Inspiration

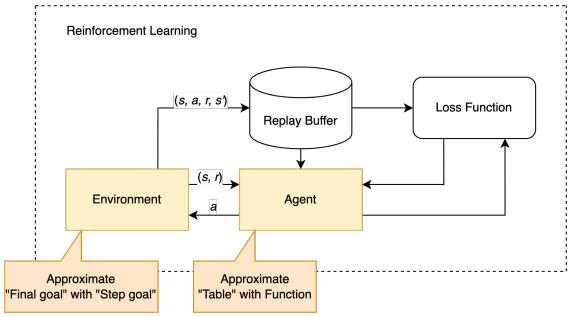
Let's say you want to invest and make money. Your goal is to maximize your return with a portfolio of stocks via some model. However, the ideal model could be very hard to realize since stock price changes are complex and often difficult to estimate. In addition, the model should be able to make decisions despite noisy inputs and perform effectively under different conditions.

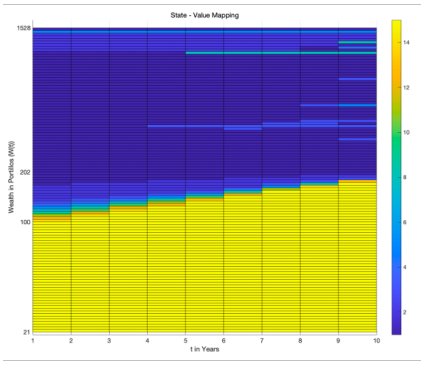
Therefore, we developed a reinforcement learning system that provides periodic investment recommendations. We provide investors with a straightforward process: inquiring about the stocks you wish to invest in and the return rate you wish for after how many years, our system can quickly and dynamically provide you with the most feasible alternative for meeting your objectives.

Breaking down the problem

It is imperative that we have a decision-making system that maximizes the Hit Ratio to the cumulative expected return during an investment cycle. We also need to be able to re-allocate our funds in the Portfolio so as to dynamically balance risk and return and manage risk according to our horizon, which means the amount of money we have and the length of time required to achieve our objectives.

To successfully solve the problem with RL, however, is not as obvious as one might expect. Because the goal is sparse and a large proportion of states can hardly be visited while being important. The key idea is to use approximation methods to guide RL agents. For reinforcement learning agents, we approximate the Q table with deep learning networks and for the environment we approximate sparse reward with stepwise reward.



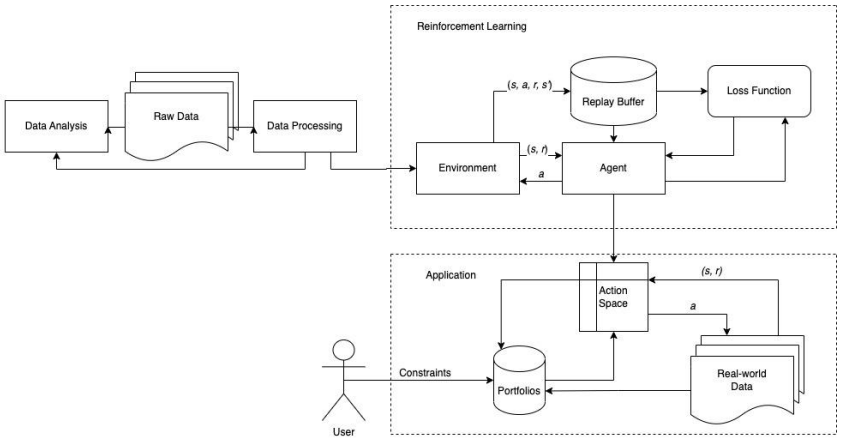


How did we implement it?

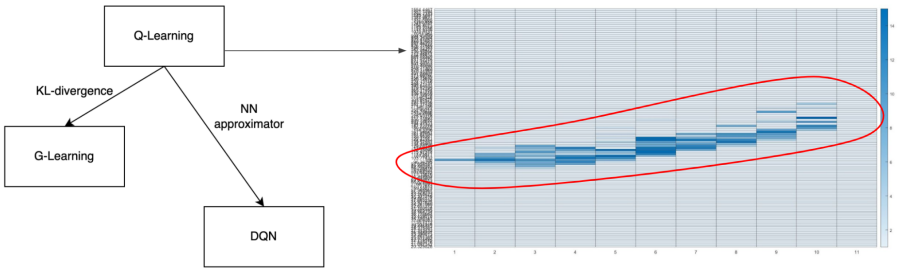
Before we implement reinforcement learning, we replicated the DP strategy of Das et al. (2020), which gives the theoretical optimal solution as our benchmark. The success probability is about 66%.

The reason we go beyond dynamic programming is that transition probability is required while it's hard to estimate in real-world scenarios. In contrast to it, reinforcement learning could learn without knowing the exact transition probability.

This is an overview of our reinforcement learning system. We first collected data with Yahoo finance and also from relevant papers. Then we used the RL Toolbox to test different agents trained in different environments. After comparing the result, we used the best agent to be the backend of our application. For our Goal-based wealth management application, with the help of Financial Toolbox, the user can set constraints for their portfolio, based on which our agent manage the portfolio to interact with the real world.

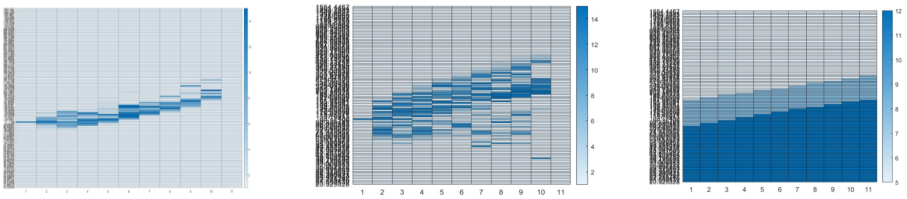


As for agents, our first discovery was that tabular agents, such as Q learning, have few opportunities to visit all states and learn policies for unvisited states. The Q learning agent only learned proper policy for limited states as shown below. For other states, however, the Q agent learnt no policy while proper actions may still fulfil the goal. We have two options for dealing with this. First, we use G learning to force the agent to explore more. We added KL-divergence against uniform distribution to force exploration. Second, we use Neural Networks to approximate the table. From DP strategy we can observe that action is a function of wealth and time, which changes continuously from state to state. Thus, we tried to approximate the function with the Neural Network which could infer the action for unvisited states.



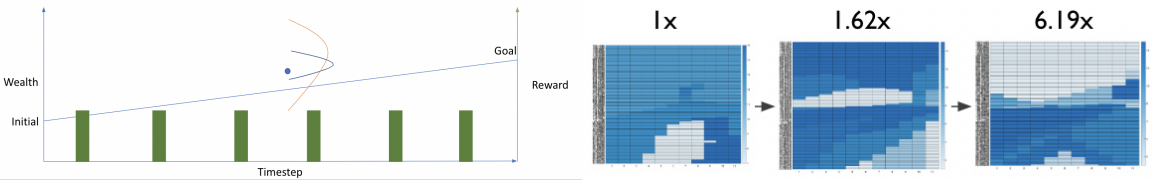
Below is the comparison of 3 different RL algorithm policies and success probabilities. DQN has the highest success rate because it has the ability to infer states that have not been visited.

	Agent		
Metrics	**Q-Learning**	**G-Learning**	**DQN**
Success probability	40.50%	33%	**48%**



Q-Learning G-Learning DQN

As for environments, we approximate the sparse reward with step reward. Sparse reward fails to guide RL agents with large action space and environment with long episodes. Hence, it is important to approximate the sparse reward with a more informative reward.



We developed several environments based on intuitions and research papers. The *Line Reward* and *Scale Reward* environments are original environments, where the *Line Reward* gives positive reward if above a predefined curve at each step. The above(left) figure is an example with a straight line connecting initial wealth and goal. By means of a curve, we can guide the agent's behaviour at different times. The *Scale Reward* gives a scaled reward if the goal is fulfilled at each step, and the latter the larger. The three action grids above are DQN policies trained with different scaled rewards that the last reward is 1, 1.6 and 6 times to the first reward, respectively. The agent becomes defensive as we expected and identical to the DP result. The multi factors reward is an implementation of a relevant paper that aggregates max drawdown, sharpe ratio etc.

Results

We present our DQN experiment results on the three environments. We have observed a significant improvement compared with the sparse setting. And the best result is DQN with 16 hidden units on line reward environment, which is 61%. Only -0.07 away from the theoretically highest success rate.

		Train Environment		
DQN		Sparse	Line	Multifactor
Hidden units		**Success probability on Sparse Env.**		
	8	38.30%	49.60%	48.20%
	16	48.00%	**61.00%**	47.40%
	64	57.00%	53.00%	50.40%

Key Takeaways

This project was an entirely new experience for all of us. We had not previously been introduced to MATLAB, so we were initially concerned about how far we could go with it. However, we quickly discovered that this apprehension was superfluous because MATLAB is a very user-friendly programming tool that not only provides free courses on all aspects of programming, but also has many full-featured and powerful toolboxes. For example, the reinforcement learning toolbox that we use extensively here includes all types of agents, so we only need to know its name to use it. Then we can directly invoke the desired model in MATLAB and freely and easily set various variables. This directly simplified our project and saved us a significant amount of time. At the same time, we don't have to be afraid of using unused functions when writing code. The comprehensive Help Center of MATLAB not only contains a detailed description of each function, but also a large number of examples and even other people's questions similar to the ones we encountered. We believe that our experience demonstrates that MATLAB is an excellent tool for anyone interested in using programming to solve real-world problems.

Below are a few of our ideas for future improvements and features:

- Analyze our users, design our user interface, and gather feedback.
- Create a mobile app and a website to make it easier to use for investors.
- Add the ability to track each investor's investment intention to provide personalized investment suggestions.

We would like to thank our two mentors, Alejandra Pena-Ordieres and Valerio Sperandeo, again for patiently leading us through the process of getting started, providing us with information on our topics, helping us solve problems, and guiding us on the right path.

https://github.com/bwfbowen/RL_GBWM