## A Appendix

### Index of the appendix

In the following, we briefly recap the contents of the appendix.

- Appendix A.1 contains additional related works
- Appendix A.2 reports all proofs and derivations.
- Appendix A.3 illustrates implementation details and pseudocode.
- Appendix A.4 provides the hyperparameters used in the experiments and further results.

### A.1 Additional related works

Recently we learned about Policy Evaluation Networks (PENs) (Harb et al., 2020) which are closely related to our work and share the same motivation. There are many differences between our approach to learning $V(\theta)$ and theirs (Harb et al., 2020). For example, we do not use a fingerprint mechanism (Harb et al., 2020) for embedding the weights of complex policies. Instead, we simply parse all the policy weights as inputs to the value function, even in the nonlinear case. However, fingerprinting (Harb et al., 2020) may be important for representing nonlinear policies without losing information about their structure and for saving memory required to store the weights.

Other differences concern the optimization problem: we do not predict a bucket index for discretized reward, but perform a regression task. Therefore our loss is simply the mean squared error between the prediction of $V(\theta)$ and the reward obtained by $\pi_\theta$, while their loss (Harb et al., 2020) is the KL divergence between the predicted and target distributions. Both approaches optimize the undiscounted objective when learning $V(\theta)$.

Harb et al. (2020) focus on the offline setting. They first collect a batch of randomly initialized policies and perform rollouts to collect reward from the environment. The PSSVF $V(\theta)$ is then trained using the data collected. Once V is trained, many gradient ascent steps through V yield new, unseen, randomly initialized policies in a zero-shot manner, exhibiting improved performance. In our offline experiment in section 4.4, we used a more difficult setting where each episode in the environment is composed by sub-trajectories generated using different policies. Episode-based approaches like PSSVF are not suitable for this task, while our PSVF can effectively learn how to improve the policy without waiting for the end of an episode.

Harb et al. (2020) train their value function using small nonlinear policies of one hidden layer and 30 neurons on Swimmer-v3. They evaluate 2000 deterministic policies on 500 episodes each (1 million policy evaluations), achieving a final expected return of $\approx$ 180 on new policies trained from scratch through V and a maximum observed return of 250. On the other hand, in our offline experiment using a linear PSSVF, after only 100 policy evaluations, we obtain a return of 297.

In our main experiments, we showed that a fingerprint mechanism is not necessary for the tasks we analyzed: even when using a much bigger 2-layers MLP policy, we are able to outperform the results in PEN. Although Harb et al. (2020) use Swimmer-v3 "to scale up their experiments", our results suggest that Swimmer-v3 does not conclusively demonstrate possible benefits of their policy embedding.

### A.2 Proofs and derivations

**Theorem 3.1.** *For any Markov Decision Process, the following holds:*

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim d^{\pi_b}(s), a\sim\pi_b(.|s)}\left[\frac{\pi_\theta(a|s)}{\pi_b(a|s)}\left(Q(s,a,\theta)\nabla_\theta \log\pi_\theta(a|s) + \nabla_\theta Q(s,a,\theta)\right)\right]. \quad (9)$$

*Proof.*

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \int_{\mathcal{S}} d^{\pi_b}(s) V(s,\theta)\,\mathrm{d}s \tag{13}$$

$$= \nabla_\theta \int_{\mathcal{S}} d^{\pi_b}(s) \int_{\mathcal{A}} \pi_\theta(a|s) Q(s,a,\theta)\,\mathrm{d}a\,\mathrm{d}s \tag{14}$$

$$= \int_{\mathcal{S}} d^{\pi_b}(s) \int_{\mathcal{A}} [Q(s,a,\theta)\nabla_\theta \pi_\theta(a|s) + \pi_\theta(a|s)\nabla_\theta Q(s,a,\theta)]\,\mathrm{d}a\,\mathrm{d}s \tag{15}$$

$$= \int_{\mathcal{S}} d^{\pi_b}(s) \int_{\mathcal{A}} \frac{\pi_b(a|s)}{\pi_b(a|s)} \pi_\theta(a|s)[Q(s,a,\theta)\nabla_\theta \log \pi_\theta(a|s) + \nabla_\theta Q(s,a,\theta)]\,\mathrm{d}a\,\mathrm{d}s \tag{16}$$

$$= \mathbb{E}_{s\sim d^{\pi_b}(s), a\sim \pi_b(.|s)} \left[ \frac{\pi_\theta(a|s)}{\pi_b(a|s)} \left( Q(s,a,\theta)\nabla_\theta \log \pi_\theta(a|s) + \nabla_\theta Q(s,a,\theta) \right) \right] \tag{17}$$

□

**Theorem 3.2.** *Under standard regularity assumptions (Silver et al., 2014), for any Markov Decision Process, the following holds:*

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim d^{\pi_b}(s)} \left[ \nabla_a Q(s,a,\theta)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s) + \nabla_\theta Q(s,a,\theta)|_{a=\pi_\theta(s)} \right]. \tag{12}$$

*Proof.*

$$\nabla_\theta J(\pi_\theta) = \int_{\mathcal{S}} d^{\pi_b}(s) \nabla_\theta Q(s,\pi_\theta(s),\theta)\,\mathrm{d}s \tag{18}$$

$$= \int_{\mathcal{S}} d^{\pi_b}(s) \left[ \nabla_a Q(s,a,\theta)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s) + \nabla_\theta Q(s,a,\theta)|_{a=\pi_\theta(s)} \right] \mathrm{d}s \tag{19}$$

$$= \mathbb{E}_{s\sim d^{\pi_b}(s)} \left[ \nabla_a Q(s,a,\theta)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s) + \nabla_\theta Q(s,a,\theta)|_{a=\pi_\theta(s)} \right] \tag{20}$$

□

## A.3 Implementation details

### A.3.1

In this appendix, we report the implementation details for PSSVF, PSVF, PAVF and the baselines. We specify for each hyperparameter, which algorithms and tasks are sharing them.

Shared hyperparameters:

- Policy architecture (continuous control tasks): We use three different deterministic policies: a linear mapping between states and actions; a single-layer MLP with 32 neurons and tanh activation; a 2-layers MLP (64,64) with tanh activations. All policies contain a bias term and are followed by a tanh nonlinearity in order to bound the action.
- Policy architecture (discrete control tasks): We use three different deterministic policies: a linear mapping between states and a probability distribution over actions; a single-layer MLP with 32 neurons and tanh activation; a 2-layers MLP (64,64) with tanh activations. The deterministic action $a$ is obtained choosing $a = \arg\max \pi_\theta(a|s)$. All policies contain a bias term.
- Policy initialization: all weights and biases are initialized using the default Pytorch initialization for PVFs and DDPG and are set to zero for ARS.
- Critic architecture: 2-layers MLP (512,512) with bias and ReLU activation functions for PSVF, PAVF; 2-layers MLP (256,256) with bias and ReLU activation functions for DDPG.
- Critic initialization: all weights and biases are initialized using the default Pytorch initialization for PVFs and DDPG.
- Batch size: 128 for DDPG, PSVF, PAVF; 16 for PSSVF.
- Actor's frequency of updates: every episode for PSSVF; every batch of episodes for ARS; every 50 time steps for DDPG, PSVF, PAVF.

- Critic's frequency of updates: every episode for PSSVF; every 50 time steps for DDPG, PSVF, PAVF.
- Replay buffer: the size is 100k; data are sampled uniformly.
- Optimizer: Adam for PVFs and DDPG.

Tuned hyperparameters:

- Number of directions and elite directions for ARS ([directions, elite directions]): tuned with values in $[[1,1],[4,1],[4,4],[16,1],[16,4],[16,16]]$.
- Policy's learning rate: tuned with values in $[1e-2, 1e-3, 1e-4]$.
- Critic's learning rate: tuned with values in $[1e-2, 1e-3, 1e-4]$.
- Noise for exploration: the perturbation for the action (DDPG) or the parameter is sampled from $\mathcal{N}(0, \sigma I)$ with $\sigma$ tuned with values in $[1, 1e-1]$ for PSSVF, PSVF, PAVF; $[1e-1, 1e-2]$ for DDPG; $[1, 1e-1, 1e-2, 1e-3]$ for ARS.

Environment hyperparameters:

- Environment interactions: 1M time steps for Swimmer-v3 and Hopper-v3; 100k time steps for all other environments.
- Discount factor for TD algorithms: 0.999 for Swimmer; 0.99 for all other environments.
- Survival reward in Hopper: True for DDPG, PSVF, PAVF; False for ARS, PSSVF.

Algorithm-specific hyperparameters:

- Critic's number of updates: 50 for DDPG, 5 for PSVF and PAVF; 10 for PSSVF.
- Actor's number of updates: 50 for DDPG, 1 for PSVF and PAVF; 10 for PSSVF.
- Observation normalization: False for DDPG; True for all other algorithms.
- Starting steps in DDPG (random actions and no training): first $1\%$.
- Polyak parameter in DDPG: 0.995.

**ARS** For ARS, we used the official implementation provided by the authors and we modified it in order to use nonlinear policies. More precisely, we used the implementation of ARSv2-t (Mania et al., 2018), which uses observation normalization, elite directions and an adaptive learning rate based on the standard deviation of the return collected. To avoid divisions by zero, which may happen if all data sampled have the same return, we perform the standardization only in case the standard deviation is not zero. In the original implementation of ARS (Mania et al., 2018), the survival bonus for the reward in the Hopper environment is removed to avoid local minima. Since we wanted our PSSVF to be close to their setting, we also applied this modification. We did not remove the survival bonus from all TD algorithms and we did not investigate how this could affect their performance. We provide a comparison of the performance of PSSVF with and without the bonus in figure 5.

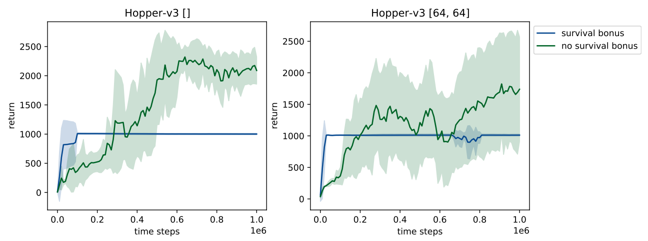

Figure 5: Performance of PSSVF with and without the survival bonus for the reward in Hopper-v3 when using the hyperparameters maximizing the average return. Learning curves are averaged over 5 seeds

**DDPG** For DDPG, we used the Spinning Up implementation provided by OpenAI (Achiam, 2018), which includes target networks for the actor and the critic and no learning for a fixed set of time steps, called starting steps. We did not include target networks and starting steps in our PVFs, although they could potentially help stabilizing training. The implementation of DDPG that we used (Achiam, 2018) does not use observation normalization. In preliminary experiments we observed that it failed to significantly increase or decrease performance, hence we did not use it. Another difference between our TD algorithms and DDPG consists in the number of updates of the actor and the critic. Since DDPG's critic needs to keep track of the current policy, the critic and the actor are updated in a nested form, with the first's update depending on the latter and vice versa. Our PSVF and PAVF do not need to track the policy learned, hence, when it is time to update, we need only to train once the critic for many gradient steps and then train the actor for many gradient steps. This requires less compute. On the other hand, when using nonlinear policies, our PVFs suffer the curse of dimensionality. For this reason, we profited from using a bigger critic. In preliminary experiments, we observed that DDPG's performance did not change significantly through a bigger critic. We show differences in performance for our methods when removing observation normalization and when using a smaller critic (MLP(256,256)) in figure 6. We observe that the performance is decreasing if observation normalization is removed. However, only for shallow policies in Swimmer and deep policies in Hopper there seems to be a significant benefit. Future work will assess when bigger critics help.

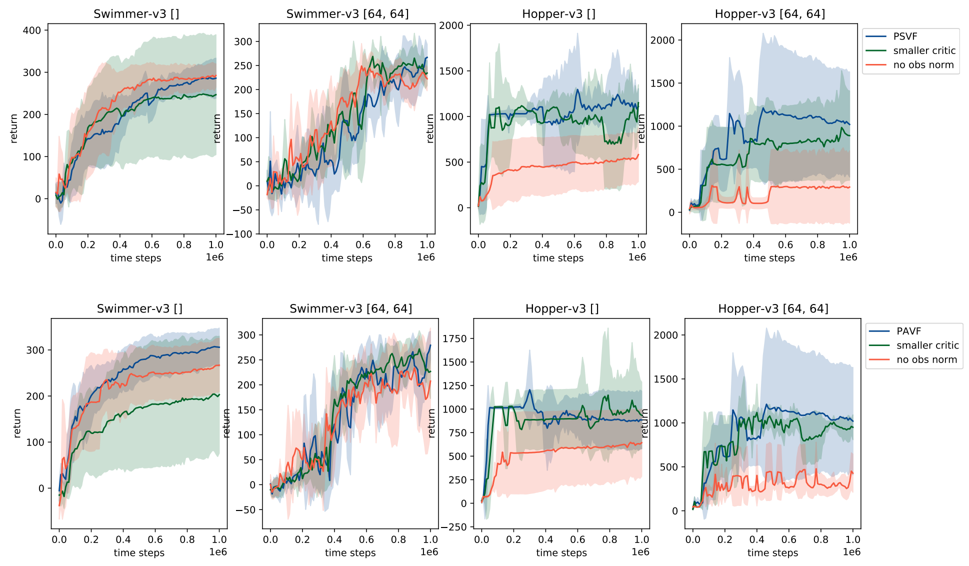

Figure 6: Learning curves for PSVF and PAVF for different environments and policies removing observation normalization and using a smaller critic. We use the hyperparameters maximizing the average return. Learning curves are averaged over 5 seeds.

**Discounting in Swimmer** For TD algorithms, we chose a fixed discount factor $\gamma = 0.99$ for all environments but Swimmer-v3. This environment is known to be challenging for TD based algorithms because discounting causes the agents to become too short-sighted. We observed that, with the standard discounting, DDPG, PSVF and PAVF were not able to learn the task. However, making the algorithms more far-sighted greatly improved their performance. In figure 7 we report the return obtained by DDPG, PSVF and PAVF for different values of the discount factor in Swimmer.

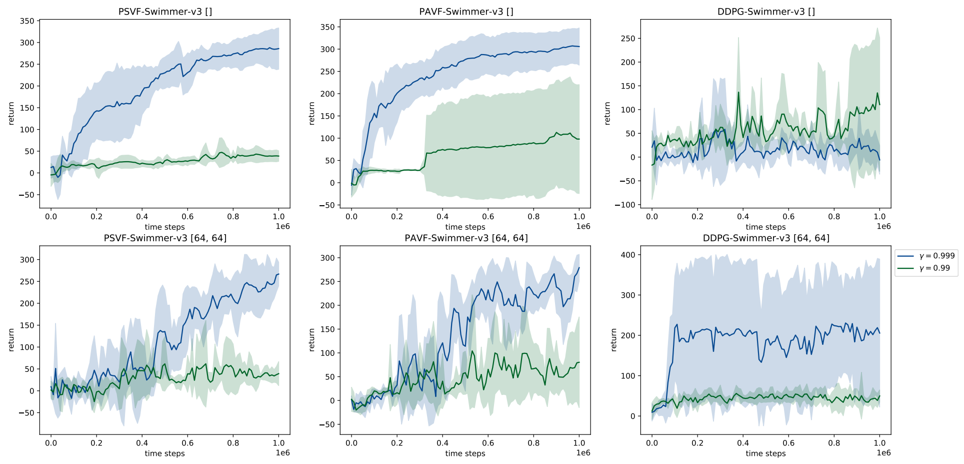

Figure 7: Effect of different choices of the discount factor in Swimmer-v3 for PSVF, PAVF and DDPG, with shallow and deep policies. We use the hyperparameters maximizing the average return. Learning curves are averaged over 5 seeds

### A.3.2 PSEUDOCODE

**Algorithm 2** Actor-critic with TD prediction for $V(s, \theta)$

```
    **Input**: Differentiable critic $V_{\mathbf{w}} : \mathcal{S} \times \Theta \rightarrow \mathcal{R}$ with parameters $\mathbf{w}$; deterministic actor $\pi_\theta$ with parameters $\theta$; empty replay buffer $D$
    **Output** : Learned $V_{\mathbf{w}} \approx V(s, \theta)$, learned $\pi_\theta \approx \pi_{\theta^*}$
  Initialize critic and actor weights $\mathbf{w}, \theta$
  **repeat**:
      Observe state s, take action $a = \pi_\theta(s)$, observe reward $r$ and next state $s'$
      Store $(s, \theta, r, s')$ in the replay buffer $D$
      **if** it's time to update **then**:
          **for** many steps **do**:
              Sample a batch $B_1 = \{(s, \tilde{\theta}, r, s')\}$ from $D$
              Update critic by stochastic gradient descent:
              $\nabla_{\mathbf{w}} \frac{1}{|B_1|} \mathbb{E}_{(s,\tilde{\theta},r,s') \in B_1} [V_{\mathbf{w}}(s, \tilde{\theta}) - (r + \gamma V_{\mathbf{w}}(s', \tilde{\theta}))]^2$
          **end for**
          **for** many steps **do**:
              Sample a batch $B_2 = \{(s)\}$ from $D$
              Update actor by gradient ascent: $\nabla_\theta \frac{1}{|B_2|} \mathbb{E}_{s \in B_2} [V_{\mathbf{w}}(s, \theta)]$
          **end for**
      **end if**
  **until** convergence
```

**Algorithm 3** Stochastic actor-critic with TD prediction for $Q(s, a, \theta)$

```
**Input**: Differentiable critic $Q_\mathbf{w} : \mathcal{S} \times \mathcal{A} \times \Theta \to \mathcal{R}$ with parameters $\mathbf{w}$; stochastic differentiable actor $\pi_\theta$ with parameters $\theta$; empty replay buffer $D$
**Output** : Learned $Q_\mathbf{w} \approx Q(s, a, \theta)$, learned $\pi_\theta \approx \pi_{\theta^*}$
Initialize critic and actor weights $\mathbf{w}, \theta$
repeat:
    Observe state s, take action $a = \pi_\theta(s)$, observe reward $r$ and next state $s'$
    Store $(s, a, \theta, r, s')$ in the replay buffer $D$
    if it's time to update then:
        for many steps do:
            Sample a batch $B_1 = \{(s, a, \tilde{\theta}, r, s')\}$ from $D$
            Update critic by stochastic gradient descent:
            $\nabla_\mathbf{w} \frac{1}{|B_1|} \mathbb{E}_{(s,a,\tilde{\theta},r,s') \in B_1} [Q_\mathbf{w}(s, a, \tilde{\theta}) - (r + \gamma Q_\mathbf{w}(s', a' \sim \pi_{\tilde{\theta}}(s'), \tilde{\theta}))]^2$
        end for
        for many steps do:
            Sample a batch $B_2 = \{(s, a, \tilde{\theta})\}$ from $D$
            Update actor by stochastic gradient ascent:
            $\frac{1}{|B_2|} \mathbb{E}_{(s,a,\tilde{\theta}) \in B_2} \left[ \frac{\pi_\theta(a|s)}{\pi_{\tilde{\theta}}(a|s)} \left( Q(s, a, \theta) \nabla_\theta \log \pi_\theta(a|s) + \nabla_\theta Q(s, a, \theta) \right) \right]$
        end for
    end if
until convergence
```

**Algorithm 4** Deterministic actor-critic with TD prediction for $Q(s, a, \theta)$

```
**Input**: Differentiable critic $Q_\mathbf{w} : \mathcal{S} \times \mathcal{A} \times \Theta \to \mathcal{R}$ with parameters $\mathbf{w}$; differentiable deterministic actor $\pi_\theta$ with parameters $\theta$; empty replay buffer $D$
**Output** : Learned $Q_\mathbf{w} \approx Q(s, a, \theta)$, learned $\pi_\theta \approx \pi_{\theta^*}$
Initialize critic and actor weights $\mathbf{w}, \theta$
repeat:
    Observe state s, take action $a = \pi_\theta(s)$, observe reward $r$ and next state $s'$
    Store $(s, a, \theta, r, s')$ in the replay buffer $D$
    if it's time to update then:
        for many steps do:
            Sample a batch $B_1 = \{(s, a, \tilde{\theta}, r, s')\}$ from $D$
            Update critic by stochastic gradient descent:
            $\nabla_\mathbf{w} \frac{1}{|B_1|} \mathbb{E}_{(s,a,\tilde{\theta},r,s') \in B_1} [Q_\mathbf{w}(s, a, \tilde{\theta}) - (r + \gamma Q_\mathbf{w}(s', \pi_{\tilde{\theta}}(s'), \tilde{\theta}))]^2$
        end for
        for many steps do:
            Sample a batch $B_2 = \{(s)\}$ from $D$
            Update actor by stochastic gradient ascent:
            $\frac{1}{|B_2|} \mathbb{E}_{s \in B_2} [\nabla_\theta \pi_\theta(s) \nabla_a Q_\mathbf{w}(s, a, \theta)|_{a=\pi_\theta(s)} + \nabla_\theta Q_\mathbf{w}(s, a, \theta)|_{a=\pi_\theta(s)}]$
        end for
    end if
until convergence
```

## A.4 Experimental details

### A.4.1 LQR

For our visualization experiment, we employ an instance of the Linear Quadratic Regulator. Here, the agent observes a 1-D state, corresponding to its position and chooses a 1-D action. The transitions are $s' = s + a$ and there is a quadratic negative term for the reward: $R(s, a) = -s^2 - a^2$. The agent starts in state $s_0 = 1$ and acts in the environment for 50 time steps. The state space is bounded in [-2,2]. The goal of the agent is to reach and remain in the origin. The agent is expected to perform small steps towards the origin when it uses the optimal policy. For this task, we do not use a tanh nonlinearity and we do not use observation normalization. We use a learning rate of $1e-3$ for the policy and $1e-2$ for the PSSVF. Weights are perturbed every episode using $\sigma = 0.5$. The policy is initialized with weight $3.2$ and bias $-3.5$. All the other hyperparameters are set to their default. The true $J(\theta)$ is computed by running 10,000 policies in the environment with parameters in $[-5,-5] \times [-5,5]$. $V_w(\theta)$ is computed by measuring the output of the PSSVF on the same set of policies. Each red arrow in figure 1 represents 200 update steps of the policy.

### A.4.2 Offline experiments

**Zero-shot learning** In this task we use the same hyperparameters found in tables 3, 5 and 6. When new policies are trained offline, we use a learning rate of 0.02. We evaluate the performance of the policies learned from scratch evaluating them with 5 test trajectories every 5 gradient steps.

**Offline learning with fragmented behaviors** In this task, data are generated by perturbing a randomly initialized policy every 200 time steps and using it to act in the environment. We use $\sigma = 0.5$ for the perturbations. After the dataset is collected, the PSVF is trained using a learning rate of $1e-3$ with a batch size of 128. When the policy is learned, we use a learning rate of 0.02. All other hyperparameters are set to default values.

### A.4.3 Full experimental results

**Methodology** In order to ensure a fair comparison of our methods and the baselines, we adopt the following procedure. For each hyperparameter configuration, for each environment and policy architecture, we run 5 instances of the learning algorithm using different seeds. We measure the learning progress by running 100 evaluations while learning the deterministic policy (without action or parameter noise) using 10 test trajectories. We use two metrics to determine the best hyperparameters: the average return over policy evaluations during the whole training process and the average return over policy evaluations during the last 20% time steps. For each algorithm, environment and policy architecture, we choose the two hyperparameter configurations maximizing the performance of the two metrics and test them on 20 new seeds, reporting average and final performance in table 1 and 2 respectively.

Figures 8 and 9 report all the learning curves from the main paper and for a small non linear policy with 32 hidden neurons.

### A.4.4 Sensitivity analysis

In the following, we report the sensitivity plots for all algorithms, policies and environments. In particular, figure 10, 11, 12, 13 and 14 show the performance of each algorithm given different hyperparameters tried during training. We observe that in general deep policies are more sensitive and, apart for DDPG, achieve often a better performance than smaller policies. The higher sensitivity displayed by ARS is in part caused by the higher number of hyperparameters we tried when tuning the algorithm.

### A.4.5 Table of best hyperparameters

We report for each algorithm, environment, and policy architecture the best hyperparameters found when optimizing for average return or final return in tables 3, 4, 5, 6 and 7.

Table 1: Average return with standard deviation (across 20 seeds) for hypermarameters optimizing the average return during training. Square brackets represent the number of neurons per layer of the policy. [] represents a linear policy.

| **Policy: []** | MountainCar Continuous-v0 | Inverted Pendulum-v2 | Reacher -v2 | Swimmer -v3 | Hopper -v3 |
|---|---|---|---|---|---|
| ARS | $63 \pm 6$ | $886 \pm 72$ | $-9.2 \pm 0.3$ | $228 \pm 89$ | $1184 \pm 345$ |
| PSSVF | $85 \pm 4$ | $944 \pm 33$ | $-11.7 \pm 0.9$ | $259 \pm 47$ | $1392 \pm 287$ |
| DDPG | $0 \pm 0$ | $612 \pm 169$ | $-8.6 \pm 0.9$ | $95 \pm 112$ | $629 \pm 145$ |
| PSVF | $84 \pm 20$ | $926 \pm 34$ | $-19.7 \pm 6.0$ | $188 \pm 71$ | $917 \pm 249$ |
| PAVF | $82 \pm 21$ | $913 \pm 40$ | $-17.0 \pm 7.7$ | $231 \pm 56$ | $814 \pm 223$ |
| **Policy:[32]** | | | | | |
| ARS | $37 \pm 11$ | $851 \pm 46$ | $-9.6 \pm 0.3$ | $139 \pm 78$ | $1003 \pm 66$ |
| PSSVF | $60 \pm 33$ | $701 \pm 138$ | $10.4 \pm 0.5$ | $189 \pm 35$ | $707 \pm 668$ |
| DDPG | $0 \pm 0$ | $816 \pm 36$ | $-5.7 \pm 0.3$ | $61 \pm 32$ | $1384 \pm 125$ |
| PSVF | $71 \pm 25$ | $529 \pm 281$ | $-11.9 \pm 1.2$ | $226 \pm 33$ | $864 \pm 272$ |
| PAVF | $71 \pm 27$ | $563 \pm 228$ | $-10.9 \pm 1.1$ | $222 \pm 28$ | $793 \pm 322$ |
| **Policy: [64,64]** | | | | | |
| ARS | $28 \pm 8$ | $812 \pm 239$ | $-9.8 \pm 0.3$ | $129 \pm 68$ | $964 \pm 47$ |
| PSSVF | $72 \pm 22$ | $850 \pm 93$ | $-10.7 \pm 0.2$ | $158 \pm 59$ | $922 \pm 568$ |
| DDPG | $0 \pm 0$ | $834 \pm 36$ | $-5.5 \pm 0.4$ | $92 \pm 117$ | $767 \pm 627$ |
| PSVF | $80 \pm 9$ | $580 \pm 107$ | $-10.7 \pm 0.6$ | $137 \pm 38$ | $843 \pm 282$ |
| PAVF | $73 \pm 10$ | $399 \pm 219$ | $-10.7 \pm 0.5$ | $142 \pm 26$ | $875 \pm 136$ |

| **Policy: []** | Acrobot-v1 | CartPole-v1 |
|---|---|---|
| ARS | $-161 \pm 23$ | $476 \pm 13$ |
| PSSVF | $-137 \pm 14$ | $443 \pm 105$ |
| PSVF | $-148 \pm 25$ | $459 \pm 28$ |
| **Policy:[32]** | | |
| ARS | $-296 \pm 38$ | $395 \pm 141$ |
| PSSVF | $-251 \pm 80$ | $463 \pm 18$ |
| PSVF | $-270 \pm 113$ | $413 \pm 61$ |
| **Policy: [64,64]** | | |
| ARS | $-335 \pm 35$ | $416 \pm 105$ |
| PSSVF | $-281 \pm 117$ | $452 \pm 34$ |
| PSVF | $-397 \pm 71$ | $394 \pm 71$ |

Table 2: Final return with standard deviation (across 20 seeds) for hypermarameters optimizing the final return during training.

| **Policy: []** | MountainCar Continuous-v0 | Inverted Pendulum-v2 | Reacher -v2 | Swimmer -v3 | Hopper -v3 |
|---|---|---|---|---|---|
| ARS | $73 \pm 5$ | $657 \pm 477$ | $-8.6 \pm 0.5$ | $334 \pm 34$ | $1443 \pm 713$ |
| PSSVF | $84 \pm 28$ | $970 \pm 126$ | $-10.0 \pm 1.0$ | $350 \pm 8$ | $1560 \pm 911$ |
| DDPG | $0 \pm 1$ | $777 \pm 320$ | $-7.3 \pm 0.4$ | $146 \pm 152$ | $704 \pm 234$ |
| PSVF | $76 \pm 36$ | $906 \pm 289$ | $-16.5 \pm 1.6$ | $238 \pm 107$ | $1067 \pm 340$ |
| PAVF | $68 \pm 42$ | $950 \pm 223$ | $-17.2 \pm 15.4$ | $298 \pm 40$ | $720 \pm 281$ |
| **Policy:[32]** | | | | | |
| ARS | $54 \pm 20$ | $936 \pm 146$ | $-9.2 \pm 0.4$ | $239 \pm 117$ | $1048 \pm 68$ |
| PSSVF | $89 \pm 22$ | $816 \pm 234$ | $-10.2 \pm 1.0$ | $294 \pm 41$ | $1204 \pm 615$ |
| DDPG | $0 \pm 0$ | $703 \pm 283$ | $-4.6 \pm 0.6$ | $179 \pm 150$ | $1290 \pm 348$ |
| PSVF | $84 \pm 31$ | $493 \pm 462$ | $-11.3 \pm 0.8$ | $290 \pm 70$ | $1003 \pm 572$ |
| PAVF | $92 \pm 7$ | $854 \pm 295$ | $-10.1 \pm 0.9$ | $307 \pm 34$ | $967 \pm 411$ |
| **Policy: [64,64]** | | | | | |
| ARS | $11 \pm 30$ | $976 \pm 83$ | $-9.4 \pm 0.4$ | $157 \pm 54$ | $1006 \pm 47$ |
| PSSVF | $91 \pm 16$ | $898 \pm 227$ | $-10.7 \pm 0.6$ | $224 \pm 99$ | $1412 \pm 691$ |
| DDPG | $0 \pm 0$ | $943 \pm 73$ | $-4.4 \pm 0.4$ | $196 \pm 151$ | $1437 \pm 752$ |
| PSVF | $93 \pm 1$ | $1000 \pm 0$ | $-10.6 \pm 1.0$ | $257 \pm 26$ | $1247 \pm 344$ |
| PAVF | $93 \pm 2$ | $827 \pm 267$ | $-10.6 \pm 0.4$ | $232 \pm 42$ | $1005 \pm 155$ |

| **Policy: []** | Acrobot-v1 | CartPole-v1 |
|---|---|---|
| ARS | $-126 \pm 26$ | $499 \pm 2$ |
| PSSVF | $-97 \pm 6$ | $482 \pm 53$ |
| PSVF | $-100 \pm 18$ | $500 \pm 0$ |
| **Policy:[32]** | | |
| ARS | $-215 \pm 97$ | $471 \pm 110$ |
| PSSVF | $-116 \pm 33$ | $500 \pm 0$ |
| PSVF | $-244 \pm 151$ | $488 \pm 36$ |
| **Policy: [64,64]** | | |
| ARS | $-182 \pm 45$ | $492 \pm 18$ |
| PSSVF | $-233 \pm 139$ | $500 \pm 0$ |
| PSVF | $-406 \pm 51$ | $499 \pm 2$ |

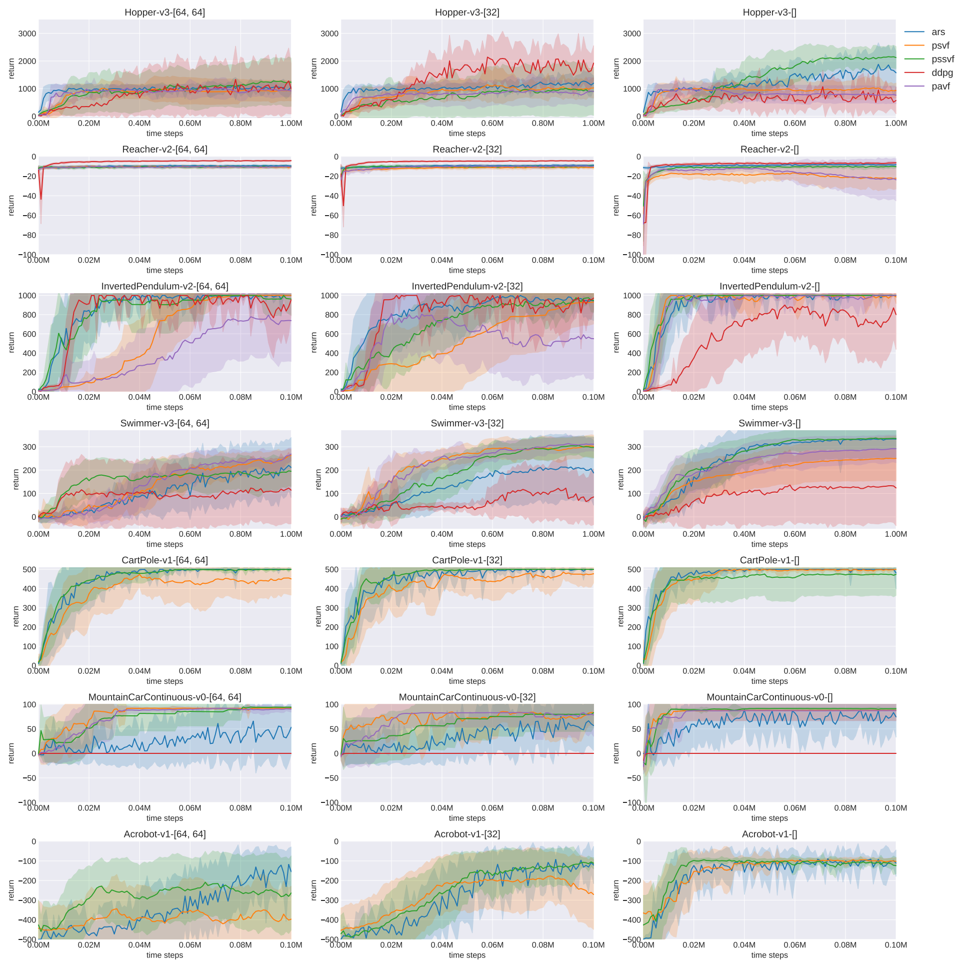

Figure 8: Learning curves representing the average return as a function of the number of time steps in the environment (across 20 runs) with different environments and policy architectures. We use the **best hyperparameters found while maximizing the average reward** for each task. For each subplot, the square brackets represent the number of neurons per policy layer. [] represents a linear policy.

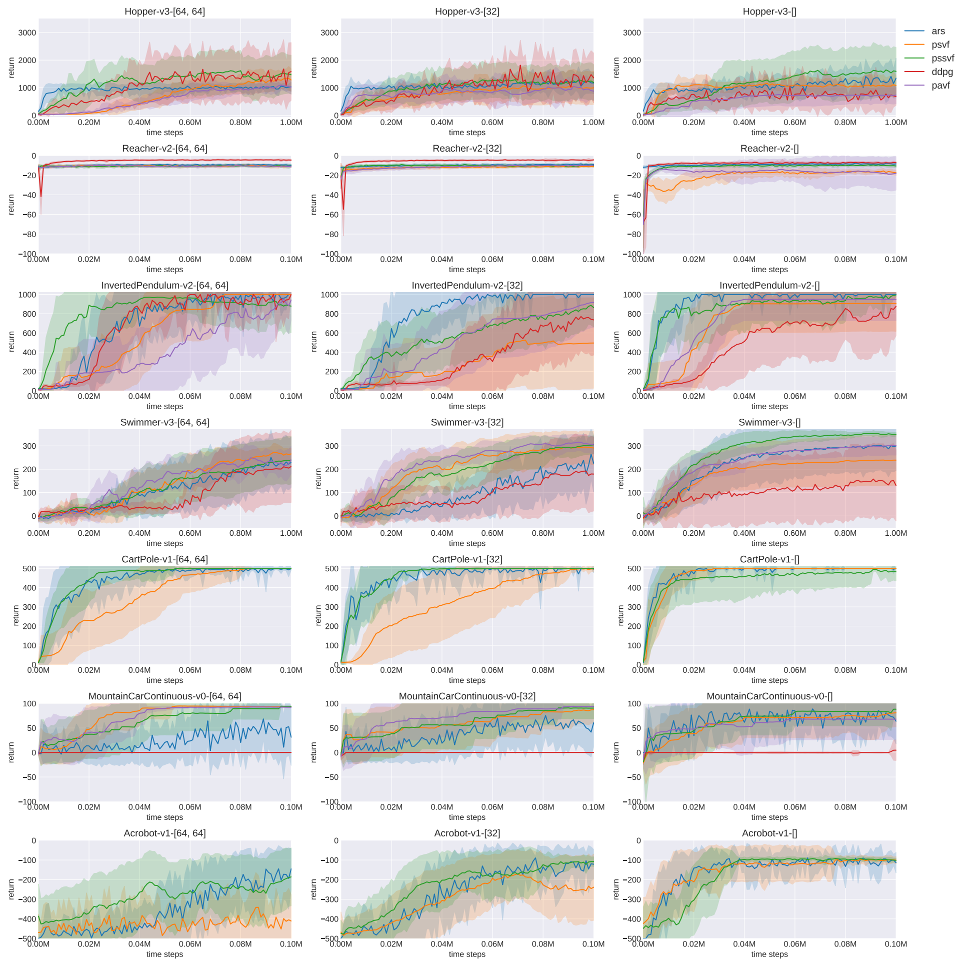

Figure 9: Learning curves representing the average return as a function of the number of time steps in the environment (across 20 runs) with different environments and policy architectures. We use the **best hyperparameters found while maximizing the final reward for each task**. For each subplot, the square brackets represent the number of neurons per policy layer. [] represents a linear policy.

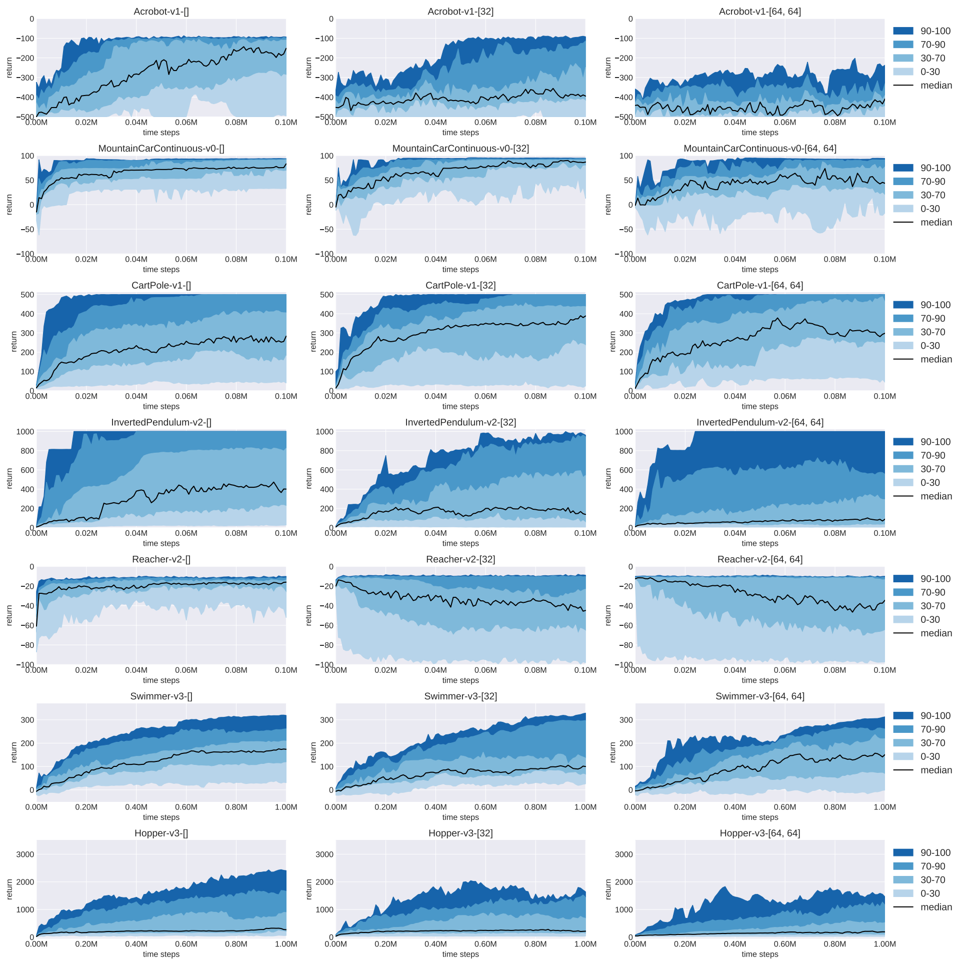

Figure 10: **Sensitivity of PSSVFs** to the choice of the hyperparameter. Performance is shown by percentile using all the learning curves obtained during hyperparameter tuning. The median performance is depicted as a dark line. For each subplot, the numbers in the square brackets represent the number of neurons per layer of the policy. [] represents a linear policy.

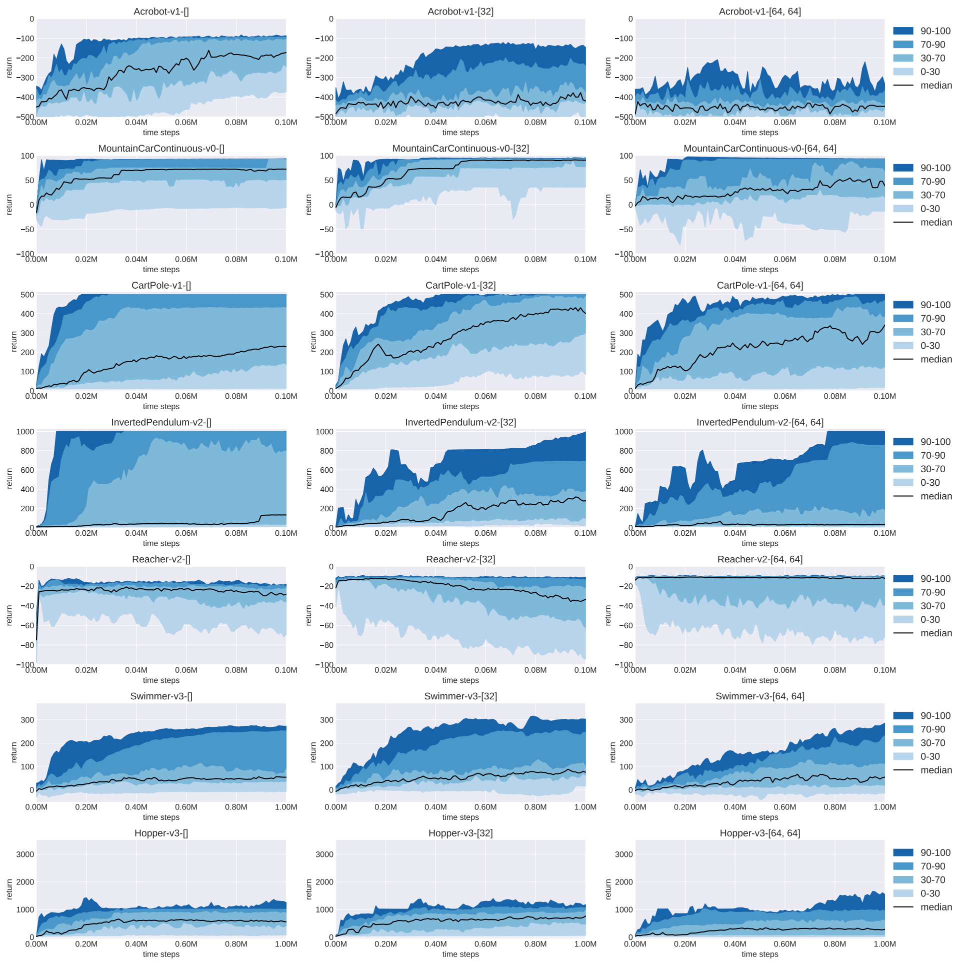

Figure 11: **Sensitivity of PSVFs** to the choice of the hyperparameter. Performance is shown by percentile using all the learning curves obtained during hyperparameter tuning. The median performance is depicted as a dark line. For each subplot, the numbers in the square brackets represent the number of neurons per layer of the policy. [] represents a linear policy.

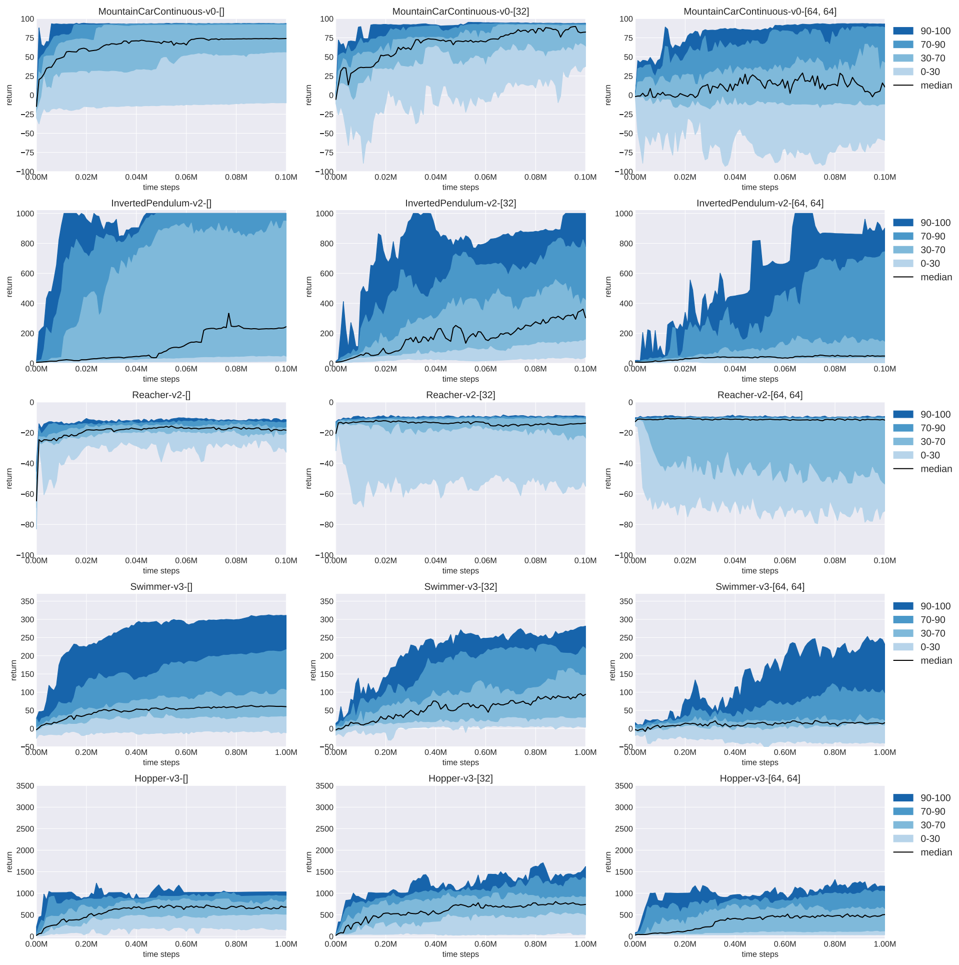

Figure 12: **Sensitivity of PAVFs** to the choice of the hyperparameter. Performance is shown by percentile using all the learning curves obtained during hyperparameter tuning. The median performance is depicted as a dark line. For each subplot, the numbers in the square brackets represent the number of neurons per layer of the policy.

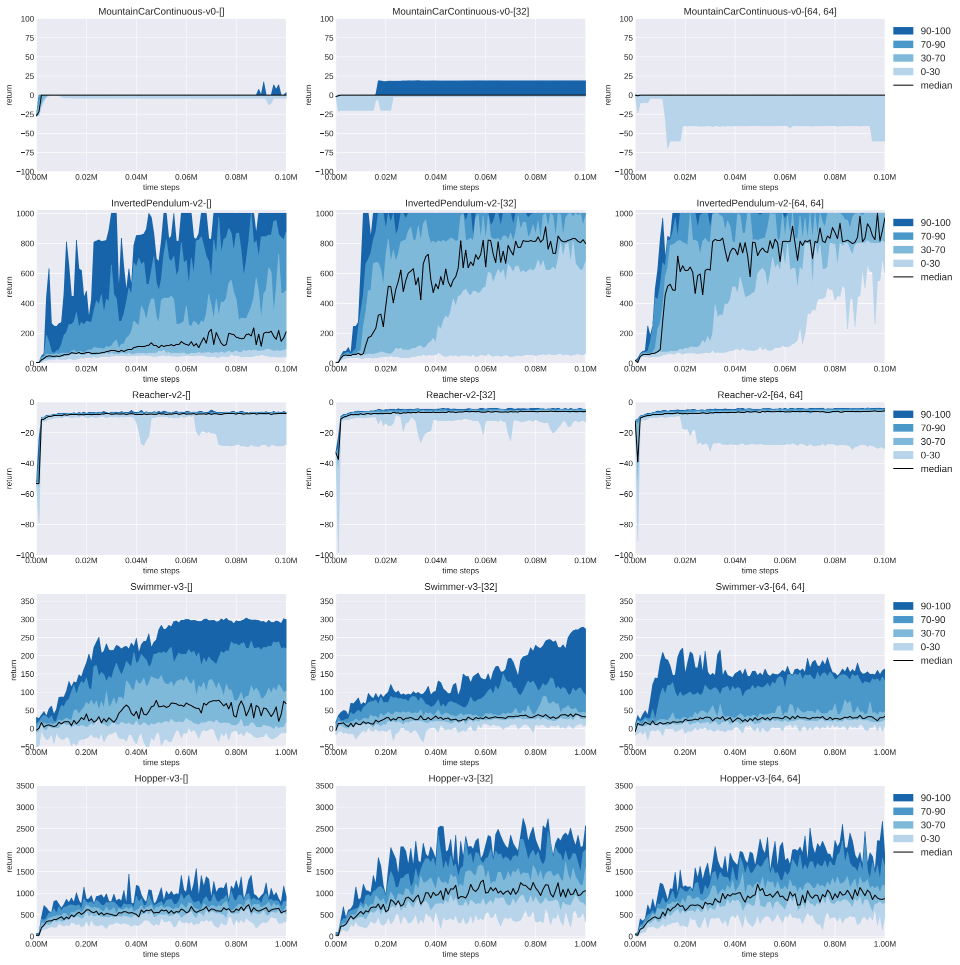

Figure 13: **Sensitivity of DDPG** to the choice of the hyperparameter. Performance is shown by percentile using all the learning curves obtained during hyperparameter tuning. The median performance is depicted as a dark line. For each subplot, the numbers in the square brackets represent the number of neurons per layer of the policy.

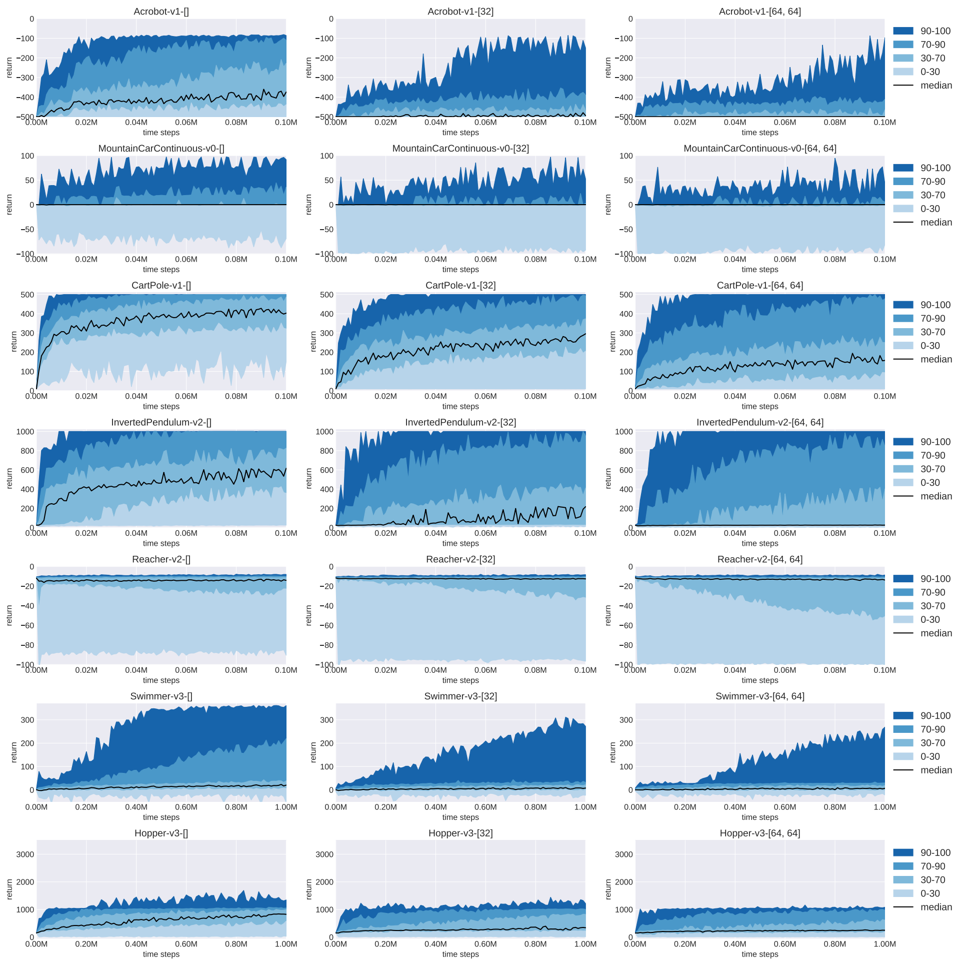

Figure 14: **Sensitivity of ARS** to the choice of the hyperparameter. Performance is shown by percentile using all the learning curves obtained during hyperparameter tuning. The median performance is depicted as a dark line. For each subplot, the numbers in the square brackets represent the number of neurons per layer of the policy.

Table 3: Table of best hyperparameters for PSSVFs

| **Learning rate policy** | Policy: | [] | | [32] | | [64,64] | |
|---|---|---|---|---|---|---|---|
| | Metric: | avg | last | avg | last | avg | last |
| Acrobot-v1 | | 1e-2 | 1e-3 | 1e-4 | 1e-4 | 1e-4 | 1e-4 |
| MountainCarContinuous-v0 | | 1e-2 | 1e-3 | 1e-4 | 1e-4 | 1e-4 | 1e-4 |
| CartPole-v1 | | 1e-3 | 1e-3 | 1e-3 | 1e-3 | 1e-4 | 1e-4 |
| Swimmer-v3 | | 1e-3 | 1e-3 | 1e-3 | 1e-3 | 1e-2 | 1e-4 |
| InvertedPendulum-v2 | | 1e-3 | 1e-3 | 1e-3 | 1e-3 | 1e-4 | 1e-4 |
| Reacher-v2 | | 1e-4 | 1e-4 | 1e-4 | 1e-4 | 1e-4 | 1e-4 |
| Hopper-v3 | | 1e-4 | 1e-4 | 1e-4 | 1e-3 | 1e-4 | 1e-4 |
| **Learning rate critic** | | | | | | | |
| Acrobot-v1 | | 1e-2 | 1e-3 | 1e-2 | 1e-2 | 1e-2 | 1e-2 |
| MountainCarContinuous-v0 | | 1e-3 | 1e-2 | 1e-3 | 1e-2 | 1e-2 | 1e-2 |
| CartPole-v1 | | 1e-2 | 1e-2 | 1e-3 | 1e-3 | 1e-2 | 1e-2 |
| Swimmer-v3 | | 1e-3 | 1e-3 | 1e-2 | 1e-2 | 1e-3 | 1e-2 |
| InvertedPendulum-v2 | | 1e-2 | 1e-2 | 1e-3 | 1e-2 | 1e-3 | 1e-3 |
| Reacher-v2 | | 1e-3 | 1e-3 | 1e-3 | 1e-3 | 1e-4 | 1e-4 |
| Hopper-v3 | | 1e-3 | 1e-3 | 1e-2 | 1e-2 | 1e-2 | 1e-2 |
| **Noise for exploration** | | | | | | | |
| Acrobot-v1 | | 1.0 | 1.0 | 1e-1 | 1e-1 | 1e-1 | 1e-1 |
| MountainCarContinuous-v0 | | 1.0 | 1.0 | 1e-1 | 1e-1 | 1e-1 | 1e-1 |
| CartPole-v1 | | 1.0 | 1.0 | 1.0 | 1.0 | 1e-1 | 1e-1 |
| Swimmer-v3 | | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1e-1 |
| InvertedPendulum-v2 | | 1.0 | 1.0 | 1.0 | 1.0 | 1e-1 | 1e-1 |
| Reacher-v2 | | 1e-1 | 1e-1 | 1e-1 | 1e-1 | 1e-1 | 1e-1 |
| Hopper-v3 | | 1.0 | 1.0 | 1e-1 | 1.0 | 1e-1 | 1e-1 |

Table 4: Table of best hyperparameters for ARS

| **Learning rate policy** | Policy: | [] | | [32] | | [64,64] | |
|---|---|---|---|---|---|---|---|
| | Metric: | avg | last | avg | last | avg | last |
| Acrobot-v1 | | 1e-2 | 1e-3 | 1e-2 | 1e-2 | 1e-2 | 1e-2 |
| MountainCarContinuous-v0 | | 1e-2 | 1e-2 | 1e-2 | 1e-2 | 1e-2 | 1e-2 |
| CartPole-v1 | | 1e-2 | 1e-2 | 1e-2 | 1e-2 | 1e-2 | 1e-2 |
| Swimmer-v3 | | 1e-2 | 1e-2 | 1e-2 | 1e-2 | 1e-2 | 1e-2 |
| InvertedPendulum-v2 | | 1e-2 | 1e-2 | 1e-2 | 1e-2 | 1e-2 | 1e-2 |
| Reacher-v2 | | 1e-2 | 1e-2 | 1e-3 | 1e-2 | 1e-3 | 1e-3 |
| Hopper-v3 | | 1e-2 | 1e-2 | 1e-2 | 1e-2 | 1e-2 | 1e-2 |
| **Number of directions and elite directions** | | | | | | | |
| Acrobot-v1 | | (4,4) | (4,4) | (1,1) | (1,1) | (1,1) | (1,1) |
| MountainCarContinuous-v0 | | (1,1) | (1,1) | (1,1) | (16,4) | (1,1) | (1,1) |
| CartPole-v1 | | (4,4) | (4,4) | (1,1) | (1,1) | (4,1) | (4,1) |
| Swimmer-v3 | | (1,1) | (1,1) | (1,1) | (4,1) | (1,1) | (1,1) |
| InvertedPendulum-v2 | | (4,4) | (4,4) | (1,1) | (4,4) | (4,1) | (16,1) |
| Reacher-v2 | | (16,16) | (16,16) | (1,1) | (16,4) | (1,1) | (1,1) |
| Hopper-v3 | | (4,1) | (4,1) | (1,1) | (1,1) | (1,1) | (1,1) |
| **Noise for exploration** | | | | | | | |
| Acrobot-v1 | | 1e-2 | 1e-3 | 1e-1 | 1e-1 | 1e-1 | 1e-1 |
| MountainCarContinuous-v0 | | 1e-1 | 1e-1 | 1e-1 | 1e-1 | 1e-1 | 1e-1 |
| CartPole-v1 | | 1e-2 | 1e-2 | 1e-1 | 1e-1 | 1e-2 | 1e-2 |
| Swimmer-v3 | | 1e-1 | 1e-1 | 1e-2 | 1e-1 | 1e-1 | 1e-1 |
| InvertedPendulum-v2 | | 1e-2 | 1e-2 | 1e-2 | 1e-2 | 1e-2 | 1e-2 |
| Reacher-v2 | | 1e-2 | 1e-2 | 1e-2 | 1e-2 | 1e-2 | 1e-2 |
| Hopper-v3 | | 1e-1 | 1e-1 | 1e-1 | 1e-1 | 1e-1 | 1e-1 |

Table 5: Table of best hyperparameters for PSVFs

| **Learning rate policy** | Policy: | [] | | [32] | | [64,64] | |
|---|---|---|---|---|---|---|---|
| | Metric: | avg | last | avg | last | avg | last |
| Acrobot-v1 | | 1e-2 | 1e-2 | 1e-4 | 1e-4 | 1e-4 | 1e-2 |
| MountainCarContinuous-v0 | | 1e-2 | 1e-3 | 1e-2 | 1e-4 | 1e-3 | 1e-4 |
| CartPole-v1 | | 1e-2 | 1e-2 | 1e-2 | 1e-4 | 1e-3 | 1e-4 |
| Swimmer-v3 | | 1e-3 | 1e-3 | 1e-3 | 1e-3 | 1e-3 | 1e-3 |
| InvertedPendulum-v2 | | 1e-2 | 1e-3 | 1e-4 | 1e-4 | 1e-4 | 1e-4 |
| Reacher-v2 | | 1e-3 | 1e-2 | 1e-4 | 1e-4 | 1e-4 | 1e-4 |
| Hopper-v3 | | 1e-3 | 1e-3 | 1e-4 | 1e-4 | 1e-4 | 1e-3 |
| **Learning rate critic** | | | | | | | |
| Acrobot-v1 | | 1e-3 | 1e-4 | 1e-2 | 1e-2 | 1e-3 | 1e-2 |
| MountainCarContinuous-v0 | | 1e-4 | 1e-3 | 1e-2 | 1e-4 | 1e-3 | 1e-3 |
| CartPole-v1 | | 1e-2 | 1e-2 | 1e-2 | 1e-3 | 1e-2 | 1e-4 |
| Swimmer-v3 | | 1e-4 | 1e-4 | 1e-4 | 1e-4 | 1e-4 | 1e-4 |
| InvertedPendulum-v2 | | 1e-3 | 1e-2 | 1e-3 | 1e-4 | 1e-4 | 1e-3 |
| Reacher-v2 | | 1e-2 | 1e-2 | 1e-3 | 1e-3 | 1e-4 | 1e-4 |
| Hopper-v3 | | 1e-2 | 1e-2 | 1e-4 | 1e-4 | 1e-2 | 1e-4 |
| **Noise for exploration** | | | | | | | |
| Acrobot-v1 | | 1.0 | 1.0 | 1e-1 | 1e-1 | 1e-1 | 1e-1 |
| MountainCarContinuous-v0 | | 1.0 | 1e-1 | 1e-1 | 1.0 | 1e-1 | 1e-1 |
| CartPole-v1 | | 1.0 | 1.0 | 1.0 | 1e-1 | 1e-1 | 1e-1 |
| Swimmer-v3 | | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| InvertedPendulum-v2 | | 1.0 | 1.0 | 1e-1 | 1e-1 | 1e-1 | 1e-1 |
| Reacher-v2 | | 1.0 | 1.0 | 1.0 | 1.0 | 1e-1 | 1e-1 |
| Hopper-v3 | | 1.0 | 1.0 | 1e-1 | 1e-1 | 1e-1 | 1.0 |

Table 6: Table of best hyperparameters for PAVFs

| **Learning rate policy** | Policy: | [] | | [32] | | [64,64] | |
|---|---|---|---|---|---|---|---|
| | Metric: | avg | last | avg | last | avg | last |
| MountainCarContinuous-v0 | | 1e-2 | 1e-3 | 1e-3 | 1e-4 | 1e-4 | 1e-4 |
| Swimmer-v3 | | 1e-3 | 1e-3 | 1e-3 | 1e-3 | 1e-3 | 1e-3 |
| InvertedPendulum-v2 | | 1e-2 | 1e-3 | 1e-3 | 1e-4 | 1e-4 | 1e-4 |
| Reacher-v2 | | 1e-3 | 1e-3 | 1e-4 | 1e-4 | 1e-4 | 1e-4 |
| Hopper-v3 | | 1e-3 | 1e-4 | 1e-4 | 1e-4 | 1e-4 | 1e-3 |
| **Learning rate critic** | | | | | | | |
| MountainCarContinuous-v0 | | 1e-4 | 1e-4 | 1e-4 | 1e-3 | 1e-4 | 1e-3 |
| Swimmer-v3 | | 1e-4 | 1e-4 | 1e-4 | 1e-4 | 1e-4 | 1e-4 |
| InvertedPendulum-v2 | | 1e-3 | 1e-2 | 1e-2 | 1e-4 | 1e-2 | 1e-3 |
| Reacher-v2 | | 1e-3 | 1e-3 | 1e-3 | 1e-2 | 1e-3 | 1e-3 |
| Hopper-v3 | | 1e-4 | 1e-3 | 1e-3 | 1e-2 | 1e-4 | 1e-3 |
| **Noise for exploration** | | | | | | | |
| MountainCarContinuous-v0 | | 1.0 | 1e-1 | 1e-1 | 1e-1 | 1e-1 | 1e-1 |
| Swimmer-v3 | | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| InvertedPendulum-v2 | | 1.0 | 1.0 | 1e-1 | 1e-1 | 1e-1 | 1e-1 |
| Reacher-v2 | | 1e-1 | 1e-1 | 1e-1 | 1.0 | 1.0 | 1.0 |
| Hopper-v3 | | 1.0 | 1.0 | 1e-1 | 1e-1 | 1e-1 | 1.0 |

Table 7: Table of best hyperparameters for DDPG

| **Learning rate policy** | Policy: | [] | | [32] | | [64,64] | |
|---|---|---|---|---|---|---|---|
| | Metric: | avg | last | avg | last | avg | last |
| MountainCarContinuous-v0 | | 1e-2 | 1e-2 | 1e-2 | 1e-4 | 1e-3 | 1e-3 |
| Swimmer-v3 | | 1e-3 | 1e-3 | 1e-2 | 1e-2 | 1e-2 | 1e-2 |
| InvertedPendulum-v2 | | 1e-4 | 1e-4 | 1e-3 | 1e-3 | 1e-3 | 1e-4 |
| Reacher-v2 | | 1e-4 | 1e-3 | 1e-2 | 1e-2 | 1e-3 | 1e-3 |
| Hopper-v3 | | 1e-2 | 1e-2 | 1e-2 | 1e-4 | 1e-2 | 1e-2 |
| **Learning rate critic** | | | | | | | |
| MountainCarContinuous-v0 | | 1e-4 | 1e-4 | 1e-4 | 1e-3 | 1e-3 | 1e-3 |
| Swimmer-v3 | | 1e-3 | 1e-3 | 1e-3 | 1e-3 | 1e-2 | 1e-3 |
| InvertedPendulum-v2 | | 1e-3 | 1e-3 | 1e-3 | 1e-4 | 1e-3 | 1e-3 |
| Reacher-v2 | | 1e-3 | 1e-3 | 1e-3 | 1e-3 | 1e-3 | 1e-3 |
| Hopper-v3 | | 1e-3 | 1e-3 | 1e-4 | 1e-4 | 1e-4 | 1e-4 |
| **Noise for exploration** | | | | | | | |
| MountainCarContinuous-v0 | | 1e-2 | 1e-2 | 1e-2 | 1e-1 | 1e-1 | 1e-1 |
| Swimmer-v3 | | 1e-1 | 1e-1 | 1e-2 | 1e-2 | 1e-2 | 1e-1 |
| InvertedPendulum-v2 | | 1e-1 | 1e-1 | 1e-2 | 1e-2 | 1e-2 | 1e-2 |
| Reacher-v2 | | 1e-1 | 1e-2 | 1e-1 | 1e-1 | 1e-1 | 1e-1 |
| Hopper-v3 | | 1e-1 | 1e-1 | 1e-1 | 1e-2 | 1e-1 | 1e-2 |