Improving Exploration in Deep Reinforcement Learning by State Planning Policies

Anonymous Author(s)
Affiliation
Address
email

Abstract

We introduce an improvement for reinforcement learning (RL) algorithms for continuous setting called state planning policy RL (SPP-RL). In SPP-RL, the actor plans for the next state provided the current state. To communicate the actor output to the environment, we incorporate an inverse dynamics control model and train it using supervised learning. We evaluate our improvement using the off-policy state-of-the-art reinforcement learning algorithms: TD3 and SAC. The target states need to be physically relevant; the overall learning procedure is formulated as a constrained optimization problem, solved via the classical Lagrangian multipliers method. We benchmark the state planning RL approach using a set of Safety-gym level 0 (no safety cost involved) environments and the AntPush env.. We find that SPP-RL significantly beats the baselines in terms of average return. We assign the performance boost to the more efficient SPP-RL agent exploration, performed in the target-state space rather than the action space. We report numerical experiments confirming this finding.

1 Introduction

Research on reinforcement learning (RL) has brought many successful applications in diverse fields of science and technology. RL application areas can be split into two classes: discrete (e.g., board games) and continuous (e.g., robotic problems). Here, we are interested in continuous simulation environments, mostly in robotics. The RL is concerned with training a policy governing agent motion via interactions with the environment to maximize the expected total return.

Traditionally, RL is based on searching for the optimal policy within the space of state-action mappings; the policy is a function assigning an action to take depending on the current state. We propose an improvement based on the principle of training an actor (a policy) operating entirely in the state space (state-state mappings). We call such policies the state planning policies (SPP), whose actions determine desired trajectories in the state space. The task of training SPP may initially seem infeasible due to a significantly larger dimension of states than actions. Nonetheless, quite surprisingly, we show that the approach is feasible and often leads to significant improvements in average performance and decreased sample efficiency for a class of robotic locomotion tasks.

We call our approach *State Planning Policy Reinforcement Learning (SPP-RL)*. It is a generic approach for problems specified using continuous environments. The main building block of SPP-RL – the RL agent can be implemented using virtually any model-free RL algorithm. We chose to develop our approach using the state-of-the-art off-policy DDPG [18], TD3 [8], and SAC [11] algorithms. Note that, in SPP-RL we need another trainable model to communicate the policy output to the environment; as such, we incorporate a learnable inverse dynamics control model (IDM), see Fig. 1. The overall algorithm optimizes the policy simultaneously with IDM. To ensure that the policy target states satisfy physical and under-actuation constraints, we formulate a constrained optimization

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

objective for policy training. Our work lies within the category of RL methods that have already implemented state-state policies, including work on hierarchical RL [21], the D3G algorithm [6], and behavioral cloning from observation [27]. In this work, we show the properties and advantages of state-planning policies that have not been demonstrated earlier.

Summary of results. Although the SPP-RL algorithm searches for the optimal policy within a much larger space, our performance benchmarks revealed that SPP-RL implementations often outperform their vanilla RL counterparts. Experiments in Safety-Gym Level 0 environments [25] (without safety cost) demonstrate that SPP-TD3 and SPP-SAC outperform by a great margin TD3 and SAC, respectively. Experiments in AntPush task [21] show that SPP-TD3 outperforms hierarchical RL method HIRO [21] and provides some interpretability of the agent behavior.

We hypothesize that the superior performance of SPP-RL in the tested continuous environments originates in more efficient state-space exploration by state-state policies than traditional state-action policies; here noise is being added to target states rather than actions. To argue this, we performed series of experiments, including evaluation of a shadow agent utilizing experience from SPP and vanilla replay buffers (Sec. 5.2) and a study of the distributions of states gathered in different replay buffers (Sec. 5.4).

We implemented SPP-RL methods as a modular PyTorch library shared as open-source. SPP-RL algorithms are derived from their vanilla RL counterparts, making extending the library with new RL algorithms straightforward. We also share videos with test episodes of the trained agents to accompany benchmark plots [1].

1.1 Related work

We present a (non-exhaustive) list of related works; refer to Tab. 1 for a perspective on related work. The closest approach to ours is the D3G algorithm introduced by [6], which includes state planning policies, and introduces a novel form of the value function defined on state-next state pairs. There are two main ways our method is distinct. First, SPP employs the classical formulation of the value function. Also, we do not include a forward dynamics model nor the cycle loss. Instead, to guarantee consistency of the policy target-states in SPP, we formulate a constrained optimization problem (compare Fig. 2) solved via Lagrangian optimization.

Our work builds on the classical RL algorithms going back to REINFORCE [30], Asynchronous Actor-Critic [19], and especially the off-policy actor-critic algorithms including Q-Prop [9], DDPG [18], SAC [11, 12], and TD3 [8]. State planning policies have been used in hierarchical RL (HRL) methods like HIRO [21] and FuN [28]. Contrary to HRL SPP-RL approach does not employ a hierarchy of multiple policies nor state conditioned value functions.

Training predictive models (like IDMs) is fundamental for the model-based RL approach including algorithms: a locally linear latent dynamics model [29], model-based planning for discrete and continuous actions [14], model-predictive control [5], and model based policy optimization [16]. We deployed IDMs for mapping current-target states to actions; other applications of IDMs in RL include the context of planning: search on replay buffer [7], episodic memory graph [31], and topological memory for navigation [26]. Existing many other applications of IDM in context of RL including: policy adaptation during deployment [13], sim to real transfer [4], adversarial exploration [15], curiosity-driven exploration [23], and video pre-training for minecraft [3].

Technique	State-state policy	Inverse/forward model	State cond. Q funct.	Policy Hierarchy	Planning horizon
SPP (ours)	yes	inverse	no	single policy	single step
D3G	yes	inverse & forward	yes	single policy	single step
HRL	yes(upper level)	inverse	yes(upper level)	multiple policies	multiple steps
Planning	yes	inverse	yes	N/A	multiple steps
Model based	no	forward	N/A	single policy	single step

Table 1: A Perspective on Related Work

1.2 Background

Following the standard setting used in RL literature, we work with infinite horizon *Markov decision process* (MDP) formalism $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$, where $\mathcal{S}$ is a *state space*, $\mathcal{A}$ is a *action space*, $P\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a *transition probability distribution*, $r\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a *reward function*, ρ_0 is an *initial state distribution*, and $\gamma \in (0, 1)$ is a *discount factor*. From now on we assume that the MDP is fixed. In RL the agent interacts with E in discrete steps by selecting an action a_t for the state s_t at time t, causing the state transition $s_{t+1} = E(s_t, a_t)$, as a result the agent collects a scalar reward $r_{t+1}(s_t, a_t)$, the return is defined as the sum of discounted future reward $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i)$. The goal in RL is to learn a policy that maximizes the expected return from the start distribution.

2 State Planning Policy Reinforcement Learning Approach

Our SPP approach is rooted in state-state reinforcement learning, by which we mean setting in which RL agent is trained to plan goals in the state-space, the approach already employed e.g., in HRL, planning, D3G RL algorithms (see Tab. 1). In SPP a state planning policy π given the current state s_t outputs z_t – the desired target state to be reached by the environment in the next step. Forcing the environment to reach the desired state requires translating the target state to a suitable action a_t. Hence, we employ an additional model capable of mapping the current state-target state pair (s_t, z_t) to the action a_t – a (trainable) IDM model. Ideally, we like to have consistency $z_t(s_t) \approx s_{t+1}$. The consistency cannot be guaranteed a-priori, is rather achieved in SPP setting by employing a constrained optimization approach. A diagram illustrating SPP approach is presented in Fig. 1. We have freedom of choice of the particular RL algorithm (RL agent) and IDMs implementations. Currently, we use feed-forward neural networks, and RL Agent using implementations of the state-of-the-art off-policy RL algorithms: DDPG [18], TD3 [8] and SAC [11]. We present details of SPP-RL implementation in Sec. 4 using as the example SPP-DDPG. The encountered experiences during the execution of an off-policy RL algorithm are stored in replay buffer $\mathcal{D}$.

The main building block of SPP-RL are the state planning policies, intuitively a state planning policy selects a desired trajectory in the state-space of the environment.

Definition 1. We call a *state planning policy* a map $\pi_\theta\colon \mathcal{S} \to \mathcal{P}(\mathcal{S})$ parametrized using a vector of parameters $\theta \in \mathbb{R}^{n(\theta)}$, and we denote $\pi_\theta(z|s)$ a probability of the desired *target state* $z \in \mathcal{S}$ for the given current state $s \in \mathcal{S}$.

We call a *deterministic state planning policy* a parametrized map $\pi_\theta\colon \mathcal{S} \to \mathcal{S}$, and we denote $\pi_\theta(s) = z$.

We assume that π has continuous and bounded derivatives with respect to θ. We will call state planning policy whenever it is clear from the context deterministic/stochastic and omit the parameter subscript $\pi = \pi_\theta$.

Besides the state planning policy (Def. 1) the second main building block of the overall SPP agent is a model for mapping the current state-target pair (s_t, z_t) to suitable action a_t. Following the existing literature, we call such model the *inverse dynamics control model* (IDM), or simply the *control model*.

Definition 2. For a given MDP $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$. Let $s, z \in \mathcal{S}$. We define the *control model*:

$$\mathsf{CM}\colon \mathcal{S} \times \mathcal{S} \to \mathcal{A}, \quad \mathsf{CM}(s, z) = a,$$

i.e. for the given two states CM computes the action a. We call s, z the initial state and the target state respectively, where a informally satisfies $\text{argmax}_{b\in\mathcal{S}} P(s, a, b) \sim z$ for stochastic E, or $z \approx E(s, a)$ for deterministic E.

Obviously, in order to work, SPP requires consistency of the target states generated by the policy with the actual next-states of the environment. We call this property the state consistency property (or simply consistency) of π, refer to Fig. 2. As it may be intractable to verify SPP for all possible interactions in continuous environments, we are interested in guaranteeing the state consistency for the experiences stored in the replay buffer. It is analogous to the behavioral cloning from observation loss [27].

Property 1. Let $\mathcal{D}$ be a replay buffer, CM be an IDM and π be a (SPP) policy. We say that π has the *state consistency* property with threshold $d > 0$ if it holds that

$$\mathop{\mathbb{E}}_{\substack{(s_t, s_{t+1}) \in \mathcal{D} \\ z_t \sim \pi(s_t)}} \left[\|s_{t+1} - z_t\|_2^2 \right] \leq d,$$

for deterministic π we have $z_t = \pi(s_t)$. Given (z_t, s_{t+1}), we call distance $\|z_t - s_{t+1}\|_2^2$ the *state-consistency distance* (refer Fig. 2). We will often assume that d is known from context and omit 'with threshold $d > 0$'.

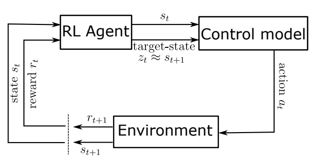


Figure 1: Diagram presenting our SPP method

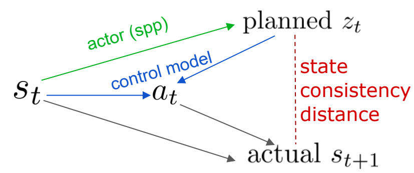


Figure 2: Diagram presenting the idea of state consistency property, ultimately we want to achieve $z_t \approx s_{t+1}$.

3 Ensuring State Consistency Property by Constrained Optimization

Our SPP algorithm is utilizing three parametrized models: IDM CM_ψ, policy model π_θ, and Q-function model(s) Q_ϕ. In our current implementation, all of the models are feed-forward neural networks. One way of ensuring the state consistency property (Prop. 1) is to modify the policy training loss, such that the expected values are maximized under fixed state consistency distance penalty. However, such an approach has many disadvantages, e.g., choosing appropriate learning temperature (λ) for the state consistency penalty is a very delicate issue, see the ablation study in Appendix 5.5. It is easy to notice that setting its value too low would result in π biased towards nonphysical target-states. On the other hand, setting its value too high would make π overly conservative for off-policy states. Hence, we find a solution relying on constrained optimization more appealing for the studied problem. Namely, the objective for policy training is to maximize the sum of discounted rewards assuming a fixed threshold for the state consistency distance.

Definition 3. Let π be a state planning policy (Def. 1), CM be a control model (Def. 2), $\mathcal{D}$ be the replay buffer with experience generated by executing an off-policy RL algorithm. In particular z_t's are target states evaluated on-the-fly by π (susceptible to be changed during the course of algorithm), and s_{t+1}'s are E next-states. Let $d > 0$ be a fixed hyperparameter.

We define *the constrained objective for state planning policy* π as follows

$$\max_{\pi} \mathop{\mathbb{E}}_{\substack{\tau \sim \pi, \mathsf{CM} \\ a_i = \mathsf{CM}(s_i, z_i)}} \left[\sum_{i=0}^{T} \gamma^i r(s_i, a_i) \right], \tag{1a}$$

$$\text{s.t.} \mathop{\mathbb{E}}_{\substack{(s_t, s_{t+1}) \in \mathcal{D} \\ z_t \sim \pi(s_t)}} \left[\|s_{t+1} - z_t\|_2^2 \right] \leq d, \tag{1b}$$

where d is a hyperparameter for determining the allowed threshold for the expected divergence of predictions from actual next-states, (1a) is an expectation over the policy trajectories generated by both of the state planning policy and IDM (trajectory is composed out of tuples (s_t, z_t, a_t)). The whole optimization process of (1a) is being performed off-policy, see Sec. 4. The constrained objective in (1) is being solved using the standard Lagrange multiplier method. The max-min Lagrangian objective $\mathcal{L}(\pi, \lambda)$ for the constrained optimization problem takes the form $\max_\pi \min_{\lambda \geq 0} \mathcal{L}(\pi, \lambda) = \mathop{\mathbb{E}}_{\tau \sim \pi} [R_0(\pi)] - \lambda \left(\mathop{\mathbb{E}}_{\mathcal{D}} \left[\|s_{t+1} - z_t\|_2^2 \Big|_{z_t \sim \pi(s_t)} \right] - d \right)$. For more details refer to App. B.1.

4 Algorithm Implementation

We briefly present here details of the SPP Algorithm implementation, more detailed discussion can be found in Appendix B. We implemented SPP-DDPG, SPP-TD3, and SPP-SAC as a modular Python library within PyTorch framework [22]. All gradient optimization steps were performed using the Adam optimizer by [17]. Many of the algorithmic choices were motivated by the Spinning Up RL on-line resource [2]. We publish the modular SPP-RL software package as open-source [1]. For illustrative purposes, we present the pseudo-code of the full SPP-DDPG algorithm in Algorithm 1. SPP-SAC and SPP-TD3 algorithms are presented in Appendix B. We emphasize that SPP algorithms are not using any extra samples, i.e. the samples utilized for ICM training are added to the buffer and then reutilized for RL training, and if the buffer is full new samples are not being added anymore. An important caveat of our policy implementation in SPP-DDPG, not present in the vanilla DDPG, is that the output of π is being normalized in order to reflect the physical bounds. Contrary to vanilla DDPG where π outputs actions within well-defined uniform bounds, in SPP a suitable normalization of target state π output is being computed online – depends on the past E observations. Implementation of π is a feed-forward neural network (refer to Appendix C for details) with tangential outputs bounded within $[-1, 1]$. Hence, we normalize the output of neural network π by utilizing the current mean and min/max values of the past observations in replay buffer $\mathcal{D}$. We recompute mean and min/max values after each episode of the algorithm. Our base implementation

Algorithm 1: SPP-DDPG Algorithm

input : environment E; initial model parameters θ, ϕ, ψ; state planning distance threshold d; empty replay buffer $\mathcal{D}$; the DDPG algorithm hyperparameters
output: trained model parameters θ, ϕ, ψ; total return
repeat
  Sample random action $a \sim \mathcal{U}$;
  Store experience $(s_t, a_t, z_t = s_{t+1}, r_{t+1}, s_{t+1})$ in replay buffer $\mathcal{D}$; (use next-state as the initial actor actions)
until *random exploration is done*;
repeat
  if *buffer $\mathcal{D}$ is not full* **then**
    Compute actor prediction $z_t = \pi(s_t) + \varepsilon$, where $\varepsilon \sim \mathcal{N}$;
    Compute action $a_t = \mathsf{CM}(s_t, z_t)$ and observe reward r_{t+1} and next state s_{t+1};
    Store experience $(s_t, z_t, a_t, s_{t+1}, r_{t+1})$ in $\mathcal{D}$;
  end
  if *it's time to update* CM **then**
    Sample $\{b_i = \{((s_t, s_{t+1}), a)\}\}_{i=1}^{b}$ b batches of samples from replay buffer $\mathcal{D}$;
    SGD train CM using the batches and MSE loss;
  end
  if *it's time to update actor and critic* **then**
    for *update steps* **do**
      Randomly sample $\mathcal{B} = \{(s_t, z_t, a_t, s_{t+1}, r_{t+1})\}$ set of batches from $\mathcal{D}$;
      Compute $\tilde{a}_{k+1} = \mathsf{CM}(s_{t+1}, \pi(s_{t+1}))$;
      Compute targets $y = r_{t+1} + \gamma Q^{\pi,\mathsf{CM}}_{\phi_{targ}}(s_{t+1}, \tilde{a}_{k+1})$ (using target parameters ϕ_{targ});
      Update $\phi = \phi - \frac{l_\phi}{|\mathcal{B}|} \cdot \nabla_\phi \sum_{\mathcal{B}} \left(y - Q^{\pi,\mathsf{CM}}_{\phi}(s_t, a_t)\right)^2$;
      Update policy parameters (ascent w.r.t θ of max-min Lagrangian obj.) $\theta =$
      $\theta + l_\theta \left(\frac{1}{|\mathcal{B}|} \cdot \nabla_\theta \sum_{\mathcal{B}} Q^{\pi,\mathsf{CM}}_{\phi}(s_t, a_t)\Big|_{a_t = \mathsf{CM}(s_t, \pi_\theta(s_t))} - \frac{\lambda}{|\mathcal{B}|} \cdot \nabla_\theta \sum_{\mathcal{B}} \|s_{t+1} - \pi_\theta(s_t)\|_2^2 \right)$;
      Update (descent w.r.t. λ of max-min Lagrangian obj.)
      $\lambda = \lambda + l_\lambda \left(\frac{1}{|\mathcal{B}|} \sum_{\mathcal{B}} \|s_{t+1} - \pi_\theta(s_t)\|_2^2 - d \right)$;
      Update actor & critic $\phi_{targ} = (1-\tau)\phi_{targ} + \tau\phi$; $\theta_{targ} = (1-\tau)\theta_{targ} + \tau\theta$;
    end
  end
until *convergence*;

of DDPG algorithm is parametrized by the usual hyper-parameters including episode length, update batch size, Polyak averaging parameter (τ), actor and critic learning rate l, maximal episode length, number of test episodes, γ. To ensure that we perform minimization (2) within the domain of positive λ values, we optimize the parameter of the softplus function. All relevant hyper-parameters are provided in Appendix C.

5 Experimental Evaluation

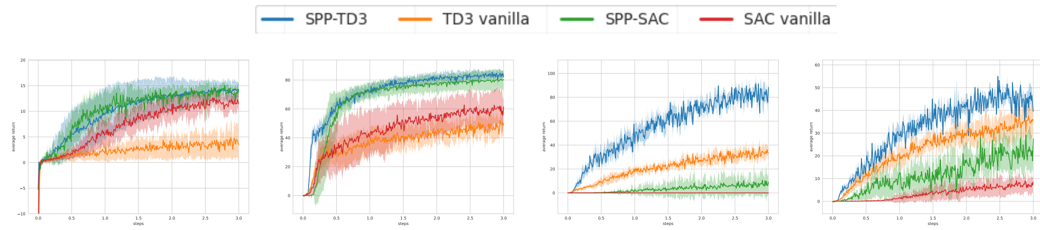
To show the feasibility of our method, we performed experiments on a set of benchmarks using continuous environments, most of them having large space dimensions. We performed all of the reported experiments using the default vector state input. We show that SPP-RL implementations compare favorably to their vanilla RL counterparts. As our SPP approach differs considerably from the vanilla off-policy RL, we performed a thorough hyper-parameter sweep from scratch. We provide the hyper-parameter values from the actual SPP implementations in Appendix C. For the sake of presentation we share videos with example test episodes rendered using the trained actors and high-resolution benchmark plots. All of our experiments are reproducible, we share the sources, training, evaluation, and trained models online [1]. We also evaluated SPP-RL in classical MuJoCo tasks, and the performance is comparable to vanilla (see [1]). All of the reported experiments were run using CPU only, and a single experiment was always run on a single CPU core, i.e., we have not performed collecting experience in parallel. The experiments were performed on an example machine: AMD Ryzen Tr. 1920X, 64 Gb RAM, Ubuntu OS 18.04. Example average time of execution of 10^6 steps stands at SPP-DDPG 5hrs $58'$, DDPG 3hrs, $7'$, (SPP-)SAC 19hrs $20'$, SAC 11hrs.

5.1 Safety-Gym (Locomotion Tasks)

We use environments from the safety-gym suite by [25]. Currently, we employed only Level 0 environments (which does not involve the cost function for violating the safety). We find Level 0 tasks from the safety-gym suite as the perfect ground to study the performance of SPP-RL in robotic locomotion environments, the goal being to steer agents (robots) to solve planar goal-reaching tasks. Moreover, we concentrate on difficulties arising from higher dimensionality of the state (and actions) space rather than maximizing returns under safety constraints. The experiments were performed using solely the vector state input. We leave investigating the higher-level environments considering the cost function as a topic of future research. We chose a subset of the most challenging Level 0 tasks, including Car-Push, Doggo-Goal, Doggo-Button environments. We also create a custom environment (termed Doggo-Columns) based on Doggo-Goal with additional 10 fixed pillars placed in the arena, obscuring the paths toward the goal. We evaluate our SPP-TD3 implementation against state-of-the-art off-policy algorithms like TD3 and SAC. The results presented in Fig. 3 clearly show that SPP-TD3 is superior to vanilla off-policy algorithms within the studied safety-gym environments. Also, there is a noticeable difference in the learned behavior of the trained agents. The agents trained using the SPP-RL approach show smarter and more efficient behavior; for instance, the trained using SPP doggo robot learned an efficient gait of moving backward to mark a goal or press a button. For comparison, we publish videos of the trained agents online [1]. We argue that the performance boost exhibited by SPP-RL algorithms over vanilla counterparts is due to improved exploration. In Sec. 5.2 we show results from evaluating a TD3 shadow agent, i.e. vanilla TD3 agent utilizing for training some portion of experience from SPP-TD3 replay buffer. In Sec 5.4 we investigate differences in distribution of states collected by both of the methods.

5.2 More Efficient Exploration in SPP-RL

Our experimental evaluation using the safety-gym environments show that SPP-RL implementations outperform by a great margin their vanilla off-policy RL counterparts (TD3 and SAC) in terms of the average returns (See Fig.4). In this section, we argue the performance boost of SPP-RL compared to the vanilla RL counterparts. Our intuition is that exploration performed in the target-state space rather than in the action space may be more efficient in some cases. Exploration using SPP policies results in more viable experience being collected in the replay buffer, leading to more efficient Actor & Critic training. It is also possible that the constrained optimization induces some kind of curriculum. To confirm the mentioned intuition, we evaluated the performance of a TD3 shadow agent i.e., a vanilla TD3 Actor&Critic trained using (partially) experience collected by the SPP-TD3 agent. Both of the agents were trained in parallel. The TD3 shadow agent updates were performed using samples drawn from two of the replay buffers. The replay buffers of the SPP-TD3 and TD3 agent were used according to a 50/50 ratio. The results are presented in Fig. 4. Such TD3 shadow agent outperforms vanilla TD3, and eventually, its performance matches SPP-TD3 agent's in all of the studied safety-gym environments, excluding Doggo-Button. We present plots of the discretized distributions of states encoded using a random encoder and cross-entropy of two distributions w.r.t. the quantity of gathered experience.



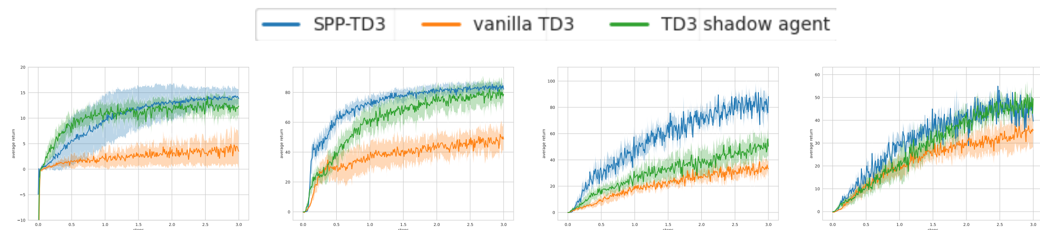
(a) Car-Push, (SPP)TD3 & SAC

(b) Doggo-Goal, (SPP)TD3 & SAC

(c) Doggo-Button, (SPP)TD3 & SAC

(d) Doggo-Columns, (SPP)TD3 & SAC

Figure 3: Experimental comparison of SPP-TD3 with corresponding vanilla off-policy RL on set of safety-gym level 0 environments. Figures show test return computed every 5k frames averaged over 10 different seeds. The continuous curve is the mean, whereas the faded color regions std. deviation. D3G did not converge (return oscillated around zero, or it diverged in CarPush to a large negative score - removed from the plot for clarity). Fine-tuning D3G to make it work in this setting is beyond scope of the research. Refer to Appendix C for exact hyperparameters that we used to perform those experiments.



(a) Car-Push

(b) Doggo-Goal

(c) Doggo-Button

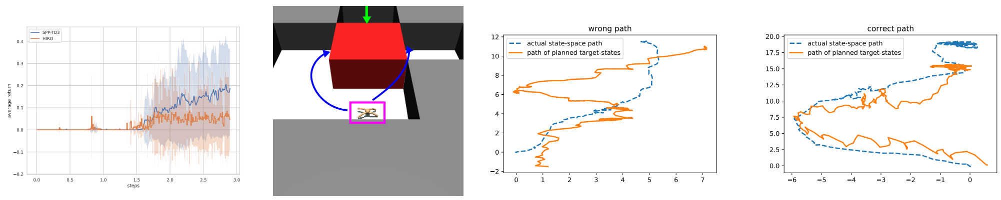
(d) Doggo-Columns

Figure 4: Experimental evaluation of (SPP)TD3 agents and a TD3 shadow agent, i.e. vanilla TD3 agent trained utilizing experience collected by a SPP-TD3 agent, on a set of safety-gym level 0 environments. Presented metrics are same as in Fig. 3.

5.3 Harder Exploratory Task AntPush

We describe an experiment in AntPush environment from [21]. The experiments were performed using solely the vector state input. The task is to control the ant such that it reaches the goal. The goal is hidden within a chamber behind a block. Therefore, Ant needs to learn to walk around the block and push it to the right first before eventually reaching the goal. Success is defined as finishing the episode within a radius 5 from the goal. We benchmark SPP-TD3 against the state-of-the-art hierarchical RL HIRO method by [21]. Specifically, we used the implementation [24]. Instead of reporting the achieved success rate of a single training run like in [21], which may be spurious if a lucky seed is chosen, we report the mean and std.dev. of the AntPush success rate using 10 random seeded training runs. Our experiments revealed that HIRO is highly susceptible to the random seed used. Only a single HIRO agent out of 10 trained using random seeds in total achieved a positive success rate, comparing to 7 out of 10 SPP-TD3 agents successfully learned to solve the task. The performance reported in Fig. 5a shows SPP-TD3 is eventually superior to HIRO. Example two solution paths are marked on Fig. 5b (blue curves). The right path is suboptimal as Ant blocks the entrance to the chamber where the goal is. The left path is optimal, Ant traverses to the left to push the red block away and open the passage towards the goal (green arrow).

Finally, Figs 5c, 5d show the obtained paths in the state space (blue dashed), and the policy target states z_t's (orange solid), only the first two coordinates corresponding to the position of the Ant body in x, y coordinates are illustrated. Observe that in the case of the suboptimal path in Fig. 5c, the planned path diverts to the left from the actual path (blue dashed), which indicates that the agent learned and attempted the correct behavior of pushing the red brick away and successfully open the entrance to the goal. In this case, however, the block is not movable; it is stuck as it was pushed forward before, hence as we see, the actual path in the state-space diverts in the middle. Figs 5c,5d show that apparently, the policy target states path being more erratic than the actual path in the state space. Erratic behavior can be mitigated by adjusting the hyperparameter d in (1b) (the smaller d, the closer the paths will be). Nonetheless, the policy target paths (orange) in Figs 5c,5d could be potentially used to cluster agent behavior, qualitatively differentiating two example agents executing (sub)optimal path. This information could then be used to pick appropriate agents for deployment.



(a) AntPush, SPPTD3 (blue) & HIRO (orange)

(b) Goal in AntPush env (green arrow), and two possible paths (blue curves)

(c) A suboptimal Ant path example in the state space (blue dashed) and target states (orange solid)

(d) Example of an optimal path in the state space (blue dashed) and target states (orange solid)

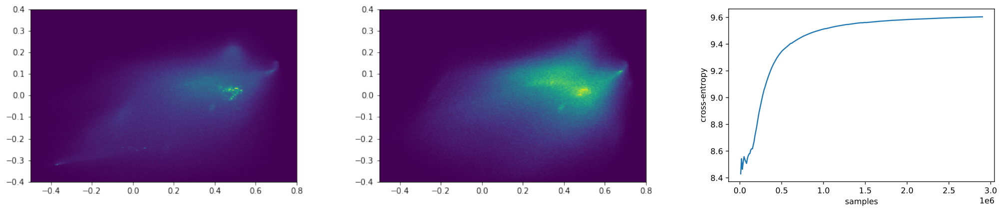
Figure 5: Experiment in AntPush environment from [21]. Fig. 5a shows success rate for SPP-TD3 & HIRO computed every 5k frames averaged over 10 different seeds (10 independent training runs were used). The continuous curve is the mean, whereas the faded color regions std. dev. Fig. 5b show two possible solution Ant paths. Figs 5c,5d show the paths in the state space (blue dashed) and target states (orange solid), the coordinates describing the position of the Ant body (x, y) are used.

5.4 SPP-RL vs Vanilla RL Replay Buffer

As argued in Sec. 5.2, the performance boost visible in SPP-RL vs. vanilla RL approaches is presumably attributed to more efficient exploration performed by SPP-RL algorithms than vanilla RL. One empirical argument is given in Sec.5.2. Here we provide empirical evidence that the distribution of observations in replay buffers gathered by SPP-RL and vanilla RL implementations differs considerably. In Fig. 6 we present an empirical study of distributions of states gathered in SPP-SAC and vanilla SAC replay buffers for the Doggo-Goal task. The state space in this task has 72 dimensions. Hence to make it amenable to visual investigation and entropy computation, we encode the state vectors using a random encoder. The encoder architecture that we used for this task is a simple architecture with random weights (not optimized): $72 \to 20\tanh \to 10\tanh \to 2$. Fig. 6 show plots of discretized state distributions gathered by example run of vanilla SAC and SPP-SAC respectively using the Doggo-Goal environment and encoded using the random encoder, observe that the state distributions are different, i.e., the distribution for vanilla SAC is visibly more concentrated than the one for SPP-SAC. We also compute cross-entropy to quantify the difference between those distributions as the training of both algorithms progresses and the replay buffer is filled up. Observe that the cross-entropy is increasing as the replay buffer is being filled up, suggesting that vanilla RL and SPP-RL algorithms gather different observation distributions in the replay buffer. We also performed an analogous analysis for the (SPP)TD3 approach, but it looked qualitatively similar and is not reported here.

5.5 Ablation Study of the Lagrangian Objective

Using the Doggo-Goal environment, we performed an ablation study of the SPP-TD3 the most important feature. We investigate the impact of the Lagrangian objective (1) on the overall SPP-RL performance. We compare the implementation with Lagrangian objective to the implementation utilizing fixed λ values (parameter not trained using dual optimization), including $\lambda = 1, 0.5, 0.1, 0.01$. Performance varies greatly depending on this parameter, demonstrating how delicate the matter of choosing appropriate λ (when fixed) per given environment is. Observe that $\lambda = 1$ results in a lack of convergence, and $\lambda = 0.1$ or 0.01 results in even better performance than the SPP-TD3 implementation with the Lagrangian multipliers. However, employing the Lagrange multipliers provides a natural way of solving the constrained objective optimization and avoids separate fine-tuning of λ



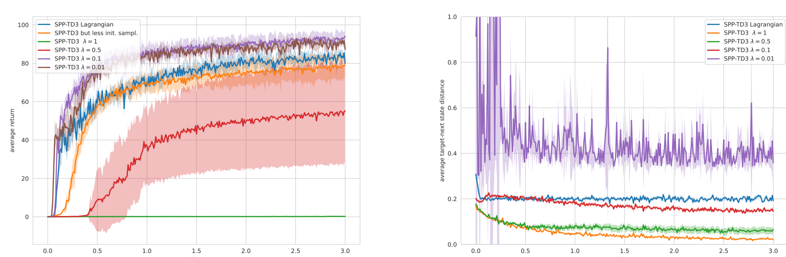
(a) Encoded state density from vanilla SAC buffer

(b) Encoded state density from SPP-SAC buffer

(c) Cross-entropy of the two encoded states distributions (discretized), vanilla SAC vs SPP-SAC

Figure 6: Visualizations of states distribution (density histogram plot) in the replay buffer at the end of training taken from single vanilla SAC, SPP-SAC runs. States are encoded in 2D using the random encoder.

per each environment. Moreover, the average target – next-state distance in case of $\lambda = 0.01$ is way above the set target for the Lagrangian method (0.2), whereas the Lagrangian objective successfully keeps it close to the target. We also present in Fig. 7. The study *SPP-TD3 less init. sample.*, when much less randomly generated experience is added to the replay buffer at the beginning of executing Alg. 3 (the first *repeat until* block). In some cases, like SPP-TD3 for Doggo-Goal, see Table 4, we choose to include a lot of random samples in the buffer (400k). However, this has no considerable effect on performance (Fig. 7a). The SPP-RL agent utilizing much fewer random samples (the same number as vanilla RL) has comparable performance. Other ablations that we tested included: SPP-TD3 $Q(s, s')$ critic (state-state critic), replacing the traditional target Q function computation using state-action pairs with state-next state pairs, and π not being normalized using the current mean and min/max values of the replay buffer observations. We do not show this ablations, as the algorithm did not converge for these settings.



(a) Average return plot

(b) Average target – next-state Euclidean distance

Figure 7: Ablation Study for the Lagrangian objective of SPP-TD3 algorithm, performed in the Doggo-Goal environment. The continuous curve is the mean, whereas the faded color regions std. deviation. computed from 5 independent runs.

6 Conclusions

We evaluated the state planning policies in reinforcement learning, where the policy selects target states for the environment. Experiments performed on continuous benchmark environments often show the superior performance of SPP-RL compared to state-of-the-art vanilla off-policy RL algorithms. There are various avenues for future work pertaining to this research. One path is to include in our approach physically informed control models. Another important work path is to implement a long-term policy planning method scheme and application in the safety RL setting of SPP approach.

References

[1] Spp-rl supplementary material webpage. `https://sites.google.com/view/spprl`, 2021.

[2] J. Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[3] B. Baker, I. Akkaya, P. Zhokhov, J. Huizinga, J. Tang, A. Ecoffet, B. Houghton, R. Sampedro, and J. Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos, 2022.

[4] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba. Transfer from simulation to real world through learning deep inverse dynamics model, 2016.

[5] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[6] A. Edwards, H. Sahni, R. Liu, J. Hung, A. Jain, R. Wang, A. Ecoffet, T. Miconi, C. Isbell, and J. Yosinski. Estimating q(s,s') with deep deterministic dynamics gradients. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2825–2835. PMLR, 13–18 Jul 2020.

[7] B. Eysenbach, R. R. Salakhutdinov, and S. Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[8] S. Fujimoto, H. van Hoof, and D. Meger. Addressing Function Approximation Error in Actor-Critic Methods. *arXiv e-prints*, page arXiv:1802.09477, Feb. 2018.

[9] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. 2017.

[10] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1352–1361, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[12] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[13] N. Hansen, R. Jangir, Y. Sun, G. Alenyà, P. Abbeel, A. A. Efros, L. Pinto, and X. Wang. Self-supervised policy adaptation during deployment. In *International Conference on Learning Representations*, 2021.

[14] M. Henaff, W. F. Whitney, and Y. LeCun. Model-based planning with discrete and continuous actions, 2017.

[15] Z.-W. Hong, T.-J. Fu, T.-Y. Shann, and C.-Y. Lee. Adversarial active exploration for inverse dynamics model learning. In L. P. Kaelbling, D. Kragic, and K. Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 552–565. PMLR, 30 Oct–01 Nov 2020.

[16] M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-based policy optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In Y. Bengio and Y. LeCun, editors, *ICLR*, 2016.

[19] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[21] O. Nachum, S. S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. 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, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[23] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

[24] Z. Qin. Repository implementing hiro algorithm. `https://github.com/ziangqin-stu/rl_hiro`, 2021.

[25] A. Ray, J. Achiam, and D. Amodei. Benchmarking safe exploration in deepreinforcement learning, 2019.

[26] N. Savinov, A. Dosovitskiy, and V. Koltun. Semi-parametric topological memory for navigation. In *International Conference on Learning Representations*, 2018.

[27] F. Torabi, G. Warnell, and P. Stone. Behavioral cloning from observation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4950–4957. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[28] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 3540–3549. JMLR.org, 2017.

[29] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2746–2754. Curran Associates, Inc., 2015.

[30] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992.

[31] G. Yang, A. Zhang, A. S. Morcos, J. Pineau, P. Abbeel, and R. Calandra. Think, act and learn on an episodic memory graph. In *ICLR 2020 workshop: Beyond tabula rasa in RL (BeTR-RL)*, 2020.