A HaDES Algorithm

Algorithm 1 HaDES

Require: dataset size n, environment Env
Require: number of meta steps T, pop. size P, learning rate α, std σ
Initialize $\mathcal{D}_\phi = \{(s,a)_1, \ldots, (s,a)_n\}$ ▷ e.g. randomly or sampled
for meta_step $= 0, \ldots, T$ **do**
  Initialize $\xi \sim \mathcal{N}(0, \sigma)$
  for $i = 0, \ldots, P$ **do**
    if i is even **then**
      Perturb $\mathcal{D}_\phi$ with noise $\xi_i = \xi$ to get $\mathcal{D}_i$ ▷ antithetic noise
    else
      Perturb $\mathcal{D}_\phi$ with noise $\xi_i = -\xi$ to get $\mathcal{D}_i$
      Update $\xi \sim \mathcal{N}(0, \sigma)$
    end if
    Initialize policy π_θ
    Train policy π_θ on $\mathcal{D}_i$ using BC
    Unroll π_θ and compute expected return $J_i = J(\pi_\theta, \text{Env}|\mathcal{D}_i)$
  end for
  Approximate $\nabla_\phi J \approx \frac{1}{P\sigma} \sum_i J_i \xi_i$
  Update $\mathcal{D}_\phi = \mathcal{D}_\phi + \alpha \nabla_\phi J$
end for
Train policy π_θ on final $\mathcal{D}_\phi$
return $(\pi_\theta, \mathcal{D}_\phi)$

B Implementation details

B.1 Hyperparameters

Parameter	Value
LR	0.005
NUM_ENVS	4
NUM_STEPS	1024
UPDATE_EPOCHS	400
MAX_GRAD_NORM	0.5
ACTIVATION	tanh
WIDTH	512
ANNEAL_LR	False
GREEDY_ACT	False
CONST_NORMALIZE_OBS	False
NORMALIZE_OBS	True
NORMALIZE_REWARD	True
popsize	2048
dataset_size	64
rollouts_per_candidate	1
n_generations	2000
sigma_init	0.03
sigma_decay	1.0
lrate_init	0.05
Evo. strategy	OpenES

Table 2: Hyperparameters for HaDES in Brax. Top: inner loop parameters. Bottom: Outer loop parameters.

Parameter	Value
NET	mlp
LR	0.03
NUM_ENVS	8
NUM_STEPS	1024
UPDATE_EPOCHS	64
MAX_GRAD_NORM	0.5
ACTIVATION	relu
WIDTH	512 or 256
ANNEAL_LR	True
GREEDY_ACT	False
CONST_NORMALIZE_OBS	True
NORMALIZE_OBS	False
NORMALIZE_REWARD	False
popsize	2048
dataset_size	16
rollouts_per_candidate	2
n_generations	5000
sigma_init	0.5
sigma_limit	0.01
sigma_decay	1.0
lrate_init	0.05
lrate_decay	1.0
Evo. strategy	SNES
temperature	20.0

Table 3: Hyperparameters for HaDES in MinAtar. Top: inner loop parameters. Bottom: Outer loop parameters.

B.2 Experimental Details

For all RL tasks we use Brax (Freeman et al., 2021), a suite of continuous control environments, and MinAtar (Young & Tian, 2019), a set of Atari-like environments.

For dataset distillation, we report results on two image classification tasks: MNIST (LeCun, 1998), which is composed of handwritten digits, and FashionMNIST (Xiao et al., 2017), which features different clothing items.

For the evolutionary algorithm, we use OpenES Salimans et al. (2017) for Brax and image classification, and use SNES Wierstra et al. (2014) for MinAtar. In the inner loop, we minimize either the cross-entropy loss (discrete cases) or the negative log likelihood of the synthetic actions (continuous action cases). All of our runs use 8 Nvidia V100 GPUs and take between 1 and 17 seconds per outer loop generation. Detailed generation times are reported in Table 4. These times include outer loop operations (all methods), inner loop policy training (HaDES only), and inner loop policy evaluation (all methods). HaDES-R is slightly slower than HaDES-F since it trains two policies instead of just one. Because we train policies from scratch every generation, the times reported are strict upper bounds to how long it takes to train a policy on the final distilled datasets.

In image classification and MinAtar, we assign labels (i.e. classes or discrete actions) uniformly, whereas in Brax we evolve the dataset labels alongside the observations since the environments feature continuous actions.

We implement our algorithm in JAX (Bradbury et al., 2018) using the PureJaxRL (Lu et al., 2022), gymnax (Lange, 2022) and evosax (Lange, 2023) libraries to enable parallel training on hardware accelerators. We also use virtual batch normalization (Salimans et al., 2016) to stabilize training, which was previously found to be crucial in stabilizing ES (Salimans et al., 2017).

Environment Name	ES Neuroevolution	HaDES-F	HaDES-R
Hopper	4.0	5.6	6.3
Walker2d	3.4	7.9	8.5
Reacher	2.7	6.7	7.2
Inverted Double Pendulum	2.6	4.4	4.6
Ant	6.6	11.1	12.1
Halfcheetah	10.7	14.8	16.9
Humanoid	7.8	13.9	15.1
HumanoidStandup	8.6	14.8	16.4
SpaceInvaders-MinAtar	1.6	1.5	1.6
Breakout-MinAtar	1.3	1.6	1.8
Asterix-MinAtar	1.9	2.1	2.4
Freeway-MinAtar	2.3	2.2	2.9

Table 4: Runtime of the different neuroevolution methods in seconds/generation. Times averaged over 3 seeds rounded to the nearest tenth of a second. Standard deviation omitted, but the difference between the fastest and slowest runs for any setting is usually smaller than 0.2 seconds.

C Additional Results

C.1 Impact of distillation budget on performance

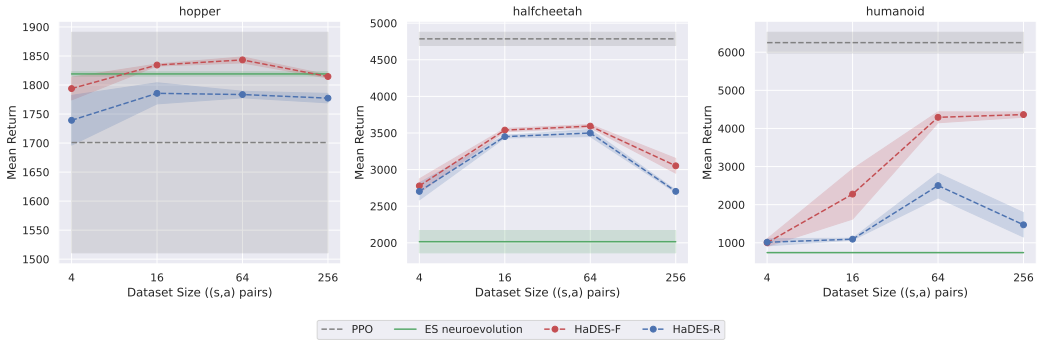

Figure 6: Final return of HaDES policies as a function of distillation budget (i.e. dataset size). ES neuroevolution and PPO returns also plotted for reference.

In Figure 6, we investigate the impact of the distillation budget on the final performance of policies trained with HaDES on three different environments. What we observe is that dataset sizes that are too small degrade performance, likely because they cannot contain all the information required to train an expert policy. This is particularly noticeable in Humanoid, where for a dataset of 4 state-action pairs, the score drops as low as 1013. However, a score of 1000 corresponds to a humanoid policy that keeps its balance and stays immobile, with lower scores indicating that the policy falls and causes an early termination. This is an indication that for distillation budgets that are too low to capture expert behaviour, HaDES does not fail to learn, and will still optimize return within the constraints of the budget.

On the opposite end of the spectrum, we also observe return dropping for large dataset sizes ($|\mathcal{D}| = 256$), despite the increased expressivity. This is possibly due to ES (and therefore HaDES) scaling poorly to a large number of parameters. This problem can be alleviated by relying on better ES methods, or by using a factorized approach to distillation.

C.2 Dataset generalization across architectures and hyperparameters

Here we plot generalization plots for additional environments. As expected, HaDES-R generalizes better than HaDES-F both to new hyperparameters and to new architectures.

Figure 7: Dataset transfer to hopper architecture and training parameters.

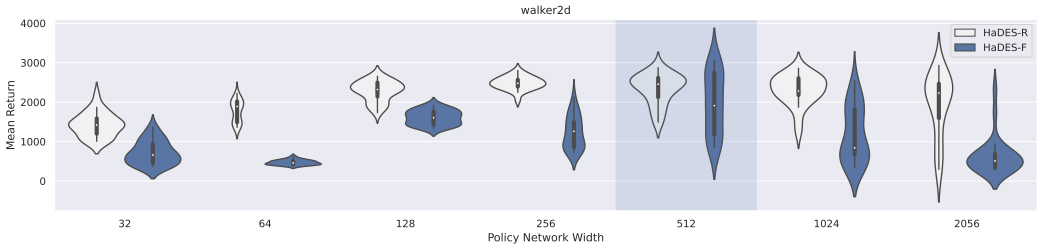

Figure 8: Dataset transfer to walker2d architecture and training parameters.

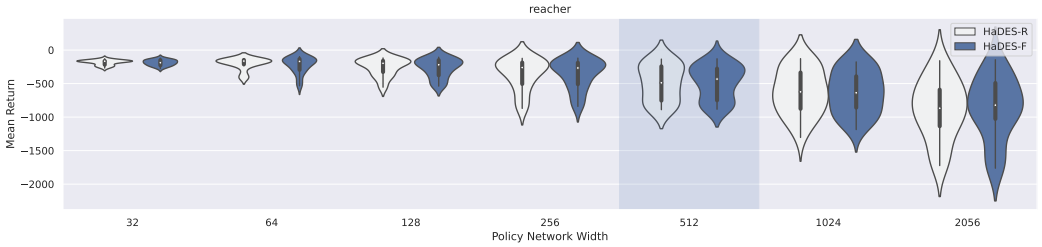

Figure 9: Dataset transfer to reacher architecture and training parameters.

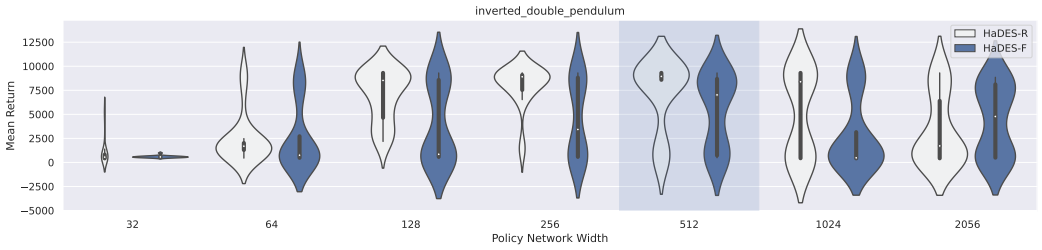

Figure 10: Dataset transfer to inverted_double_pendulum architecture and training parameters.

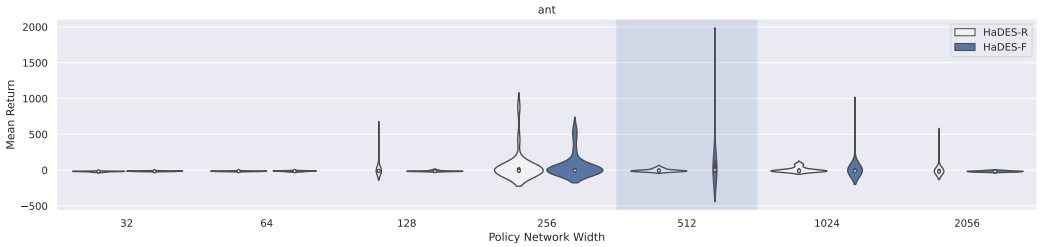

Figure 11: Dataset transfer to ant architecture and training parameters.

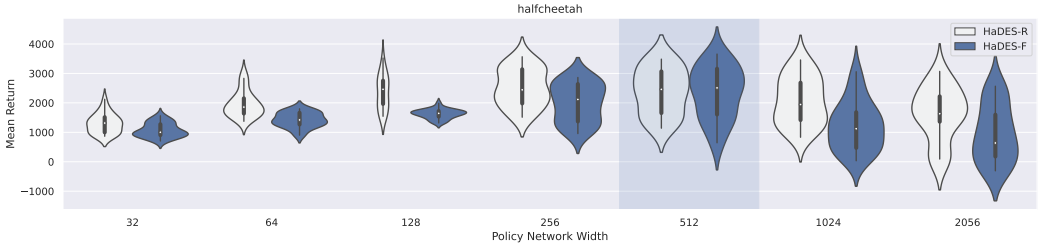

Figure 12: Dataset transfer to halfcheetah architecture and training parameters.

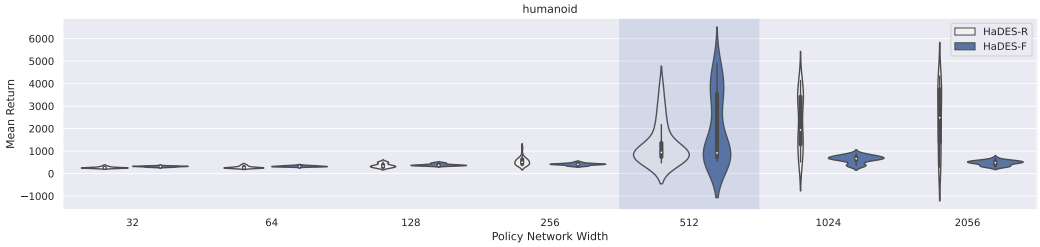

Figure 13: Dataset transfer to humanoid architecture and training parameters.

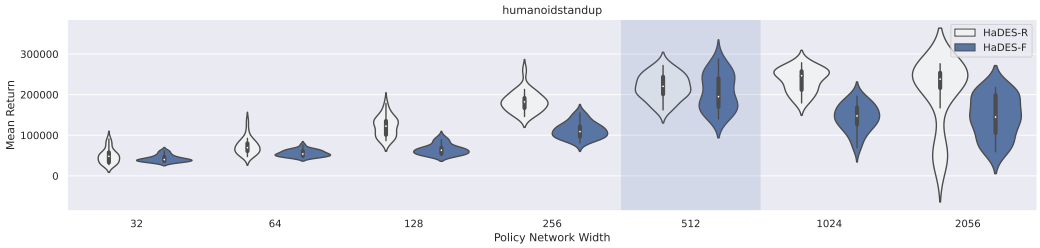

Figure 14: Dataset transfer to humanoidstandup architecture and training parameters.