Appendix for the paper entitled *Improving Exploration in Deep Reinforcement Learning by State Planning Policies*

A Appendix

Videos visualizing the trained agents, and link to the git software repository with SPP-RL algorithms implementations are available online https://sites.google.com/view/spprl.

B SPP-RL Algorithm Details

B.1 Lagrangian Optimization

To solve the constrained optimization problem we use the idea of Lagrange multipliers going back to the classical calculus. Lagrange multipliers were already applied in the RL context in [25], where authors developed on-policy RL algorithms for safety RL. This method uses an adaptive penalty coefficient to enforce constraints. We find this method plausible, as it translates a constrained objective into solving a classical max-min problem that can be incorporated within any RL algorithm. The max-min Lagrangian objective $\mathcal{L}(\pi, \lambda)$ for the constrained optimization problem (1) takes the form

$$\max_{\pi} \min_{\lambda \geq 0} \mathcal{L}(\pi, \lambda) = \mathop{\mathbb{E}}_{\tau \sim \pi} [R_0(\pi)] - \lambda \left(\mathop{\mathbb{E}}_{\mathcal{D}} \left[\| s_{t+1} - z_t \|_2^2 \Big|_{z_t \sim \pi(s_t)} \right] - d \right). \tag{2}$$

Intuitively, when the constraint (1b) is satisfied, then the min w.r.t. λ is attained for $\lambda = 0$, and causes λ to decrease. On the other hand, if the constraint is not satisfied, then the min is attained for $\lambda = \infty$, and hence causing λ to increase, the constraint satisfaction receives more weight in the overall max-min objective, eventually forcing the constraint to be fulfilled.

We solve the max-min Lagrangian objective for SPP policy using simultaneous gradient ascent w.r.t. the policy parameters θ, descent w.r.t. the Lagrange multiplier λ, updates. Such an approach can be viewed as an instance of a primal-dual algorithm for constrained MDP. It is known in the literature that time-steps of Actor, Critic & dual variable gradient updates should be adjusted separately to guarantee convergence. The SPP algorithms use three learning rates: actor (l_θ), critic (l_ϕ), and dual variable (l_λ), which were adjusted using a hyper-parameter optimization. The actor and critic learning rates were equal in all of the studied algorithms, whereas the dual variable was fixed in all algorithms to $l_\lambda = 0.0001$.

B.2 SPP-SAC Algorithm

The original DDPG algorithm, which is the base of SPP-DDPG presented in Algorithm 1 aims at training a deterministic policy, as contrary to for example SAC algorithm by [11, 12] that trains a stochastic policy. Its main principle is rooted in the very successful deep Q-learning method, originally applied for solving the Atari benchmark problem [20]. However, DDPG is suitable for continuous action spaces like in the MuJoCo tasks. SAC is an algorithm that combines ideas of DDPG and soft Q-learning [10] utilizing the advantages of both approaches, SPP-SAC is presented in Algorithm 2.

B.3 SPP-TD3 Algorithm

Twin Delayed Policy Gradient (TD3) [8] is an improved DDPG algorithm using several tricks, which can be summarized as Clipped Double-Q Learning, Delayed Policy Updates, Target Policy Smoothing. SPP-TD3 algorithm is presented in Algorithm 3.

C Hyperparameters Used in Experiments

We present hyper-parameters used in our experiments in Table 2 for the SPP-DDPG, in Table 6 for the SPP-SAC. SPP-TD3 hyper-parameters were dependent on the set of benchmarks, and

Algorithm 2: SPP-SAC Algorithm

repeat
  Sample random action $a \sim \mathcal{U}$;
  Store experience $(s_t, z_t, a_t, s_{t+1}, r_t)$ in $\mathcal{D}$; (use next-state as the initial actor actions)
until *random exploration is done*;
repeat
  if *buffer* $\mathcal{D}$ *is not full* **then**
    Sample actor prediction $z_t \sim \pi(s_t)$;
    Compute action $a_t = \mathsf{CM}(s_t, z_t)$;
    Observe reward r_t and next state s_{t+1};
    Store experience $(s_t, z_t, a_t, s_{t+1}, r_t)$ in $\mathcal{D}$;
    If s_{t+1} is terminal, reset environment state;
  end
  if *it's time to update* CM **then**
    Randomly sample $\{b_i = \{(s_t, s_{t+1}), a_t\}\}_{i=1}^{n}$ n 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 $\mathcal{B} = \{(s_t, z_t, a_t, s_{t+1}, r_t)\}$ set of batches from $\mathcal{D}$;
      Compute actions $\hat{a}_{t+1} = \mathsf{CM}(s_{t+1}, \hat{z}_{t+1})$, using samples $\hat{z}_{t+1} \sim \pi(z_{t+1}|s_{t+1})$;
      Compute targets $y = r_t + \gamma \min_{i=1,2} Q^{\pi,\mathsf{CM}}_{\phi_{targ,i}}(s_{t+1}, \hat{a}_{t+1})$;
      Compute $\phi = \phi - l_\phi \cdot \nabla_\phi \frac{1}{|\mathcal{B}|} \sum_{\mathcal{B}} \left(y - Q^{\pi,\mathsf{CM}}_{\phi}(s_t, a_t) \right)^2$;
      Compute $\theta = \theta + l_\theta \cdot \nabla_\theta \frac{1}{|\mathcal{B}|} \sum_{\mathcal{B}} (\min_{i=1,2} Q^{\pi,\mathsf{CM}}_{\phi}(s_t, \tilde{a}_{\theta,t}(s_t)) - \alpha \log \pi_\theta(\tilde{z}_{\theta,t}(s_t)|s_t)$,
       where $\tilde{z}_{\theta,t}(s_t) \sim \pi(z_t|s_t)$ is differentiable wrt θ;
      Compute $\theta = \theta - \frac{l_\theta \lambda}{|\mathcal{B}|} \cdot \nabla_\theta \sum_{\mathcal{B}} \| s_{t+1} - \tilde{z}_{\theta,t}(s_t) \|_2^2$;
      Update $\lambda = \lambda + l_\lambda \left(\frac{1}{|\mathcal{B}|} \sum_{\mathcal{B}} \| s_{t+1} - \tilde{z}_{\theta,t}(s_t) \|_2^2 - d \right)$;
      Update target networks $\phi_{targ,i} = (1-\tau)\phi_{targ,i} + \tau\phi_i$, for i = 1,2;
    end
  end
until *convergence*;

hyper-parameters specific for MuJoCo environments are in Tab. 3, for SafetyGym in Tab. 4, and for AntPush in Tab. 5.

In SPP-DDPG implementation we used the following architectures:

- Actor: $\dim(\mathcal{S}) \longrightarrow 256$ ReLU $\longrightarrow 256$ ReLU $\longrightarrow \dim(\mathcal{A})$ tanh;
- Critic: $\dim(\mathcal{S}) + \dim(\mathcal{A}) \longrightarrow 256$ ReLU $\longrightarrow 256$ ReLU $\rightarrow 1$;
- CM: $2 \cdot \dim(\mathcal{S}) \longrightarrow 100$ tanh $\longrightarrow 50$ tanh $\longrightarrow \dim(\mathcal{A})$ tanh;

In SPP-TD3 implementation we used the following architectures:

- Actor: $\dim(\mathcal{S}) \longrightarrow 256$ ReLU $\longrightarrow 256$ ReLU $\longrightarrow \dim(\mathcal{A})$ tanh;
- Critic: $\dim(\mathcal{S}) + \dim(\mathcal{A}) \longrightarrow 256$ ReLU $\longrightarrow 256$ ReLU $\rightarrow 1$;
- CM: $2 \cdot \dim(\mathcal{S}) \longrightarrow 100$ tanh $\longrightarrow 50$ tanh $\longrightarrow \dim(\mathcal{A})$ tanh;

In SPP-SAC implementation we used the following architectures:

- Actor: $\dim(\mathcal{S}) \longrightarrow 256$ ReLU $\longrightarrow 256$ ReLU $\longrightarrow \dim(\mathcal{A})$ tanh;
- Critic: $\dim(\mathcal{S}) + \dim(\mathcal{A}) \longrightarrow 256$ ReLU $\longrightarrow 256$ ReLU $\rightarrow 1$;
- CM: $2 \cdot \dim(\mathcal{S}) \longrightarrow 64$ tanh $\longrightarrow 32$ tanh $\longrightarrow \dim(\mathcal{A})$ tanh;

Algorithm 3: SPP-TD3 Algorithm

input : environment E; initial model parameters $\theta, \phi_1, \phi_2, \psi$; state planning distance threshold d; empty replay buffer $\mathcal{D}$; TD3 algorithm hyperparameters
output: trained model parameters $\theta, \phi_1, \phi_2, \psi$; total return
repeat
  Sample random action $a \sim \mathcal{U}$;
  Store experience $(s_t, a_t, z_t, r_{t+1}, s_{t+1})$ 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 $z_t = \text{clip}\left(\pi(s_t) + \varepsilon, z_{low}, z_{high}\right)$, 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 target states $\widetilde{z}_{t+1}(s_{t+1}) = \text{clip}\left(\pi_{\theta_{targ}}(s_{t+1}) + \text{clip}(\varepsilon, -c, c), z_{low}, z_{high}\right)$, where $\varepsilon \sim \mathcal{N}(0, \sigma)$;
      Compute target actions $\widetilde{a}_{t+1}(s_{t+1}) = \mathsf{CM}(s_{t+1}, \widetilde{z}_{t+1}(s_{t+1}))$;
      Compute targets $y(r, s_{t+1}) = r + \gamma \min_{i=1,2} Q^{\pi,\mathsf{CM}}_{\phi_{targ,i}}(s_{t+1}, \widetilde{a}_{t+1}(s_{t+1})$ (using target parameters $\phi_{targ,1}, \phi_{targ,2}$);
      Update $\phi_1 = \phi_1 - \frac{l_\phi}{|\mathcal{B}|} \cdot \nabla_{\phi_1} \sum_{\mathcal{B}} \left(y(r, s_{t+1}) - Q^{\pi,\mathsf{CM}}_{\phi_1}(s_t, a_t)\right)^2$;
      Update $\phi_2 = \phi_2 - \frac{l_\phi}{|\mathcal{B}|} \cdot \nabla_{\phi_2} \sum_{\mathcal{B}} \left(y(r, s_{t+1}) - Q^{\pi,\mathsf{CM}}_{\phi_2}(s_t, a_t)\right)^2$;
      if *current_step mod policy_delay = 0* **then**
        Update $\theta = \theta +$
        $l_\theta \left(\frac{1}{|\mathcal{B}|} \cdot \nabla_\theta \sum_{\mathcal{B}} Q^{\pi,\mathsf{CM}}_{\phi}(s_t, a_t)\Big|_{a_t = \mathsf{CM}(s_t, \pi_\theta(s_t))} - \frac{\lambda}{|\mathcal{B}|} \cdot \nabla_\theta \sum_{\mathcal{B}} \|s_{t+1} - \pi_\theta(s_t)\|_2^2 \right)$;
        Update $\lambda = \lambda + l_\lambda \left(\frac{1}{|\mathcal{B}|} \sum_{\mathcal{B}} \|s_{t+1} - \pi_\theta(s_t)\|_2^2 - d \right)$;
        Update actor & critics $\phi_{targ,i} = (1-\tau)\phi_{targ,i} + \tau\phi_i$; $\theta_{targ} = (1-\tau)\theta_{targ} + \tau\theta$;
      end
    end
  end
until *convergence*;

Hyperparameter	Value
γ	0.99
τ	0.005
actor/critic learning rate $l_\phi = l_\theta$	0.0005
λ learning rate l_λ	0.0001
episode length	5000
batch size	100
test episodes	10
update freq. & grad.steps.	50
exploration noise param.	0.05
replay buffer size	same as number of env. interacts
parameters specific for SPP-DDPG	
d state consistency	0.2
CM hyper-parameters	
init. rand. samples	20000
learning rate	0.005
batch size	128
update frequency	500 steps
update epochs	1
update batches	100

Table 2: SPP-DDPG Algorithm hyperparameters, DDPG used the same hyperparameters, except exploration noise (set to 0.1).

Hyperparameter	Value
γ	0.99
τ	0.005
actor/critic learning rate $l_\phi = l_\theta$	0.0001
λ learning rate l_λ	0.0001
episode length	5000
batch size	100
test episodes	10
update freq. & grad.steps.	50
exploration noise param.	0.2
policy noise	0.2
noise clip.	0.5
replay buffer size	same as number of env. interacts
parameters specific for SPP-TD3	
d state consistency	0.2
CM hyper-parameters	
init. rand. samples	25000
learning rate	0.001
batch size	128
update frequency	500 steps
update epochs	1
update batches	200

Table 3: SPP-TD3 Algorithm hyperparameters for MuJoCo environments, TD3 used the same (common) hyperparameters.

Hyperparameter	Value
γ	0.99
τ	0.005
actor/critic learning rate $l_\phi = l_\theta$	0.0002
λ learning rate l_λ	0.0001
episode length	5000
batch size	100
test episodes	10
update freq. & grad.steps.	50
exploration noise param.	0.1
policy noise	0.2
noise clip.	0.5
replay buffer size	same as number of env. interacts
parameters specific for SPP-TD3	
d state consistency	0.2
CM hyper-parameters	
init. rand. samples	400000
learning rate	0.001
batch size	128
update frequency	500 steps
update epochs	1
update batches	200

Table 4: SPP-TD3 Algorithm hyperparameters for SafetyGym environments. Observe that for this environment specifically we used larger number of CM pretrain samples

Hyperparameter	Value
γ	0.99
τ	0.005
actor/critic learning rate $l_\phi = l_\theta$	0.0001
λ learning rate l_λ	0.0001
episode length	5000
batch size	100
test episodes	10
update freq. & grad.steps.	50
exploration noise param.	1
policy noise	0.2
noise clip.	0.5
replay buffer size	250000
parameters specific for SPP-TD3	
d state consistency	0.2
CM hyper-parameters	
init. rand. samples	100000
learning rate	0.001
batch size	128
update frequency	500 steps
update epochs	1
update batches	200

Table 5: SPP-TD3 Algorithm hyperparameters for AntPush environment. Observe that for this environment specifically we used fixed buffer size which is significantly smaller than the total number of env interacts, and a much larger exploration noise param equal to 1.

Hyperparameter	Value
γ	0.99
τ	0.005
actor/critic learning rate $l_\phi = l_\theta$	0.001
λ learning rate l_λ	0.0001
batch size	100
test episodes	10 each 1000 frames
update freq. & grad.steps.	50
α	0.2
α learning rate	0.001
replay buffer size	$1e6$
parameters specific for SPP-SAC	
d state consistency	0.2
CM hyper-parameters	
init. rand. samples	10000
learning rate	0.001
batch size	100
update frequency	1000 steps
update batches	100

Table 6: SPP-SAC Algorithm hyperparameters