Composing Meta-Policies for Autonomous Driving Using Hierarchical Deep Reinforcement Learning

Richard Liaw[2], Sanjay Krishnan[2], Animesh Garg[2,3], Daniel Crankshaw[2], Joseph E. Gonzalez[2], Ken Goldberg[1,2]

***Abstract*— Rather than learning new control policies for each new task, it is possible, when tasks share some structure, to compose a "meta-policy" from previously learned policies. This paper reports results from experiments using Deep Reinforcement Learning on a continuous-state, discrete-action autonomous driving simulator. We explore how Deep Neural Networks can represent meta-policies that switch among a set of previously learned policies, specifically in settings where the dynamics of a new scenario are composed of a mixture of previously learned dynamics and where the state observation is possibly corrupted by sensing noise. We also report the results of experiments varying dynamics mixes, distractor policies, magnitudes/distributions of sensing noise, and obstacles. In a fully observed experiment, the meta-policy learning algorithm achieves 2.6x the reward achieved by the next best policy composition technique with 80% less exploration. In a partially observed experiment, the meta-policy learning algorithm converges after 50 iterations while a direct application of RL fails to converge even after 200 iterations.**

I. Introduction

Consider the problem of designing a cruise control policy for a car. The manufacturer has an analytic model for a control policy π_0 that works on new cars. However, the manufacturer notices that as older cars wear, the controller is no longer effective. The manufacturer then collects data for a prototypical 5-year-old car at the end of the warranty cycle; identifies the system from data, and arrives at a policy π_5. Given a car of the same make/model but with an unknown age, how can the manufacturer leverage π_0 and π_5 to control it?

One approach would be to ignore π_0 and π_5 entirely, and apply a model-free control approach, such as Reinforcement Learning, to directly learn a new policy. While this may in principle converge to the desired result, RL could spend an extensive amount of time exploring before learning. However, a 6-month old car's cruise control policy may not be very different from π_0. Instead of learning a policy from scratch, it may be better to selectively apply π_0 or π_5 in certain circumstances, e.g., at higher speeds the car behaves more like an old car than a new one. This issue can be addressed by a meta-policy since it will adaptively learn when to use π_0 or π_5. Instead of applying RL over the original action space, we can apply RL to select between the two choices of applying either π_0 or π_5. The reduction of action space to k (in general) discrete choices greatly reduces the complexity of many problems and allows for quicker convergence.

[1] IEOR, [2] EECS, UC Berkeley, CA USA; [3] CS, Stanford University, Stanford CA US; The AUTOLAB at UC Berkeley (automation.berkeley.edu); {rliaw, sanjaykrishnan, animesh.garg, jegonzal, goldberg}@berkeley.edu, crankshaw@eecs.berkeley.edu

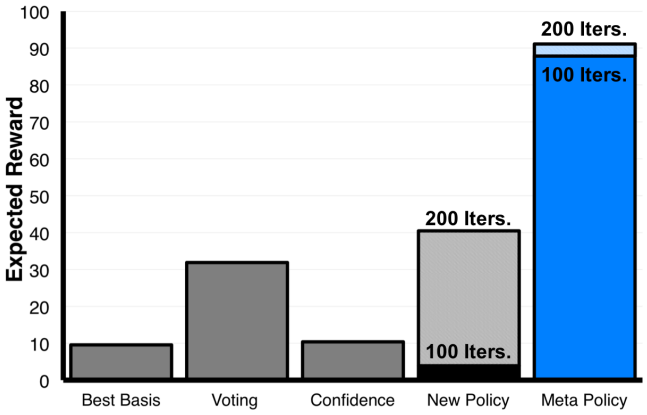

Fig. 1: We evaluate the use of meta policies on a car simulator from [1]. Two policies are trained on known dynamical regimes and the goal is to control the car given an unknown composition of these regimes. The best of the two policies achieves a reward of 9.55, a voting ensembling technique achieves 31.92, and a confidence ensembling technique achieves 10.32. Training a new policy with RL achieves a reward of 89.16 after 500 iterations of exploration, while meta policy learning achieves a comparable reward 87.90 after only 100 iterations.

Composing previously learned policies to address a novel task is an important problem in robot learning. One may be able to collect data for many specific tasks, e.g., driving in different conditions or over different terrain types. From this data, we train a set of *policies* to solve the specific tasks, and the **policy composition problem** is given a new task, learn a policy that is composed from the set. This has been studied as a variant of Hierarchical Reinforcement Learning (HRL) [12, 4, 8, 14, 26, 9], where a Reinforcement Learning algorithm learns a state-dependent meta-policy that switches between a set of closed-loop controllers called options. We take inspiration from this prior work and evaluate how these approaches can be integrated with new results from Deep Reinforcement Learning [19].

To formalize the problem, suppose we are given a set of policies $\{\pi_1, ..., \pi_k\}$ which we call *basis policies*, and all we know about each policy is that it is a function from a continuous state space S to discrete action set A. We assume that we can query each of the policies for actions at any state $s \in S$. Given a new task specified by a transition function $T(s,a)$ and reward function $R(s,a)$, the objective is to find an optimal meta policy $\pi^{meta} : S \mapsto \{1, \ldots, k\}$ that at each state selects one of the k policies to apply. Even though there are a discrete number of basis policies, the selection function can be quite complex and difficult to represent exactly. One solution is to parametrize this function π_θ^{meta} with a highly

expressive function class such as a Neural Network. Recent research in Deep RL has proposed many techniques to learn such parametrized policies [22].

One challenge in RL is the problem of partial observation in the form of dropped and/or noisy sensor measurements. In the cruise control example, consider the case when older cars have noisier velocity sensors. While RL is robust to process noise, sensing noise (where the robot cannot directly observe the state of the world) remains to be a challenge. This leads to policies that depend on history and not just the current state. Fortunately, intriguing new results in Recurrent Neural Networks (RNNs) can address this problem. RNNs are neural networks that have cycles, allowing it to capture temporal events that depend on the past. RNNs are particularly useful in the context of partial observation since the network can learn which previous states to store in memory. We use an RNN architecture based on Gated Recurrent Units (GRU) to represent the meta policies and the basis policies [7].

We experimentally evaluate these techniques in the context of a continuous-state, discrete-action driving simulator [1]. We explore how DL-based meta policies can address driving tasks where the dynamics of the new task are unknown but composed of a mixture of previously seen dynamics, when some of the provided policies are irrelevant or noisy, and when the state observation is corrupted by sensing noise (i.e., partial observation).

Figure 1 illustrates the potential benefits of using a meta-policy in a fully observed driving example. We construct a driving task where policies are trained on simulated cars with a steering bias (veers to one side) and a speed bias (goes faster than normal). Given a novel task, which is a mix of these two regimes, we compare directly applying RL against different techniques that compose the two policies. Results suggest that the meta-policy approach learns a composition that achieves a higher reward for the same amount of exploration compared to alternative composition techniques [27].

Our contributions are:

1) This paper explores the policy composition problem with Neural Networks in a driving simulator where the dynamics of the new task are an unknown composition of previously seen dynamics and the state observation is possibly corrupted by sensing noise.
2) We evaluate the extent to which this meta policy learning approach efficiently learns a viable composition, is robust to "distractor" policies, and performs in degenerate cases, in comparison to alternative policy composition techniques and direct policy learning using RL.
3) Our experimental results suggest that meta policy learning approaches are highly efficient in this new Neural Network setting since such policies can require large amounts of exploration. In a fully observed experiment, the meta policy learning approach achieves 2.6x of the reward by the next best policy composition technique and achieves this reward with 80% less exploration than directly applying Reinforcement Learning (ignoring previously trained policies). In a partially observed experiment, the meta policy learning approach converges after 50 iterations while the direct application of RL fails to converge even after 200 iterations.

II. Background and Related Work

This section outlines the basic problem setting and related work. We consider a finite-horizon Markov Decision Process (MDP), which is defined as a 5-tuple:

$$\mathcal{M} = \langle S, A, T(\cdot,\cdot), R(\cdot,\cdot), N \rangle.$$

S is the state-space, A is the action space, $T : S \times A \mapsto Prob(S)$ is the transition function that maps states and actions to a probability measure over subsequent states, $R : S \times A \mapsto \mathbb{R}$ is a reward function over the state and action space, and N is the number of discrete time-steps (time-horizon). A policy $\pi : S \mapsto A$ is a function that maps states S to actions A.

A number of variants (and simplifications) of the policy composition problems have been studied in prior work.

Best-In-Set Problem: Consider first the best-in-set problem where the agent must merely match the performance of the best policy in the set of given policies. Suppose, we have a set of policies $\{\pi_1, ..., \pi_k\}$ and an MDP $\mathcal{M}$, find the π_i with the highest expected reward. This problem can be solved with a Multi-Arm Bandit (MAB) algorithm, where each arm is a policy π_i and each pull is rolling the policy out and evaluating the expected reward [11]. Such ideas have been applied to robotics in works such as [18, 16, 25], where MABs are used to select from a library of state-machines or motion plans.

We focus on the case where the dynamics of the new scenario are unknown but composed of a mixture of previously seen dynamics. As seen in Figure 1, none of the policies individually may achieve a high reward, but a composition of them might do so. However, we do evaluate meta policy learning in the degenerate case where there exists a strong single basis policy (Section VI-C), and we also discuss how a hybrid MAB/RL formulation may apply in future work (Section VIII).

Ensembled Policies: Another approach to the composition problem is to have a *passive* switching condition, i.e., one that is not learned from exploration. For example, we can have a voting based system where all of the policies are queried at each state, and their actions are aggregated [27], in particular, actions are stochastically drawn proportional to the number of constituent policies that selected that action. In our simulated driving domain, we compare the meta policy approach to voting and find that the meta policy learning approach achieves 2.6x the reward. Another approach is a confidence-based switching rule, similar to that in [6] but applied in the context of LfD, where the robot selects the policy that has a higher confidence in that region of the state-space. In our simulated driving domain, we find that the meta policy learning approach achieves 8.7x the reward to this approach.

Hierarchical Reinforcement Learning (HRL): In HRL, policies can be constructed with closed loop sub-policies called options [12, 4, 8, 14, 26, 9]. Formally, an option is

defined as a three-tuple of $\langle I, \pi, B \rangle$, where I is the initiation set - the set of states where the option can be taken; $\pi : S \rightarrow A$ is a specific policy taken under this option, and $B : s \rightarrow [0,1]$ is a termination condition. An option can only be taken at s_t if $s_t \in I$. The policy composition problem is a special case of HRL where options terminate after each time-step (also called one-step options).

The HRL approach for policy composition has been applied since the 1990s [12, 13, 3, 14, 17, 20]. We highlight a few of the representative works. Kaelbing et al. studied voronoi decompositions of a continuous state-space to discretize an RL task [12]. Kalmar et al. and Asada et al. [13, 3] explored the discrete-state, discrete-action case experimentally on robots. Konidaris et al. studied composing segmented skills to perform a task [14].

We take inspiration from this prior work and evaluate how these approaches can be incorporated into a Deep RL framework. One of the new opportunities with Deep RL are Recurrent Neural Networks (RNN). These networks efficiently store temporal state and can capture policies that depend on history. This allows us to handle some problems with partial observation, e.g., sensing noise, which has always been a challenging problem in RL research [5].

Transfer in RL: Transfer Learning in RL has also been studied where the training happens on one task and then the goal is to apply the previous information to reduce training time in a novel scenario (see the survey by Taylor and Stone [26]). One of the first works that studied this problem was Selfridge et al. [23], who explored control for the inverted pendulum by first training on a lighter pole. The key difference is that Selfridge et al. studied the case where experience from prior training is transferred between tasks using a Q function and not policies, and others adopted similar transfer models [2, 24]. We hope to explore such models in more detail in future work, but this study focuses on a transfer model where policies are given as input.

A. Problem Statement

This paper explores the following problem in continuous-state, discrete-action driving simulator:

Problem 1 (Policy Composition Problem): Given a set $\Pi = \{\pi_1, ..., \pi_k\}$ **basis** policies defined over the same continuous state-space S and discrete action space A. Let π^{meta} be a meta policy, i.e., a policy that given a state $s \in S$ selects exactly one $\pi \in \Pi$:

$$\pi^{meta} : S \mapsto \{1, ..., k\}$$

For an MDP $\mathcal{M}$ with unknown dynamics, but known to be a mixture of previously seen dynamics used to train Π, and a given reward function $R(s,a)$, find the meta policy that optimizes the reward.

III. Fully Observed Algorithm

First, we describe the technique used to optimize the meta policy when the MDP is fully observed. This approach follows from prior work in Hierarchical RL [4], but we describe how we use a Multi-Layer Perceptron to represent the meta policy.

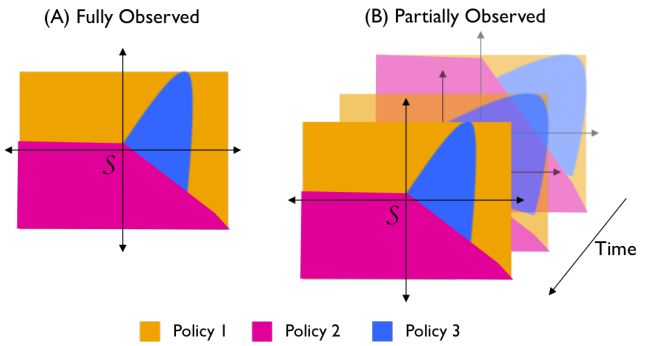

Fig. 2: (A) In fully observed case, meta policy can be thought of as partitioning the state-space S into distinct regions where each of the k basis policies is applied. (B) In partially observed case, meta policy is partitioning the history of the task S^t into such regions.

A. Parametrized Meta Policies

In general, π^{meta} can be difficult to represent exactly. One option is to consider parametrized functions π_θ^{meta} where $\theta \in \Theta$ represents one function in a parameter space. Let $\pi_\theta^{meta}(s)$ be a parametrized function mapping states to actions. Then, when $\pi_\theta^{meta}(s)$ is applied to M generates a distribution over N-step trajectories. Each $\pi_\theta^{meta}(s)$ has an expected reward:

$$R(\theta) = \mathbf{E}_{(s_n,a_n)\sim\pi_\theta^{meta}(s)}\left[\sum_{n=0}^{N} R(s,a)\right] \quad (1)$$

However, the choice of parametrization is important. A meta policy is a function that maps states to one of k basis policies. This can be thought of as partitioning the state-space into regimes associated with each policy (Figure 2A). However, these regimes may not define simply connected regions and actually may be quite complex. There may be multiple regions of the state-space that correspond to the same basis policy. In the cruise control example, it could be that a medium age car behaves like a new car at low speeds when driving straight (fewer effects of wear and tear), and high speeds when turning (turning dynamics dominate any small issues).

One solution is to use Neural Networks (NN) architectures to parametrically encode π_θ^{meta} and then optimize θ with respect to the reward in Eq. (1). For the fully-observed case, we apply a Multi-Layer Perceptron Neural Network. This network has two hidden layers (32 x 32), both with tanh nonlinearity functions and a softmax output layer to account for the domain's discrete action-space.

B. Trust-Region Policy Optimization (TRPO)

To learn the meta policy, we apply a version of a reinforcement learning algorithm called policy gradient. Policy gradient algorithms optimize the objective in Equation 1 with gradient ascent:

$$\theta^{(i+1)} = \theta^{(i)} + \lambda \cdot \nabla_{\theta^{(i)}} R(\theta^{(i)})$$

Since, we do not have access to the true distribution, only samples, we replace $R(\theta)$ with an empirical estimate $\tilde{R}(\theta)$:

$$\theta^{(i+1)} = \theta^{(i)} + \lambda \cdot \nabla_{\theta^{(i)}} \tilde{R}(\theta^{(i)})$$

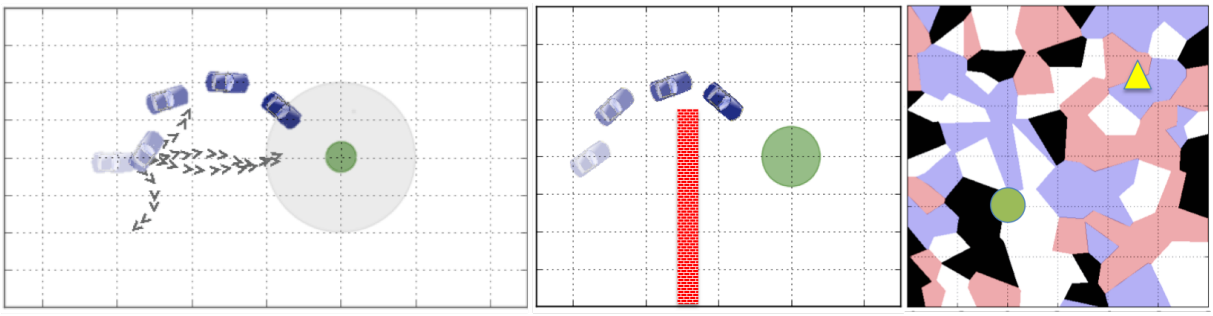

Fig. 3: **Domains**: In both Car-SD and Car-Barrier, agent observes (x, y, v, θ) state tuple. (A) In Car-SD, the car is in a 7 x 4 map and is tasked with driving to the goal state. The environment has two disjoint regions with different dynamics - one dynamics in a 1 unit radius (shaded) around the goal state, and the other dynamics in all other parts of the map. The dynamics of the domain are different in training basis policies than when training the meta policy. (B) In Car-Barrier, the car is in a 5 x 4 map and needs to navigate around the barrier to reach the goal. The dynamics are uniform dynamics throughout this map. (C) In the Hybrid Control environment, the agent is in a 12 x 12 map and needs to navigate from its starting position (triangle) to the green end goal through disjoint partitions of different dynamics. The partitions are formed from a voronoi mapping of 100 random points, each assigned with one of 4 dynamics as colorcoded above.

Policy gradient is initialized with a parameter $\theta^{(0)}$ and "rolls out" p trajectories from the policy $\pi_{\theta^{(0)}}^{meta}$ to calculate an estimated gradient of the reward function. However, the gradient estimates can have high variance, so one solution is to optimize this efficiently is to use a technique called Trust Region Policy Optimization [22]. Trust region methods take gradient steps in a small neighborhood around the current parameter value, clipping the size of the step if it exceeds a specified step-size parameter, preventing excessive oscillation due to noise.

IV. Partially Observed Algorithm

Next, we describe the partially observed setting and the meta policy learning algorithm with RNNs.

A. Partial Observation

The main challenge with partial observation is that optimal policies depend on history, not just the current state. In RL, any amount of un-modeled observation noise (as opposed to process noise) is considered partial observation. One of the new opportunities with Deep RL are Recurrent Neural Networks (RNN). These networks efficiently store temporal state and can capture policies that depend on history. We explore two scenarios: (1) meta policy is memory-less (MLP from previous section), and (2) meta policy is history dependent (RNN). Both scenarios use basis policies that are history-dependent (trained with an RNN). Figure 2b illustrates the difference between a history-dependent and memory-less meta policy.

B. Overview of Recurrent Networks

We briefly review the RNN architecture we use to represent both the basis policies and meta policies. As cyclic neural networks, RNNs are able to capture temporal events that depend on the past. They extract and encode relevant signal from the history in low dimensional embedding and are therefore particularly useful in the context of partial observations.

One challenge with RNNs is the *vanishing gradient* problem, where dependencies over long time-scales cannot be modeled well. The RNN architecture that we use is a Gated Recurrent Unit (GRU). A gated recurrent unit (GRU) was proposed by [7] to capture dependencies of different time scales and address this issue. GRUs require less data to train than the popular (Long Short Term Memory) LSTM solutions to the same problem. In all cases, the meta policy was trained using Trust Region Policy Optimization (step size of 0.001 and a linear feature baseline).

V. Experimental Setup

A. Experimental Settings

We implement meta policy learning with RLLab [10], which is a library for Deep RL. The library provides a comprehensive set of policy optimization algorithm implementations along with a suite of continuous control tasks for evaluation. We use the RLLab implementation of Trust Region Policy Optimization.

Evaluation Metrics – We use the average reward as the number of iterations increase as a metric of evaluation and comparison of meta policy against other baselines. Another metric we used to evaluate the framework was the number of iterations needed until the mean reward exceeds a given reward threshold with respect to the number of basis policies.

B. Experimental Environments

In two of our evaluation platforms, we used a remote control car simulator [1] in which a car learns to drive to a pre-defined target. We chose this domain because of its challenging non-holonomic movements, ease of visualization, intuitive behavior, and its ability to relate to highly complex problems with continuous state spaces. The remote control car domain has a four-dimensional state-space of x, y, v, θ, where (x, y) are the position, v is the velocity, and θ is the direction of the car. For any state, the car can take one of 9 actions, formed from pairs of (+0.03, 0, -0.03) velocity and (+30, 0, -30) degree turning angle. The main intention is to evaluate and characterize the reinforcement learning agent that chooses amongst policies trained on different dynamics.

1. Car with Split Dynamics (Car-SD) – In this environment (Figure 3a), the agent is in a 7 x 4 map and is tasked with driving to the goal state. We use two different dynamics

models, each located in a disjoint partition of the state space. The car is placed three units away from the goal which has a success radius of 0.2 units with a constraint that the car must be going slower near the goal - even in the region of the goal, it must have an absolute velocity less than 0.2 units in order to succeed.

One dynamics model D_1 has a constant position bias (modeling slippage) and the other model D_2 with a constant turning angle bias (modeling alignment issues). The partition with D_1 is located in a 1 unit radius around the goal position, and the rest of the environment has D_2. In this environment, the car needs to drive to the goal while passing through regimes with different dynamics. The time horizon is set as 500 steps. The car receives a penalty proportional to the distance from the goal for each step taken. The car receives a reward of -10 when it collides with the environment boundaries (walls) and a reward of 100 for reaching the goal. The meta policy for this environment uses two basis policies that are each trained on duplicates of this environment layout but with only one dynamics model present.

2. Car with Barrier (Car-Barrier) – We also construct a variant of the Car domain with obstacles. In this environment (Figure 3b), the car is placed in a 5 x 4 map, 3 units away from the goal state which has a success radius of 0.4. The obstacle environment ensures that there is a high penalty for taking a random action in an inopportune place, like right next to a barrier. This allows us to better characterize the policy recovery problem, where one policy can navigate around the obstacles and the others are random noise. We use a uniform dynamics model for the entire environment. The time horizon for the task is set as 200 steps. The car receives a step penalty of -1 for each step taken; a reward of 500 when inside the radius of the goal and a penalty of -100 when colliding with another structure (i.e., the barrier).

3. Hybrid Control Environment – In this environment (Figure 3c), the agent is in a 12 x 12 map and needs to navigate from its starting position to the goal state. We generate N linear dynamics models (A, B for $x_{t+1} = Ax_t + Bu_t$), where N is the number of controllers we will be giving the meta policy. For each (A,B), we solve for a gain matrix K that will be used as the controller ($x_{t+1} = (A - BK)x_t$). We take the 4 of the N dynamics models and use them exclusively in the different dynamics partitions. The partitions are formed from a voronoi mapping of 100 random points, each assigned with one of 4 dynamics. The reward function is a negative quadratic reward, and Gaussian noise is added. The agent begins at an initial state and needs to reach within a 0.4 radius around the goal state within 100 time-steps.

VI. Experimental Results

We present results in two domains, a simulated car driving domain (discrete actions) and hybrid linear control domain (continuous actions).

A. Baseline Comparison

In the first experiment, we test the following hypotheses: (1) meta policy learning learns a composition that achieves a higher reward than alternatives, (2) training a new policy with RL, disregarding the previously trained policies, requires more exploration to achieve the same reward as meta policy learning. We evaluate the frameworks on the Car-SD domain, which has two dynamics stitched together - D_1 (position bias) and D_2 (angle bias). Two basis policies for the meta policy are trained on 7 x 4 uniform dynamics maps with the start state and goal state at the same position. One policy is trained on a map with only D_1 and the other with only D_2. Training is done with Q-Learning ($\alpha = 0.95, \lambda = 0.4$) with a randomized Radial Basis Function Q-Function approximator of 2000 randomly placed kernels.

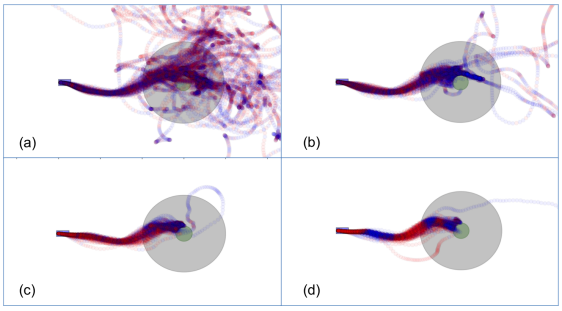

Fig. 4: Car-SD: Above are the traces of 100 rollouts, visualizing the mapping from state to subpolicy. From (a) to (d), each map shows the current policy at 10, 50, 100, 500 iterations of training respectively. A policy with clear segments is drawn as the iterations increase, but it is interesting to note that the multiple policies are used in one dynamics region, including those that were not originally trained on that dynamics model.

We evaluate meta policy learning against the following baselines:

1) **Basis Policies:** We evaluate the reward earned by each of the individual basis policies if no more learning occurs.
2) **Voting:** We apply an ensembling technique where actions are stochastically drawn, proportional to the number of constituent policies that selected that action [27].
3) **Transition Estimator:** We motivate this example by using state transition history as a heuristic for applying the optimal basis policy for a certain region. Given that the policies are trained on dynamics, we generate 100 rollouts $T = [s_0, a_0, s_1, a_1, ...a_{n-1}, s_n]$. and fit a Kernel Density Estimator over (s_i, a_i, s_{i+1}) transition tuples. On the stitched domain, each agent is queried for a confidence estimate, and the action is stochastically drawn, proportional to the confidence estimate.
4) **New Policy:** We apply RL directly to the new domain and evaluate the number of iterations required before achieving a given reward.

Figure 1 illustrates the results. First, each of the basis policies applied individually does not achieve a high reward on the mixed domain. The ensembling techniques, voting and confidence, perform marginally better than the basis policies. Since these techniques are not adaptive, they have a fixed level of performance. The voting based estimator achieves 3x the reward of the best basis policy while the transition estimator achieves 10% better than the best basis policy. Next, we evaluate the adaptive approaches, meta policy learning

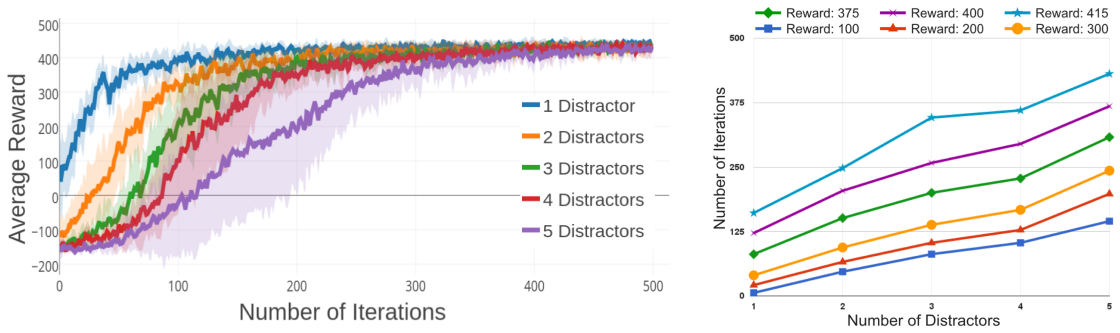

Fig. 5: **Distractors in Car-Barrier**: (a) We evaluate meta policy learning in its ability to reject distractor (random) policies. We increase the number of distractor policies, while maintaining one optimal policy and measure the number of iterations to convergence to several different reward levels. (b) Results suggest that it is possible to consistently recover a successful policy, even as the number of noisy agents are varied for roughly linearly more iterations.

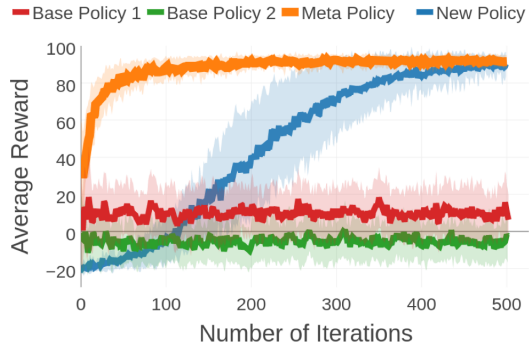

Fig. 6: **Distinct Dynamics**: Given only basis policies, the agent will not successfully reach the goal. Using TRPO, Meta RL quickly converges to a successful composition of the basis policies, significantly faster than training a policy from scratch.

and directly applying reinforcement learning. Meta policy learning converges after 100 iterations of TRPO (with a batch size of 1000, discount factor of 0.995, and step size of 0.001). In comparison, directly learning a policy requires 5x times as many iterations to converge to the same reward. In this case, having access to previously trained policies on the independent dynamical regimes avoids nearly 8000 trajectory rollouts.

Figure 6 illustrates the meta policy's convergence. Initially, the meta policy explores by trying both basis policies in different states (Figure 4a). As the meta policy converges, it learns when to apply each policy (Figure 4d). Figure 4d is interesting as it shows that even though there are only two dynamical regimes in the domain, the typical trajectory has three switches. In RL problems, due to the delayed reward, it is sometimes beneficial to take a locally suboptimal action (i.e., applying a policy trained on a different dynamics regime) to achieve a higher ultimate reward. In this case, the car leverages the angle-biased policy to guide itself to the right orientation.

B. Sensitivity to Noise

Next, we evaluate the Car-SD domain with sensing noise, a form of partial observation. We evaluate the convergence of meta policy learning as a function of the probability and magnitude of noise. We evaluate: (1) training a history-dependent meta policy (RNN), (2) training a memory-less meta policy (MLP), (3) training a new history-dependent meta policy with RL (RNN) (4) training a new memory-less meta policy with RL (MLP). We use TRPO for all training, and we train all policies with a batch size of 4000, discount factor of 0.995, and step size of 0.001.

Figure 8 displays the effect of noise on policy performance on the new policies. This illustrates that history dependent policies achieve a higher reward for larger amounts of noise. History-dependent policies are not strictly better since they inherently have a larger number of parameters and require more data to converge. For small amounts of noise, this tradeoff is in favor of the memory-less policies. On the other hand, as the noise rate increases, history-dependent policies are more accurate.

The surprising result is when we consider the meta policies. As in the fully observed case, meta policy learning converges to the maximum possible reward with less exploration than RL–exploiting the previously trained policies. The benefits of meta policy learning are even more pronounced under large amounts of observation noise. We find that even with 4x as many iterations, directly applying RL to learn a new policy (memoryless or history-dependent) fails to achieve the same performance as meta policy learning.

C. Distractor Policies

We now evaluate the degenerate case of meta policy learning, where out of the set of k policies, there is one viable policy, and all of the others are noise (take random actions)– and we call these random policies *distractors*. We evaluate two hypotheses: (1) meta policy learning will recover the near-optimal policy, (2) the amount of exploration to recover the policy is roughly linear in the number of distractors. These experiments were run on a variant of the car domain, Car-Barrier.

The viable basis policy was learned using Q-Learning over 1000 episodes with a Radial Basis Function approximator. We plot the results in Figure 5a-b. Results suggest that the algorithm can recover the viable policy despite with multiple distractors. With up-to 5 distractors, the meta policy trained with TRPO converges to the reward of the viable policy with no more than 500 iterations (with a batch size of 2000,

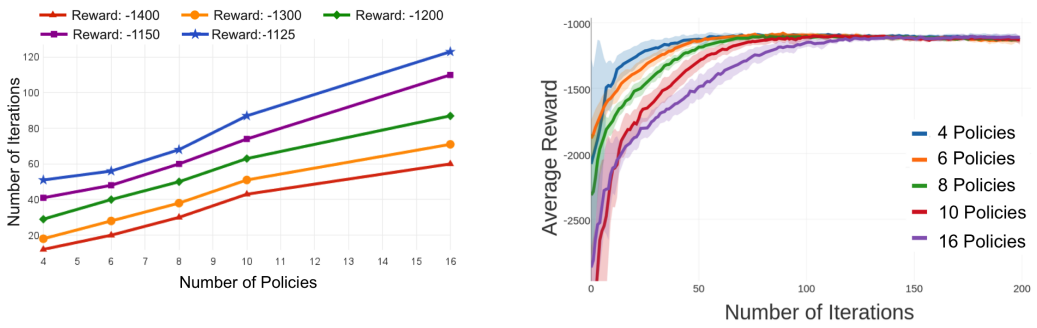

Fig. 7: **Multiple policies in Hybrid Control**: Meta Policy Learning not only converges in the hybrid control example but can also reject extraneous policies as in the discrete-action case.

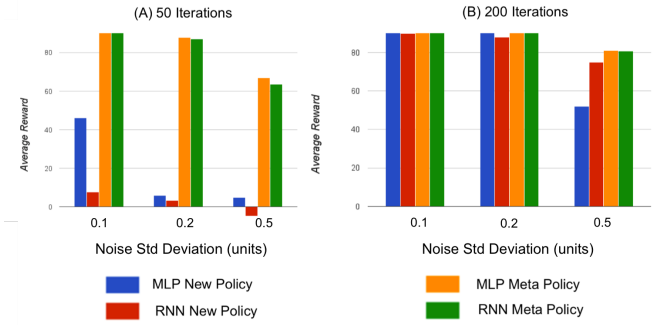

Fig. 8: **Sensitivity to Noise** We evaluate the Car-SD domain with sensing noise and vary the magnitude of noise. The meta policy learning approach converges in 50 iterations, while after 200 iterations the new policy approach has not yet converged. Surprisingly, we find that a memory-less meta policy (using a MLP) instead of an RNN has a faster convergence rate.

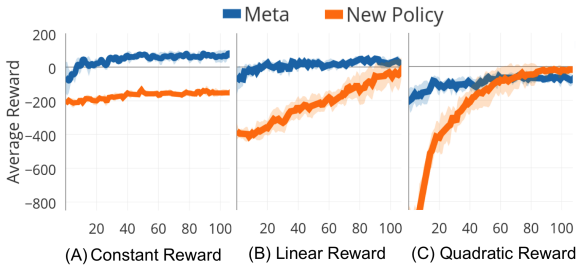

Fig. 9: Policy convergence graphs with respect to different reward strengths on Car-SD. We vary the strength of the reward in each experiment, (A) using a constant reward of -1, (B) a negative linear reward proportional to euclidean distance to goal, and (C) a negative quadratic reward.

discount factor of 0.995, and step size of 0.001). We also see that increasing the number of agents delay the convergence rate in a roughly linear fashion. Figure 5a shows the number of iterations to achieve a fixed reward.

D. Hybrid Control

Next, we evaluate meta policy learning in a continuous action space using the Hybrid Control environment. In spirit, this experiment is similar to the distractor policy experiment, but instead of random controllers, we use sub-optimal controllers. The experiment setup has disjoint partitions of 4 different dynamics models. No single controller can bring successfully reach the goal by itself. We run meta policy learning (with a batch size of 1000, discount factor of 0.995, and step size of 0.001) using up to 16 different controllers as the set of basis policies, where 4 of the controllers always being solutions to the four environment dynamics models. From Figure 7, we also observe a linear order of convergence.

VII. Conclusion

This paper presents an initial exploration of how deep reinforcement learning can be used to address the policy composition problem. We present results in a simulated driving domain consisting of varying dynamics mixes, distractor policies, magnitudes/distributions of sensing noise, and obstacles. In the fully observed setting, the meta policy learning approach achieves 2.6x of the reward by the next best policy composition technique and achieves this reward with 80% less exploration than directly applying Reinforcement Learning (ignoring previously trained policies). In the partially observed setting, meta policy learning converges to high-reward policies in roughly 50 iterations while the direct application of RL fails to converge even after 200 iterations.

VIII. Discussion & Future Work

It is well-known that the convergence rate of RL is highly dependent on how rewards are shaped [21]. A well-shaped reward can serve to guide the learner towards the optimal policy without much exploration. However, it may not be possible to design a well-shaped reward with the available data especially in sequential tasks where success depends on satisfying a sequence of conditions [15]. We provide a rough characterization of the effects of rewards in the Car-SD domain, shown in Figure 9. There is a clear increase in convergence rate of training the new RL policy as we increase the strength of the reward. Conversely, the evidence suggests that meta policy learning would provide the greatest benefit when rewards are sparse or delayed. Exploring the link between reward shaping and hierarchical reinforcement learning is an important avenue of future work.

One of the key arguments against meta policy learning is the potential optimality gap. Meta policy learning's objective is to find the best composite policy, which can achieve a significantly lower reward than the optimal policy for the MDP. Another direction for future work is to study this gap in further detail. We will explore hierarchical MDP approaches that allow for learning in the basis policies, perhaps reducing

this potential gap. Another important theoretical direction is to quantify this gap analytically for common problem types.

In terms of the lower bound, meta policy learning also can fall short. Consider the degenerate case when there exists a single policy that dominates throughout the state-space. The best-in-set problem can be solved with a multi-arm bandits (MAB) approach. When applied to the experiment in Section VI-C found that MAB approach is far more efficient than using RL. We evaluated the distractor policy experiment using an Upper Confidence Bound strategy and found that with 3 distractor policies this approach applies the correct policy within 4000 steps - which is two orders of magnitude faster than using hierarchical RL. We hope to explore using MAB to initialize policy gradient iterations such that the meta policy learning iterations at least begin with a performance no worse than the best basis policy.

ACKNOWLEDGMENT

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Sponsors. This research was performed in part in the AUTOLAB at UC Berkeley in affiliation with the AMP Lab, BDD, BAIR, the CITRIS "People and Robots" (CPAR) Initiative: http://robotics.citris-uc.org Multilateral Manipulation by Human-Robot Collaborative Systems and by Google, Cisco, Siemens, Cloudminds.

REFERENCES

[1] G. A. (2013). [Online]. Available: Rc-cardomain

[2] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda, "Vision-based behavior acquisition for a shooting robot by using a reinforcement learning." Citeseer.

[3] M. Asada, E. Uchibe, S. Noda, S. Tawaratsumida, and K. Hosoda, "Coordination of multiple behaviors acquired by a vision-based reinforcement learning," in *IROS*, 1994.

[4] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003.

[5] A. R. Cassandra, "A survey of pomdp applications," in *AAAI*, vol. 1724. Citeseer, 1998.

[6] S. Chernova and M. Veloso, "Confidence-based policy learning from demonstration using gaussian mixture models," in *AAMS*. ACM, 2007, p. 233.

[7] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[8] G. Comanici and D. Precup, "Optimal policy switching algorithms for reinforcement learning," in *AAMS*, 2010, pp. 709–714.

[9] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *JAIR*, vol. 13, pp. 227–303, 2000.

[10] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continous control," in *ICML*, 2016.

[11] K. G. Jamieson, M. Malloy, R. D. Nowak, and S. Bubeck, "lilúcb: An optimal exploration algorithm for multi-armed bandits." in *COLT*, 2014.

[12] L. P. Kaelbling, "Hierarchical learning in stochastic domains: Preliminary results," in *ICML*, 1993.

[13] Z. Kalmár, C. Szepesvári, and A. Lörincz, "Module-based reinforcement learning: Experiments with a real robot," *Machine Learning*, vol. 31, no. 1-3, pp. 55–85, 1998.

[14] G. Konidaris and A. G. Barto, "Efficient skill learning using abstraction selection." in *IJCAI*, vol. 9, 2009, pp. 1107–1112.

[15] S. Krishnan, A. Garg, R. Liaw, L. Miller, F. T. Pokorny, and K. Goldberg, "Hirl: Hierarchical inverse reinforcement learning for long-horizon tasks with delayed rewards," *arXiv preprint arXiv:1604.06508*, 2016.

[16] M. Laskey, J. Mahler, Z. McCarthy, F. Pokorny, S. Patil, J. van den Berg, D. Kragic, P. Abbeel, and K. Goldberg, "Multi-arm bandit models for 2d sample based grasp planning with uncertainty," in *CASE. IEEE*, 2015.

[17] L.-J. Lin, "Hierarchical learning of robot skills by reinforcement," in *Neural Networks, 1993., IEEE International Conference on*. IEEE, 1993, pp. 181–186.

[18] P. Matikainen, P. M. Furlong, R. Sukthankar, and M. Hebert, "Multi-armed recommendation bandits for selecting state machine policies for robotic systems," in *ICRA*. IEEE, 2013, pp. 4545–4551.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[20] J. Morimoto and K. Doya, "Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning," *RAS*, vol. 36, no. 1, pp. 37–51, 2001.

[21] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, 1999.

[22] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR, abs/1502.05477*, 2015.

[23] O. Selfridge, R. Sutton, and A. Barto, "Training and tracking in robotics."

[24] S. P. Singh, "Transfer of learning by composing solutions of elemental sequential tasks," *Machine Learning*, 1992.

[25] V. Srivastava, P. Reverdy, and N. E. Leonard, "Surveillance in an abruptly changing world via multiarmed bandits," in *CDC*, 2014.

[26] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *JMLR*, vol. 10, no. Jul, pp. 1633–1685, 2009.

[27] M. A. Wiering and H. Van Hasselt, "Ensemble algorithms in reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics*, 2008.