Journal of Mathematical Sciences: Advances and Applications
Volume 76, 2024, Pages 39-57
Available at http://scientificadvances.co.in
DOI: http://dx.doi.org/10.18642/jmsaa_7100122308

CONVERGENCE OF THE VALUE ITERATION METHOD FOR BELLMAN OPTIMAL PROBLEM AND APPLICATIONS

XUANYU LIU, SHAO HUANG, MENGQIU FAN, XIN YIN, YAODONG ZHAO* and LIMING ZHOU

National Key Laboratory of Electromagnetic Space Security
Southwest China Research Institute of Electronic Equipment
Chengdu 610036
P. R. China
e-mail: xyliu_2024@163.com
huangshaolee@163.com
fanmengqiu@cetc.com.cn
yinxin_2015@163.com
zhyd0921@163.com
zhouuestc@163.com

2020 Mathematics Subject Classification: 65Dxx.
Keywords and phrases: reinforcement learning, Bellman optimal problem, value iteration, convergence analysis, acceleration techniques.
*Corresponding author.
Received August 15, 2024

Abstract

Reinforcement Learning (RL) has emerged as a widely applicable and effective paradigm for addressing decision-making problems across diverse domains. RL encapsulates decision-making problems within the framework of Markov decision processes (MDPs), with the resolution of the Bellman optimal problem (BOP) being the quintessential endeavor. We present a rigorous proof of the convergence for the value iteration method of BOP, highlighting its exponential rate. Building upon this foundation, we introduce two novel acceleration techniques-transition set and multiple step update-that enhance the efficiency of Q-Learning and Deep Q-Networks. Our extensive numerical experiments confirm the effectiveness of these techniques.

1. Introduction

Reinforcement learning (RL) has found widespread applications in addressing decision-making problems across various domains, ranging from game playing [8, 13], robotics [6, 2], to natural language processing [4, 11], finance [5, 12], and autonomous driving [1, 3]. Its ability to learn optimal behaviour through interaction with an environment, guided by rewards and penalties, has yielded remarkable results. For instance, in the realm of robotics, RL techniques have been utilized to train agents for tasks such as robot navigation, manipulation, and even autonomous vehicle control [15]. In finance, RL algorithms have been deployed for portfolio management, algorithmic trading, and risk assessment [10]. These successful applications underscore the versatility and effectiveness of RL methodologies in solving complex decision-making challenges.

In the realm of RL, the decision-maker adeptly acquires the capability to formulate optimal decisions through continuous interaction with the environment. This process is effectively encapsulated within the structured confines of Markov decision processes (MDPs), which serves as a comprehensive framework for formalizing the interactive dynamics. Concurrently, the Bellman optimal problem encapsulates the essence of this interactive learning in a precise mathematical construct, offering a rigorous foundation for understanding and solving for optimal policies.

The Bellman optimal problem is typically solved by using two primary methods: value iteration and policy iteration. Value iteration method (VIM) updates the value function iteratively until it converges to the optimal value function. It uses the Bellman optimality equation to update each state's value based on the values of its successor states. Policy iteration is another method that alternates between policy evaluation and policy improvement until the optimal policy is found. In fact, RL algorithms are often divided into value-based and policy-based, essentially depending on whether value iteration or policy iteration is used to solve the Bellman optimal problem.

The landscape of RL is rich with a multitude of algorithms, like Q Learning (QL) [17], Deep Q-Network (DQN) [9, 18], Policy gradient methods [16, 14], Actor-critic methods [7], and Deep deterministic policy gradient (DDPG) [6], each tailored to tackle specific challenges within decision-making frameworks. Among these algorithms, VIM stands as a cornerstone, serving as the foundation for many value-based RL algorithms like QL and DQN. The theoretical analysis of VIM holds significant promise for enhancing and refining RL algorithms, offering invaluable insights into their convergence properties and optimization potential.

The primary objective of this paper is to provide a clear mathematical description of RL, with a specific focus on the VIM algorithm. By rigorously proving its convergence and introducing novel acceleration techniques, we aim to establish a solid theoretical foundation for VIM and contribute to the advancement of RL methodologies.

This paper is organized as follows: In Section 2, we introduce the mathematical model of MDPs and provide an overview of RL. In Section 3, we present a detailed exposition of Bellman optimal problem and the VIM algorithm and offers rigorous proofs of its convergence. In Section 4, we propose two acceleration techniques tailored for QL and DQN, respectively. In Section 5, we show numerical experiments to validate the effectiveness of these acceleration techniques.

2. Markov Decision Processes and Reinforcement Learning

The Markov decision processes (MDPs) is a basic model for reinforcement learning to describe a problem of learning from interaction to achieve a goal. The learner and decision maker is called the agent. The thing it interacts with, comprising everything outside the agent, is called the environment. These interact continually, the agent selecting actions and the environment responding to these actions and presenting new situations to the agent. There are five basic elements in MDPs, including the set of states $\mathcal{S}$, the set of actions $\mathcal{A}$, the immediate reward r, the state transition function p, and the discount factor γ.

To give a brief and clear analysis, we do not consider the possibility distribution during the agent-environment interface and assume that the interface is based on a deterministic rule.

Assumption (A) The sets of states $\mathcal{S}$ and actions $\mathcal{A}$ are finite. There are a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and a state transition function $p : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. And when the agent takes action a in state s, the reward $r(s, a)$ and the new state $p(s, a)$ are deterministic. The discount factor $\gamma \in (0, 1)$.

We have to claim that the analysis can also be extended to the situation where the reward and state transition is not deterministic and modelled by some possibility distribution.

A deterministic policy π is a mapping from the set of states $\mathcal{S}$ to the set of actions $\mathcal{A}$. And we denote the space of deterministic policies by

$$\prod := \{\pi | \pi : \mathcal{S} \to \mathcal{A}\}. \tag{1}$$

If the agent takes actions under a policy $\pi \in \prod$ and the environment starts with a state s_0, then as the interface goes by, there is a sequence

$$s_0, a_0, r_0, s_1, a_1, r_1, \cdots, s_t, a_t, r_t, \cdots,$$

where

$$a_t = \pi(s_t), \quad r_t = r(s_t, a_t), \quad s_{t+1} = p(s_t, a_t), \quad \forall\, t \in \mathbb{N}. \tag{2}$$

The return G of this process is the discounted cumulative reward, defined by

$$G = \sum_{t=0}^{\infty} \gamma^t r_t.$$

For simplicity, we denote the sequence with initial state s under policy π by $(s_0, s_1, \cdots | s_0 = s) \sim \pi$. The discount factor $\gamma \in (0, 1)$ and a bounded reward function are to endure the convergence of the series.

The goal of reinforcement learning is to find the optimal policy and obtain the return as high as possible. Now we introduce the value of a policy and define the optimal policy. The value of a state $s \in \mathcal{S}$ under a policy $\pi \in \prod$ is defined by

$$v_\pi(s) = \left[\sum_{k=0}^{\infty} \gamma^k r_k \right]_{(s_0, s_1, \cdots | s_0 = s) \sim \pi}. \tag{3}$$

We call the function v_π the state-value function for policy π. Then the optimal policy is the solution of the problem: Find $\pi^* \in \prod$ such that

$$\pi^* = \underset{\pi \in \prod}{\arg\max}\, v_\pi(s) \text{ for any } s \in \mathcal{S}. \tag{4}$$

3. Bellman Optimal Problem and Value Iteration

In this section, we first present the definition of the action-value function and introduce the Bellman optimal problem. Then we bring in VIM algorithm and give a rigorous convergence analysis.

The value of taking action a in state s under a policy π is defined by

$$q_\pi(s, a) = \left[\sum_{k=0}^{\infty} \gamma^k r_k\right]_{(s_0, s_1, \cdots | s_0 = s, a_0 = a) \sim \pi}. \tag{5}$$

We call the q_π the action-value function for policy π. We denote the space of action-value function by

$$\mathcal{Q} := \{q | q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}\}, \tag{6}$$

equipped with the infinity norm

$$\|q\|_\infty := \max_{s' \in \mathcal{S},\, a' \in \mathcal{A}} |q(s, a)|. \tag{7}$$

We see from (5) and (3) that

$$v_\pi(s) = q_\pi(s, \pi(s)) \text{ for any } s \in \mathcal{S}, \tag{8}$$

and

$$q_\pi(s, a) = r(s, a) + \gamma v_\pi(p(s, a)) \text{ for any } s \in \mathcal{S},\, a \in \mathcal{A}. \tag{9}$$

Substituting (8) into (9) yields the Bellman equation

$$q_\pi(s, a) = r(s, a) + \gamma q_\pi(p(s, a), \pi(p(s, a))) \text{ for any } s \in \mathcal{S},\, a \in \mathcal{A}. \tag{10}$$

Now we build a mapping between the policy and the action-value function. On the one hand, given a $\pi \in \prod$, since the reward function is bounded and the discount factor $\gamma \in (0, 1)$, the series (5) is convergent. This yields the existence and uniqueness of q_π. On the other hand, given a action-value function $q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, we define the greedy policy π_q by

$$\pi_q(s) = \operatorname*{argmax}_{a' \in \mathcal{A}} q(s, a') \text{ for any } s \in \mathcal{S}. \tag{11}$$

Considering the Bellman equation of a greedy policy, we introduce the Bellman optimality problem: Find $q \in \mathcal{Q}$ such that

$$q(s, a) = r(s, a) + \gamma \max_{a' \in \mathcal{A}} q(p(s, a), a') \text{ for any } s \in \mathcal{S},\, a \in \mathcal{A}. \tag{12}$$

The following two lemmas show that the greedy policy of the unique solution of the Bellman optimality problem is exactly an optimal policy.

Lemma 3.1. *If Assumption* (A) *is satisfied and* $q_1^*, q_2^* \in \mathcal{Q}$ *are both the solution of the Bellman optimality problem* (12), *then* q_1^*, q_2^*.

Proof. We have from (12) that

$$q_1^*(s, a) - q_2^*(s, a) = \gamma\Big(\max_{a' \in A} q_1^*(p(s, a), a') - \max_{a' \in A} q_2^*(p(s, a), a')\Big),$$

$$\forall\ s \in \mathcal{S}, a \in \mathcal{A}.$$

Assume that $\max\ \max_{a' \in A} q_1^*(p(s, a), a') > \max_{a' \in A} q_2^*(p(s, a), a')$ and denote $\hat{a} := \operatorname{argmax}_{a' \in A} q_1^*(p(s, a), a')$. Then we have

$$|q_1^*(s, a) - q_2^*(s, a)| = \gamma\Big(q_1^*(p(s, a), \hat{a}) - \max_{a' \in A} q_2^*(p(s, a), a')\Big)$$

$$\leq \gamma(q_1^*(p(s, a), \hat{a}) - q_2^*(p(s, a), \hat{a})),\quad \forall\ s \in \mathcal{S}, a \in \mathcal{A}.$$

Since s, a is arbitrary, we obtain

$$\|q_1^* - q_2^*\|_\infty \leq \gamma\|q_1^* - q_2^*\|_\infty \Rightarrow (1 - \gamma)\|q_1^* - q_2^*\|_\infty \leq 0.$$

Since $\gamma \in (0, 1)$, we have $q_1^* = q_2^*$. And this finishes the proof. □

Lemma 3.2. *If Assumption* (A) *is satisfied and* q^* *is the unique solution of* (12), *then the greedy policy of* q^* (*defined by* (11)), π_{q^*} *satisfies* (4).

Proof. Denote the greedy policy of q^* by π^*. Define the error $e_s := v_{\pi^*}(s) - v_\pi(s),\ \forall\ s \in \mathcal{S}$.

We first show the change of the error in the state sequence under arbitrary policy. Given $\pi \in \prod$ and $s_t \in \mathcal{S}$, we have from (8), (10), (12) and (9) that

$$\begin{aligned} e_{s_t} &= v_{\pi*}(s_t) - v_\pi(s_t) = (q_{\pi*}(s_t, a_t^*) - q_{\pi*}(s_t, a_t)) \\ &\quad + (q_{\pi*}(s_t, a_t) - q_\pi(s_t, a_t)) \\ &= (q_{\pi*}(s_t, a_t^*) - q_{\pi*}(s_t, a_t)) + \gamma\left(\max_{a' \in A} q_{\pi*}(s_{t+1}, a') - q_\pi(s_{t+1}\ a_{t+1})\right) \\ &= (q_{\pi*}(s_t, a_t^*) - q_{\pi*}(s_t, a_t)) + \gamma(v_{\pi*}(s_{t+1}) - v_\pi(s_{t+1})) \geq \gamma e_{s_{t+1}}. \end{aligned} \quad (13)$$

Here $a_t = \pi(s_t)$, $a_t^* = \pi^*(s_t)$, $s_{t+1} = p(s_t, a_t)$ and $a_{t+1} = \pi(s_{t+1})$. By deduction, we have for any $m, n \in \mathbb{N}$ and $m < n$ that

$$e_{s_m} \geq \gamma e_{s_{m+1}} \geq \cdots \geq \gamma^{n-m} e_{s_n}, \text{ where } (s_0, s_1, \cdots) \sim \pi. \quad (14)$$

For any initial state $s \in \mathcal{S}$, the state sequence is $(s_0, s_1, \cdots | s_0 = s) \sim \pi$. Since $\mathcal{S}$ is finite, the sequence must arrive a loop and we assume that $s_n = s_m$ for some $m, n \in \mathbb{N}$ and $m < n$. Using (14) and the fact $\gamma \in (0, 1)$, we obtain $(1 - \gamma^{n-m}) e_{s_n} \geq 0$ and thus $e_{s_n} \geq 0$. Using (14) again, we have

$$e_s = e_{s_0} \geq \gamma^n e_{s_n} \geq 0.$$

Recalling the definition $e_s = v_{\pi*}(s) - v_\pi(s)$, we show that

$$v_{\pi*}(s) \geq v_\pi(s) \quad \forall\ \pi \in \prod, s \in \mathcal{S}. \quad (15)$$

This completes our proof. □

Lemma 3.2 gives an idea to find the optimal policy: solve the Bellman optimality problem and get its greedy policy. This is the basic idea for many classic value-based reinforcement learning algorithms like QL and DQN. QL solves the Bellman optimality problem by iteration while DQN by gradient decent.

Under Assumption (A), with known reward function r and transition function p, we can solve the Bellman optimality problem by Richardson iteration and this is VIM algorithm. Given a initial action-value function $q_0 \in \mathcal{Q}$ with $q_0 \equiv 0$, VIM updates the action-value function by

$$q_{k+1}(s, a) = (1 - \alpha)q_k(s, a) + \alpha\Big(r(s, a) + \gamma \max_{a' \in \mathcal{A}} q_k(p(s, a), a')\Big), \quad \forall s \in \mathcal{S}, a \in \mathcal{A}, \tag{16}$$

where $\alpha \in (0, 1]$ is the step parameter.

The following theorem shows that VIM has an exponential convergence rate.

Theorem 3.1 (Convergence)**.** *If Assumption* (A) *is satisfied and* q^* *is the solution of the Bellman optimality problem* (12)*, then the action-value function sequence* $\{q_k\}_{k=0}^{\infty}$ *in VIM* (16) *satisfies*

$$\|q_k - q^*\|_\infty \le (|1 - \alpha| + |\alpha|\gamma)^k \frac{1}{1 - \gamma}\|r\|_\infty. \tag{17}$$

Proof. Define $e_k : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for $k = 0, 1 \cdots$ with $e_k(s, a) := q_k(s, a) - q^*(s, a)$. We have from (16) that

$$e_{k+1}(s, a) = (1 - \alpha)e_k(s, a) + \alpha\gamma\Big(\max_{a'} q_k(p(s, a), a') - \max_{a'} q^*(p(s, a), a')\Big).$$

This yields

$$\begin{aligned}\|e_{k+1}\|_\infty &\le |1 - \alpha|\,\|e_k\|_\infty + |\alpha|\gamma \max_{a'}|q_k(s_{t+1}, a') - q^*(s_{t+1}, a')| \\ &\le |1 - \alpha|\,\|e_k\|_\infty + |\alpha|\gamma\|e_k\|_\infty \le (|1 - \alpha| + |\alpha|\gamma)\|e_k\|_\infty \\ &\le (|1 - \alpha| + |\alpha|\gamma)^{k+1}\|e_0\|_\infty. \end{aligned} \tag{18}$$

Since $q_0 \equiv 0$, using (10) leads to

$$\|e_0\|_\infty = \|q^*\|_\infty \le \|r\|_\infty + \gamma\|q^*\|_\infty \Rightarrow \|q^*\|_\infty \le \frac{1}{1-\gamma}\|r\|_\infty.$$

Combining with (18) gives

$$\|e_{k+1}\|_\infty \le (|1-\alpha| + |\alpha|\gamma)^{k+1}\frac{1}{1-\gamma}\|r\|_\infty,$$

and this completes our proof. □

4. Acceleration Techniques

In this section, we introduce two classic algorithms in reinforcement learning: QL [17] and DQN [9]. Then we propose two acceleration techniques, transition set and multiple steps update, to speed up the algorithms.

In QL, the Q value is updated once per step and an observation about reward and next state is necessary. Then the action value of current state-action pair is updated by its Bellman optimality description. In every step, a transition (s_t, a_t, r_t, s_{t+1}) is collected to update the Q value of corresponding pair. However, the update is based on only the current interface. Considering taking advantages of the past experiences, we propose the transition set technique. Specifically, we create a transition set and store all the transitions we obtain. Then in every step, we can not only update the Q value of current state-action pair, but also all the pairs in the transition set. The accelerated QL algorithm with transition set is shown in Algorithm 1.

Algorithm 1. Accelerated Q Learning Algorithm

Initialize $q(s, a) = 0$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$ and transition set $\mathcal{T}$.

Repeat (for each episode):

Initialize state s_0

Repeat (for each step of episode, $t = 0, 1, \cdots, T$):

Choose action $a_t = \arg\max_{a \in \mathcal{A}} q_t(s_t, a)$ with ϵ-greedy strategy

Take action a_t, observe r_t, s_{t+1} and store (s_t, a_t, r_t, s_{t+1}) in transition set $\mathcal{T}$

Update the Q value by

$$q_{t+1}(s, a) = (1 - \alpha) q_t(s, a) + \alpha \left(r + \gamma \max_{a' \in \mathcal{A}} q_t(s', a') \right), \quad \forall\, (s, a, r, s') \in \mathcal{T}. \tag{19}$$

We see from Theorem 3.1 that the Q value converges in an exponential rate if we update all the state-action pairs in every step. Then in our accelerated algorithm, the Q value will converge in an exponential rate after the agent finishes exploration. We also expect a faster convergence rate even when the transition set is not complete, which can be seen in numerical experiments afterwards.

DQN shares basically the same framework as q leaning. They both collect the transition and update Q value function in one step. QL models the Q value function with a table while DQN with a neural network. Experience replay and target network are two successful skills in DQN. Experience replay is to obtain uncorrelated samples, which makes the gradient for the stochastic optimization problem more accurate. The goal of target network is to cancel the dependence of the bias term in loss function on the parameter and then the gradient decent could actually lower the loss. The basic DQN updates the parameter along a zigzag

path, taking a single step in a particular direction with each update. A more effective approach involves taking multiple steps in the same direction to enhance convergence, and we call this technique "multiple steps update". Applying the transition set and multiple steps update techniques, the accelerated DQN algorithm is shown in Algorithm 2.

Algorithm 2. Accelerated Deep Q-Network Algorithm

Initialize the Q-network $q(s, a; \theta)$ and the transition set $\mathcal{T}$

Repeat (for each episode):

Initialize state s_0

Repeat (for each step of episode, $t = 0, 1, \cdots, T$):

Choose action $a_t = \underset{a \in \mathcal{A}}{\arg\max}\, q(s_t, a; \theta)$ with ϵ-greedy strategy

Take action a_t and observe r_t, s_{t+1}

Store transition (s_t, a_t, r_t, s_{t+1}) in transition set $\mathcal{T}$

Repeat (for each step of update)

Compute $y_i = r_i + \gamma \max_{a \in \mathcal{A}} q(s_i', a; \theta)$ for all elements in transition set $\mathcal{T}$

Update the Q-network by gradient descent

$$\theta \leftarrow \theta - \alpha_t \cdot \frac{1}{n} \sum_{i \le n} (y_i - q(s_i, a_i; \theta)) \cdot \nabla_\theta q(s_i, a_i; \theta) \quad (20)$$

5. Numerical Experiments

In this section, we provide some numerical experiments to show the effect of our acceleration techniques.

In every experiment, we execute training using randomly initialized Q value over a defined number of episodes. Each episode maintains uniform step counts and initial states. Continuous Q value training ensures subsequent episodes inherited the final Q values of the previous episode. We compare the performance of basic reinforcement learning algorithms (QL and DQN) and their accelerated algorithms. At each step, we compute the Q value error $e_k := \|q_k - q^*\|_\infty$ and the number of known state-action pairs $N_{sa,\,k}$. Additionally, return for each episode is calculated.

Example 1. Consider a scenario of a treasure hunt for an adventurer on a 1-dimensional map, where the treasure is located at the far right of the map. In this example, the adventurer's position on the map serves as the state in MDPs. And the available actions for the adventurer include moving left or right. The reward is 1 when the adventurer gets close to the treasure and 0 otherwise.

We train for 50 episodes, with each episode consisting of 30 steps. The Q value error, number of known state-action pairs, and returns during the training is presented in Figures 1, 2, and 3, respectively. For the accelerated QL algorithm, Q value error begins a rapid decline at about 150 steps and exponentially decays upon collecting all state-action pairs $(N_{sa,k} = 12)$, supporting the result in Theorem 3.1. In contrast, for the basic QL algorithm, even after experiencing all state-action pairs, the Q value error exhibits no noticeable decrease until about 950 steps. Furthermore, the accelerated QL algorithm achieves a quicker convergence to a stable and high return.

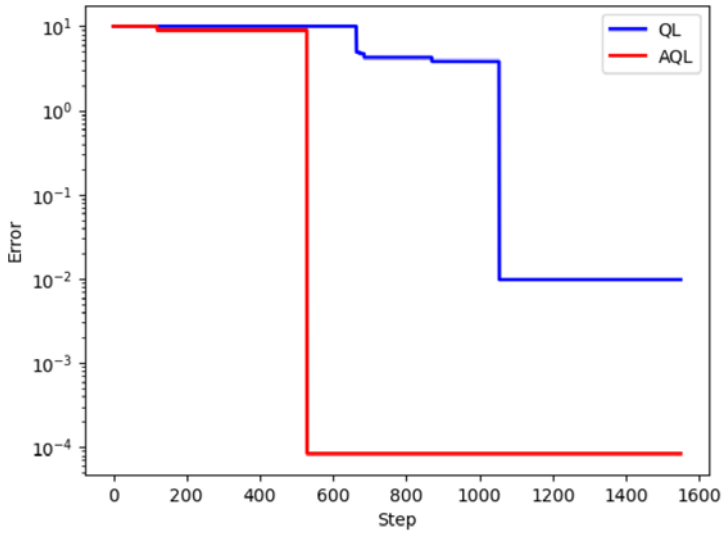

Figure 1. Q value error e_k of the basic QL and accelerated QL.

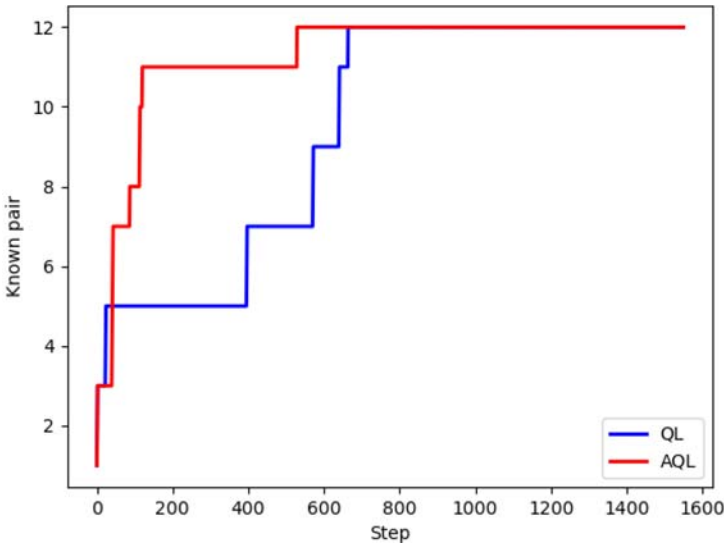

Figure 2. Number of known state-action pairs $N_{sa,\,k}$ of the basic QL and accelerated QL.

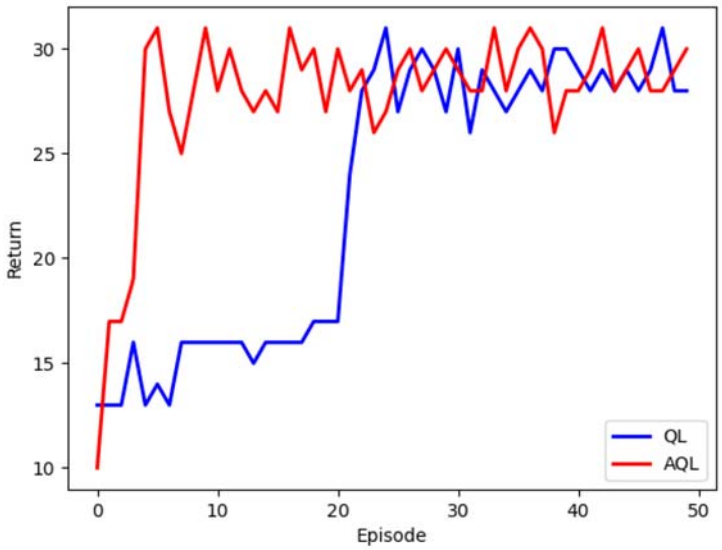

Figure 3. Returns of the basic QL and accelerated QL.

Example 2. Consider an agent moves in a gridworld [15]. The cells of the grid correspond to the states of the environment. At each cell, four actions are possible: north, south, east, and west, which deterministically cause off the grid leave its location unchanged, but also result in a reward of – 1. Other actions result in a reward of 0, except those that move the agent out of the special states A and B. From state A, all four actions yield a reward of 10 and take the agent to A'. From state B, all actions yields a reward of 5 and take the agent to B'.

We train for 100 episode, with each episode consisting of 200 steps. The Q value error, number of known state-action pairs, and returns during the training is presented in Figures 4, 5, and 6, respectively. In achieving a stable convergent state for Q value error, the accelerated DQN takes approximately 2500 steps, while the basic DQN requires nearly 7000 steps. Additionally, concerning return of an episode, the accelerated DQN notably reaches a stable high-return state at a faster rate. These observations validate the effectiveness of out acceleration techniques.

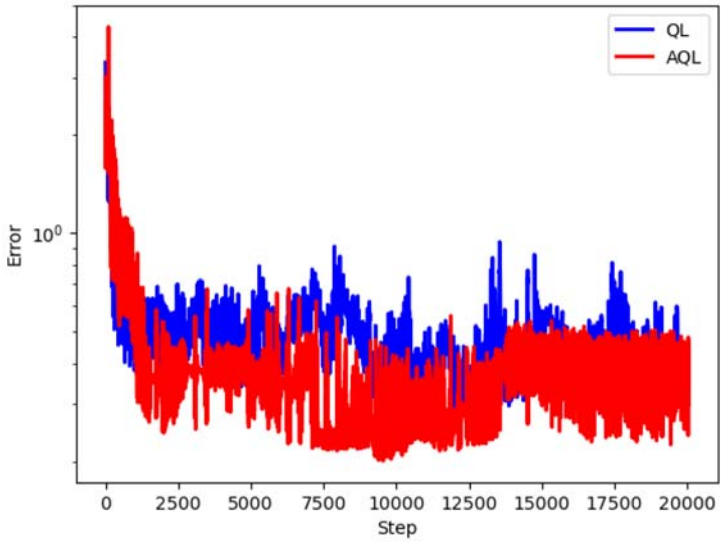

Figure 4. Q value error e_k of the basic DQN and accelerated DQN.

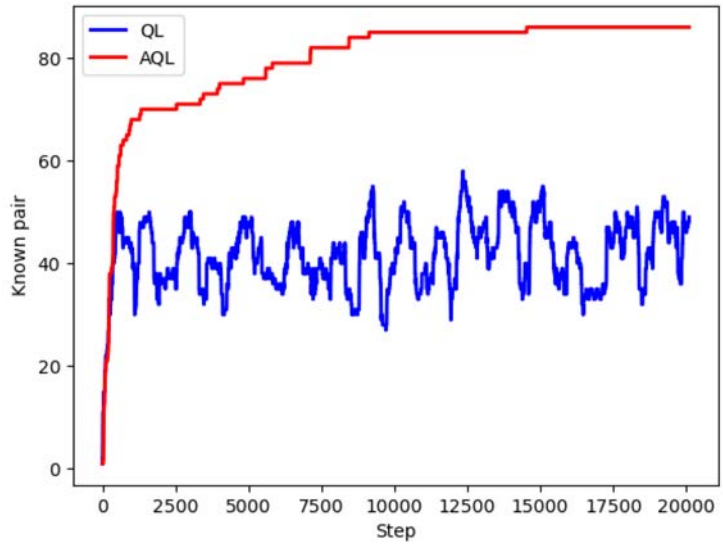

Figure 5. Number of known state-action pairs $N_{sa,\,k}$ of the basic DQN and accelerated DQN.

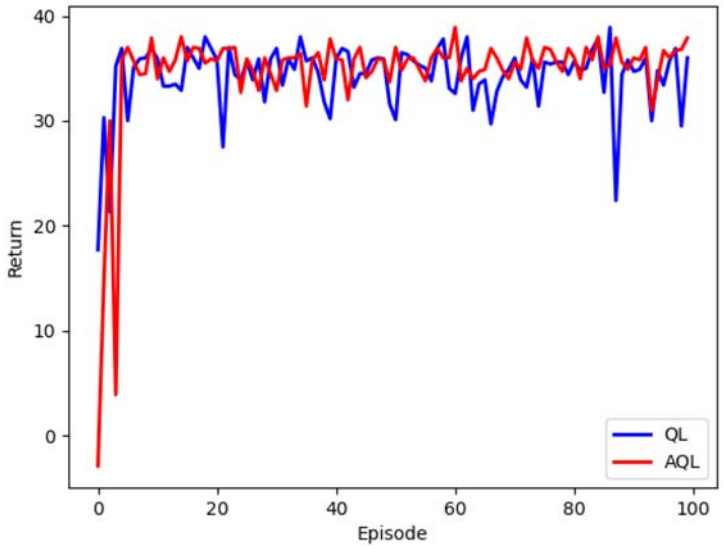

Figure 6. Returns of the basic DQN and accelerated DQN.

6. Conclusion

In this paper, we have examined the convergence of the value iteration method of Bellman optimal problem and demonstrated its exponential decay in a deterministic environment. This observation provides insights into potential enhancements for RL algorithms that rely on VIM, such as Q-Learning and DQN. Drawing upon the concepts derived from the convergence analysis, we introduce two acceleration techniques: transition set and multiple step update. These techniques are incorporated into Q-Learning and DQN to expedite the algorithms, and numerical experiments are conducted to showcase the acceleration effect.

References

[1] R. Chopra and S. S. Roy, End-to-end reinforcement learning for self-driving car, Advanced Computing and Intelligent Engineering (2020), 53-61.

DOI: https://doi.org/10.1007/978-981-15-1081-6_5

[2] T. Haarnoja, A. Zhou, P. Abbeel and S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, ArXiv: 1801.01290 (2018).

DOI: https://doi.org/10.48550/arXiv.1801.01290

[3] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J. M. Allen, V.-D. Lam, A. Bewley and A. Shah, Learning to drive in a day, In 2019 International Conference on Robotics and Automation (ICRA) (2018), 8248-8254.

DOI: https://doi.org/10.48550/arXiv.1807.00412

[4] J. Li, W. Monroe, A. Ritter, D. Jurafsky, M. Galley and J. Gao, Deep reinforcement learning for dialogue generation, In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1192-1202, Austin, Texas. Association for Computational Linguistics.

DOI: https://doi.org/10.18653/v1/D16-1127

[5] Z. Liang, H. Chen, J. Zhu, K. Jiang and Y. Li, Adversarial deep reinforcement learning in portfolio management, ArXiv: 1808.09940, (2018).

DOI: https://doi.org/10.48550/arXiv.1808.09940

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, Continuous control with deep reinforcement learning, ArXiv: 1509.02971, (2015).

DOI: https://doi.org/10.48550/arXiv.1509.02971

[7] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver and K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in International Conference on Machine Learning (2016), 1928-1937.

DOI: https://doi.org/10.48550/arXiv.1602.01783

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, Playing Atari with deep reinforcement learning, ArXiv: 1312.5602 (2013).

DOI: https://doi.org/10.48550/arXiv.1312.5602

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015), 529-533.

DOI: https://doi.org/10.1038/nature14236

[10] J. Moody and M. Saffell, Learning to trade via direct reinforcement, IEEE Transactions on Neural Networks 12(4) (2001), 875-889.

DOI: https://doi.org/10.1109/72.935097

[11] K. Narasimhan, T. Kulkarni and R. Barzilay, Language understanding for text-based games using deep reinforcement learning, In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1-11, Lisbon, Portugal. Association for Computational Linguistics.

DOI: https://doi.org/10.18653/v1/D15-1001

[12] Y. Nevmyvaka, Y. Feng and M. Kearns, Reinforcement learning for optimized trade execution, Proceedings of the 23rd International Conference on Machine Learning (2006), 673-680.

DOI: https://doi.org/10.1145/1143844.1143929

[13] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel and D. Hassabis, Mastering the game of go with deep neural networks and tree search, Nature 529 (2016), 484-489.

DOI: https://doi.org/10.1038/nature16961

[14] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller, Deterministic policy gradient algorithms, Proceedings of the 31st International Conference on International Conference on Machine Learning 32 (2024), 387-395.

[15] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction, IEEE Transactions on Neural Networks 9(5) (1998), 1054-1054.

DOI: https://doi.org/10.1109/TNN.1998.712192

[16] R. S. Sutton, D. McAllester, S. Singh and Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, Proceedings of the 12th International Conference on Neural Information Processing Systems (2000), Pages 1057-1063.

[17] C. J. C. H. Watkins and P. Dayan, Q-Learning, Machine Learning 8 (1992), 279-292.

DOI: https://doi.org/10.1007/BF00992698

[18] Z. Yang, Y. Xie and Z. Wang, A theoretical analysis of deep q-learning, ArXiv: 1901.00137 (2019).

■