Appendix

8. Proofs

8.1. Proof of Theorem 3.2

Proof. Unbiasedness follows immediately since the $(\varepsilon'_i)_{i=1}^N$ each have the same marginal distribution as the $(\varepsilon_i)_{i=1}^N$. For the MSE claim, we provide a proof for the case $N \leq d$; the general case is analogous. Let $(\varepsilon_i)_{i=1}^N$ be iid $N(0, I)$, and let $(\varepsilon'_i)_{i=1}^N$ be marginally $\mathcal{N}(0, I)$ and almost-surely orthogonal. Write

$$F^{(i)} = \frac{1}{2\sigma}\left(F(\theta + \sigma\varepsilon_i)\varepsilon_i - F(\theta - \sigma\varepsilon_i)\varepsilon_i\right),$$
$$F'^{(i)} = \frac{1}{2\sigma}\left(F(\theta + \sigma\varepsilon'_i)\varepsilon'_i - F(\theta - \sigma\varepsilon'_i)\varepsilon'_i\right),$$

for each $i = 1, \ldots, N$, and note that

$$\begin{aligned}
\mathrm{MSE}(\widehat{\nabla}_N^{\mathrm{AT,ort}} F_\sigma(\theta)) =& \mathbb{E}\left[\left\|\frac{1}{N}\sum_{i=1}^N F'^{(i)} - \nabla F_\sigma(\theta)\right\|_2^2\right] \\
=& \mathbb{E}\left[\left\|\frac{1}{N}\sum_{i=1}^N F'^{(i)}\right\|_2^2\right] - \|\nabla F_\sigma(\theta)\|_2^2 \\
=& \frac{1}{N^2}\left(\sum_{i=1}^N \mathbb{E}\left[\|F'^{(i)}\|_2^2\right] + \sum_{i\neq j}\mathbb{E}\left[\langle F'^{(i)}, F'^{(j)}\rangle\right]\right) - \|\nabla F_\sigma(\theta)\|_2^2 . \qquad (9)
\end{aligned}$$

By analogous reasoning, we have the following expression for the MSE of the iid estimator:

$$\mathrm{MSE}(\widehat{\nabla}_N^{\mathrm{AT}} F_\sigma(\theta)) = \frac{1}{N^2}\left(\sum_{i=1}^N \mathbb{E}\left[\|F^{(i)}\|_2^2\right] + \sum_{i\neq j}\mathbb{E}\left[\langle F^{(i)}, F^{(j)}\rangle\right]\right) - \|\nabla F_\sigma(\theta)\|_2^2. \qquad (10)$$

Since $F^{(i)}$ and $F'^{(i)}$ are equal in distribution, we have $\mathbb{E}\left[\|F^{(i)}\|_2^2\right] = \mathbb{E}\left[\|F'^{(i)}\|_2^2\right]$. Now note that $\langle F'^{(i)}, F'^{(j)}\rangle = 0$ almost surely for $i \neq j$, so $\mathbb{E}\left[\langle F'^{(i)}, F'^{(j)}\rangle\right] = 0$ in Equation (9). Note also that since $F^{(i)}$ and $F^{(j)}$ are independent for $i \neq j$, we have $\mathbb{E}\left[\langle F^{(i)}, F^{(j)}\rangle\right] = \langle \nabla F_\sigma(\theta), \nabla F_\sigma(\theta)\rangle = \|\nabla F_\sigma(\theta)\|_2^2 \geq 0$ in Equation (10). Therefore, the stated result follows. □

9. Implementation details

In this section, we give further information on the construction of exploration directions using Hadamard-Rademacher random matrices and quasi-Monte Carlo strategies, as well as precise details of the Toeplitz parametrisations used for policy networks in the experiments.

9.1. Exploration directions with Hadamard-Rademacher random matrices and quasi-Monte Carlo

Here, we provide precise algorithmic details as to how Hadamard-Rademacher random matrices and quasi-Monte Carlo sequences can be used to construct exploration directions, complementing the discussion in Sections 3.2 and 3.3.

Algorithm 1 sets out the computation required to generate exploration directions from Hadamard-Rademacher random matrices.

Algorithm 2 describes the computation required to generate exploration directions from a quasi-Monte Carlo sequence. The first step of the algorithm is to draw a set of samples which resemble draws from $\mathrm{Unif}([0,1]^d)$. Rather than sampling i.i.d. from this distribution, instead a call to a standard QMC sampler is used – these samples are designed to "fill the space" more efficiently that i.i.d. samples from $\mathrm{Unif}([0,1]^d)$ typically would. There are many QMC sampling algorithms that may be used; in our experiments, we use generalized Halton sequences (additional details are given in Section 11.1), but see (Dick & Pillichshammer, 2010) for a extensive survey of commonly-used QMC sampling methods. The second step is to transform these samples from occupying the unit hypercube $[0,1]^d$, to approximating a collection of multivariate Gaussian samples in $\mathbb{R}^d$; this is acheived by applying the Gaussian CDF coordinate-wise to the hypercube samples.

Algorithm 1 Hadamard-Rademacher exploration directions

Sample the matrices $\mathbf{D}_1, \ldots, \mathbf{D}_k$ by drawing i.i.d. Rademacher random variables for each diagonal entry.
Set $\mathbf{G} = \mathbf{I}$, the identity matrix
for j=1,. . . ,k **do**
  Set $\mathbf{G} \leftarrow \mathbf{DG}$
  Compute $\mathbf{G} \leftarrow \mathbf{HG}$ via the Fast Walsh-Hadamard Transform.
end for
Compute $\mathbf{G} \leftarrow d^{-\frac{k-1}{2}}\mathbf{G}$
Use the resulting rows of $\mathbf{G}$ as exploration direction, in place of Gaussian vectors $(\varepsilon_i)_{i=1}^N$ in the estimators described in Section 2.

Algorithm 2 Quasi-Monte Carlo exploration directions

Generate a sequence of quasi-Monte Carlo samples $(\mathbf{x}_i)_{i=1}^N \subset [0,1]^d$ via a standard QMC algorithm.
For each sample $\mathbf{x}_i$ ($i = 1, \ldots, N$), compute the *transformed sample* ε_i given by applying the standard Normal CDF to each coordinate of $\mathbf{x}_i$.
Use the $(\varepsilon_i)_{i=1}^N$ as exploration directions in the estimators described in Section 2.

9.2. Toeplitz network structures

The Toeplitz structure is enforced by encoding this part of the network with vectors of size: $m + n - 1$, where m stands for the number of rows and n for the number of columns. The entire network is vectorized and in the inference phase de-vectorized into a sequence of structured matrices. Note that we never explicitly backpropagate through the network, we only run forward passes. To update parameters of the network, we always use vectorized representations.

10. Related work

In this section, we briefly mention other work related to our approach. Whilst our methods are focused on variance reduction for *isotropic* Gaussian smoothings of an objective function F, there has been much work on adapting the smoothing online, to reflect the local properties of F at the current set of parameters; a principal example of such an approach is CMA-ES (Hansen et al., 2003). These adaptive approaches have been shown to yield considerable improvements in performance versus isotropic baselines in certain circumstances. Here, we observe that these adaptive approaches are complementary to our variance reduction techniques, and in principle these variance reduction techniques could be extended to methods that invoke covariance adaptation (by, for example, enforcing that exploration directions are orthogonal under a whitening transform with respect to the current covariance matrix). We leave it as an open question for future as to how such exploration methods can be implemented in a computational efficient manner across a distributed system. These ES approaches differ from other recent methods for continuous control, such as DDPG (Lillicrap et al., 2015), in that they do not take advantage of any Markov structure in the environment. We remark, however, that exploration in DDPG is achieved by injecting Gaussian noise into the actions of an agent, and there may be interesting further work in understanding whether the variance reduction techniques studied here are applicable in these contexts too.

11. Experimental Details for Section 6.1

11.1. Further Experimental Details

We provide full details of the experimental setup for the optimsiation problems solved in Section 6.1. Gradient estimates from each ES strategy were supplied to MATLAB's built-in `fminunc` gradient-based optimisation function, using the `quasi-newton` option. The final objective value reported for a given optimisation problem and exploration method was given by the output of the `fminunc` method, and the number of function evaluations reported for a given optimsiation problem and exploration method was the total number of function evaluations recorded during the call to `fminunc`. The number of exploration directions was taken to be equal to the dimensionality of the optimisation problem in all circumstances, unless otherwise stated.

For the QMC method described in the main paper, we MATLAB's built-in `haltonset` function to generate a generalized Halton sequence in the unit hypercube, apply a reverse-radix scrambling, and then apply coordinate-wise inverse Gaussian

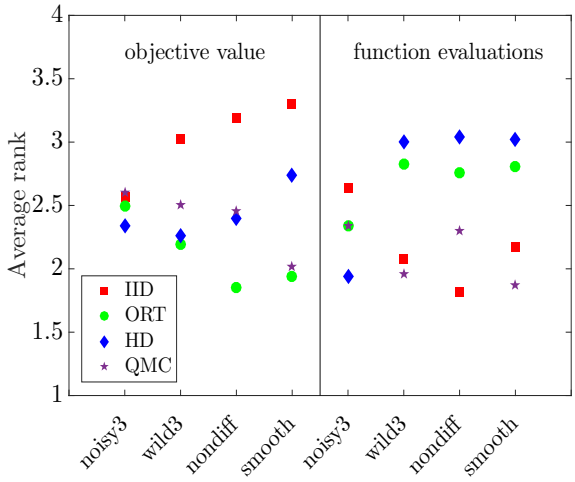


Figure 4. Average rankings across DFO tasks for the antithetic estimator (6) with a variety of exploration distributions. The standard deviation of the exploration distribution was 10^{-6} for all methods. Rankings based on final objective value are given on the left-hand side of the figure, whilst rankings based on function evaluations are given on the right-hand side. Lower ranks are better.

cumulative density functions to obtain multivariate Gaussian samples. The `leap` and `skip` parameters of `haltonset` were set to 700 and 1000 respectively. The deterministic reverse-radix scrambling is applied to the quasi-Monte Carlo stream to the stream of points via MATLAB's built-in `scramble` function.

11.2. Ranking Comparison

Here, we give the comparison of the methods described in Section 6.1 based on rankings, as described in the main paper. The results are broadly in line with the comparison based on normalized scores.

11.3. Further Experiment Results: Varying Exploration Noise

In this section, we study the effect of varying the exploration noise parameter σ on the findings of Section 6.1. In Section 6.1, σ was set to 10^{-6} in all experiments. Here, we give corresponding results with σ set to 10^{-7} (see Figures 5 and 6) and 10^{-5} (see Figures 7 and 8). Overall, the relative behaviour of the exploration methods remains similar as the exploration noise is varied.

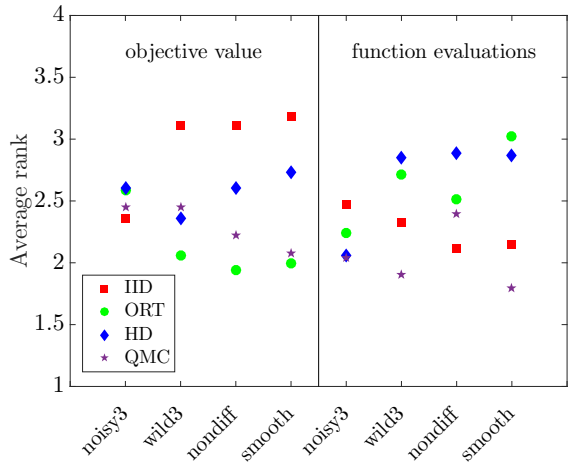


Figure 5. $\sigma = 10^{-7}$, average ranks.

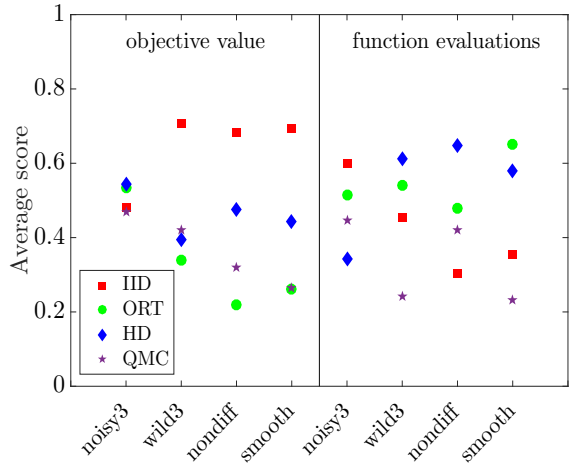


Figure 6. $\sigma = 10^{-7}$, average scores.

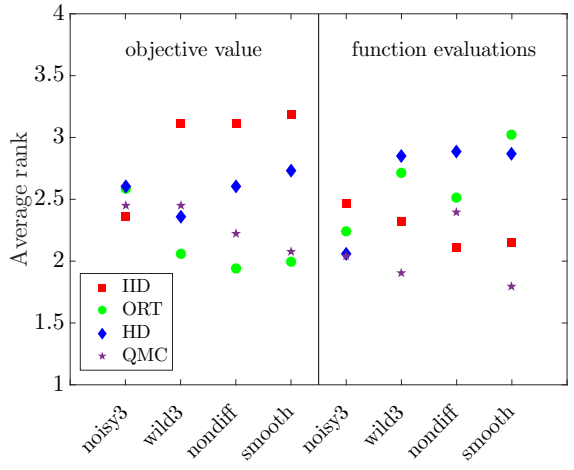


Figure 7. $\sigma = 10^{-5}$, average ranks.

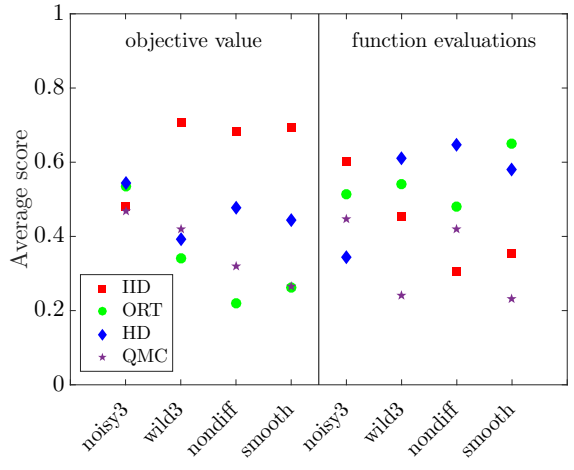


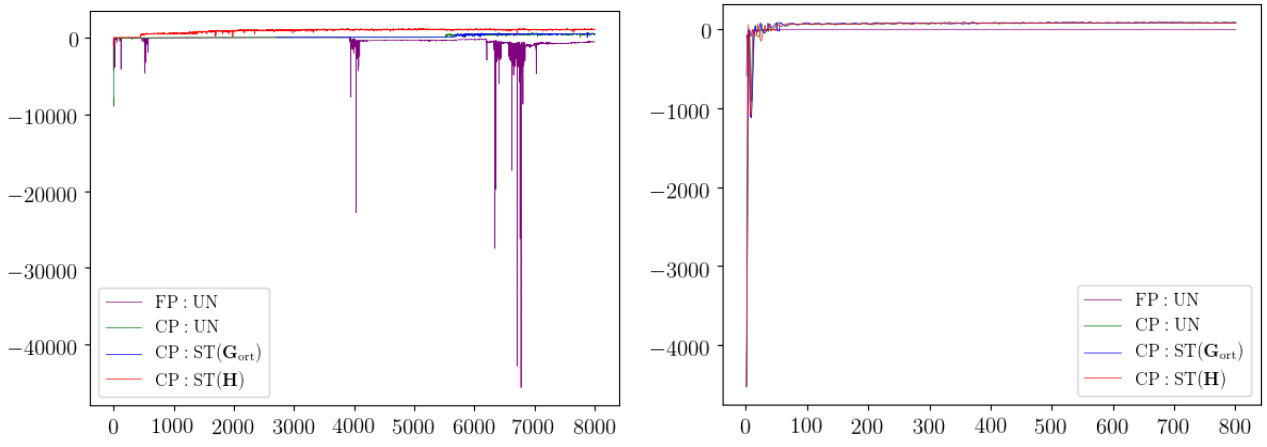
Figure 8. $\sigma = 10^{-5}$, average scores.

12. Additional OpenAI Gym Learning Curves

In this section, we provide learning curves for all environments and algorithms described in Section 6.2. We also run experiments on more (20 random seeds), computing the mean reward and standard deviation as well as and add comparison to a standard finite difference method. For the standard finite difference method (FD) all runs are the same since the environment and exploration is completely deterministic, thus standard deviation is 0. The termination of the training procedure is dictated by how much the total reward changed over a specific time interval (if the changes are sufficiently small the optimization procedure terminates).

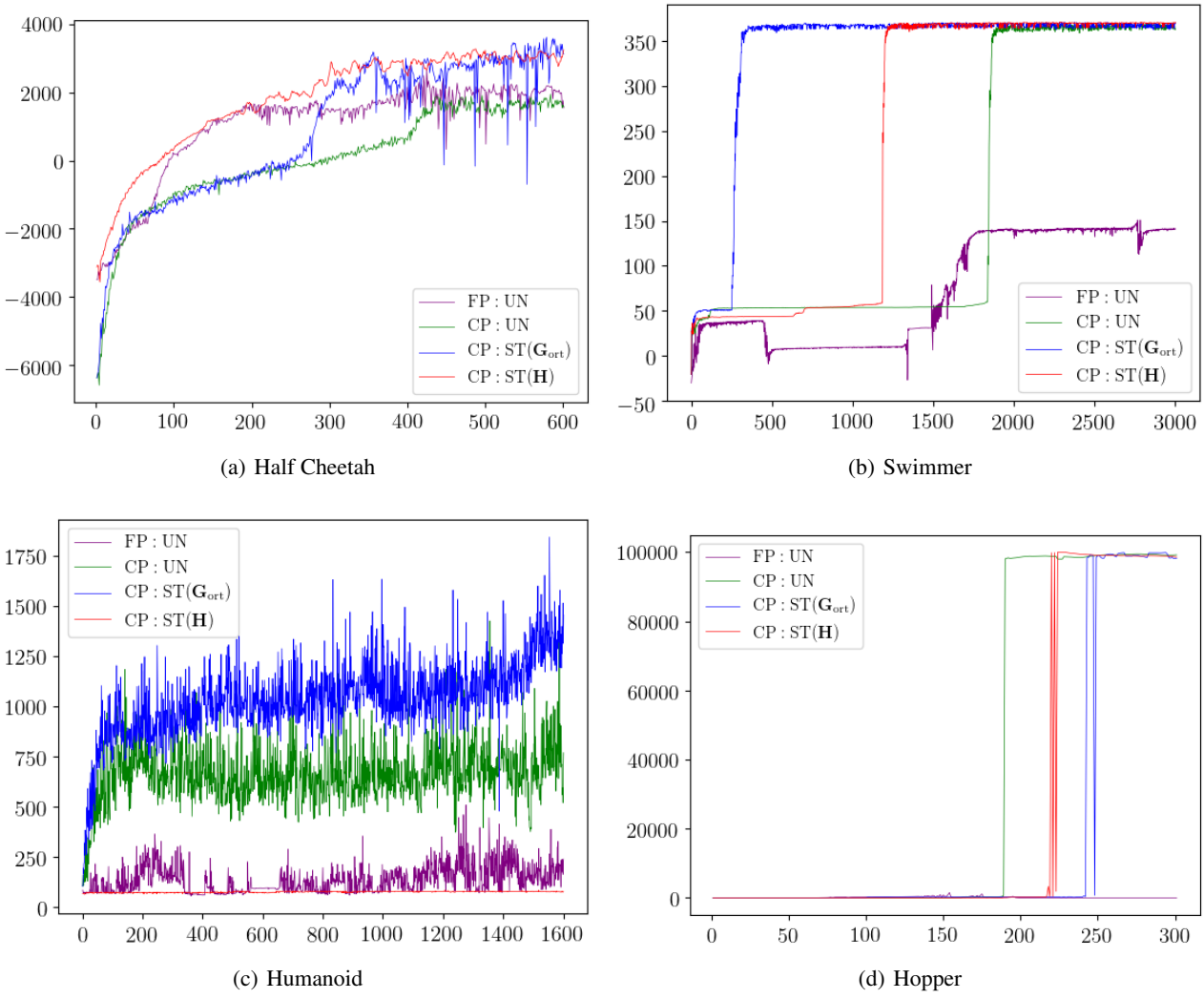
	CP:ST($\mathbf{G}_{\text{ort}}$)	CP:ST($\mathbf{H}$)	CP:UN	FP:UN	CP:FD	FP:FD
AN	514.7/0.2	**1150.23**/2.6	563.78/0.3	-13.11/1.2	505.00	-10.23
SW	370.0/0.1	**371.0**/0.1	367.1/1.2	151.1/5.2	313.22	174.48
HC	**3623.56**/2.3	3277.11/3.3	1942.55/1.4	2656.39/2.7	1672.65	3011.22
HO	99888.11/4.2	**99893.21**/2.7	99460.36/1.8	1536.00/2.2	94032.65	10032.69
HU	**1849.3**/2.2	88.13/1.9	1430.01/2.8	511.93/5.2	1325.79	624.78
WA	**10001.05**/2.8	9980.63/3.2	9754.48/3.2	459.33/4.2	9003.11	531.20
PU	-49.68/0.2	-43.33/0.3	**-35.25**/0.1	-47.37/0.3	-49.57	-51.42
RE	**-4.11**/0.24	-12.31/0.44	-74.31/0.67	-149.63/1.2	-85.62	-181.57
ST	-113.62/1.3	-88.77/0.3	**-49.43**/2.3	-66.61/0.2	-51.06	-90.94
TH	-361.55/5.2	-241.55/1.1	-267.32/3.4	**-190.51**/1.8	-415.49	-382.12
CMC	91.89/0.2	**94.11**/0.1	90.03/0.4	-0.11/0.1	90.79	-0.11
PE	-128.55/0.1	**-125.38**/0.82	-3290.22/5.4	-5088.34/4.3	-3582.62	-6627.83

Table 4. Mean total rewards obtained from 20 random seeds on different robotics OpenAI Gym tasks and corresponding standard deviations (mean/std) for different neural network architectures and exploration strategies. Additional columns: CP:FD corresponds to the structured neural network and standard finite difference method for gradient approximation; and FP:FD corresponds to the unstructured neural network with standard finite difference method for gradient approximation. For the FD method all runs are the same (see: comment in the main text) thus standard deviation is 0 and we do not report it. Highest rewards are shown in bold (AN:Ant, SW:Swimmer, HC:HalfCheetah, HO:Hopper, HU:Humanoid, WA:Walker2d, PU:Pusher, RE:Reacher, ST:Striker, TH:Thrower, CMC:Continuous Mountain Car, PE:Pendulum).



(a) Ant (b) Continuous Mountain Car

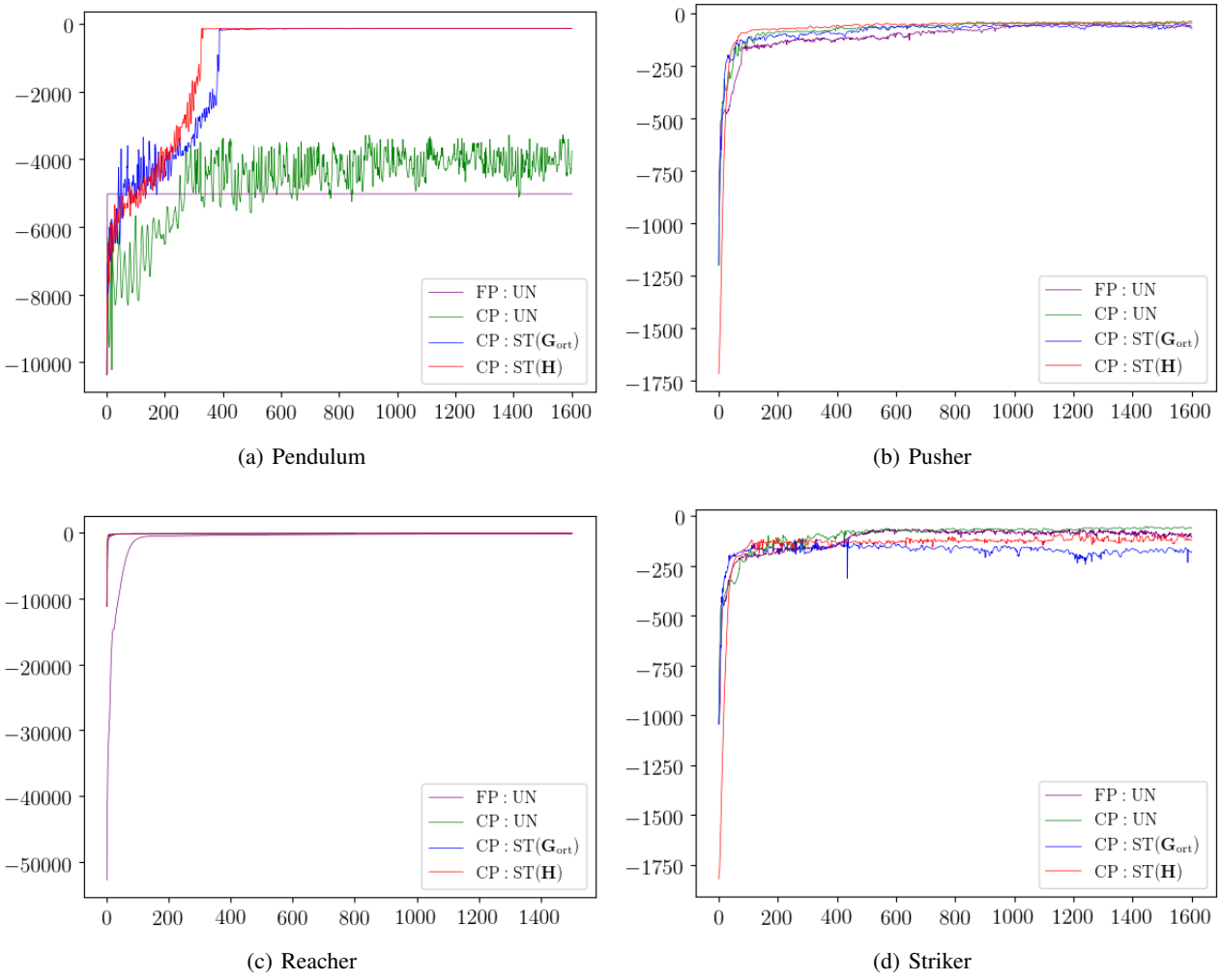
Figure 9. Learning curves for different OpenAI Gym envs.



(a) Half Cheetah (b) Swimmer

(c) Humanoid (d) Hopper

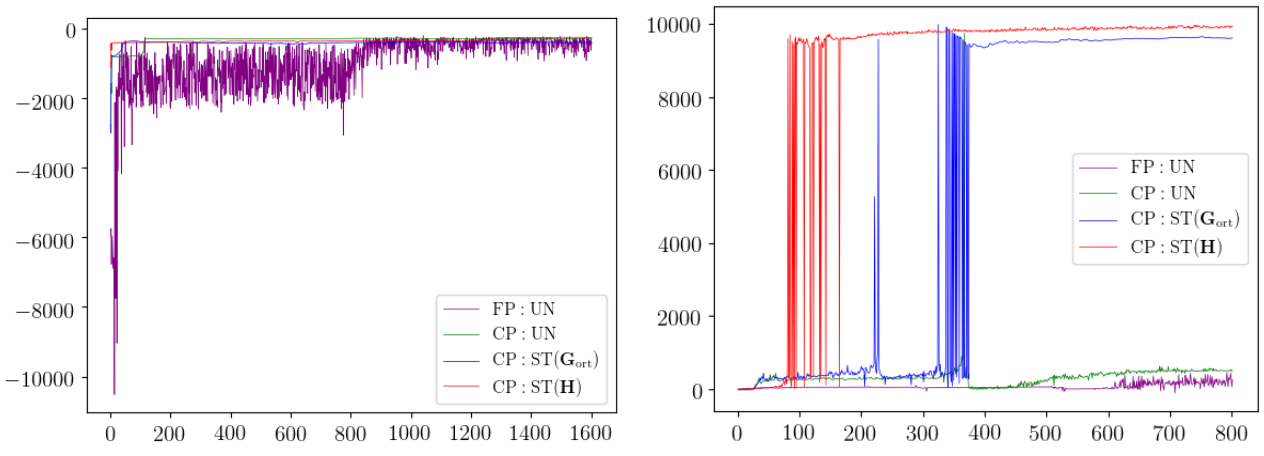
Figure 10. Learning curves for different OpenAI Gym envs.



(a) Pendulum (b) Pusher

(c) Reacher (d) Striker

Figure 11. Learning curves for different OpenAI Gym envs.



(a) Thrower (b) Walker2d

Figure 12. Learning curves for different OpenAI Gym envs.