On the Expressivity of Neural Networks for Deep Reinforcement Learning

Kefan Dong [*1] **Yuping Luo** [*2] **Tianhe Yu** [3] **Chelsea Finn** [3] **Tengyu Ma** [3]

Abstract

We compare the model-free reinforcement learning with the model-based approaches through the lens of the expressive power of neural networks for policies, Q-functions, and dynamics. We show, theoretically and empirically, that even for one-dimensional continuous state space, there are many MDPs whose optimal Q-functions and policies are much more complex than the dynamics. For these MDPs, model-based planning is a favorable algorithm, because the resulting policies can approximate the optimal policy significantly better than a neural network parameterization can, and model-free or model-based policy optimization rely on policy parameterization. Motivated by the theory, we apply a simple multi-step model-based bootstrapping planner (BOOTS) to bootstrap a weak Q-function into a stronger policy. Empirical results show that applying BOOTS on top of model-based or model-free policy optimization algorithms at the test time improves the performance on benchmark tasks.

1. Introduction

Model-based deep reinforcement learning (RL) algorithms offer a lot of potentials in achieving significantly better sample efficiency than the model-free algorithms for continuous control tasks. We can largely categorize the model-based deep RL algorithms into two types: 1. model-based policy optimization algorithms which learn policies or Q-functions, parameterized by neural networks, on the estimated dynamics, using off-the-shelf model-free algorithms or their variants (Luo et al., 2019; Janner et al., 2019; Kaiser et al., 2019; Kurutach et al., 2018; Feinberg et al., 2018; Buckman et al., 2018), and 2. model-based planning algorithms, which plan with the estimated dynamics (Nagabandi et al., 2018; Chua et al., 2018; Wang & Ba, 2019).

A deeper theoretical understanding of the pros and cons of model-based and the model-free algorithms in the continuous state space case will provide guiding principles for designing and applying new sample-efficient methods. The prior work on the comparisons of model-based and model-free algorithms mostly focuses on their sample efficiency gap, in the case of tabular MDPs (Zanette & Brunskill, 2019; Jin et al., 2018), linear quadratic regulator (Tu & Recht, 2018), and contextual decision process with sparse reward (Sun et al., 2019).

In this paper, we theoretically compare model-based RL and model-free RL in the continuous state space through the lens of *approximability* by neural networks. What is the representation power of neural networks for expressing the Q-function, the policy, and the dynamics?

Our main finding is that even for the case of one-dimensional continuous state space, there can be a massive gap between the approximability of Q-function and the policy and that of the dynamics. The optimal Q-function and policy can require *exponentially more neurons* to approximate by neural networks than the dynamics.

We construct environments where the dynamics are simply piecewise linear functions with constant pieces, but the optimal Q-functions and the optimal policy require an exponential (in the horizon) number of linear pieces, or exponentially wide neural networks, to approximate.[1] The approximability gap can also be observed empirically on (semi-) randomly generated piecewise linear dynamics with a decent chance. (See Figure 1 for two examples.) This indicates that the such MDPs are common in the sense that they do not form a degenerate set of measure zero.

We note that for tabular MDPs, it has long been known that for factored MDPs, the dynamics can be simple whereas the value function is not (Koller & Parr, 1999). This is to our knowledge the first theoretical study of the expressivity of

[*]Equal contribution [1]Institute for Interdisciplinary Information Sciences, Tsinghua University [2]Computer Science Department, Princeton University [3]Department of Computer Science, Stanford University. Correspondence to: Tengyu Ma <tengyuma@stanford.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

[1]In turn, the dynamics can also be much more complex than the Q-function. Consider the following situation: a subset of the coordinates of the state space can be arbitrarily difficult to express by neural networks, but the reward function can only depend on the rest of the coordinates and remain simple.

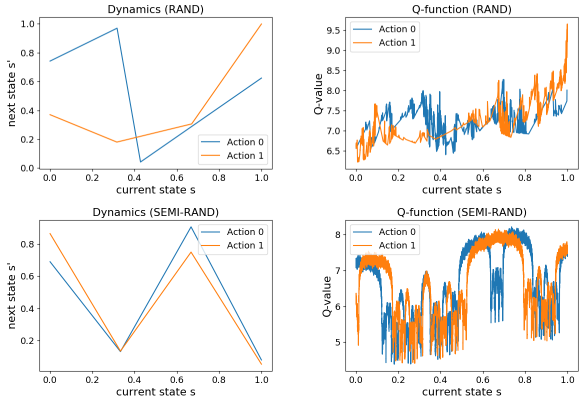

Figure 1. **Left:** the dynamics of two randomly generated MDPs (from the RAND, and SEMI-RAND methods outlined in Section 5.1 and detailed in Appendix D.1). **Right:** The corresponding Q-functions which are more complex than the dynamics. (More details in Section 5.1.).

neural networks in the contexts of deep reinforcement learning. Moreover, it's perhaps somewhat surprising that an approximation power gap can occur even for one-dimensional state space with continuous dynamics.

The theoretical construction shows a dichotomy between model-based planning algorithms vs (model-based or model-free) policy optimization algorithms. When the approximability gap occurs, any deep RL algorithms with policies parameterized by neural networks will suffer from a sub-optimal performance. These algorithms include both model-free algorithms such as DQN (Mnih et al., 2015) and SAC (Haarnoja et al., 2018), and model-based policy optimization algorithms such as SLBO (Luo et al., 2019) and MBPO (Janner et al., 2019). To validate the intuition, we empirically apply these algorithms to the constructed or the randomly generated MDPs. Indeed, they fail to converge to the optimal rewards even with sufficient samples, which suggests that they suffer from the lack of expressivity.

On the other hand, model-based planning algorithms should not suffer from the lack of expressivity, because they only use the learned, parameterized dynamics, which are easy to express. In fact, even a partial planner can help improve the expressivity of the policy. If we plan for k steps and then resort to some Q-function for estimating the total reward of the remaining steps, we can obtain a policy with 2^k more pieces than what Q-function has. (Theorem 4.5)

In summary, our contributions are:

1. We construct continuous state space MDPs whose Q-functions and policies are proved to be more complex than the dynamics (Sections 4.1 and 4.2.)

2. We empirically show that with a decent chance, (semi-) randomly generated piecewise linear MDPs also have complex Q-functions (Section 5.1.)

3. We show theoretically and empirically that the model-free RL or model-based policy optimization algorithms suffer from the lack of expressivity for the constructed MDPs (Sections 5.1), whereas model-based planning solve the problem efficiently (Section 5.2.)

4. Inspired by the theory, we propose a simple model-based bootstrapping planner (BOOTS), which can be applied on top of any model-free or model-based Q-learning algorithms at the test time. Empirical results show that BOOTS improves the performance on MuJoCo benchmark tasks, and outperforms previous state-of-the-art on MuJoCo Humanoid environment. (Section 5.3)

2. Related Work

Comparisons with Prior Theoretical Work. Model-based RL has been extensively studied in the tabular case (see Zanette & Brunskill (2019); Azar et al. (2017) and the references therein), but much less so in the context of deep neural networks approximators and continuous state space. Luo et al. (2019) give sample complexity and convergence guarantees suing principle of optimism in the face of uncertainty for non-linear dynamics.

Below we review several prior results regarding model-based versus model-free dichotomy in various settings. We note that our work focuses on the angle of expressivity, whereas the work below focuses on the sample efficiency.

Tabular MDPs. The extensive study in tabular MDP setting leaves little gap in their sample complexity of model-based and model-free algorithms, whereas the space complexity seems to be the main difference (Strehl et al., 2006). The best sample complexity bounds for model-based tabular RL (Azar et al., 2017; Zanette & Brunskill, 2019) and model-free tabular RL (Jin et al., 2018) only differ by a poly(H) multiplicative factor (where H is the horizon.)

Linear Quadratic Regulator. Dean et al. (2018) and Dean et al. (2017) provided sample complexity bound for model-based LQR. Recently, Tu & Recht (2018) analyzed sample efficiency of the model-based and model-free problem in the setting of Linear Quadratic Regulator, and proved a $O(d)$ gap in sample complexity, where d is the dimension of state space. Unlike tabular MDP case, the space complexity of model-based and model-free algorithms has little difference. The sample-efficiency gap mostly comes from that dynamics learning has d-dimensional supervisions, whereas Q-learning has only one-dimensional supervision.

Contextual Decision Process (with function approximator). Sun et al. (2019) prove an exponential information-

theoretical gap between mode-based and model-free algorithms in the factored MDP setting. Their definition of model-free algorithms requires an exact parameterization: the value-function hypothesis class should be exactly the family of optimal value-functions induced by the MDP family. This limits the application to deep reinforcement learning where over-parameterized neural networks are frequently used. Moreover, a crucial reason for the failure of the model-free algorithms is that the reward is designed to be sparse.

Related Empirical Work. A large family of model-based RL algorithms uses existing model-free algorithms of its variant on the learned dynamics. MBPO (Janner et al., 2019), STEVE (Buckman et al., 2018), and MVE (Feinberg et al., 2018) are model-based Q-learning-based policy optimization algorithms, which can be viewed as modern extensions and improvements of the early model-based Q-learning framework, Dyna (Sutton, 1990). SLBO (Luo et al., 2019) is a model-based policy optimization algorithm using TRPO as the algorithm in the learned environment.

Another way to exploit the dynamics is to use it to perform model-based planning. Racanière et al. (2017) and Du & Narasimhan (2019) use the model to generated additional extra data to do planning implicitly. Chua et al. (2018) study how to combine an ensemble of probabilistic models and planning, which is followed by Wang & Ba (2019), which introduces a policy network to distill knowledge from a planner and provides a prior for the planner. Piché et al. (2018) uses methods in Sequential Monte Carlo in the context of control as inference. Oh et al. (2017) trains a Q-function and then perform lookahead planning. Nagabandi et al. (2018) uses random shooting as the planning algorithm.

Heess et al. (2015) backprops through a stochastic computation graph with a stochastic gradient to optimize the policy under the learned dynamics. Levine & Koltun (2013) distills a policy from trajectory optimization. Rajeswaran et al. (2016) trains a policy adversarially robust to the worst dynamics in the ensemble. Clavera et al. (2018) reformulates the problem as a meta-learning problem and using meta-learning algorithms. Predictron (Silver et al., 2017) learns a dynamics and value function and then use them to predict the future reward sequences.

Another line of work focus on how to improve the learned dynamics model. Many of them use an ensemble of models (Kurutach et al., 2018; Rajeswaran et al., 2016; Clavera et al., 2018), which are further extended to an ensemble of probabilistic models (Chua et al., 2018; Wang & Ba, 2019). Luo et al. (2019) designs a discrepancy bound for learning the dynamics model. Talvitie (2014) augments the data for model training in a way that the model can output a real observation from its own prediction. Malik et al. (2019) calibrates the model's uncertainty so that the model's output distribution should match the frequency of predicted states. Oh et al. (2017) learns a representation of states by predicting rewards and future returns using representation.

3. Preliminaries

Markov Decision Process. A Markov Decision Process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, f, r, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $f : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ the transition dynamics that maps a state action pair to a probability distribution of the next state, γ the discount factor, and $r \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ the reward function. Throughout this paper, we will consider deterministic dynamics, which, with slight abuse of notation, will be denoted by $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$.

A deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$ maps a state to an action. The value function for the policy is defined as is defined $V^\pi(s) \stackrel{\text{def}}{=} \sum_{h=1}^{\infty} \gamma^{h-1} r(s_h, a_h)$. where $a_h = \pi(s_h), s_1 = s$ and $s_{h+1} = f(s_h, a_h)$.

An RL agent aims to find a policy π that maximizes the expected total reward defined as

$$\eta(\pi) \stackrel{\text{def}}{=} \mathbb{E}_{s_1 \sim \mu} \left[V^\pi(s_1) \right],$$

where μ is the distribution of the initial state.

Bellman Equation. Let $\pi^\star$ be the optimal policy, and $V^\star$ the optimal value function (that is, the value function for policy $\pi^\star$). The value function V^π for policy π and optimal value function $V^\star$ satisfy the Bellman equation and Bellman optimality equation, respectively. Let Q^π and $Q^\star$ defines the state-action value function for policy π and optimal state-action value function. Then, for a deterministic dynamics f, we have

$$\begin{cases} V^\pi(s) = Q^\pi(s, \pi(s)), \\ Q^\pi(s,a) = r(s,a) + \gamma V^\pi(f(s,a)), \end{cases} \quad \begin{cases} V^\star(s) = \max_{a \in \mathcal{A}} Q^\star(s,a), \\ Q^\star(s,a) = r(s,a) + \gamma V^\star(f(s,a)). \end{cases} \tag{1}$$

Denote the Bellman operator for dynamics f by $\mathcal{B}_f$: $(\mathcal{B}_f[Q])(s,a) = r(s,a) + \max_{a'} Q(f(s,a), a')$.

Neural Networks. We focus on fully-connected neural nets with ReLU function as activations. A one-dimensional input and one-dimensional output ReLU neural net represents a piecewise linear function. A two-layer ReLU neural net with d hidden neurons represents a piecewise linear function with at most $(d+1)$ pieces. Similarly, a H-layer neural net with d hidden neurons in each layer represents a piecewise linear function with at most $(d+1)^H$ pieces (Pascanu et al., 2013).

Problem Setting and Notations. In this paper, we focus on continuous state space, discrete action space MDPs with $\mathcal{S} \subset \mathbb{R}$. We assume the dynamics is deterministic (that is,

$s_{t+1} = f(s_t, a_t)$), and the reward is known to the agent. Let $\lfloor x \rfloor$ denote the floor function of x, that is, the greatest integer less than or equal to x. We use $\mathbb{I}[\cdot]$ to denote the indicator function.

4. Approximability of Q-functions and Dynamics

We show that there exist MDPs in one-dimensional continuous state space that have simple dynamics but complex Q-functions and policies. Moreover, any polynomial-size neural networks function approximator of the Q-function or policy will result in a sub-optimal expected total reward, and learning Q-functions parameterized by neural networks requires fundamentally an exponential number of samples (Section 4.2). In Section 4.3, we show that the expressivity issue can be alleviated by model-based planning in the test time.

4.1. A provable construction of MDPs with complex Q

Recall that we consider the infinite horizon case and $0 < \gamma < 1$ is the discount factor. Let $H = (1-\gamma)^{-1}$ be the "effective horizon" — the rewards after $\gg H$ steps becomes negligible due to the discount factor. For simplicity, we assume that $H > 3$ and it is an integer. (Otherwise we take just take $H = \lfloor (1-\gamma)^{-1} \rfloor$.) Throughout this section, we assume that the state space $\mathcal{S} = [0, 1)$ and the action space $\mathcal{A} = \{0, 1\}$.

Definition 4.1. *Given the effective horizon $H = (1-\gamma)^{-1}$, we define an MDP M_H as follows. Let $\kappa = 2^{-H}$. The dynamics f by the following piecewise linear functions with at most three pieces.*

$$f(s,0) = \begin{cases} 2s & \text{if } s < 1/2 \\ 2s-1 & \text{if } s \geq 1/2 \end{cases}$$

$$f(s,1) = \begin{cases} 2s+\kappa & \text{if } s < \frac{1-\kappa}{2} \\ 2s+\kappa-1 & \text{if } \frac{1-\kappa}{2} \leq s \leq \frac{2-\kappa}{2} \\ 2s+\kappa-2 & \text{otherwise.} \end{cases}$$

The reward function is defined as

$$r(s,0) = \mathbb{I}[1/2 \leq s < 1]$$
$$r(s,1) = \mathbb{I}[1/2 \leq s < 1] - 2(\gamma^{H-1} - \gamma^H)$$

The initial state distribution μ is uniform distribution over the state space $[0, 1)$.

The dynamics and the reward function for $H = 4$ are visualized in Figures 2a, 2b. Note that by the definition, the transition function for a fixed action a is a piecewise linear function with at most 3 pieces. Our construction can be modified so that the dynamics is Lipschitz and the same conclusion holds (see Appendix C).

Attentive readers may also realize that the dynamics can be also be written succinctly as $f(s,0) = 2s \mod 1$ and $f(s,1) = 2s + \kappa \mod 1$[2], which are key properties that we use in the proof of Theorem 4.2 below.

Optimal Q-function $Q^\star$ and the optimal policy $\pi^\star$. Even though the dynamics of the MDP constructed in Definition 4.1 has only a constant number of pieces, the Q-function and policy are very complex: (1) the policy is a piecewise linear function with exponentially number of pieces, (2) the optimal Q-function $Q^\star$ and the optimal value function $V^\star$ are actually *fractals* that are not differentiable anywhere. These are formalized in the theorem below.

Theorem 4.2. *For $s \in [0, 1)$, let $s^{(k)}$ denotes the k-th bit of s in the binary representation.*[3] *The optimal policy $\pi^\star$ for the MDP defined in Definition 4.1 has 2^{H+1} number of pieces. In particular,*

$$\pi^\star(s) = \mathbb{I}[s^{(H+1)} = 0]. \tag{2}$$

And the optimal value function is a fractal with the expression:

$$V^\star(s) = \sum_{h=1}^{H} \gamma^{h-1} s^{(h)} + \gamma^{H-1}\left(2s^{(H+1)} - 2\right) + \sum_{h=H+1}^{\infty} \gamma^{h-1}\left(1 + 2(s^{(h+1)} - s^{(h)})\right). \tag{3}$$

The close-form expression of $Q^\star$ can be computed by $Q^\star(s,a) = r(s,a) + V^\star(f(s,a))$, which is also a fractal.

We approximate the optimal Q-function by truncating the infinite sum to $2H$ terms, and visualize it in Figure 2c. We discuss the main intuitions behind the construction in the following proof sketch of the Theorem. A rigorous proof of Theorem 4.2) is deferred to Appendix B.1.

Proof Sketch. The key observation is that the dynamics f essentially shift the binary representation of the states with some addition. We can verify that the dynamics satisfies $f(s,0) = 2s \mod 1$ and $f(s,1) = 2s + \kappa \mod 1$ where $\kappa = 2^{-H}$. In other words, suppose $s = 0.s^{(1)}s^{(2)}\cdots$ is the binary representation of s, and let left-shift$(s) = 0.s^{(2)}s^{(3)}\cdots$.

$$f(s,0) = \text{left-shift}(s)$$
$$f(s,1) = (\text{left-shift}(s) + 2^{-H}) \mod 1$$

Moreover, the reward function is approximately equal to the first bit of the binary representation

$$r(s,0) = s^{(1)}, \quad r(s,1) \approx s^{(1)}$$

[2]The mod function is defined as $x \mod 1 \stackrel{\text{def}}{=} x - \lfloor x \rfloor$.

[3]Or more precisely, we define $s^{(h)} \triangleq \lfloor 2^h s \rfloor \mod 2$

(a) Visualization of dynamics for action $a = 0, 1$.

(b) The reward function $r(s, 0)$ and $r(s, 1)$.

(c) Approximation of optimal Q-function $Q^\star(s, a)$

Figure 2. A visualization of the dynamics, the reward function, and the approximated Q-function of the MDP defined in Definition 4.1, and the approximation of its optimal Q-function for the effective horizon $H = 4$. We can also construct slightly more involved construction with Lipschitz dynamics and very similar properties. Please see Appendix C.

(Here the small negative drift of reward for action $a = 1$, $-2(\gamma^{H-1} - \gamma^H)$, is only mostly designed for the convenience of the proof, and casual readers can ignore it for simplicity.) Ignoring carries, the policy pretty much can only affect the H-th bit of the next state $s' = f(s, a)$: the H-th bit of s' is either equal to $(H + 1)$-th bit of s when action is 0, or equal its flip when action is 1. Because the bits will eventually be shifted left and the reward is higher if the first bit of a future state is 1, towards getting higher future reward, the policy should aim to create more 1's. Therefore, the optimal policy should choose action 0 if the $(H + 1)$-th bit of s is already 1, and otherwise choose to flip the $(H + 1)$-th bit by taking action 1.

A more delicate calculation that addresses the carries properly would lead us to the form of the optimal policy (Equation (2).) Computing the total reward by executing the optimal policy will lead us to the form of the optimal value function (equation (3).) (This step does require some elementary but sophisticated algebraic manipulation.)

With the form of the $V^\star$, a shortcut to a formal, rigorous proof would be to verify that it satisfies the Bellman equation, and verify $\pi^\star$ is consistent with it. We follow this route in the formal proof of Theorem 4.2) in Appendix B.1. □

4.2. The Approximability of Q-function

A priori, the complexity of $Q^\star$ or $\pi^\star$ does not rule out the possibility that there exists an approximation of them that do an equally good job in terms of maximizing the rewards. However, we show that in this section, indeed, there is no neural network approximation of $Q^\star$ or $\pi^\star$ with a polynomial width. We prove this by showing any piecewise linear function with a sub-exponential number of pieces cannot approximate either $Q^\star$ or $\pi^\star$ with a near-optimal total reward.

Theorem 4.3. *Let M_H be the MDP constructed in Definition 4.1. Suppose a piecewise linear policy π has a near optimal reward in the sense that $\eta(\pi) \geq 0.92 \cdot \eta(\pi^\star)$, then it has to have at least $\Omega\left(\exp(cH)/H\right)$ pieces for some universal constant $c > 0$. As a corollary, no constant depth neural networks with polynomial width (in H) can approximate the optimal policy with near optimal rewards.*

Consider a policy π induced by a value function Q, that is, $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a)$. Then,when there are two actions, the number of pieces of the policy is bounded by twice the number of pieces of Q. This observation and the theorem above implies the following inapproximability result of $Q^\star$.

Corollary 4.4. *In the setting of Theorem 4.3, let π be the policy induced by some Q. If π is near-optimal in a sense that $\eta(\pi) \geq 0.92 \cdot \eta(\pi^\star)$, then Q has at least $\Omega\left(\exp(cH)/H\right)$ pieces for some universal constant $c > 0$.*

The intuition behind the proof of Theorem 4.3 is as follows. Recall that the optimal policy has the form $\pi^\star(s) = \mathbb{I}[s^{(H+1)} = 0]$. One can expect that any polynomial-pieces policy π behaves suboptimally in most of the states, which leads to the suboptimality of π. Detailed proof of Theorem 4.3 is deferred to Appendix B.2.

Beyond the expressivity lower bound, we also provide an exponential sample complexity lower bound for Q-learning algorithms parameterized with neural networks (see Appendix B.4).

4.3. Approximability of model-based planning

When the Q-function or the policy are too complex to be approximated by a reasonable size neural network, both model-free algorithms or model-based policy optimization algorithms will suffer from the lack of expressivity, and as a consequence, the sub-optimal rewards. However, model-based planning algorithms will not suffer from the lack of

expressivity because the final policy is not represented by a neural network.

Given a function Q that are potentially not expressive enough for approximating the optimal Q-function, we can simply apply the Bellman operator with a learned dynamics $\hat{f}$ for k times to get a bootstrapped version of Q:

$$\mathcal{B}^k_{\hat{f}}[Q](s,a) = \max_{a_1,\cdots,a_k} \left(\sum_{h=0}^{k-1} r(s_h,a_h) + Q(s_k,a_k) \right)$$

where $s_0 = s, a_0 = a$ and $s_{h+1} = \hat{f}(s_h, a_h)$.

Given the bootstrapped Q, we can derive a greedy policy w.r.t it:

$$\pi^{\text{boots}}_{k,Q,\hat{f}}(s) = \max_a \mathcal{B}^k_{\hat{f}}[Q](s,a) \tag{4}$$

The following theorem shows that for the MDPs constructed in Section 4.1, using $\mathcal{B}^k_{\hat{f}}[Q]$ to represent the optimal Q-function requires fewer pieces in Q than representing the optimal Q-function with Q directly.

Theorem 4.5. *Consider the MDP M_H defined in Definition 4.1. There exists a constant-piece piecewise linear dynamics $\hat{f}$ and 2^{H-k+1}-piece piecewise linear function Q, such that the bootstrapped policy $\pi^{\text{boots}}_{k,Q,\hat{f}}(s)$ achieves the optimal total rewards.*

By contrast, recall that in Theorem 4.3, we show that approximating the optimal Q function directly with a piecewise linear function, it requires $\approx 2^H$ piecewise. Thus we have a multiplicative factor of 2^k gain in the expressivity by using the k-step bootstrapped policy. Here the exponential gain is only magnificent enough when k is close to H because the gap of approximability is huge. However, in more realistic settings — the randomly-generated MDPs and the MuJoCo environment — the bootstrapping planner improve the performance significantly. Proof of Theorem 4.5 is deferred to Appendix B.6.

The model-based planning can also be viewed as an implicit parameterization of Q-function. In the grid world environment, Tamar et al. (2016) parameterize the Q-function by the dynamics. For environments with larger state space, we can also use the dynamics in the parameterization of Q-function by model-based planning. A naive implementation of the bootstrapped policy (such as enumerating trajectories) would require 2^k-times running time. However we can use approximate algorithms for the trajectory optimization step in Eq. (4) such as Monte Carlo tree search (MCTS) and cross entropy method (CEM).

5. Experiments

In this section we provide empirical results that supports our theory. We validate our theory with randomly generated MDPs with one dimensional state space (Section 5.1). Sections 5.2 and 5.3 shows that model-based planning indeed helps to improve the performance on both toy and real environments.

5.1. The Approximability of Q-functions of randomly generated MDPs

In this section, we show the phenomena that the Q-function not only occurs in the crafted cases as in the previous subsection, but also occurs more robustly with a decent chance for (semi-) randomly generated MDPs. (Mathematically, this says that the family of MDPs with such a property is not a degenerate measure-zero set.)

It is challenging and perhaps requires deep math to characterize the fractal structure of Q-functions for random dynamics, which is beyond the scope of this paper. Instead, we take an empirical approach here. We generate random piecewise linear and Lipschitz dynamics, and compute their Q-functions for the finite horizon, and then visualize the Q-functions or count the number of pieces in the Q-functions. We also use DQN algorithm (Mnih et al., 2015) with a finite-size neural network to learn the Q-function.

We set horizon $H = 10$ for simplicity and computational feasibility. The state and action space are $[0, 1)$ and $\{0, 1\}$ respectively. We design two methods to generate random or semi-random piecewise dynamics with at most four pieces. First, we have a uniformly random method, called RAND, where we independently generate two piecewise linear functions for $f(s, 0)$ and $f(s, 1)$, by generating random positions for the kinks, generating random outputs for the kinks, and connecting the kinks by linear lines (See Appendix D.1 for a detailed description.)

In the second method, called SEMI-RAND, we introduce a bit more structure in the generation process, towards increasing the chance to see the phenomenon. The functions $f(s, 0)$ and $f(s, 1)$ have 3 pieces with shared kinks. We also design the generating process of the outputs at the kinks so that the functions have more fluctuations. The reward for both of the two methods is $r(s, a) = s, \forall a \in \mathcal{A}$. (See Appendix D.1 for a detailed description.)

Figure 1 illustrates the dynamics of the generated MDPs from SEMI-RAND. More details of empirical settings can be found in Appendix D.1.

The optimal policy and Q can have a large number of pieces. Because the state space has one dimension, and the horizon is 10, we can compute the exact Q-functions by recursively applying Bellman operators, and count the number of pieces. We found that, 8.6% fraction of the 1000 MDPs independently generated from the RAND method has policies with more than 100 pieces, much larger than the number of pieces in the dynamics (which is 4). Using the

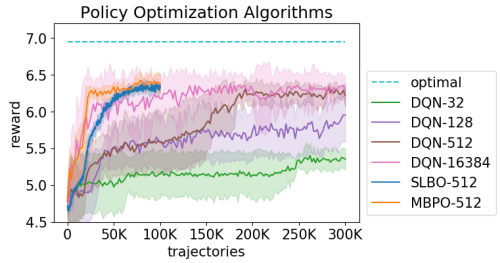

Figure 3. The performance of DQN, SLBO, and MBPO on the bottom dynamics in Figure 1. The number after the acronym is the width of the neural network used in the parameterization of Q. We see that even with sufficiently large neural networks and sufficiently many steps, these algorithms still suffers from bad approximability and cannot achieve optimal reward.

SEMI-RAND method, a 68.7% fraction of the MDPs has polices with more than 10^3 pieces. In Section D.1, we plot the histogram of the number of pieces of the Q-functions. Figure 1 visualize the Q-functions and dynamics of two MDPs generated from RAND and SEMI-RAND method. These results suggest that the phenomenon that Q-function is more complex than dynamics is degenerate phenomenon and can occur with non-zero measure. For more empirical results, see Appendix D.2.

Model-based policy optimization methods also suffer from a lack of expressivity. As an implication of our theory in the previous section, when the Q-function or the policy are too complex to be approximated by a reasonable size neural network, both model-free algorithms or model-based policy optimization algorithms will suffer from the lack of expressivity, and as a consequence, the sub-optimal rewards. We verify this claim on the randomly generated MDPs discussed in Section 5.1, by running DQN (Mnih et al., 2015), SLBO (Luo et al., 2019), and MBPO (Janner et al., 2019) with various architecture size.

For the ease of exposition, we use the MDP visualized in the bottom half of Figure 1. The optimal policy for this specific MDP has 765 pieces, and the optimal Q-function has about 4×10^4 number of pieces, and we can compute the optimal total rewards.

First, we apply DQN to this environment by using a two-layer neural network with various widths to parameterize the Q-function. The training curve is shown in Figure 3. Model-free algorithms can not find near-optimal policy even with 2^{14} hidden neurons and 1M trajectories, which suggests that there is a fundamental approximation issue. This result is consistent with Fu et al. (2019), in a sense that enlarging Q-network improves the performance of DQN algorithm at convergence.

Second, we apply SLBO and MBPO in the same environment. Because the policy network and Q-function in SLBO and MBPO cannot approximate the optimal policy and value function, we see that they fail to achieve near-optimal rewards, as shown in Figure 3.

Algorithm 1 Model-based Bootstrapping Planner (BOOTS) + RL Algorithm X

training: run Algorithm X, store the all samples in the set R, store the learned Q-function Q, and the learned dynamics $\hat{f}$ if it is available in Algorithm X.
testing:
  if $\hat{f}$ is not available, learn $\hat{f}$ from the data in R
  execute the policy BOOTS(s) at every state s

function BOOTS(s)
  Given: query oracle for function Q and $\hat{f}$
  Compute

$$\pi^{\text{boots}}_{k,Q,\hat{f}}(s) = \arg\max_a \max_{a_1,\cdots,a_k} r(s,a) + \cdots + r(s_{k-1},a_{k-1}) + Q(s_k,a_k) \quad (5)$$

  using a zero-th order optimization algorithm (which only requires oracle query of the function value) such as cross-entropy method or random shooting

5.2. Model-based planning on randomly generated MDPs

We implement, that planning with the learned dynamics (with an exponential-time algorithm which enumerates all the possible future sequence of actions), as well as bootstrapping with partial planner with varying planning horizon. A simple k-step model-based bootstrapping planner is applied on top of existing Q-functions (trained from either model-based or model-free approach). The bootstrapping planner is reminiscent of MCTS using in AlphaGo (Silver et al., 2016; 2018). However, here, we use the learned dynamics and deal with continuous state space. Algorithm 1, called BOOTS, summarizes how to apply the planner on top of any RL algorithm with a Q-function (straightforwardly).

Note that the planner is only used *in the test time* for a fair comparison. The dynamics used by the planner is learned using the data collected when training the Q-function. As shown in Figure 5, the model-based planning algorithm not only has the bests sample-efficiency, but also achieves the optimal reward. In the meantime, even a partial planner helps to improve both the sample-efficiency and performance. More details of this experiment are deferred to Appendix D.3.

5.3. Model-based planning on MuJoCo environments

We work with the OpenAI Gym environments (Brockman et al., 2016) based on the MuJoCo simulator (Todorov et al., 2012). We apply BOOTS on top of three algorithms: (a)

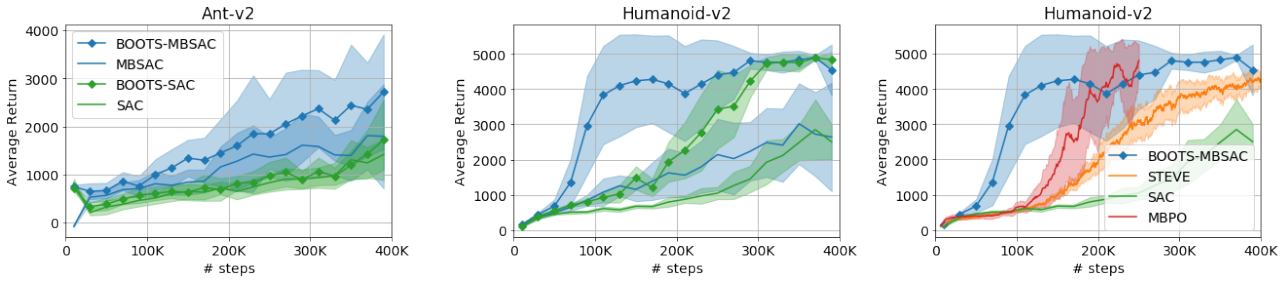

Figure 4. **Left two:** Comparison of BOOTS-MBSAC vs MBSAC and BOOTS-SAC vs SAC on Ant and Humanoid environments. Particularly on the Humanoid environment, BOOTS improves the performance significantly. The test policy for MBSAC and SAC are the deterministic policy that takes the mean of the output of the policy network. **Right:** BOOTS-MBSAC significantly outperforms previous state-of-the-art algorithms on Humanoid.

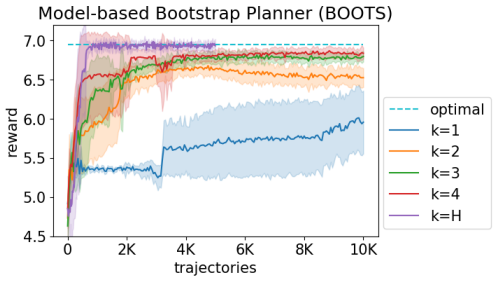

Figure 5. Performance of BOOTS-DQN with various planning steps. A near-optimal reward is achieved with even $k = 3$, indicating that the bootstrapping with the learned dynamics improves the expressivity of the policy significantly.

SAC (Haarnoja et al., 2018), the state-of-the-art model-free RL algorithm; (b) a computationally efficient variant of MBPO (Janner et al., 2019) that we developed using ideas from SLBO (Luo et al., 2019), which is called **MBSAC**, see Appendix A for details; (c) MBPO (Janner et al., 2019), the previous state-of-the-art model-based RL algorithm.

We use $k = 4$ steps of planning throughout the experiments in this section. We mainly compare BOOTS-SAC with SAC, and BOOTS-MBSAC with MBSAC, to demonstrate that BOOTS can be used on top of existing strong baselines. See Figure 4 for the comparison on Gym Ant and Humanoid environments. We also found that BOOTS has little help for other simpler environments as observed in Clavera et al. (2020), and we suspect that those environments have much less complex Q-functions so that our theory and intuitions do not apply.

We also compare BOOTS-MBSAC with other other model-based and model-free algorithms on the Humanoid environment (Figure 4). We see a strong performance surpassing the previous state-of-the-art MBPO. For Ant environment, because our implementation MBSAC is significantly weaker than MBPO, even with the boost from BOOTS, still BOOTS-MBSAC is far behind MBPO. [4]

6. Conclusion

Our study suggests that there exists a significant representation power gap of neural networks between for expressing Q-function, the policy, and the dynamics in both constructed examples and empirical benchmarking environments. We show that our model-based bootstrapping planner BOOTS helps to overcome the approximation issue and improves the performance in synthetic settings and in the difficult MuJoCo environments. We raise some interesting open questions.

- Can we theoretically generalize our results to high-dimensional state space, or continuous actions space? Can we theoretically analyze the number of pieces of the optimal Q-function of a stochastic dynamics?
- In this paper, we measure the complexity by the size of the neural networks. It's conceivable that for real-life problems, the complexity of a neural network can be better measured by its weights norm. Could we build a more realistic theory with another measure of complexity?
- The BOOTS planner comes with a cost of longer test time. How do we efficiently plan in high-dimensional dynamics with a long planning horizon?
- The dynamics can also be more complex (perhaps in another sense) than the Q-function in certain cases. How do we efficiently identify the complexity of the optimal Q-function, policy, and the dynamics, and how do we deploy the best algorithms for problems with different characteristics?

[4] For STEVE, we use the official code at https://github.com/tensorflow/models/tree/master/research/steve

Acknowledgements

We thank Yuanhao Wang and Zhizhou Ren for helpful comments on the earlier version of this paper. Toyota Research Institute ("TRI") provided funds to assist the authors with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity. The work is in part supported by SDSI and SAIL. T. M is also supported in part by Lam Research and Google Faculty Award. K. D is supported in part by the Tsinghua Academic Fund Undergraduate Overseas Studies. Y. L is supported by NSF, ONR, Simons Foundation, Schmidt Foundation, Amazon Research, DARPA and SRC.

References

Azar, M. G., Osband, I., and Munos, R. Minimax regret bounds for reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 263–272. JMLR. org, 2017.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, pp. 8224–8234, 2018.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pp. 4754–4765, 2018.

Clavera, I., Rothfuss, J., Schulman, J., Fujita, Y., Asfour, T., and Abbeel, P. Model-based reinforcement learning via meta-policy optimization. In *Conference on Robot Learning*, pp. 617–629, 2018.

Clavera, I., Fu, Y., and Abbeel, P. Model-augmented actor-critic: Backpropagating through paths. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Skln2A4YDB.

Dean, S., Mania, H., Matni, N., Recht, B., and Tu, S. On the sample complexity of the linear quadratic regulator. *CoRR*, abs/1710.01688, 2017. URL http://arxiv.org/abs/1710.01688.

Dean, S., Mania, H., Matni, N., Recht, B., and Tu, S. Regret bounds for robust adaptive control of the linear quadratic regulator. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pp. 4192–4201, 2018.

Du, Y. and Narasimhan, K. Task-agnostic dynamics priors for deep reinforcement learning. *arXiv preprint arXiv:1905.04819*, 2019.

Feinberg, V., Wan, A., Stoica, I., Jordan, M., Gonzalez, J., and Levine, S. Model-based value expansion for efficient model-free reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018.

Fu, J., Kumar, A., Soh, M., and Levine, S. Diagnosing bottlenecks in deep q-learning algorithms. In *International Conference on Machine Learning*, pp. 2021–2030, 2019.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pp. 1856–1865, 2018.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, Y. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pp. 2944–2952, 2015.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. *ArXiv*, abs/1906.08253, 2019.

Jin, C., Allen-Zhu, Z., Bubeck, S., and Jordan, M. I. Is q-learning provably efficient? In *Advances in Neural Information Processing Systems*, pp. 4863–4873, 2018.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Sepassi, R., Tucker, G., and Michalewski, H. Model-based reinforcement learning for atari. *ArXiv*, abs/1903.00374, 2019.

Kakade, S. and Langford, J. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 267–274. Morgan Kaufmann Publishers Inc., 2002.

Koller, D. and Parr, R. Computing factored value functions for policies in structured mdps. In *IJCAI*, volume 99, pp. 1332–1339, 1999.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

Levine, S. and Koltun, V. Guided policy search. In *International Conference on Machine Learning*, pp. 1–9, 2013.

Luo, Y., Xu, H., Li, Y., Tian, Y., Darrell, T., and Ma, T. Algorithmic framework for model-based deep reinforcement

learning with theoretical guarantees. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

Malik, A., Kuleshov, V., Song, J., Nemer, D., Seymour, H., and Ermon, S. Calibrated model-based deep reinforcement learning. *arXiv preprint arXiv:1906.08312*, 2019.

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

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566. IEEE, 2018.

Oh, J., Singh, S., and Lee, H. Value prediction network. In *Advances in Neural Information Processing Systems*, pp. 6118–6128, 2017.

Pascanu, R., Montufar, G. F., and Bengio, Y. On the number of inference regions of deep feed forward networks with piece-wise linear activations. 2013.

Piché, A., Thomas, V., Ibrahim, C., Bengio, Y., and Pal, C. Probabilistic planning with sequential monte carlo methods. 2018.

Racanière, S., Weber, T., Reichert, D., Buesing, L., Guez, A., Rezende, D. J., Badia, A. P., Vinyals, O., Heess, N., Li, Y., et al. Imagination-augmented agents for deep reinforcement learning. In *Advances in neural information processing systems*, pp. 5690–5701, 2017.

Rajeswaran, A., Ghotra, S., Ravindran, B., and Levine, S. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*, 2016.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., et al. The predictron: End-to-end learning and planning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3191–3199. JMLR. org, 2017.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. ISSN 0036-8075. doi: 10.1126/science.aar6404.

Strehl, A. L., Li, L., Wiewiora, E., Langford, J., and Littman, M. L. Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pp. 881–888. ACM, 2006.

Sun, W., Jiang, N., Krishnamurthy, A., Agarwal, A., and Langford, J. Model-based rl in contextual decision processes: Pac bounds and exponential improvements over model-free approaches. In *Conference on Learning Theory*, pp. 2898–2933, 2019.

Sutton, R. S. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bulletin*, 2:160–163, 1990.

Talvitie, E. Model regularization for stable sample rollouts. In *UAI*, pp. 780–789, 2014.

Tamar, A., Wu, Y. B., Thomas, G., Levine, S., and Abbeel, P. Value iteration networks. In *IJCAI*, 2016.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Tu, S. and Recht, B. The gap between model-based and model-free methods on the linear quadratic regulator: An asymptotic viewpoint. *arXiv preprint arXiv:1812.03565*, 2018.

Wang, T. and Ba, J. Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649*, 2019.

Zanette, A. and Brunskill, E. Tighter problem-dependent regret bounds in reinforcement learning without domain knowledge using value function bounds. In *International Conference on Machine Learning*, pp. 7304–7312, 2019.

Zhang, H., Dauphin, Y. N., and Ma, T. Fixup initialization: Residual learning without normalization. *arXiv preprint arXiv:1901.09321*, 2019.