TEMPORAL DIFFERENCE MODEL LEARNING: MODEL-FREE DEEP RL FOR MODEL-BASED CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Model-free reinforcement learning (RL) has been proven to be a powerful, general tool for learning complex behaviors. However, its sample efficiency is often impractically large for solving challenging real-world problems, even for off-policy algorithms such as Q-learning. A limiting factor in classic model-free RL is that the learning signal consists only of scalar rewards, ignoring much of the rich information contained in state transition tuples. Model-based RL uses this information, by training a predictive model, but often does not achieve the same asymptotic performance as model-free RL due to model bias. We introduce temporal difference models (TDMs), a family of goal-conditioned value functions that unify model-free learning and model-based learning. TDMs combine the benefits of model-free and model-based RL: they leverage the rich information in state transitions to learn very efficiently, while still attaining asymptotic performance that exceeds that of direct model-based RL methods. Our experimental results show that, on a range of continuous control tasks, TDMs provide a substantial improvement in efficiency compared to state-of-the-art model-free methods.

1 INTRODUCTION

Reinforcement learning algorithms provide a formalism for autonomous learning of complex behaviors. When combined with rich and expressive function approximators such as deep neural networks, reinforcement learning can provide for impressive results on tasks ranging from playing games (Mnih et al., 2015; Silver et al., 2016), to flying and driving (Lillicrap et al., 2015; Zhang et al., 2016), to controlling robotic arms (Levine et al., 2016; Gu et al., 2017). However, these deep reinforcement learning algorithms often require a large amount of experience to arrive at an effective solution, which can severely limit their application to real-world problems where this experience might need to be gathered directly on a real physical system. Part of the reason for this is that direct, model-free reinforcement learning learns only from the reward: experience that receives no reward provides minimal supervision to the learner.

In contrast, model-based reinforcement learning algorithms obtain a large amount of supervision from every sample, since they can use each sample to better learn how to predict the system dynamics – that is, to learn the "physics" of the problem. Once the dynamics are learned, near-optimal behavior can in principle be obtained by planning through these dynamics. Model-based algorithms tend to be substantially more efficient (Deisenroth et al., 2013; Nagabandi et al., 2017), but often at the cost of larger asymptotic bias: when the dynamics cannot be learned perfectly, as is the case for most complex problems, the final policy can be highly suboptimal. Therefore, conventional wisdom holds that model-free methods are less efficient but achieve the best asymptotic performance, while model-based methods are more efficient but do not produce policies that are as optimal.

Can we devise methods that retain the efficiency of model-based reinforcement learning while still achieving the asymptotic performance of model-free learning? This is the question that we study in this paper. The search for methods that combine the best of model-based and model-free learning has been ongoing for decades, with techniques such as synthetic experience generation (Sutton, 1990), partial model-based backpropagation (Nguyen & Widrow, 1990; Heess et al., 2015), and layering model-free learning on the residuals of model-based estimation (Chebotar et al.) being a few examples. However, a direct unification of model-free and model-based RL has remained elusive. An effective unification of model-free and model-based RL should be able to smoothly

transition from learning models to learning policies, obtaining rich supervision from every sample to quickly gain a moderate level of proficiency, while still converging to an unbiased solution.

To arrive at a connection between model-free and model-based reinforcement learning, we study a variant of goal-conditioned value functions (Sutton et al., 2011; Schaul et al., 2015; Andrychowicz et al., 2017). Goal-conditioned value functions learn to predict the value function for every possible goal state. That is, they answer the following question: what is the expected reward for reaching a particular state, given that the agent is attempting (as optimally as possible) to reach it? The particular choice of reward function determines what such a method actually does, but many choices provide a tantalizing glimmer of a connection to model-based learning: if we can predict how easy it is to reach any state from any current state, we must have some kind of understanding of the underlying "physics." In this work, we show that this connection can in fact be made precise: for a specific choice of reward, goal-conditioned value functions in fact learn models. These models can be used as part of an optimal control method for model-based planning. Extension toward more model-free learning is achieving by acquiring "multi-step models" that can be used to plan over progressively coarser temporal resolutions, eventually arriving at a fully model-free formulation.

The principal contribution of our work is a new reinforcement learning algorithm that makes use of this connection between model-based and model-free learning to learn a specific type of goal-conditioned value function, which we call a temporal difference model (TDM). This value function can be learned very efficiently, with sample complexities that are competitive with model-based reinforcement learning, and can then be used with an MPC-like method to accomplish desired tasks. Our empirical experiments demonstrate that this method achieves substantially better sample complexity than fully model-free learning on a range of challenging continuous control tasks, while outperforming purely model-based methods in terms of final performance. Furthermore, the connection that our method elucidates between model-based and model-free learning may lead to a range of interesting future methods.

2 PRELIMINARIES

In this section, we introduce the reinforcement learning (RL) formalism, temporal difference Q-learning methods, and goal-conditioned value functions. We will build on these components to develop temporal difference models (TDMs) in the next section. RL deals with decision making problems that consist of a state space $\mathcal{S}$, action space $\mathcal{A}$, transition dynamics $P(s' \mid s, a)$, and an initial state distribution p_0. The goal of the learner is encapsulated by a reward function $r(s, a, s')$. Typically, long or infinite horizon tasks also employ a discount factor γ, and the standard objective is to find a policy $\pi(a \mid s)$ that maximizes the expected sum of rewards, $\mathbb{E}_\pi[\sum_t \gamma^t r(s_t, a_t, s_{t+1})]$, where $s_0 \sim p_0$, $a_t \sim \pi(a_t|s_t)$, and $s_{t+1} \sim P(s' \mid s, a)$.

Q-functions. We will focus in particular on RL algorithms that learn a Q-function. The Q-function represents the expected total (discounted) reward that can be obtained by the optimal policy after taking action a_t in state s_t, and can be defined recursively as following:

$$Q(s_t, a_t) = \mathbb{E}_{p(s_{t+1}|s_t,a_t)}[r(s_t, a_t, s_{t+1}) + \gamma \max_a Q(s_{t+1}, a)]. \tag{1}$$

The optimal policy can then recovered according to $\pi(a_t|s_t) = \delta(a_t = \arg\max_a Q(s_t, a))$. Q-learning algorithms (Watkins & Dayan, 1992; Riedmiller, 2005) learn the Q-function via an off-policy stochastic gradient descent algorithm, estimating the expectation in the above equation with samples collected from the environment and computing its gradient. Q-learning methods can use transition tuples (s_t, a_t, s_{t+1}, r_t) collected from any exploration policy, which generally makes them more efficient than direct policy search, though still less efficient than purely model-based methods.

Model-based reinforcement learning and optimal control. Model-based reinforcement learning takes a different approach to maximize the expected reward. In model-based RL, the aim is to train a model of the form $f(s_t, a_t)$ to predict the next state s_{t+1}. Once trained, this model can be used to choose actions, either by backpropagating reward gradients into a policy, or planning directly through the model. In the latter case, a particularly effective method for employing a learned model is model-predictive control (MPC), where a new action plan is generated at each time step, and the first action of that plan is executed, before replanning begins from scratch. MPC can be formalized

as the following optimization problem:

$$a_t = \underset{a_{t:t+T}}{\text{argmax}} \sum_{i=t}^{t+T} r(s_i, a_i) \text{ where } s_{i+1} = f(s_i, a_i) \ \forall \ i \in \{t, ..., t+T-1\}. \tag{2}$$

The above optimization can be solved either as a constrained or unconstrained optimization problem, through a variety of optimization methods. We can also write the dynamics constraint in the above equation in terms of an implicit dynamics, according to

$$a_t = \underset{a_{t:t+T}, s_{t+1:t+T}}{\text{argmax}} \sum_{i=t}^{t+T} r(s_i, a_i) \text{ such that } C(s_i, a_i, s_{i+1}) = 0 \ \forall \ i \in \{t, ..., t+T-1\}, \tag{3}$$

where $C(s_i, a_i, s_{i+1}) = 0$ if and only if $s_{i+1} = f(s_i, a_i)$. This implicit version will be important in understanding the connection between model-based and model-free RL.

Goal-conditioned value functions. Q-functions trained for a specific reward are specific to the corresponding task, and learning a new task requires optimizing an entirely new Q-function. Goal-conditioned value functions can address this limitation by conditioning the Q-value for some task description vector $s_g \in \mathcal{G}$ in a goal space $\mathcal{G}$. This induces a parameterized reward $r(s_t, a_t, s_{t+1}, s_g)$, which in turn gives rise to parameterized Q-functions of the form $Q(s, a, s_g)$. A number of goal-conditioned value function methods have been proposed in the literature, such as universal value functions (Schaul et al., 2015) and Horde (Sutton et al., 2011). When the goal corresponds to an entire state, such goal-conditioned value functions essentially predict how well an agent can reach a particular state, *when it is trying to reach it*. The knowledge contained in such a value function is intriguingly close to a model: knowing how well you can reach any state is closely related to understanding the physics of the environment. With Q-learning, such value function can be learned for any goal s_g using the same off-policy (s_t, a_t, s_{t+1}) tuples. When previously visited states can be relabeled with the reward for any goal, this leads to a natural data augmentation strategy, since each tuple can be replicated many times for many different goals without additional data collection. Andrychowicz et al. (2017) used this property to produce an effective curriculum for solving multi-goal task with delayed rewards. As we discuss below, relabeling past experience with different goals provides a critical piece of the connection between model-based and model-free learning.

3 Temporal Difference Model Learning

In this section, we introduce a type of goal-conditioned value functions called temporal difference models (TDMs) that provide a direct connection to model-based RL. We will first motivate this connection by relating the model-based MPC optimizations in Equations (2) and (3) to goal-conditioned value functions, and then present our temporal difference model derivation, which extends this connection from a purely model-based setting into one that becomes increasingly model-free.

3.1 From Goal-Conditioned Value Functions to Models

Goal-conditioned value functions by themselves already provide some notion of model-based learning, but cannot be readily incorporated into a true model-based reinforcement learning framework. Although goal-conditioned value functions somewhat relax the restriction of needing to reoptimize a new Q-function for every task, the value function is still only useful for tasks that involve reaching goals in the goal set with a specific reward – namely, the one that was used to train the goal-conditioned value function. Let us consider the choice of reward function $r_d(s_t, a_t, s_{t+1}, g)$ in a goal-conditioned value function. Although a variety of options have been explored in the literature (Sutton et al., 2011; Schaul et al., 2015; Andrychowicz et al., 2017), a particularly intriguing connection to model-based RL emerges if we set $\mathcal{G} = \mathcal{S}$, such that $g \in \mathcal{G}$ corresponds to a *goal state* $s_g \in \mathcal{S}$, and we consider reward functions of the following form:

$$r_d(s_t, a_t, s_{t+1}, s_g) = -D(s_{t+1}, s_g),$$

where $D(s_{t+1}, s_g)$ is a distance, such as the Euclidean distance $D(s_{t+1}, s_g) = \|s_{t+1} - s_g\|_2$. In the case where $\gamma = 0$, we have $Q(s_t, a_t, s_g) = -D(s_{t+1}, s_g)$ at convergence of Q-learning, which means that $Q(s_t, a_t, s_g) = 0$ implies that $s_{t+1} = s_g$. We can then plug this Q-function directly into

the model-based planning optimization in Equation (3), denoting the task control reward as r_c, such that the solution to

$$a_t = \underset{a_{t:t+T}, s_{t+1:t+T}}{\text{argmax}} \sum_{i=t}^{t+T} r_c(s_i, a_i) \text{ such that } Q(s_i, a_i, s_{i+1}) = 0 \ \forall \ i \in \{t, ..., t+T-1\} \tag{4}$$

yields a model-based plan. We have now derived a precise connection between model-free and model-based RL, in that model-free learning of goal-conditioned value functions can be used to directly produce an implicit model that can be used with MPC-based planning. However, this connection by itself is not very useful: the resulting implicit model is fully model-based, and does not provide any kind of long-horizon capability. In the next section, we show how to extend this connection into the long-horizon setting by introducing the temporal difference model (TDM).

3.2 Long-Horizon Learning with Temporal Difference Models

If we consider the case where $\gamma > 0$, the optimization in Equation (4) no longer corresponds to any optimal control method. In fact, when $\gamma = 0$, Q-values have well-defined units: units of distance between states. For $\gamma > 0$, no such interpretation is possible. The key insight in temporal difference models is to introduce a different mechanism for aggregating long-horizon rewards. Instead of evaluating Q-values as discounted sums of rewards, we introduce an additional input τ, which represents the planning horizon, and define the Q-learning recursion as

$$Q(s_t, a_t, s_g, \tau) = \mathbb{E}_{p(s_{t+1}|s_t,a_t)}[-D(s_{t+1}, s_g)\mathbb{1}[\tau = 0] + \max_a Q(s_{t+1}, a, s_g, \tau - 1)\mathbb{1}[\tau \neq 0]]. \tag{5}$$

Intuitively, the Q-function uses a reward of $-D(s_{t+1}, s_g)$ when $\tau = 0$ (at which point the episode terminates), and decrements τ by one at every other step. Since this is still a well-defined Q-learning recursion, it can be optimized with off-policy data and, just as with goal-conditioned value functions, we can resample new goals s_g and new horizons τ for each tuple (s_t, a_t, s_{t+1}), even ones that were not actually used when the data was collected. In this way, the TDM can be trained very efficiently, since every tuple provides supervision for every possible goal and every possible horizon.

The intuitive interpretation of the TDM is that it tells us how close the agent will get to a given goal state s_g after τ time steps, *when it is attempting to reach that state in τ steps*. The TDM can be incorporated into a variety of planning and optimal control schemes at test time, including a temporally coarsened version of Equation (4), given by

$$\begin{aligned} a_t = & \underset{a_{t:K:t+T}, s_{t+K:K:t+T}}{\text{argmax}} \sum_{i=t,t+K,...,t+T} r_c(s_i, a_i) \\ & \text{such that } Q(s_i, a_i, s_{i+K}, K-1) = 0 \ \forall \ i \in \{t, t+K, ..., t+T-K\}, \end{aligned} \tag{6}$$

where we optimize over every K^{th} state and action. As the TDM becomes effective for longer and longer horizons, we can increase K until, in the limit, $K = T$, and planning needs to be performed over only a single time step:

$$a_t = \underset{a_t, a_{t+T}, s_{t+T}}{\text{argmax}} \ r_c(s_{t+T}, a_{t+T}) \text{ such that } Q(s_t, a_t, s_{t+T}, T-1) = 0. \tag{7}$$

This formulation does result in some loss of generality, since we no longer optimize the reward at the intermediate steps. This limits the multi-step formulation to terminal reward problems, but does allow us to accommodate arbitrary reward functions on the terminal state s_{t+T}, which still describes a broad range of practically relevant tasks. In the next section, we describe how TDMs can be implemented and used in practice for continuous state and action spaces.

4 Training and Using Temporal Difference Models

The TDM can be trained with any off-policy Q-learning algorithm, such as DQN (Mnih et al., 2015), DDPG (Lillicrap et al., 2015), NAF (Gu et al., 2016), and SDQN (Metz et al., 2017). During off-policy Q-learning, TDMs can benefit from arbitrary relabeling of the goal states g and the horizon τ, given the same (s_t, a_t, s_{t+1}) tuples from the behavioral policy as done in (Andrychowicz et al., 2017). This relabeling enables simultaneous, data-efficient learning of short-horizon and

long-horizon behaviors for arbitrary goal states, unlike previously proposed goal-conditioned value functions that only learn for a single time scale, typically determined by a discount factor (Schaul et al., 2015; Andrychowicz et al., 2017). In this section, we describe a practical TDM algorithm. We discuss specific design decisions, including the reward function, the Q-function architecture to enable explicit MPC with TDMs, and the dynamic goal state and horizon sampling schemes.

4.1 Reward Function Specification

Q-learning typically optimizes scalar rewards, but TDMs enable us to increase the amount of supervision available to the Q-function by using a vector-valued reward. Specifically, if the distance $D(s, s_g)$ factors additively over the dimensions we can train a vector-valued Q-function that predicts per-dimension distance, with the reward function for dimension j given by $-D_j(s_j, s_{g,j})$. We use the ℓ_1 norm in our implementation, which corresponds to absolute value reward $-|s_j - s_{g,j}|$. The resulting vector-valued Q-function can learn distances along each dimension separately, providing it with more supervision from each training point. Empirically, we found that this modifications provides a substantial boost in sample efficiency.

We can optionally make an additional improvement to TDMs if we know that the task reward r_c depends only on some subset of the state or, more generally, state features. In that case, we can train the TDM to predict distances along only those dimensions or features that are used by r_c, which in practice can substantially simplify the corresponding prediction problem. In our experiments, we illustrate this property by training TDMs for an arm reaching and pushing task that predict distances from an end-effector and pushed object, without accounting for internal joints of the arm.

4.2 Q-function Architecture for Explicit MPC with TDMs

While the TDM optimal control formulation Equation (7) drastically reduces the number of states and actions to be optimized for long-term planning, it requires solving a constrained optimization problem, which is more computationally expensive than unconstrained problems. We can remove the need for a constrained optimization through a specific architectural decision in the design of the function approximator for $Q(s, a, s_g, \tau)$. We define the Q-function as $Q(s, a, s_g, \tau) = -\|f(s, a, s_g, \tau) - g\|$, where $f(s, a, s_g, \tau)$ outputs a state vector. By training the TDM with a standard Q-learning method, $f(s, a, s_g, \tau)$ is trained to explicitly predict the state that will be reached by a policy attempting to reach s_g in τ steps. This model can then be used to choose the action with a fully explicit MPC formulation:

$$a_t = \underset{a_t, a_{t+T}, s_{t+T}}{\operatorname{argmax}} \; r_c(f(s_t, a_t, s_{t+T}, T-1), a_{t+T}) \tag{8}$$

It is also straightforward to derive a multi-step version of this formulation, as in Equation (6).

4.3 Dynamic Goal and Horizon Resampling

While Q-learning is valid for any value of g and τ for each transition tuple (s_t, a_t, s_{t+1}), the way in which these values are sampled during training can affect learning efficiency. We found that an effective strategy for sampling s_g is to mix the following three types of samples, with the mixture ratios as hyperparameters of the method: (1) uniformly sample future states along the actual trajectory in the buffer (i.e., for s_t, choose $s_g = s_{t+k}$ for a random k); (2) uniformly sample goal states from the replay buffer; (3) uniformly sample goals from a uniform range of valid states. The horizon τ is sampled uniformly at random between 0 and a maximum horizon τ_{max}.

4.4 Algorithm Summary

The algorithm is summarized as Algorithm 1. A crucial difference from prior goal-conditioned value function methods (Schaul et al., 2015; Andrychowicz et al., 2017) is that our algorithm can be used to act according to an arbitrary terminal reward function r_c, both during exploration and at test time. Like other off-policy algorithms (Mnih et al., 2015; Lillicrap et al., 2015), it consists of exploration and Q-function fitting. Noise is injected into the action chosen by the control algorithm for exploration, and Q-function fitting uses standard Q-learning techniques, with target networks and experience replay (Mnih et al., 2015; Lillicrap et al., 2015). If we view the Q-function fitting as

Algorithm 1 Temporal Difference Model Learning

Require: Task reward function $r_c(s, a)$, parameterized TDM $Q_w(s, a, s_g, \tau)$, replay buffer $\mathcal{B}$

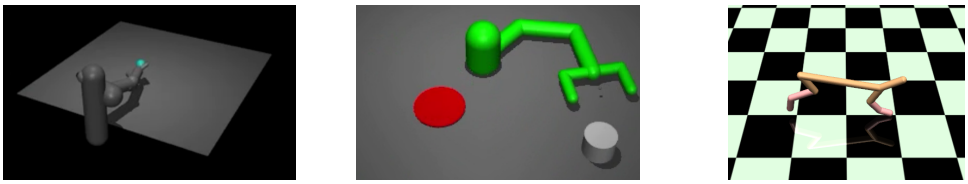
Initialize target Q network $Q' \leftarrow Q_w$
for $n = 0, ..., N-1$ episodes **do**
  $s_0 \sim p(s_0)$
  for $t = 0, ..., T-1$ time steps **do**
    $a_t^* = \text{MPC}(r_c, s_t, Q_w, T-t)$ // Eq. 6, Eq. 7, or Eq. 8
    $a_t = \text{AddNoise}(a_t^*)$ // Noisy exploration
    $s_{t+1} \sim p(s_t, a_t)$ // Step environment
    Store $\{s_t, a_t, s_{t+1}\}$ in the replay buffer $\mathcal{B}$.
    for $i = 0, I-1$ iterations **do**
      Sample M transitions $\{s_m, a_m, s'_m\}$ from the replay $\mathcal{B}$.
      Relabel time horizons and goal states $\tau_m, s_{g,m}$ // Section 4.3
      $y_m = -\|s'_m - s_{g,m}\| \mathbb{1}[\tau_m = 0] + \max_a Q'(s'_m, a, s_{g,m}, \tau_m - 1)\mathbb{1}[\tau_m \neq 0]$
      $L(w) = \sum_m (Q_w(s_m, a_m, s_{g,m}, \tau_m) - y_m)^2 / M$ // Compute the loss
      Minimize$(w, L(w))$ // Optimize
      SlowlyUpdate(Q', Q_w) // Update target network
    end for
  end for
end for

model fitting, the algorithm also resembles iterative model-based RL, which alternates between collecting data using the learned dynamics model for planning (Deisenroth & Rasmussen, 2011). Since we focus on continuous tasks in our experimental evaluation, we use the DDPG algorithm (Lillicrap et al., 2015), though any Q-learning method could be used.

5 Related Work

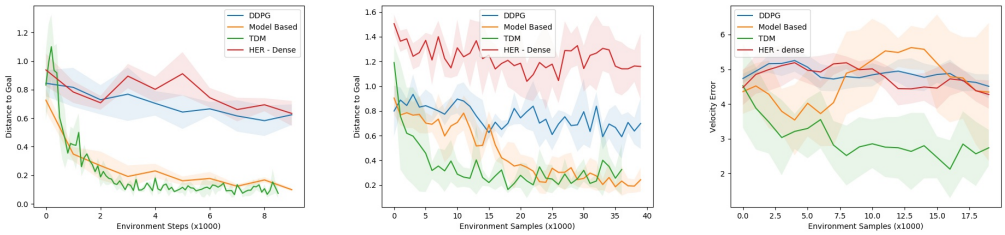
Combining model-based and model-free reinforcement learning techniques is a well-studied problem, though no single solution has demonstrated all of the benefits of model-based and model-free learning. Some methods first learn a model and use this model to simulate experience (Sutton, 1990; Gu et al., 2016) or compute better gradients for model-free updates (Heess et al., 2015; Nguyen & Widrow, 1990). Other methods use model-free algorithms to correct for the local errors made by the model (Chebotar et al.; Bansal et al., 2017). Common to those methods are that the model-based RL mostly uses the standard next step predictive model fitted to transition tuples using regression, while our approach is proposing a generalization of the definitions and learning methods of "predictive models" and how to use them in explicit or implicit optimal control, derived from model-free techniques. As a result, our approach achieves much better sample efficiency in practice on a variety of challenging reinforcement learning tasks than model-free alternatives, while exceeding the performance of purely model-based approaches.

A central component of our method is to train goal-conditioned value functions, or more generally contextual value functions. Many variants of contextual value functions have been proposed in the literature Foster & Dayan (2002); Sutton et al. (2011); Schaul et al. (2015); Dosovitskiy & Koltun (2016). Critically, unlike the works on contextual policies (Caruana, 1998; Da Silva et al., 2012; Kober et al., 2012) which require on-policy trajectories with each new goal, the value function approaches such as Horde (Sutton et al., 2011) and UVF (Schaul et al., 2015) can reuse off-policy data to learn rich contextual value functions using the same data. A particularly related UVF extension is a method called hindsight experience replay (HER) Andrychowicz et al. (2017). Both HER and our method retroactively relabel past experience with goal states that are different from the goal aimed for during data collection. However, unlike our method, the standard UVF formulation in HER uses a single temporal scale when learning, and does not explicitly provide for a connection between model-based and model-free learning. Furthermore our approach is task-generic and does not require any manual selection of state dimensions for goals as in HER. The practical result of these differences is that our approach empirically achieves substantially better sample complexity on a wide range of complex continuous control tasks than HER, while the theoretical connection between model-based and model-free learning sheds light on the underlying reasons for the efficiency of UVF methods such as HER and our algorithm.



(a) 7-DoF Reacher (b) Pusher (c) Half Cheetah

Figure 1: The simulated tasks in our experiments: (a) reaching target locations, (b) pushing a puck to a random target, and (c) training the cheetah to run at target velocities.



(a) 7-Dof Reacher (b) Pusher (c) Half Cheetah

Figure 2: The comparison of TDM with the baseline methods in model-free (DDPG), model-based, and goal-conditioned value functions (HER - Dense). Our method, which uses model-free learning, is significantly more sample-efficient than model-free alternatives including DDPG and HER and slightly even improves upon the best model-based performance. In particular, while TDM and HER are both goal-conditioned value functions, our method performs significantly better in terms of sample efficiency due to our novel design choices and the use of MPC.

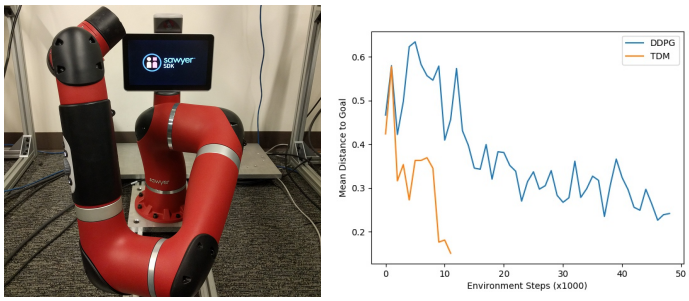
6 EXPERIMENTS

Our experiments examine how the sample efficiency of TDMs compares to both model-based and model-free RL algorithms. We also aim to study the importance of several key design decisions in TDMs, and evaluate the algorithm on a real-world robotic platform. For the model-free comparison, we compare to DDPG (Lillicrap et al., 2015), which typically achieves the best sample efficiency on benchmark tasks (Duan et al., 2016), as well as hindsight experience replay (HER), which uses goal-conditioned value functions (Andrychowicz et al., 2017). We were unable to obtain good results from the sparse version of HER, and therefore compare to the dense reward version. For the model-based comparison, we compare to Nagabandi et al. (2017), a recent work that reports highly efficient learning with neural network dynamics models. We perform the comparison on three simulated tasks: a 7 DoF arm reaching various random end-effector targets, an arm pushing a puck to a target location, and a planar cheetah attempting to reach a goal velocity (either forward or backward). The tasks are shown in Figure 1. The pushing task requires long-horizon reasoning to reach and push the puck, and the cheetah task requires handling contact discontinuities, both of which are challenging for model-based methods. We also evaluated our method on a real-world robotic arm reaching end-effector positions, to study its applicability to real-world tasks.

For both the simulated and real-world 7-DoF arm, our TDM is trained on all state components. For the pushing task, our TDM is trained on the hand and puck Cartesian coordinates. For the half cheetah task, our TDM is trained on the horizontal velocity of the cheetah. Full experimental details and task specifications are in the Appendix. Videos of the experiments are available here: `https://sites.google.com/site/tdmlearning/`.

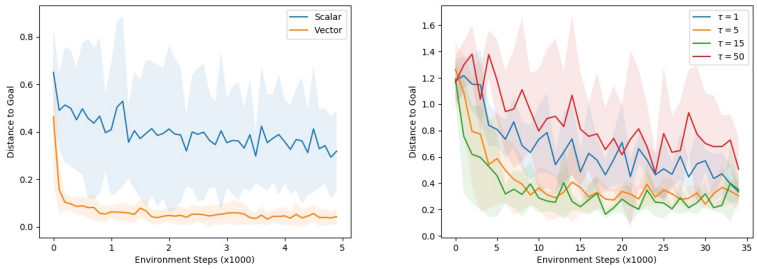
6.1 TDMS VS MODEL-FREE, MODE-BASED, AND DIRECT GOAL-CONDITIONED RL

The results on the simulated tasks are shown in Figure 2. The pure model-based method learns substantially faster on the reacher and pusher tasks when compared to the prior model-free algorithms, though the gap narrows on the harder pusher task. Interestingly, TDMs actually learn faster than the model-based algorithm in both settings, and substantially outperform the model-based algorithm on the harder pusher task. These results demonstrate that TDMs are not only orders of magnitude



(a) Sawyer Robot (b) Performance curves on Sawyer

Figure 3: The comparison of TDM with DDPG on a real robot. Our method still leads to improved sample efficiency and faster learning.



(a) Scalar vs Vectorized TDMs (b) TDMs with different τ_{max}

Figure 4: Ablation experiments for (a) scalar vs. vectorized TDMs on 7-DoF reacher and (b) different τ_max on pusher. The vectorized variant performs substantially better, while the horizon effectively interpolates between model-based and model-free learning.

faster than model-free learning (i.e., DDPG and HER), but can also exceed the sample-efficiency of standard model-based methods. On the variable-velocity cheetah task, we find that TDMs also substantially outperform model-based learning in terms of final performance. The model-free DDPG algorithm does successfully learn this task, but requires drastically more samples. The comparison with HER (Andrychowicz et al., 2017) is particularly illuminating: although this method also uses goal-conditioned models, it lacks the direct connection to model-based RL, and generally is much closer to DDPG in terms of sample efficiency (e.g., on the reacher task).

Since the sample efficiency of TDM training is well within the range that is feasible for real-world physical experiments, we ran the algorithm on a Sawyer robotic arm to learn a real-world analogue of the reaching task. The results, shown in Figure 3, show that the algorithm learns quickly and successfully.

6.2 Ablation Studies

TDM methods come with specific design choices. In this section, we discuss two key design choices that provide substantially improved performance. First, Figure 4a examines the tradeoffs between the vectorized and scalar rewards. The results show that the vectorized formulation learns substantially faster than the naïve scalar variant. Second, Figure 4b compares the learning speed for different horizon values τ_{max}. Performance degrades when the horizon is too low, and learning becomes slower when the horizon is too high. Interestingly, the performance gap between $\tau_{max} = 0$ and $\tau_{max} = \{5, 15\}$ seems to be consistent with the performance gap observed in Figure 2b between TDM and model-based learning, suggesting that the method is effectively interpolating between model-based and model-free learning.

7 CONCLUSION

In this paper, we derive a connection between model-based and model-free reinforcement learning, and present a novel reinforcement learning algorithm that exploits this connection to greatly improve on the sample efficiency of state-of-the-art model-free deep reinforcement learning algorithms. Our temporal difference models can be viewed both as goal-conditioned value functions and implicit dynamics models, which enables them to be trained efficiently on off-policy data while still minimizing the effects of model bias. As a result, they achieve asymptotic performance that compares favorably with model-free algorithms, but with a sample complexity that is comparable to purely model-based methods.

The connection between model-based and model-free reinforcement learning explored in this paper provides a number of avenues for future work. While we demonstrate the use of TDMs with a very basic planning approach, TDMs can in principle be combined with any optimal control or trajectory optimization algorithm that can make use of an implicit model. Further exploring how TDMs can be incorporated into powerful constrained optimization methods for model-predictive control (MPC) or trajectory optimization is an exciting avenue for future work. Another direction for future is to further explore how TDMs can be applied to complex state representations, such as images, where simple distance metrics may no longer be effective. Although direct application of TDMs to these domains is not straightforward, a number of works have studied how to construct metric embeddings of images that could in principle provide viable distance functions. Finally, the promise to enable sample-efficient learning with the performance of model-free RL and the efficiency of model-based RL is to enable widespread RL application on real-world physical systems, and many more real-world applications in robotics, autonomous driving and flight, and other control domains could be explored in future work.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *arXiv preprint arXiv:1707.01495*, 2017.

Somil Bansal, Roberto Calandra, Ted Xiao, Sergey Levine, and Claire J Tomlin. Goal-driven dynamics learning via bayesian optimization. *arXiv preprint arXiv:1703.09260*, 2017.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Rich Caruana. Multitask learning. In *Learning to learn*, pp. 95–133. Springer, 1998.

Yevgen Chebotar, Karol Hausman, Marvin Zhang, Gaurav Sukhatme, Stefan Schaal, and Sergey Levine. Combining model-based and model-free updates for deep reinforcement learning.

Bruno Da Silva, George Konidaris, and Andrew Barto. Learning parameterized skills. *arXiv preprint arXiv:1206.6398*, 2012.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.

Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*, 2016.

Y. Duan, R. Chen, X. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

David Foster and Peter Dayan. Structure in the space of value functions. *Machine Learning*, 49(2): 325–346, 2002.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pp. 2829–2838, 2016.

Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 3389–3396. IEEE, 2017.

Nicolas Heess, Greg Wayne, David Silver, Timothy Lillicrap, Yuval Tassa, and Tom Erez. Learning Continuous Control Policies by Stochastic Value Gradients. *arXiv*, pp. 1–13, 2015. ISSN 10495258. URL `http://arxiv.org/abs/1510.09142`.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jens Kober, Andreas Wilhelm, Erhan Oztop, and Jan Peters. Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots*, 33(4):361–379, 2012.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *arXiv preprint arXiv:1708.02596*, 2017.

Derrick H Nguyen and Bernard Widrow. Neural networks for self-learning control systems. *IEEE Control systems magazine*, 10(3):18–23, 1990.

Martin Riedmiller. Neural fitted q iteration-first experiences with a data efficient neural reinforcement learning method. In *ECML*, volume 3720, pp. 317–328. Springer, 2005.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1312–1320, 2015.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning*, pp. 216–224, 1990.

Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 528–535. IEEE, 2016.

A EXPERIMENT DETAILS

In this section, we detail the experimental setups used in our results.

A.1 OFF-POLICY RL SETUPS

In all our experiments, we used DDPG (Lillicrap et al., 2015) as the base off-policy model-free RL algorithm for learning the TDMs $Q(s, a, g, s_\tau)$. Experience replay (Mnih et al., 2015) has size of 200 thousand transitions, and the soft target networks (Lillicrap et al., 2015) are used with update rate of 0.001. Learning rates of the critic and the actor are chosen from {1e-4, 1e-3} and {1e-4,1e-3} respectively. Adam (Kingma & Ba, 2014) is used as the base optimizer with default parameters except the learning rate.

While any distance metric for the TDM reward function can be used, we chose L1 norm $-\|s_{t+1} - s_g\|_1$ to ensure that the scalar and vectorized TDMs are consistent.

A.2 HYPERPARAMETERS

A.3 GOAL STATE SAMPLING STRATEGY

For sampling the goal state s_g, we considered three schemes: (1) uniformly from future states within the trajectory, (2) uniformly from the entire experience replay, and (3) uniformly from a fixed range specified manually .The design choice for resampling goals g in goal-conditioned value functions is also explored in Andrychowicz et al. (2017) to successfully develop curriculum for goal-reaching tasks with sparse rewards. We found that it did not matter which of these strategies we used.

A.4 NETWORK ARCHITECTURE AND VECTOR-BASED SUPERVISION

The neural network architecture for the critic network is a two-hidden-layer neural network with sizes $(300, 300)$ and softplus hidden activations, while that for the actor network is a two-hidden-layer neural network with the same size, ReLU hidden activations, and a tanh output activation. The goal state g and τ are directly concatenated to the state s and passed into both networks.

Since we know that the true Q-function must learn to predict (negative) distances, we incorporate this prior knowledge into the Q-function by parameterizing it as $Q(s, a, s_g, \tau) = -\|_1 f(s, a, s_g, \tau) - s_g\|_1$. Here, f is a vector outputted by a feed-forward neural network and has the same dimension as the goal. This parameterization ensures that the Q-function outputs non-positive values, while encouraging the Q-function to learn what we call a goal-conditioned model: f is encouraged to predict what state will be reached at after τ, when the policy is trying to reach goal s_g.

The scalar supervision regresses

$$Q(s_t, a_t, s_g, \tau) = -\sum_j |f_j(s_t, a_t, s_g, \tau) - s_{g,j}|$$

onto

$$\begin{aligned} & r(s_t, a_t, s_{t+1}, s_g) + \mathbb{1}[\tau = 0] + Q(s_{t+1}, a^*, s_g, \tau - 1)\mathbb{1}[\tau \neq 0] \\ = & -\sum_j \{|s - s_{t+1}|\mathbb{1}[\tau = 0] + |f_j(s_t, a^*, s_g, \tau - 1) - s_{g,j}|\mathbb{1}[\tau \neq 0]\} \end{aligned}$$

where $a^* = \text{argmax}_a Q(s_{t+1}, a, s_g, \tau - 1)$. The vectorized supervision instead supervises each components of f, so that

$$|f_j(s_t, a_t, s_g, \tau) - s_{g,j}|$$

regresses onto

$$|s - s_{t+1}|\mathbb{1}[\tau = 0] + |f_j(s_t, a^*, s_g, \tau - 1) - s_{g,j}|\mathbb{1}[\tau \neq 0]$$

for each dimension j of the state.

A.5 TASK AND REWARD DESCRIPTIONS

Benchmark tasks are designed on MuJoCo physics simulator (Todorov et al., 2012).

7-DoF reacher.: The state consists of 7 joint angles, 7 joint angular velocities, and 3 XYZ observation of the tip of the arm, making it 17 dimensional. The action controls torques for each joint, totally 7 dimensional. The reward function during optimization control and for the model-free baseline is the negative Euclidean distance between the XYZ of the tip and the target XYZ. The targets are sampled randomly from all reachable locations of the arm at the beginning of each episode. The robot model is taken from the striker and pusher environments in OpenAI Gym MuJoCo domains (Brockman et al., 2016) and has the same joint limits and physical parameters.

Many tasks can be solved by expressing a desired goal state or desired goal state components. For example, the 7-Dof reacher solves the task when the end effector XYZ component of its state is equal to the goal location, (x^*, y^*, z^*). One advantage of using a goal-conditioned model f as in Equation (8) is that this desire can be accounted for directly: if we already know the desired values of some components in s_{t+T}, then wen can simply fix those components of s_{t+T} and optimize over the other dimensions. For example for the 7-Dof reacher, the optimization problem in Equation (8) needed to choose an action becomes

$$a_t = \underset{a_t, s_{t+T}[0:14]}{\operatorname{argmax}} \; r_c(f(s_t, a_t, s_{t+T}[0:14] || [x^*, y^*, z^*]))$$

where $||$ denotes concatenation; $s_{t+T}[0:14]$ denotes that we only optimize over the first 14 dimensions (the joint angles and velocities), and we omit a_{t+T} since the reward is only a function of the state. Intuitively, this optimization chooses whatever goal joint angles and joint velocities make it easiest to reach (x^*, y^*, z^*). It then chooses the corresponding action to get to that goal state in T time steps. We implement the optimization over $s[0:14]$ with stochastic optimization: sample 10,000 different vectors and choose the best value. Lastly, instead of optimizing over the actions, we use the policy trained in DDPG to choose the action, since the policy is already trained to choose an action with maximum Q-value for a given state, goal state, and planning horizon. We found this optimization scheme to be reliable, but any optimizer can be used to solve Equation (8),(7), or (6).

Pusher: The state consists of 3 joint angles, 3 joint angular velocities, the XY location of the hand, and the XY location of the puck. The action controls torques for each of the 3 joints. The reward function is the negative Euclidean distance between the puck and the puck. Once the hand is near (with 0.1) of the puck, the reward is increased by 2 minus the Euclidean distance between the puck and the goal location. This reward function encourages the arm to reach the puck. Once the arm reaches the puck, bonus reward begins to have affect, and the arm is encouraged to bring the puck to the target.

As in the 7-DoF reacher, we set components of the goal state for the optimal control formulation. Specifically, we set the goal hand position to be the puck location. To copy the two-stage reward shaping used by our baselines, the goal XY location for the puck is initially its current location until the hand reaches the puck, at which point the goal position for the puck is the target location. Since there are no other states to optimize over, the optimal control problem is trivial.

Half-Cheetah: The environment is the same as in Brockman et al. (2016). The only difference is that the reward is the ℓ-2 norm between the velocity and desired velocity v^*. Our optimal control formulation is again trivial since we set the goal velocity to be v^*.