TSINGHUA PBCSF 清华五道口 | IFR 金融科技研究院

The Rise of Deep Reinforcement Learning and Applications in Stock Market Research

Contents

The Rise of Deep Reinforcement Learning and Applications in Stock Market Research

1. Introduction

Deep reinforcement learning has become a flourishing subfield of machine learning in the past decade. Two remarkable and well-known successful cases of using deep reinforcement learning to solve sophisticated games, Atari 2600 and AlphaGo, substantially catalyze the research interest in this direction globally. An important goal of machine learning research is to create intelligent systems that are able to automate complex decision making and achieve human-level control. Deep reinforcement learning holds the promise to be an important component of this system.

Reinforcement learning is an experience-driven autonomous learning method. The essence of it is learning from interaction with the environment. Broadly, reinforcement learning has its early roots in the behaviorist psychology (trial-and-error learning) and optimal control (its solutions using value functions and dynamic programming). These two subfields provide the foundations for the modern reinforcement learning.

While reinforcement learning has made important progresses across various domains, a bottleneck is lack of scalability. Specifically, to successfully implement reinforcement learning for solving real-world tasks, the learning agents confront a challenge of deriving efficient representation of the environment (Mnih et al., 2015). In other words, reinforcement learning per se lacks scalability and is inherently limited to low-dimensional problems, a.k.a. "the curse of dimensionality".

The advances in deep learning provide a powerful tool to tackle this bottleneck confronted by reinforcement learning. Deep learning has powerful function approximation and automatic feature learning properties, which enables the reinforcement learning agent to effectively handle the unstructured environment. Incorporating deep learning into the reinforcement learning framework gives rise to the so-called deep reinforcement learning. Combining deep representation learning with the reinforcement learning framework makes it possible to learn complex policies in high dimensional environments and solve such complex high-dimensional problems end-to-end.

In this study, we aim to provide a brief overview of the core ideas and algorithms in reinforcement learning and deep reinforcement learning,

and summarize recent applications of deep reinforcement learning in stock market research.

The rest of this study is organized as follows. Section 2 introduces the background, core concepts, mathematical set-up, and key algorithms reinforcement learning. Based on this, section 3 discusses the rise of deep reinforcement learning and surveys important and influential deep reinforcement learning algorithms. In section 4, we briefly summarize recent applications of deep reinforcement learning algorithms on stock market research. Then, we highlight primary challenges confronting deep reinforcement learning and discuss promising research directions going forward. Then, we summarize in section 5.

2. Reinforcement Learning

The fundamental idea underlying reinforcement learning is learning by interacting with the environment. This branch of machine learning focuses on much more on goal-directed learning from interaction than other forms of machine learning (Sutton and Barto, 2018). In this section, we firstly discuss the core ideas and essential features of reinforcement learning informally. Then we go on to formalize the problem and set up the mathematical model of it based on the Markov Decision Processes.

2.1 Key Concepts and Ideas of Reinforcement Learning

Informally, reinforcement learning is "a computational approach to understanding and automating goal-oriented learning and decision making"(Sutton and Barto, 2018). It learns to map situations to actions in order to maximize a reward signal. In doing so, the RL agent is not told what actions to take under a given situation, but has to figure out the most-rewarding actions by trying them out. The two most distinguishing features of reinforcement learning are trial-and-error search and delayed reward (Sutton and Barto, 2018).

The primary elements in a reinforcement learning system include: an agent, an environment, a policy, a reward signal, and optionally a value function and a model of the environment. At each time step, the agent (partially) observes the state of the environment and then take an action following a policy, which would change the state of the environment and send a reward signal to the agent. The goal of the agent is to maximize the total rewards. In particular, the policy plays a central role in the reinforcement learning system. It specifies a rule that tells the agent what action to take in a given state. We will formalize policies and the environment dynamic in this section later.

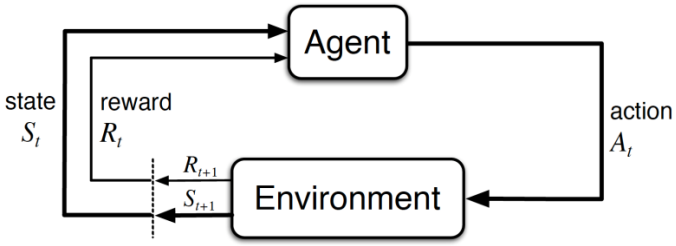

Figure 1: Agent learn by interacting with the environment (Sutton and Barto, 2018)

A fundamental dilemma in reinforcement learning is the trade-off between exploitation and exploration, which is not present in the supervised learning and unsupervised learning. This challenge originates from the trial-and-error search feature. On the one hand, the learner should take actions that is known to yield high rewards by exploiting its past experiences, on the other, the agent needs to explore what actions lead to higher reward in order to maximize the total rewards over the life time. In fact, one active research area is designing the exploration strategies that can achieve a better balance between exploitation and exploration.

Another feature of the reinforcement learning is goal-oriented, which means the learning agent explicitly considers the whole problem while interacting with an uncertain environment (Sutton and Barto, 2018).

2.2 Mathematical Framework of Reinforcement Learning

Markov Decision Processes (MDPs) is the standard mathematical framework to formalize the single-agent sequential decision-making problems. For the denotation, we follow OpenAI Spinning Up (OpenAI, 2018). The MDPs is a 5-element tuple <S, A, T, R, r>:

- A state space S (discrete or continuous).
- An action space A (discrete or continuous).
- Transition dynamics T (deterministic or stochastic).
- An immediate reward function R.
- A discount factor $\gamma \in (0, 1)$.

The state can be viewed as all information about the uncertain environment that is available to the agent. The transition dynamics satisfy the Markov property, which means that the transitions only depend on the most recent state and action and not the previous history. In other words, only the current state and action affects the next state. The Markov property depends on the assumption that the environment state is fully observable to the agent, which may not be true in reality. A generalization of MDP is the Partially Observable Markov Decision Processes (POMDPs), which assumes the only part of the state is accessible to the agent.

To date, much of the reinforcement learning theory literature focuses on finite MDPs that has finite state, action, and reward sets.

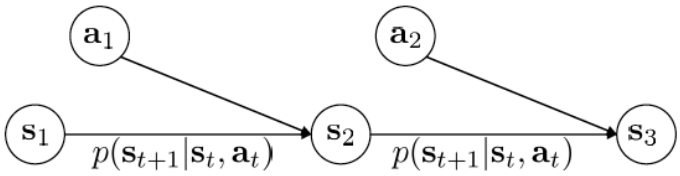

Figure 2: The Markov Decision Processes (Levine, 2019)

In this mathematical setup, the policy π is a function that maps the state space to a distribution over the action space, which can be either deterministic or stochastic. Starting from an initial state and following a particular policy, the agent will generate a sequence of states and actions, which is called a trajectory/rollout/episode, denoted by τ. Typically, the first state in the trajectory is randomly sampled from a start-state distribution.

The reward signal is a function of the current state and action pair, and optionally the next state. The central optimization problem in the reinforcement learning system is therefore to select a policy to maximize the expected accumulative rewards (instead of merely the immediate reward) over the entire trajectory. From this sense, designing the reward

function is of critical importance, and this active line of research is termed as reward shaping.

Value functions

The value of a state refers to the expected return the agent would receive if the agent starts from that state and follow a particular policy thereafter. Intuitively, it indicates how good a state is for the agent in terms of expected return. While the reward signal specifies what is good at a time step, the value function characterizes what is good over the long run. The value function of a state s under a policy π is formally defined as

$$V^{\pi}(s) = \mathop{\mathrm{E}}_{\tau\sim\pi}\left[R(\tau)\,|s_0 = s\right].$$

A closely related term is the action-value function $Q^{\pi}(s,a)$, which refers to the expected return if the agent starting from state s and taking an action a and following a policy π henceforth, denoted as

$$Q^{\pi}(s,a) = \mathop{\mathrm{E}}_{\tau\sim\pi}\left[R(\tau)\,|s_0 = s, a_0 = a\right].$$

There are different ways to estimate the value function and action-value function. For instance, we can use the Monte Carlo methods or a parameterized function approximator to estimate the value function as well as the action-value function from experience. The optimal policy in a state will select the action that maximizes the expected return in that

state. The corresponding optimal value functions are defined formally as follows:

$$V^*(s) = \max_{\pi} \mathop{\mathrm{E}}_{\tau\sim\pi} [R(\tau)\,|s_0 = s]$$

$$Q^*(s,a) = \max_{\pi} \mathop{\mathrm{E}}_{\tau\sim\pi} [R(\tau)\,|s_0 = s, a_0 = a].$$

One view in the reinforcement learning community is that value functions are crucial for efficient search in the policy space (Sutton and Barto, 2018), while another may prefer to search in the space of polices directly without resorting to the value functions.

Bellman Equations

The value functions satisfy the self-consistency condition described by the Bellman Equation, which expresses a recursive relationship between the value of a state and the values of successor states. More specifically, it specifies the value of a state equals the expected reward the agent can get from that state plus the value of the expected next state. The Bellman equations for both the value functions and optimal value functions are

$$V^{\pi}(s) = \mathop{\mathrm{E}}_{\substack{a\sim\pi \\ s'\sim P}} [r(s,a) + \gamma V^{\pi}(s')]\,,$$

$$Q^{\pi}(s,a) = \mathop{\mathrm{E}}_{s'\sim P} \left[r(s,a) + \gamma \mathop{\mathrm{E}}_{a'\sim\pi} [Q^{\pi}(s',a')] \right],$$

$$V^*(s) = \max_{a} \mathop{\mathrm{E}}_{s'\sim P} [r(s,a) + \gamma V^*(s')]\,,$$

$$Q^*(s,a) = \mathop{\mathrm{E}}_{s'\sim P} \left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \right].$$

As we will see, the Bellman equations provide a basis for many

approaches to approximate and learn the value functions. For finite MDPs, there is a unique solution for the Bellman equations for the optimal value functions. However, for real-world problems, we generally do not solve the Bellman equations exactly, but settle for approximate solutions. Once we have the optimal value functions, it is relatively easy to derive the optimal policy from them.

Advantage functions

The idea of the advantage functions is akin to removing a baseline from a signal. It provides information about how good an action compared to the average. We formalize this idea by defining the advantage function as

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s).$$

The advantage function is widely used in policy-based reinforcement learning algorithms as we will discuss later.

In general, there are three classes of methods for solving the finite MDPs problems: dynamic programming, Monte Carlo methods, and temporal-difference learning. Each class has its strengths and weaknesses. Fortunately, we can combine them to create solutions that can outperform any one class alone.

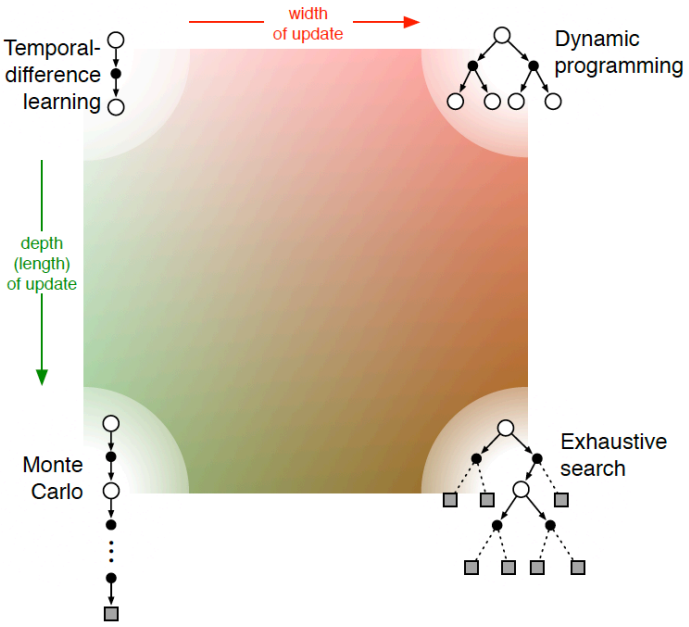

Figure 3: Two dimensions of the space of reinforcement learning methods: the depth and width of the updates (Sutton and Barto, 2018)

2.3 Reinforcement learning algorithms

The mainstream reinforcement learning algorithms can be roughly divided into three classes: value-based methods, policy-based methods, and hybrid methods that combined these two approaches. At the same time, it is worthy to note that there are other important branching points in the reinforcement learning algorithms, each characterizes the algorithms from a different and important aspect. While the we organize various algorithms following the first categorization criteria, we will investigate the algorithms from a variety of branching points in order to gain a comprehensive and deep understanding of them. To this end, we

firstly summarize these critical branching points before diving into the ocean of reinforcement learning algorithms.

Model-based vs. Model-free

The distinguishing point is whether a model of the environment is available to the agent.

For model-based reinforcement learning, the agent will learn a transition model to simulate the environment, and therefore the agent can learn without interacting with the environment directly. This is especially useful when interacting with the environment is very costly in real-world tasks. Then the learned environment models can be used for planning. By contrast, for model-free RL, such knowledge of the environment dynamics is not available. The agent learns though trial-and-error explicitly. Of course, there is no free lunch. The upside of model-based methods is higher sample efficiency, but the downside is that they suffer from model bias.

On-policy vs. Off-policy

On-policy reinforcement learning algorithms conduct updates using trajectories generated by the policy, whereas off-policy approaches can make use of trajectories generated by other policies. On-policy methods

have lower sample efficiency than their off-policy counterparts as the former cannot use old data or data from different polices. However, the reliance on on-policy data makes the on-policy algorithms more stable than off-policy methods.

Finite-horizon vs. Infinite-horizon

For Infinite-horizon reinforcement learning, the number of time steps in the MDP goes to infinite. In this case, we need a discount factor when calculating the expected rewards. Correspondingly, there are finite number of time steps in the finite Finite-horizon reinforcement learning. These two kinds of methods are appropriate to model different real-world tasks in practice.

Finally, also note that while there are various reinforcement learning algorithms, they can be framed into a three-step loop as shown below. In the rest of this section, we will focus on investigating these steps when analyzing important RL algorithms in the rest of this section.

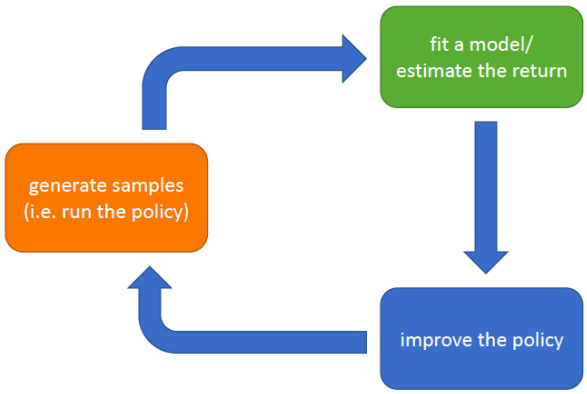

Figure 4: The anatomy of reinforcement learning algorithms (Levine, 2019)

2.3.1 Value-based methods

Now we are ready to study the real meat of the study and we begin with the value-based methods. This class are based on the value function or action value function (a.k.a. "Q-function") that we formulated in the previous part. The agent takes an action based on the evaluation of the value function. As noted earlier, once we have the optimal value function, we can derive the optimal policy from it. If we have the optimal value function $V^*(s)$, we can derive the optimal policy by conducting a one-step-ahead search and finding the action(s) that is greed with respect to the value function. Alternatively, if the optimal action value function $Q^*(s,a)$ is available, we find the optimal policy simply by choosing the action(s) that maximize $Q^*(s,a)$. From this perspective, efficiently estimating the value function plays a central role for the value-based methods.

Sarsa

We begin our study of value-based methods with the Sarsa algorithm. It is uses on-policy temporal-difference learning method, combining the ideas from Monte Carlo methods and dynamic programming, to learn the action-value function (Sutton and Barto, 2018). At each time step, Sarsa uses the episodes generated by the current policy to update the action-value function as below

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big].$$

Sarsa continuously update the action value function in this way and concurrently update the policy greedily with respect to the updated value function.

Q-learning

Q-learning is an off-policy temporal difference learning algorithm. While Q-learning still uses the current policy to choose state and action pairs to visit, the learned Q function would directly approximate the optimal action value function independent of the policy being followed (Sutton and Barto, 2018). Formally, the update of the Q function is defined as

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

Learning this way enables earlier convergence compared to the on-policy

value-based Sarsa.

2.3.2 Policy-based methods

Policy Gradient

Unlike the value-based methods, the policy-based methods learn the policy directly without resorting to the value functions. (In DRL, the policy is parameterized as a deep neural network as we will see in section 3.) In contrast to the value-based methods, the policy search approaches optimize the policy performance directly without consulting the value function estimates, which give rise to several advantages. For example, learning a parameterized policy can handle the continuous action space naturally and also is a good way of inject prior knowledge about the policy. However, there is no dominating algorithms so far and it is important to understand the tradeoff between different classes of reinforcement learning methods. In general, the policy search approach trades off sample efficiency for higher stability.

The core idea underlying the policy gradient is using gradient ascent to estimate the parameters of the policy by maximizing the expected rewards. Formally, let π_θ be a stochastic policy parameterized by θ, and $J(\pi_\theta)$ the expected return under this policy. Then the policy

gradient is $\nabla_\theta(J(\pi_\theta))$. Given a learning rate of α, we can use update the policy using the standard gradient ascent (or other gradient ascent algorithms)

$$\theta_{k+1} = \theta_k + \alpha \, \nabla_\theta J(\pi_\theta)|_{\theta_k}.$$

The general form of the policy gradient is

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\Phi_t\right],$$

with Φ_t be the sum of all rewards over the trajectory, the "reward-to-go", or other forms. It is called the policy gradient theorem. This crucial result reduces the performance gradient to an expectation, which has as a simple form and can be estimated by samples.

Vanilla Policy Gradient

We start with the Vanilla Policy Gradient (VPG), which is arguably the simplest algorithm among the policy search class (Sutton et al, 2000). For VPG, we formulate the policy gradient using the advantage function introduced earlier. Mathematically, in the general form of the policy gradient, we replace Φ_t with the advantage function for the current policy $A^{\pi\theta}(s_t, a_t)$ (Schulman et al, 2016). VPG is on-policy algorithm as it conducts exploration by sampling actions based on the most recent policy. This approach is susceptible to local optima.

Trust Region Policy Optimization

A problem with the VPG is that a large step size can be dangerous as it may collapse the policy performance. The Trust Region Policy Optimization (TRPO) addresses this challenge by limiting each policy gradient update to prevent the updated policy deviates too much away from the previous one (Schulman et al, 2015). Such deviation is measured by the KL divergence between the current policy and the updated policy. In this way, the on-policy TRPO is able to improve the policy monotonically with non-trivial step sizes.

Proximal Policy Optimization (PPO)

The design of PPO shares the same motivation with the TRPO. But PPO manages to attain comparable stability and reliability with TRPO in terms of empirical performance, but only uses first-order optimization (instead of second-order optimization as in TRPO) (Schlman et al., 2017). Hence, PPO is easier to understand and simpler to implement than TRPO. To do so, PPO uses a surrogate objective with clipped probability ratios to form a lower bound of the policy performance. And then optimizing the policy by alternating between sampling data from the policy and performing a number of epochs of optimization on the sampled data.

2.3.3 Hybrid methods

Actor-Critic Methods

The hybrid methods combine the valued-based and policy-based methods, which learn approximations to both the policy and value functions. For the classic actor-critic method, the learned policy plays the role of choosing action and hence is the actor, and the estimated value function would criticize the action made by the actor and therefore is the critic. In this way, we the actor-critic style methods are able to effectively reduce variance and accelerate learning.

Deterministic Policy Gradient

The Deterministic Policy Gradient (DPG) is a model-free off-policy actor-critic algorithm (Silver et al., 2014). Theoretically, the deterministic policy gradient is shown to be the expected gradient of the action-value function, and therefore it can be estimated more efficiently than the stochastic policy gradient. In addition, the deterministic policy is in fact a limiting case of the stochastic policy gradient with the policy variance tends to zero, which implies that the policy gradient machinery also applies to the deterministic policy gradients (Silver et al., 2014).

The DPG learns a Q-function and a policy concurrently, which are

updated to improve each other in an actor-critic style. Specifically, the actor updates the policy in the direction of the off-policy deterministic policy gradient (which is the gradient of the action-value function), and the critic estimates the action-value function using Q-learning. The DPG enjoys higher sample efficiency as they can reuse old data. As we will see in the next section, Deep Deterministic Policy Gradient extends the DPG to high-dimensional state-action space.

3. Deep Reinforcement Learning

3.1 The Rise of Deep Reinforcement Learning

The integration of reinforcement learning and deep learning is arguably one of the most exciting and fruitful directions in the machine learning field in recent years, which give rise to the flourishing deep reinforcement learning.

In this subfield, we use deep neural networks to approximate a stochastic policy. Common stochastic policies in include categorical policies and diagonal Gaussian policies, which are typically used for discrete action spaces and continuous action spaces separately. For training stochastic polices, we need to first sample action from the policy and then compute the log likelihoods of the sampled actions.

3.2 Key Deep Reinforcement Learning Algorithms

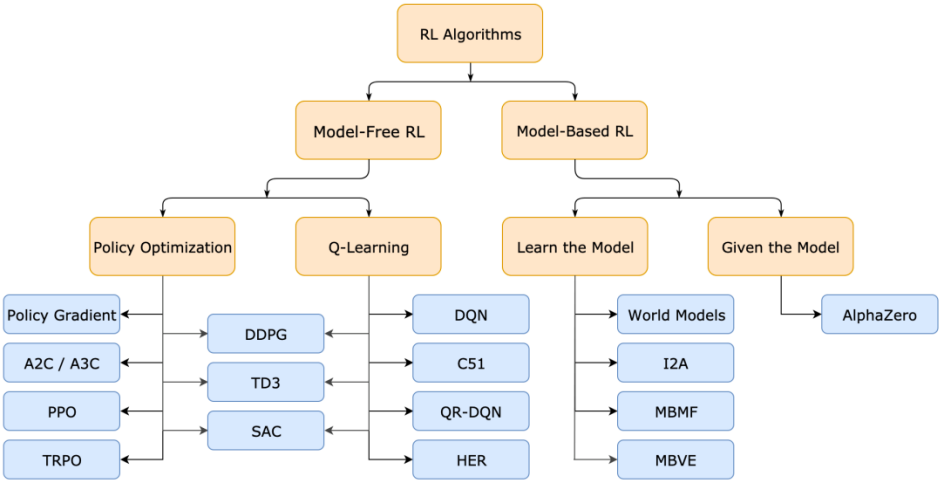

Figure 5: A non-exhaustive taxonomy of the RL algorithms (OpenAI, 2018)

3.2.1 Value-based Methods

DQN

The success and influence of the classic Deep Q-Network (DQN) (Mnih et al., 2013; 2015) has secured its position in the machine learning history. In fact, it substantially launched the deep reinforcement learning field. DQN extends the Q-learning algorithm by using deep neural networks as the Q-function approximator, which enables it to learn from high-dimensional state spaces directly. In practice, two practical techniques significantly improve the DQN performance: (1) experience replay (a.k.a. replay buffer) (break the correlation between successive experience samples); (2) target networks (provide consistent targets and reduce

variance). These two technical innovations enable the DQN to learn value functions using deep neural networks in a stable and robust way.

However, while the DQN can handle discrete and low-dimensional action spaces well, there is no straightforward way to extend it to continuous action spaces which are common for many real-world cases such as continuous control. The design of the DDPG is partly motivated to address this constraint of DQN.

3.2.2 Hybrid Methods

Deep Deterministic Policy Gradient

The Deep Deterministic Policy Gradient (DDPG) extends the DPG to high-dimensional state-action space (Lillicrap et al., 2016). To do so, it combines the insights from the DQN and DPG outlined in Section 2. The DDPG is a model-free, off-policy, actor-critic algorithm that uses deep function approximators to learn policies in high-dimensional and continuous action spaces. In particular, the DDPG adopts the same technical innovations replay buffer and target networks as in DQN.

Since it is not straightforward to extend DQN to continuous action space, this motivates the DDPG to use the actor-critic method as in the DPG.

The critic, an action-value function approximator, is learned by exploiting the Bellman equation using off-policy data. This is typically done by minimizing a mean-squared Bellman error (MSBE) function, which indicates how closely the current action-value approximator comes to satisfying the Bellman equation. As it is challenging to compute the maximum over actions in the target, the DDPG uses the target policy network instead. The learning for the actor side is relatively easy. A parameterized actor function is updated following the policy gradient, using the expected gradient of the action-value function approximator we learned. The pseudocodes can be found in Appendix 1.

However, a limitation of DDPG is that it typically requires a lot of episodes to train. In addition, its brittleness and hyperparameter sensitivity makes it challenging to implement in real-world tasks.

Twin Delayed DDPG

A common problem of DDPG is that the errors of the Q-function approximator always lead to overestimation of the true Q-values. The Twin Delayed DDPG (TD3) improves the DDPG algorithm by introducing innovative mechanisms to address the problems in DDPG (Fujimoto et al., 2018). First of all, TD3 uses clipped double-Q learning

to replace the critic in the actor-critic setting. A second technique is delayed policy updates. In TD3, the policy network is updated at a lower frequency than the value network. Finally, target policy smoothing regularization is introduced to reduce the variance of the target. In practice, it is done by adding random noise to the target policy. These techniques together significantly improve the performance of TD3 compared to DDPG.

Soft Actor Critic

Beside TD3, another descent of DDPG is Soft Actor Critic (SAC) (Haarnoja et al., 2018). In contrast to learning a deterministic policy as in DDPG, SAC optimizes a stochastic policy in an off-policy way. It can be implemented in discrete or continuous action spaces.

A key feature of SAC is entropy regularization. The policy is trained to maximize the sum of the expected future return and entropy, which is closely related to the fundamental exploration and exploration trade-off of reinforcement learning. We briefly introduce the entropy-regularized reinforcement learning. The core difference is that we alter the reinforcement learning objective by augmenting the maximum reward with a maximum entropy term, which improves the exploration and

robustness. The augmented objective is denoted as

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot|\mathbf{s}_t)) \right].$$

In particular, α is the entropy regularization coefficient that can be either fixed or varying over the course of training. This hyperparameter controls the tradeoff between exploration and exploitation, which should be tuned according to the environment. It is easy to see that higher entropy encourages more exploration. Based on this new objective, the value functions in the maximum entropy reinforcement learning should be altered accordingly.

For implementation, SAC also adopts the clipped double-Q learning technique to improve the performance similar to TD3. Empirical results show that SAC outperforms prior on-policy and off-policy reinforcement learning algorithms in terms of sample efficiency and stability.

Asynchronous Advantage Actor Critic

Although the classic DQN addresses the fundamental stability problem in deep reinforcement learning, it is computationally expensive. In particular, while the experience replay buffer contributed to reduce non-stationarity and decorrelates updates, the drawbacks are higher memory and computation cost and updates using data generated from older

policies.

An alternative line of research exploits parallel computation to speed up learning and improve on stability, which motivates the design of Asynchronous Advantage Actor Critic (A3C) (Mnih et al., 2016) algorithm. The key underlying idea is to asynchronously execute multiple agents in parallel on multiple instances of environment, which can improve the robustness of both on-policy and off-policy deep reinforcement learning performance. In particular, A3C combines advantage updates with actor-critic framework and uses parallel actor learners to update a shared model to stabilize the learning process. In this way, A3C outperforms prior deep reinforcement learning algorithms and becomes of the most popular starting point for subsequent works in this field. A closely related algorithm is Advantage Actor Critic (A2C), in which only one agent is used (Wang et al., 2017).

3.3 Current Challenges and Research Directions

We end our survey of the aforementioned mainstream deep reinforcement learning algorithms with a brief discussion of current challenges and some of the promising further research directions (this is by no means exhaustive).

Meta Learning

The current machine learning research focuses on end-to-end learning, which means to learn to solve a problem absolutely from scratch without accessing any domain-specific knowledge. However, this type of learning usually requires a lot of data to train and does not generalize well to different but similar tasks. On the contrary, human beings can learn to solve a problem from only a few examples and find it easy to solve a similar task taking advantage of previously acquired skills. Inspired by the way human beings learn, meta learning, a.k.a. learning to learn, aims to not only speed up learning but also learn a range of skills concurrently. For more details, we direct the readers to a comprehensive survey of meta learning (Finn, 2018).

Imitation Learning and Inverse Reinforcement Learning

Imitation learning refers to learning from demonstrations, which is a straightforward way to acquiring new skills. It is traditionally known as behavior cloning. A downside of this approach is the difficulty to adapt to new situations. There are ongoing studies on tackling this challenge.

Inverse reinforcement learning (IRL) aims to estimate an unknown

reward function from observed trajectories. IRL can be combined with reinforcement learning to better learn from demonstrations. Theoretically, it is shown that IRL can be reduced to measure matching (Ho and Ermon, 51). This motivates the design of generative adversarial imitation learning (GAIL), which can directly extract a policy from data.

Multi-agent Reinforcement Learning

For many real-world scenarios, there are multiple agents acting in an environment instead of a single one. In such case, the MDPs framework is no longer adequate to model the problem. Instead, we can resort to the stochastic game to formulate a set-up, which allows us to consider new questions, such as communication among agents and how they should cooperate. A current hot topic is on about integrating mutli-agent deep reinforcement learninging with game theory (Heinrich and Silver, 2016).

4. Recent Applications of DRL in Finance

In this section, we investigate application of deep reinforcement learning methods in financial research. Due to space constraint, we choose to focus on recent applications of advanced deep reinforcement learning algorithms in stock market research.

Xiong et al. (2018) explore to apply deep reinforcement learning to train

an adaptive stock trading strategy to maximize return. Specifically, the authors choose to use the DDPG algorithm to find the optimal trading strategy in complex and dynamic stock market and demonstrate that DDPG outperforms traditional min-variance portfolio allocation method and the Dow Jones Industrial Average in terms of return. As a closely-related study, Li et al. (2019) also investigate the potential of utilizing deep reinforcement learning for stock portfolio allocation. Similar to Xiong et al. (2018), they propose an adaptive DDPG framework to incorporate optimistic or pessimistic deep reinforcement learning that is reflected in the prediction error.

Another work in this line of research is automating swing trading using deep reinforcement learning (Azhikodan et al., 2019). While they the DDPG algorithm as well, the authors highlight the importance of incorporating stock value trend prediction into the reinforcement learning system. To do so, they implement a sentiment analysis model using a RCNN to predict stock trend from financial news. In addition to these studies, Li et al. (2019) investigated how to use deep reinforcement learning models to construct stock market investment strategy and more broadly the impact of artificial intelligence on financial investment.

As we can see, adopting deep reinforcement learning to solve sophisticated financial problems is a burgeoning area in recent years. It is promising to explore more applications in the future.

5. Summary

The combination of the reinforcement learning framework and high-capacity function approximators with deep neural networks makes it possible for the agent to learn complex policies in high dimensional environments and solve such complex high-dimensional problems end-to-end. Looking forward, deep reinforcement learning holds the promise to be an important component of intelligent systems that are able to automate complex decision making and achieve human-level control.

In particular, a promising application of deep reinforcement learning is to automate sequential decision making in finance. To this end, we hope this study will motivate further research on applying deep reinforcement learning algorithms on a variety of financial problems.

Appendix

1. Pseudocode of Algorithms

(1)Approximate policy iteration algorithm guaranteeing non-decreasing expected return (Schulman et al, 2015)

Initialize π_0.
for $i = 0, 1, 2, \ldots$ until convergence **do**
Compute all advantage values $A_{\pi_i}(s, a)$.
Solve the constrained optimization problem

$$\pi_{i+1} = \arg\max_{\pi} \left[L_{\pi_i}(\pi) - \left(\frac{2\epsilon'\gamma}{(1-\gamma)^2} \right) D_{\mathrm{KL}}^{\max}(\pi_i, \pi) \right]$$

where $\epsilon' = \max_s \max_a |A_\pi(s, a)|$

and $L_{\pi_i}(\pi) = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s, a)$

end for

(2)PPO, Actor-Critic Style

---

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

---

(3)DDPG algorithm (Lillicrap et al, 2016)

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 μ' 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

---

(4)TD3 algorithm (Fujimoto et al., 2018)

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ
with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta_1' \leftarrow \theta_1$, $\theta_2' \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
  $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta_i' \leftarrow \tau\theta_i + (1 - \tau)\theta_i'$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

References

[1] Mnih V, Kavukcuoglu K, Silver D, Rusu A, et al. Human-level control through deep learning. Nature, Vol. 508, 2015.

[2] Sutton R, Barto A. Reinforcement learning: an introduction. The MIT Press. Second edition. 2018.

[3] Levine S. Deep reinforcement learning, decision making, and control. Department of Electrical Engineering and Computer Science, UC Berkeley. 2019.

[4] OpenAI. Spinning up in deep reinforcement learning. OpenAI, 2018.

[5] Sutton R, McAllester D, Singh S, Mansour Y. Policy gradient methods for reinforcement learning with function approximation. In Proceedings of the Annual Conference on Advances in Neural Information Processing Systems (NIPS 2000), 2000.

[6] Schulman J. Optimizing expectations: from deep reinforcement learning to stochastic computation graphs. Ph.D dissertation. UC Berkeley. 2016.

[7] Schulman J, Levine S, Moritz P, Jordan M, Abbeel P. Trust region policy optimization. In Proceedings of the International Conference on Machine Learning (ICML 2015), 2015.

[8] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithm. OpenAI. 2017.

[9] Silver D, Lever G, Heess N, Degris t, Wierstra D, Riedmiller M. Deterministic policy gradient algorithms. In Proceedings of the International Conference on Machine Learning (ICML 2014), 2014.

[10] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M. Playing Atari with deep reinforcement learning. 2013.

[11] Lillicrap T, Hunt J, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D. Continuous control with deep reinforcement learning. In Proceedings of the International Conference on Learning Representations (ICLR 2016), 2016.

[12] Fujimoto S, van Hoof H, Meger D. Addressing function approximation error in actor-critic methods. In Proceedings of the International Conference on Machine Learning (ICML 2018), 2018.

[13] Haarnoja T, Zhou A, Abbeel P, Levine S. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. Berkeley Artificial Intelligence Research, UC Berkeley. 2018.

[14] Mnih V, Badia A, Mirza M, Graves A, Harley T, Lillicrap T, Silver D, Kavukcuoglu K. Asynchronous methods for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning (ICML 2016), 2016.

[15] Wang J, Kurth-Nelson Z, Tirumala D, Soyer H, Leibo J, Munos R, Blundell C, Kumaran D, Botvinick M. Learning to reinforcement learn. In CogSci, 2017.

[16] Finn C. Learning to learn with gradients. PhD dissertation. UC Berkeley. 2018.

[17] Ho J, Ermon S. Generative adversarial imitation learning. In NIPS, 2016.

[18] Heinrich J, Silver D. Deep reinforcement learning from self-play in imperfect-information games. 2016.

[19] Xiong X, et al. Practical deep reinforcement learning approach for stock trading arXiv preprint, arXiv:1811.07522, 2018.

[20] Xinyi, et al. Optimistic bull or pessimistic bear: adaptive deep reinforcement learning for stock portfolio allocation. arXiv preprint arXiv:1907.01503, 2019.

[21] Li, Y., P. Ni, and V. Chang. An empirical research on the investment strategy of stock market based on deep reinforcement learning model. 2019.

[22] Azhikodan, A.R., A.G. Bhat, and M.V. Jadhav. Stock Trading Bot Using Deep Reinforcement Learning, in Innovations in Computer Science and Engineering. 2019, Springer. p. 41-49.

[23] Liang, Z., et al. Adversarial deep reinforcement learning in portfolio management. arXiv preprint arXiv:1808.09940, 2018.

联系人：张娜　　邮箱：zhangna@pbcsf.tsinghua.edu.cn