practice in DDPGfD (Vecerik et al. [2017]) to implement demonstration buffer. As the name suggests, demonstration buffer will store expert Demonstration first. The expert policy is from Dual Thrust strategy which will be explained later.

To enable efficient propagation of the reward information they apply prioritized expereince replay(PER) (Schaul et al. [2015]). In PER, The probability of sampling a particular transition i is proportional to its priority, $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$, where α is hyperparameter and p_i is the priority of the transition. DDPGfD uses $p_i = \delta_i^2 + \lambda_3 \left|\nabla_a Q\left(s_i, a_i \mid \theta^Q\right)\right|^2 + \epsilon + \epsilon_D$, where δ_i is the last TD error calculated for this transition, the second term represents the loss applied to the actor, ϵ is a small positive constant to ensure all transitions are sampled with some probability, ϵ_D is a positive constant for demonstration transitions to increase their probability of getting sampled, and λ_3 is used to weight the contributions. To account for the change in the distribution, updates to the network are weighted with importance sampling weights, $w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta$. β is hyperparameter.

2.4 Methodology–Behavior Cloning

Behavior cloning is try to clone the expert policy. In hindsight, we can create a prophetic trading expert. That is, we introduce intra-day greedy actions as expert actions. As shown by the picture, we take a long position at lowest price, and a short position at highest price. Therefore, it is always a relatively optimal greedy strategy.

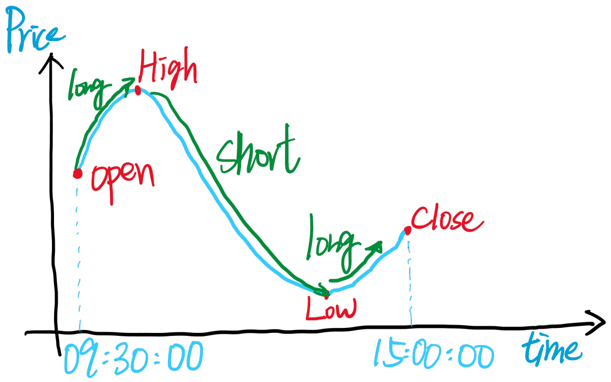


Figure 1: intra-day greedy actions as expert actions

The next is about the loss function. You could see the down figure. Since Behavior cloning is a kind of supervised learning approach, we take the square of the policy difference between actor and expert. In addition, we apply the Q-filter technique. That is, we record BC Losses only when the critic Q indicates that the expert actions perform better than the actor action.
Finally, as shown by the picture below, we combine the BC loss with the original objective function, and calculate its policy gradient for updating the actor.

BC Loss with Q-filter: $\boxed{L' = -\mathbb{E}\left[\left\|\mu^{\theta}(h_t^i) - \bar{a}_t^i\right\|^2 \mathbf{1}_{Q(h_t^i,\bar{a}_t^i)>Q(h_t^i,\mu^{\theta}(h_t^i)}\right]}$

Modified policy gradient to the actor: $\nabla_{\theta}\bar{J} = \lambda_1 \nabla_{\theta} J + \lambda_2 \nabla_{\theta} L'$

3 Detailed Implementation

Please explain your implementation in detail. You may do this with the help of pseudo code or a figure of system architecture. Please also highlight which parts of the algorithm lead to the most difficulty in your implementation.

3.1 Whole iRDPG Algorithm

iRDPG proposed by Liu et al. [2020] is abbreviation of imitative Recurrent Deterministic Policy Gradient. As the name suggests, which improve RDPG by introducing imitation learning.

As the figure showed bellow. We can see the architecture of iRDPG. The bottom part is the market observation. It will first be fed into the GRU layer to encode its historical information, and then the GRU hidden state will be send into the Agent. In the agent module, the behavior cloning loss is used to supervised the actor. And the Demonstration Buffer allows Actors to learn the experience of expert.

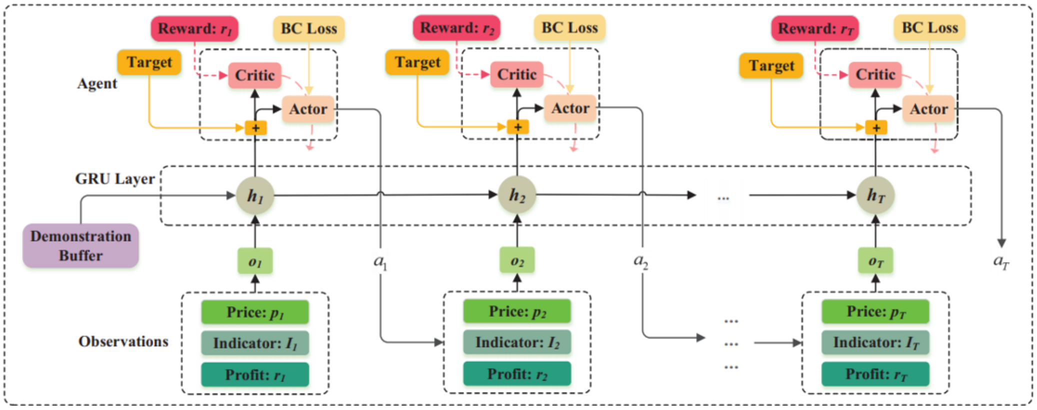
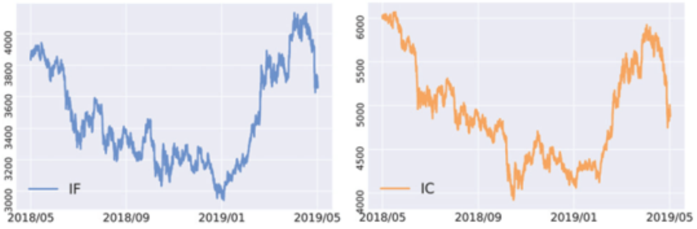


Figure 2: The overview of iRDPG model

3.2 Experiment Setup

In our experiment, we use minute-bar OHLC prices data of futures, i.e. minute-frequent futures data(IF and IC) JoinQuant.
Training set starts from 2016/01/01 to 2018/05/08.
Testing set spans from 2018/05/09 to 2019/05/08.
Transaction fee : 2.3*10-5
The constant slippage: 0.2
Initialize account: 500000 CNY
Each training episode will be broken off once positions are lost around 30% or lacking margin.

* The IF data are based on the index calculated on account of the prices of the top 300 stocks from both Shanghai and Shenzhen exchange centers.
* The IC data are based on another similar index, which focuses on the stocks with mid and small capitalization.



(a) IF (b) IC

Figure 3: IF and IC stock-index futures in our test set

3.3 Dual Thrust Strategy

It is famous and has been commonly used in futures, forex, and equity markets. As the introduction in the Demonstration Buffer, we select the Dual Thrust strategy as the demonstration trading policy. First, the DT determines a reasonable price oscillation Range, and the Range is calculated in this way by using OHLC in the previous n periods. Then use this range to determine the buyline and sellline. Finally, if the price goes beyond the buyline, a long position will be taken. And the short position will be taken if the price goes under the sellLine.

The long signal is calculated by Buyline (cap)=open+K1×Range.
The short signal is calculated by SellLine (floor) = open–K2×Range
, where K1 and K2 are the parameters. When K1 is greater than K2, it is much easier to trigger the long signal and vice versa. The idea of Dual Thrust is similar to a typical breakout system, However, dual thrust uses the historical price to construct update the look back period - theoretically making it more stable in any given period.

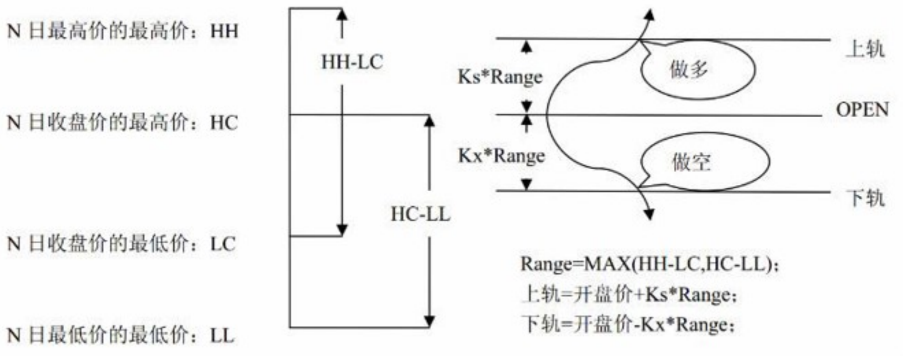


Figure 4: Dual Thrust Strategy

3.4 The Most Difficulties in Our Implementation

First, no member is familiar with the market characteristics and trading rules of China stock index futures, but fortunately the minute-frequent price data can be friendly downloaded on JoinQuant website for a student registration member. Beside, although the POMDP framework can better represent the noisy minute-frequent financial data, it is still quite a challenge to tune the agent.
No source code, the simulated trading environment is also coded from scratch, its complexity is beyond our imagination. Besides, although the differential sharp ratio (DSR) looks having nice property for considering both dynamically changing return and risk, but the formula often faces zero values for both numerator and denominator and how to set the proper value of adaptation rate η.
The ablation experiment takes so much time because the hyperparameters are quite different for each setting. For example, when adding the new imitation techniques to the original RDPG, the characteristics for each training process are very different, the trial and error need much efforts.

4 Empirical Evaluation

Please showcase your empirical results in this section. Please clearly specify which sets of experiments of the original paper are considered in your report. Please also report the corresponding hyperparameters of each experiment.

4.1 Training History

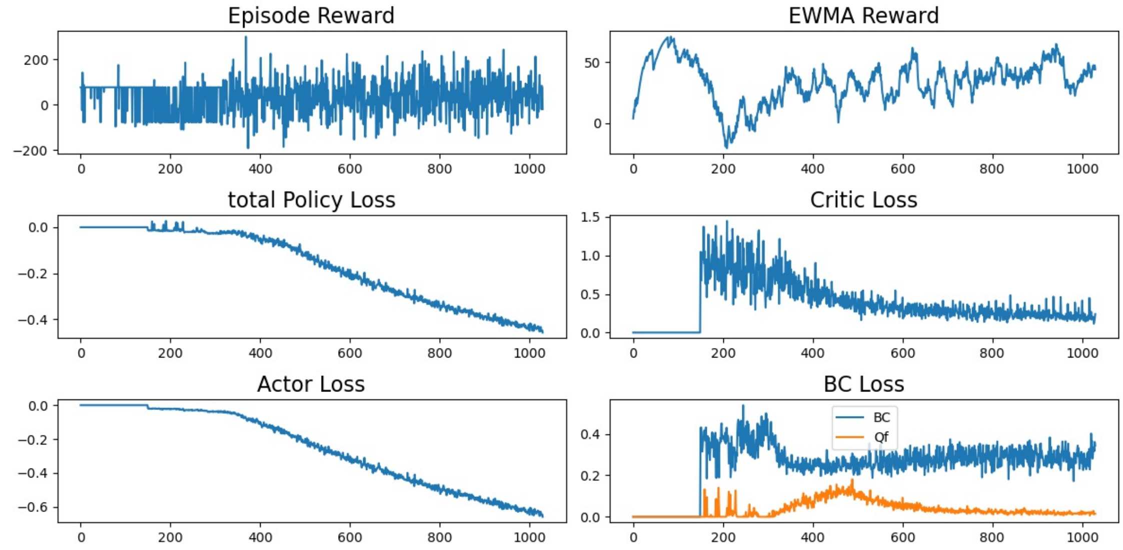
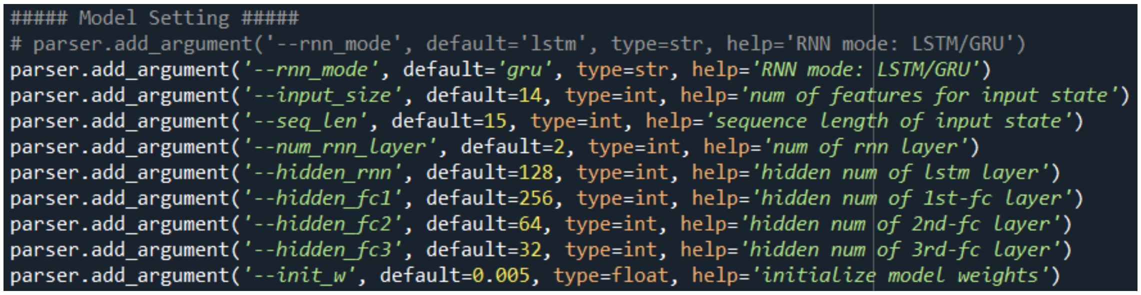


Figure 5: The training curve

This paper didn't show any training history, so here shows our training results for the iRDPG agent. The total policy loss is the summation of actor loss and BC loss after Q-filter. For the first 150 episodes, we only conduct sampling without training. So in the early training stage, the EWMA reward is relatively high because these are expert demonstrations. Then, from the BC Loss figure, the yellow curve is the BC loss after Q-filter. And we can see that the yellow curve decreases much, even though the original BC loss doesn't decrease. It is evident that the iRDPG agent learns better policies than the expert as the training time goes by.

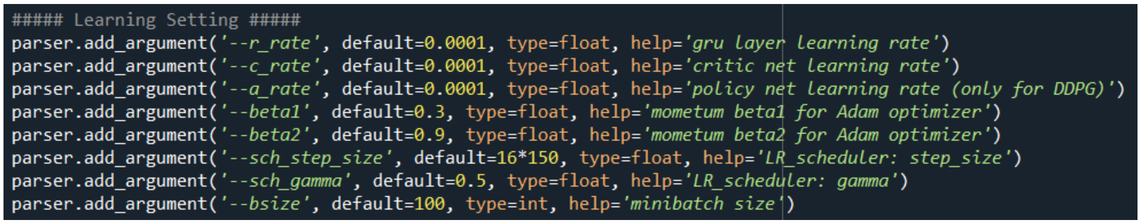
4.2 Hyperparameters



```
##### Model Setting #####
# parser.add_argument('--rnn_mode', default='lstm', type=str, help='RNN mode: LSTM/GRU')
parser.add_argument('--rnn_mode', default='gru', type=str, help='RNN mode: LSTM/GRU')
parser.add_argument('--input_size', default=14, type=int, help='num of features for input state')
parser.add_argument('--seq_len', default=15, type=int, help='sequence length of input state')
parser.add_argument('--num_rnn_layer', default=2, type=int, help='num of rnn layer')
parser.add_argument('--hidden_rnn', default=128, type=int, help='hidden num of lstm layer')
parser.add_argument('--hidden_fc1', default=256, type=int, help='hidden num of 1st-fc layer')
parser.add_argument('--hidden_fc2', default=64, type=int, help='hidden num of 2nd-fc layer')
parser.add_argument('--hidden_fc3', default=32, type=int, help='hidden num of 3rd-fc layer')
parser.add_argument('--init_w', default=0.005, type=float, help='initialize model weights')
```

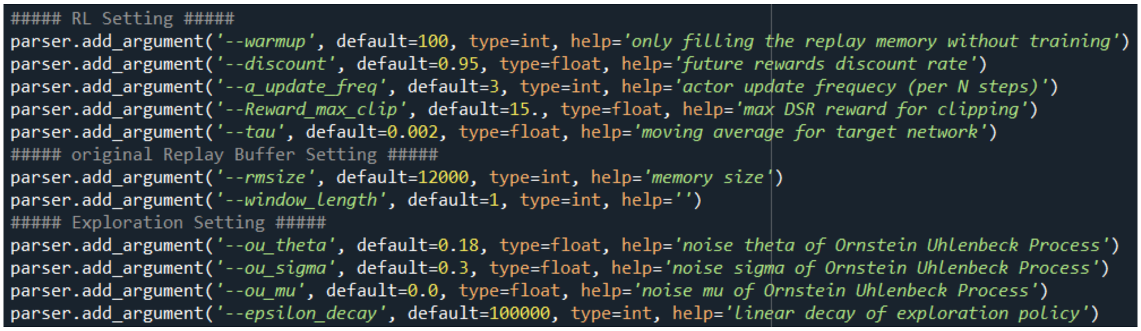
Figure 6: Parameters for agent model

Figure 6 presents the agent model setting. we can ether set the RNN layer as LSTM or GRU with two layers. The input size means input features which contain OHLC prices, buyline-selline of Dual Thrust, simple moving averages with previous 7 and 56 minites, RSI, KD, MACD, account profit and margin, etc. And the state sequence length is 15 minutes. In addition, both actor and critic has same three hidden layers but different input and output.



```
##### Learning Setting #####
parser.add_argument('--r_rate', default=0.0001, type=float, help='gru layer learning rate')
parser.add_argument('--c_rate', default=0.0001, type=float, help='critic net learning rate')
parser.add_argument('--a_rate', default=0.0001, type=float, help='policy net learning rate (only for DDPG)')
parser.add_argument('--beta1', default=0.3, type=float, help='mometum beta1 for Adam optimizer')
parser.add_argument('--beta2', default=0.9, type=float, help='mometum beta2 for Adam optimizer')
parser.add_argument('--sch_step_size', default=16*150, type=float, help='LR_scheduler: step_size')
parser.add_argument('--sch_gamma', default=0.5, type=float, help='LR_scheduler: gamma')
parser.add_argument('--bsize', default=100, type=int, help='minibatch size')
```

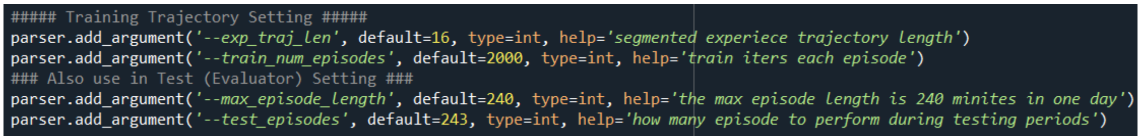
Figure 7: Parameters for Adam optimizer and LR-scheduler



```
##### RL Setting #####
parser.add_argument('--warmup', default=100, type=int, help='only filling the replay memory without training')
parser.add_argument('--discount', default=0.95, type=float, help='future rewards discount rate')
parser.add_argument('--a_update_freq', default=3, type=int, help='actor update frequecy (per N steps)')
parser.add_argument('--Reward_max_clip', default=15., type=float, help='max DSR reward for clipping')
parser.add_argument('--tau', default=0.002, type=float, help='moving average for target network')
##### original Replay Buffer Setting #####
parser.add_argument('--rmsize', default=12000, type=int, help='memory size')
parser.add_argument('--window_length', default=1, type=int, help='')
##### Exploration Setting #####
parser.add_argument('--ou_theta', default=0.18, type=float, help='noise theta of Ornstein Uhlenbeck Process')
parser.add_argument('--ou_sigma', default=0.3, type=float, help='noise sigma of Ornstein Uhlenbeck Process')
parser.add_argument('--ou_mu', default=0.0, type=float, help='noise mu of Ornstein Uhlenbeck Process')
parser.add_argument('--epsilon_decay', default=100000, type=int, help='linear decay of exploration policy')
```

Figure 8: Parameters for RL-algorithm, and random-process

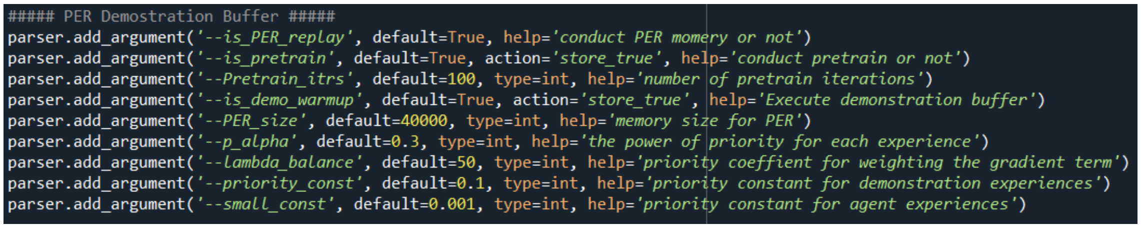
In figure 8, we set the warmup to be 100 episodes. Discount rate is 0.95. The soft replace rate is 0.002. Note, we update the actor every three step after updating the critic. Besides, the exploration is based on the Ornstein Uhlenbeck Process.



```
##### Training Trajectory Setting #####
parser.add_argument('--exp_traj_len', default=16, type=int, help='segmented experiece trajectory length')
parser.add_argument('--train_num_episodes', default=2000, type=int, help='train iters each episode')
### Also use in Test (Evaluator) Setting ###
parser.add_argument('--max_episode_length', default=240, type=int, help='the max episode length is 240 minites in one day')
parser.add_argument('--test_episodes', default=243, type=int, help='how many episode to perform during testing periods')
```

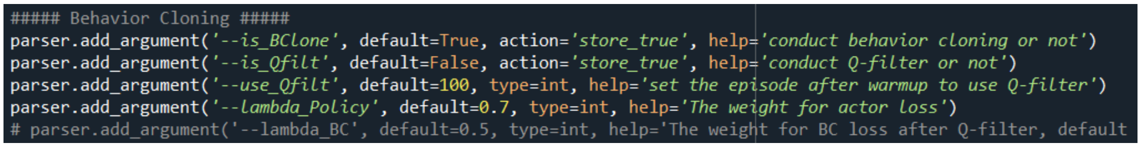
Figure 9: Parameters for training trajectory

In figure 9, the maximum steps of each episode is 240 since there are 240 trading minutes each day. And we segment every 16 steps to form an experience for train the recurrent DPG network. In addition, there are 243 trading days in the testing period.



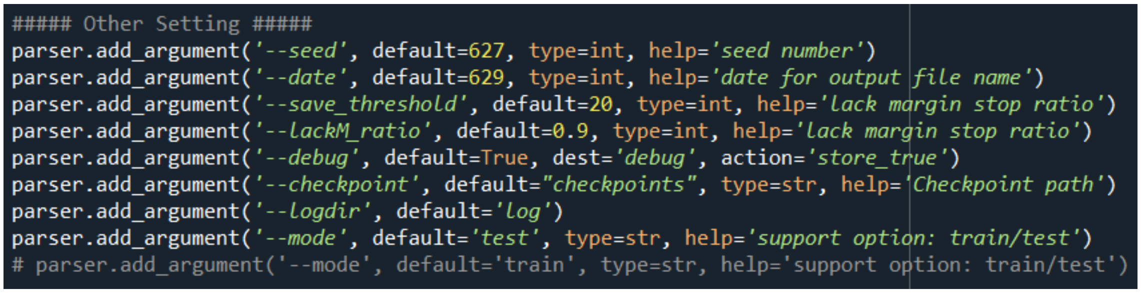
```
##### PER Demostration Buffer #####
parser.add_argument('--is_PER_replay', default=True, help='conduct PER momery or not')
parser.add_argument('--is_pretrain', default=True, action='store_true', help='conduct pretrain or not')
parser.add_argument('--Pretrain_itrs', default=100, type=int, help='number of pretrain iterations')
parser.add_argument('--is_demo_warmup', default=True, action='store_true', help='Execute demonstration buffer')
parser.add_argument('--PER_size', default=40000, type=int, help='memory size for PER')
parser.add_argument('--p_alpha', default=0.3, type=int, help='the power of priority for each experience')
parser.add_argument('--lambda_balance', default=50, type=int, help='priority coeffient for weighting the gradient term')
parser.add_argument('--priority_const', default=0.1, type=int, help='priority constant for demonstration experiences')
parser.add_argument('--small_const', default=0.001, type=int, help='priority constant for agent experiences')
```

Figure 10: Parameters for demonstration buffer and PER



```
##### Behavior Cloning #####
parser.add_argument('--is_BClone', default=True, action='store_true', help='conduct behavior cloning or not')
parser.add_argument('--is_Qfilt', default=False, action='store_true', help='conduct Q-filter or not')
parser.add_argument('--use_Qfilt', default=100, type=int, help='set the episode after warmup to use Q-filter')
parser.add_argument('--lambda_Policy', default=0.7, type=int, help='The weight for actor loss')
# parser.add_argument('--lambda_BC', default=0.5, type=int, help='The weight for BC loss after Q-filter, default
```

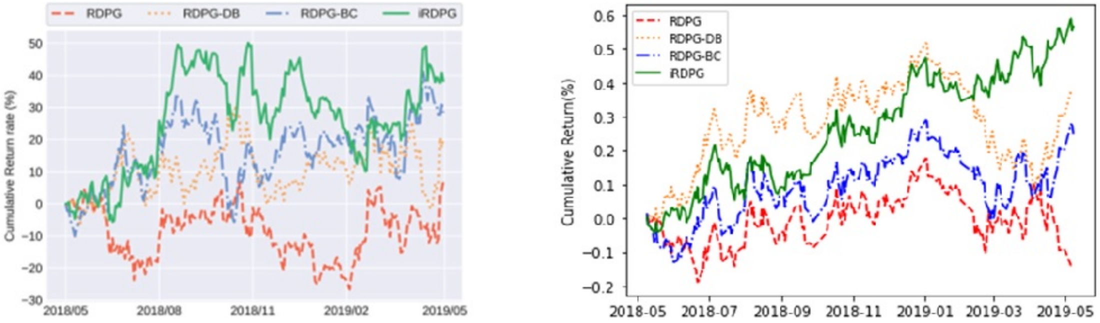
Figure 11: Parameters for behavior cloning



```
##### Other Setting #####
parser.add_argument('--seed', default=627, type=int, help='seed number')
parser.add_argument('--date', default=629, type=int, help='date for output file name')
parser.add_argument('--save_threshold', default=20, type=int, help='lack margin stop ratio')
parser.add_argument('--lackM_ratio', default=0.9, type=int, help='lack margin stop ratio')
parser.add_argument('--debug', default=True, dest='debug', action='store_true')
parser.add_argument('--checkpoint', default="checkpoints", type=str, help='Checkpoint path')
parser.add_argument('--logdir', default='log')
parser.add_argument('--mode', default='test', type=str, help='support option: train/test')
# parser.add_argument('--mode', default='train', type=str, help='support option: train/test')
```

Figure 12: Parameters for other settings

4.3 Ablation Experiments

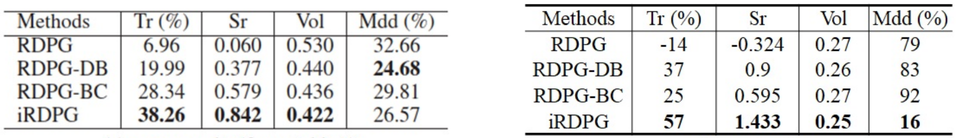


(a) paper results (from figure 5)

(b) our results

Figure 13: Cumulative return rates of ablation methods

This figure shows the comparison of ablation experiments. The LHS are the paper results (from Figure 5), and the RHS are our results. Both figures are the trading simulations by applying the trained agent to the one year test data (IF stock index futures from 2018/5/9 to 2019/5/8). And the four settings are RDPG, RDPG plus demonstration buffer (RDPG-DB), RDPG plus behavior cloning (BC), and the comprehensive iRDPG. Obviously, we can see that both results have similar trend, that is the iRDPG got the best trading performance. Also, we can find that if we train the RDPG without imitation technique, the RDPG trading performances are both not stable as shown by these two red curve.



Methods	Tr (%)	Sr	Vol	Mdd (%)
RDPG	6.96	0.060	0.530	32.66
RDPG-DB	19.99	0.377	0.440	**24.68**
RDPG-BC	28.34	0.579	0.436	29.81
iRDPG	**38.26**	**0.842**	**0.422**	26.57

(a) paper results (from Table 2)

Methods	Tr (%)	Sr	Vol	Mdd (%)
RDPG	-14	-0.324	0.27	79
RDPG-DB	37	0.9	0.26	83
RDPG-BC	25	0.595	0.27	92
iRDPG	**57**	**1.433**	**0.25**	**16**

(b) our results

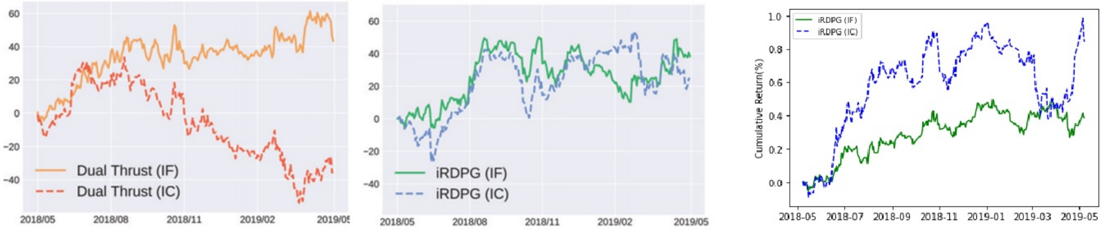
Figure 14: Ablation experiments of IF

The above two tables are the comparison of ablation trading performances on the following most widely used criteria. The LHS are the paper results (from Table 2), and the RHS are our results. Both tables are the trading performances by applying the trained agent to the one year test data (IF stock index futures from 2018/5/9 to 2019/5/8). Again, we can find that the iRDPG got the best performance.

- Total return rate **Tr** $:= \frac{P_{end} - P_{start}}{P_{start}}$ (P is the total value of the position and cash).
- Sharpe ratio **Sr** $:= \frac{\mathbb{E}(\mathbf{r})}{\sigma[\mathbf{r}]}$ considers benefits and risks synthetically and reflects the excess return over unit systematic risk.

- Volatility **Vol** $:= \sigma[\mathbf{r}]$ (r denotes the historical sequence of return rate.) measures the uncertainty of return rate and reflects the risk level of strategies.
- Maxium Drawdown **Mdd** $:= max\frac{(P_i - P_j)}{(P_i)}, j > i$ measures the largest decline in history and shows the worst possible scenario.

4.4 Generalization Ability



(a) Dual Thrust

(b) iRDPG (from Figure 6)

(c) iRDPG (our results)

Figure 15: Generalizability of iRDPG on IF and IC

These figures present the generalizability of iRDPG agent. Figure(15-a) shows that if we apply the best parameter of Dual Thrust Strategy from IF to IC, it will fail as shown by the red curve. However, such failure will not happen by using the iRDPG agent. The iRDPG agent is trained in the IF training set and is tested in the IF and IC test set respectively. Figure(15-b) is the results from the paper Figure6, and Figure (15-c) is our implementation results. It is obvious that the agent got nice trading performance on both IF data and IC data. And such a trend in our results is consistent with the paper results. (Note: The reason why we can not successfully derive these results before the oral presentation on 6/23 is that we had different normalization settings for agent state value on IF and IC.)

5 Conclusion

Please provide succinct concluding remarks for your report. You may discuss the following aspects:

- The potential future research directions
- Any technical limitations
- Any latest results on the problem of interest

5.1 Consistent Conclusions with the Original Paper

1. Imitation learning techniques is successfully introduced to balance the exploration and exploitation of the trading agent.
2. The iRDPG suggests that the trading agent can benefit from experiences of classical trading strategies such as Dual Thrust Strategy.
3. The profitability and ability to resist risks of iRDPG were verified.
4. Comparison experiments provided its generalizability for different financial markets.

5.2 Technical Limitation and Possible Future works

Although the iRDPG agent can finally learn better policies than expert, this framework still needs well performed demonstrations in the early training stage. Therefore, in the future, maybe it can try other technique such as "policy optimization from demonstration (POfD)" (Kang et al. [2018]), which allows imperfect expert policies in the beginning.

References

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(98)00023-X. URL `https://www.sciencedirect.com/science/article/pii/S000437029800023X`.

Nicolas Heess, Jonathan J. Hunt, Timothy P. Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *CoRR*, abs/1512.04455, 2015a. URL `http://arxiv.org/abs/1512.04455`.

Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015b.

Yang Liu, Qi Liu, Hongke Zhao, Zhen Pan, and Chuanren Liu. Adaptive quantitative trading: an imitative deep reinforcement learning approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2128–2135, 2020.

Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

JoinQuant. Dataset link. URL `https://www.joinquant.com/help/api/help#name:Future`.

Bingyi Kang, Zequn Jie, and Jiashi Feng. Policy optimization with demonstrations. In *International Conference on Machine Learning*, pages 2469–2478. PMLR, 2018.