Combining (Deep) Reinforcement Learning and Goal Based Investing

Yuwen Jin, Minghao Kang, Han Luo

Advisor: Dr. Cristian Homescu, Zhenyu Cui

School of Business

Stevens Institute of Technology

May 2021

Abstract

These years Reinforcement learning (RL) has become to an efficient exponential methodology, Lots of investment firm and bank are using it to build their trading and investment strategy. Goal-Based Investing (GBI) is an investment approach where performance is measured by the success of investments in meeting the investor's financial goals. In this paper, we implemented RL in GBI to help individuals reach their wealth managing goal, namely we formed a financial plan. Compared with traditional portfolio investment, our strategy has significant improvement and better performance.

1 Introduction

Goals-Based Investment (GBI) refers to the management of an investor's portfolios with a view to meeting long-term financial goals, as opposed to only optimizing a risk-return trade off(Das et al. (2019)). Different from traditional approach to the portfolio optimization problem which aims to minimize the overall portfolio risk, the responsibility of GBI is to maximize the probability of achieving the wealth goal.

Reinforcement Learning is the task of learning how agents ought to take sequences of actions in an environment in order to maximize cumulative rewards(François-Lavet et al. (2018)). It connect the environment and policy to make a learning procedure, through repeated experience and multiple decision, finding an optimal policy, and generating a long term strategy that can maximize cumulative reward. In a given environment, the agent policy provides some intermediate and terminal rewards, while the agent learns sequentially. When an agent picks an action, she can not infer ex-post the rewards induced by other action choices. Agent's actions have consequences, influencing not only rewards, but also future states of the world.

In this project, we implemented dynamic programming approach via reinforcement learning (RL), a paradigm of learning by trial-and-error, solely from rewards or punishments, to GBI. On setting investing goal, we formed a financial plan which helps clients on wealth management, and guarantee specific amount of earning every week. Our object is to ensure profits for weekly payment.

The algorithms we utilized are Deep Q network(DQN), an enhancement of Q-learning with neural network but is still a value-based method, and Deep Deterministic Policy Gradient (DDPG), which adapts to continue action space situations. We set average performance of the stock(s) and also we invest as benchmark.

By hyper-parameter tuning on RL algorithm and neural network modification, we get results as expected: the performance of portfolio under RL algorithm is better than the benchmark. This paper is

to prove that combining RL and GBI is efficient, and that RL can avoid factors of human behavior in the traditional approaches.

The remaining structure of this paper is as follows: in Section 2, we will introduce the methodology we used, including investment strategy(2.1), reinforcement learning algorithms(2.2) and hyperparameter tuning methods(2.3). Section 3 contains our practice result from both numerical and graphic aspect, including corresponding analysis. In Section 4, we will make a conclusion base on our result, and introduce the potential of our project and the direction of its development in the future.

This is the performance comparison of the 2 algorithms we used, together with average stock performance as benchmark:

2 Methodolog

2.1 Investment strategy

2.1.1 Goal Based Investing

GBI focuses on investor's wealth goal rather than the minimum risk or maximum return, meaning that investor can set a wealth goal which does not have to be a single ultimate goal. It can be a wealth plan that allows investor can take out money or infuse capital anytime, and still can achieve goal in the end. Such as the short-term car loan, or the long-term retirement plan. All these situations require plan that allows investors to withdraw or save money periodically over a period of time and ultimately achieve the financial growth they set as the goal at the beginning. Therefore, different with MPT, the risk and expected return is different in each small period of the whole period. If investor got a good performance in the previous period, the current expected return and risk preference would be decrease, Because the financial pressure to achieve the final goal is decrease because of the performance at the last stage.

Different from utility-based portfolio problems, the essence of GBI is to find the dynamic portfolio strategy that maximize a goal-based objective function. The GBI objective function expresses the probability that investor make profit no less then a pre-setted goal C_t at time t. Its mathematical representation is as follows(Das et al. (2019)):

$$\max Prob[W_t \geq C_t]$$

where C(0) = 0. At the beginning, in order to evaluate the initial portfolio wealth $W(0)$, developer use stochastic model. And as we mentioned previously that the strategy should allow investors withdraw money or consumption periodically, consumption take-out or additional investment should also be considered.

In this project, we formed a financial plan which takes $100,000$ dollars as initial wealth, and provide weekly payments of 200 dollars (guarantee 0.2% profit weekly) to clients.

2.1.2 Markowitz Mean-Variance Portfolio Theory

In general, Markowitz Portfolio Theory(MPT) is the main theory when people are trying to solve the weight allocation problem. It is a mathematical framework for assembling a portfolio of assets such that the expected return is maximized for a given level of risk, or the risk with a given expect return is minimized.

$$\frac{1}{2}\omega^T \Sigma \omega - \lambda \omega^T \rho$$

Where

$$\textit{covariance matrix: } \Sigma = var(\sum_i x_i r_i) = \sum_{ij} x_i x_j cov(r_i, r_j)$$

and the constrain is

$$\sum_i \omega_i E(\rho_i) \geq \mu, \sum_i \omega_i \leq 1, \omega_i \geq 0$$

The reason why we introduce mean-variance here is because in the second stage of our DQN method, we use an optimizer outside DQN objects to allocate money among different stocks environments, in

order to move from single stock to portfolio. And this is the key that our DQN method able to handle portfolio management.

2.2 (Deep) Reinforcement Learning (RL)

Reinforcement learning is an area of machine learning. The main idea of the reinforcement learning(RL) is that an artificial agent may learn by interacting with its environment, similarly to a biological agent (Irlam (2020)). Markov Decision Process(MDP) is the main basic idea be applied in RL. Its definition is that the decision maker may always choose the action base on the current state. This means that the whole decision process is an iteration procedure. Decision makers never know the consequences of their decisions before they make them. Mathematically, it can be expressed as following:

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_t, s_{t-1}, s_{t-2}, ..., s_1, s_0)$$

where P is a probability that relative to the current state and current action. S is all the indicators that can be observed at the current time point.

After we find the expression of each state, we have to value each state s(t) with a value function R(t). And here exist another important factor discounted value. Because of the time series, we need to think about the value of the profit at time t to time t+ I. Because of human behavior and financial markets factors, this value tends to decrease over time, and that's the importance of the discount value. Therefore we cite discounted value to represent the real time profit R to the expected return in the future:

$$G_t = R_{t+1} + \gamma R_{t+2} + ...\gamma^k R_{t+k+1} = \sum_{n=0}^{\infty} \gamma^k R_{t+k+1}$$

in which G is the cumulative return and γ is the discount factor. And the goal of RL is maximize the cumulative return to achieve its wealth goal.

2.2.1 Deep Q-Learning (DQN)

DQN method is one of the most import method in RL. Recall Bellman function(Ivanov and D'yakonov (2019)):

$$v(s) = E[R_{t+1} + \lambda v(S_{t+1}|S_t = s)]$$

The goal of DQN method is to minimize the following loss:

$$L(\omega) = E[(r + \gamma \max_{a}^{'} Q(s', a', \omega) - Q(s, a, \omega))^2]$$

To take both exploration and exploitation into consideration, we use $\epsilon - greedy$ policy, meaning that with specific probability we choose action randomly, otherwise we follow decision of our agent.

RL Settings

- Environments: each DQN envorinment-agent pair is used to represent a single stock.
- State s: state features contains daily stock return, indicators including RSI, KDJ, BOLL and moving average, in a fixed window.
- Action a: We provide 3 actions with this model: buy(with all remained money), sell(flat), and wait(do nothing).
- Reward r: we set our utility as a combination of goal, profit and risk:

$$reward = -\lambda \mathbb{I}_{(t>5, \frac{cp_t}{cp_{t-5}}<0.002)} + y_t - ra * \sigma$$

  The first term is the expected negative reward for underachievement of the target goal where cp stands for 'cumulative profit', ra is risk aversion and σ stands for volatility.

Pseudocode see below:

Algorithm 1 Deep Q-learning (Ivanov and D'yakonov (2019))

$\boldsymbol{B}$ - batch size, $\boldsymbol{K}$ - target network update frequency, $\epsilon(t) \in (0, 1]$ - greedy exploration parameter, Q^*_θ - neural network, SGD - optimizer.

Initialize weights of θ arbitrary, $\overline{\theta} \leftarrow \theta$
for each interaction step t **do**
    select a randomly with probability $\epsilon(t)$, else $a = \underset{a}{\operatorname{argmax}}\, Q^*_\theta(s, a)$
    observe transition $(s, a, r^{'}, s^{'}, done)$
    add observed transition to experience replay;
    sample batch of size $\boldsymbol{B}$ from experience replay;
    for each transition $\boldsymbol{T}$ from the batch compute target:

$$y(T) = r(s^{'}) + \gamma \underset{a^{'}}{max}\, Q(s^{'}, a^{'}, \overline{\theta})$$

    compute loss:

$$Loss = \frac{1}{B} \sum_{T} (Q^*(s, a, \theta)) - y(T))^2$$

    make a step of gradient descent using $\frac{\partial Loss}{\partial \theta}$
    if t mod $\boldsymbol{K}$ = 0: $\overline{\theta} \leftarrow \theta$
end for

With DQN algorithm, we first implement it on single stock and compare the agent's performance with the stock movement. While moving forward to portfolio, we created multiple environment-agent pairs, each pair for one single stock and nested them into an optimizer which takes mean-variance model as objective. Every 15 trading days we reallocate the money to realize dynamic allocation.

2.2.2 Deep Deterministic Policy Gradient(DDPG)

Deep learning models can be used in reinforcement learning to solve high-dimensional problems(Snow (2020)). To deal with the continuous action space of portfolio weighting, we can use Google DeepMind's off-policy and model free algorithm called deep deterministic policy gradient(Lillicrap et al. (2015)).

DDPG combines both Q-learning and Policy gradients. DDPG being an actor-critic technique consists of two models: Actor and Critic. The actor is a policy network that takes the state as input and outputs the exact action (continuous), instead of a probability distribution over actions. The critic is a Q-value network that takes in state and action as input and outputs the Q-value.

In this case we have 2 loss functions, loss function for Critic (Q) and Actor (mu):

$$J_Q = \frac{1}{N}\sum_{i=1}^{N}(r_i + \gamma(1-d)Q_{targ}(s^{'}, \mu_{targ}(s_i^{'})) - Q(s_i, \mu(s_i)^2)$$

$$J_\mu = \frac{1}{N}\sum_{i=1}^{N} Q(s_i, \mu(s_i)^2)$$

We first analyze the Actor (policy network) loss. The loss is simply the sum of Q-values for the states. For computing the Q values we use the Critic network and pass the action computed by the Actor-network. We want to maximize this result as we wish to have maximum Q-values. The Critic loss is a simple TD-error where we use target networks to compute Q-value for the next state. We need to minimize this loss. To propagate the error backwards, we need derivatives of the Q-functions. For the critic loss the derivatives of Q-values are straightforward as μ is treated as constant, however, for actor loss the mu-function is contained inside Q-value. For this, we would use the chain rule,

$$J_\mu = E[Q(s, \mu(s)]$$

$$\bigtriangledown_{theta^\mu} J_\mu = E[\bigtriangledown_\mu Q(s, \mu(s)\ \bigtriangledown_{theta^\mu}\ \mu(s)]$$

RL Settings To start formalizing the problem, we set the number of stocks to N.

- Environment: the customized environment is based on OpenAI gym (Jiang et al. (2017)), and the RL agents are trained with tensorflow.

- State s: state features contains (1) $y_{c,o} = \frac{close\ price}{open\ price}$, (2) $y_{h,l} = \frac{high\ price}{low\ price}$, (3) $y_v = trading\ volume$

  Specifically for cash maintained, $\frac{high\ price}{low\ price}$ is always 1 and we don't really care its other features.

- Action bais a_b: $a_g \leftarrow a_b \sim \mathcal{N}(a_g, \sigma^2)$ with a decreasing probability

  similar to $\epsilon - greedy$ policy in DQN, in DDPG we add random bais by replacing action a_g with a_b, where σ^2 is pre-setted bias variance, and the probability of adding the bais is decreasing with training processing, meaning that we trust more on thee algorithm

- Action a: the vector of positions(weights) on each assets,

$$\omega_t = [\omega_{c,t}, \omega_{1,t}, ..., \omega_{N,t}]$$

Here $\omega_{i,t}$ is the fraction of investment in stock i at time stamp t. and $\omega_{c,t}$ represents the fraction of cash maintained. All weights fall between $(0, 1)$ and sum up to 1. Then the cumulative gain (PnL) can be written as:

$$PnL = \prod_{t=1}^{T} y_{c,o,t} \times \omega_{t-1}$$

- Reward r:

$$r_t(s_t, a_t) = R_t - \lambda E_t|C_{t+1} - \eta \times max(0, \forall a_t - 0.7)$$

The first term is log return of the account, the second term is the expected negative reward for underachievement of the target goal, and the third term is penalty on concentrated investment because a high proportion on a single stock will lead to higher risk.

Pseudocode see below:

Algorithm 2 Deep Deterministic Policy Gradient

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network $Q^{'}$ and $\mu^{'}$ with weights $\theta^{Q^{'}} \leftarrow \theta^Q$, $\theta^{\mu^{'}} \leftarrow \theta^\mu$
Initialize replay buffer $\boldsymbol{R}$
for episode = 1, $\boldsymbol{M}$ **do**
    Initialize a random process $\mathscr{N}$ for action exploration
    Receive initial observation state s_1
    for t = 1, T **do**
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathscr{N}_t$ according to the current policy and exploration noise
        Execute action at and observe reward r_t and observe new state s_{t+1}
        Store transition $(s_t, a_t, r_t, s_{t+1}, done)$ in R
        Sample a random minibatch of N transitions $(s_i, a_i, r_i, s_{i+1}, done)$ from R
        Set $y_i = r_i + \gamma Q^{'}(s_{i+1}, \mu^{'}(s_{i+1}|\theta^{\mu^{'}})|\theta^{Q^{'}})$
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
        Update the actor policy using the sampled policy gradient:

$$\bigtriangledown_{\theta^\mu} J \approx \frac{1}{N}\sum_i \bigtriangledown_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \bigtriangledown_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

        Update the target networks:

$$\theta^{Q^{'}} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q^{'}}$$

$$\theta^{\mu^{'}} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu^{'}}$$

    end for
end for

2.3 Hyperparameter Tuning

While training (deep) reinforcement learning algorithm, there are some hyper-parameters we want to consider:

- epoch/episode: each loop that implement the policy in the environment from start to finish is considered an episode
- Batch size: the size while sampling from memory(buffer)
- Window length: the length of rolling window to package stock information in a state
- learning rate: when the gradient descent algorithm is used for optimization, the weight update rule is multiplied by a coefficient in front of the gradient term, which is called the learning rate

- action bias var(DDPG): the variance we use while adding exploitation component(bias) to generated action (mean is the generated action)
- gamma: discount factor
- tau: the mixing ratio of the newly added elements while softly updating target networks
- ...

Sometimes RL algorithms are very sensitive to data set and these hyper-parameters. Therefore we need hyper-parameter tuning.

Generally, there are 4 common way to do hyper-parameter tuning: the grid search, random search, gradient-based optimization, and bayesian optimization. Other than these, sometimes people use heuristic Tuning, in which people select hyper-parameters based on previous experience.

The goal of hyper-parameter tuning is to find the optimal group of hyper-parameter:

$$x^* = \underset{x \in \mathscr{X}}{\operatorname{argmin}} f(x)$$

where $f(x)$ is the objective function to minimize. If we want to minimize the loss of our network, we set the loss as $f(x)$. If we want to maximize Q value then we set negative Q as $f(x)$

2.3.1 Bayes Optimization

When there are too many parameters, methods like grid search will need really long time to go through every possible values.Bayesian optimization assumes that there is a functional relationship between the hyper parameters and the final loss function we need to optimize. Bayesian optimization is a sequential design strategy for global optimization of black-box functions that does not assume any functional forms. It is usually employed to optimize expensive-to-evaluate functions.

Pseudocode see below:

Algorithm 1 Sequential Model-Based Optimization

Input: $f, \mathcal{X}, S, \mathcal{M}$
$\mathcal{D} \leftarrow \text{INITSAMPLES}(f, \mathcal{X})$
for $i \leftarrow |\mathcal{D}|$ **to** T **do**
  $p(y \mid \mathbf{x}, \mathcal{D}) \leftarrow \text{FITMODEL}(\mathcal{M}, \mathcal{D})$
  $\mathbf{x}_i \leftarrow \arg\max_{\mathbf{x} \in \mathcal{X}} S(\mathbf{x},\ p(y \mid \mathbf{x}, \mathcal{D}))$
  $y_i \leftarrow f(\mathbf{x}_i)$ ▷ Expensive step
  $\mathcal{D} \leftarrow \mathcal{D} \cup (\mathbf{x}_i, y_i)$
end for

Figure 1: Bayes Optimization

3 Results & Analysis

In this section we will show all the result we have, including good results, bad results, why they are good or bad and what we did to those bad results. We will show it chronologically.

All data we use is from Yahoo Finance and with 10 years horizon.

3.1 Deep Q-Learning on single stock

Parameter's value has a huge impact to Reinforcement learning's result, so, before we train our agent, we have to find out what are the best value of parameters. The are 3 parameters that we consider to modify

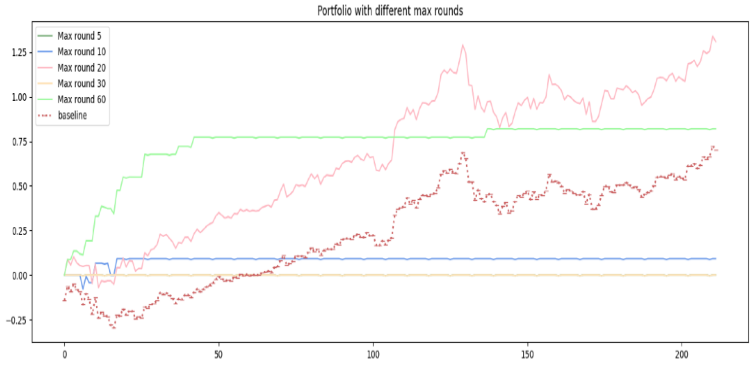

Figure 2: Max Training Rounds

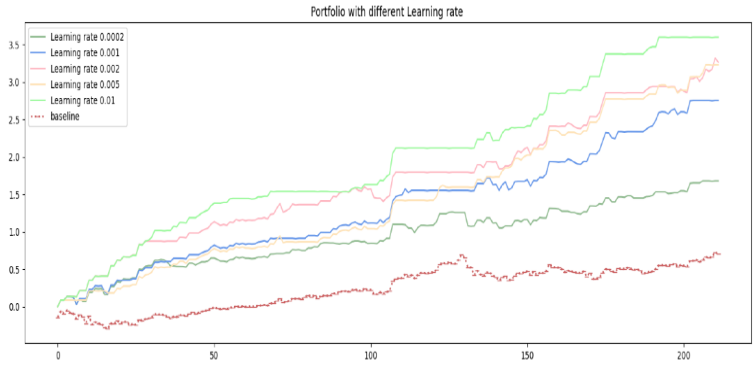

Figure 3: Learning Rate

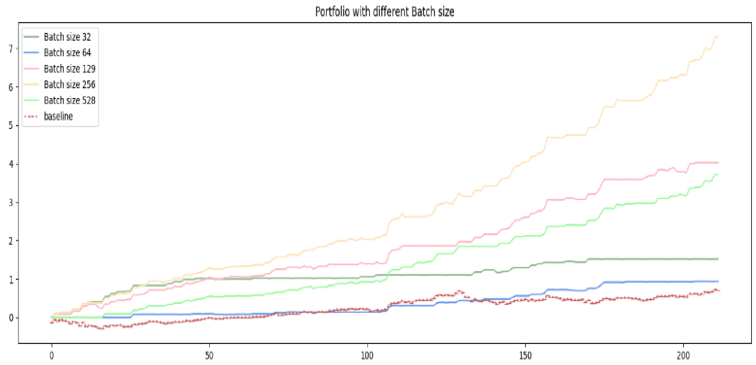

Figure 4: Batch Size

After we test different value of parameters, we pick the value of parameters that gives us the best result and combine them together for the real training process. The values that gives us the best result:

1) Max round =20
2) Learning rate = 0.001
3) Batch size = 256

After we decide hyper-parameters, we train our agent use ATVI stock information from 01-01-2011 to 12-31-2019 and test it on time period from 01-01-2020 to 12-31-2020. The agent performance on testing period is shown below:

In figure 5 and figure 6 the x axis is the trading days, and the y axis is return rate. As you see from these two plots, our strategy is out performance the benchmark and in the trading plots, we can see

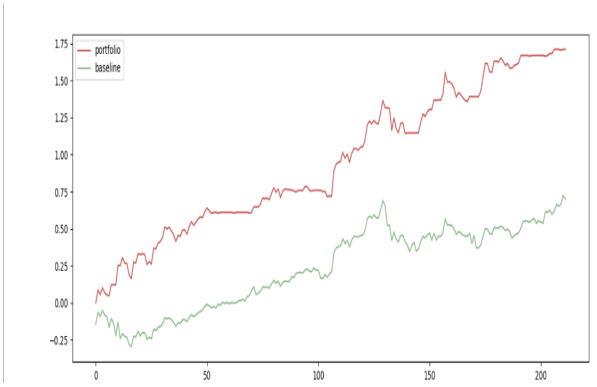

Figure 5: performance

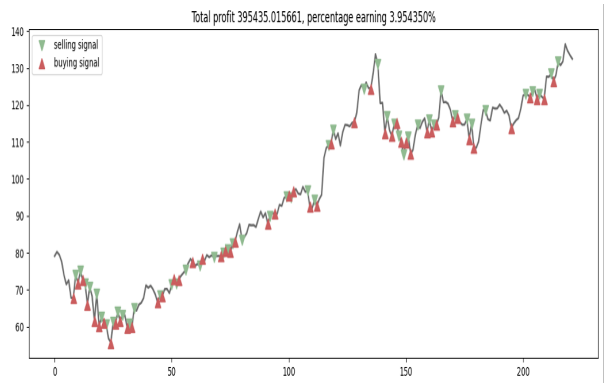

Figure 6: trading signal

our agent following the rule of buy low and sell high. More detail, In the beginning of testing period, the stock price of ATVI is dropping however, our agent's account value is increasing, it means our agent has ability to predict the trend of stock.

In figure 6, it shows when our agent trade and what is the action. Most of time our agent is buying low and selling high and there are lots of trades which means our agent did the right decision most of time because our account value is increasing. In the end of one testing period, our account value increased to 1.75 times initial value. Compare with the benchmark, it is our performance.

3.2 Deep Q-Learning on portfolio management with outer optimizer

After we train our agent in single stock, we think it is more important that to let our agent to create a portfolio, by adding a optimizer, and create different neural networks, our agent finally able to invest in more than one stocks.

Figure 7 is for 3 stock, he blue line is our account value, it shows that our portfolio is out performance of all those stocks. In the beginning 130 days, it trades more than rest of treading days. The good thing is it also didn't drop down when stocks are dropping. Figure 8 is portfolio for 5 stocks, in this graph, the red line is our account value, it doesn't performance well compare with 3 stock portfolios. But in the last 1/3 of trading days, it increases a lot because of the optimizer allocate most of money into one stock that increase a lot. Although the result is fine, but action like this will increases the risk

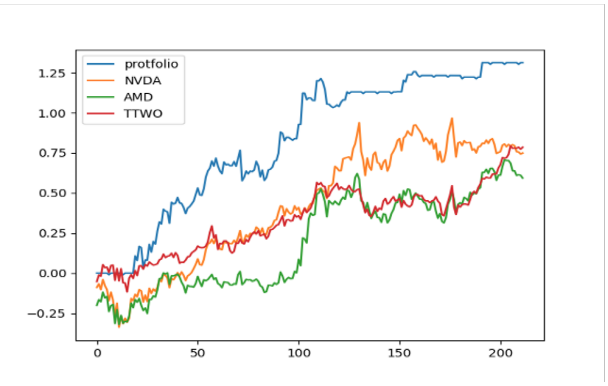

Figure 7: Three stock portfolio

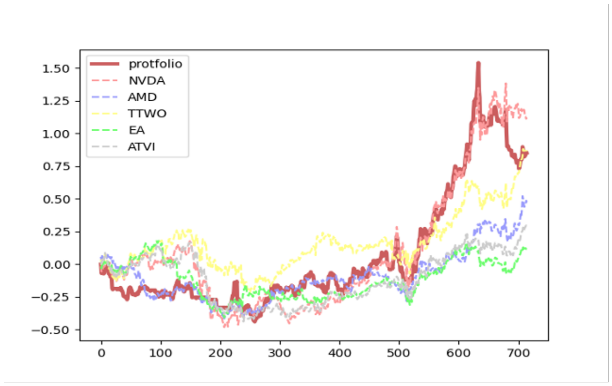

Figure 8: Five stock portfolio

of portfolio

3.3 Deep Deterministic Policy Gradient

Moving forward to DDPG, we now use continuous action space to represent positions(investing weights) on each assets. For This algorithm, our goal is let it able to return weights for each stock which means it can return actions that is continue. This is also the main different between DQN and DDPG We have test this DDPG algorithm as soon as we finish implement it. Without anything modification of parameters, it doesn't gives us a good result, at least, it is not as good as DQN's final result

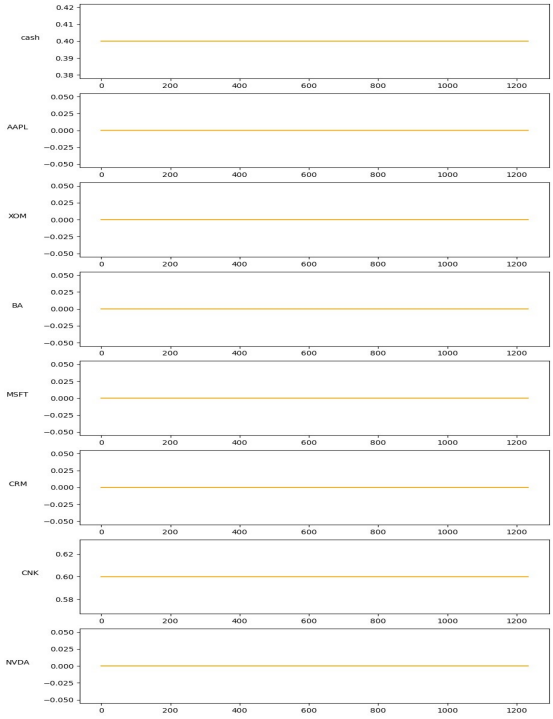

Figure 9: Weights

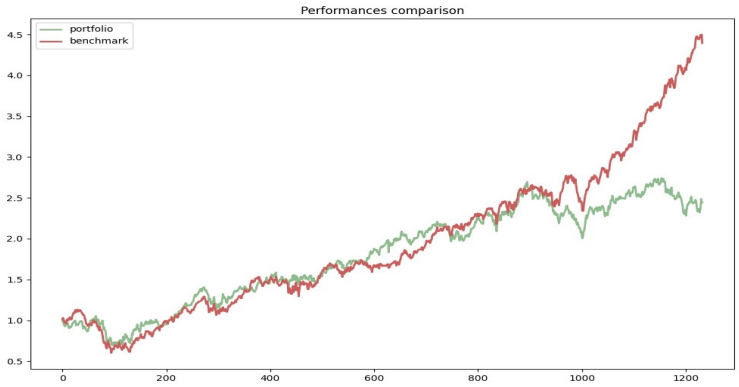

Figure 10: performance

According to figure 10, the cumulative gaining of our portfolio is almost the same as the benchmark at the first 2/3 of the testing period, how ever it fell behind in the last 1/3 of testing process. And according to figure 9, there are no significant(visible in plot) weight changes anywhere furing the

whole period.

In order to exclude the influence of non-optimal hyper-parameters, we did hyper-parameter tuning via Bayesian Optimization method, and with open source code in Python.We tried both to minimize loss in critic network, and to maximize Q value.

iter	target	action...	actor_...	batch ...	critic...	gamma	tau	window...
1	-0.000245	0.4363	0.008576	349.5	0.009739	0.9572	0.000361	24.02
2	-0.000161	0.7814	0.004135	293.1	0.005615	0.9743	0.000888	30.64
3	-0.000155	0.2672	0.005114	67.52	0.003106	0.9696	0.000680	10.04
4	-0.000174	0.526	0.005321	494.7	0.003478	0.9858	0.000287	10.59
5	-9.255e-0	0.6385	0.007018	283.0	0.005819	0.9609	0.000218	22.63
6	-5.196e-0	0.2533	0.008418	64.08	0.002391	0.9538	0.0001	5.278
7	-0.000186	0.5859	0.007947	512.8	0.009688	0.9879	0.0001	5.808
8	-0.000237	0.3544	0.005366	65.3	0.006935	0.9633	0.0001	30.66
9	-0.000108	0.2263	0.005038	510.3	0.003243	0.9634	0.0001	30.74
10	-7.274e-0	0.6996	0.003457	65.53	0.003971	0.98	0.0001	5.503
11	-0.000188	0.06128	0.000113	512.5	0.007113	0.9682	0.0001	5.231
12	-0.000118	0.6237	0.006446	64.56	0.001986	0.9556	0.0001	30.86
13	-0.000205	0.7961	0.007591	509.7	0.002144	0.9604	0.0001	30.83
14	-0.000313	0.1115	0.007651	67.16	0.001923	0.9666	0.0001	5.382
15	-0.000604	0.7275	0.006639	509.5	0.003231	0.9896	0.0001	5.392
16	-8.02e-05	0.4591	0.006798	65.52	0.008694	0.961	0.0001	30.83
17	-0.000316	0.5872	0.002812	65.31	0.00102	0.9626	0.0001	30.45
18	-0.000158	0.7343	0.003042	509.9	0.009835	0.988	0.0001	30.88
19	-5.354e-0	0.4558	0.009497	64.46	0.005047	0.9507	0.0001	5.373
20	-0.000105	0.09723	0.003215	512.6	0.009807	0.9701	0.0001	5.05
21	-0.000125	0.7794	0.002345	64.2	0.001865	0.9663	0.0001	5.654
22	-8.503e-0	0.06896	0.001482	511.7	0.006047	0.9631	0.0001	30.91
23	-0.000115	0.666	0.002406	66.08	0.008937	0.9804	0.0001	30.83
24	-8.459e-0	0.6811	0.007013	512.4	0.008375	0.9766	0.0001	5.119
25	-0.000201	0.507	0.003513	64.93	0.005497	0.9771	0.0001	30.71
26	-6.408e-0	0.1825	0.002517	512.7	0.000982	0.969	0.0001	30.84
27	-3.746e-0	0.6424	0.001288	65.58	0.004778	0.9891	0.0001	5.351
28	-0.000167	0.4448	0.009175	511.1	0.005531	0.9776	0.0001	5.126
29	-4.015e-0	0.7372	0.009621	65.55	0.00869	0.9676	0.0001	5.093
30	-0.000177	0.6014	0.007024	69.46	0.000767	0.963	0.0001	30.92

Figure 11: Bayesian Optimization minimize critic loss

iter	target	action...	actor_...	batch ...	critic...	gamma	tau	window...
1	0.001133	0.3352	0.004515	94.9	0.006377	0.9794	0.000840	17.11
2	-0.000692	0.543	0.006345	251.2	0.006693	0.9864	0.000558	15.28
3	0.000292	0.1154	0.005513	419.8	0.006724	0.9778	0.000576	23.37
4	0.001473	0.2238	0.007259	258.0	0.000638	0.9851	0.000227	16.29
5	-0.000385	0.2955	0.007162	108.2	0.002323	0.9872	0.000956	7.603
6	0.01064	0.596	0.009827	512.0	0.009799	0.9546	0.0001	6.191
7	0.01688	0.4092	0.009338	511.9	0.001578	0.967	0.0001	29.1
8	0.006029	0.2338	0.005388	512.9	0.00548	0.9685	0.0001	30.28
9	0.003147	0.6193	0.003364	162.4	0.001007	0.9685	0.0001	30.72
10	0.000966	0.2904	0.007324	462.7	0.006091	0.987	0.0001	5.073
11	-0.000214	0.3785	0.007952	341.2	0.004195	0.9715	0.0001	5.118
12	0.002191	0.7072	0.001898	512.6	0.005649	0.9625	0.0001	30.6
13	0.000574	0.07187	0.004559	65.22	0.005884	0.9766	0.0001	5.116
14	-2.262e-0	0.3515	0.006591	340.8	0.004917	0.9832	0.0001	30.94
15	0.008226	0.3023	0.002863	470.7	0.000437	0.9669	0.0001	30.79
16	-0.000198	0.06058	0.008035	183.9	0.004995	0.9503	0.0001	5.022
17	0.001581	0.6611	0.005343	64.32	0.006566	0.9681	0.0001	30.34
18	0.009356	0.4018	0.00952	512.2	0.005924	0.9806	0.0001	5.321
19	0.002729	0.3816	0.003334	215.9	0.00246	0.9746	0.0001	30.81
20	0.000922	0.08872	0.000411	382.2	0.009647	0.981	0.0001	5.003
21	0.00256	0.1193	0.00164	493.0	0.009808	0.9658	0.0001	17.84
22	-8.613e-0	0.4801	0.009738	302.9	0.000287	0.9807	0.0001	30.87
23	0.001533	0.4301	0.001257	293.8	0.006542	0.9671	0.0001	5.106
24	-0.000529	0.4562	0.006939	512.7	0.00219	0.9679	0.0001	5.533
25	0.005036	0.5973	0.003947	383.9	0.001339	0.971	0.0001	31.0
26	0.002772	0.4378	0.00469	123.7	0.000840	0.9713	0.0001	30.94
27	0.005442	0.1362	0.007364	150.8	0.003285	0.9529	0.0001	5.001
28	0.006784	0.7938	0.002771	221.4	0.000504	0.9518	0.0001	5.344
29	0.000153	0.2388	0.007998	424.8	0.001077	0.988	0.0001	5.07
30	6.408e-0	0.6803	0.008424	444.0	0.001946	0.9677	0.0001	30.59

Figure 12: Bayesian Optimization maximize Q value

Output above in figure 11 and figure 12 are generated with function in Python package 'bayesian-optimization'. Everytime when a new record appears to be pink, we update our temporary optimal hyper-parameter. We didn't have enough time for large K cross validation so we leave that to future

work.

For now, on observing the difference between these 2 optimal hyper-parameter group, we found that when we want to minimize loss, both batch size and rolling window length are close to their lower bound, whereas when we want to maximize Q value, both batch size and rolling window length are close to their upper bound. So we think it may because that maximizing Q value needs more information, while minimizing loss prefer more concise data. We leaf the proof to future work too, and take the hyper-parameter according to figure 12 to fit the model again.

Figure 13: Performance after hyper-parameter tuning

The agent still not performance as well as we expect. Finally we realized that it's because the networks we set up are too complex for our data set. Initially, we design our network structure according to the reference (Snow (2020)). To verify the thoughts, we gradually reduced hidden layers and the neurons in both actor and critic networks. we do see more convincing training process, and finally it works as expected on our data set(figure 14, 15).

Now, our portfolio is much better than before, and it is over performance the benchmark. More importantly, the allocation weights is changing so our DDPG agent finally becomes able to realize dynamic allocation in multi-stocks using. Also, we have some numerical result for this testing period(figure 16).

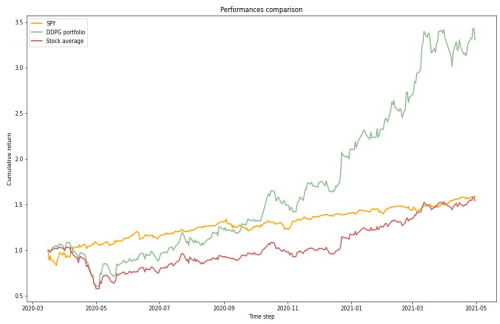

Figure 14: performance

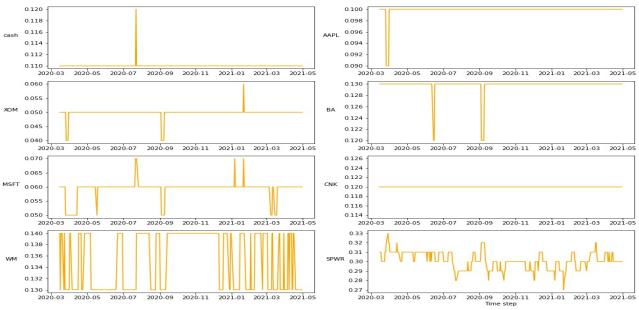

Figure 15: Actions (Weights)

	strategy	benchmark
PnL	1.323625	0.515160
Daily Mean Arithmetic	0.003402	0.001825
Volatility	0.029951	0.027104
Max 10 days draw down	0.512394	0.555372
Skewness	1.114219	0.199409
Kurtosis	9.329003	4.846490
VaR	-0.035899	-0.036303

Figure 16: numerical result for DDPG

According to the table (figure 16), PnL of our DDPG algorithm is much higher than the goal we set and max 10 day's draw down is even lower than the benchmark given that the volatility is close. Also the VaR of benchmark and our portfolio are close as well.

3.4 Comparison between DQN and DDPG

To make fair and reliable comparison, we use the same data set (same stock pool and testing period) to test our 2 algorithms. The stocks we choose are: EA, TTWO, AMD, NVDA, ATVI.

Graphic results

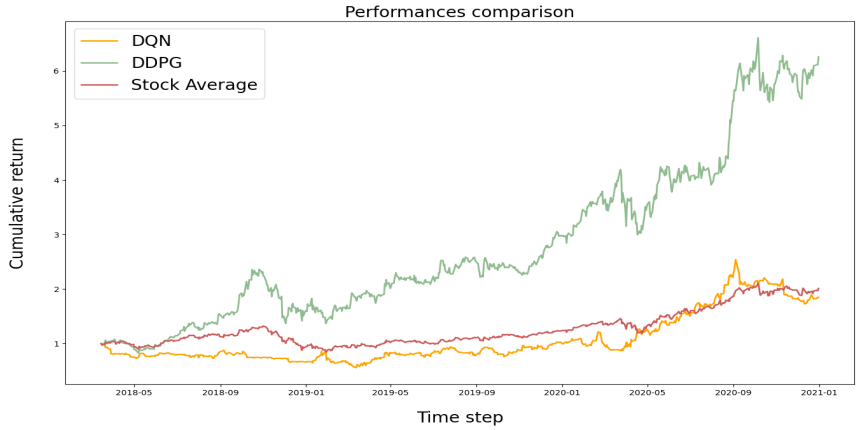

Figure 17: Comparison among two algorithms and stock average performance

From the figure we see DDPG makes much more profit than both DQN and stock average, while DQN is actually not as good as stock average.

Numerical results

	DDPG	DQN	Stock average
Cumulative PnL	5.410469	1.84949	2.011380
Daily Mean Arithmetic	0.003128	0.0023128	0.001164
Volatility	0.032501	0.0334	0.018591
Max 10 days draw down	0.432204	0.219	0.250432
Skewness	-0.211859	-0.1356051	-0.237946
Kurtosis	3.979704	3.494826	2.781880
VaR	-0.042335	-0.041870	-0.027583
Sharpe Ratio	135.18	24.92	53.49

Where risk free rate in calculation of Sharpe Ratio is 1.7%, approximate to the rate of 10-year U.S. treasury bound.

On observing the statistics, with this specific data set, DDPG makes a lot more profit than the other

two strategies, while it has similar volatility. For Max 10 days drop down, DDPG method has higher Max 10 days drop down which means it has higher risk. However, the gap of max 10 days drop down between DDPG DQN is not as big as the gap of PnL. So, the marginal value of DDPG is higher than DQN. From this table, we can say that DDPG is better than DQN because the PnL is way higher than DQN's PnL and risk is not very high at meantime. Also, DDPG method has very high Sharpe ratio compare with DQN method, it mean use 1 unit's risk, investor will get more return.

However, we also test DDPG on single stock because of the complexity problem we had before, and this time it performances not as well as expected:

Figure 18: DDPG single stock

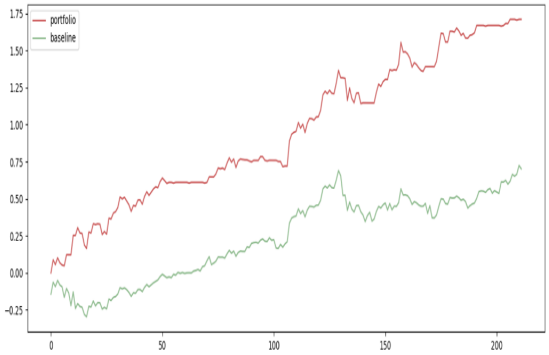

Figure 19: DQN single stock

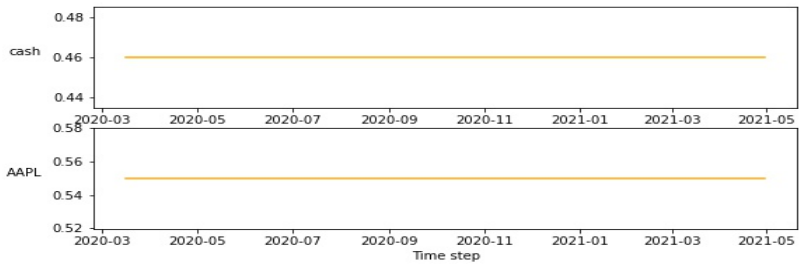

Figure 20: DDPG single stock weight

According to figure 18 and 19, DQN performances way better than DDPG. And from Figure 20, we

see DDPG agent does not trade at all. In this case we think it;s possible that as the complexity of data decreasing, the performance of DDPG method is deceasing as well. Similar to that we think as complexity of data increasing, the performance of DQN method gets worse.

We then test DDPG on 2 stocks, this time it becomes better then on single stock:

Figure 21: DDPG two stock

Figure 22: DQN two stock

According to figure 21 and figure 22, DDPG method starts trading under two stocks environment, and the performance is better(beats benchmark). To move forward, we may adapt to more test stock pools to verify our thoughts.

4 Conclusion

To summarize, we implemented Reinforcement Learning algorithm on goal-based investing, namely a weekly paid financial plan. DQN and DDPG are both included in our project. To be specific, DQN method returns discrete actions so we build different neural networks for each stock and nested an optimizer outside to realize dynamic allocation. DDPG method adapts to continuous action space so we pass data of multiple stocks into one DDPG object and take its returns as weights on cash and each stock. Both of the two algorithms are sensitive to the data and hyper-parameters. Moreover, while training DDPG agent, compatibility of complexity of data and neural networks matters a lot. So we applied Bayes Optimization on hyper-parameter tuning. However, we still have room for improvement on this.

According to the results we have, we found that our DDPG method has better performance than DQN. And both DQN and DDPG are over performance than the stock average. Also, our DQN method did good job on invest in single stock. It makes us know that our reward function is suit for this investing problem

To move forward, first we could take retirement plan instead of financial plan as our objective. We may allow 'additional invest' halfway, and consider 'present' value of future cash flow as temporary goal C_t instead of rolling profit. Second, we can improve our hyper-parameter tuning approach in terms of efficiency and sufficiency. We may try to find out appropriate way to measure complexity of data sets and of neural networks. In this case we may see some numerical relationship between them, and thus be able to provide suggestion on neural network design ad well as verifying our guess on relationship between data complexity and RL(DQN/DDPG) algorithm performance. Moreover, we could try more (deep) reinforcement learning methods in investing industry, explore more possibility of the combination.

References

Das, S. R., D. Ostrov, A. Radhakrishnan, and D. Srivastav (2019). Dynamic portfolio allocation in goals-based wealth management. *Computational Management Science*, 1–28.

François-Lavet, V., P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau (2018). An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560*.

Irlam, G. (2020). Multi scenario financial planning via deep reinforcement learning ai. *Available at SSRN 3516480*.

Ivanov, S. and A. D'yakonov (2019). Modern deep reinforcement learning algorithms. *arXiv preprint arXiv:1906.10025*.

Jiang, Z., D. Xu, and J. Liang (2017). A deep reinforcement learning framework for the financial portfolio management problem.

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Snow, D. (2020). Machine learning in asset management—part 2: Portfolio construction—weight optimization. *The Journal of Financial Data Science 2*(2), 17–24.