DIVIDE-AND-CONQUER REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Standard model-free deep reinforcement learning (RL) algorithms sample a new initial state for each trial, allowing them to optimize policies that can perform well even in highly stochastic environments. However, problems that exhibit considerable initial state variation typically produce high-variance gradient estimates for model-free RL, making direct policy or value function optimization challenging. In this paper, we develop a novel algorithm that instead optimizes an ensemble of policies, each on a different "slice" of the initial state space, and gradually unifies them into a single policy that can succeed on the whole state space. This approach, which we term *divide-and-conquer RL*, is able to solve complex tasks where conventional deep RL methods are ineffective. Our results show that divide-and-conquer RL greatly outperforms conventional policy gradient methods on challenging grasping, manipulation, and locomotion tasks, and exceeds the performance of a variety of prior methods. Videos of policies learned by our algorithm can be viewed here.

1 INTRODUCTION

Deep reinforcement learning (RL) algorithms have demonstrated an impressive potential for tackling a wide range of complex tasks, from game playing (Mnih et al., 2015) to robotic manipulation (Levine et al., 2016; Kumar et al., 2016; Popov et al., 2017; Andrychowicz et al., 2017). However, many of the standard benchmark tasks in reinforcement learning, including the Atari benchmark suite (Mnih et al., 2013) and all of the OpenAI gym continuous control benchmarks (Brockman et al., 2016) lack the kind of diversity that is present in realistic environments. One of the most compelling use cases for such algorithms is to create autonomous agents that can interact intelligently with diverse stochastic environments. However, such environments present a major challenge for current algorithms. The most standard avenue for incorporating variability into RL training is to choose a stochastic initial state distribution for the underlying Markov decision process. Highly stochastic initial state distributions lead to high-variance policy gradient estimates, which in turn hamper effective learning. Similarly, variability can also be incorporated via picking a distribution over the goal state.

In this paper, we explore RL algorithms that are especially well-suited for tasks with a high degree of variability both initial and goal states. We argue that a large class of practically interesting real-world problems fall into this category, but current RL algorithms are poorly equipped to handle them, as illustrated in our experimental evaluation. Our main observation is that, for tasks with a high degree of initial state variability, it is often much easier to obtain effective solutions to individual parts of the initial state space and then merge these solutions into a single policy, than to solve the entire task as a monolithic stochastic MDP. To that end, we can partition the state distribution into a set of distinct "slices," and train a separate policy for each slice. For example, if we imagine the task of picking up a block with a robotic arm, different slices might correspond to different initial positions of the block. Similarly, for placing the block, different slices will correspond to the different goal positions. For each slice, the algorithm might train a different policy with a distinct strategy. By employing a combination of mutual KL-divergence constraints and supervised distillation, we can gradually merge these distinct policies into a single global policy, which succeeds on the entire distribution at convergence.

It may at first seem surprising that this procedure provides benefit. After all, if the final global policy can solve the entire task, then surely each local policy also has the representational capacity to capture a strategy that is effective on the entire initial state space. However, it is worth considering

that a policy in a reinforcement learning algorithm must be able to represent not only the final optimal policy, but also all of the intermediate policies during learning. By decomposing these intermediate policies over the different slices of the initial state space, our method enables effective learning even on tasks with very diverse initial state and goal distributions. Since variation in the initial state distribution leads to high variance gradient estimates, this strategy also benefits from the fact that gradients can be better estimated in the local slices leading to accelerated learning. Intermediate supervised distillation steps help share information between the local policies which accelerates learning for slow learning policies, and helps policies avoid local minima.

The main contribution of our paper is a reinforcement learning algorithm for tasks with diversity, which we term divide-and-conquer (DnC) reinforcement learning. We present our algorithm and motivate its mathematical formulation. A detailed comparative empirical evaluation of the introduced approach against standard reinforcement learning methods and prior techniques is presented. We demonstrate a substantial improvement in performance on torque-controlled robotic grasping and manipulation, as well as on a variety of difficult locomotion scenarios.

2 Related Work

Prior work has addressed reinforcement learning tasks requiring diverse behaviors, both in locomotion (Heess et al., 2017) and manipulation (Osa et al., 2016; Kober et al., 2012; Rajeswaran & V. Kumar, 2017; Andrychowicz et al., 2017; Nair et al., 2017). However, these methods typically make a number of simplifications, such as the use of demonstrations to help guide reinforcement learning (Osa et al., 2016; Kober et al., 2012; Rajeswaran & V. Kumar, 2017; Nair et al., 2017), or the use of a higher-level action representation, such as Cartesian end-effector control for a robotic arm (Osa et al., 2016; Kober et al., 2012; Andrychowicz et al., 2017; Nair et al., 2017). We show that our proposed algorithm can solve a complex grasping task that requires picking up an object from various positions. We further demonstrate the strength of DnC by learning behaviours directly in the low-level torque action space, without the need of demonstration or high-level action representation. The selection of benchmark tasks in this work is significantly more complex than those leveraging cartesian position action spaces in prior work (Andrychowicz et al., 2017; Nair et al., 2017) or the relatively simple picking setup proposed by Popov et al. (2017), which consists of minimal task variation and a variety of additional shaping rewards. In the domain of locomotion, we show that our approach substantially outperforms the direct policy search method proposed by Heess et al. (2017). Curriculum learning has been applied to similar problems in reinforcement learning, with approaches that require the practitioner to design a sequence of progressively harder subsets of the initial state distribution, culminating in the original task (Asada et al., 1996; Karpathy & van de Panne, 2012). Our method allows for arbitrary decompositions, and we further show that DnC can work with automatically generated decompositions without intervention in Appendix D.

Our method is related to guided policy search (GPS) algorithms (Levine & Koltun, 2013; Levine et al., 2016). These algorithms train several "local" policies by using a trajectory-centric reinforcement learning method, and a single "global" policy, typically represented by a deep neural network, which attempts to mimic the local policies. The local policies are constrained to the global policy, typically via a KL-divergence constraint. Our method also trains local policies, though the local policies are themselves represented by more flexible, nonlinear neural network policies. Use of neural network policies significantly improves the representational power of the individual controllers and facilitates the use of various off-the-shelf reinforcement learning algorithms. Furthermore, we constrain the various policies to one another, rather than to a single central policy, which we find substantially improves performance, as discussed in Section 5.

Along similar lines, Teh et al. (2017) propose an approach similar to GPS for the purpose of transfer learning, where a single policy is trained to mimic the behavior of policies trained in specific domains. Our approach resembles GPS, in that we decompose a single complex task into local pieces, but also resembles Teh et al. (2017), in that we use nonlinear neural network local policies. Although Teh et al. (2017) propose a method intended for transfer learning, we adapted it to our problem formulation for comparison. We present results demonstrating that our approach substantially outperforms the method proposed by Teh et al. (2017).

3 PRELIMINARIES

An episodic Markov Decision Process (MDP) is classically modelled as $\mathbf{M} = (\mathcal{S}, \mathcal{A}, P, r, \rho)$ where $\mathcal{S}, \mathcal{A}$ are continuous sets of states and actions respectively. $P(s', s, a)$ is the transition probability distribution, $r : \mathcal{S} \to \mathbb{R}$ is the reward function, and $\rho : S \to \mathbb{R}_+$ is the initial state distribution. We consider an alternative MDP formulation, where the initial state distribution is conditioned on some variable ω, which we refer to as a "context." Formally, $\Omega = (\omega_i)_{i=1}^n$ is a finite set of contexts, and $\rho : \Omega \times S \to \mathbb{R}_+$ is a joint distribution over contexts ω and initial states s_0. One can imagine that sampling initial states is a two stage process: first, contexts are sampled as $\rho(\omega)$, and then initial states are drawn given the sampled context as $\rho(s|\omega)$. Note that for an arbitrary MDP, one can embed the context into the state as an additional state variable with independent distribution structure that factorizes as $P(s_0, \omega) = P(s_0|\omega)P(\omega)$. We clarify at this point that this formulation of an MDP with context is unrelated to a similarly named "contextual MDP" (Hallak et al., 2015). We hope to find a stochastic policy $\pi : \mathcal{S}, \mathcal{A} \to \mathbb{R}_+$ under which the expected reward of the policy $\eta(\pi) = \mathbb{E}_{\tau\sim\pi}[r(\tau)]$ is maximized.

4 DIVIDE-AND-CONQUER REINFORCEMENT LEARNING

In this section, we derive our divide-and-conquer reinforcement learning algorithm. We first motivate the approach by describing a policy learning framework for the contextual MDP setting described above, and then introduce a practical algorithm that can implement this framework for complex reinforcement learning problems.

4.1 LEARNING POLICIES IN MDPS WITH CONTEXT

The addition of contextual structure allows for two immediate extensions of the MDP $\mathbf{M}$. First, we consider an augmented MDP $\mathbf{M}'$ where each state contains information about the context $(\mathcal{S} \times \Omega, \mathcal{A}, P, r, \rho)$; a trajectory in this MDP is $\tau = ((\omega, s_0), a_0, (\omega, s_1), a_1, \dots)$. We also consider the class of context-restricted MDPs: for a context ω, we have $\mathbf{M}_\omega = (\mathcal{S}, \mathcal{A}, P, r, \rho_\omega)$, where $\rho_\omega(s) = \mathbb{P}(s|\Omega = \omega)$; i.e. the context is always fixed to ω .

A stochastic policy π in the augmented MDP $\mathbf{M}'$ decouples into a family of simpler stochastic policies $\pi = (\pi_i)_{i=1}^n$, where $\pi_i : \mathcal{S}, \mathcal{A} \to [0, 1]$, and $\pi_i(s, a) = \pi((\omega_i, s), a)$. Notice that π_i is a policy for the context-restricted MDP $\mathbf{M}_{\omega_i}$, and that a family of optimal policies in the class of context-restricted MDPs induces an optimal policy π in $\mathbf{M}'$, and vice versa.This implies that policy search in the augmented MDP reduces into policy search in the context-restricted MDPs.

Given a stochastic policy in the augmented MDP $(\pi_i)_{i=1}^n$, we can induce a stochastic policy π_c in the original MDP, by defining $\pi_c(s, a) = \sum_{\omega\in\Omega} p(\omega|s)\pi_\omega(s, a)$, where $p(\cdot|s)$ is a belief distribution of what context the trajectory is in. From here on, we refer to π_c as the central or global policy, and π_i as the context-specific or local policies.

Our insight is that it is important for each local policy to not only be good for its designated context, but also be capable of working in other contexts. This is because it might be difficult to accurately infer the context at deployment time in the original MDP, and local policies that work more broadly can compensate for errors in context estimation. Requiring that local policies be capable of working broadly allows for sharing of information so that local policies designated for difficult contexts can bootstrap their solutions off easier contexts. In the extreme case, where local policies generalize well to many other contexts, we obtain policies that are capable of operating even if no context information is provided, as required in the original MDP $\mathbf{M}$.

In order to find the optimal policy for the original MDP, we search for a policy $\pi = (\pi_i)_{1\dots n}$ in the augmented MDP that maximizes $\eta(\pi) - \alpha\mathbb{E}_\pi[D_{KL}(\pi\|\pi_c)]$: maximizing expected reward for each instance while remaining close to a central policy, where α is a penalty hyperparameter. This encourages the central policy π_c to work for all the contexts, thereby transferring to the original MDP. Using Jensen's inequality to bound the KL divergence between π and π_c, we minimize the right hand side as a bound for minimizing the intractable KL divergence optimization problem.

$$\mathbb{E}_\pi[D_{KL}(\pi\|\pi_c)] \leq \sum_{i,j} \rho(\omega_i)\rho(\omega_j)\mathbb{E}_{\pi_i}[D_{KL}(\pi_i\|\pi_j)] \tag{1}$$

4.2 THE DIVIDE-AND-CONQUER REINFORCEMENT LEARNING ALGORITHM

We now present a policy optimization algorithm which takes advantage of this contextual starting state, in a manner illustrated by the previous section. For a task with predefined contexts (either intrinsic to the problem or hand-selected), we find a good policy π_c by finding local policies $(\pi_i)_{i=1}^n$ that maximize expected reward in the individual contexts, while remaining constrained to be similar to one another. We modify policy gradient algorithms for optimizing local policies with these requirements. Policy gradient methods directly optimize the parameters of a stochastic policy through local gradient-based methods. Taking gradient steps using the score function gradient estimator yields the REINFORCE algorithm (Williams, 1992), whereas taking the curvature of the parameter space into account is the basis for Natural Policy Gradients (Kakade, 2002). We base our algorithm on Trust Region Policy Optimization, TRPO, (Schulman et al., 2015), a policy gradient method which takes gradient steps according to a surrogate loss

$$\mathcal{L}(\pi) = -\mathbb{E}_{\pi_{old}}\left[A(s,a)\frac{\pi(a|s)}{\pi_{old}(a|s)}\right] \tag{2}$$

while constraining the mean divergence from the old policy to a fixed constant. We choose TRPO for its practical performance on high-dimensional continuous control problems, but our procedure extends easily to other policy gradient methods as well.

In our framework, we optimize $\eta(\pi) - \alpha\mathbb{E}_\pi[D_{KL}(\pi\|\pi_c)]$, where α determines the relative balancing effect of expected reward and divergence. Using the bound obtained in (1), we have a surrogate loss of form

$$\mathcal{L}(\pi_1 \dots \pi_n) = -\sum_{i=1}^{n}\mathbb{E}_{\pi_{i,old}}\left[A(s,a)\frac{\pi_i(a|s)}{\pi_{i,old}(a|s)}\right] + \alpha\left(\sum_{i,j}\rho(\omega_i)\rho(\omega_j)\mathbb{E}_{\pi_i}\left[D_{KL}(\pi_i\|\pi_j)\right]\right) \tag{3}$$

The KL divergence terms constrain each local policy π_i to be close to the other π_j on its own context, while encouraging the policy to resemble other local policies on their respective tasks. This decomposition is evident when looking at terms that a π_i contributes to in (4). This formulation has interesting nuances: the gradient for π_i uses trajectories from context ω_i, but the constraint on other contexts also enforces dependence on the data from all $\omega_{1\dots n}$. Thus, despite only taking actions in a restricted context, the policy is trained with data from the full context distribution.

$$\mathcal{L}(\pi) \propto_{\pi_i} \underbrace{-\mathbb{E}_{\pi_{i,old}}\left[A(s,a)\frac{\pi_i(a|s)}{\pi_{i,old}(a|s)}\right]}_{\text{Maximizes } \eta(\pi_i)} + \alpha\rho(\omega_i)\sum_j \rho(\omega_j)\Big(\underbrace{\mathbb{E}_{\pi_i}[D_{KL}(\pi_i\|\pi_j)]}_{\text{Constraint on own context}} + \underbrace{\mathbb{E}_{\pi_j}[D_{KL}(\pi_j\|\pi_i)]}_{\text{Constraint on other contexts}}\Big) \tag{4}$$

On each iteration, trajectories from each context-restricted MDP $\mathbf{M}_{\omega_i}$ are collected using π_i. Each local policy π_i takes a conjugate gradient step (as in TRPO) with the surrogate loss in succession. Unlike performing joint optimization over all the local policies, the alternating scheme ensures that varied levels of reward signals do not cause any local policy to dominate another in the update process. It should be noted that the cost of evaluating and optimizing the KL constraints grows quadratically with the number of contexts, as the number of penalty terms is quadratic. For practical tasks however, the number of contexts will be in the order of 5-10, and the quadratic cost imposes minimal overhead on the TRPO policy gradient step.

The trajectory collection and policy optimization process is repeated for a certain number of iterations, ultimately resulting in local policies $(\pi_i)_{i=1}^n$ that have been trained on their individual contexts. We must now retrieve π_c, a central policy in the original task from the local policies, which is done by minimizing the KL divergence between π and π_c. This problem neatly simplifies into a maximum likelihood problem with samples from all the various policies.

$$\mathcal{L}_{center}(\pi_c) = \mathbb{E}_\pi[D_{KL}(\pi(\cdot|s)\|\pi_c(\cdot|s))] \propto \sum_i \rho(\omega_i)\mathbb{E}_{\pi_i}\left[-\log\pi_c(s,a)\right] \tag{5}$$

If π_c performs inadequately on the full task, we repeat the training of the local policies, $\pi_1 \dots \pi_n$; again initializing all the π_i to start at π_c.We repeat this local policy search and global policy optimization until convergence. The algorithm is laid out fully in pseudocode below.

```
function DnC(Contexts ω_1 ... ω_n)
    T ← Number of Training Loops
    R ← Distillation Period
    Randomly initialize central policy π_c
    for T iterations do
        Set π_i = π_c for all i = 1 ... n
        for R iterations do
            Collect trajectories T_i in context ω_i using policy π_i for all i = 1 ... n
            for all local policies π_i do
                Take gradient step in surrogate loss L wrt π_i
        Minimize L_center w.r.t. π_c using previously sampled states (T_i)_{i=1}^n
    return π_c
```

5 Experimental Evaluation

We focus our analysis on tasks spanning two different domains – manipulation and locomotion. Manipulation tasks involve handling a free object with a robotic arm and locomotion tasks involve tackling challenging terrains. We illustrate a variety of behaviors in both settings. Tasks were designed to bring out complex contact rich behaviors in settings with considerable variation in required behaviors. All of our environments are designed and simulated in Mujoco (Todorov et al., 2012).

Our experiments and analysis aim to address the following questions:

1. Can DnC solve highly complex tasks in a variety of domains, especially tasks that cannot be solved with current conventional policy gradient methods?
2. How does the form of the constraint on the ensemble policies in DnC affect the performance of the final policy, as compared to previously proposed constraints?

We compare **DnC** to the following prior methods and ablated variants:

- **TRPO.** TRPO (Schulman et al., 2015) represents a state-of-the-art policy gradient method, which we use for the standard RL comparison without decomposition into contexts. TRPO is provided with the same batch size as the sum of the batches over all of the policies in our algorithm, to ensure a fair comparison.
- **Distral.** Originally formulated as transfer learning in a discrete action space (Teh et al., 2017), we extend Distral to our stochastic initial state continuous control setting, where each context ω is a different task. For proper comparison between the algorithms, we port Distral to the TRPO objective, since empirically TRPO outperforms other policy gradient methods in this domain. This algorithm, which resembles the structure of guided policy search, also trains an ensemble of policies, but constrains them at each gradient step against a single global policy trained with supervised learning, and omits the distillation step that our method performs every R iterations (we use $R = 100$ in our experiments).
- **Unconstrained DnC.** We run the DnC algorithm without any KL constraints. This reduces to running TRPO to train policies $(\pi_i)_{i=1}^n$ on each context, and distilling the resulting local policies every R iterations.
- **Centralized DnC.** Whereas DnC doesn't perform inference on context, centralized DnC uses an oracle to perfectly identify the context ω from the state s. The resulting algorithm is equivalent to the Distral objective, but distills every R steps.

Performance of these methods is highly dependent on choice of α, the penalty hyperparameter which tunes how tightly coupled the local policies are. For each task, we run a hyperparameter sweep for each method, showing results for the best penalty. Furthermore, performance of policy gradient methods like TRPO vary significantly from run to run, and we run each experiment with 5 random seeds, reporting mean statistics and standard deviations. The experimental procedure is detailed more extensively in Appendix A.

6 ROBOTIC MANIPULATION

For robotic manipulation, we simulate the Kinova Jaco, a 7 DoF robotic arm with 3 fingers. The agent receives full state information, which includes the current absolute location of external objects such as boxes. The agent uses low-level joint torque control to perform the required actions. Note that use of low-level torque control significantly increases complexity, as raw torque control on a 7 DoF arm requires delicate movements of the joints to perform each task. We describe the specific tasks below, and full details may be found in Appendix B.

Picking. The *Picking* task requires the Jaco to pick up a small block and raise it as high as possible. The agent receives reward only when the block is in the agent's hand.

The starting position of the block is randomized within a fixed 30cm by 30cm square surface on the table, and we split the task into four contexts, each partition representing a corner of the square. Picking up the block from different locations within the workspace require diverse poses, making this task challenging in the torque control framework. TRPO can only solve the picking task from a 4cm by 4cm workspace, and from wider configurations, the Jaco fails to grasp with a high success rate with policies learnt via TRPO.

Lobbing. The *Lobbing* task requires the Jaco to flick a block into a target box, which is placed far enough that the arm cannot reach it directly. The agent receives reward at the end of an episode proportional to block flicking quality. The location of the target box is randomized over a 1m by 1m square, and we partition it into the four quadrants of the square that the target will be in. This problem inherits many challenges from the picking task. Furthermore, the sequential nature of grasping and flicking necessitates that information pass temporally and requires synthesis of multiple skills.

Catching. In the *Catching* task, a ball is thrown at the robot, and the arm must catch it in the air. Fixed reward is awarded every step that the ball is in or next to the hand. The initial position and velocity of the ball are randomized for each trajectory; we partition the task into quadrants based on the initial position of the ball. This task is particularly challenging due the temporal sensitivity of the problem. The end-effector needs to be in perfect sync with the flying object successfully finish the grasp. This extreme temporal dependency renders stochastic estimates of the gradients ineffective in guiding the learning.

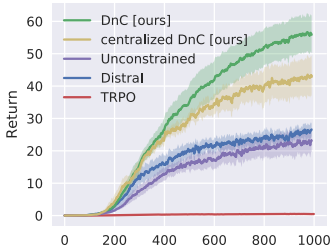
DnC exceeds the performance of the alternative methods on all of the manipulation tasks, as shown in Figure 1. For each task, we include the average reward, as well as a success rate measure, which provides a more interpretable impression of the performance of the final policy. TRPO by itself is unable to solve any of the tasks, with success rates below 10% in each case. The policies learned by TRPO are qualitatively reasonable, but lack the intricate details required to address the variability of the task.

TRPO fails because of the high stochasticity in the problem and the diversity of optimal behaviour for various initial states, because the algorithm cannot make progress on the full task with such noisy gradients. When we partition the manipulation tasks into contexts, the behavior within each context is much more homogeneous.

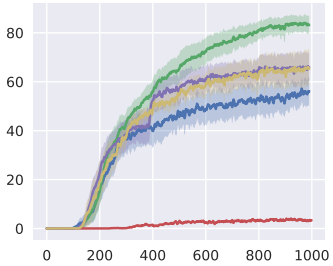
Figure 1 shows that, when trained with contexts, DnC outperforms both the adapted Distral variant and the two ablations of our method. On the picking task, DnC has a 16% higher success rate than the next best method, which is an ablated variant of DnC, and on the lobbing task it is better by 10%. Both the pairwise KL penalty and the periodic reset in DnC appear to be crucial for the algorithm's performance. In contrast to the methods that share information exclusively though a single global policy, the pairwise KL terms allow for more efficient information exchange. On the Picking task, the centralized variant of DnC struggles to pick up the object pockets along the boundaries of the contexts, likely because the local policies differ too much in these regions, and centralized distillation is insufficient to produce effective behavior.

On the catching task (Figure 1c), the baselines which are distilled every 100 iterations (the DnC variants) all perform well, whereas Distral lags behind. The policy learned by Distral grasps the ball from an awkward orientation, so the grip is unstable and the ball quickly drops out. Since Distral does not distill and reset the local policies, it fails to escape this local optimal behaviour.

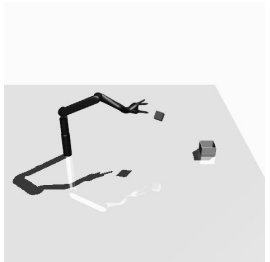
(a) *Picking task*



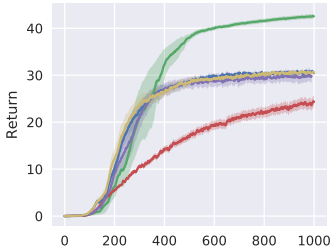
(b) Average Return on *Picking*



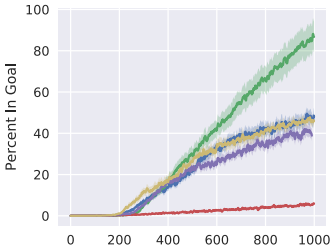
(c) Success rate on *Picking*



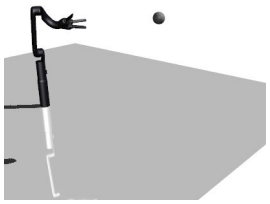
(d) *Lobbing task*



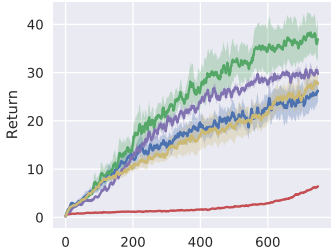
(e) Average Return on *Lobbing*



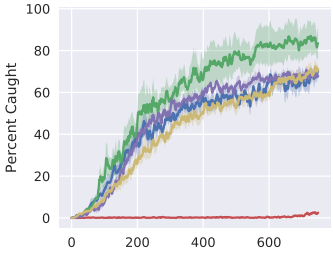
(f) Success rate on *Lobbing*



(g) *Catching task*

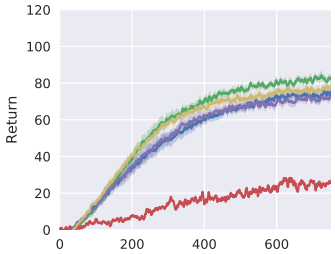


(h) Average Return on *Catching*

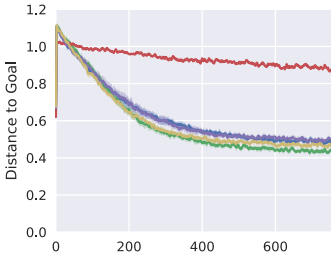


(i) Success rate on *Catching*

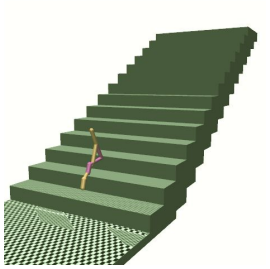
(j) *Ant*



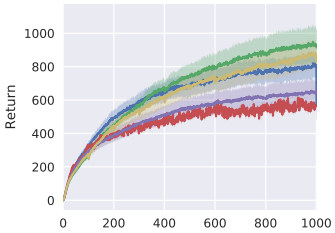
(k) Average Return on *Ant*



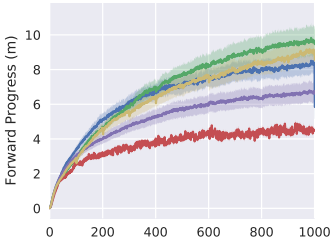
(l) Distance from goal on *Ant*



(m) *Stairs*



(n) Average Return on *Stairs*



(o) Forward Progress on *Stairs*

Figure 1: Average returns and success rate on Picking, Lobbing, Catching, Ant, and Stairs. On all of the tasks, DnC RL achieves the best results. On the Stairs and Ant tasks, DnC performs comparably to the centralized variant, while on the Picking, Lobbing, and Catching task the full algorithm outperforms all others by a wide margin. All of the experiments are shown with 5 random seeds.

	Picker	Lobber	Catcher	Ant Position	Stairs
TRPO	0.5 ± 0.2	23.5 ± 1.2	6.6 ± 1.3	18.6 ± 2.7	503.3 ± 61.2
Distral	26.9 ± 2.9	30.7 ± 0.7	25.8 ± 2.6	76.5 ± 3.0	810.4 ± 78.0
Unconstrained	45.7 ± 9.9	30.2 ± 1.1	28.9 ± 0.2	71.3 ± 3.1	645.6 ± 129.4
Centralized DnC (ours)	43.3 ± 6.2	30.2 ± 0.5	28.0 ± 2.3	78.8 ± 3.6	$\mathbf{869.6 \pm 74.5}$
DnC (ours)	$\mathbf{57.3 \pm 5.7}$	$\mathbf{42.6 \pm 1.0}$	$\mathbf{37.6 \pm 4.5}$	$\mathbf{83.7 \pm 2.7}$	$\mathbf{930.7 \pm 99.2}$

Table 1: Overall performance comparison between DnC and competing methods, based on final average return. Performance varies from run to run, so we run each experiment with five random seeds. For each of the tasks, the best performing method is DnC or centralized DnC.

Although the choice of context in these experiments all follow a standard grid decomposition, performance of DnC under other decomposition schemes are also of interest. In Appendix D, we evaluate DnC on automatically generated decompositions in the *Picking* task via K-means clustering on the initial state space. We find that DnC performs well on the automated partitioning scheme, on par with manually chosen contexts. Furthermore, performance increases and variance in performance decreases as the number of contexts are increased.

7 Locomotion

Our locomotion tasks involve learning parameterized navigation skills in two domains.

Ant Position In the *Ant Position* task, the quadruped ant is tasked with reaching a particular goal position; the exact goal position is randomly selected along the perimeter of a circle 5m in radius for each trajectory. We break the goal targets into four contexts of equal arc length. The ant is penalized for its distance from the goal every timestep. Although moving the ant in a single direction is solved, training an ant to walk to an arbitrary point is difficult because the task is symmetric and the global gradients may be dampened by noise in several directions.

Stairs In the *Stairs* task, a planar (2D) bipedal robot must climb a set of stairs, where the stairs have varying heights and lengths. The agent is rewarded for forward progress. Unlike the other tasks in this paper, there exists a single gait that can solve all possible heights, since a policy that can clear the highest stair can also clear lower stairs with no issues. However, optimal behavior that maximizes reward will maintain more specialized gaits for various heights. We create five contexts, each corresponding to a stair height interval of equal size, since the stair height affects gait more than stair length. The agent locally observes the structure of the environment via a perception system that conveys the information about the height of the next step. This task is particularly interesting because it requires the agent to compose and maintain various gaits in an diverse environment with rich contact dynamics.

As in manipulation, we find that DnC performs either on par or better than all of the alternative methods on each task. TRPO is able to solve the Ant task, but requires 400 million samples, whereas variants of our method solve the task in a tenth of the sample complexity. Initial behaviour of TRPO has the ant moving in random directions throughout a trajectory, unable to clearly associate movement in a direction with the goal reward. On the *Stairs* task, TRPO learns to take long striding gaits that perform well on shorter stairs but cause the agent to trip on the taller stairs, because the reward signal from the shorter stairs is much stronger. In DnC, by separating the gradient updates by context, we can mitigate the effect of a strong reward signal on a context from affecting the policies of the other contexts.

We notice large differences between the gaits learned by the baselines and DnC on the *Stairs* task. DnC learns a striding gait on shorter stairs, and a jumping gait on taller stairs, but it is clearly visible that the two gaits share structure. In contrast, the other partitioning algorithms learn hopping motions that perform well on tall stairs, but are suboptimal on shorter stairs, so brittle to context.

8 Discussion and Future Work

In this paper, we proposed divide-and-conquer reinforcement learning, an RL algorithm that separates complex tasks into a set of local tasks, each of which can be used to learn a separate policy. These separate policies are constrained against one another to arrive at a single, globally coherent solution, which can then be used to solve the task from any initial state. Our experimental results show that divide-and-conquer reinforcement learning substantially outperforms standard RL algorithms that samples initial and goal states from their respective distributions at each trial, as well as previously proposed methods that employ ensembles of policies. For each of the domains in our experimental evaluation, standard policy gradient methods are generally unable to find a successful solution.

Although our approach improves on the power of standard reinforcement learning methods, it does introduce additional complexity due to the need to train ensembles of policies. Sharing of information across the policies is accomplished by means of KL-divergence constraints, but no other explicit representation sharing is provided. A promising direction for future research is to both reduce the computational burden and improve representation sharing between trained policies with both shared and separate components. Exploring this direction could yield methods that are more efficient both computationally and in terms of experience.

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. *CoRR*, abs/1707.01495, 2017.

Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23(2): 279–303, May 1996. ISSN 1573-0565. doi: 10.1023/A:1018237008823. URL https://doi.org/10.1023/A:1018237008823.

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

A. Hallak, D. Di Castro, and S. Mannor. Contextual Markov Decision Processes. *ArXiv e-prints*, February 2015.

Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments. *CoRR*, abs/1707.02286, 2017.

Sham M Kakade. A natural policy gradient. In *Advances in neural information processing systems*, pp. 1531–1538, 2002.

Andrej Karpathy and Michiel van de Panne. *Curriculum Learning for Motor Skills*, pp. 325–330. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-30353-1. doi: 10.1007/978-3-642-30353-1_31. URL https://doi.org/10.1007/978-3-642-30353-1_31.

Jens Kober, Katharina Mülling, and Jan Peters. Learning throwing and catching skills. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pp. 5167–5168, 2012.

Vikash Kumar, Emanuel Todorov, and Sergey Levine. Optimal control with learned local models: Application to dexterous manipulation. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 378–383. IEEE, 2016.

Sergey Levine and Vladlen Koltun. Guided policy search. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pp. 1–9, 2013. URL http://jmlr.org/proceedings/papers/v28/levine13.html.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end learning of deep visuomotor policies. *Journal of Machine Learning Research (JMLR)*, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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.

Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel V. Todorov. Interactive control of diverse complex characters with neural networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 28*, pp. 3132–3140. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5764-interactive-control-of-diverse-complex-characters-with-neural-networks.pdf.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. *CoRR*, abs/1709.10089, 2017.

Takayuki Osa, Jan Peters, and Gerhard Neumann. Experiments with hierarchical reinforcement learning of multiple grasping policies. In *Proceedings of the International Symposium on Experimental Robotics (ISER)*, 2016.

Ivaylo Popov, Nicolas Heess, Timothy P. Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin A. Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *CoRR*, abs/1704.03073, 2017.

A. Rajeswaran and J. Schulman E. Todorov S. Levine V. Kumar, A. Gupta. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *ArXiv e-prints*, 2017.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.

Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *CoRR*, abs/1707.04175, 2017. URL http://arxiv.org/abs/1707.04175.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pp. 5026–5033. IEEE, 2012. ISBN 978-1-4673-1737-5. URL http://dblp.uni-trier.de/db/conf/iros/iros2012.html#TodorovET12.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

A EXPERIMENTAL DETAILS

To ensure consistency, all the methods tested are implemented on the TRPO objective function, allowing for comparisons between the various types of constraint. In particular, the Distral algorithm is ported from a soft Q-learning setting to TRPO. TRPO was chosen as it outperforms other policy gradient methods on challenging continuous control tasks. To properly compare TRPO to the partition-based methods for sample efficiency, we increase the number of timesteps of simulation used per policy update for TRPO. Explicitly, if B is the number of timesteps simulated used for a single local policy iteration in DnC, and N the number of local policies, then we use $B*N$ timesteps for each policy iteration in TRPO.

Stochastic policies are parametrized as $\pi_\theta(a|s) \sim \mathcal{N}(\mu_\theta(s), \Sigma_\theta)$. The mean,$\mu_\theta(\cdot)$, is a fully-connected neural network with 3 hidden layers containing $150, 100$, and 50 units respectively. Σ is a learned diagonal covariance matrix, and is initially set to $\Sigma = I$.

The primary hyperparameters of concern are the TRPO learning rate $\bar{D}_{KL}$ and the penalty α. The TRPO learning rate is global to the task; for each task, to find an appropriate learning rate, we ran TRPO with five learning rates $\{.0025, .005, .01, .02, .04\}$. The penalty parameter is not shared across the methods, since a fixed penalty might yield different magnitudes of constraint for each method. We ran DnC, Centralized DnC, and Distral with five penalty parameters on each task. The penalty parameter with the highest final reward was selected for each algorithm on each task. Because of variance of performance between runs, each experiment was replicated with five random seeds, reporting average and SD statistics.

	Picker	Lobber	Catcher	Ant Position	Stairs
State Space Dimension	34	40	34	146	41
Action Space Dimension	7	7	7	8	6
# Steps per Local Iteration	30000	30000	30000	50000	50000
# Iterations	1000	1000	500	500	1000
Learning Rate	.01	.02	.02	.01	.02

B TASK DESCRIPTIONS

All the tasks in this work have the agent operate via low-level joint torque control. For Jaco-related tasks, the action space is 7 dimensional, and the control frequency is 20Hz. For target-based tasks, instead of using the true distance to the target, we normalize the distance so the initial distance to the target is 1.

$$\text{normalized distance to target} = \frac{\text{distance to target}}{\text{initial distance to target}}$$

Picking The observation space includes the box position, box velocity, and end-effector position. On each trajectory, the box is placed in an arbitrary location within a 30cm by 30cm square surface of the table. We partition the initial state space into four quadrants: each context corresponds with a quadrant of the square surface. More precisely, if the initial box locations were given by $[0, 30] \times [0, 30]$, then the partitions are

$$\{[0, 15] \times [0, 15], [0, 15] \times [15, 30], [15, 30] \times [0, 15], [15, 30] \times [15, 30]\}$$

$$R(s) = \mathbf{1}\{\text{Box in air and Box within 8cm of Jaco end-effector}\}$$

Lobbing. The observation space includes the box position,box velocity, end-effector position, and target position. On each trajectory, the target location is randomized over a 1m by 1m square. Just as in the *Picking* task, we partition the initial state space into four quadrants according to the target position.

An episode runs until the box is lobbed and lands on the ground. Reward is received only on the final step of the episode when the lobbed box lands; reward is proportional to the box's time in air,t_{air}, and the box's normalized distance to target, d'_{target}.

$$R(s) = t_{air} + 40 \max(0, 1 - d'_{target})$$

Catching. The observation space includes the ball position, ball velocity, and end-effector position. Both the ball position and velocity are randomized every trajectory; we partition only according to the initial ball position. The initial state space is broken up into four contexts, by the quadrant of the initial ball position, just as in the *Picking* task.

$$R(s) = \mathbf{1}\{\text{Ball in air and Ball within 8cm of Jaco end-effector}\}$$

Ant Position. The target location of the ant is chosen randomly on a circle with radius 5m. We partition the initial state space according to the angle of this target position θ. For the four partitions, these corresponds to angles in

$$\left\{(0, \frac{\pi}{2}), (\frac{\pi}{2}, \pi), (\pi, \frac{3\pi}{2}), (\frac{3\pi}{2}, 2\pi)\right\}$$

The reward function takes into account the normalized distance of the ant to target, d'_{target}, and as with the standard quadruped, the magnitude of torque,$\|a\|$, and the magnitude of contact force, $\|c\|$.

$$R(s, a) = 1 - d'_{target} - 0.01\|a\| - 0.001\|c\|$$

Stairs. The planar bipedal robot has a perception system which is used to communicate the local terrain. The height of the platforms are given at 25 points evenly spaced from 0.5 meters behind the robot to 1 meters in front. On each trajectory, the heights of stairs are randomized between 5 cm and 25 cm, and lengths randomized between 50 cm and 60 cm. We partition based on the heights of the stairs, breaking the task into five equal contexts.

$$\{(5, 9), (9, 13), (13, 17), (17, 21), (21, 25)\}$$

The reward weighs the forward velocity v_x, and the torque magnitude $\|a\|$.

$$R(s, a) = v_x - 0.5\|a\| + 0.01$$

C ORACLE-BASED ABLATIONS

Whereas DnC maintains a global policy to run on all contexts, we consider in this section ablations whose final output is an ensemble of local policies, choosing the appropriate policy on each trajectory via oracle.

Final Local Policies We run the DnC algorithm, and return the ensemble of final local policies instead of the resulting global policy. This method is expected to outperform DnC, since the global policy should be strictly worse than the local ensemble. However, as seen in the table below, the gap in performance is low for the majority of tasks, showing that minimal information is lost in transferring from the ensemble of local policies to the global policy.

	Picker	Lobber	Catcher	Ant Position	Stairs
Final Global Policy (DnC)	57.3 ± 5.7	42.6 ± 1.0	37.6 ± 4.5	83.7 ± 2.7	930.7 ± 99.2
Final Local Policies	58.7 ± 4.9	42.7 ± 0.4	46.6 ± 2.9	85.7 ± 1.9	1070 ± 140.1

No Distillation We run the DnC algorithm, discarding the distillation step every R iterations. This is equivalent to training local policies with pairwise KL constraints till convergence, and considering the resulting ensemble of local policies. We notice that DnC significantly outperforms the variant without distillation on three of the tasks, and has equivalent performance on the other two. We hypothesize this is because the local policies often become trapped in local minima, and the distillation step helps adjust the policy out of the optima. This is consistent with observations in previous work involving trajectory optimization (Mordatch et al., 2015), where adding a central neural network to which trajectories were distilled significantly increased performance.

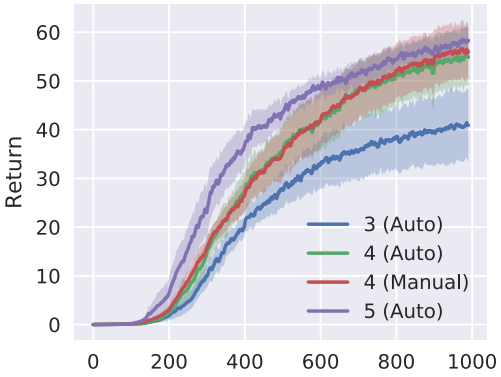
	Picker	Lobber	Catcher	Ant Position	Stairs
Distillation (DnC)	57.3 ± 5.7	42.6 ± 1.0	37.6 ± 4.5	83.7 ± 2.7	930.7 ± 99.2
No Distillation	30.2 ± 6.8	40.6 ± 1.0	20.3 ± 2.7	69.8 ± 1.4	919.5 ± 28.2

D AUTOMATED PARTITIONING

DnC relies on a pre-existing partition of the initial state space of the task. Although partitions were manually generated in the empirical study, we show in this section that results with automatically generated partitions yield similar scores.

To generate partitions, we sample 10,000 initial states from the task, and run K-means clustering to partition the initial state space into K contexts, then sampling from the induced partition. We compare DnC run on our manual partition scheme to results on generated partitions of three,four, and five contexts on the *Picking* task.

	Average Return
3 Partitions (Auto)	42.5 ± 7.1
4 Partitions (Manual)	57.3 ± 5.7
4 Partitions (Auto)	56.1 ± 6.1
5 Partitions (Auto)	58.7 ± 1.7



The generated decomposition into four contexts performs on par with the manual partitioning. Partitioning into three contexts yields lower rewards, and with higher variance in performance, as the stochasticity within each context is relatively high. Partitioning into five contexts yields slightly better performance on average, and with significantly less variance in performance, although there are diminishing returns as the number of partitions is increased.