FREQUENCY-BASED SEARCH-CONTROL IN DYNA

Anonymous authors
Paper under double-blind review

ABSTRACT

Model-based reinforcement learning has been empirically demonstrated as a successful strategy to improve sample efficiency. Particularly, Dyna architecture, as an elegant model-based architecture integrating learning and planning, provides huge flexibility of using a model. One of the most important components in Dyna is called search-control, which refers to the process of generating state or state-action pairs from which we query the model to acquire simulated experiences. Search-control is critical to improve learning efficiency. In this work, we propose a simple and novel search-control strategy by searching high frequency region on value function. Our main intuition is built on Shannon sampling theorem from signal processing, which indicates that a high frequency signal requires more samples to reconstruct. We empirically show that a high frequency function is more difficult to approximate. This suggests a search-control strategy: we should use states in high frequency region of the value function to query the model to acquire more samples. We develop a simple strategy to locally measure the frequency of a function by gradient norm, and provide theoretical justification for this approach. We then apply our strategy to search-control in Dyna, and conduct experiments to show its property and effectiveness on benchmark domains.

1 INTRODUCTION

Model-based reinforcement learning (MBRL) (Lin, 1992; Sutton, 1991b; Todd et al., 2012; Sutton & Barto, 2018) methods have been successfully applied to many benchmark domains (Gu et al., 2016; Ha & Schmidhuber, 2018; Kaiser et al., 2019). One of the most classical MBRL architecture is Dyna (please see Algorithm 2 in Appendix A.3 for pseudo-code) proposed by Sutton (1991a), which integrates model-free and model-based policy updates in an online RL setting. At each time step, an agent uses the real experiences to learn a model and to perform model-free policy update; during the *planning* stage, simulated experiences are acquired from the model to further improve the policy. One closely related method in model-free learning setting is experience replay (ER) (Lin, 1992; Adam & Busoniu, 2012), which utilizes a buffer to store experiences and at each time step, those experiences are randomly sampled to update policy. Though ER can be thought of as a simplified form of MBRL (van Seijen & Sutton, 2015), a model provides great flexibility of acquiring simulated experiences. Sutton & Barto (2018) uses the term search-control to describe the mechanism of selecting state or state-action pairs to query the model to generate simulated experiences. We call the corresponding data structure to store those state or state-action pairs *search-control queue*. Search-control is of vital importance in Dyna, as it can largely affect sample efficiency. For example, in deterministic environment, if we simply sample visited state-action pair for search-control, the next state and reward should be exactly the same as those in ER buffer. Such naive search-control is unlikely to outperform ER as a model can suffer to model error (Talvitie, 2014; 2017). Prioritized sweeping (Moore & Atkeson, 1993) is designed to speed up value iteration process: the simulated transitions are updated based on the absolute temporal difference error. Such methods have been adapted to continuous domains with function approximation by Sutton et al. (2008); Pan et al. (2018); Corneil et al. (2018). Alternative strategies are explored by Gu et al. (2016) and Pan et al. (2019), the former utilizes local linear model to general optimal trajectories through iLQR (Li & Todorov, 2004) and the latter acquire states by hill climbing on value function estimates.

All of those search-control methods are derived from a RL perspective, in the sense that they attempt to acquire simulated experiences which have either high temporal difference error or high returns. In this paper, we propose an alternative perspective to view search-control strategy. Consider the

following problem. Given a perfect model, which state-action pairs we should use to query the model? A natural answer is that those states whose values are potentially difficult to estimate should be queried more often. Shannon sampling theorem establishes the connection between a signal's bandwidth limit and number of samples required to reconstruct such signal. It is later applied to learning theory by Smale et al. (2004); Smale & Zhou (2005); Jiang (2019). Based on this insight, we establish connections between a function's local frequency and gradient norm. Then, based on the hill climbing approach developed by Pan et al. (2019), we are able to adapt our approach to search-control in Dyna.

In this paper, we first review some basic background in MBRL. Afterwards, we review some concepts in signal processing and conduct experiments in supervised learning to show that a high frequency function is more difficult to approximate and providing more samples for that function can improve learning efficiency. We then propose a method to locally measure the frequency of a point in function's domain and provide theoretical proof for our method. By hill climbing approach Pan et al. (2019), we adapt our method to search-control in Dyna. We conduct experiments on both benchmark and challenging domains to illustrate the properties and utilities of our method.

2 BACKGROUND

Reinforcement learning (RL) problems are typically formulated as Markov Decision Processes (MDPs) (Sutton & Barto, 2018; Szepesvári, 2010). An MDP $(\mathcal{S}, \mathcal{A}, \mathbb{P}, R, \gamma)$ is determined by state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $\mathbb{P}$, reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$, and discount factor $\gamma \in [0, 1]$. At each step t, an agent observes a state $s_t \in \mathcal{S}$, and takes an action $a_t \in \mathcal{A}$. The environment receives a_t, and transits to next state $s_{t+1} \sim \mathbb{P}(\cdot|s_t, a_t)$. The agent receives a reward scalar $r_{t+1} \in \mathbb{R}$ generated by reward function $r_{t+1} = R(s_t, a_t, s_{t+1})$. The agent maintains a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ to choose actions. For a given state-action pair (s, a), the action-value function of policy π is defined as $Q_\pi(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a; A_{t+1:\infty} \sim \pi]$ where $G_t \stackrel{\text{def}}{=} \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})$ is the return of $s_0, a_0, s_1, a_1, ...$ following policy π and transition $\mathbb{P}$. Value-based RL methods learn action value function (Watkins & Dayan, 1992), and act greedily using action values to take actions. Policy-based RL methods perform gradient update of parameters to learn policies with high expected rewards Sutton et al. (1999). Both value- and policy-based RL methods can be easily adopted in Dyna framework.

Model-based RL. In general, a model is considered as some mapping taking a state-action as input and outputs some projection into the future. A model can be local (Tassa et al., 2012; Gu et al., 2016) or global (Ha & Schmidhuber, 2018; Pan et al., 2018), deterministic (Sutton et al., 2008) or stochastic (Deisenroth & Rasmussen, 2011; Ha & Schmidhuber, 2018), feature-to-feature (Corneil et al., 2018; Ha & Schmidhuber, 2018) or observation-to-observation (Gu et al., 2016; Pan et al., 2018; Kaiser et al., 2019), one-step (Gu et al., 2016; Pan et al., 2018), or multi-step (Sorg & Singh, 2010; Oh et al., 2017), or decision-aware (Joseph et al., 2013; Farahmand et al., 2017; Silver et al., 2017). Modelling the environment dynamics through RKHS (reproducing kernel Hilbert space) embedding has been also studied (Grunewalder et al., 2012), where the bellman operator is approximated in RKHS and hence boostrap target for value updating can be directly acquired.

In this work, for simplicity, we consider one-step environment dynamics model, which takes a state-action pair as input and returns the next state and reward. However, notice that our search-control method can be naturally adapted to different types of models. The most relevant works to ours include the classical Dyna (Sutton, 1991a;b) and the variant of Dyna (Pan et al., 2019) (please see Algorithm 3 in Appendix A.3 for pseudo-code). The latter designs a hill climbing method on value estimates for search-control. We now review the key steps of this algorithm, which are helpful to understand our algorithm. At each time step, ER buffer is maintained, then a state is randomly sampled from the buffer and is used as the initial state to perform hill climbing on the learned value function. Specifically, a natural gradient ascent method[1] is developed and states along the gradient ascent trajectories are stored into search-control queue. During planning stage, states are sampled from the queue and are paired with on-policy actions (i.e. actions taken according to current Q network), then the model is queried to get next states and rewards. Those simulated transitions are mixed with

[1] According to the original paper, natural gradient is used to guarantee a certain level of coordinate invariant property, so it can handle state variables with vastly different numerical scales.

samples from ER buffer to train neural networks. The intuition behind such search-control is that a value-based agent tends to guide the agent to move towards high value region; by taking gradient ascent on value function, it foresees possible future states. With a model, simulated experiences along those states can be used to quickly correct value function during planning stage (Pan et al., 2019). In this work, we suggest to consider search-control from a supervised learning perspective and propose a novel question to ask: which part of state space needs more samples? It is natural to think of that those states whose values are potentially *difficult to learn* should be queried more often.

3 UNDERSTANDING THE DIFFICULTY OF FUNCTION APPROXIMATION

Using experiments of regression tasks, we illustrate that high frequency region of a function is difficult to approximate, and it is worthwhile for the learning algorithm to assign more training data to those regions. To make this insight practically useful, we propose gradient norm as a criterion to measure the local frequency information of a function around a point. We formally establish theoretical connection between our proposed criterion and the local frequency of function, which lays the foundation of our frequency-based search-control method in next section.

3.1 WHAT TYPE OF FUNCTION IS DIFFICULT TO APPROXIMATE

Consider the ℓ_2 regression problem. Given a training set $D = \{(x_i, y_i)\}_{i=1:n}$, our goal is to learn an unknown target function by empirical risk minimization. Formally, we aim to solve

$$f = \arg\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} (f(x_i) - y_i)^2,$$

where $\mathcal{H}$ is some hypothesis space. Our question is what kind of regions in the domain of the underlying true mapping $f^*(x) = \mathbb{E}[Y|X = x]$ are difficult to approximate. Then we can assign more training data to those regions to help learning.

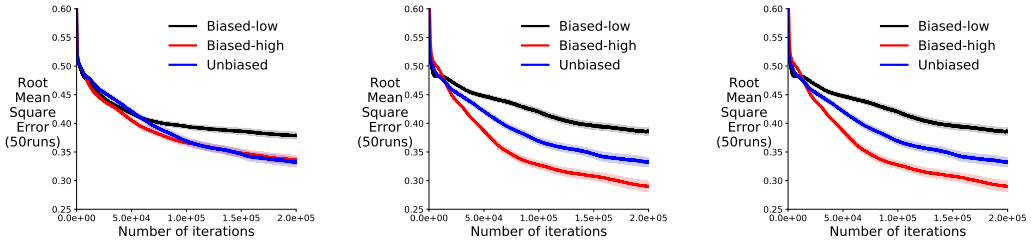
In signal processing field, with sampling rate[2] f_{sample}, perfect reconstruction is guaranteed for any signal with highest frequency $< \frac{f_{\text{sample}}}{2}$. This is well known as Nyquist-Shannon sampling theorem (Zayed, 1993). Sampling theory has been applied in sample efficiency analysis of machine learning algorithms (Smale et al., 2004; Smale & Zhou, 2005; Jiang, 2019). In general, difficulty of learning increases as the bandwidth limit (highest frequency) of target function increases. Although problems are different in machine learning, the sampling theorem provides a high-level intuition for us: regions with more high frequency signals require more learning data. To make this high-level intuition concrete, we consider the following function:

$$f_{\sin}(x) = \begin{cases} \sin(8\pi x) & x \in [-2, 0), \\ \sin(\pi x) & x \in [0, 2]. \end{cases} \tag{1}$$

It is easy to check that the regions $[-2, 0)$ and $[0, 2]$ contain signals with frequency ratio $8 : 1$. By the intuition, $[-2, 0)$ would require more training data than $[0, 2]$. Given the same amount of training data, and the same machine learning algorithm, we should expect assigning more proportion of training data on $[-2, 0)$ to perform better than uniformly or oppositely putting training data ways.

A demonstration experiment. To empirically verify the intuition, we conduct a simple linear regression task, with $f_{\sin}$ as the target function. Our expectation is that, since the high frequency region $[-2, 0)$ is more difficult to learn, it requires more training data. The training set $D = \{(x_i, y_i)\}_{i=1:n}$ is generated by sampling $x \in [-2, 2]$, and adding Gaussian noise $N(0, \sigma^2)$ on Eq. (1), where the standard deviation is set to be $\sigma = 0.1$. We present ℓ_2 regression learning curves of training datasets with different biased sampling ratios $p_b \in \{60\%, 70\%, 80\%\}$, as shown in Fig. 1 (a)-(c). We observe that biased training data sampling ratios towards high frequency region (more samples in high frequency region) clearly speed up learning. This is consistent with the intuitive insight and it provides a heuristic that assigning more data to high frequency regions leads to better learning results.

[2]Sampling rate refers to number of samples per second used to reconstruct continuous signals.



(a) Bias level 60% (b) Bias level 70% (c) Bias level 80%

Figure 1: Learning curves are testing error as functions of mini-batch update number. Bias level 60% means 60% of the training data are from the high frequency region $[-2, 0)$ and is labeled as **Biased-high**. Similarly, **Biased-low** means 60% of the training data are from the low frequency region $[0, 2]$. We include unbiased training dataset as a reference (**Unbiased**). The total numbers of training data are the same across all experiments. The testing set is unbiased and the results are averaged over 50 random seeds with the shade as standard error.

3.2 Identifying high frequency regions of a function

In the above experiment, since each region only contains signal with one constant frequency, it is quite easy to identify high frequency region of $f_{\sin}$. However, in practice, targets will not be in that nice and artificial form. We are facing two main difficulties to identify high frequency regions. *On one hand*, in machine learning problems, we have no access to the underlying target functions. Actually, only function approximation, such as trained neural network, is available. *On the other hand*, frequency is more a global property rather than local information. Every single point within the domain has impact on the frequency of a function. And it is unpractical, if not impossible, to do that global calculation over the whole function domain. To make the high frequency heuristic practically useful, we need some simple criteria that: (a) use function approximation; (b) can be efficiently calculated; (c) characterize local frequency information. Inspired by the function $f_{\sin}$ in Eq. (1), a natural idea is to calculate the first order $f'(x) \overset{\text{def}}{=} \frac{df(x)}{dx}$ or second order derivative $f''(x) \overset{\text{def}}{=} \frac{d^2 f(x)}{dx^2}$ because they both satisfy (a) and (b). To do "sanity check" for property (c), consider the following illustrative examples.

Example 1. *For $f_{\sin}$ defined in Eq.* (1)*, calculate the integrals of squared first order derivative $f'_{\sin}$ on high frequency region $[-2, 0)$ and low frequency region $[0, 2]$, respectively:*

$$\int_{-2}^{0} \left[f'_{\sin}(x)\right]^2 dx = 64\pi^2, \quad \int_{0}^{2} \left[f'_{\sin}(x)\right]^2 dx = \pi^2.$$

Example 2. *Let $f : [-\pi, \pi] \to \mathbb{R}$ be a real valued function. We have*

$$\int_{-\pi}^{\pi} \left[f'(x)\right]^2 dx = \pi \cdot \sum_{n=1}^{\infty} n^2 \left(a_n^2 + b_n^2\right), \quad \int_{-\pi}^{\pi} \left[f''(x)\right]^2 dx = \pi \cdot \sum_{n=1}^{\infty} n^4 \left(a_n^2 + b_n^2\right).$$

$a_n, b_n \in \mathbb{R}$, $n = 1, 2, \ldots$ are Fourier coefficients of frequency $\frac{n}{2\pi}$, defined as

$$a_n \overset{\text{def}}{=} \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(nx) dx, \quad b_n \overset{\text{def}}{=} \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(nx) dx.$$

Example 1 shows that the integral of squared first order derivative ratio is $64 : 1$ (the frequency ratio is $8 : 1$). And the region with large gradient norm is high frequency region. Example 2 generalizes the calculation of Example 1, indicating that for one dimensional real valued functions, integrals of squared gradient norm and Hessian norm are closely related to frequency information. In particular, Hessian norm emphasize more on high frequency information (comparing n^4 factor with n^2).

Empirical demonstration. Our calculation in the above examples implies that regions with large gradient and Hessian norm correspond to high frequency regions. Based on the same spirit of the

l_2 regression task in Section 3.2, we empirically verify this insight. Our expectation is that biased training dataset towards high gradient norm and Hessian norm would achieve better learning results. In Fig. 2(a), **Biased-GradientNorm** is uniformly sampling $x \in [-2, 2]$ for 60% of training data, and sampling proportional to gradient norm (i.e., $p(x) \propto |f'_{\sin}(x)|$) for the remaining 40%; while **Biased-HessianNorm** is sampling proportional to Hessian norm (i.e., $p(x) \propto |f''_{\sin}(x)|$) for the remaining 40% of training data. As shown in Fig. 2(b)(c), sampling according to gradient norm or Hessian norm indeed leads to denser point distribution in high frequency region $[-2, 0)$. And in Fig. 2(a), we observe that such biased training datasets provide fast learning, similar to high frequency biased training datasets in Fig. 1. In particular, **Biased-HessianNorm** learns faster than **Biased-GradientNorm**, which is consistent with our calculation in Example 2.

As a result of passing "sanity check" of calculations and experiments, given a function $f : \mathcal{X} \mapsto \mathcal{Y}$ and a point $x \in \mathcal{X}$, we propose to measure frequency of f around a small neighborhood of x (we call this *local frequency*) using the following function: $g(x) \stackrel{\text{def}}{=} \|\nabla_x f(x)\| + \|H_f(x)\|$, where $\|\nabla_x f(x)\|$ is gradient norm at x, and $\|H_f(x)\|$ is norm of Hessian matrix at x. We claim that local frequency of f around x is proportional to $g(x)$.[3] We theoretically justify this claim. For real-valued functions in Euclidean spaces, our theory establishes a connection between local gradient norm, local function energy [4], and local frequency distribution. The proof can be found in Appendix A.2.

Theorem 1. *Given any function $f : \mathbb{R}^n \mapsto \mathbb{R}$, for any frequency vector $k \in \mathbb{R}^n$, define its local Fourier coefficient of $x \in \mathbb{R}^n$,*

$$\hat{f}(k) \stackrel{\text{def}}{=} \int_{\|y-x\| \le 1} f(y) \exp\left\{-2\pi i \cdot y^\top k\right\} dy,$$

for local function around x, i.e., $\{y : \|y - x\| \le 1\}$. Assume the local function "energy" is finite,

$$\int_{\|y-x\| \le 1} [f(y)]^2 \, dy = \int_{\mathbb{R}^n} \|\hat{f}(k)\|^2 dk < \infty, \quad \forall x \in \mathbb{R}^n. \tag{2}$$

We have $\forall x \in \mathbb{R}^n$,

$$\int_{\|y-x\| \le 1} \|\nabla f(y)\|^2 \, dy = 4\pi^2 \cdot \left[\int_{\|y-x\| \le 1} [f(y)]^2 \, dy\right] \cdot \left[\int_{\mathbb{R}^n} \pi_{\hat{f}}(k) \cdot \|k\|^2 \, dk\right], \tag{3}$$

where

$$\pi_{\hat{f}}(k) \stackrel{\text{def}}{=} \frac{\|\hat{f}(k)\|^2}{\int_{\mathbb{R}^n} \|\hat{f}(k)\|^2 dk}, \quad \forall k \in \mathbb{R}^n. \tag{4}$$

$\pi_{\hat{f}}$ is called "local frequency distribution" of $f(x)$. $\pi_{\hat{f}}$ is a probability distribution over $\mathbb{R}^n$, i.e.,

$$\int_{k \in \mathbb{R}^n} \pi_{\hat{f}}(k) dk = 1, \text{ and } \pi_{\hat{f}}(k) \ge 0, \quad \forall k \in \mathbb{R}^n.$$

Remark 1. *We use a distribution $\pi_{\hat{f}}$ in Eq. (4) to characterize local frequency behaviour for reasons. **First**, comparing frequencies of regions is more naturally captured by a distribution than one single scalar, since signals usually are within a range of frequencies. **Second**, to eliminate the impact of the function energy Eq. (2), we normalize the Fourier coefficient $\hat{f}$ to get $\pi_{\hat{f}}$.*

Remark 2. *For a frequency vector $k \in \mathbb{R}^n$, the larger its norm $\|k\|$, the higher its frequency. Given any x and its local function (i.e., $f(\cdot)$ around x), $\pi_{\hat{f}}(k)$ is the proportion/percentage that frequency k occupies. Therefore, the integral of $\pi_{\hat{f}}(k) \cdot \|k\|^2$ can perfectly reflect local frequency behaviour.*

Remark 3. *Consider f as a value function in reinforcement learning setting. Theorem 1 indicates that regions with large gradient norm can either have large absolute value function, or high local frequency, or both. To prevent finding regions only have large negative value function, our theory implies that it is reasonable to take both gradient norm and value function into account, as our proposed method does in next section.*

[3] Our formal derivation justifies the first order connection, while it can be naturally extended to the second order.

[4] We consider the notion of energy in signal processing terminology: the energy of a continuous time signal $x(t)$ is defined as $\int x(t)^2 dt$. In our theory, function f is the signal.

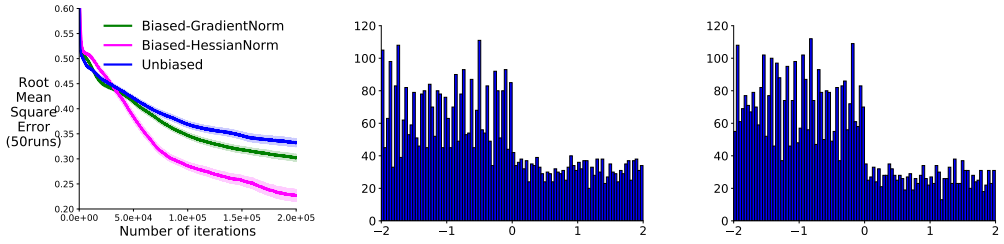
Discussion with Uncertainty Principle. The Uncertainty Principle says that a function cannot be too concentrated in both spatial and frequency space, i.e.,

$$\left[\int_{\|y-x\|\leq 1}(y-x)^2\cdot[f(y)]^2\,dy\right]\cdot\left[\int_{\mathbb{R}^n}\|\hat{f}(k)\|^2\cdot\|k\|^2\,dk\right]\geq\frac{1}{16\pi^2}. \tag{5}$$

Combing Eq. (5) with our Eq. (3),

$$\left[\int_{\|y-x\|\leq 1}(y-x)^2\cdot[f(y)]^2\,dy\right]\cdot\left[\int_{\|y-x\|\leq 1}\|\nabla f(y)\|^2\,dy\right]\geq\frac{1}{4},$$

which means the more concentrated f is locally around x, the larger local gradient norm must be. On the other hand, if local gradient norm is small, then f cannot be too concentrated around x.



(a) RMSE vs. number of iterations (b) count vs. x (c) count vs. x

Figure 2: We show the learning curve of l_2 regression on three training data sets in Figure (a) and show the distribution of the two biased training data points in (b)(c) respectively. The total number of training data points are the same across all experiments. The testing set is unbiased and the results are averaged over 50 random seeds with the shadow indicating standard error.

4 Frequency-based search-control in Dyna

In this section, we present the Dyna architecture with our frequency-based search-control in algorithm 1. Notice that we omit the technical details such as preconditioning and noisy gradient as introduced by Pan et al. (2019) for the hill climbing process. We refer readers to the Appendix A.5 for implementation details.

We want to query a model with states in high frequency region. To search states in high frequency region, we can do hill climbing on $g(s)=||\nabla_s V(s)||+||H_v(s)||$. However, Theorem 1 suggests that states with large gradient norm can either have large absolute value, or high local frequency, or both. The trap we want to avoid is finding negative value states with large magnitude, as those states may be rarely visited under optimal policy. A natural strategy to get around this problem is to combine our hill climbing method with the previous hill climbing on value function (Pan et al., 2019), as the latter tends to generate high value states. We propose the following method for combination. At each time step, with certain probability, we perform hill climbing on either

$$s\leftarrow s+\alpha\nabla_s V(s) \tag{6}$$

or

$$s\leftarrow s+\alpha\nabla_s g(s) \tag{7}$$

and store states along the gradient trajectory in search-control queue. When hill climbing on the value function, we sample initial state from the ER buffer as suggested by the previous work (Pan et al., 2019). When hill climbing on $g(s)$, we use the initial state sampled from the search-control queue. This way ensures that the initial state for searching high frequency region has relatively high value; then hill climbing on $\nabla g(s)$ should encourage to move forward to high frequency region, by Theorem 1. We discuss other intuitive choices which we tested in the Appendix A.4.

Similar to the previous work Pan et al. (2019), we obtain the state-value function in both 7 and 6 by taking the maximum action-value, i.e. $V(s)=\max_a Q(s,a)\approx\max_a Q_\theta(s,a)$ where θ is the

Algorithm 1 Dyna architecture with Frequency-based search-control

B: the ER buffer, B_s: search-control queue
$\mathcal{M} : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \times \mathbb{R}$, the model outputs next state and reward
m: number of states we want to fetch by hill climbing, d: number of planning steps
ϵ_a: the threshold for accepting a state
Q, Q': current and target Q networks, respectively
b: the mini-batch size, $\beta \in (0, 1)$: the proportion of simulated samples in a mini-batch
τ: update target network Q' every τ updates to Q
$t \leftarrow 0$ is the time step, n_τ is the number of parameter updates
while true **do**
  Observe s_t, take action a_t (i.e. ϵ-greedy w.r.t. Q)
  Observe s_{t+1}, r_{t+1}, add $(s_t, a_t, s_{t+1}, r_{t+1})$ to B
  // Gradient ascent hill climbing
  With probability $p, 1-p$, choose hill climbing rule 7 or 6 respectively;
  sample s from B_s if choose rule 7, or from B otherwise; set $c \leftarrow 0, \tilde{s} \leftarrow s$
  while $c < m$ **do**
    update s by executing the chosen hill climbing rule
    if s is out of boundary **then**: // resample the initial state and hill climbing rule
      With probability $p, 1-p$, choose hill climbing rule 7 or 6 respectively;
      sample s from B_s if choose 7, or from B otherwise; set $c \leftarrow 0, \tilde{s} \leftarrow s$
      continue
    if $||s - \tilde{s}||_2 / \sqrt{n} > \epsilon_a$ **then**: // n is the state dimension, i.e. $\mathcal{S} \subset \mathbb{R}^n$
      add s to B_s, $\tilde{s} \leftarrow s, c \leftarrow c + 1$
  for d times **do** // d planning updates: sample d mini-batches
    sample βb states from B_s and pair them with on-policy actions, and query $\mathcal{M}$ to get next states and rewards
    sample $b(1-\beta)$ transitions from B an stack these with the simulated transitions
    use the mixed mini-batch for parameter (i.e. DQN) update
    $n_\tau \leftarrow n_\tau + 1$
    if $mod(n_\tau, \tau) == 0$ **then**:
      $Q' \leftarrow Q$
  $t \leftarrow t + 1$

parameter in Q network. As an analogy to Dyna architecture as shown in 2, during planning stage, we sampled multiple mixed mini-batches to update the parameters (i.e. we call multiple planning steps/updates). The mixed mini-batch was also used in the work by Gu et al. (2016) and has the effect of alleviating off-policy sampling issue as studied by Pan et al. (2019).

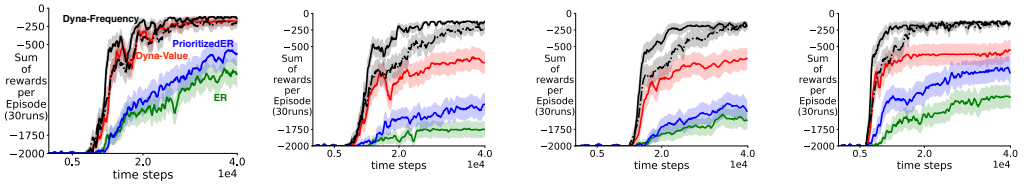
5 EXPERIMENTS

In the experiments, we first carefully study the properties of our algorithm on the benchmark MountainCar domain. Then we illustrate the utility of our algorithm on a challenging self-designed Maze-GridWorld domain, by which we illustrate the practical implication of high frequency region.

We are able to fix on using the same set of parameters across all experiments: learning rate 0.001, target network moving rate 1000, gradient ascent step size (in search-control) 0.01, mixing rate $\beta = 0.5$ and $m = 20$, i.e. at each environment time step we fetch 20 states by hill climbing. We fix $p = 0.5$ across all experiments, hence the hill climbing rule 7 and 6 are chosen with equal chance. We used natural projected gradient ascent for hill climbing as introduced by Pan et al. (2019). Though we mainly focuses on search-control instead of how to learn a model, we indeed include the result of using an online learned model for our algorithm. We refer readers to the Appendix A.5 for reproducible research.

5.1 UTILITY OF FREQUENCY-BASED SEARCH-CONTROL

MountainCar (Brockman et al., 2016) domain is well-studied, and it is known that the value function under the optimal value function has sharp changing regions (Sutton & Barto, 2018), which should



(a) plan steps 10, $\sigma = 0$ (b) plan steps 10, $\sigma = 0.1$ (c) plan steps 30, $\sigma = 0$ (d) plan steps 30, $\sigma = 0.1$

Figure 3: Evaluation curves (sum of episodic reward v.s. environment time steps) of **Dyna-Value**, **PrioritizedER**, **Dyna-Frequency**, **ER** on MountainCar with different number of planning updates with different reward noise variance. Notice that the **dashed** line denotes the evaluation curve of our algorithm with an online learned model. At each time step, the reward is sampled from the Gaussian distribution $N(-1, \sigma^2), \sigma \in \{0.0, 0.1\}$. $\sigma = 0$ indicates the original deterministic reward. All results are averaged over 30 random seeds.

be beneficial for our algorithm. The agent needs to learn to reach the goal state within as few steps as possible. At each step, the agent receives reward -1. The purposes of experiments on this domain are: 1) verify that our search-control strategy can outperform several natural competitors under different number of planning updates; 2) show that our search-control strategy is robust to environment noise.

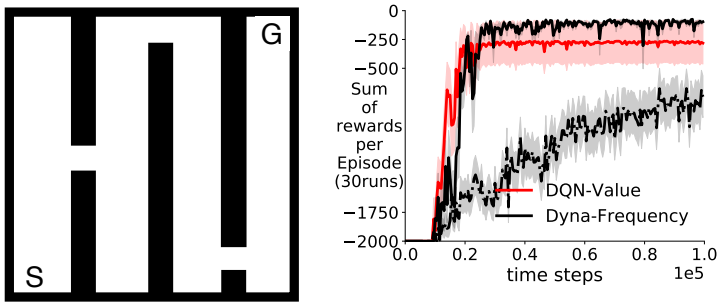
We use the following intuitive competitors. **Dyna-Frequency** is Dyna with our search-control introduced in algorithm 1; **Dyna-Value** is using algorithm 3 from the previous work by Pan et al. (2019); **PrioritizedER** is DQN with prioritized experience replay Schaul et al. (2016); **ER** is simply DQN with experience replay (ER) (Mnih et al., 2015). Figure 3 shows the learning curves of all those algorithms using 10 planning updates (a)(b) and 30 planning updates (c)(d) under different stochasticity. In Figure 3(b)(d), we add zero mean Gaussion noise to the original reward (i.e. $-1 + X, X \sim N(0, \sigma^2)$).

Important observations are as following. 1) With increased number of planning updates, algorithms do not necessarily perform better, as shown in Figure 3(c). However, our algorithm appears to gain more through more number of updates since the difference between **Dyna-Frequency** and **Dyna-Value** seems to be clearer in Figure 3(c) than in Figure 3(a). 2) Since both **Dyna-Value** and our algorithm fetch exactly the same number of states (i.e. $m = 20$) by search-control, the superior performance of our algorithm indicates the advantage of using high frequency samples. 3) **PrioritizedER** does clearly worse than our algorithm and **Dyna-Value**, which probably implies the utility of the generalization power of the value function to acquire additional samples. 4) our algorithm maintains superior performance in the presence of noise. One reason is that, noisy perturbation basically leads to more "energy" in all frequencies. When we take derivative, those high frequency terms are amplified. Hence, even with perturbation, high frequency region remains.

5.2 A Case Study: MazeGridWorld

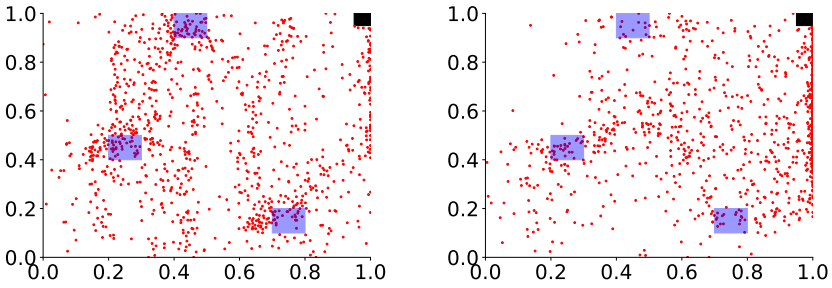
We now illustrate the utility of our method on a challenging MazeGridWorld domain as shown in Figure 4(a). The domain has continuous state space $\mathcal{S} = [0, 1]^2$ and four discrete actions $\{up, down, left, right\}$. There are three walls in the middle, each of which has hole for the agent to go through. Each episode starts from the left bottom and ends at right top and the agent gets reward -1 at each time step, hence the agent should learn to use as few steps as possible to reach the goal area. Model-free methods completely fail on this domain, and we mainly study our algorithm and the **Dyna-Value** algorithm.

Figure 4(b) shows the evaluation curves of the two algorithms. An important difference between our algorithm and the previous work is in the variance of the evaluation curve, which implies a robust policy learned by our method. In Figure 5, we further investigate the state distribution in search-control queues of the two algorithms by uniformly sampling 1000 states from the two queues. Notice that a very important difference between the two distributions is that our search-control queue has a clearly high density around the *bottleneck* area (i.e. the hole areas where the agent can go across the walls). Learning a stable policy around such area is extremely important: the agent simply fails to reach the goal state if they cannot pass any one of the holes. This distinguishes our algorithm with the previous work, which appears to acquire states near the goal area.



(a) MazeGridWorld (b) Episodic reward vs. time steps

Figure 4: (a) is a visualization of the MazeGridWorld domain. (b) shows evaluation curves of **Dyna-Value** and **Dyna-Frequency**. **Dashed** line indicates using an online learned model of our algorithm. All results are averaged over 30 random seeds.



(a) frequency-based search-control states (b) value-based search-control states

Figure 5: The state distribution in the search-control queue of our algorithm **Dyna-Frequency** (a) and **Dyna-Value** (b) at $50,000$ environment time step. The blue shadow indicates the hole area where the agent can go through the wall. The black box on the right top is the goal area.

6 Discussion

In this work, we motivate and study a new category of methods for search-control by considering the approximation difficulty of a function. We provide a method for identifying the high frequency region of a function's domain with theoretical justification. Experiments are conducted to illuminate our theory. We incorporate our method into Dyna, and empirically investigate its utility. We achieve competitive learning performances on difficult domain. There are several promising future directions. First, it is worth exploring the combination between different search-control strategies. Second, we may borrow methods from active learning (Settles, 2010; Hanneke, 2014), which concerns about using as few samples as possible to learn. The main obstacle of applying active learning to search-control may be the computation efficiency. For example, the methods based on model uncertainty reduction (Lewis & Gale, 1994; Seung et al., 1992; Settles et al., 2008) which typically requires to iterate over the whole dataset. It is interesting to explore how to use those active learning methods for search-control in MBRL algorithms.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, and et al. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015. Software available from tensorflow.org.

S Adam and L Busoniu. Experience Replay for Real-Time Reinforcement Learning Control. *Systems*, 2012.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. arXiv:1606.01540, 2016.

Tzuu-Shuh Chiang, Chii-Ruey Hwang, and Shuenn Jyi Sheu. Diffusion for global optimization in $\mathbb{R}^n$. *SIAM Journal on Control and Optimization*, pp. 737–753, 1987.

Dane S. Corneil, Wulfram Gerstner, and Johanni Brea. Efficient model-based deep reinforcement learning with variational state tabulation. *ICML*, pp. 1049–1058, 2018.

M Deisenroth and C E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning*, 2011.

Amir-Massoud Farahmand, Andre Barreto, and Daniel Nikovski. Value-Aware Loss Function for Model-based Reinforcement Learning. 54:1486–1494, 20–22 Apr 2017.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2010.

Steffen Grunewalder, Guy Lever, Luca Baldassarre, Massi Pontil, and Arthur Gretton. Modelling transition dynamics in MDPs with RKHS embeddings. In *International Conference on Machine Learning*, 2012.

Shixiang Gu, Timothy P. Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous Deep Q-Learning with Model-based Acceleration. In *ICML*, pp. 2829–2838, 2016.

David Ha and Jürgen Schmidhuber. World models. *CoRR*, abs/1803.10122, 2018.

Steve Hanneke. Theory of disagreement-based active learning. *Foundations and Trends in Machine Learning*, 7(2-3):131–309, 2014.

Hui Jiang. A new perspective on machine learning: How to do perfect supervised learning. volume abs/1901.02046, 2019.

Joshua Joseph, Alborz Geramifard, John W Roberts, Jonathan P How, and Nicholas Roy. Reinforcement learning with misspecified model classes. In *ICRA*, pp. 939–946. IEEE, 2013.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari. *CoRR*, abs/1903.00374, 2019.

David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. *CoRR*, 1994.

Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. 2004.

Long-Ji Lin. Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching. *Machine Learning*, 1992.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, and et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, pp. 103–130, 1993.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *NeurIPS*, pp. 6118–6128. Curran Associates, Inc., 2017.

Yangchen Pan, Muhammad Zaheer, Adam White, Andrew Patterson, and Martha White. Organizing experience: a deeper look at replay mechanisms for sample-based planning in continuous state domains. In *IJCAI*, pp. 4794–4800, 2018.

Yangchen Pan, Hengshuai Yao, Amir-massoud Farahmand, and Martha White. Hill climbing on value estimates for search-control in dyna. *CoRR*, abs/1906.07791, 2019.

Richard L. Roberts, Gareth O.and Tweedie. Exponential convergence of langevin distributions and their discrete approximations. *Bernoulli*, pp. 341–363, 1996.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. In *ICLR*, 2016.

Burr Settles. Active learning literature survey. 2010.

Burr Settles, Mark Craven, and Soumya Ray. Multiple-instance active learning. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis (eds.), *Advances in Neural Information Processing Systems 20*, pp. 1289–1296. Curran Associates, Inc., 2008.

H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT, pp. 287–294, New York, NY, USA, 1992. ACM.

David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, André M.S. Barreto, and Thomas Degris. The predictron: End-to-end learning and planning. In *ICML*, pp. 3191–3199, 2017.

Steve Smale and Ding-Xuan Zhou. Shannon sampling II: Connections to learning theory. *Applied and Computational Harmonic Analysis*, 19(3):285 – 302, 2005.

Steve Smale, René Thom, and Ding-Xuan Zhou. Shannon sampling and function reconstruction from point values. 2004.

Jonathan Sorg and Satinder Singh. Linear options. pp. 31–38, 2010.

R. S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*. MIT Press, 1999.

Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bulletin*, 2(4):160–163, 1991a.

Richard S. Sutton. Integrated modeling and control based on reinforcement learning and dynamic programming. In *Advances in Neural Information Processing Systems*, 1991b.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

Richard S. Sutton, Csaba Szepesvári, Alborz Geramifard, and Michael Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. In *UAI*, pp. 528–536, 2008.

Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Morgan Claypool Publishers, 2010.

Erik Talvitie. Model regularization for stable sample roll-outs. In *Uncertainty in Artificial Intelligence*, 2014.

Erik Talvitie. Self-Correcting Models for Model-Based Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*, 2017.

Y. Tassa, T. Erez, and E. Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. *International Conference on Intelligent Robots and Systems*, 2012.

Peter M. Todd, Thomas T. Hills, and Trevor W. Robbins. Model-based reinforcement learning as cognitive search: Neurocomputational theories. 2012.

Harm van Seijen and Richard S. Sutton. A deeper look at planning as learning from replay. In *ICML*, pp. 2314–2322, 2015.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *ICML*, pp. 681–688, 2011.

A.I. Zayed. *Advances in Shannon's Sampling Theory*. Taylor & Francis, 1993.

A APPENDIX

We provide calculations for Example 1 and Example 2 in Section A.1; theoretical proof of Theorem 1 in Section A.2. Background of Dyna is reviewed in Section A.3 and additional discussions regarding search-control are provided in Section A.4. Experimental details for reproducing our empirical results are in Section A.5.

A.1 CALCULATIONS FOR EXAMPLE 1 AND EXAMPLE 2

Example 1. For $f_{\sin}$ defined in Eq. (1), calculate the integrals of squared first order derivative $f'_{\sin}$ on high frequency region $[-2, 0)$ and low frequency region $[0, 2]$, respectively:

$$\int_{-2}^{0} \left[f'_{\sin}(x)\right]^2 dx = 64\pi^2, \quad \int_{0}^{2} \left[f'_{\sin}(x)\right]^2 dx = \pi^2.$$

Proof. Taking derivative and integral,

$$\int_{-2}^{0} \left[f'(x)\right]^2 dx = 64\pi^2 \int_{-2}^{0} \left[\cos\left(8\pi x\right)\right]^2 dx = 64\pi^2,$$
$$\int_{0}^{2} \left[f'(x)\right]^2 dx = \pi^2 \int_{0}^{2} \left[\cos\left(\pi x\right)\right]^2 dx = \pi^2.$$

□

Example 2. Let $f : [-\pi, \pi] \to \mathbb{R}$ be a real valued function. We have

$$\int_{-\pi}^{\pi} \left[f'(x)\right]^2 dx = \pi \cdot \sum_{n=1}^{\infty} n^2 \left(a_n^2 + b_n^2\right), \quad \int_{-\pi}^{\pi} \left[f''(x)\right]^2 dx = \pi \cdot \sum_{n=1}^{\infty} n^4 \left(a_n^2 + b_n^2\right).$$

$a_n, b_n \in \mathbb{R}$, $n = 1, 2, \ldots$ are Fourier coefficients of frequency $\frac{n}{2\pi}$, defined as

$$a_n \stackrel{\text{def}}{=} \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos\left(nx\right) dx, \quad b_n \stackrel{\text{def}}{=} \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin\left(nx\right) dx.$$

Proof. The Fourier series of $f(x)$ is

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(nx\right) + \sum_{n=1}^{\infty} b_n \sin\left(nx\right),$$

where $a_0 \stackrel{\text{def}}{=} \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) dx$. Taking derivative of f,

$$f'(x) = \sum_{n=1}^{\infty} \left[-n a_n \sin\left(nx\right)\right] + \sum_{n=1}^{\infty} \left[n b_n \cos\left(nx\right)\right].$$

Taking square of f',

$$\begin{aligned}\left[f'(x)\right]^2 = &\sum_{n=1}^{\infty}\sum_{m=1}^{\infty} \left[nm a_n a_m \sin\left(nx\right)\sin\left(mx\right)\right] - \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} \left[nm a_n b_m \sin\left(nx\right)\cos\left(mx\right)\right] \\ &- \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} \left[mn a_m b_n \sin\left(mx\right)\cos\left(nx\right)\right] + \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} \left[nm b_n b_m \cos\left(nx\right)\cos\left(mx\right)\right].\end{aligned}$$

Taking integral,

$$\begin{aligned}\int_{-\pi}^{\pi} \left[f'(x)\right]^2 dx = &\int_{-\pi}^{\pi} \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} \left[nm a_n a_m \sin\left(nx\right)\sin\left(mx\right)\right] dx - \int_{-\pi}^{\pi} \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} \left[nm a_n b_m \sin\left(nx\right)\cos\left(mx\right)\right] dx \\ &- \int_{-\pi}^{\pi} \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} \left[mn a_m b_n \sin\left(mx\right)\cos\left(nx\right)\right] dx + \int_{-\pi}^{\pi} \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} \left[nm b_n b_m \cos\left(nx\right)\cos\left(mx\right)\right] dx \\ = &\sum_{n=1}^{\infty}\sum_{m=1}^{\infty} \left[nm a_n a_m \pi \delta_{n,m} - 0 - 0 + nm b_n b_m \pi \delta_{n,m}\right] \\ = &\pi \cdot \sum_{n=1}^{\infty} n^2 \left(a_n^2 + b_n^2\right),\end{aligned}$$

where

$$\delta_{n,m} \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } n = m, \\ 0, & \text{otherwise.} \end{cases}$$

Using similar arguments, taking derivative of $f'(x)$,

$$f''(x) = \sum_{n=1}^{\infty} \left[-n^2 a_n \cos(nx)\right] + \sum_{n=1}^{\infty} \left[-n^2 b_n \sin(nx)\right].$$

Taking integral,

$$\int_{-\pi}^{\pi} \left[f''(x)\right]^2 dx = \pi \cdot \sum_{n=1}^{\infty} n^4 \left(a_n^2 + b_n^2\right). \qquad \square$$

A.2 Proof for Theorem 1

Theorem 1. Given any function $f : \mathbb{R}^n \mapsto \mathbb{R}$, for any frequency vector $k \in \mathbb{R}^n$, define its local Fourier coefficient of $x \in \mathbb{R}^n$,

$$\hat{f}(k) \stackrel{\text{def}}{=} \int_{\|y-x\| \le 1} f(y) \exp\left\{-2\pi i \cdot y^\top k\right\} dy,$$

for local function around x, i.e., $\{y : \|y - x\| \le 1\}$. Assume the local function "energy" is finite,

$$\int_{\|y-x\| \le 1} \left[f(y)\right]^2 dy = \int_{\mathbb{R}^n} \|\hat{f}(k)\|^2 dk < \infty, \quad \forall x \in \mathbb{R}^n.$$

We have $\forall x \in \mathbb{R}^n$,

$$\int_{\|y-x\| \le 1} \|\nabla f(y)\|^2 dy = 4\pi^2 \cdot \left[\int_{\|y-x\| \le 1} \left[f(y)\right]^2 dy\right] \cdot \left[\int_{\mathbb{R}^n} \pi_{\hat{f}}(k) \cdot \|k\|^2 dk\right],$$

where

$$\pi_{\hat{f}}(k) \stackrel{\text{def}}{=} \frac{\|\hat{f}(k)\|^2}{\int_{\mathbb{R}^n} \|\hat{f}(k)\|^2 dk}, \quad \forall k \in \mathbb{R}^n.$$

$\pi_{\hat{f}}$ is called "local frequency distribution" of $f(x)$. $\pi_{\hat{f}}$ is a probability distribution over $\mathbb{R}^n$, i.e.,

$$\int_{k \in \mathbb{R}^n} \pi_{\hat{f}}(k) dk = 1, \text{ and } \pi_{\hat{f}}(k) \ge 0, \quad \forall k \in \mathbb{R}^n.$$

Proof. Consider the following function defined locally around x,

$$f_x(y) \stackrel{\text{def}}{=} \begin{cases} f(y), & \text{if } \|y - x\| \le 1, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

The coefficient of frequency vector k in the Fourier series of f_x is

$$\begin{aligned} \hat{f}(k) &\stackrel{\text{def}}{=} \int_{\|y-x\| \le 1} f(y) \exp\left\{-2\pi i \cdot y^\top k\right\} dy \\ &= \int_{\|y-x\| \le 1} f_x(y) \exp\left\{-2\pi i \cdot y^\top k\right\} dy. \end{aligned}$$

And the Fourier series of $f_x(y)$, $\forall y$, such that $\|y - x\| \le 1$, is,

$$f_x(y) = \int_{\mathbb{R}^n} \hat{f}(k) \exp\left\{2\pi i \cdot y^\top k\right\} dk.$$

The gradient is

$$\nabla f(y) = \nabla f_x(y) = \int_{\mathbb{R}^n} \hat{f}(k) \exp\left\{2\pi i \cdot y^\top k\right\} (2\pi i \cdot k) \, dk.$$

To calculate gradient norm, we use complex conjugate,

$$\nabla f^*(y) = \int_{\mathbb{R}^n} \hat{f}^*(k') \exp\left\{-2\pi i \cdot y^\top k'\right\} (-2\pi i \cdot k')\, dk',$$

where

$$\hat{f}^*(k') = \int_{\|y'-x\|\leq 1} f_x(y') \exp\left\{2\pi i \cdot {y'}^\top k'\right\} dy'$$

is the complex conjugate of $\hat{f}(k')$. Therefore,

$$\begin{aligned}\|\nabla f(y)\|^2 &= \langle \nabla f(y), \nabla f^*(y)\rangle \\ &= \int_{\mathbb{R}^n}\int_{\mathbb{R}^n} \hat{f}(k)\hat{f}^*(k') \exp\left\{2\pi i \cdot y^\top (k-k')\right\} \left(4\pi^2 k^\top k'\right) dk dk'.\end{aligned}$$

Taking integral of $\|\nabla f(y)\|^2$,

$$\begin{aligned}&\int_{\|y-x\|\leq 1} \|\nabla f(y)\|^2\, dy \\ &= \int_{\mathbb{R}^n}\int_{\mathbb{R}^n} \hat{f}(k)\hat{f}^*(k') \left[\int_{\|y-x\|\leq 1} \exp\left\{2\pi i \cdot y^\top (k-k')\right\} dy\right] \left(4\pi^2 k^\top k'\right) dk dk' \\ &= \int_{\mathbb{R}^n}\int_{\mathbb{R}^n} \hat{f}(k)\hat{f}^*(k')\delta_{k-k',\mathbf{0}} \left(4\pi^2 k^\top k'\right) dk dk' \\ &= 4\pi^2 \int_{\mathbb{R}^n} \|\hat{f}(k)\|^2 \cdot \|k\|^2\, dk.\end{aligned}$$

Recall the definition of local function "energy" around x,

$$\begin{aligned}\int_{\mathbb{R}^n} \|\hat{f}(k)\|^2 dk &= \int_{\mathbb{R}^n} \left\langle \hat{f}(k), \hat{f}^*(k)\right\rangle dk \\ &= \int_{\|y-x\|\leq 1}\int_{\|y'-x\|\leq 1} f_x(y) f_x(y') \left[\int_{\mathbb{R}^n} \exp\left\{2\pi i \cdot k^\top (y'-y)\right\} dk\right] dy dy' \\ &= \int_{\|y-x\|\leq 1}\int_{\|y'-x\|\leq 1} f_x(y) f_x(y')\delta_{y'-y,\mathbf{0}} dy dy' \\ &= \int_{\|y-x\|\leq 1} f_x^2(y) dy \\ &= \int_{\|y-x\|\leq 1} f^2(y) dy \\ &< \infty. \quad \text{(by assumption)}\end{aligned}$$

The local gradient information is related to local energy and frequency distribution,

$$\int_{\|y-x\|\leq 1} \|\nabla f(y)\|^2\, dy = 4\pi^2 \cdot \left[\int_{\|y-x\|\leq 1} f^2(y) dy\right] \cdot \left[\int_{\mathbb{R}^n} \pi_{\hat{f}}(k) \cdot \|k\|^2\, dk\right],$$

where

$$\pi_{\hat{f}}(k) \stackrel{\text{def}}{=} \frac{\|\hat{f}(k)\|^2}{\int_{\mathbb{R}^n} \|\hat{f}(k)\|^2 dk}, \quad \forall k \in \mathbb{R}^n,$$

is the local frequency distribution. □

A.3 Background in Dyna

In this section, we provide the vanilla Dyna (Sutton, 1991a) in Algorithm 2, and the hill climbing Dyna by Pan et al. (2019) in Algorithm 3.

Algorithm 2 Generic Dyna Architecture: Tabular Setting

Initialize $Q(s,a)$ and model $\mathcal{M}(s,a)$, $\forall(s,a) \in \mathcal{S} \times \mathcal{A}$
while true **do**
  observe s, take action a by ϵ-greedy w.r.t $Q(s,\cdot)$
  execute a, observe reward R and next state s'
  Q-learning update for $Q(s,a)$
  update model $\mathcal{M}(s,a)$ (i.e. by counting)
  store (s,a) into search-control queue
  for i=1:d **do**
    sample $(\tilde{s},\tilde{a})$ from search-control queue
    $(\tilde{s}', \tilde{R}) \leftarrow \mathcal{M}(\tilde{s},\tilde{a})$ // simulated transition
    Q-learning update for $Q(\tilde{s},\tilde{a})$ // planning update

Algorithm 3 HC-Dyna architecture

B_s: search-control queue, B: the experience replay buffer
m: number of states to fetch by search-control
b: the mini-batch size
while true **do**
  Observe s_t, take action a_t (i.e. ϵ-greedy w.r.t. action value function)
  Observe s_{t+1}, r_{t+1}, add $(s_t, a_t, s_{t+1}, r_{t+1})$ to B
  sample s from visited states, i.e. ER buffer B
  // Hill climbing by gradient ascent
  while get less than m states **do**
    $s \leftarrow s + \nabla_s V(s), V(s) = \max_s Q_\theta(s,a)$
    store s into search control queue B_s
  // Planning stage
  for d times **do**
    // sample states from B_s and pair them with on-policy actions, query the model to get next states and rewards
    // mix simulated and real experiences into a mini-batch and use it to update parameters (i.e. DQN update)
  $t \leftarrow t+1$

A.4 A DISCUSSION ON SEARCH-CONTROL DESIGN BASED ON HILL CLIMBING

There are possible ways to combine different hill climbing strategies. Here are some unsuccessful trials. For example, climbing on direct combinations of $V(s)$ and $g(s)$, such as $V(s)+g(s)$, or $V(s)g(s)$, does not work well. These are possible reasons. First, such combination can lead to unstable gradient behaviour, for example, on some domain $\nabla g(s)$ can be huge or it can be zero. Second, such combination can alter the trajectory solely based on either $g(s)$ or $V(s)$, and the effect is unclear. It may lead to state with neither high value or high frequency. Last, and probably the most important, hill climbing on $V(s)$ and on $g(s)$ have fundamentally different insights. The former is based on the intuition that the value information should be propagated from the high value region to low value region, as a result, it requires to store states along the whole trajectory, including those in low value region. This is empirically verified by Pan et al. (2019). However, the latter is based on the insight that the function value in high frequency region is more difficult to approximate and needs more samples, while propagating those information back to low frequency region is not that important. To see why, consider the ideal case where we have a true value target available, which turns the problem into regression; then we simply need more samples in the high frequency region. As a result, this approach does not emphasize on recording states throughout the whole hill climbing strategy (i.e. recording states in the low frequency region could be unnecessary). Consequently, our design choice of getting initial state from search-control queue for frequency-based hill climbing is well principled.

A theoretical interpretation of the search-control queue distribution. Given our search-control queue, it is natural to ask that what would be the state distribution in the queue, as this may be helpful

to develop other search-control strategies. Pan et al. (2019) establishes the connection between the state distribution in the search-control queue filled by value-based hill climbing through Langevin dynamics. The basic idea is to show that the hill climbing process is tracking a discretized version of a stochastic differential equation, whose limiting distribution is Gibbs (Roberts, 1996; Chiang et al., 1987; Welling & Teh, 2011). As a result, the state distribution in search-control queue is approximately $p(s) \propto \exp(V(s))$. Following a similar reasoning line, our search-control queue should approximately be a state distribution: $p(s) \propto \exp(V(s)) + \exp(g(s))$. Notice that, removing the first term leads to a sampling distribution resembles to the one in our supervised learning experiment, where the training data distribution is biased towards gradient norm Fig. 2. It is worth investigating whether there is some solid theoretical connection between sampling distribution $p(x) \propto \exp(g(x))$ and sample complexity.

A.5 Reproducible research

All of our implementations are based on tensorflow with version $1.13.0$ (Abadi et al., 2015). For DQN update, we use Adam optimizer. We use mini-batch size $b = 32$ except on the supervised learning experiment where we use 128. For reinforcement learning experiment, we use buffer size 100k. All activation functions are tanh except the output layer of the Q-value is linear. Except the output layer parameters which were initialized from a uniform distribution $[-0.003, 0.003]$, all other parameters are initialized using Xavier initialization (Glorot & Bengio, 2010). For model learning, we use a 64×64 relu units neural network to predict $s' - s$ given a state-action pair with mini-batch size 128 and learning rate 0.0001.

For the supervised learning experiment shown in Section 3, we use 16×16 tanh units neural network, with learning rate 0.001 for all algorithms. The learning curve is plotted by computing the testing error every 20 iterations. When generating Fig. 2, in order to sample points according to $p(x) \propto |f'(x)|$ or $p(x) \propto |f''(x)|$, we use $10,000$ even spaced points on the domain $[-2, 2]$ and the probabilities are computed by normalization across the 10k points.

The experiment on MountainCar is based on the implementation from OpenAI (Brockman et al., 2016), we use 32×32 tanh layer, with target network moving rate 1000 and learning rate 0.001. Exploration noise is 0.1 without decaying. For all algorithms, we use warm up steps $= 5000$ (i.e. random action is taken in the first 5k time steps).

For the experiment on MazeGridWorld, each wall's width is 0.1 and each hole has height 0.1. The left-top point of the hole in the first wall (counting from left to right) has coordinate $(0.2, 0.5)$; the hole in the second wall has coordinate $(0.4, 1.0)$ and the third one is $0.7, 0.2$. Each action leads to 0.05 unit move perturbed by a Gaussian noise from $N(0, 0.01)$. On this domain, we use 64×64 tanh units for Q networks, and number of search-control samples is set as 50 for both algorithms. As a supplement to the Section 5.2, we also provide the state distribution from ER buffer in Figure 6. One can see that ER buffer has very different state distribution with search-control queue.

A.6 Algorithmic details

We provide the pseudo-code in Algorithm 4 with sufficient details to recreate our experimental results. The hill climbing rules we used is the same as introduced by Pan et al. (2019). Define $v_s \stackrel{\text{def}}{=} \nabla_s \max_a Q(s, a)$, $g_s \stackrel{\text{def}}{=} \nabla_s g(s) = \nabla_s(||v_s||_2^2 + ||\nabla_s v_s||_2^2)$. Note that we use a square norm to ensure numerical stability. Then for value-based search-control, we use

$$s \leftarrow s + \frac{\alpha}{||\hat{\Sigma}_s v_s||} \hat{\Sigma}_s v_s + X_i, X_i \sim N(0, \eta \hat{\Sigma}_s) \tag{8}$$

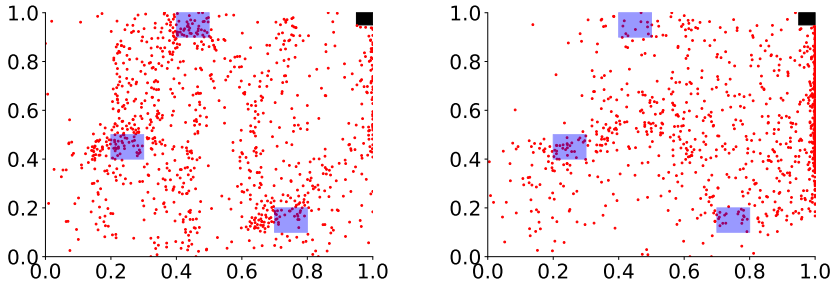
and for frequency-based search-control, we use

$$s \leftarrow s + \frac{\alpha}{||\hat{\Sigma}_s g_s||} \hat{\Sigma}_s g_s + X_i, X_i \sim N(0, \eta \hat{\Sigma}_s) \tag{9}$$

where $\hat{\Sigma}_s$ is empirical covariance matrix estimated from visited states, and we set $\eta = 0.01, \alpha = 0.01$ across all experiments. Notice that comparing with the previous work, we omitted the projection step as we found it is unnecessary in our experiments.

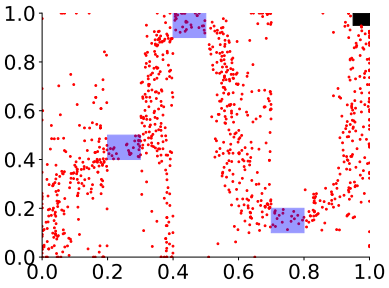
Algorithm 4 Dyna architecture with Frequency-based search-control with additional details

B_s: search-control queue, B: the experience replay buffer
$\mathcal{M} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S} \times \mathbb{R}$, the environment model
m: number of search-control samples to fetch at each step
p: probability of choosing value-based hill climbing rule (we set $p = 0.5$ for all experiments)
$\beta \in [0, 1]$: mixing factor in a mini-batch, i.e. βb samples in a mini-batch are simulated from model
n: number of state variables, i.e. $\mathcal{S} \subset \mathbb{R}^n$
ϵ_a: empirically learned threshold as sample average of $||s_{t+1} - s_t||_2/\sqrt{n}$
d: number of planning steps
Q, Q': current and target Q networks, respectively
b: the mini-batch size
τ: update target network Q' every τ updates to Q
$t \leftarrow 0$ is the time step
$n_\tau \leftarrow 0$ is the number of parameter updates
// Gradient ascent hill climbing
With probability $p, 1 - p$, choose hill climbing Eq. (8) o Eq. (9) respectively;
sample s from B_s if choose rule Eq. (8), or from B otherwise; set $c \leftarrow 0, \tilde{s} \leftarrow s$
while $c < m$ **do**
    update s by executing the chosen hill climbing rule
    if s is out of boundary **then**: // resample the initial state and hill climbing rule
        With probability $p, 1 - p$, choose hill climbing rule Eq. (8) or Eq. (9) respectively;
        sample s from B_s if choose Eq. (7), or from B otherwise; set $c \leftarrow 0, \tilde{s} \leftarrow s$
        continue
    if $||s - \tilde{s}||_2/\sqrt{n} > \epsilon_a$ **then**:
        add s to B_s, $\tilde{s} \leftarrow s, c \leftarrow c + 1$
// d planning updates: sample d mini-batches
for d times **do** // d planning updates
    sample βb states from B_s and pair them with on-policy actions, and query $\mathcal{M}$ to get next states and rewards
    sample $b(1 - \beta)$ transitions from B an stack these with the simulated transitions
    use the mixed mini-batch for parameter (i.e. DQN) update
    $n_\tau \leftarrow n_\tau + 1$
    if $mod(n_\tau, \tau) == 0$ **then**:
        $Q' \leftarrow Q$
$t \leftarrow t + 1$

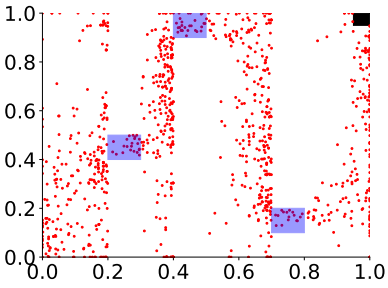


(a) frequency-based search-control states

(b) value-based search-control states



(c) frequency-based ER states



(d) value-based ER states

Figure 6: The state distribution in the search-control queue and ER buffer at $50,000$ environment time step. The blue shadow indicates the hole area where the agent can go through the wall. The black box on the right top is the goal area.