A Derivation

The AMT and ES update rules are given as

$$\sigma_{(t)} \leftarrow \sigma_{(t-1)} + \frac{\alpha_{es}}{n\sigma_{(t-1)}} SmoothL1(R_{max,(t-1)}, R_{avg,(t-1)})$$

$$\theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(t)}} \sum_{i=1}^{N} R_i \epsilon_i$$

Using the expression for $\sigma_{(t)}$ in the ES update yields the following

$$\theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n(\sigma_{(t-1)} + \frac{\alpha_{es}}{n\sigma_{(t-1)}} SmoothL1(R_{max,(t-1)}, R_{avg,(t-1)}))} \sum_{i=1}^{n} R_i \epsilon_i$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(t-1)}(1 + \frac{\alpha_{es}}{n\sigma^2_{(t-1)}} SmoothL1(R_{max,(t-1)}, R_{avg,(t-1)}))} \sum_{i=1}^{n} R_i \epsilon_i$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(t-1)}\Lambda_{(t-1)}} \sum_{i=1}^{n} R_i \epsilon_i$$

where $\Lambda_{(t-1)} = 1 + \frac{\alpha_{es}}{n\sigma^2_{(t-1)}} SmoothL1(R_{max,(t-1)}, R_{avg,(t-1)})$. Expanding $\sigma_{(t-1)}$ using the AMT update rule gives us

$$\theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\Lambda_{(t-1)}(\sigma_{(t-2)} + \frac{\alpha_{es}}{n\sigma_{(t-2)}} SmoothL1(R_{max,(t-2)}, R_{avg,(t-2)}))} \sum_{i=1}^{n} R_i \epsilon_i$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\Lambda_{(t-1)}\sigma_{(t-2)}(1 + \frac{\alpha_{es}}{n\sigma^2_{(t-2)}} SmoothL1(R_{max,(t-2)}, R_{avg,(t-2)}))} \sum_{i=1}^{n} R_i \epsilon_i$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\Lambda_{(t-1)}\sigma_{(t-2)}\Lambda_{(t-2)}} \sum_{i=1}^{n} R_i \epsilon_i$$

where $\Lambda_{(t-2)} = 1 + \frac{\alpha_{es}}{n\sigma^2_{(t-2)}} SmoothL1(R_{max,(t-2)}, R_{avg,(t-2)})$. Expanding this recursively gives us the following

$$\theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(1)}\Lambda_{(t-1)}\Lambda_{(t-2)}...\Lambda_{(1)}} \sum_{i=1}^{n} R_i \epsilon_i$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(1)} \prod_{t'=1}^{t-1} \Lambda_{(t')}} \sum_{i=1}^{n} R_i \epsilon_i$$

$$= \theta_{(t+1)} \leftarrow \theta_{(t)} + \frac{\alpha_{es}}{n\sigma_{(1)}\hat{\Lambda}} \sum_{i=1}^{n} R_i \epsilon_i$$

Hence, yielding the AMT update in the form of initial mutation rate $\sigma_{(1)}$.

B Policy Improvement

We will now show that AMT improves the policy of winners in the population. Let us consider two successive gradient intervals g indexed by (l) and $(l-1)$. Let $p_{(l)}$ and $p_{(l-1)}$ be the probabilities of convergence to the optimal policy $\pi^*_{\theta_{es}}(a_t|s_t)$ in the weight space at (l) and $(l-1)$ respectively.

We start by evaluating the mutation rates at (l) and $(l-1)$ which are given as $\sigma_{(l)} > \sigma_{(l-1)}$. We can now evaluate the probabilities of convergence to $\pi^*_{\theta_{es}}(a_t|s_t)$ as

$$p_{(l)} \geq p_{(l-1)}$$

Using this fact, we can evaluate the winners (indexed by q) in the sorted reward population F.

$$
\begin{aligned}
\sum_{q=1}^{w} p_{(l)}^{(q)} F_{(l)}^{(q)} &\geq \sum_{q=1}^{w} p_{(l-1)}^{(q)} F_{(l-1)}^{(q)} \\
= \mathbf{E}_{F_{(l)}^{(q)} \sim F_{(l)}}[F_{(l)}^{(q)}] &\geq \mathbf{E}_{F_{(l-1)}^{(q)} \sim F_{(l-1)}}[F_{(l-1)}^{(q)}] \\
= \mathbf{E}[W_{(l)}] &\geq \mathbf{E}[W_{(l-1)}]
\end{aligned}
$$

Here, $p_{(l)}^{(q)}$ is the probability of convergence of actor q (having observed reward $F_{(l)}^{(q)}$) to its optimal policy $\pi_{\theta_{es}}^{(q),*}(a_t|s_t)$ at interval (l). $W_{(l)}$ represents the set of winners at (l). The mathematical expression obtained represents that the set of winners $W_{(l)}$ formed at the next gradient interval (l) is at least as good as the previous set of winners $W_{(l-1)}$, i.e.- $\pi_{\theta_{es},(l)}^{(q)}(a_t|s_t) \geq \pi_{\theta_{es},(l-1)}^{(q)}(a_t|s_t)$. This guarantees policy improvement among winners of the population.

C Additional Results

C.1 Performance

We evaluate the performance and sample-efficiency of ESAC on 21 MuJoCo and DeepMind Control Suite [38]. Figure Figure 7 presents learning behavior of ESAC in comparison to SAC, TD3, PPO and ES on all 21 tasks. Training setup for all agents was kept same with different values of hyperparameters (presented in subsection D.2). ESAC demonstrates improved returns on 16 out of 21 tasks.

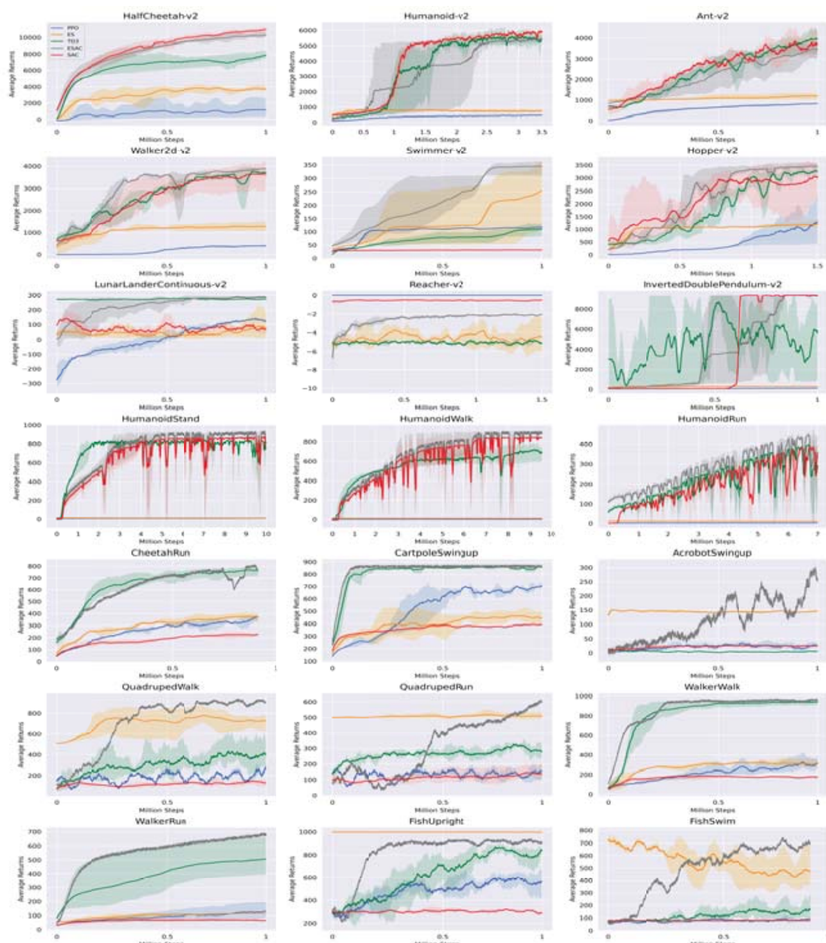

Figure 7: Average Returns on MuJoCo and DeepMind Control Suite tasks. ESAC's demonstrates state-of-the-art performance on out of 21 tasks.

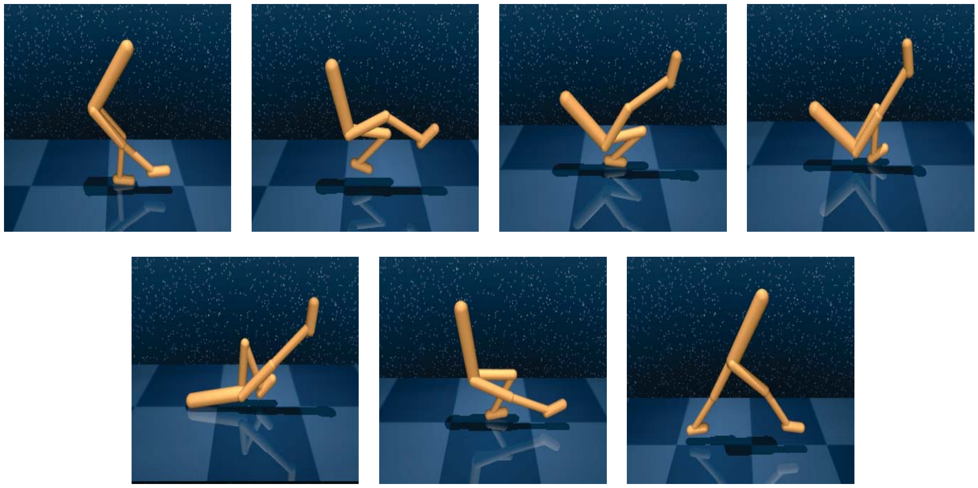

Figure 8: Robust behavior of ESAC observed on the WalkerWalk task. The ESAC policy prevents the walking robot from falling down when the robot loses its balance while walking. Although ESAC falls short of optimal performance as a result of sparse rewards, it exhibits robust policies on complex tasks due to successive evolutions.

C.2 Scalability

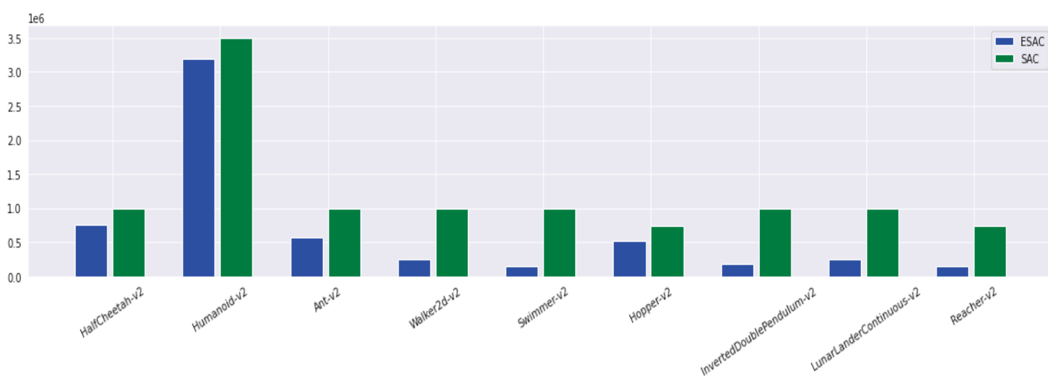

Figure 9: Complete results on the number of backprop updates for MuJoCo control tasks and LunarLanderContinuous environment from OpenAI's Gym suite. ESAC exponentially anneals gradient-based SAC updates and leverages winner selection and crossovers for computational efficiency and performance improvement.

D Hyperparameters

D.1 MuJoCo

Hyperparameter values for our experiments are adjusted on the basis of complexity and reward functions of tasks. In the case of MuJoCo control tasks, training-based hyperparameters are kept mostly the same with the number of SAC episodes varying as per the complexity of task. Tuning was carried out on HalfCheetah-v2, Ant-v2, Hopper-v2 and Walker2d-v2 tasks. Out of these, Ant-v2 presented high variance indicating the requirement of a lower learning rate. Number of SAC updates in SAC and ESAC implementations were kept 1 for a fair comparison with TD3. All tasks have a common discount factor $\gamma = 0.99$, SAC learning rate $\alpha = 3 \times 10^{-4}$, population size $n = 50$, mutation rate $\sigma = 5 \times 10^{-3}$ and winner fraction $e = 0.4$. ES learning rate α_{es} was kept fixed at 5×10^{-3} for all tasks except Ant-v2 having $\alpha_{es} = 1 \times 10^{-4}$.

The only variable hyperparameter in our experiments is number of SAC episodes executed by the SAC agent. Although the ESAC population in general is robust to its hyperparameters, the SAC agent is sensitive to the number of episodes. During the tuning process, the number of episodes were kept constant to a value of 10 for all the tasks. However, this led to inconsistent results on some of the environments when compared to SAC baseline. As a result, tuning was carried out around this value to obtain optimal results corresponding to each task. The SAC agent executed a total of 10 episodes for each gradient interval g for Ant-v2, Walker2d-v2, Hopper-v2 and Humanoid-v2 tasks; 5 episodes for HalfCheetah-v2, LunarLanderContinuous-v2, Reacher-v2 and InvertedPendulum-v2 tasks; and 1 episode for Swimmer-v2 task.

D.2 DeepMind Control Suite

The DeepMind control suite presents a range of tasks with sparse rewards and varying complexity for the same domain. Hyperparameter values for these tasks are different from that of MuJoCo control tasks. Tuning was carried on the Cheetah, Quadruped and Walker tasks in order to obtain sample-efficient convergence. In the case of SAC, granularity of hyperparameter search was refined in order to observe consistent behavior. However, different values of temperature parameter produced varying performance. As in the MuJoCo case, we kept the number of updates fixed to 1 in order to yield a fair comparison with TD3.

All tasks have a common discount factor $\gamma = 0.99$, SAC learning rate $\alpha = 3 \times 10^{-4}$, population size $n = 50$, mutation rate $\sigma = 1 \times 10^{-2}$, winner fraction $e = 0.4$ and ES learning rate 1×10^{-2}. As in the case of MuJoCo tasks, the SAC is found to be sensitive to the number of episodes during the gradient interval g. These were kept constant at 5 and then tuned around this value for optimal performance. Final values of the SAC episodes were 5 for CartpoleSwingup, WalkerWalk and WalkerRun tasks and 1 for the remaining tasks.