Learning to Execute: Efficient Learning of Universal Plan-Conditioned Policies in Robotics

Anonymous Author(s)
Affiliation
Address
email

Abstract

Applications of Reinforcement Learning (RL) in robotics are often limited by high data demand. On the other hand, approximate models are readily available in many robotics scenarios, making model-based approaches like planning a data-efficient alternative. Still, the performance of these methods suffers if the model is imprecise or wrong. In this sense, the respective strengths and weaknesses of RL and model-based planners are complementary. In the present work, we investigate how both approaches can be integrated into one framework that combines their strengths. We introduce Learning to Execute (L2E), which leverages information contained in approximate plans to learn universal policies that are conditioned on plans. Using robotic manipulation as an example, we show that L2E exhibits both increased sample efficiency and success rates when compared to pure RL and increased success rates when compared to planning.

1 Introduction

A central goal of robotics research is to design intelligent machines that can solve arbitrary and formerly unseen tasks while interacting with the physical world. Reinforcement Learning (RL) (Sutton & Barto, 2018) is a generic framework to automatically learn such intelligent behavior with little human engineering. Still, teaching an RL agent to actually exhibit general-purpose problem-solving behavior is, while possible in principle, prohibitive in practice. This is due to practical restrictions including limited computational resources and limited data availability. The latter limitation is particularly dramatic in robotics, where interaction with the physical world is costly.

On the other hand, for many robotics scenarios, there is a rough model of the environment available. This can be exploited, e.g., using optimization-based planning approaches (Mordatch et al., 2012; Kuindersma et al., 2016; Toussaint et al., 2018), and potentially offers a more data-efficient way to reason about an agent's interaction with the world. Still, if the model is incorrect or imprecise, directly following the resulting plan will not be successful.

The present work starts from the observation that both pure RL approaches and pure planning approaches have strengths and limitations that are fairly complementary. RL makes no assumptions about the environment but is data-hungry, and model-based planning generally implies model simplifications but is data-efficient. For robotic manipulation tasks, it seems natural to try and integrate both approaches into one framework that combines the strengths of both. In the present work we seek to address the open question of how this can be achieved.

We introduce a novel approach that we call Learning to Execute (L2E). Our approach translates sparse-reward *goal-conditioned* Markov Decision Processes (MDPs) (Bellman, 1957) into *plan-conditioned* MDPs. L2E exploits a simple planning module to create crude plans, which are then used to teach any off-the-shelf off-policy RL agent to execute them. L2E makes use of final-volume-preserving

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

reward shaping (FV-RS) (Schubert et al., 2021), allowing it to train a universal plan-conditioned policy with high data efficiency. The contributions of this work are:

- We introduce L2E, which uses RL to efficiently learn to execute approximate plans from a model-based planner. To this end, we describe formally how FV-RS can be used to translate goal-conditioned MDPs into plan-conditioned MDPs.
- We introduce plan replay strategies to efficiently learn universal plan-conditioned policies.
- We demonstrate using a robotic pushing example that L2E exhibits increased sample efficiency when compared to Hindsight Experience Replay (Andrychowicz et al., 2017), and higher success rates when compared to executing the plan using a learned inverse dynamics model.

We discuss work related to ours in section 2, explain background and notation in section 3, and introduce our method in section 4. We present our experimental results in section 5, and conclude with a discussion in section 6.

2 Related Work

2.1 Goal-Conditioned Policies

Goal-conditioned or universal policies (Kaelbling, 1993; Moore et al., 1999; Foster & Dayan, 2002; Schaul et al., 2015; Veeriah et al., 2018; Nasiriany et al., 2019) not only act based on the state the agent finds itself in, but also based on the goal it tries to achieve. Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) is a particularly efficient way to learn universal policies. Here, achieved outcomes of the agent's interaction with the environment are interpreted as desired goals in order to improve sample efficiency in sparse-reward settings.

L2E draws great inspiration from this work, but in contrast to HER, L2E learns a universal plan-conditioned policy. This means that the L2E policy in general can execute multiple plans leading to the same goal. Although this presents a more complex learning task, we show in our experiments that by incorporating plan information using plan-based FV-RS, the sample efficiency of L2E is significantly improved over HER.

2.2 Plan- and Trajectory-Conditioned Policies

Plan-conditioned policies create behavior that depends on plans that are input to the decision making. Lynch et al. (2020) learn plans and how to execute them from data generated by a human "playing" with a teleoperated robot. The resulting policy is conditional on a latent space of encoded plans. Our work differs from this paradigm in that human interaction is not needed. Both Lynch et al. (2020) and Co-Reyes et al. (2018) directly imitate a planned trajectory by maximizing its likelihood. In contrast, the plans used in the present work are not directly imitated. Using FV-RS guarantees that the fully trained L2E agent will reach its goal after finite time even if the plan provided is wrong. Guo et al. (2019) learn trajectory-conditioned policies to self-imitate diverse (optimal and suboptimal) trajectories from the agent's past experience.

We instead assume in this work that the plan is provided by an external model-based planner. This allows the L2E agent to use external information during training that can not be concluded from its own experience yet.

2.3 Learning from Demonstration

L2E learns how to execute plans in order to achieve different tasks. In this sense, it is related to Learning from Demonstration (LfD) techniques that exploit demonstrations when learning a task. Existing work (Argall et al., 2009; Hussein et al., 2017; Ravichandar et al., 2020) differs significantly both in how the demonstration examples are collected and how the policy is then derived. Taylor et al. (2011) derive an approximate policy from human demonstration, and then use this to bias the exploration in a final RL stage. Hester et al. (2017) train a policy on both expert data and collected data, combining supervised and temporal difference losses. Salimans & Chen (2018) use a single demonstration as starting points to which the RL agent is reset at the beginning of each episode. In

Cabi et al. (2019), demonstrations are combined with reward sketching done by a human. Interactive human feedback during training is another source of information used in Thomaz et al. (2006); Knox & Stone (2010). Christiano et al. (2016) consider a setting that is quite closely related to what we consider in the present work. Here, an inverse dynamics model is used to transfer knowledge from a policy in the source domain to a policy in the target domain.

Similarly, L2E applies the planner's knowledge to the target domain. In the experiments section, we therefore compare L2E against executing the plan using a learned inverse dynamic model. We find that L2E results in significantly higher success rates.

3 Background

3.1 Goal-Conditioned MDPs and RL

We consider settings that can be described as discrete-time MDPs $M = \langle \mathbb{S}, \mathbb{A}, T, \gamma, R, P_S \rangle$. $\mathbb{S}$ and $\mathbb{A}$ denote the set of all possible states and actions, respectively. $T : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to \mathbb{R}_0^+$ is the transition probability (density); $T(s'|s,a)$ is the probability of the next state being s' if the current state is s and a is chosen as the action. The agent receives a real-valued reward $R(s,a,s')$ after each transition. Immediate and future rewards are traded off by the discount factor $\gamma \in [0,1)$. $P_S : \mathbb{S} \to \mathbb{R}_0^+$ is the initial state distribution.

The goal of RL is to learn an optimal policy $\pi^* : \mathbb{S} \times \mathbb{A} \to \mathbb{R}_0^+$ that maximizes the expected discounted return. In other words, RL algorithms generally try to find

$$\pi^* = \operatorname*{argmax}_{\pi} \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s_{t+1} \sim T(\cdot|s_t,a_t),\, a_t \sim \pi(\cdot|s_t), s_0 \sim P_S} \left[R(s_t, a_t, s_{t+1}) \right] \tag{1}$$

from collected transition and reward data $D = \{(s_i, a_i, r_i, s_i')\}_{i=0}^n$. More specifically for this work, we are interested in applications in robotics, where both $\mathbb{S}$ and $\mathbb{A}$ are typically continuous. There exists a range of algorithms for this case. For the experiments in this paper, soft actor-critic (SAC) (Haarnoja et al., 2018) is used.

In a goal-conditioned MDP $M_G = \langle \mathbb{S}, \mathbb{G}, \mathbb{A}, T, \gamma, R_G, P_S, P_G \rangle$, the reward function $R_G(s,a,s',g)$ has an additional input parameter, the goal $g \in \mathbb{G}$. $P_G : \mathbb{G} \to \mathbb{R}_0^+$ is the distribution of goals. The optimal goal-conditioned policy π_G^* behaves optimally with respect to any of these goals.

3.2 Final-Volume-Preserving Reward Shaping

We use approximate plans as an additional source of information for the RL agent. For sparse-reward goal-driven MDPs, FV-RS (Schubert et al., 2021) offers an efficient way to include additional information by adding an additional term

$$R(s,a,s') \to R_{\text{FV}}(s,a,s') = R(s,a,s') + F_{\text{FV}}(s,a,s') \tag{2}$$

to the reward function, accelerating exploration. In general, the optimal policy π^* corresponding to the original MDP and the optimal policy π_{FV}^* corresponding to the shaped MDP will be different. FV-RS however restricts the allowed modifications $F_{\text{FV}}(s,a,s')$ in such a way that after finite time, the optimally controlled agent ends up in a subset of the volume in which it would have ended up without shaping. As a result, external information can be made available for the RL algorithm without changing the long-term behavior of the resulting optimal policy.

Specifically in the present work, we consider goal-conditioned MDPs in which the goal-conditioned reward R_G of the underlying MDP is either 1, if the goal is reached, or 0 everywhere else. We further assume that the L2E agent is given an external plan p, represented as an intended trajectory in state space. We intend to reward the agent for staying close to the plan, and for advancing towards the goal along the plan. A natural way of achieving this is to use a plan-based shaping reward (Schubert et al., 2021)

$$F_{\text{FV}}(s,a,s',p) = \frac{1 - R_G(s,a,s',f(p))}{2} \frac{k(s)+1}{L} \exp\left(-\frac{d^2(s, p_{k(s)})}{2\sigma^2} \right) \quad . \tag{3}$$

Here, $\sigma \in (0,\infty)$, $k(s) = \operatorname{argmin}_i(d(p_i, s))$, and $d(\cdot,\cdot)$ is a measure of distance in state space. For the pushing experiments discussed in this work, $d(\cdot,\cdot)$ is the euclidean distance in state space ignoring

the coordinates corresponding to the orientation of the box. F_{FV} assigns a shaping reward of at most $1/2$ for moving closer towards the end of the plan. For a sufficiently high discount factor γ, F_{FV} is final-volume preserving, meaning that the long-term behavior of the optimal agent is unchanged.

4 Learning to Execute

L2E considers goal-conditioned MDPs M_G (see section 3.1), for which an approximate planner Ω is available. L2E uses FV-RS to construct a corresponding plan-conditioned MDP M_P from a goal-conditioned MDP M_G and a planner Ω. In the following sections 4.1 to 4.3, we introduce our notion of a plan-conditioned MDP M_P and describe the components of the L2E algorithm. We then summarize the L2E algorithm in section 4.4.

4.1 Plan-Conditioned MDPs

Plans are provided by a model-based planner, which can be described as a distribution $\Omega : \mathbb{P} \times \mathbb{S} \times \mathbb{G} \to \mathbb{R}_0^+$ over a set of plans $\mathbb{P}$. Given an initial state and a goal, $\Omega(p|s, g)$ is the probability that the planner outputs p as a possible plan of how to achieve g from state s. The distinction between goals and plans is that plans are conditional on *both a goal and an initial state*. Therefore, both initial state and goal can be inferred using the plan only.

In a plan-conditioned MDP $M_P = \langle \mathbb{S}, \mathbb{P}, \mathbb{A}, T, \gamma, R_P, P_S, P_P \rangle$, a plan $p \in \mathbb{P}$ is given to the reward function $R_P(s, a, s', p)$ as an additional input parameter. $P_P : \mathbb{P} \to \mathbb{R}_0^+$ is the distribution of plans. The optimal plan-conditioned policy π_P^* behaves optimally with respect to any of these plans, creating a distribution $\pi_P^*(\cdot \mid a, p)$ over actions that is conditional on the current state and the current plan.

4.2 Constructing the Plan-Conditioned MDP

We use FV-RS to shape the reward function R_G of the original goal-conditioned MDP $M_G = \langle \mathbb{S}, \mathbb{G}, \mathbb{A}, T, \gamma, R_G, P_S, P_G \rangle$ with a plan-dependent term $F_{\text{FV}}(s, a, s', p)$ (see equation 3)

$$R_G(s, a, s', g) \to R_G^{\text{FV}}(s, a, s', g, p) = R_G(s, a, s', g) + F_{\text{FV}}(s, a, s', p) \quad . \tag{4}$$

A common property of the planner Ω is that the goal is encoded in the plan leading to it already, and can be inferred in a deterministic way. More precisely, there exists a surjective function $f : \mathbb{P} \to \mathbb{G}$ such that

$$\forall g \in \mathbb{G}\ \forall s \in \mathbb{S} : p \sim \Omega(\cdot|s, g) \Rightarrow g = f(p) \quad . \tag{5}$$

In this case, we can interpret the shaped reward function

$$R_P(s, a, s', p) = R_G^{\text{FV}}(s, a, s', f(p), p) \tag{6}$$

as a plan-conditioned reward function of a plan-conditioned MDP $M_P = \langle \mathbb{S}, \mathbb{G}, \mathbb{A}, T, \gamma, R_P, P_P \rangle$. The distribution over initial states and plans P_P of M_P is still missing, and can be constructed as

$$P_P(s, p) = \int \Omega(p|s, g) P_S(s) P_G(g) \mathrm{d}g \quad . \tag{7}$$

In practice, P_P can be sampled from by sampling $s \sim P_S$, $g \sim P_G$ and then subsequently sampling $p \sim \Omega(\cdot|s, g)$.

Thus, we have constructed a plan-conditioned MDP M_P by combining a goal-conditioned MDP M_G with an approximate planner Ω and a FV-RS shaping function F_{FV}. For reference later in this paper, we write as a shorthand notation $M_P = \mathcal{C}(M_G, \Omega, F_{\text{FV}})$. Furthermore, we will refer to M_P as the corresponding plan-conditioned MDP to M_G and vice versa.

In contrast to potential-based reward shaping (Ng et al., 1999), FV-RS does not leave the optimal policy invariant. As a result, generally $\exists p \in \mathbb{P} : \pi_G^*(\cdot|\cdot, f(p)) \not\equiv \pi_P^*(\cdot|\cdot, p)$. In words, the optimal policy of M_P and the optimal policy of M_G will not result in identical behavior. In fact, while $\pi_G^*(\cdot|\cdot, g)$ learns one policy for each goal g, $\pi_P^*(\cdot|\cdot, p)$ can learn different behavior for each plan in the set of plans $\{p \in \mathbb{P} \mid f(p) = g\}$ leading towards the same goal g.

4.3 Plan Replay Strategy

In order to efficiently learn a universal plan-conditioned L2E policy, the reward for experienced episodes is evaluated with respect to many different plans. In HER (Andrychowicz et al., 2017), it is assumed that each state $s \in \mathbb{S}$ can be assigned an achieved goal. Recorded episodes are then replayed with respect to goals that were achieved during the episode, i.e. the recorded transitions are re-evaluated with respect to these goals. This ensures that the recorded transitions were successful in reaching the replayed goals, resulting in highly informative data.

In L2E, transitions are replayed with respect to plans. However, there is no meaningful relation between each state $s \in \mathbb{S}$ and a unique "achieved plan". Therefore, the L2E agent replays transitions with past plans that were recorded at some point during training and were stored in its replay buffer D. The replay plans are chosen according to a plan replay strategy S_n.

A plan replay strategy S_n provides a distribution over n replay plans, conditioned on the replay buffer D and the buffer containing the current episode D_{ep} (see algorithm 1 for a definition of D and D_{ep}). For replay, n plans are sampled according to this strategy $\{p_1, \ldots, p_n\} \sim S_n(\cdot \mid D_{\text{ep}}, D)$. We consider two types of replay strategies. Uniform replay S_n^{uni} samples n unique plans uniformly from the replay buffer D. Reward-biased replay S_{nm}^{bias} first uniformly samples m unique plans from the replay buffer D, and then returns the n plans p_i that would have resulted in the highest sum of rewards $\sum_{(s_k, a_k, s'_k) \in D_{\text{ep}}} R_P(s_k, a_k, s'_k, p_i)$ for the episode stored in D_{ep}. The idea behind using reward-biased replay is to bias the replay towards transitions resulting in higher reward.

Algorithm 1: Learning to Execute (L2E)

Input : Goal-conditioned MDP M_G, approximate planner Ω, FV-RS shaping function F_{FV}, plan replay strategy S_n, off-policy RL Algorithm $\mathcal{A}$

Output : Universal plan-conditioned optimal policy π_P^* for the corresponding plan-conditioned MDP $M_P = \mathcal{C}(M_G, \Omega, F_{\text{FV}})$

```
Construct plan-conditioned MDP M_P = C(M_G, Ω, F_FV) as detailed in section 4.2;
Initialize replay buffer D ← {};
while π*_P not converged do
    Initialize episode buffer D_ep ← {};
    Sample initial state and goal (s_0, g) ~ P_G;
    Sample plan p ~ Ω(·|s_0, g);
    s ← s_0;
    while Episode not done do
        Sample action a ~ π*_P(· | s, p);
        Sample transition s' ~ T(· | s, a);
        Collect shaped reward r ← R_P(s, a, s', p);
        Add to episode buffer D_ep ← D_ep ∪ {(s, a, r, s', p)};
        s ← s';
    end
    Add episode to replay buffer D ← D ∪ D_ep;
    Get replay plans {p_1, ..., p_n} ~ S_n(· | D_ep, D);
    for p_replay in p_1, ..., p_n do
        for (s, a, r, s', p) in D_ep do
            Calculate replay reward r_replay ← R_P(s, a, s', p_replay);
            Add replayed transition to buffer D ← D ∪ {(s, a, r_replay, s', p_replay)};
        end
    end
    Update policy using off-policy RL algorithm π*_P ← A(π*_P, D)
end
```

4.4 L2E Algorithm

The L2E algorithm is outlined in algorithm 1. First, the corresponding plan-conditioned MDP $M_P = \mathcal{C}(M_G, \Omega, F_{\text{FV}})$ is constructed from the original goal-conditioned MDP M_G, the planner Ω and the shaping function F_{FV} as described in section 4.2. The agent acts in the environment trying

to follow one randomly sampled plan per episode. The episode is then added to the replay buffer, along with data from episode replays with respect to other plans. These other plans are sampled from the replay buffer according to the replay strategy S_n. A generic off-policy RL algorithm is used to update the agent using the replay buffer. This process is repeated until convergence.

We would like to emphasize that the L2E algorithm is agnostic to the exact type of off-policy RL algorithm. By combining state and plan into a "super state" for the purpose of passing the replay buffer to the off-policy RL algorithm, L2E can be interfaced with any off-the-shelf implementation.

5 Experiments

We evaluate the L2E agent against several baselines using a pushing task. This task combines unstable dynamics with challenging exploration. A video of the experiments is available in the supplementary material. The complete code to fully reproduce the figures in this paper from scratch can be found in the supplementary material as well[1]. This includes the implementation of the pushing environment, the implementation of the L2E agent, and the specific code used for the experiments in this paper.

The experiments section is structured as follows. In section 5.1 we discuss the pushing environment and planner that are used in the experiments. In section 5.2 we briefly discuss implementation details of our method. In section 5.3 we discuss the baselines against which we compare our method. In section 5.4 we discuss our experimental results.

5.1 Pushing Environment and Planner

Figure 1a shows a rendering of the pushing environment. We use the open-source Nvidia PhysX engine (phy, 2021) to simulate a box of size 0.4×0.4 being pushed on a table of size 3×3 by a spherical end effector of radius 0.06. The 10D state space of both the goal-conditioned MDP M_G and the corresponding plan-conditioned MDP M_P consists of the 3D position of the end effector, the 3D position of the box, and the 4D quaternion for the orientation of the box. The agent controls the 3D velocity of the end effector. The maximum velocity in any direction is 0.1 per time step. The results of the agent's actions are noisy. At each step, the actual change in position can be off up to 10% from the input velocity.

In the goal-conditioned MDP M_G, each goal is represented as a desired 2D box position on the table. The goal-dependent sparse reward function R_G is 1 if the box is within 0.1 of this desired goal, and 0 if not. The initial state-goal distribution P_G is uniform across the table for the non-colliding box position and goal position. The end effector is always initialized at the origin and the box is always initialized with a fixed orientation parallel to the table.

For the plan-conditioned MDP M_P, we use a crude manhattan-like planner Ω that deterministically outputs plans consisting of two separate contacts leading the initial state to the goal as shown in figure 1a. Plans are represented as a trajectory of length 50, consisting of 6D elements representing end effector position and box position. The orientation of the box is not specified in the plan.

We construct the plan-conditioned MDP M_P as described in section 4.2, using this planner and the FV-RS function in equation 3. We use the width parameter $\sigma = 0.5$ throughout the experiments.

5.2 L2E Implementation

We use the open-source Stable Baselines3 Raffin et al. (2019) SAC agent for all experiments. To allow for a fair comparison, we use the same hyperparameters for the SAC algorithm in all L2E and HER runs. Both L2E and the HER baseline are trained using rollout episodes of length 250.

We analytically encode the plans into a 4D latent space, which is then used to train the RL agent. For the actor and critic networks, we use 6 hidden layers with sizes decreasing geometrically from 1024 to 64. We train the networks using 0.0003 as the learning rate in batches of size 256. The replay buffer stores 1×10^6 transitions. The L2E and HER agents need less than 1GB of GPU memory and are trained using an Nvidia RTX-3090 GPU and AMD EPYC 7502 CPU. For full details on the implementation, please also refer to the code given in the supplementary material.

[1]After this paper is de-anonymized, we will additionally provide a link to our code on github here.

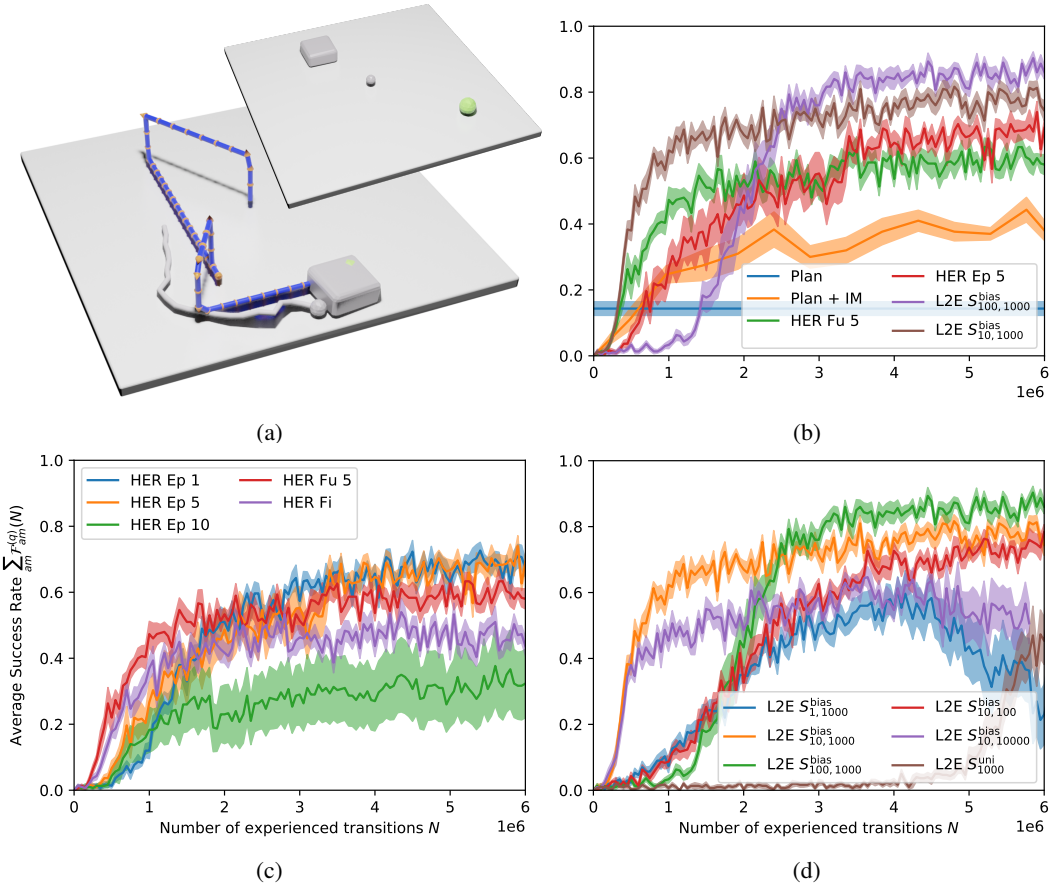

Figure 1: Results for the pushing environment. The lines represent averages over both test rollouts and across agents, the colored areas indicate the confidence intervals (standard deviations of the mean). Renderings created with open-source code by Orthey (2021). (a) The simulated pushing setup (inset). The spherical end effector tries to push the box to the green target. Once the box is pushed off the table, it can not be recovered. An approximate planner creates simple manhattan-like plans (blue line in large figure). The path of the box is also planned, but not shown here. The recent end effector movement is indicated in grey. The learned behavior can differ significantly from the original plan. (b) Comparison to the baselines introduced in section 5.3. L2E shows significantly higher sample efficiency when compared to HER, and significantly higher success rates when compared to executing the plan directly (Plan) or using an inverse dynamics model (Plan + IM). (c) To allow for a fair comparison, we compare different replay strategies for the HER baseline. We find that the "episode" replay strategy results in better long-term performance, but "future" and "final" generally allow for faster learning in the beginning. (d) For L2E, we find that using biased replay significantly improves sample efficiency. The L2E agents with uniform replay strategy S^{uni}_{1000} only catches up to the biased agents at $N \approx 8 \times 10^6$ (not shown).

5.3 Baselines

We compare L2E against several baselines that we briefly introduce in the following.

5.3.1 Plan Execution

As basic baseline, we compare against direct execution of the plan using a simple controller. Here, at each step the point p_k in the planned trajectory that is closest to the current position s is selected. To measure the distance, we use $d(\cdot, \cdot)$ introduced in section 3.2. The action is then calculated as the delta between the end effector position at this point and the following point p_{k+1} of the trajectory. We stop the run if either the goal or the time limit of 250 steps is reached.

5.3.2 Plan Execution with Inverse Dynamics Model

We use an inverse dynamics model to transfer the plan-based controller described in section 5.3.1 to the simulated environment. This can be seen as a transfer method as introduced by Christiano et al. (2016).

We learn an inverse dynamics model (IM) $\phi : \mathbb{S} \times \mathbb{S} \to \mathbb{A}$ from simulation data. From input pairs of state s and desired next state s', this model learns to predict feasible actions $a = \phi(s, s')$. To collect data, we reset the environment to random initial states, and then perform random actions. In order to bias the data collection towards more informative samples, an action leading the end effector directly towards the box is randomly selected at 10% of all time steps. After 250 time steps, we reset and repeat the process. We then train a neural network to predict actions from pairs of state and next state using mean squared error loss. We use the open-source Pytorch (Paszke et al., 2019) package.

Evaluation is done in the same way as described in section 5.3.1, with the only difference that at each step we calculate the action $a = \phi(s, p_{k+1})$ using the inverse dynamics model. Since the planned trajectory does not specify the box's orientation, we always assume the desired orientation to be parallel to the table.

5.3.3 HER

We use the open-source Stable Baselines3 Raffin et al. (2019) implementation of HER to learn universal goal-conditioned policies for the goal-conditioned MDP M_G. To allow for a fair comparison, we run several experiments comparing different HER replay strategies. As described in section 5.2, we use SAC with the same hyperparameters both for HER and for L2E.

5.4 Results

Both at training and evaluation time, we run episodes of length 250. For each method q (i.e., L2E and all baselines), we independently train $A = 10$ agents. After N environment transitions, we evaluate the agents. We reset to random initial positions and goals/plans and run the experiment until the goal is reached or until the episode ends. We repeat this process $M = 30$ times for each agent, and store whether the rollout was successful. We denote the result of the m-th evaluation of the a-th agent for method q, evaluated after learning for N environment transitions, as $\mathcal{F}^{(q)}_{am}(N)$.

As can be seen from the video given in the supplementary, even though the L2E agent uses plan information as a guiding bias during exploration, it learns strategies that differ considerably from the original plan. For example, while the simple planner (see figure 1a) suggests to re-establish the contact during the sequence, the L2E agent almost always moves and turns the box using a single contact.

5.4.1 Comparing L2E to Baselines

The results for the pushing setup are summarized in figure 1b. We observe that both L2E versions outperform all baselines in terms of the asymptotical performance. L2E with biased replay strategy $S_{10,1000}$ exhibits a high sample efficiency, resulting in success rates significantly higher than 50% after 4000 episode rollouts or 1 Million time steps. Directly executing the plan results in very low success rates of significantly less than 20% on average. Executing the plan with an IM still shows significantly worse long-term performance than the RL methods. HER results in better policies than the IM baselines, but is relatively data hungry.

Pushing is an unstable interaction that requires reasoning for several time steps ahead. A typical failure mode of the IM baselines (see also videos) is that the box moves away from the intended trajectory too much, so that the agent is not able to correct for it within one time step. In contrast, the L2E agent learns to deviate from the planned trajectory if this is required to reach the goal.

We find that L2E, combining a model-based planner and a universal plan-conditioned policy, achieves higher success rates than only using a model-based planner and both better sample efficiency and higher success rates than learning a universal goal-conditioned policy with HER.

5.4.2 Optimizing HER Replay Strategy

To allow for a fair comparison, we spent a considerable amount of effort to optimize the HER replay strategy. Multiple experiments comparing different choices for the HER replay strategy are shown in figure 1c. Please also refer to Andrychowicz et al. (2017) for a detailed description of the sampling strategies listed. We find that using 5 achieved goals from the same episode ("Ep 5") for replay results in the best long-term performance. The initial sample efficiency is however higher when using future achieved goals from the same episode ("Fu 5") or the last achieved goal of each episode ("Fi"). This is in good agreement with what was found for a comparable pushing task in Andrychowicz et al. (2017).

5.4.3 Optimizing L2E Replay Strategy

We compare different replay strategies for L2E in figure 1d. We find that using biased sampling is crucial to achieve high sample efficiency. The uniform strategy S^{uni}_{1000} catches up to the biased agent only after ca. 8 Million transitions (not shown in the figure). However, overly pronounced biasing is also disadvantageous (see $S^{\text{bias}}_{10,10000}$ in comparison to $S^{\text{bias}}_{10,1000}$). In this case, the L2E agent does not learn to generalize well to the test scenario.

6 Discussion and Conclusions

We introduced L2E, an algorithm that links RL and model-based planning using FV-RS. RL generally results in well-performing policies but needs large amounts of data, while model-based planning is data-efficient but does not always result in successful policies. By combining the two, L2E seeks to exploit strengths of both approaches. We demonstrated that L2E in fact shows both higher sample efficiency when compared to purely model-free RL and higher success rates when compared to executing plans of a model-based planner. This observation holds true even if a learned inverse dynamics model is used for transfer when executing the plan.

The specific FV-RS function used in this work applies to MDPs with sparse rewards. We focused on this since sparse rewards are common in robotic manipulation. They often present notoriously hard exploration tasks, making external plan-based information provided by L2E particularly useful. However, FV-RS in general is not restricted to sparse-reward settings, and by using a different shaping function, L2E could be applied in other settings as well.

We used very simple plans to support the RL agent during training. We demonstrated that these plans are already sufficient to significantly speed up learning in our experiments. While for these plans, we used an analytical encoding, for very complex scenarios it might be beneficial to learn an encoding using an auxiliary objective (see, e.g., Co-Reyes et al. (2018)). As another practical consideration, please note that L2E can be used in combination with any off-policy RL algorithm. L2E reformulates a goal-conditioned MDP as a plan-conditioned MDP, and provides a replay strategy to efficiently solve the latter. L2E is agnostic to how this data is then used by the RL agent.

The original HER paper (Andrychowicz et al., 2017) also considers the use of a simple form of reward shaping in combination with HER. It is found that reward shaping dramatically reduces the performance of HER in a robotic pushing task. In the present work, we show in contrast that including plan information using FV-RS shaping improves the performance of RL in a similar task. A possible explanation to reconciliate these seemingly contradictory results is already offered by Andrychowicz et al. (2017): While simple domain-agnostic shaping functions can be a distraction for the RL agent, domain-specific reward shaping functions can be beneficial. This view is supported, e.g., by similar results by Popov et al. (2017). Andrychowicz et al. (2017) state that however "designing such shaped rewards requires a lot of domain knowledge". In this context, one could view L2E as an automated way to extract such domain-specific knowledge from model-based planners and make it available. We specifically believe that L2E can be useful in robotic manipulation tasks, where domain knowledge is in many cases readily available. Here, L2E offers a way to exploit this.

References

Nvidia physx product site, May 2021. URL `https://developer.nvidia.com/gameworks-physx-overview`.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, pp. 5048–5058, 2017.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pp. 679–684, 1957.

Serkan Cabi, Sergio Gómez Colmenarejo, Alexander Novikov, Ksenia Konyushkova, Scott Reed, Rae Jeong, Konrad Zolna, Yusuf Aytar, David Budden, Mel Vecerik, et al. Scaling data-driven robotics with reward sketching and batch reinforcement learning. *arXiv preprint arXiv:1909.12200*, 2019.

Paul Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba. Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*, 2016.

John Co-Reyes, YuXuan Liu, Abhishek Gupta, Benjamin Eysenbach, Pieter Abbeel, and Sergey Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *International Conference on Machine Learning*, pp. 1009–1018. PMLR, 2018.

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

Yijie Guo, Jongwook Choi, Marcin Moczulski, Shengyu Feng, Samy Bengio, Mohammad Norouzi, and Honglak Lee. Memory based trajectory-conditioned policies for learning from sparse rewards. *arXiv preprint arXiv:1907.10247*, 2019.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pp. 1861–1870. PMLR, 2018.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, et al. Deep q-learning from demonstrations. *arXiv preprint arXiv:1704.03732*, 2017.

Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, pp. 1094–1099. Citeseer, 1993.

W Bradley Knox and Peter Stone. Combining manual feedback with subsequent mdp reward signals for reinforcement learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pp. 5–12. Citeseer, 2010.

Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous robots*, 40(3):429–455, 2016.

Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on Robot Learning*, pp. 1113–1132. PMLR, 2020.

Andrew W Moore, Leemon Baird, and Leslie P Kaelbling. Multi-value-functions: E cient automatic action hierarchies for multiple goal mdps. In *Proceedings of the international joint conference on artificial intelligence*, pp. 1316–1323, 1999.

Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (TOG)*, 31(4):1–8, 2012.

Soroush Nasiriany, Vitchyr H Pong, Steven Lin, and Sergey Levine. Planning with goal-conditioned policies. *arXiv preprint arXiv:1911.08453*, 2019.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pp. 278–287, 1999.

Andreas Orthey. Bpy-scripting. `https://github.com/aorthey/bpy-scripting`, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

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

Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

Harish Ravichandar, Athanasios S Polydoros, Sonia Chernova, and Aude Billard. Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3, 2020.

Tim Salimans and Richard Chen. Learning montezuma's revenge from a single demonstration. *arXiv preprint arXiv:1812.03381*, 2018.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320, 2015.

Ingmar Schubert, Ozgur S. Oguz, and Marc Toussaint. Plan-based relaxed reward shaping for goal-directed tasks. In *ICLR 2021 - 9th International Conference on Learning Representations*, 2021.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

Matthew E Taylor, Halit Bener Suay, and Sonia Chernova. Integrating reinforcement learning with human demonstrations of varying ability. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 617–624, 2011.

Andrea Lockerd Thomaz, Cynthia Breazeal, et al. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *Aaai*, volume 6, pp. 1000–1005. Boston, MA, 2006.

Marc A Toussaint, Kelsey Rebecca Allen, Kevin A Smith, and Joshua B Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. 2018.

Vivek Veeriah, Junhyuk Oh, and Satinder Singh. Many-goals reinforcement learning. *arXiv preprint arXiv:1806.09605*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Our claims made in the abstract and introduction are demonstrated in the experiments section.
   (b) Did you describe the limitations of your work? [Yes] See section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We see no immediate societal impact of our work.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [Yes] We state the requirements for the background methods to be applicable in section 3, for L2E to be applicable in section 4, and further discuss them in section 6.

(b) Did you include complete proofs of all theoretical results? [N/A] We do not state formal theorems in this work.

3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We include the entire open-source code base to fully reproduce all figures in this paper in the supplementary material. This includes README files to explain how it can be used. We will also share a link to our github repo once the paper is de-anonymized. Although this should be sufficient to easily and fully reproduce all results in this paper, we will also make the data files containing the results shown in this paper available upon request.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See section 5.2 and section 5.4.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Figure 1 shows mean values as well confidence intervals (standard deviation of the mean).

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See section 5.2.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We use open-source libraries that are cited where used.

(b) Did you mention the license of the assets? [Yes] We mention in the text that the assets are open source. More details on the exact license can be found in the references.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See question 3(a).

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We do not use external data assets.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We do not use external data assets.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We do not use crowdsourcing or conducted research with human subjects.

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] We do not use crowdsourcing or conducted research with human subjects.

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] We do not use crowdsourcing or conducted research with human subjects.