Hill Climbing on Value Estimates for Search-control in Dyna

Yangchen Pan, Hengshuai Yao, Amir-massoud Farahmand, and Martha White

Abstract

Dyna is an architecture for model-based reinforcement learning (RL), where simulated experience from a model is used to update policies or value functions. A key component of Dyna is search-control, the mechanism to generate the state and action from which the agent queries the model, which remains largely unexplored. In this work, we propose to generate such states by using the trajectory obtained from Hill Climbing (HC) the current estimate of the value function. This has the effect of propagating value from high-value regions and of preemptively updating value estimates of the regions that the agent is likely to visit next. We derive a noisy projected natural gradient algorithm for hill climbing, and highlight a connection to Langevin dynamics. We provide an empirical demonstration on four classical domains that our algorithm, HC-Dyna, can obtain significant sample efficiency improvements. We study the properties of different sampling distributions for search-control, and find that there appears to be a benefit specifically from using the samples generated by climbing on current value estimates from low-value to high-value region.

1 Introduction

Experience replay (ER) [Lin, 1992] is currently the most common way to train value functions approximated as neural networks (NNs), in an online RL setting [Adam et al., 2012; Wawrzyński and Tanwani, 2013]. The buffer in ER is typically a recency buffer, storing the most recent transitions, composed of state, action, next state and reward. At each environment time step, the NN gets updated by using a mini-batch of samples from the ER buffer, that is, the agent replays those transitions. ER enables the agent to be more sample efficient, and in fact can be seen as a simple form of model-based RL [van Seijen and Sutton, 2015]. This connection is specific to the Dyna architecture [Sutton, 1990; Sutton, 1991], where the agent maintains a search-control (SC) queue of pairs of states and actions and uses a model to generate next states and rewards. These simulated transitions are used to update values. ER, then, can be seen as a variant of Dyna with a nonparameteric model, where search-control is determined by the observed states and actions.

By moving beyond ER to Dyna with a learned model, we can potentially benefit from increased flexibility in obtaining simulated transitions. Having access to a model allows us to generate unobserved transitions, from a given state-action pair. For example, a model allows the agent to obtain on-policy or exploratory samples from a given state, which has been reported to have advantages [Gu et al., 2016; Pan et al., 2018; Santos et al., 2012; Peng et al., 2018]. More generally, models allow for a variety of choices for search-control, which is critical as it emphasizes different states during the planning phase. Prioritized sweeping [Moore and Atkeson, 1993] uses the model to obtain predecessor states, with states sampled according to the absolute value of temporal difference error. This early work, and more recent work [Sutton et al., 2008; Pan et al., 2018; Corneil et al., 2018], showed this addition significantly outperformed Dyna with states uniformly sampled from observed states. Most of the work on search-control, however, has been limited to sampling visited or predecessor states. Predecessor states require a reverse model, which can be limiting. The range of possibilities has yet to be explored for search-control and there is room for many more ideas.

In this work, we investigate using sampled trajectories by hill climbing on our learned value function to generate states for search-control. Updating along such trajectories has the effect of propagating value from regions the agent currently believes to be high-value. This strategy enables the agent to preemptively update regions where it is likely to visit next. Further, it focuses updates in areas where approximate values are high, and so important to the agent. To obtain such states for search-control, we propose a noisy natural projected gradient algorithm. We show this has a connection to Langevin dynamics, whose distribution converges to the Gibbs distribution, where the density is proportional to the exponential of the state values. We empirically study different sampling distributions for populating the search-control queue, and verify the effectiveness of hill climbing based on estimated values. We conduct experiments showing improved performance in four benchmark domains, as compared to DQN1, and illustrate the usage of our architecture for continuous control.

1We use DQN to refer to the algorithm by [Mnih et al., 2015] that uses ER and target network, but not the exact original architecture.
2 Background

We formalize the environment as a Markov Decision Process (MDP) \((S, A, P, R, \gamma)\), where \(S\) is the state space, \(A\) is the action space, \(P : S \times A \times S \to [0, 1]\) is the transition probabilities, \(R : S \times A \times S \to \mathbb{R}\) is the reward function, and \(\gamma \in [0, 1]\) is the discount factor. At each time step \(t = 1, 2, \ldots\), the agent observes a state \(s_t \in S\) and takes an action \(a_t \in A\), transitions to \(s_{t+1} \sim P(\cdot | s_t, a_t)\) and receives a scalar reward \(r_{t+1} \in \mathbb{R}\) according to the reward function \(R\).

Typically, the goal is to learn a policy to maximize the expected return starting from some fixed initial state. One popular algorithm is Q-learning, by which we can obtain approximate action-values \(Q_\theta : S \times A \to \mathbb{R}\) for parameters \(\theta\). The policy corresponds to acting greedily according to these action-values: for each state, select action \(\arg \max_{a \in A} Q(s, a)\). The Q-learning update for a sampled transition \(s_t, a_t, r_{t+1}, s_{t+1}\) is

\[
\theta \leftarrow \theta + \alpha \delta_t \nabla Q_\theta(s_t, a_t)
\]

where \(\delta_t \equiv r_{t+1} + \max_{a' \in A} Q_\theta(s_{t+1}, a') - Q_\theta(s_t, a_t)\)

Though frequently used, such an update may not be sound with function approximation. Target networks [Mnih et al., 2015] are typically used to improve stability when training NNs, where the bootstrap target on the next step is a fixed, older estimate of the action-values.

ER and Dyna can both be used to improve sample efficiency of DQN. Dyna is a model-based method that simulates (or replays) transitions, to reduce the number of required interactions with the environment. A model is sometimes available a priori (e.g., from physical equations of the dynamics) or is learned using data collected through interacting with the environment. The generic Dyna architecture, with explicit pseudo-code given by [Sutton and Barto, 2018, Chapter 8], can be summarized as follows. When the agent interacts with the real world, it updates both the action-value function and the learned model using the real transition. The agent then performs \(n\) planning steps. In each planning step, the agent samples \((s, \tilde{a})\) from the search-control queue, generates next state \(s'\) and reward \(\tilde{r}\) from \((s, \tilde{a})\) using the model, and updates the action-values using Q-learning with the tuple \((s, \tilde{a}, \tilde{r}, s')\).

3 A Motivating Example

In this section we provide an example of how the value function surface changes during learning on a simple continuous-state GridWorld domain. This provides intuition on why it is useful to populate the search-control queue with states obtained by hill climbing on the estimated value function, as proposed in the next section.

Consider the GridWorld in Figure 1(a), which is a variant of the one introduced by [Peng and Williams, 1993]. In each episode, the agent starts from a uniformly sampled point from the area \([0, 0.05]^2\) and terminates when it reaches the goal area \([0.95, 1.0]^2\). There are four actions \{UP, DOWN, LEFT, RIGHT\}; each leads to a 0.05 unit move towards the corresponding direction. As a cost-to-goal problem, the reward is \(-1\) per step.

In Figure 1, we plot the value function surface after 0, 14k and 20k mini-batch updates to DQN. We visualize the gradient ascent trajectories with 100 gradient steps starting from five states \((0.1, 0.1), (0.9, 0.9), (0.1, 0.9), (0.9, 0.1),\) and \((0.3, 0.4)\). The gradient of the value function used in the gradient ascent is

\[
\nabla_s V(s) = \nabla_s \max_a Q_\theta(s, a),
\]

At the beginning, with a randomly initialized NN, the gradient with respect to state is almost zero, as seen in Figure 1(b). As the DQN agent updates its parameters, the gradient ascent generates trajectories directed towards the goal, though after only 14k steps, these are not yet contiguous, as seen Figure 1(c). After 20k steps, as in Figure 1(d), even though the value function is still inaccurate, the gradient ascent trajectories take all initial states to the goal area. This suggests that as long as the estimated value function roughly reflects the shape of the optimal value function, the trajectories provide a demonstration of how to reach the goal—or high-value regions—and speed up learning by focusing updates on these relevant regions.

More generally, by focusing planning on regions the agent thinks are high-value, it can quickly correct value function estimates before visiting those regions, and so avoid unnecessary interaction. We demonstrate this in Figure 1(e), where the agent obtains gains in performance by updating from high-value states, even when its value estimates have the wrong shape. After 20k learning steps, the values are flipped by negating the sign of the parameters in the output layer of the NN.

HC-Dyna, introduced in Section 5, quickly recovers compared to DQN and OnPolicy updates from the ER buffer. Planning steps help pushing down these erroneously high-values, and the agent can recover much more quickly.

4 Effective Hill Climbing

To generate states for search control, we need an algorithm that can climb on the estimated value function surface. For general value function approximators, such as NNs, this can be difficult. The value function surface can be very flat or very rugged, causing the gradient ascent to get stuck in local optima and hence interrupt the gradient traveling process. Further, the state variables may have very different numerical scales. When using a regular gradient ascent method, it is likely for the state variables with a smaller numerical scale to immediately go out of the state space. Lastly, gradient ascent is unconstrained, potentially generating unrealizable states.

In this section, we propose solutions for all these issues. We provide a noisy invariant projected gradient ascent strategy to generate meaningful trajectories of states for search-control. We then discuss connections to Langevin dynamics, a model for heat diffusion, which provides insight into the sampling distribution of our search-control queue.

4.1 Noisy Natural Projected Gradient Ascent

To address the first issue, of flat or rugged function surfaces, we propose to add Gaussian noise on each gradient ascent step. Intuitively, this provides robustness to flat regions and avoids getting stuck in local maxima on the function surface, by diffusing across the surface to high-value regions.

To address the second issue of vastly different numerical scales among state variables, we use a standard strategy to be
Figure 1: (b-d) The value function on the GridWorld domain with gradient ascent trajectories. (e) shows learning curves (sum of rewards per episode v.s. time steps) where each algorithm needs to recover from a bad NN initialization (i.e. the value function looks like the reverse of (d)).

Figure 2: The search-control queue filled by using (+) or not using (−) natural gradient on MountainCar-v0.

invariant to scale: natural gradient ascent. A popular choice of natural gradient is derived by defining the metric tensor as the Fisher information matrix [Amari and Douglash, 1998; Amari, 1998; Thomas et al., 2016]. We introduce a simple and computationally efficient metric tensor: the inverse of covariance matrix of the states $\Sigma_s^{-1}$. This choice is simple, because the covariance matrix can easily be estimated online.

We can define the following inner product:

$$\langle s, s' \rangle = s^\top \Sigma_s^{-1} s', \forall s, s' \in S,$$

which induces a vector space—the Riemannian manifold—where we can compute the distance of two points $s$ and $s + \Delta$ that are close to each other by $d(s, s + \Delta) \equiv \Delta^\top \Sigma_s^{-1} \Delta$. The steepest ascent updating rule based on this distance metric becomes $s \leftarrow s + \alpha \Sigma_s g$, where $g$ is the gradient vector.

We demonstrate the utility of using the natural gradient scaling. Figure 2 shows the states from the search-control queue filled by hill climbing in early stages of learning (after 8000 steps) on MountainCar. The domain has two state variables with very different numerical scale: position $\in [-1.2, 0.6]$ and velocity $\in [-0.07, 0.07]$. Using a regular gradient update, the queue shows a state distribution with many states concentrated near the top since it is very easy for the velocity variable to go out of boundary. In contrast, the one with natural gradient, shows clear trajectories with an obvious tendency to the right top area (position $\geq 0.5$), which is the goal area.

We use projected gradient updates to address the third issue regarding unrealistic states. We explain the issue and solution using the Acrobat domain. The first two state variables are $\cos \theta, \sin \theta$, where $\theta$ is the angle between the first robot arm’s link and the vector pointing downwards. This induces the restriction that $\cos^2 \theta + \sin^2 \theta = 1$. The hill climbing process could generate many states that do not satisfy this restriction.

This could potentially degrade performance, since the NN needs to generalize to these states unnecessarily. We can use a projection operator $\Pi$ to enforce such restrictions, whenever known, after each gradient ascent step. In Acrobat, $\Pi$ is a simple normalization. In many settings, the constraints are simple box constraints, with projection just inside the boundary.

Now we are ready to introduce our final hill climbing rule:

$$s \leftarrow \Pi (s + \alpha \Sigma_s g + \mathcal{N}),$$

where $\mathcal{N}$ is Gaussian noise and $\alpha$ a stepsize. For simplicity, we set the stepsize to $\alpha = 0.1/\|\Sigma_s g\|$ across all results in this work, though of course there could be better choices.

4.2 Connection to Langevin Dynamics

The proposed hill climbing procedure is similar to Langevin dynamics, which is frequently used as a tool to analyze optimization algorithms or to acquire an estimate of the expected parameter values w.r.t. some posterior distribution in Bayesian learning [Welling and Teh, 2011]. The overdamped Langevin dynamics can be described by a stochastic differential equation (SDE) $dW(t) = \nabla U(W_k) dt + \sqrt{2}dB_t$, where $B_t \in \mathbb{R}^d$ is a $d$-dimensional Brownian motion and $U$ is a continuous differentiable function. Under some conditions, it turns out that the Langevin diffusion $(W_k)_{k \geq 0}$ converges to a unique invariant distribution $p(x) \propto \exp (U(x))$ [Chiang et al., 1987].

By apply the Euler-Maruyama discretization scheme to the SDE, we acquire the discretized version $Y_{k+1} = Y_k + \alpha_{k+1} \nabla U(Y_k) + \sqrt{2\alpha_{k+1}} Z_{k+1}$ where $(Z_k)_{k \geq 1}$ is an i.i.d. sequence of standard $d$-dimensional Gaussian random vectors and $(\alpha_k)_{k \geq 1}$ is a sequence of step sizes. This discretization scheme was used to acquire samples from the original invariant distribution $p(x) \propto \exp (U(x))$ through the Markov chain $(Y_k)_{k \geq 1}$ when it converges to the chain’s stationary distribution [Roberts, 1996]. The distance between the limiting distribution of $(Y_k)_{k \geq 1}$ and the invariant distribution of the underlying SDE has been characterized through various bounds [Durmus and Moulines, 2017].

When we perform hill climbing, the parameter $\theta$ is constant at each time step $t$. By choosing the function $U$ in the SDE above to be equal to $V_\theta$, we see that the state distribution $p(s)$ in our search-control queue is approximately

$$p(s) \propto \exp (V_\theta(s)).$$

Different assumptions on $(\alpha_k)_{k \geq 1}$ and properties of $U$ can give convergence claims with different strengths. Also refer to [Welling and Teh, 2011] for the discussion on the use of a preconditioner.
Algorithm 1 HC-Dyna

**Input:** budget $k$ for the number of gradient ascent steps (e.g., $k = 100$), stochasticity $\eta$ for gradient ascent (e.g., $\eta = 0.1$), $\rho$ percentage of updates from SC queue (e.g., $\rho = 0.5$), $d$ the number of state variables, i.e., $S \subset \mathbb{R}^d$

Initialize empty SC queue $B_{sc}$ and ER buffer $B_{er}$

$p_{ss}^{(t)} \leftarrow 1$ (empirical covariance matrix)

$\mu_{ss} \leftarrow 0 \in \mathbb{R}^{d\times d}, \mu_s \leftarrow 0 \in \mathbb{R}^d$ (auxiliary variables for computing empirical covariance matrix), sample average will be maintained for $\mu_{ss}, \mu_s$

$\epsilon_a \leftarrow \epsilon_a (threshold for accepting a state)$

for $t = 1, 2, \ldots$ do

Observe $(s, a, s', r)$ and add it to $B_{er}$

$\mu_{ss} \leftarrow \frac{\mu_{ss}(t-1) + ss^T}{t}, \mu_s \leftarrow \frac{\mu_s(t-1) + s}{t}$

$\Sigma_s \leftarrow \mu_s - \mu_s \mu_s^T$

$\epsilon_a \leftarrow (1 - \beta)\epsilon_a + \beta \text{distance}(s', s)$ for $\beta = 0.001$

Sample $s_0$ from $B_{er}$, $s \leftarrow \infty$

for $i = 0, \ldots, k$ do

$g_{si} \leftarrow \nabla_a Q(s_i) = \nabla_a \max_a Q \theta(s_i, a)$

$s_{i+1} \leftarrow \Pi(s_{i+1} = \frac{0.1}{||\Sigma_s g_{si}||} \sum_s g_{si} + X_i), X_i \sim \mathcal{N}(0, \eta \Sigma_s)$

if distance$(s, s_{i+1}) \geq \epsilon_a$ then

Add $s_{i+1}$ into $B_{sc}$, $s \leftarrow s_{i+1}$

for $n$ times do

Sample a mixed mini-batch $b$, with proportion $\rho$ from $B_{sc}$ and $1 - \rho$ from $B_{er}$

Update parameters $\theta$ (i.e., DQN update) with $b$

An important difference between the theoretical limiting distribution and the actual distribution acquired by our hill climbing method is that our trajectories would also include the states during the *burn-in* or transient period, which refers to the period before the stationary behavior is reached. We would want to point out that those states play an essential role in improving learning efficiency as we will demonstrate in section 6.2.

5 Hill Climbing Dyna

In this section, we provide the full algorithm, called Hill Climbing Dyna, summarized in Algorithm 1. The key component is to use the Hill Climbing procedure developed in the previous section, to generate states for search-control (SC). To ensure some separation between states in the search-control queue, we use a threshold $\epsilon_a$ to decide whether or not to add a state into the queue. We use a simple heuristic to set this threshold on each step, as the following sample average:

$\epsilon_a \approx \epsilon_a^{(T)} = \sum_{t=1}^{T} \frac{||s_{t+1} - s_t||^2}{\sqrt{T}}$. The start state for the gradient ascent is randomly sampled from the ER buffer.

In addition to using this new method for search control, we also found it beneficial to include updates on the experience generated in the real world. The mini-batch sampled for training has $\rho$ proportion of transitions generated by states from the SC queue, and $1 - \rho$ from the ER buffer. For example, for $\rho = 0.75$ with a mini-batch size of 32, the updates consists of $24(= 32 \times 0.75)$ transitions generated from states in the SC queue and 6 transitions from the ER buffer. Previous work using Dyna for learning NN value functions also used such mixed mini-batches [Holland et al., 2018].

One potential reason this addition is beneficial is that it alleviates issues with heavily skewing the sampling distribution to be off-policy. Tabular Q-learning is an off-policy learning algorithm, which has strong convergence guarantees under mild assumptions [Tsitsiklis, 1994]. When moving to function approximation, however, convergence of Q-learning is much less well understood. The change in sampling distribution for the states could significantly impact convergence rates, and potentially even cause divergence. Empirically, previous prioritized ER work pointed out that skewing the sampling distribution from the ER buffer can lead to a biased solution [Schaul et al., 2016]. Though the ER buffer is not on-policy, because the policy is continually changing, the distribution of states is closer to the states that would be sampled by the current policy than those in SC. Using mixed states from the ER buffer, and those generated by Hill Climbing, could alleviate some of the issues with this skewness.

Another possible reason that such mixed sampling could be necessary is due to model error. The use of real experience could mitigate issues with such error. We found, however, that this mixing has an effect even when using the true model. This suggests that this phenomenon indeed is related to the distribution over states.

We provide a small experiment in the GridWorld, depicted in Figure 1, using both a continuous-state and a discrete-state version. We include a discrete state version, so we can demonstrate that the effect persists even in a tabular setting when Q-learning is known to be stable. The continuous-state setting uses NNs—as described more fully in Section 6—with a mini-batch size of 32. For the tabular setting, the mini-batch size is 1; updates are randomly selected to be from the SC queue or ER buffer proportional to $\rho$. Figure 3 shows the performance of HC-Dyna as the mixing proportion increases from 0 (ER only) to 1.0 (SC only). In both cases, a mixing rate around $\rho = 0.5$ provides the best results. Generally, using too few search-control samples do not improve performance; focusing too many updates on search-control samples seems to slightly speed up early learning, but then later learning suffers. In all further experiments in this paper, we set $\rho = 0.5$.

6 Experiments

In this section, we demonstrate the utility of (DQN-)HC-Dyna in several benchmark domains, and then analyze the learning effect of different sampling distributions to generate states for the search-control queue.
6.1 Results in Benchmark Domains

In this section, we present empirical results on four classic domains: the GridWorld (Figure 1(a)), MountainCar, CartPole and Acrobat. We present both discrete and continuous action results in the GridWorld, and compare to DQN for the discrete control and to Deep Deterministic Policy Gradient (DDPG) for the continuous control [Lillicrap et al., 2016]. The agents all use a two-layer NN, with ReLU activations and 32 nodes in each layer. We include results using both the true model and the learned model, on the same plots. We further include multiple planning steps n, where for each real environment step, the agent does n updates with a mini-batch of size 32.

In addition to ER, we add an on-policy baseline called OnPolicy-Dyna. This algorithm samples a mini-batch of states (not the full transition) from the ER buffer, but then generates the next state and reward using an on-policy action. This baseline distinguishes when the gain of HC-Dyna algorithm is due to on-policy sampled actions, rather than because of the states in our search-control queue.

Discrete Action

The results in Figure 4 show that HC-Dyna nevem harms performance over ER and OnPolicy-Dyna, and in some cases significantly improves performance, (b) these gains persist even under learned models and (c) there are clear gains from HC-Dyna even with a small number of planning steps. Interestingly, using multiple mini-batch updates per time step can significantly improve the performance of all the algorithms. DQN, however, has very limited gain when moving from 10 to 30 planning steps on all domains except Gridworld, whereas HC-Dyna seems to more noticeably improve from more planning steps. This implies a possible limit of the usefulness of only using samples in the ER buffer.

We observe that the on-policy actions do not always help. The Gridworld domain is in fact the only one where on-policy actions (OnPolicy-Dyna) shows an advantage as the number of planning steps increase. This result provides evidence that the gain of our algorithm is due to the states in our search-control queue, rather than on-policy sampled actions. We also see that even though both model-based methods perform worse when the model has to be learned compared to when the true model is available, HC-Dyna is consistently better than OnPolicy-Dyna across all domains/settings.

To gain intuition for why our algorithm achieves superior performance, we visualize the states in the search-control queue for HC-Dyna in the Gridworld domain (Figure 5). We also show the states in the ER buffer at the same time step, for both HC-Dyna and DQN to contrast. There are two interesting outcomes from this visualization. First, the modification to search-control significantly changes where the agent explores, as evidenced by the ER buffer distribution. Second, HC-Dyna has many states in the SC queue that are near the goal region even when its ER buffer samples concentrate on the left part on the square. The agent can still update around the goal region even when it is physically in the left part of the domain.

Continuous Control

Our architecture can easily be used with continuous actions, as long as the algorithm estimates values. We use DDPG [Lillicrap et al., 2016] as an example for use inside HC-Dyna. DDPG is an actor-critic algorithm that uses the deterministic policy gradient theorem [Silver et al., 2014]. Let $\pi_{\psi}(\cdot): S \rightarrow A$ be the actor network parameterized by $\psi$, and $Q_{\theta}(\cdot, \cdot): S \times A \rightarrow \mathbb{R}$ be the critic. Given an initial state value $s$, the gradient ascent direction can be computed by $\nabla_{\psi} Q_{\psi}(s, \pi_{\psi}(s))$. In fact, because the gradient step causes small changes, we can further approximate this gra-
6.2 Investigating Sampling Distributions for Search-control

We next investigate the importance of two choices in HC-Dyna: (a) using trajectories to high-value regions and (b) using the agent’s value estimates to identify these important regions. To test this, we include following sampling methods for comparison: (a) HC-Dyna: hill climbing by using \( \hat{V}_Q \) (our algorithm); (b) Gibbs: sampling \( \propto \exp(\hat{V}_Q) \); (c) HC-Dyna-Vstar: hill climbing by using \( \hat{V}^* \) and (d) Gibbs-Vstar: sampling \( \propto \exp(V^*) \), where \( V^* \) is a pre-learned optimal value function. We also include the baselines OnPolicy-Dyna, ER and Uniform-Dyna, which uniformly samples states from the whole state space. All strategies mix with ER, using \( \rho = 0.5 \), to better give insight into performance differences.

To facilitate sampling from the Gibbs distribution and computing the optimal value function, we test on a simplified TabularGridWorld domain of size \( 20 \times 20 \), without any obstacles. Each state is represented by an integer \( i \in \{1, ..., 400\} \), assigned from bottom to top, left to right on the square with \( 20 \times 20 \) grids. HC-Dyna and HC-Dyna-Vstar assume that the state space is continuous on the square \([0, 1]^2\) and each grid can be represented by its center’s \((x, y)\) coordinates. We use the finite difference method for hill climbing.

Comparing to the Gibbs Distribution

As we pointed out the connection to the Langevin dynamics in Section 4.2, the limiting behavior of our hill climbing strategy is approximately a Gibbs distribution. Figure 7(a) shows that HC-Dyna performs the best among all sampling distributions, including Gibbs and other baselines. This result suggests that the states during the burn-in period matter. Figure 7(b) shows the state count by randomly sampling the same number of states from the HC-Dyna’s search-control queue and from that filled by Gibbs distribution. We can see that the Gibbs one concentrates its distribution only on very high value states.

Comparing to True Values

One hypothesis is that the value estimates guide the agent to the goal. A natural comparison, then, is to use the optimal values, which should point the agent directly to the goal. Figure 8(a) indicates that using the estimates, rather than true values, is more beneficial for planning. This result highlights that there does seem to be some additional value to focusing updates based on the agent’s current value estimates. Comparing state distribution of Gibbs-Vstar and HC-Dyna-Vstar in Figure 8(b) to Gibbs and HC-Dyna in Figure 7(b), one can see that both distributions are even more concentrated, which seems to negatively impact performance.

7 Conclusion

We presented a new Dyna algorithm, called HC-Dyna, which generates states for search-control by using hill climbing on value estimates. We proposed a noisy natural projected gradient ascent strategy for the hill climbing process. We demonstrate that using states from hill climbing can significantly improve sample efficiency in several benchmark domains. We empirically investigated, and validated, several choices in our algorithm, including the use of natural gradients, the utility of mixing with ER samples, the benefits of using estimated values for search control. A natural next step is to further investigate other criteria for assigning importance to states. Our HC strategy is generic for any smooth function; not only for value estimates. A possible alternative is to investigate importance based on error in a region, or based more explicitly on optimism or uncertainty, to encourage systematic exploration.

Acknowledgments

We would like to acknowledge funding from the Canada CIFAR AI Chairs Program, Amii and NSERC.
References


