A Theoretical Analysis of Deep Q-Learning

Jianqing Fan* Zhaoran Wang† Yuchen Xie† Zhuoran Yang*

February 25, 2020

Abstract

Despite the great empirical success of deep reinforcement learning, its theoretical foundation is less well understood. In this work, we make the first attempt to theoretically understand the deep Q-network (DQN) algorithm (Mnih et al., 2015) from both algorithmic and statistical perspectives. In specific, we focus on a slight simplification of DQN that fully captures its key features. Under mild assumptions, we establish the algorithmic and statistical rates of convergence for the action-value functions of the iterative policy sequence obtained by DQN. In particular, the statistical error characterizes the bias and variance that arise from approximating the action-value function using deep neural network, while the algorithmic error converges to zero at a geometric rate. As a byproduct, our analysis provides justifications for the techniques of experience replay and target network, which are crucial to the empirical success of DQN. Furthermore, as a simple extension of DQN, we propose the Minimax-DQN algorithm for zero-sum Markov game with two players. Borrowing the analysis of DQN, we also quantify the difference between the policies obtained by Minimax-DQN and the Nash equilibrium of the Markov game in terms of both the algorithmic and statistical rates of convergence.

1 Introduction

Reinforcement learning (RL) attacks the multi-stage decision-making problems by interacting with the environment and learning from the experiences. With the breakthrough in deep learning, deep reinforcement learning (DRL) demonstrates tremendous success in solving highly challenging problems, such as the game of Go (Silver et al., 2016, 2017), computer games (Vinyals et al., 2019), robotics (Kober and Peters, 2012), dialogue systems (Chen et al., 2017). In DRL, the value or policy functions are often represented as deep neural networks and the related deep learning techniques can be readily applied. For example, deep Q-network (DQN) (Mnih et al., 2015), asynchronous advantage actor-critic (A3C) (Mnih et al., 2016), trust region policy optimization (TRPO) (Schulman et al., 2015), proximal policy optimization (PPO) (Schulman et al., 2017) build upon

*Department of Operations Research and Financial Engineering, Princeton University. Research supported by the NSF grant DMS-1662139 and DMS-1712591, the ONR grant N00014-19-1-2120, and the NIH grant 2R01-GM072611-14.

†Department of Industrial Engineering and Management Sciences, Northwestern University

classical RL methods (Watkins and Dayan, 1992; Sutton et al., 2000; Konda and Tsitsiklis, 2000) and have become benchmark algorithms for artificial intelligence.

Despite its great empirical success, there exists a substantial gap between the theory and practice of DRL. In particular, most existing theoretical work on reinforcement learning focuses on the tabular case where the state and action spaces are finite, or the case where the value function is linear. Under these restrictive settings, the algorithmic and statistical perspectives of reinforcement learning are well-understood via the tools developed for convex optimization and linear regression. However, in presence of nonlinear function approximators such as deep neural network, the theoretical analysis of reinforcement learning becomes intractable as it involves solving a highly nonconvex statistical optimization problem.

To bridge such a gap in DRL, we make the first attempt to theoretically understand DQN, which can be cast as an extension of the classical Q-learning algorithm (Watkins and Dayan, 1992) that uses deep neural network to approximate the action-value function. Although the algorithmic and statistical properties of the classical Q-learning algorithm are well-studied, theoretical analysis of DQN is highly challenging due to its differences in the following two aspects.

First, in online gradient-based temporal-difference reinforcement learning algorithms, approximating the action-value function often leads to instability. Baird (1995) proves that this is the case even with linear function approximation. The key technique to achieve stability in DQN is experience replay (Lin, 1992; Mnih et al., 2015). In specific, a replay memory is used to store the trajectory of the Markov decision process (MDP). At each iteration of DQN, a mini-batch of states, actions, rewards, and next states are sampled from the replay memory as observations to train the Q-network, which approximates the action-value function. The intuition behind experience replay is to achieve stability by breaking the temporal dependency among the observations used in training the deep neural network.

Second, in addition to the aforementioned Q-network, DQN uses another neural network named the target network to obtain an unbiased estimator of the mean-squared Bellman error used in training the Q-network. The target network is synchronized with the Q-network after each period of iterations, which leads to a coupling between the two networks. Moreover, even if we fix the target network and focus on updating the Q-network, the subproblem of training a neural network still remains less well-understood in theory.

In this paper, we focus on a slight simplification of DQN, which is amenable to theoretical analysis while fully capturing the above two aspects. In specific, we simplify the technique of experience replay with an independence assumption, and focus on deep neural networks with rectified linear units (ReLU) (Nair and Hinton, 2010) and large batch size. Under this setting, DQN is reduced to the neural fitted Q-iteration (FQI) algorithm (Riedmiller, 2005) and the technique of target network can be cast as the value iteration. More importantly, by adapting the approximation results for ReLU networks to the analysis of Bellman operator, we establish the algorithmic and statistical rates of convergence for the iterative policy sequence obtained by DQN. As shown in the main results in §3, the statistical error characterizes the bias and variance that arise from approximating the action-value function using neural network, while the algorithmic error geometrically decays to zero as the number of iteration goes to infinity.

Furthermore, we extend DQN to two-player zero-sum Markov games (Shapley, 1953). The proposed algorithm, named Minimax-DQN, can be viewed as a combination of the Minimax-Q learning algorithm for tabular zero-sum Markov games (Littman, 1994) and deep neural networks for function approximation. Compared with DQN, the main difference lies in the approaches to compute the target values. In DQN, the target is computed via maximization over the action space. In contrast, the target obtained computed by solving the Nash equilibrium of a zero-sum matrix game in Minimax-DQN, which can be efficiently attained via linear programming. Despite such a difference, both these two methods can be viewed as approximately applying the Bellman operator to the Q-network. Thus, borrowing the analysis of DQN, we also establish theoretical results for Minimax-DQN. Specifically, we quantify the suboptimality of policy returned by the algorithm by the difference between the action-value functions associated with this policy and with the Nash equilibrium policy of the Markov game. For this notion of suboptimality, we establish the both algorithmic and statistical rates of convergence, which implies that the action-value function converges to the optimal counterpart up to an unimprovable statistical error in geometric rate.

Our contribution is three-fold. First, we establish the algorithmic and statistical errors of the neural FQI algorithm, which can be viewed as a slight simplification of DQN. Under mild assumptions, our results show that the proposed algorithm obtains a sequence of Q-networks that geometrically converges to the optimal action-value function up to an intrinsic statistical error induced by the approximation bias of ReLU network and finite sample size. Second, as a byproduct, our analysis justifies the techniques of experience replay and target network used in DQN, where the latter can be viewed as a single step of the value iteration. Third, we propose the Minimax-DQN algorithm that extends DQN to two-player zero-sum Markov games. Borrowing the analysis for DQN, we establish the algorithmic and statistical convergence rates of the action-value functions associated with the sequence of policies returned by the Minimax-DQN algorithm.

1.1 Related Works

There is a huge body of literature on deep reinforcement learning, where these algorithms are based on Q-learning or policy gradient (Sutton et al., 2000). We refer the reader to Arulkumaran et al. (2017) for a survey of the recent developments of DRL. In addition, the DQN algorithm is first proposed in Mnih et al. (2015), which applies DQN to Artari 2600 games (Bellemare et al., 2013). The extensions of DQN include double DQN (van Hasselt et al., 2016), dueling DQN (Wang et al., 2016), deep recurrent Q-network (Hausknecht and Stone, 2015), asynchronous DQN (Mnih et al., 2016), and variants designed for distributional reinforcement learning (Bellemare et al., 2017; Dabney et al., 2018b,a). All of these algorithms are corroborated only by numerical experiments, without theoretical guarantees. Moreover, these algorithms not only inherit the tricks of experience replay and the target network proposed in the original DQN, but develop even more tricks to enhance the performance. Furthermore, recent works such as Schaul et al. (2016); Andrychowicz et al. (2017); Liu and Zou (2018); Zhang and Sutton (2017); Novati and Koumoutsakos (2019) study the effect of experience replay and propose various modifications.

In addition, our work is closely related to the literature on batch reinforcement learning (Lange et al., 2012), where the goal is to estimate the value function given transition data. These problems

are usually formulated into least-squares regression, for which various algorithms are proposed with finite-sample analysis. However, most existing works focus on the settings where the value function are approximated by linear functions. See Bradtke and Barto (1996); Boyan (2002); Lagoudakis and Parr (2003); Lazaric et al. (2016); Farahmand et al. (2010); Lazaric et al. (2012); Tagorti and Scherrer (2015) and the references therein for results of the least-squares policy iteration (LSPI) and Bellman residue minimization (BRM) algorithms. Beyond linear function approximation, a recent work (Farahmand et al., 2016) studies the performance of LSPI and BRM when the value function belongs to a reproducing kernel Hilbert space. However, we study the fitted Q-iteration algorithm, which is a batch RL counterpart of DQN. The fitted Q-iteration algorithm is proposed in Ernst et al. (2005), and Riedmiller (2005) proposes the neural FQI algorithm. Finite-sample bounds for FQI have been established in Murphy (2005); Munos and Szepesvári (2008) for large classes of regressors. However, their results are not applicable to ReLU networks due to the huge capacity of deep neural networks. Furthermore, various extensions of FQI are studied in Antos et al. (2008a); Farahmand et al. (2009); Tosatto et al. (2017); Geist et al. (2019) to handle continuous actions space, ensemble learning, and entropy regularization. The empirical performances of various batch RL methods have been examined in Levine et al. (2017); Agarwal et al. (2019); Fujimoto et al. (2019).

Moreover, Q-learning, and reinforcement learning methods in general, have been widely applied to dynamic treatment regimes (DTR) (Chakraborty, 2013; Laber et al., 2014; Tsiatis, 2019), where the goal is to find sequential decision rules for individual patients that adapt to time-evolving illnesses. There is a huge body of literature on this line of research. See, e.g., Murphy (2003); Zhao et al. (2009); Qian and Murphy (2011); Zhao et al. (2011); Zhang et al. (2012); Zhao et al. (2012); Goldberg and Kosorok (2012); Nahum-Shani et al. (2012); Goldberg et al. (2013); Schulte et al. (2014); Song et al. (2015); Zhao et al. (2015); Linn et al. (2017); Zhou et al. (2017); Shi et al. (2018); Zhu et al. (2019) and the references therein. Our work provides a theoretical underpinning for the application of DQN to DTR (Liu et al., 2019b) and motivates the principled usage of DRL methods in healthcare applications (Yu et al., 2019).

Furthermore, our work is also related to works that apply reinforcement learning to zero-sum Markov games. The Minimax-Q learning is proposed by Littman (1994), which is an online algorithm that is an extension Q-learning. Subsequently, for Markov games, various online algorithms are also proposed with theoretical guarantees. These works consider either the tabular case or linear function approximation. See, e.g., Bowling (2001); Conitzer and Sandholm (2007); Prasad et al. (2015); Wei et al. (2017); Pérolat et al. (2018); Srinivasan et al. (2018); Wei et al. (2017) and the references therein. In addition, batch reinforcement learning is also applied to zero-sum Markov games by Lagoudakis and Parr (2002); Pérolat et al. (2015, 2016a,b); Zhang et al. (2018), which are closely related to our work. All of these works consider either linear function approximation or a general function class with bounded pseudo-dimension (Anthony and Bartlett, 2009). However, there results cannot directly imply finite-sample bounds for Minimax-DQN due to the huge capacity of deep neural networks.

Finally, our work is also related a line of research on the model capacity of ReLU deep neural networks, which leads to understanding the generalization property of deep learning (Mohri et al.,

2012; Kawaguchi et al., 2017). Specifically, Bartlett (1998); Neyshabur et al. (2015b,a); Bartlett et al. (2017); Golowich et al. (2018); Liang et al. (2019) propose various norms computed from the networks parameters and establish capacity bounds based upon these norms. In addition, Maass (1994); Bartlett et al. (1999); Schmidt-Hieber (2020+); Bartlett et al. (2019); Klusowski and Barron (2016); Barron and Klusowski (2018); Suzuki (2019); Bauer et al. (2019) study the Vapnik-Chervonenkis (VC) dimension of neural networks and Dziugaite and Roy (2017); Neyshabur et al. (2018) establish the PAC-Bayes bounds for neural networks. Among these works, our work is more related to Schmidt-Hieber (2020+); Suzuki (2019), which relate the VC dimension of the ReLU networks to a set of hyperparameters used to define the networks. Based on the VC dimension, they study the statistical error of nonparametric regression using ReLU networks. In sum, theoretical understanding of deep learning is pertinent to the study of DRL algorithms. See Kawaguchi et al. (2017); Neyshabur et al. (2017); Fan et al. (2019) and the references therein for recent developments on theoretical analysis of the generalization property of deep learning.

1.2 Notation

For a measurable space with domain $\mathcal{S}$, we denote by $\mathcal{B}(\mathcal{S}, V)$ the set of measurable functions on $\mathcal{S}$ that are bounded by V in absolute value. Let $\mathcal{P}(\mathcal{S})$ be the set of all probability measures over $\mathcal{S}$. For any $\nu \in \mathcal{P}(\mathcal{S})$ and any measurable function $f \colon \mathcal{S} \to \mathbb{R}$, we denote by $\|f\|_{\nu,p}$ the ℓ_p-norm of f with respect to measure ν for $p \geq 1$. In addition, for simplicity, we write $\|f\|_\nu$ for $\|f\|_{2,\nu}$. In addition, let $\{f(n), g(n)\}_{n\geq 1}$ be two positive series. We write $f(n) \lesssim g(n)$ if there exists a constant C such that $f(n) \leq C \cdot g(n)$ for all n larger than some $n_0 \in \mathbb{N}$. In addition, we write $f(n) \asymp g(n)$ if $f(n) \lesssim g(n)$ and $g(n) \lesssim f(n)$.

2 Background

In this section, we introduce the background. We first lay out the formulation of the reinforcement learning problem, and then define the family of ReLU neural networks.

2.1 Reinforcement Learning

A discounted Markov decision process is defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$. Here $\mathcal{S}$ is the set of all states, which can be countable or uncountable, $\mathcal{A}$ is the set of all actions, $P \colon \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the Markov transition kernel, $R \colon \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathbb{R})$ is the distribution of the immediate reward, and $\gamma \in (0, 1)$ is the discount factor. In specific, upon taking any action $a \in \mathcal{A}$ at any state $s \in \mathcal{S}$, $P(\cdot \,|\, s, a)$ defines the probability distribution of the next state and $R(\cdot \,|\, s, a)$ is the distribution of the immediate reward. Moreover, for regularity, we further assume that $\mathcal{S}$ is a compact subset of $\mathbb{R}^r$ which can be infinite, $\mathcal{A} = \{a_1, a_2, \ldots, a_M\}$ has finite cardinality M, and the rewards are uniformly bounded by $R_{\max}$, i.e., $R(\cdot \,|s, a)$ has a range on $[-R_{\max}, R_{\max}]$ for any $s \in \mathcal{S}$ and $a \in \mathcal{A}$.

A policy $\pi \colon \mathcal{S} \to \mathcal{P}(\mathcal{A})$ for the MDP maps any state $s \in \mathcal{S}$ to a probability distribution $\pi(\cdot \,|\, s)$ over $\mathcal{A}$. For a given policy π, starting from the initial state $S_0 = s$, the actions, rewards, and states

evolve according to the law as follows:

$$\Big\{(A_t, R_t, S_{t+1}) : A_t \sim \pi(\cdot \,|\, S_t), R_t \sim R(\cdot \,|\, S_t, A_t), S_{t+1} \sim P(\cdot \,|\, S_t, A_t)\Big\}, t = 0, 1, \cdots,$$

and the corresponding value function $V^\pi \colon \mathcal{S} \to \mathbb{R}$ is defined as the cumulative discounted reward obtained by taking the actions according to π when starting from a fixed state, that is,

$$V^\pi(s) = \mathbb{E}\bigg[\sum_{t=0}^{\infty} \gamma^t \cdot R_t \,\bigg|\, S_0 = s\bigg]. \tag{2.1}$$

The policy π can be controlled by decision makers, yet the functions P and R are given by the nature or the system that are unknown to decision makers.

By the law of iterative expectation, for any policy π,

$$V^\pi(s) = \mathbb{E}\big[Q^\pi(s, A) \,\big|\, A \sim \pi(\cdot \,|\, s)\big], \qquad \forall s \in \mathcal{S}, \tag{2.2}$$

where $Q^\pi(s, a)$, called an action value function, is given by

$$Q^\pi(s, a) = \mathbb{E}\bigg[\sum_{t=0}^{\infty} \gamma^t \cdot R_t \,\bigg|\, S_0 = s, A_0 = a\bigg] = r(s, a) + \gamma \cdot \mathbb{E}\big[V^\pi(S') \,\big|\, S' \sim P(\cdot \,|\, s, a)\big], \tag{2.3}$$

with $r(s, a) = \int r R(\mathrm{d}r \,|\, s, a)$ is the expected reward at state s given action a. Moreover, we define an operator P^π by

$$(P^\pi Q)(s, a) = \mathbb{E}\big[Q(S', A') \,\big|\, S' \sim P(\cdot \,|\, s, a), A' \sim \pi(\cdot \,|\, S')\big], \tag{2.4}$$

and define the Bellman operator T^π by $(T^\pi Q)(s, a) = r(s, a) + \gamma \cdot (P^\pi Q)(s, a)$. Then Q^π in (2.3) is the unique fixed point of T^π.

The goal of reinforcement learning is to find the optimal policy, which achieves the largest cumulative reward via dynamically learning from the acquired data. To characterize optimality, by (2.2), we naturally define the optimal action-value function Q^* as

$$Q^*(s, a) = \sup_\pi Q^\pi(s, a), \qquad \forall (s, a) \in \mathcal{S} \times \mathcal{A}. \tag{2.5}$$

where the supremum is taken over all policies. In addition, for any given action-value function $Q \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, we define the greedy policy π_Q as any policy that selects the action with the largest Q-value, that is, for any $s \in \mathcal{S}$, $\pi_Q(\cdot \,|s)$ satisfies

$$\pi_Q(a \,|\, s) = 0 \quad \text{if} \quad Q(s, a) \neq \max_{a' \in \mathcal{A}} Q(s, a'). \tag{2.6}$$

Based on Q^*, we define the optimal policy π^* as any policy that is greedy with respect to Q^*. It can be shown that $Q^* = Q^{\pi^*}$. Finally, we define the Bellman optimality operator T via

$$(TQ)(s, a) = r(s, a) + \gamma \cdot \mathbb{E}\Big[\max_{a' \in \mathcal{A}} Q(S', a') \,\Big|\, S' \sim P(\cdot \,|\, s, a)\Big]. \tag{2.7}$$

Then we have the Bellman optimality equation $TQ^* = Q^*$.

Furthermore, it can be verified that the Bellman operator T is γ-contractive with respect to the supremum norm over $\mathcal{S} \times \mathcal{A}$. That is, for any two action-value functions $Q, Q' \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, it holds that $\|TQ - TQ'\|_\infty \le \gamma \cdot \|Q - Q'\|_\infty$. Such a contraction property yields the celebrated value iteration algorithm (Sutton and Barto, 2011), which constructs a sequence of action-value functions $\{Q_k\}_{k \ge 0}$ by letting $Q_k = TQ_{k-1}$ for all $k \ge 1$, where the initialization function Q_0 is arbitrary. Then it holds that $\|Q_k - Q^*\|_\infty \le \gamma^k \cdot \|Q_0 - Q^*\|_\infty$, i.e., $\{Q_k\}_{k \ge 0}$ converges to the optimal value function at a linear rate. This approach forms the basis of the neural FQI algorithm, where the Bellman operator is empirically learned from a batch of data dynamically and the action-value functions are approximated by deep neural networks.

2.2 Deep Neural Network

We study the performance of DQN with rectified linear unit (ReLU) activation function $\sigma(u) = \max(u, 0)$. For any positive integer L and $\{d_j\}_{i=0}^{L+1} \subseteq \mathbb{N}$, a ReLU network $f \colon \mathbb{R}^{d_0} \to \mathbb{R}^{d_{L+1}}$ with L hidden layers and width $\{d_j\}_{i=0}^{L+1}$ is of form

$$f(x) = W_{L+1}\sigma(W_L\sigma(W_{L-1} \ldots \sigma(W_2\sigma(W_1 x + v_1) + v_2) \ldots v_{L-1}) + v_L), \tag{2.8}$$

where $W_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ and $v_\ell \in \mathbb{R}^{d_\ell}$ are the weight matrix and the shift (bias) vector in the ℓ-th layer, respectively. Here we apply σ to to each entry of its argument in (2.8). In deep learning, the network structure is fixed, and the goal is to learn the network parameters (weights) $\{W_\ell, v_\ell\}_{\ell \in [L+1]}$ with the convention that $v_{L+1} = 0$. For deep neural networks, the number of parameters greatly exceeds the input dimension d_0. To restrict the model class, we focus on the class of ReLU networks where most parameters are zero.

Definition 2.1 (Sparse ReLU Network)**.** For any $L, s \in \mathbb{N}$, $\{d_j\}_{i=0}^{L+1} \subseteq \mathbb{N}$, and $V > 0$, the family of sparse ReLU networks bounded by V with L hidden layers, network width d, and weight sparsity s is defined as

$$\mathcal{F}(L, \{d_j\}_{i=0}^{L+1}, s, V) = \bigg\{ f \colon \max_{\ell \in [L+1]} \|\widetilde{W}_\ell\|_\infty \le 1, \sum_{\ell=1}^{L+1} \|\widetilde{W}_\ell\|_0 \le s, \max_{j \in [d_{L+1}]} \|f_j\|_\infty \le V \bigg\}, \tag{2.9}$$

where we denote (W_ℓ, v_ℓ) by $\widetilde{W}_\ell$. Moreover, f in (2.9) is expressed as in (2.8), and f_j is the j-th component of f.

Here we focus on functions that are uniformly bounded because the value functions in (2.1) and (2.3) are always bounded by $V_{\max} = R_{\max}/(1-\gamma)$. We also assume that the network weights are uniformly bounded and bounded by one without loss of generality. In the sequel, we write $\mathcal{F}(L, \{d_j\}_{j=0}^{L+1}, s, V_{\max})$ as $\mathcal{F}(L, \{d_j\}_{j=0}^{L+1}, s)$ to simplify the notation. In addition, we restrict the networks weights to be sparse, i.e., s is much smaller compared with the total number of parameters. Such an assumption implies that the network has sparse connections, which are useful for applying deep learning in memory-constrained situations such as mobile devices (Han et al., 2016; Liu et al., 2015). Empirically, sparse neural networks are realized via various regularization techniques such as Dropout (Srivastava et al., 2014), which randomly sets a fixed portion of the

network weights to zero. Moreover, sparse network architectures have recently been advocated by the intriguing lottery ticket hypothesis (Frankle and Carbin, 2019), which states that each dense network has a subnetwork with the sparse connections, when trained in isolation, achieves comparable performance as the original network. Thus, focusing on the class of sparse ReLU networks does not sacrifice the statistical accuracy.

Moreover, we introduce the notion of Hölder smoothness as follows, which is a generalization of Lipschitz continuity, and is widely used to characterize the regularity of functions.

Definition 2.2 (Hölder Smooth Function)**.** Let $\mathcal{D}$ be a compact subset of $\mathbb{R}^r$, where $r \in \mathbb{N}$. We define the set of Hölder smooth functions on $\mathcal{D}$ as

$$\mathcal{C}_r(\mathcal{D}, \beta, H) = \Bigg\{ f \colon \mathcal{D} \to \mathbb{R} \colon \sum_{\boldsymbol{\alpha} \colon |\boldsymbol{\alpha}| < \beta} \|\partial^{\boldsymbol{\alpha}} f\|_\infty + \sum_{\boldsymbol{\alpha} \colon \|\boldsymbol{\alpha}\|_1 = \lfloor \beta \rfloor} \sup_{x, y \in \mathcal{D}, x \neq y} \frac{|\partial^{\boldsymbol{\alpha}} f(x) - \partial^{\boldsymbol{\alpha}}(y)|}{\|x - y\|_\infty^{\beta - \lfloor \beta \rfloor}} \leq H \Bigg\},$$

where $\beta > 0$ and $H > 0$ are parameters and $\lfloor \beta \rfloor$ is the largest integer no greater than β. In addition, here we use the multi-index notation by letting $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_r)^\top \in \mathbb{N}^r$, and $\partial^{\boldsymbol{\alpha}} = \partial^{\alpha_1} \ldots \partial^{\alpha_r}$.

Finally, we conclude this section by defining functions that can be written as a composition of multiple Hölder functions, which captures complex mappings in real-world applications such as multi-level feature extraction.

Definition 2.3 (Composition of Hölder Functions)**.** Let $q \in \mathbb{N}$ and $\{p_j\}_{j \in [q]} \subseteq \mathbb{N}$ be integers, and let $\{a_j, b_j\}_{j \in [q]} \subseteq \mathbb{R}$ such that $a_j < b_j$ $j \in [q]$. Moreover, let $g_j \colon [a_j, b_j]^{p_j} \to [a_{j+1}, b_{j+1}]^{p_{j+1}}$ be a function, $\forall j \in [q]$. Let $(g_{jk})_{k \in [p_{j+1}]}$ be the components of g_j, and we assume that each g_{jk} is Hölder smooth, and depends on at most t_j of its input variables, where t_j could be much smaller than p_j, i.e., $g_{jk} \in \mathcal{C}_{t_j}([a_j, b_j]^{t_j}, \beta_j, H_j)$. Finally, we denote by $\mathcal{G}(\{p_j, t_j, \beta_j, H_j\}_{j \in [q]})$ the family of functions that can be written as compositions of $\{g_j\}_{j \in [q]}$, with the convention that $p_{q+1} = 1$. That is, for any $f \in \mathcal{G}(\{p_j, t_j, \beta_j, H_j\}_{j \in [q]})$, we can write

$$f = g_q \circ g_{q-1} \circ \ldots \circ g_2 \circ g_1, \tag{2.10}$$

with $g_{jk} \in \mathcal{C}_{t_j}([a_j, b_j]^{t_j}, \beta_j, H_j)$ for each $k \in [p_{j+1}]$ and $j \in [q]$.

Here f in (2.10) is a composition of q vector-valued mappings $\{g_j\}_{j \in [q]}$ where each g_j has p_{j+1} components and its k-th component, g_{jk}, $\forall k \in [p_{j+1}]$, is a Hölder smooth function defined on $[a_j, b_j]^{p_j}$. Moreover, it is well-known the statistical rate for estimating a Hölder smooth function depends on the input dimension (Tsybakov, 2008). Here we assume that g_{jk} only depends on t_j of its inputs, where $t_j \in [p_j]$ can be much smaller than p_j, which enables us to obtain a more refined analysis that adapts to the effective smoothness of f. In particular, Definition 2.3 covers the family of Hölder smooth functions and the additive model (Friedman and Stuetzle, 1981) on $[0, 1]^r$ as two special cases, where the former suffers from the curse of dimensionality whereas the latter does not.

3 Understanding Deep Q-Network

In the DQN algorithm, a deep neural network $Q_\theta \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is used to approximate Q^*, where θ is the parameter. For completeness, we state DQN as Algorithm 3 in §A. As shown in the experiments in (Mnih et al., 2015), two tricks are pivotal for the empirical success of DQN.

First, DQN use the trick of experience replay (Lin, 1992). Specifically, at each time t, we store the transition (S_t, A_t, R_t, S_{t+1}) into the replay memory $\mathcal{M}$, and then sample a minibatch of independent samples from $\mathcal{M}$ to train the neural network via stochastic gradient descent. Since the trajectory of MDP has strong temporal correlation, the goal of experience replay is to obtain uncorrelated samples, which yields accurate gradient estimation for the stochastic optimization problem.

Another trick is to use a target network $Q_{\theta^\star}$ with parameter $\theta^\star$ (current estimate of parameter). With independent samples $\{(s_i, a_i, r_i, s_i')\}_{i\in[n]}$ from the replay memory (we use s_i' instead of s_{i+1} for the next state right after s_i and a_i to avoid notation crash with next independent sample s_{i+1} in the state space), to update the parameter θ of the Q-network, we compute the target

$$Y_i = r_i + \gamma \cdot \max_{a\in\mathcal{A}} Q_{\theta^\star}(s_i', a)$$

(compare with Bellman optimality operator (2.7)), and update θ by the gradient of

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}[Y_i - Q_\theta(s_i, a_i)]^2. \tag{3.1}$$

Whereas parameter $\theta^\star$ is updated once every T_{target} steps by letting $\theta^\star = \theta$. That is, the target network is hold fixed for T_{target} steps and then updated it by the current weights of the Q-network.

To demystify DQN, it is crucial to understand the role played by these two tricks. For experience replay, in practice, the replay memory size is usually very large. For example, the replay memory size is 10^6 in Mnih et al. (2015). Moreover, DQN use the ϵ-greedy policy, which enables exploration over $\mathcal{S}\times\mathcal{A}$. Thus, when the replay memory is large, experience replay is close to sampling independent transitions from an explorative policy. This reduces the variance of the $\nabla L(\theta)$, which is used to update θ. Thus, experience replay stabilizes the training of DQN, which benefits the algorithm in terms of computation.

To understand the statistical property of DQN, we replace the experience replay by sampling independent transitions from a given distribution $\sigma \in \mathcal{P}(\mathcal{S}\times\mathcal{A})$. That is, instead of sampling from the replay memory, we sample i.i.d. observations $\{(S_i, A_i)\}_{i\in[n]}$ from σ. Moreover, for any $i \in [n]$, let R_i and S_i' be the immediate reward and the next state when taking action A_i at state S_i. Under this setting, we have $\mathbb{E}(Y_i \,|\, S_i, A_i) = (TQ_{\theta^\star})(S_i, A_i)$, where T is the Bellman optimality operator in (2.7) and $Q_{\theta^\star}$ is the target network.

Furthermore, to further understand the necessity of the target network, let us first neglect the target network and set $\theta^\star = \theta$. Using bias-variance decomposition, the the expected value of $L(\theta)$ in (3.1) is

$$\mathbb{E}\big[L(\theta)\big] = \|Q_\theta - TQ_\theta\|_\sigma^2 + \mathbb{E}\Big\{\big[Y_1 - (TQ_\theta)(S_1, A_1)\big]^2\Big\}. \tag{3.2}$$

Here the first term in (3.2) is known as the mean-squared Bellman error (MSBE), and the second term is the variance of Y_1. Whereas $L(\theta)$ can be viewed as the empirical version of the MSBE, which has bias $\mathbb{E}\{[Y_1 - (TQ_\theta)(S_1, A_1)]^2\}$ that also depends on θ. Thus, without the target network, minimizing $L(\theta)$ can be drastically different from minimizing the MSBE.

To resolve this problem, we use a target network in (3.1), which has expectation

$$\mathbb{E}\big[L(\theta)\big] = \|Q_\theta - TQ_{\theta^*}\|_\sigma^2 + \mathbb{E}\Big\{\big[Y_1 - (TQ_{\theta^*})(S_1, A_1)\big]^2\Big\},$$

where the variance of Y_1 does not depend on θ. Thus, minimizing $L(\theta)$ is close to solving

$$\underset{\theta\in\Theta}{\text{minimize}}\, \|Q_\theta - TQ_{\theta^\star}\|_\sigma^2, \tag{3.3}$$

where Θ is the parameter space. Note that in DQN we hold $\theta^\star$ still and update θ for T_{target} steps. When T_{target} is sufficiently large and we neglect the fact that the objective in (3.3) is nonconvex, we would update θ by the minimizer of (3.3) for fixed $\theta^\star$.

Therefore, in the ideal case, DQN aims to solve the minimization problem (3.3) with $\theta^\star$ fixed, and then update $\theta^\star$ by the minimizer θ. Interestingly, this view of DQN offers a statistical interpretation of the target network. In specific, if $\{Q_\theta : \theta \in \Theta\}$ is sufficiently large such that it contains $TQ_{\theta^\star}$, then (3.3) has solution $Q_\theta = TQ_{\theta^\star}$, which can be viewed as one-step of value iteration (Sutton and Barto, 2011) for neural networks. In addition, in the sample setting, $Q_{\theta^\star}$ is used to construct $\{Y_i\}_{i\in[n]}$, which serve as the response in the regression problem defined in (3.1), with $(TQ_{\theta^\star})$ being the regression function.

Furthermore, turning the above discussion into a realizable algorithm, we obtain the neural fitted Q-iteration (FQI) algorithm, which generates a sequence of value functions. Specifically, let $\mathcal{F}$ be a class of function defined on $\mathcal{S}\times\mathcal{A}$. In the k-th iteration of FQI, let $\widetilde{Q}_k$ be current estimate of Q^*. Similar to (3.1) and (3.3), we define $Y_i = R_i + \gamma\cdot\max_{a\in\mathcal{A}}\widetilde{Q}_k(S_i', a)$, and update $\widetilde{Q}_k$ by

$$\widetilde{Q}_{k+1} = \underset{f\in\mathcal{F}}{\operatorname{argmin}}\, \frac{1}{n}\sum_{i=1}^{n}\big[Y_i - f(S_i, A_i)\big]^2. \tag{3.4}$$

This gives the fitted-Q iteration algorithm, which is stated in Algorithm 1.

The step of minimization problem in (3.4) essentially finds $\widetilde{Q}_{k+1}$ in $\mathcal{F}$ such that $\widetilde{Q}_{k+1}\approx T\widetilde{Q}_k$. Let us denote $\widetilde{Q}_{k+1} = \widehat{T}_k\widetilde{Q}_k$ where $\widehat{T}_k$ is an approximation of the Bellman optimality operator T learned from the training data in the k-th iteration. With the above notation, we can now understand our Algorithm 1 as follows. Starting from the initial estimator $\widetilde{Q}_0$, collect the data $\{(S_i, A_i, R_i, S_i')\}_{i\in[n]}$ and learn the map $\widehat{T}_1$ via (3.4) and get $\widetilde{Q}_1 = \widehat{T}_1\widetilde{Q}_0$. Then, get a new batch of sample and learn the map $\widehat{T}_2$ and get $\widetilde{Q}_2 = \widehat{T}_2\widetilde{Q}_1$, and so on. Our final estimator of the action value is $\widetilde{Q}_K = \widehat{T}_K\cdots\widehat{T}_1\widetilde{Q}_0$, which resembles the updates of the value iteration algorithm at the population level.

When $\mathcal{F}$ is the family of neural networks, Algorithm 1 is known as the neural FQI algorithm, which is proposed in Riedmiller (2005). Thus, we can view neural FQI as a modification of DQN, where we replace experience replay with sampling from a fixed distribution σ, so as to understand the statistical property. As a byproduct, such a modification naturally justifies the trick of target network in DQN. In addition, note that the optimization problem in (3.4) appears in each iteration of FQI, which is nonconvex when neural networks are used. However, since we focus solely on the statistical aspect, we make the assumption that the global optima of (3.4) can be reached, which is also contained $\mathcal{F}$. Interestingly, a recent line of research on deep learning (Du et al., 2019b,a;

Zou et al., 2018; Chizat et al., 2019; Allen-Zhu et al., 2019a,b; Jacot et al., 2018; Cao and Gu, 2019; Arora et al., 2019; Weinan et al., 2019; Mei et al., 2019; Yehudai and Shamir, 2019) has established global convergence of gradient-based algorithms for empirical risk minimization when the neural networks are overparametrized. We provide more discussions on the computation aspect in §B. Furthermore, we make the i.i.d. assumption in Algorithm 1 to simplify the analysis. Antos et al. (2008b) study the performance of fitted value iteration with fixed data used in the regression sub-problems repeatedly, where the data is sampled from a single trajectory based on a fixed policy such that the induced Markov chain satisfies certain conditions on the mixing time. Using similar analysis as in Antos et al. (2008b), our algorithm can also be extended to handled fixed data that is collected beforehand.

Algorithm 1 Fitted Q-Iteration Algorithm

Input: MDP $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, function class $\mathcal{F}$, sampling distribution σ, number of iterations K, number of samples n, the initial estimator $\widetilde{Q}_0$.

for $k = 0, 1, 2, \ldots, K-1$ **do**

Sample i.i.d. observations $\{(S_i, A_i, R_i, S_i')\}_{i\in[n]}$ with (S_i, A_i) drawn from distribution σ.

Compute $Y_i = R_i + \gamma \cdot \max_{a\in\mathcal{A}} \widetilde{Q}_k(S_i', a)$.

Update the action-value function:

$$\widetilde{Q}_{k+1} \leftarrow \operatorname*{argmin}_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n}\big[Y_i - f(S_i, A_i)\big]^2.$$

end for

Define policy π_K as the greedy policy with respect to $\widetilde{Q}_K$.

Output: An estimator $\widetilde{Q}_K$ of Q^* and policy π_K.

4 Theoretical Results

We establish statistical guarantees for DQN with ReLU networks. Specifically, let Q^{π_K} be the action-value function corresponding to π_K, which is returned by Algorithm 1. In the following, we obtain an upper bound for $\|Q^{\pi_K} - Q^*\|_{1,\mu}$, where $\mu \in \mathcal{P}(\mathcal{S}\times\mathcal{A})$ is allowed to be different from ν. In addition, we assume that the state space $\mathcal{S}$ is a compact subset in $\mathbb{R}^r$ and the action space $\mathcal{A}$ is finite. Without loss of generality, we let $\mathcal{S} = [0,1]^r$ hereafter, where r is a fixed integer. To begin with, we first specify the function class $\mathcal{F}$ in Algorithm 1.

Definition 4.1 (Function Classes)**.** Following Definition 2.1, let $\mathcal{F}(L, \{d_j\}_{j=0}^{L+1}, s)$ be the family of sparse ReLU networks defined on $\mathcal{S}$ with $d_0 = r$ and $d_{L+1} = 1$. Then we define $\mathcal{F}_0$ by

$$\mathcal{F}_0 = \big\{f\colon \mathcal{S}\times\mathcal{A}\to\mathbb{R}\colon f(\cdot, a) \in \mathcal{F}(L, \{d_j\}_{i=0}^{L+1}, s) \text{ for any } a\in\mathcal{A}\big\}. \tag{4.1}$$

In addition, let $\mathcal{G}(\{p_j, t_j, \beta_j, H_j\}_{j\in[q]})$ be set of composition of Hölder smooth functions defined on $\mathcal{S}\subseteq\mathbb{R}^r$. Similar to $\mathcal{F}_0$, we define a function class $\mathcal{G}_0$ as

$$\mathcal{G}_0 = \big\{f\colon \mathcal{S}\times\mathcal{A}\to\mathbb{R}\colon f(\cdot, a) \in \mathcal{G}(\{p_j, t_j, \beta_j, H_j\}_{j\in[q]}) \text{ for any } a\in\mathcal{A}\big\}. \tag{4.2}$$

By this definition, for any function $f \in \mathcal{F}_0$ and any action $a \in \mathcal{A}$, $f(\cdot, a)$ is a ReLU network defined on $\mathcal{S}$, which is standard for Q-networks. Moreover, $\mathcal{G}_0$ contains a broad family of smooth functions on $\mathcal{S} \times \mathcal{A}$. In the following, we make a mild assumption on $\mathcal{F}_0$ and $\mathcal{G}_0$.

Assumption 4.2. We assume that for any $f \in \mathcal{F}_0$, we have $Tf \in \mathcal{G}_0$, where T is the Bellman optimality operator defined in (2.7). That is, for any $f \in \mathcal{F}$ and any $a \in \mathcal{A}$, $(Tf)(s, a)$ can be written as compositions of Hölder smooth functions as a function of $s \in \mathcal{S}$.

This assumption specifies that the target function $\mathcal{T}\widetilde{Q}_k$ in each FQI step stays in function class $\mathcal{G}_0$. When $\mathcal{G}_0$ can be approximated by functions in $\mathcal{F}_0$ accurately, this assumption essentially implies that Q^* is close to $\mathcal{F}_0$ and that $\mathcal{F}_0$ is approximately closed under Bellman operator T. Such an completeness assumption is commonly made in the literature on batch reinforcement learning under various forms and is conjectured to be indispensable in Chen and Jiang (2019).

We remark that this Assumption (4.2) holds when the MDP satisfies some smoothness conditions. For any state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$, let $P(\cdot \,|\, s, a)$ be the density of the next state. By the definition of the Bellman optimality operator in (2.7), we have

$$(Tf)(s, a) = r(s, a) + \gamma \cdot \int_{\mathcal{S}} \Big[\max_{a' \in \mathcal{A}} f(s', a')\Big] \cdot P(s' \,|\, s, a) \mathrm{d}s'. \tag{4.3}$$

For any $s' \in \mathcal{S}$ and $a \in \mathcal{A}$, we define functions g_1, g_2 by letting $g_1(s) = r(s, a)$ and $g_2(s) = P(s' \,|\, s, a)$. Suppose both g_1 and g_2 are Hölder smooth functions on $\mathcal{S} = [0, 1]^r$ with parameters β and H. Since $\|f\|_\infty \le V_{\max}$, by changing the order of integration and differentiation with respect to s in (4.3), we obtain that function $s \to (Tf)(s, a)$ belongs to the Hölder class $\mathcal{C}_r(\mathcal{S}, \beta, H')$ with $H' = H(1+V_{\max})$. Furthermore, in the more general case, suppose for any fixed $a \in \mathcal{A}$, we can write $P(s' \,|\, s, a)$ as $h_1[h_2(s, a), h_3(s')]$, where $h_2 \colon \mathcal{S} \to \mathbb{R}^{r_1}$, and $h_3 \colon \mathcal{S} \to \mathbb{R}^{r_2}$ can be viewed as feature mappings, and $h_1 \colon \mathbb{R}^{r_1 + r_2} \to \mathbb{R}$ is a bivariate function. We define function $h_4 \colon \mathbb{R}^{r_1} \to \mathbb{R}$ by

$$h_4(u) = \int_{\mathcal{S}} \Big[\max_{a' \in \mathcal{A}} f(s', a')\Big] h_1\big[u, h_3(s')\big] \mathrm{d}s'.$$

Then by (4.3) we have $(Tf)(s, a) = g_1(s) + h_4 \circ h_2(s, a)$. Then Assumption 4.2 holds if h_4 is Hölder smooth and both g_1 and h_2 can be represented as compositions of Hölder functions. Thus, Assumption 4.2 holds if both the reward function and the transition density of the MDP are sufficiently smooth.

Moreover, even when the transition density is not smooth, we could also expect Assumption 4.2 to hold. Consider the extreme case where the MDP has deterministic transitions, that is, the next state s' is a function of s and a, which is denoted by $s' = h(s, a)$. In this case, for any ReLU network f, we have $(Tf)(s, a) = r(s, a) + \gamma \cdot \max_{a' \in \mathcal{A}} f[h(s, a), a']$. Since

$$\Big|\max_{a' \in \mathcal{A}} f(s_1, a') - \max_{a' \in \mathcal{A}} f(s_2, a')\Big| \le \max_{a' \in \mathcal{A}} \big|f(s_1, a') - f(s_2, a')\big|$$

for any $s_1, s_2 \in \mathcal{S}$, and network $f(\cdot, a)$ is Lipschitz continuous for any fixed $a \in \mathcal{A}$, function $m_1(s) = \max_{a'} f(s, a')$ is Lipschitz on $\mathcal{S}$. Thus, for any fixed $a \in \mathcal{A}$, if both $g_1(s) = r(s, a)$ and $m_2(s) = h(s, a)$ are compositions of Hölder functions, so is $(Tf)(s, a) = g_1(s) + m_1 \circ m_2(s)$.

Therefore, even if the MDP has deterministic dynamics, when both the reward function $r(s,a)$ and the transition function $h(s,a)$ are sufficiently nice, Assumption 4.2 still holds true.

In the following, we define the concentration coefficients, which measures the similarity between two probability distributions under the MDP.

Assumption 4.3 (Concentration Coefficients)**.** Let $\nu_1, \nu_2 \in \mathcal{P}(\mathcal{S}\times\mathcal{A})$ be two probability measures that are absolutely continuous with respect to the Lebesgue measure on $\mathcal{S}\times\mathcal{A}$. Let $\{\pi_t\}_{t\geq 1}$ be a sequence of policies. Suppose the initial state-action pair (S_0, A_0) of the MDP has distribution ν_1, and we take action A_t according to policy π_t. For any integer m, we denote by $P^{\pi_m}P^{\pi_{m-1}}\cdots P^{\pi_1}\nu_1$ the distribution of $\{(S_t, A_t)\}_{t=0}^m$. Then we define the m-th concentration coefficient as

$$\kappa(m;\nu_1,\nu_2) = \sup_{\pi_1,\ldots,\pi_m}\left[\mathbb{E}_{\nu_2}\left|\frac{\mathrm{d}(P^{\pi_m}P^{\pi_{m-1}}\cdots P^{\pi_1}\nu_1)}{\mathrm{d}\nu_2}\right|^2\right]^{1/2}, \tag{4.4}$$

where the supremum is taken over all possible policies.

Furthermore, let σ be the sampling distribution in Algorithm 1 and let μ be a fixed distribution on $\mathcal{S}\times\mathcal{A}$. We assume that there exists a constant $\phi_{\mu,\sigma} < \infty$ such that

$$(1-\gamma)^2\cdot\sum_{m\geq 1}\gamma^{m-1}\cdot m\cdot\kappa(m;\mu,\sigma)\leq\phi_{\mu,\sigma}, \tag{4.5}$$

where $(1-\gamma)^2$ in (4.5) is a normalization term, since $\sum_{m\geq 1}\gamma^{m-1}\cdot m = (1-\gamma)^{-2}$.

By definition, concentration coefficients in (4.4) quantifies the similarity between ν_2 and the distribution of the future states of the MDP when starting from ν_1. Moreover, (4.5) is a standard assumption in the literature. See, e.g., Munos and Szepesvári (2008); Lazaric et al. (2016); Scherrer et al. (2015); Farahmand et al. (2010, 2016). This assumption holds for large class of systems MDPs and specifically for MDPs whose top-Lyapunov exponent is finite. Moreover, this assumption essentially requires that the sampling distribution σ has sufficient coverage over $\mathcal{S}\times\mathcal{A}$ and is shown to be necessary for the success of batch RL methods (Chen and Jiang, 2019). See Munos and Szepesvári (2008); Antos et al. (2007); Chen and Jiang (2019) for more detailed discussions on this assumption.

Now we are ready to present the main theorem.

Theorem 4.4. Under Assumptions 4.2 and 4.3, let $\mathcal{F}_0$ be defined in (4.1) based on the family of sparse ReLU networks $\mathcal{F}(L^*, \{d_j^*\}_{j=0}^{L^*+1}, s^*)$ and let $\mathcal{G}_0$ be given in (4.2) with $\{H_j\}_{j\in[q]}$ being absolute constants. Moreover, for any $j\in[q-1]$, we define $\beta_j^* = \beta_j\cdot\prod_{\ell=j+1}\min(\beta_\ell, 1)$; let $\beta_q^* = 1$. In addition, let $\alpha^* = \max_{j\in[q]} t_j/(2\beta_j^* + t_j)$. For the parameters of $\mathcal{G}_0$, we assume that the sample size n is sufficiently large such that there exists a constant $\xi > 0$ satisfying

$$\max\left\{\sum_{j=1}^q (t_j+\beta_j+1)^{3+t_j},\quad \sum_{j\in[q]}\log(t_j+\beta_j),\quad \max_{j\in[q]} p_j\right\}\lesssim(\log n)^\xi. \tag{4.6}$$

For the hyperparameters L^*, $\{d_j^*\}_{j=0}^{L^*+1}$, and s^* of the ReLU network, we set $d_0^* = 0$ and $d_{L^*+1}^* = 1$. Moreover, we set

$$L^*\lesssim(\log n)^{\xi^*},\quad r\leq\min_{j\in[L^*]} d_j^*\leq\max_{j\in[L^*]} d_j^*\lesssim n^{\xi^*},\quad\text{and}\quad s^*\asymp n^{\alpha^*}\cdot(\log n)^{\xi^*} \tag{4.7}$$

for some constant $\xi^* > 1 + 2\xi$. For any $K \in \mathbb{N}$, let Q^{π_K} be the action-value function corresponding to policy π_K, which is returned by Algorithm 1 based on function class $\mathcal{F}_0$. Then there exists a constant $C > 0$ such that

$$\|Q^* - Q^{\pi_K}\|_{1,\mu} \le C \cdot \frac{\phi_{\mu,\sigma} \cdot \gamma}{(1-\gamma)^2} \cdot |\mathcal{A}| \cdot (\log n)^{1+2\xi^*} \cdot n^{(\alpha^*-1)/2} + \frac{4\gamma^{K+1}}{(1-\gamma)^2} \cdot R_{\max}. \tag{4.8}$$

This theorem implies that the statistical rate of convergence is the sum of a statistical error and an algorithmic error. The algorithmic error converges to zero in linear rate as the algorithm proceeds, whereas the statistical error reflects the fundamental difficulty of the problem. Thus, when the number of iterations satisfy

$$K \ge C' \cdot \big[\log |A| + (1-\alpha^*) \cdot \log n\big] / \log(1/\gamma)$$

iterations, where C' is a sufficiently large constant, the algorithmic error is dominated by the statistical error. In this case, if we view both γ and $\phi_{\mu,\sigma}$ as constants and ignore the polylogarithmic term, Algorithm 1 achieves error rate

$$|\mathcal{A}| \cdot n^{(\alpha^*-1)/2} = |\mathcal{A}| \cdot \max_{j \in [q]} n^{-\beta_j^*/(2\beta_j^*+t_j)}, \tag{4.9}$$

which scales linearly with the capacity of the action space, and decays to zero when the n goes to infinity. Furthermore, the rates $\{n^{-\beta_j^*/(2\beta_j^*+t_j)}\}_{j\in[q]}$ in (4.9) recovers the statistical rate of nonparametric regression in ℓ_2-norm, whereas our statistical rate $n^{(\alpha^*-1)/2}$ in (4.9) is the fastest among these nonparametric rates, which illustrates the benefit of compositional structure of $\mathcal{G}_0$.

Furthermore, as a concrete example, we assume that both the reward function and the Markov transition kernel are Hölder smooth with smoothness parameter β. As stated below Assumption 4.2, for any $f \in \mathcal{F}_0$, we have $(Tf)(\cdot, a) \in \mathcal{C}_r(\mathcal{S}, \beta, H')$. Then Theorem 4.4 implies that Algorithm 1 achieves error rate $|\mathcal{A}| \cdot n^{-\beta/(2\beta+r)}$ when K is sufficiently large. Since $|\mathcal{A}|$ is finite, this rate achieves the minimax-optimal statistical rate of convergence within the class of Hölder smooth functions defined on $[0,1]^d$ (Stone, 1982) and thus cannot be further improved. As another example, when $(Tf)(\cdot, a) \in \mathcal{C}_r(\mathcal{S}, \beta, H')$ can be represented as an additive model over $[0,1]^r$ where each component has smoothness parameter β, (4.9) reduces to $|\mathcal{A}| \cdot n^{-\beta/(2\beta+1)}$, which does not depends on the input dimension r explicitly. Thus, by having a composite structure in $\mathcal{G}_0$, Theorem 4.4 yields more refined statistical rates that adapt to the intrinsic difficulty of solving each iteration of Algorithm 1.

In the sequel, we conclude this section by sketching the proof of Theorem 4.4; the detailed proof is deferred to §6.

Proof Sketch of Theorem 4.4. Recall that π_k is the greedy policy with respect to $\widetilde{Q}_k$ and Q^{π_K} is the action-value function associated with π_K, whose definition is given in (2.3). Since $\{\widetilde{Q}_k\}_{k\in[K]}$ is constructed by a iterative algorithm, it is crucial to relate $\|Q^* - Q^{\pi_K}\|_{1,\mu}$, the quantity of interest, to the errors incurred in the previous steps, namely $\{\widetilde{Q}_k - T\widetilde{Q}_{k-1}\}_{k\in[K]}$. Thus, in the first step of the proof, we establish Theorem 6.1, also known as the error propagation (Munos and Szepesvári, 2008; Lazaric et al., 2016; Scherrer et al., 2015; Farahmand et al., 2010, 2016) in the batch reinforcement

learning literature, which provides an upper bound on $\|Q^* - Q^{\pi_K}\|_{1,\mu}$ using $\{\|\widetilde{Q}_k - T\widetilde{Q}_{k-1}\|_\sigma\}_{k\in[K]}$. In particular, Theorem 6.1 asserts that

$$\|Q^* - Q^{\pi_K}\|_{1,\mu} \leq \frac{2\phi_{\mu,\sigma}\cdot\gamma}{(1-\gamma)^2}\cdot\max_{k\in[K]}\|\widetilde{Q}_k - T\widetilde{Q}_{k-1}\|_\sigma + \frac{4\gamma^{K+1}}{(1-\gamma)^2}\cdot R_{\max}, \tag{4.10}$$

where $\phi_{\mu,\sigma}$, given in (4.5), is a constant that only depends on distributions μ and σ.

The upper bound in (4.10) shows that the total error of Algorithm 1 can be viewed as a sum of a statistical error and an algorithmic error, where $\max_{k\in[K]}\|\widetilde{Q}_k - T\widetilde{Q}_{k-1}\|_\sigma$ is essentially the statistical error and the second term on the right-hand side of (4.10) corresponds to the algorithmic error. Here, the statistical error diminishes as the sample size n in each iteration grows, whereas the algorithmic error decays to zero geometrically as the number of iterations K increases. This implies that the fundamental difficulty of DQN is captured by the error incurred in a single step. Moreover, the proof of this theorem depends on bounding $\widetilde{Q}_\ell - T\widetilde{Q}_{\ell-1}$ using $\widetilde{Q}_k - T\widetilde{Q}_{k-1}$ for any $k < \ell$, which characterizes how the one-step error $\widetilde{Q}_k - T\widetilde{Q}_{k-1}$ propagates as the algorithm proceeds. See §C.1 for a detailed proof.

It remains to bound $\|\widetilde{Q}_k - T\widetilde{Q}_{k-1}\|_\sigma$ for any $k \in [K]$. We achieve such a goal using tools from nonparametric regression. Specifically, as we will show in Theorem 6.2, under Assumption 4.2, for any $k \in [K]$ we have

$$\|\widetilde{Q}_{k+1} - T\widetilde{Q}_k\|_\sigma^2 \leq 4\cdot[\mathrm{dist}_\infty(\mathcal{F}_0,\mathcal{G}_0)]^2 + C\cdot V_{\max}^2/n\cdot\log N_\delta + C\cdot V_{\max}\cdot\delta \tag{4.11}$$

for any $\delta > 0$, where $C > 0$ is an absolute constant,

$$\mathrm{dist}_\infty(\mathcal{F}_0,\mathcal{G}_0) = \sup_{f'\in\mathcal{G}_0}\inf_{f\in\mathcal{F}_0}\|f - f'\|_\infty \tag{4.12}$$

is the ℓ_∞-error of approximating functions in $\mathcal{G}_0$ using functions in $\mathcal{F}_0$, and N_δ is the minimum number of cardinality of the balls required to cover $\mathcal{F}_0$ with respect to ℓ_∞-norm.

Furthermore, (4.11) characterizes the bias and variance that arise in estimating the action-value functions using deep ReLU networks. Specifically, $[\mathrm{dist}_\infty(\mathcal{F}_0,\mathcal{G}_0)]^2$ corresponds to the bias incurred by approximating functions in $\mathcal{G}_0$ using ReLU networks. We note that such a bias is measured in the ℓ_∞-norm. In addition, $V_{\max}^2/n\cdot\log N_\delta + V_{\max}\cdot\delta$ controls the variance of the estimator.

In the sequel, we fix $\delta = 1/n$ in (4.11), which implies that

$$\|\widetilde{Q}_{k+1} - T\widetilde{Q}_k\|_\sigma^2 \leq 4\cdot\mathrm{dist}_\infty^2(\mathcal{F}_0,\mathcal{G}_0) + C'\cdot V_{\max}^2/n\cdot\log N_\delta, \tag{4.13}$$

where $C' > 0$ is an absolute constant.

In the subsequent proof, we establish upper bounds for $\mathrm{dist}(\mathcal{F}_0,\mathcal{G}_0)$ defined in (4.12) and $\log N_\delta$, respectively. Recall that the family of composite Hölder smooth functions $\mathcal{G}_0$ is defined in (4.2).

By the definition of $\mathcal{G}_0$ in (4.2), for any $f \in \mathcal{G}_0$ and any $a \in \mathcal{A}$, $f(\cdot,a) \in \mathcal{G}(\{(p_j,t_j,\beta_j,H_j)\}_{j\in[q]})$ is a composition of Hölder smooth functions, that is, $f(\cdot,a) = g_q\circ\cdots\circ g_1$. Recall that, as defined in Definition 2.3, g_{jk} is the k-th entry of the vector-valued function g_j. Here $g_{jk} \in \mathcal{C}_{t_j}([a_j,b_j]^{t_j},\beta_j,H_j)$ for each $k \in [p_{j+1}]$ and $j \in [q]$. To construct a ReLU network that is $f(\cdot,a)$, we first show that $f(\cdot,a)$

can be reformulated as a composition of Hölder functions defined on a hypercube. Specifically, let $h_1 = g_1/(2H_1) + 1/2$, $h_q(u) = g_q(2H_{q-1}u - H_{q-1})$, and

$$h_j(u) = g_j(2H_{j-1}u - H_{j-1})/(2H_j) + 1/2$$

for all $j \in \{2, \ldots, q-1\}$. Then we immediately have

$$f(\cdot, a) = g_q \circ \cdots \circ g_1 = h_q \circ \cdots \circ h_1. \tag{4.14}$$

Furthermore, by the definition of Hölder smooth functions in Definition 2.2, for any $j \in [q]$ and $k \in [p_{j+1}]$, it is not hard to verify that $h_{jk} \in \mathcal{C}_{t_j}\big([0,1]^{t_j}, W\big)$, where we define $W > 0$ by

$$W = \max\Big\{\max_{1\le j\le q-1}(2H_{j-1})^{\beta_j}, H_q(2H_{q-1})^{\beta_q}\Big\}, \tag{4.15}$$

Now we employ Lemma 6.3, obtained from Schmidt-Hieber (2020+), to construct a ReLU network that approximates each h_{jk}, which, combined with (4.14), yields a ReLU network that is close to $f(\cdot, a)$ in the ℓ_∞-norm.

To apply Lemma 6.3 we set $m = \eta \cdot \lceil \log_2 n \rceil$ for a sufficiently large constant $\eta > 1$, and set N to be a sufficiently large integer that depends on n, which will be specified later. In addition, we set $L_j = 8 + (m+5)\cdot(1 + \lceil \log_2(t_j + \beta_j)\rceil)$. Then, by Lemma 6.3, there exists a ReLU network $\widetilde{h}_{jk}$ such that $\|\widetilde{h}_{jk} - h_{jk}\|_\infty \lesssim N^{-\beta_j/t_j}$. Furthermore, we have $\widetilde{h}_{jk} \in \mathcal{F}(L_j, \{t_j, \widetilde{d}_j, \ldots, \widetilde{d}_j, 1\}, \widetilde{s}_j)$, with

$$\widetilde{d}_j = 6(t_j + \lceil\beta_j\rceil)N, \qquad \widetilde{s}_j \le 141\cdot(t_j + \beta_j + 1)^{3+t_j}\cdot N\cdot(m+6). \tag{4.16}$$

Now we define $\widetilde{f}\colon \mathcal{S} \to \mathbb{R}$ by $\widetilde{f} = \widetilde{h}_q \circ \cdots \circ \widetilde{h}_1$ and set

$$N = \Big\lceil \max_{1\le j\le q} C\cdot n^{t_j/(2\beta_j^* + t_j)}\Big\rceil. \tag{4.17}$$

For this choice of N, we show that $\widetilde{f}$ belongs to function class $\mathcal{F}(L^*, \{d_j^*\}_{j=1}^{L^*+1}, s^*)$. Moreover, we define $\lambda_j = \prod_{\ell=j+1}^{q}(\beta_\ell \wedge 1)$ for any $j \in [q-1]$, and set $\lambda_q = 1$. Then we have $\beta_j \cdot \lambda_j = \beta_j^*$ for all $j \in [q]$. Furthermore, we show that $\widetilde{f}$ is a good approximation of $f(\cdot, a)$. Specifically, we have

$$\|f(\cdot, a) - \widetilde{f}\|_\infty \lesssim \sum_{j=1}^{q}\|\widetilde{h}_j - h_j\|_\infty^{\lambda_j}.$$

Combining this with (4.17) and the fact that $\|\widetilde{h}_{jk} - h_{jk}\|_\infty \lesssim N^{-\beta_j/t_j}$, we obtain that

$$\big[\mathrm{dist}(\mathcal{F}_0, \mathcal{G}_0)\big]^2 \lesssim n^{\alpha^*-1}. \tag{4.18}$$

Moreover, using classical results on the covering number of neural networks (Anthony and Bartlett, 2009), we further show that

$$\log N_\delta \lesssim |\mathcal{A}|\cdot s^*\cdot L^*\max_{j\in[L^*]}\log(d_j^*) \lesssim |\mathcal{A}|\cdot n^{\alpha^*}\cdot(\log n)^{1+2\xi^*}, \tag{4.19}$$

where $\delta = 1/n$. Therefore, combining (4.11), (4.13), (4.18), and (4.19), we conclude the proof. □