ENSAE 2020 Informatic Project
Application of Reinforcement Learning on Asset Allocation

PRUGNIAUD Melchior
ENSAE Paristech
melchior.prugniaud@ensae.fr

TAN Jing
ENSAE Paristech
jing.tan@ensae.fr

In recent years, AI has become more and more popular in quantitative finance. One main subject is the applications of machine learning algorithms into portfolio management. In this project, we focus on asset allocation with a deep reinforcement learning framework. We apply the deterministic policy gradient algorithm and constructed deep neural networks to optimize the portfolio return.

Background

Asset allocation in portfolio management is a process of adjusting the weights of assets. In the classical portfolio management theory, we use historical data to explore portfolio composition with the objectives of maximized portfolio return, minimized portfolio risk or simultaneously both of them. Other optimisation criteria include portfolio diversification, equal risk contribution, etc. Classical portfolio theories such as Markowitz model is often criticized for using historical data and gaining no insight into future behavior.

In real-world, the future doesn't copy the past, since the process is stochastic and usually impacted by exogenous change of market condition, especially when we have a small time interval between prices and look into the micro-structure of asset prices. Under the portfolio optimization framework, machine learning methods can work as complements of the objective function. For example hierarchical clustering algorithm can used to regroup assets and reallocate risks over clusters such called the hierarchical risk parity portfolio. They can also be used to exploit future portfolio return and approximate the reward function such as the implementation of reinforcement learning.

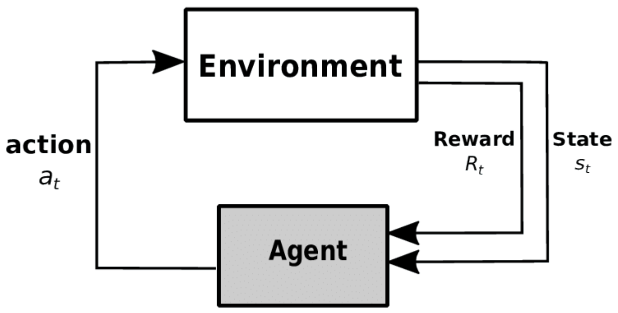


Figure 1: Agent and environment
(Amiri et al., 2018)

1 Reinforcement Learning basic knowledge

1.1 Reinforcement Learning Concepts

An agent gets some rewards from a sequence of actions starting from a given state in an environment. Reinforcement learning tries to model the complex probability distribution of rewards in relation to a very large number of state-action pairs. There are several key concepts, among which the most important are reward, action, state and environment.

Actions (a): Actions are moves of agents. At time t before we decide which move to take, we have t observations $\{x_1, x_2, ..., x_t\}$, and t-1 past actions $\{a_1, a_2, ..., a_{t-1}\}$. The action a_t is then generated from the policy network redistributes the wealth among assets.

States (s): The state at each time step is composed of the past observations and actions

$$s_t = (x_1, a_1, ..., x_{t-1}, a_{t-1}, x_t).$$

To simplify, we suppose that the states are fully observed. In our case the prices of our assets can fully reflect market condition, which coincides with the efficient market hypothesis.

Environment (E): The environment E is a Markov decision process between state space $\mathcal{S}$

and action space $\mathcal{A}$ with a initial state distribution of $\mathbb{P}(s_1)$ and transition dynamic $\mathbb{P}(s_{t+1} \mid s_t, a_{t-1})$. In other words, the environment contains information about the investment rules. It takes into account the current positions in our portfolio, market prices. In the input, the environment takes the re-balanced weights according to our investment policy then returns as output our reward and the next state.

Reward (r): The reward is a feedback of our action and state from the environment. In our problem, this represents the return of our portfolio. The future returns R_t of a sequence of actions and states is thus

$$R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i)$$

where γ is discount factor of future rewards.

Policy (π): π: $\mathcal{S} \to \mathcal{P}(\mathcal{A})$ The policy maps states to actions. It determines the strategy that the agent employs to determine the next action based on the current state. We discount rewards, or lower their estimated value, the further into the future they occur.

Q-value (Q): Q-value is the expected long-term return given the current state and current action.

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_{i>t}, s_{i>t} \sim E, a_{i>t} \sim \pi}[R_t \mid s_t, a_t]$$

It refers to the long-term return of taking action a under policy π from the current state.

More explications on reinforcement learning can be found on this website: *Reinforcement Learning Definitions*

1.2 Deep Q Network

Since the return function Q(s, a) depends on the state s and action a, applications of deep neural network can help approximate this function thus optimize the criteria for a sequence of portfolio. The goal of the agent is to choose actions that maximize the Q-value.

The deep reinforcement learning architecture treats asset allocation as a problem of continuous control of portfolio policy which attempts to maximize the delayed reward. It enables the process to learn the best actions possible in virtual environment in order to attain their goals. Thus it unifies function approximation and target optimization.

The Q-value updating formula is as below:

$$Q(s', a) \leftarrow (1-\alpha)Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a'))$$

where α is the learning rate and the next action a' is the one that maximize the next state's Q-value.

For more details, please check the following web page: *A Beginner's Guide to Deep Reinforcement Learning*

1.3 Deep Deterministic Policy Gradient

The deterministic policy gradient adapted Q-learning into continuous control space (Timothy P. Lillicrap, 2016). The target policy is not stochastic but a deterministic process, and there is a map function $\mu : \mathcal{S} \leftarrow \mathcal{A}$. By applying the Bellman equation, the Q-value can be written as

$$Q^{\mu}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma Q^{\mu} s_{t+1}, \mu_{\theta}(s_{t+1})]$$

and

$$\mu_{\theta}(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

where θ are trainable parameters of the deep network.

$$J(\mu_{\theta}) = Q^{\mu}(s, a)$$

Besides, taking derivatives of the objective function is the same as taking derivatives of the policy. DDPG presents an actor-critic, model-free algorithm. The actor is a process to tune deep learning parameters θ for the policy function and return the best policy for a specific state.

$$\pi_{\theta}(s, a) = \mathbb{P}(a \mid s, \theta)$$

And the critic is used to evaluate the policy according to the temporal error.

$$r_{t+1} + \gamma Q^{\pi_{\theta}}(s_{t+1}) - Q^{\pi_{\theta}}(s_t)$$

More explanations about policy gradient algorithm can be found in this *blog*.

2 Experiments Protocol and Problem Framing

2.1 Data Description

Our portfolio contains Bitcoin and 6 other cryptocurrencies. Bitcoin is considered as cash and basic currency in the portfolio . All prices of other cryptocurrencies are quoted in bitcoin. We can choose a start-time and an end-time to download the prices and volumns of each 30 minutes from the API of *Bitfinex*. The input data is thus a three-dimensional tensor, which represents cryptocurrency, time and financial indicator respectively.

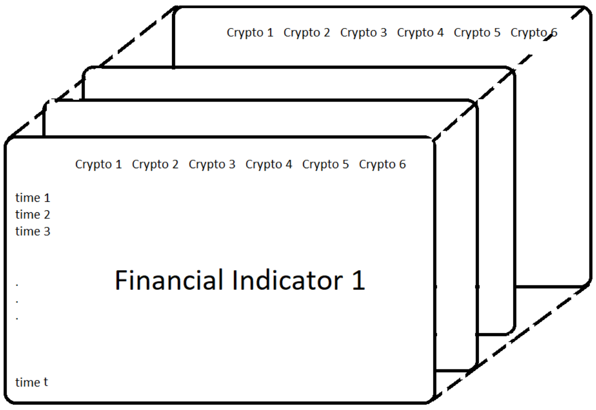

Crypto 1 Crypto 2 Crypto 3 Crypto 4 Crypto 5 Crypto 6
Crypto 1 Crypto 2 Crypto 3 Crypto 4 Crypto 5 Crypto 6
time 1
time 2
time 3
Financial Indicator 1
time t

Figure 2: Data structure

The financial indicators are the features we feed to our model. We used prices and other trading indicator often used in technical analysis of trading.

Feature	Explication
Close	Last price during specified interval
Open	First price during the period
High	Highest price level reached at
Low	Low price level reached at
Volume	Volume traded
ROC	Price rate of change.[1]
MA3	3 days' moving average
MA7	7 days' moving average
MACD	MA convergence divergence[2]

Table 1: Financial indicators

We trained our model on several periods of different length. A sliding window was defined as a set of delayed indicators plus the latest indicator. During tuning, sliding window sizes of 1 days, 2 days, and 7 days were tested. Finally we choose the window size of 48 indicators within 2 days because of the computational limits. Thus 48 is defined as length of our input matrix.

2.2 Reinforcement Learning Framework

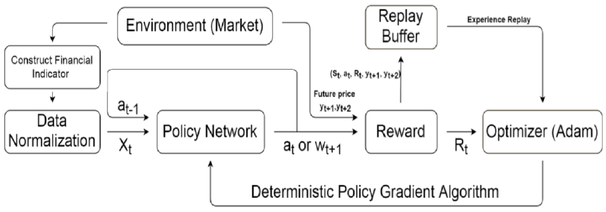


Figure 3: Problem framing[3]

Figure 3 proposed a reinforcement learning framework(Jiang et al., 2017) using deterministic policy gradient. Environment refers to the cryptocurrency market. We retrieve historical prices from the market, then construct the financial indicators and normalise prices into the form of log returns. The main objective of policy network is to maximize the reward by computing gradient of reward with respect to policy network's trainable parameter. The policy network is fed with financial indicators, previous states and the parameter gradient of reward function. Once it determined the new action to take, the reward will be calculated based on actions and future prices. The replay buffer will absorb this set of state-action-reward and help the optimize to the policy gradient.

This research model reallocated all capital each 30 minutes, the reward in next trades is not important as long as the model is able to optimize a current reward, though, a current action affects future reward due to transaction cost to a certain extent. Generally, it is necessary to have a function approximator to map an action to reward, which is what critic network does in Deterministic Deep Policy Gradient Algorithms (DDPG). For example, for portfolio management task, the relationship between reward and action can be established by a simple formula, then the model can get rid of critic network and reduce its complexity. Under the proposed model, the reward was directly calculated by dataflow graph. This concept is same as neural network except there is no trainable parameter inside the graph. In other words, it is an extension from policy network to generate a reward. Since the policy network is implemented by Keras (Tensorflow backend) and the reward function is a dataflow graph implemented by Tensorflow, the entire

[1]The Price Rate of Change Indicator (ROC) measures the percentage change in price between the current price and the price 30 minutes ago. The Formula for ROC is:

$$\text{ROC} = \left(\frac{\text{Closing Price}_p - \text{Closing Price}_{p-n}}{\text{Closing Price}_{p-n}} \right) \times 100$$

$$where:$$

$$\text{Closing Price}_p = \text{Closing price of most recent period}$$

$$\text{Closing Price}_{p-n} = \text{Closing price } n \text{ periods before}$$

most recent period.

[2]MACD, short for moving average convergence/divergence, is designed to reveal changes in the strength, direction, momentum, and duration of a trend in an asset's price.

[3]Reinforcement learning on portfolio selection check on github.

process from state to reward can also be regarded as a single Tensorflow network. As a result, the gradient can be back-propagated from reward to model's weight layer by layer.

There are several choices for the reward function, for example average logarithmic cumulative return:

$$R_{t_0} = \frac{1}{T - t_0 + 1} \sum_{t=t_0}^{T-1} \ln(r_t)$$

where r_t is the logarithm return. The goal of the model is to generate a portfolio value as high as possible, therefore, it is reasonable to utilize an average logarithmic cumulative return within a specific period as reward function. By logarithm operation, cumulative return can be expressed as summation instead of multiplication. In addition, it provides a linear relationship of each element and the gradient of reward function is easier to be computed.

Experience replay is one of the important technique to fasten the learning process of neural network. The concept of experience replay is to reuse previous states, the model is fed by these data and updated by mini-batch. With experience stored in a replay buffer, it enables the model to eliminate the temporal correlation by combining more and less recent experiences for update, and rare experience can be reused

The policy network can take different structures of deep neural networks. For example, we propose a structure of 3 convolutional layers. In the first layer, we have our data panel in input and in the last layer our inputs are 20 new features of dimension $(n_{crypto}, 1)$ and we add the latest action as another feature. By applying the filter, we get a vector of dimension $(n_{crypto}, 1)$. Next, we add the cash bias[1] into the vector and finally apply a softmax function.

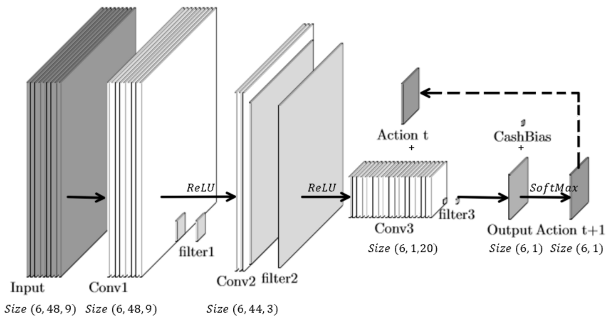


Figure 4: CNN Policy Network

2.3 Benchmark portfolio

In order to measure the performance of the deep reinforecement learning portfolio and to compare the results with other naive portfolio strategies, we created two benchmark portfolios. The first one is a equal weighted porfolio, Uniform Constant Rebalanced Portfolio (UCRP). The capital allocation to each asset is constant and we balance the portfolio weight while prices changing to make sure that the condition is respected.

Another benchmark portfolio is the Uniform Buy and Hold Portfolio (UBHP). It means that at the inception of the portfolio, capital is equally allocated to each crypto-asset and is never rebalanced.

3 Results and Parameter Calibration

3.1 Hyper parameters

Episode	200
Window size	48
Training step	820
Learning rate	2e-5 - 2e-4
Cash bias[1]	0 - 50
Sample bias[2]	1 - 1.1

Table 2: Hyper parameters

3.2 Cumulative return

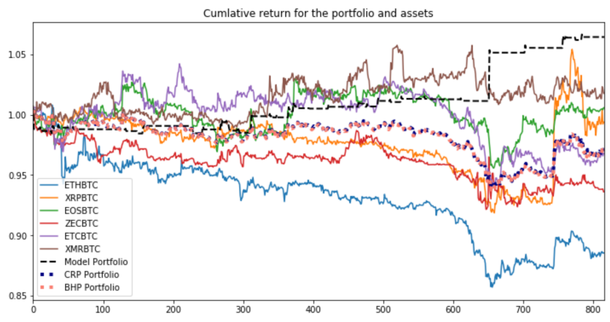


Figure 5: Cumulative return of different strategies

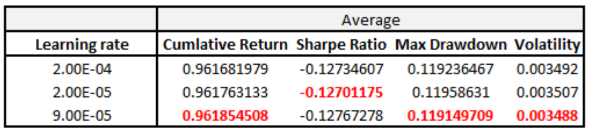
The graphic shows that our deep reinforcement portfolio has a better cumulative return than the

[1]Cash bias is a constant number associated with voting score layer. It determines the amount of cash holdings in next trading interval. The activeness of the trading strategy can be controlled by adjusting cash bias.

[2]Sample bias is usually defined in a range of 1 to 1.1, which a value above causes the model to overly bias the latest data and degrade the overall performance eventually.

benchmark portfolio. It also gained a positive return and outperformed all cryptocurrencies.

3.3 Effect of learning rate



	Average			
Learning rate	Cumlative Return	Sharpe Ratio	Max Drawdown	Volatility
2.00E-04	0.961681979	-0.12734607	0.119236467	0.003492
2.00E-05	0.961763133	-0.12701175	0.11958631	0.003507
9.00E-05	0.961854508	-0.12767278	0.119149709	0.003488

Figure 6: Effect of learning rate on DDPG

This section demonstrates how the learning rate of the model affects the trading strategy and performance. All hyper-parameters were remained the same except learning rate. The result demonstrated that the model generally achieved better performance for larger learning rate.

3.4 Several remarks about the results

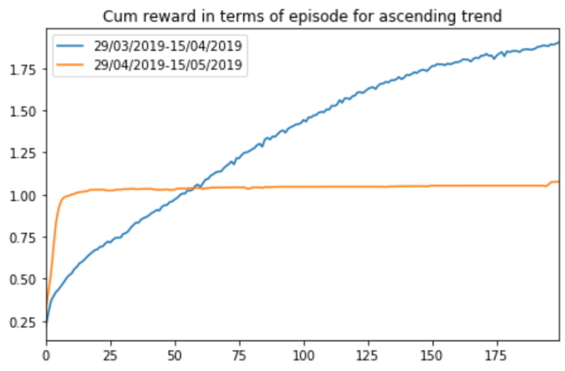


Figure 7: Cumulative rewards over-fitting and boring areas traps

Over-fitting: Our deep reinforcement learning model is empowered with large scale neural networks, carefully designed architectures, novel training algorithms and massively parallel computing devices. However, more training power comes sometimes with a potential risk of more over-fitting.

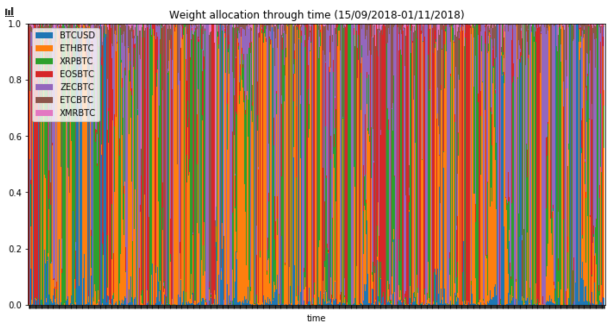


Figure 8: Over-fitted portfolio composition

Boring Areas Trap: It is possible that the agent falls into a boring areas trap. Due to lower variance differences the reward function hardly change. A lower learning rate will lower the possibility of falling in to this area however training will be very slow.

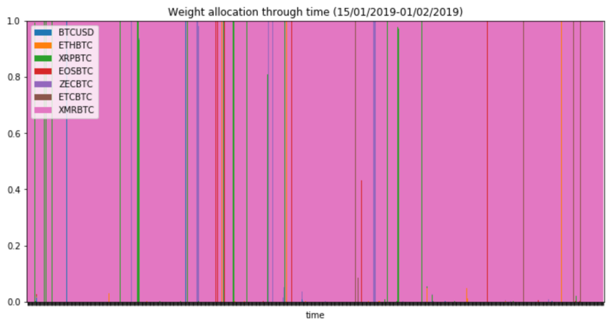


Figure 9: Trapped portfolio composition

4 Discussion

Generally speaking, the deep reinforcement learning method in asset allocation still have a long way to go before being applied into the production. It's shown in the results that the deep reinforcement learning can obtain satisfactory returns and outperform benchmarks. However, hyper-parameters are not easy to calibrate and the model is not quite stable.

We have also encountered several problem during the training phase. To train the model more efficiently with historical data, it needs a large scale of data and computation, thus GPU and distributed programming may be useful to optimise the task. What's more, another way to train the model is to create a non parametric simulator such that we can simulate a more complex but relevant environment.

The originality of financial time series compared to other reinforcement learning environment lies in the strong non-linearity and stochasticity of the dynamic systems also with the observed noisy data. How to make more in-depth theoretical analysis and further develop an appropriate framework taking these issues into consideration remain an open and interesting problem in the future.

References

Alexander Pritzel Nicolas Heess Tom Erez Yuval Tassa David Silver Daan Wierstra Timothy P. Lillicrap, Jonathan J. Hunt. 2016. Continuous control with deep reinforcement learning. *Conference paper at ICLR 2016, arXiv preprint:1509.02971*.

Zhengyao Jiang, Dixing Xu, and Jinjun Liang. 2017. A deep reinforcement learning framework for the financial portfolio management problem.

Roohollah Amiri, Hani Mehrpouyan, Lex Fridman, Ranjan Mallik, Arumugam Nallanathan, and David Matolak. 2018. A machine learning approach for power allocation in hetnets considering qos.

A Pseudo-code of DDPG Algorithm

Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
for episode = 1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for t = 1, T do
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
        Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ 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:
            $$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$
        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
```