Parameter Space Noise for Exploration

Matthias Plappert[†‡], **Rein Houthooft**[†], **Prafulla Dhariwal**[†], **Szymon Sidor**[†],
Richard Y. Chen[†], **Xi Chen**[⋆], **Tamim Asfour**[‡], **Pieter Abbeel**[⋆], **and Marcin Andrychowicz**[†]
[†] OpenAI
[‡] Karlsruhe Institute of Technology (KIT)
[⋆] UC Berkeley

Abstract

Deep reinforcement learning (RL) methods generally engage in exploratory behavior through noise injection in the action space. An alternative is to add noise directly to the agent's parameters, which can lead to more consistent exploration and a richer set of behaviors. Methods such as evolutionary strategies use parameter perturbations, but discard all temporal structure in the process and require significantly more samples. Combining parameter noise with traditional RL methods allows to combine the best of both worlds. We demonstrate that both off- and on-policy methods benefit from this approach through experimental comparison of DQN, DDPG, and TRPO on high-dimensional discrete action environments as well as continuous control tasks. Our results show that RL with parameter noise learns more efficiently than traditional RL with action space noise and evolutionary strategies individually.

1 Introduction

Exploration remains a key challenge in contemporary deep reinforcement learning (RL). Its main purpose is to ensure that the agent's behavior does not converge prematurely to a local optimum. Enabling efficient and effective exploration is, however, not trivial since it is not directed by the reward function of the underlying Markov decision process (MDP). Although a plethora of methods have been proposed to tackle this challenge in high-dimensional and/or continuous-action MDPs, they often rely on complex additional structures such as counting tables [1], density modeling of the state space [2], learned dynamics models [3–5], or self-supervised curiosity [6].

An orthogonal way of increasing the exploratory nature of these algorithms is through the addition of temporally-correlated noise, for example as done in bootstrapped DQN [7]. Along the same lines, it was shown that the addition of parameter noise leads to better exploration by obtaining a policy that exhibits a larger variety of behaviors [8, 9]. We discuss these related approaches in greater detail in Section 5. Their main limitation, however, is that they are either only proposed and evaluated for the on-policy setting with relatively small and shallow function approximators [10] or disregard all temporal structure and gradient information [9, 11, 12].

This paper investigates how parameter space noise can be effectively combined with off-the-shelf deep RL algorithms such as DQN [13], DDPG [14], and TRPO [15] to improve their exploratory behavior. Experiments show that this form of exploration is applicable to both high-dimensional discrete environments and continuous control tasks, using on- and off-policy methods. Our results indicate that parameter noise outperforms traditional action space noise-based baselines, especially in tasks where the reward signal is extremely sparse. This demonstrates that a fertile middle ground exists between evolutionary methods that discard temporal structure, and methods that rely entirely on unstructured noise injection.

Correspondence to `matthias@openai.com`

2 Background

We consider the standard RL framework consisting of an agent interacting with an environment. To simplify the exposition we assume that the environment is fully observable. An environment is modeled as a Markov decision process (MDP) and is defined by a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a distribution over initial states $p(s_0)$, a reward function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, transition probabilities $p(s_{t+1}|s_t, a_t)$, a time horizon T, and a discount factor $\gamma \in [0, 1)$. We denote by π_θ a policy parametrized by θ, which can be either deterministic, $\pi : \mathcal{S} \mapsto \mathcal{A}$, or stochastic, $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$. The agent's goal is to maximize the expected discounted return $\eta(\pi_\theta) = \mathbb{E}_\tau[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)]$, where $\tau = (s_0, a_0, \ldots, s_T)$ denotes a trajectory with $s_0 \sim p(s_0)$, $a_t \sim \pi_\theta(a_t|s_t)$, and $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$. Experimental evaluation is based on the undiscounted return $\mathbb{E}_\tau[\sum_{t=0}^{T} r(s_t, a_t)]$.[1]

2.1 Off-policy Methods

Off-policy RL methods allow learning based on data captured by arbitrary policies. This paper considers two popular off-policy algorithms, namely Deep Q-Networks (DQN, [13]) and Deep Deterministic Policy Gradients (DDPG, [14]).

Deep Q-Networks (DQN) DQN uses a deep neural network as a function approximator to estimate the optimal Q-value function, which conforms to the Bellman optimality equation:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a').$$

The policy is implicitly defined by Q as $\pi(s_t) = \text{argmax}_{a' \in \mathcal{A}} Q(s_t, a')$. Typically, a stochastic ϵ-greedy or Boltzmann policy [16] is derived from the Q-value function to encourage exploration, which relies on sampling noise in the action space. The Q-network predicts a Q-value for each action and is updated using off-policy data from a replay buffer.

Deep Deterministic Policy Gradients (DDPG) DDPG is an actor-critic algorithm, applicable to continuous action spaces. Similar to DQN, the critic estimates the Q-value function using off-policy data and the recursive Bellman equation:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma Q\left(s_{t+1}, \pi_\theta(s_{t+1})\right),$$

where π_θ is the actor or policy. The actor is trained to maximize the critic's estimated Q-values by back-propagating through both networks. For exploration, DDPG uses a stochastic policy of the form $\widehat{\pi_\theta}(s_t) = \pi_\theta(s_t) + w$, where w is either $w \sim \mathcal{N}(0, \sigma^2 I)$ (uncorrelated) or $w \sim \text{OU}(0, \sigma^2)$ (correlated).[2] Again, exploration is realized through action space noise.

2.2 On-policy Methods

In contrast to off-policy algorithms, on-policy methods require updating function approximators according to the currently followed policy. In particular, we will consider Trust Region Policy Optimization (TRPO, [18]), an extension of traditional policy gradient methods [19] using the natural gradient direction [20, 21].

Trust Region Policy Optimization (TRPO) TRPO improves upon REINFORCE [19] by computing an ascent direction that ensures a small change in the policy distribution. More specifically, TRPO solves the following constrained optimization problem:

$$\begin{aligned} \text{maximize}_\theta \quad & E_{s \sim \rho_{\theta'}, a \sim \pi_{\theta'}} \left[\frac{\pi_\theta(a|s)}{\pi'_\theta(a|s)} A(s, a) \right] \\ \text{s.t.} \quad & E_{s \sim \rho_{\theta'}}[D_{\text{KL}}(\pi_{\theta'}(\cdot|s) \| \pi_\theta(\cdot|s))] \leq \delta_{\text{KL}} \end{aligned}$$

where $\rho_\theta = \rho_{\pi_\theta}$ is the discounted state-visitation frequencies induced by π_θ, $A(s, a)$ denotes the advantage function estimated by the empirical return minus the baseline, and δ_{KL} is a step size parameter which controls how much the policy is allowed to change per iteration.

[1] If $t = T$, we write $r(s_T, a_T)$ to denote the terminal reward, even though it has has no dependence on a_T, to simplify notation.

[2] $\text{OU}(\cdot, \cdot)$ denotes the Ornstein-Uhlenbeck process [17].

3 Parameter Space Noise for Exploration

This work considers policies that are realized as parameterized functions, which we denote as π_θ, with θ being the parameter vector. We represent policies as neural networks but our technique can be applied to arbitrary parametric models. To achieve structured exploration, we sample from a set of policies by applying additive Gaussian noise to the parameter vector of the current policy: $\widetilde{\theta} = \theta + \mathcal{N}(0, \sigma^2 I)$. Importantly, the perturbed policy is sampled at the beginning of each episode and kept fixed for the entire rollout. For convenience and readability, we denote this perturbed policy as $\widetilde{\pi} := \pi_{\widetilde{\theta}}$ and analogously define $\pi := \pi_\theta$.

State-dependent exploration As pointed out by [10], there is a crucial difference between action space noise and parameter space noise. Consider the continuous action space case. When using Gaussian action noise, actions are sampled according to some stochastic policy, generating $a_t = \pi(s_t) + \mathcal{N}(0, \sigma^2 I)$. Therefore, even for a *fixed* state s, we will almost certainly obtain a different action whenever that state is sampled again in the rollout, since action space noise is completely *independent* of the current state s_t (notice that this is equally true for correlated action space noise). In contrast, if the parameters of the policy are perturbed at the beginning of each episode, we get $a_t = \widetilde{\pi}(s_t)$. In this case, the same action will be taken every time the same state s_t is sampled in the rollout. This ensures consistency in actions, and directly introduces a dependence between the state and the exploratory action taken.

Perturbing deep neural networks It is not immediately obvious that deep neural networks, with potentially millions of parameters and complicated nonlinear interactions, can be perturbed in meaningful ways by applying spherical Gaussian noise. However, as recently shown by [9], a simple reparameterization of the network achieves exactly this. More concretely, we use layer normalization [22] between perturbed layers.[3] Due to this normalizing across activations within a layer, the same perturbation scale can be used across all layers, even though different layers may exhibit different sensitivities to noise.

Adaptive noise scaling Parameter space noise requires us to pick a suitable scale σ. This can be problematic since the scale will strongly depend on the specific network architecture, and is likely to vary over time as parameters become more sensitive to noise as learning progresses. Additionally, while it is easy to intuitively grasp the scale of action space noise, it is far harder to understand the scale in parameter space. We propose a simple solution that resolves all aforementioned limitations in an easy and straightforward way. This is achieved by adapting the scale of the parameter space noise over time and relating it to the variance in action space that it induces. More concretely, we can define a distance measure between perturbed and non-perturbed policy in action space and adaptively increase or decrease the parameter space noise depending on whether it is below or above a certain threshold:

$$\sigma_{k+1} = \begin{cases} \alpha\sigma_k & \text{if } d(\pi, \widetilde{\pi}) \leq \delta, \\ \frac{1}{\alpha}\sigma_k & \text{otherwise,} \end{cases} \tag{1}$$

where $\alpha \in \mathbb{R}_{>0}$ is a scaling factor and $\delta \in \mathbb{R}_{>0}$ a threshold value. The concrete realization of $d(\cdot, \cdot)$ depends on the algorithm at hand and we describe appropriate distance measures for DQN, DDPG, and TRPO in Appendix C.

Parameter space noise for off-policy methods In the off-policy case, parameter space noise can be applied straightforwardly since, by definition, data that was collected off-policy can be used. More concretely, we only perturb the policy for exploration and train the non-perturbed network on this data by replaying it.

Parameter space noise for on-policy methods Parameter noise can be incorporated in an on-policy setting, using an adapted policy gradient, as set forth by [23]. Policy gradient methods optimize $\mathbb{E}_{\tau\sim(\pi,p)}[R(\tau)]$. Given a stochastic policy $\pi_\theta(a|s)$ with $\theta \sim \mathcal{N}(\phi, \Sigma)$, the expected return can be expanded using likelihood ratios and the re-parametrization trick [24] as

$$\nabla_{\phi,\Sigma}\mathbb{E}_\tau[R(\tau)] \approx \frac{1}{N}\sum_{\epsilon^i,\tau^i}\left[\sum_{t=0}^{T-1}\nabla_{\phi,\Sigma}\log\pi(a_t|s_t;\phi+\epsilon^i\Sigma^{\frac{1}{2}})R_t(\tau^i)\right] \tag{2}$$

[3] This is in contrast to [9], who use virtual batch normalization, which we found to perform less consistently

for N samples $\epsilon^i \sim \mathcal{N}(0, I)$ and $\tau^i \sim (\pi_{\phi+\epsilon^i \Sigma^{\frac{1}{2}}}, p)$ (see Appendix B for a full derivation). Rather than updating Σ according to the previously derived policy gradient, we fix its value to $\sigma^2 I$ and scale it adaptively as described in Appendix C.

4 Experiments

This section answers the following questions:

(i) Do existing state-of-the-art RL algorithms benefit from incorporating parameter space noise?

(ii) Does parameter space noise aid in exploring sparse reward environments more effectively?

(iii) How does parameter space noise exploration compare against evolution strategies with respect to sample efficiency?

4.1 Comparing Parameter Space Noise to Action Space Noise

The added value of parameter space noise over action space noise is measured on both high-dimensional discrete-action environments and continuous control tasks. For the discrete environments, comparisons are made using DQN, while DDPG and TRPO are used on the continuous control tasks.

Discrete-action environments For discrete-action environments, we use the Arcade Learning Environment (ALE, [25]) benchmark along with a standard DQN implementation. We compare a baseline DQN agent with ϵ-greedy action noise against a version of DQN with parameter noise. We linearly anneal ϵ from 1.0 to 0.1 over the first 1 million timesteps. For parameter noise, we adapt the scale using a simple heuristic that increases the scale if the KL divergence between perturbed and non-perturbed policy is less than the KL divergence between greedy and ϵ-greedy policy and decreases it otherwise (see Section C.1 for details). By using this approach, we achieve a fair comparison between action space noise and parameter space noise since the magnitude of the noise is similar and also avoid the introduction of an additional hyperparameter.

For parameter perturbation, we found it useful to reparametrize the network in terms of an explicit policy that represents the greedy policy π implied by the Q-values, rather than perturbing the Q-function directly. To represent the policy $\pi(a|s)$, we add a single fully connected layer after the convolutional part of the network, followed by a softmax output layer. Thus, π predicts a discrete probability distribution over actions, given a state. We find that perturbing π instead of Q results in more meaningful changes since we now define an explicit behavioral policy. In this setting, the Q-network is trained according to standard DQN practices. The policy π is trained by maximizing the probability of outputting the greedy action accordingly to the current Q-network. Essentially, the policy is trained to exhibit the same behavior as running greedy DQN. To rule out this double-headed version of DQN alone exhibits significantly different behavior, we always compare our parameter space noise approach against two baselines, regular DQN and two-headed DQN, both with ϵ-greedy exploration.

We furthermore randomly sample actions for the first 50 thousand timesteps in all cases to fill the replay buffer before starting training. Moreover, we found that parameter space noise performs better if it is combined with a bit of action space noise (we use a ϵ-greedy behavioral policy with $\epsilon = 0.01$ for the parameter space noise experiments). Full experimental details are described in Section A.1.

We chose 21 games of varying complexity, according to the taxonomy presented by [26]. The learning curves are shown in Figure 1 for a selection of games (see Appendix D for full results). Each agent is trained for $40\,\mathrm{M}$ frames. The overall performance is estimated by running each configuration with three different random seeds, and we plot the median return (line) as well as the interquartile range (shaded area). Note that performance is evaluated on the exploratory policy since we are interested in its behavior especially.

Overall, our results show that parameter space noise often outperforms action space noise, especially on games that require consistency (e.g. Enduro, Freeway) and performs comparably on the remaining ones. Additionally, learning progress usually starts much sooner when using parameter space noise. Finally, we also compare against a double-headed version of DQN with ϵ-greedy exploration to ensure that this change in architecture is not responsible for improved exploration, which our results confirm. Full results are available in Appendix D.

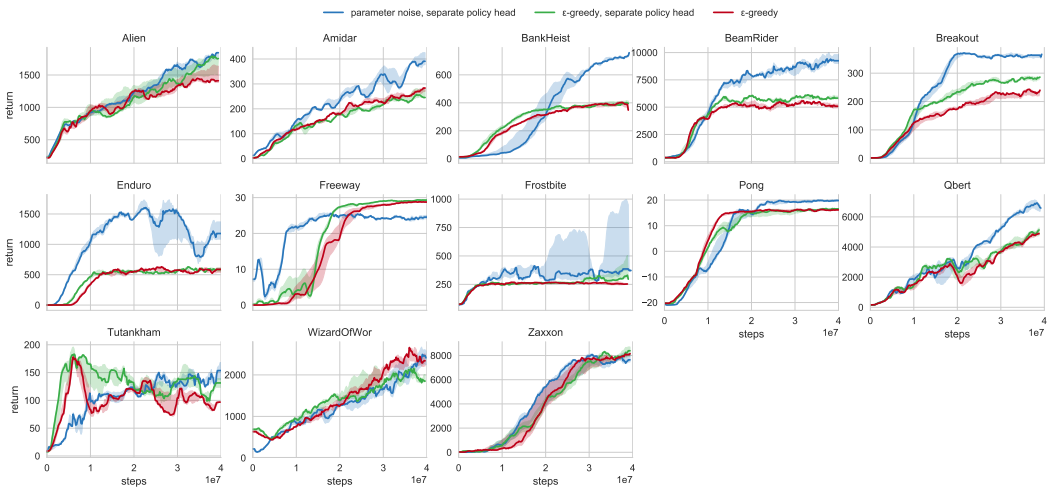

Figure 1: Median DQN returns for several ALE environment plotted over training steps.

On a final note, proposed improvements to DQN like double DQN [27], prioritized experience replay [28], and dueling networks [29] are orthogonal to our improvements and would therefore likely improve results further. We leave the experimental validation of this theory to future work.

Continuous control environments We now compare parameter noise with action noise on the continuous control environments implemented in OpenAI Gym [30]. We use DDPG [14] as the RL algorithm for all environments with similar hyperparameters as outlined in the original paper except for the fact that layer normalization [22] is applied after each layer before the nonlinearity, which we found to be useful in either case and especially important for parameter space noise.

We compare the performance of the following configurations: (a) no noise at all, (b) uncorrelated additive Gaussian action space noise ($\sigma = 0.2$), (c) correlated additive Gaussian action space noise (Ornstein–Uhlenbeck process [17] with $\sigma = 0.2$), and (d) adaptive parameter space noise. In the case of parameter space noise, we adapt the scale so that the resulting change in action space is comparable to our baselines with uncorrelated Gaussian action space noise (see Section C.2 for full details).

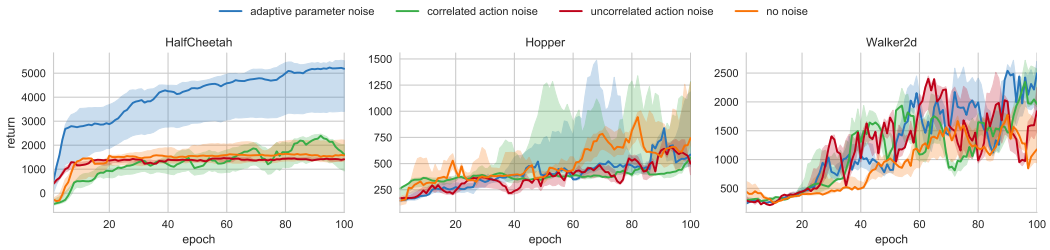

Figure 2: Median DDPG returns for continuous control environments plotted over epochs.

We evaluate the performance on several continuous control tasks. Figure 2 depicts the results for three exemplary environments. Each agent is trained for $1\,\mathrm{M}$ timesteps, where 1 epoch consists of 10 thousand timesteps. In order to make results comparable between configurations, we evaluate the performance of the agent every 10 thousand steps by using no noise for 20 episodes.

On *HalfCheetah*, parameter space noise achieves significantly higher returns than all other configurations. We find that, in this environment, all other exploration schemes quickly converge to a local optimum (in which the agent learns to flip on its back and then "wiggles" its way forward). Parameter space noise behaves similarly initially but still explores other options and quickly learns to break out of this sub-optimal behavior. Also notice that parameter space noise vastly outperforms correlated action space noise on this environment, clearly indicating that there is a significant difference between the two. On the remaining two environments, parameter space noise performs on par with other

exploration strategies. Notice, however, that even if no noise is present, DDPG is capable of learning good policies. We find that this is representative for the remaining environments (see Appendix E for full results), which indicates that these environments do not require a lot of exploration to begin with due to their well-shaped reward function.

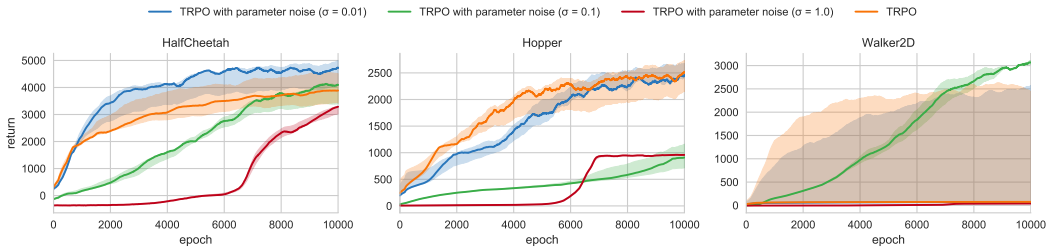

Figure 3: Median TRPO returns for continuous control environments plotted over epochs.

The results for TRPO are depicted in Figure 3. Interestingly, in the *Walker2D* environment, we see that adding parameter noise decreases the performance variance between seeds. This indicates that parameter noise aids in escaping local optima.

4.2 Does Parameter Space Noise Explore Efficiently?

The environments in the previous section required relatively little exploration. In this section, we evaluate whether parameter noise enables existing RL algorithms to learn on environments with very sparse rewards, where uncorrelated action noise generally fails [4, 7].

A scalable toy example We first evaluate parameter noise on a well-known toy problem, following the setup described by [7] as closely as possible. The environment consists of a chain of N states and the agent always starts in state s_2, from where it can either move left or right. In state s_1, the agent receives a small reward of $r = 0.001$ and a larger reward $r = 1$ in state s_N. Obviously, it is much easier to discover the small reward in s_1 than the large reward in s_N, with increasing difficulty as N grows. The environment is described in greater detail in Section A.3.

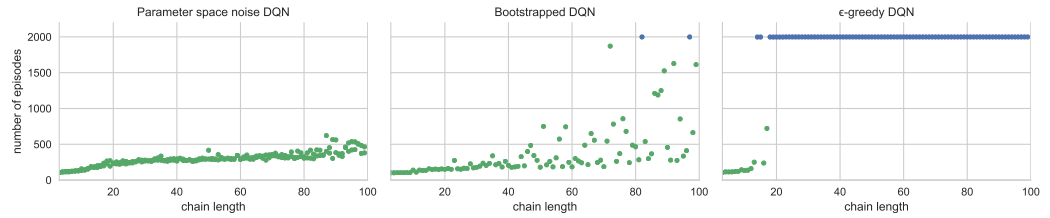

Figure 4: Median number of episodes before considered solved for DQN with different exploration strategies. Green indicates that the problem was solved whereas blue indicates that no solution was found within 2 K episodes. Note that less number of episodes before solved is better.

We compare adaptive parameter space noise DQN, bootstrapped DQN, and ϵ-greedy DQN. The chain length N is varied and for each N three different seeds are trained and evaluated. After each episode, we evaluate the performance of the current policy by performing a rollout with all noise disabled (in the case of bootstrapped DQN, we perform majority voting over all heads). The problem is considered solved if one hundred subsequent rollouts achieve the optimal return. We plot the median number of episodes before the problem is considered solved (we abort if the problem is still unsolved after 2 thousand episodes). Full experimental details are available in Section A.3.

Figure 4 shows that parameter space noise clearly outperforms action space noise (which completely fails for moderately large N) and even outperforms the more computational expensive bootstrapped DQN.

Continuous control with sparse rewards We now make the continuous control environments more challenging for exploration. Instead of providing a reward at every timestep, we use environments that only yield a non-zero reward after significant progress towards a goal. More concretely, we consider the following environments from rllab[4] [31], modified according to [3]: (a) *SparseCartpoleSwingup*, which only yields a reward if the paddle is raised above a given threshold, (b) *SparseDoublePendulum*, which only yields a reward if the agent reaches the upright position, and (c) *SparseHalfCheetah*, which only yields a reward if the agent crosses a target distance, (d) *SparseMountainCar*, which only yields a reward if the agent drives up the hill, (e) *SwimmerGather*, yields a positive or negative reward upon reaching targets. For all tasks, we use a time horizon of $T = 500$ steps before resetting.

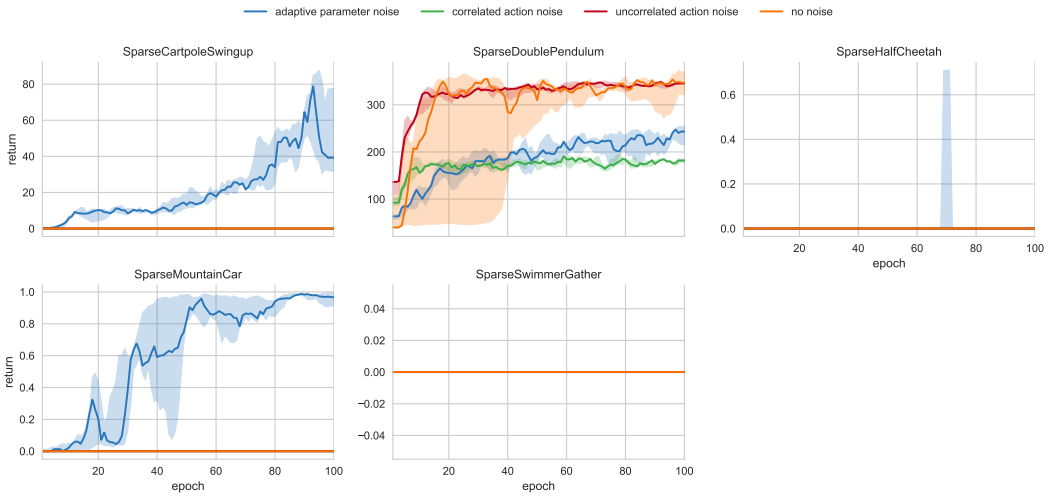

Figure 5: Median DDPG returns for environments with sparse rewards plotted over epochs.

We consider both DDPG and TRPO to solve these environments (the exact experimental setup is described in Section A.2). Figure 5 shows the performance of DDPG, while the results for TRPO have been moved to Appendix F. The overall performance is estimated by running each configuration with five different random seeds, after which we plot the median return (line) as well as the interquartile range (shaded area).

For DDPG, *SparseDoublePendulum* seems to be easy to solve in general, with even no noise finding a successful policy relatively quickly. The results for *SparseCartpoleSwingup* and *SparseMountainCar* are more interesting: Here, only parameter space noise is capable of learning successful policies since all other forms of noise, including correlated action space noise, never find states with non-zero rewards. For *SparseHalfCheetah*, DDPG at least finds the non-zero reward but never learns a successful policy from that signal. On the challenging *SwimmerGather* task, all configurations of DDPG fail.

Our results clearly show that parameter space noise can be used to improve the exploration behavior of these off-the-shelf algorithms.

4.3 Is RL with Parameter Space Noise more Sample-efficient than ES?

Evolution strategies (ES) are closely related to our approach since both explore by introducing noise in the parameter space, which can lead to improved exploration behavior [9]. However, ES disregards temporal information and uses black-box optimization to train the neural network. By combining parameter space noise with traditional RL algorithms, we can include temporal information as well rely on gradients computed by back-propagation for optimization while still benefiting from improved exploratory behavior. We now compare ES and traditional RL with parameter space noise directly.

We compare performance on the 21 ALE games that were used in Section 4.1. The performance is estimated by running 10 episodes for each seed using the final policy with exploration disabled and

[4] `https://github.com/openai/rllab`

computing the median returns. For ES, we use the results obtained by [9], which were obtained after training on $1\,000\,\mathrm{M}$ frames. For DQN, we use the same parameter space noise for exploration that was previously described and train on $40\,\mathrm{M}$ frames. Even though DQN with parameter space noise has been exposed to 25 times less data, it outperforms ES on 15 out of 21 Atari games (full results are available in Appendix D). Combined with the previously described results, this demonstrates that parameter space noise combines the desirable exploration properties of ES with the sample efficiency of traditional RL.

5 Related Work

The problem of exploration in reinforcement has been studied extensively. A range of algorithms [32–34] have been proposed that guarantee near-optimal solutions after a number of steps that are polynomial in the number of states, number of actions, and the horizon time. However, in many real-world reinforcements learning problems both the state and action space are continuous and high dimensional so that, even with discretization, these algorithms become impractical. In the context of deep reinforcement learning, a large variety of techniques have been proposed to improve exploration [1–3, 5, 7, 35, 36]. However, all are non-trivial to implement and are often computational expensive.

The idea of perturbing the parameters of a policy has been proposed by [10] for policy gradient methods. The authors show that this form of perturbation generally outperforms random exploration and evaluate their exploration strategy with the REINFORCE [37] and Natural Actor-Critic [20] algorithms. However, their policies are relatively low-dimensional compared to modern deep architectures, they use environments with low-dimensional state spaces, and their contribution is strictly limited to the policy gradient case. In contrast, our method is applied and evaluated for both on and off-policy setting, we use high-dimensional policies, and environments with large state spaces.

Our work is also closely related to evolution strategies (ES, [38, 39]), and especially neural evolution strategies (NES, [8, 40–44]). In the context of policy optimization, our work is closely related to [11] and [12]. More recently, [9] showed that ES can work for high-dimensional environments like Atari and OpenAI Gym continuous control problems. However, ES generally disregards any temporal structure that may be present in trajectories and typically suffers from sample inefficiency.

Bootstrapped DQN [7] has been proposed to aid with more directed and consistent exploration by using a network with multiple heads, where one specific head is selected at the beginning of each episode. In contrast, our approach perturbs the parameters of the network directly, thus achieving similar yet simpler (and as shown in Section 4.2, sometimes superior) exploration behavior. Concurrently to our work, [45] have proposed a similar approach that utilizes parameter perturbations for more efficient exploration.

6 Conclusion

On the one hand, evolutionary methods discard temporal structure, which makes credit assignment more difficult and results in worse sample-efficiency. On the other hand, traditional RL methods often rely solely on unstructured action noise. This work shows that combining parameter perturbations with contemporary on- and off-policy deep RL algorithms such as DQN, DDPG, and TRPO allows for structured exploration while maintaining the properties of sample efficiency and exploitation of temporal structure that traditional RL approaches enjoy. We show that parameter noise can be applied to these off-the-shelf algorithms and often results in improved performance compared to action noise. Experimental results further demonstrate that using parameter noise allows solving environments with very sparse rewards, in which action noise is unlikely to succeed. Results also indicate that RL with parameter noise exploration learns more efficiently than both RL and evolutionary strategies individually.

References

[1] H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "#Exploration: A study of count-based exploration for deep reinforcement learning," *arXiv preprint arXiv:1611.04717*, 2016.

[2] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos, "Count-based exploration with neural density models," *arXiv preprint arXiv:1703.01310*, 2017. [Online]. Available: http://arxiv.org/abs/1703.01310.

[3] R. Houthooft, X. Chen, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel, "VIME: Variational information maximizing exploration," in *Advances in Neural Information Processing Systems 29 (NIPS)*, 2016, pp. 1109–1117. [Online]. Available: http://papers.nips.cc/paper/6591-vime-variational-information-maximizing-exploration.

[4] J. Achiam and S. Sastry, "Surprise-based intrinsic motivation for deep reinforcement learning," *arXiv preprint arXiv:1703.01732*, 2017.

[5] B. C. Stadie, S. Levine, and P. Abbeel, "Incentivizing exploration in reinforcement learning with deep predictive models," *arXiv preprint arXiv:1507.00814*, 2015. [Online]. Available: http://arxiv.org/abs/1507.00814.

[6] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *ICML*, 2017.

[7] I. Osband, C. Blundell, A. Pritzel, and B. V. Roy, "Deep exploration via bootstrapped DQN," in *Advances in Neural Information Processing Systems 29 (NIPS)*, 2016, pp. 4026–4034. [Online]. Available: http://papers.nips.cc/paper/6501-deep-exploration-via-bootstrapped-dqn.

[8] Y. Sun, D. Wierstra, T. Schaul, and J. Schmidhuber, "Efficient natural evolution strategies," in *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009*, 2009, pp. 539–546. DOI: 10.1145/1569901.1569976. [Online]. Available: http://doi.acm.org/10.1145/1569901.1569976.

[9] T. Salimans, J. Ho, X. Chen, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017. [Online]. Available: http://arxiv.org/abs/1703.03864.

[10] T. Rückstieß, M. Felder, and J. Schmidhuber, "State-dependent exploration for policy gradient methods," in *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases ECML/PKDD*, 2008, pp. 234–249. DOI: 10.1007/978-3-540-87481-2_16. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-87481-2_16.

[11] J. Kober and J. Peters, "Policy search for motor primitives in robotics," in *Advances in Neural Information Processing Systems 21 (NIPS)*, 2008, pp. 849–856. [Online]. Available: http://papers.nips.cc/paper/3545-policy-search-for-motor-primitives-in-robotics.

[12] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber, "Parameter-exploring policy gradients," *Neural Networks*, vol. 23, no. 4, pp. 551–559, 2010. DOI: 10.1016/j.neunet.2009.12.004. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2009.12.004.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. DOI: 10.1038/nature14236. [Online]. Available: http://dx.doi.org/10.1038/nature14236.

[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971.

[15] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, 2015, pp. 1889–1897. [Online]. Available: http://jmlr.org/proceedings/papers/v37/schulman15.html.

[16] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT Press Cambridge, 1998, vol. 135.

[17] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical review*, vol. 36, no. 5, p. 823, 1930.

[18] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 1889–1897.

[19] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[20] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7-9, pp. 1180–1190, 2008. DOI: 10.1016/j.neucom.2007.11.026. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2007.11.026.

[21] S. Kakade, "A natural policy gradient," *Advances in neural information processing systems*, vol. 14, pp. 1531–1538, 2001.

[22] L. J. Ba, R. Kiros, and G. E. Hinton, "Layer normalization," *CoRR*, vol. abs/1607.06450, 2016. [Online]. Available: http://arxiv.org/abs/1607.06450.

[23] T. Rückstieß, M. Felder, and J. Schmidhuber, "State-dependent exploration for policy gradient methods," *Machine Learning and Knowledge Discovery in Databases*, pp. 234–249, 2008.

[24] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[25] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013. DOI: 10.1613/jair.3912. [Online]. Available: http://dx.doi.org/10.1613/jair.3912.

[26] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Advances in Neural Information Processing Systems 29 (NIPS)*, 2016, pp. 1471–1479.

[27] H. V. Hasselt, "Double Q-learning," in *Advances in Neural Information Processing Systems 23 (NIPS)*, 2010, pp. 2613–2621.

[28] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[29] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[30] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," *arXiv preprint arXiv:1606.01540*, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540.

[31] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continous control," in *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016, pp. 1329–1338.

[32] M. J. Kearns and S. P. Singh, "Near-optimal reinforcement learning in polynomial time," *Machine Learning*, vol. 49, no. 2-3, pp. 209–232, 2002. DOI: 10.1023/A:1017984413808. [Online]. Available: http://dx.doi.org/10.1023/A:1017984413808.

[33] R. I. Brafman and M. Tennenholtz, "R-MAX - A general polynomial time algorithm for near-optimal reinforcement learning," *Journal of Machine Learning Research*, vol. 3, pp. 213–231, 2002. [Online]. Available: http://www.jmlr.org/papers/v3/brafman02a.html.

[34] P. Auer, T. Jaksch, and R. Ortner, "Near-optimal regret bounds for reinforcement learning," in *Advances in Neural Information Processing Systems 21 (NIPS)*, 2008, pp. 89–96. [Online]. Available: http://papers.nips.cc/paper/3401-near-optimal-regret-bounds-for-reinforcement-learning.

[35] S. Sukhbaatar, I. Kostrikov, A. Szlam, and R. Fergus, "Intrinsic motivation and automatic curricula via asymmetric self-play," *arXiv preprint arXiv:1703.05407*, 2017. [Online]. Available: http://arxiv.org/abs/1703.05407.

[36] I. Osband, B. V. Roy, and Z. Wen, "Generalization and exploration via randomized value functions," in *Proceedings of the 33nd International Conference on Machine Learning, ICML*, 2016, pp. 2377–2386. [Online]. Available: http://jmlr.org/proceedings/papers/v48/osband16.html.

[37] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992. DOI: 10.1007/BF00992696. [Online]. Available: http://dx.doi.org/10.1007/BF00992696.

[38] I. Rechenberg and M. Eigen, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologishen Evolution*. Frommann-Holzboog Stuttgart, 1973.

[39] H.-P. Schwefel, *Numerische Optimierung von Computermodellen mittels der Evolutionsstrategie*. Birkhäuser, Basel Switzerland, 1977, vol. 1.

[40] Y. Sun, D. Wierstra, T. Schaul, and J. Schmidhuber, "Stochastic search using the natural gradient," in *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, 2009, pp. 1161–1168. DOI: 10.1145/1553374.1553522. [Online]. Available: http://doi.acm.org/10.1145/1553374.1553522.

[41] T. Glasmachers, T. Schaul, and J. Schmidhuber, "A natural evolution strategy for multi-objective optimization," in *Parallel Problem Solving from Nature - PPSN XI, 11th International Conference, Kraków, Poland, September 11-15, 2010, Proceedings, Part I*, 2010, pp. 627–636. DOI: 10.1007/978-3-642-15844-5_63. [Online]. Available: https://doi.org/10.1007/978-3-642-15844-5_63.

[42] T. Glasmachers, T. Schaul, Y. Sun, D. Wierstra, and J. Schmidhuber, "Exponential natural evolution strategies," in *Genetic and Evolutionary Computation Conference, GECCO 2010, Proceedings, Portland, Oregon, USA, July 7-11, 2010*, 2010, pp. 393–400. DOI: 10.1145/1830483.1830557. [Online]. Available: http://doi.acm.org/10.1145/1830483.1830557.

[43] T. Schaul, T. Glasmachers, and J. Schmidhuber, "High dimensions and heavy tails for natural evolution strategies," in *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011*, 2011, pp. 845–852. DOI: 10.1145/2001576.2001692. [Online]. Available: http://doi.acm.org/10.1145/2001576.2001692.

[44] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, "Natural evolution strategies," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 949–980, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2638566.

[45] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, *et al.*, "Noisy networks for exploration," *arXiv preprint arXiv:1706.10295*, 2017.

[46] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[47] A. Ranganathan, "The Levenberg-Marquardt algorithm," *Tutoral on LM algorithm*, pp. 1–5, 2004.

A Experimental Setup

A.1 Arcade Learning Environment (ALE)

For ALE [25], the network architecture as described in [13] is used. This consists of 3 convolutional layers (32 filters of size 8×8 and stride 4, 64 filters of size 4×4 and stride 2, 64 filters of size 3×3 and stride 1) followed by 1 hidden layer with 512 units followed by a linear output layer with one unit for each action. ReLUs are used in each layer, while layer normalization [22] is used in the fully connected part of the network. For parameter space noise, we also include a second head after the convolutional stack of layers. This head determines a policy network with the same architecture as the Q-value network, except for a softmax output layer. The target networks are updated every $10\,\mathrm{K}$ timesteps. The Q-value network is trained using the Adam optimizer [46] with a learning rate of 10^{-4} and a batch size of 32. The replay buffer can hold $1\,\mathrm{M}$ state transitions. For the ϵ-greedy baseline, we linearly anneal ϵ from 1 to 0.1 over the first $1\,\mathrm{M}$ timesteps. For parameter space noise, we adaptively scale the noise to have a similar effect in action space (see Section C.1 for details), effectively ensuring that the maximum KL divergence between perturbed and non-perturbed π is softly enforced. The policy is perturbed at the beginning of each episode and the standard deviation is adapted as described in Appendix C every 50 timesteps. Notice that we only perturb the policy head after the convolutional part of the network (i.e. the fully connected part, which is also why we only include layer normalization in this part of the network). To avoid getting stuck (which can potentially happen for a perturbed policy), we also use ϵ-greedy action selection with $\epsilon = 0.01$. In all cases, we perform $50\,\mathrm{K}$ random actions to collect initial data for the replay buffer before training starts. We set $\gamma = 0.99$, clip rewards to be in $[-1, 1]$, and clip gradients for the output layer of Q to be within $[-1, 1]$. For observations, each frame is down-sampled to 84×84 pixels, after which it is converted to grayscale. The actual observation to the network consists of a concatenation of 4 subsequent frames. Additionally, we use up to 30 noop actions at the beginning of the episode. This setup is identical to what is described by [13].

A.2 Continuous Control

For DDPG, we use a similar network architecture as described by [14]: both the actor and critic use 2 hidden layers with 64 ReLU units each. For the critic, actions are not included until the second hidden layer. Layer normalization [22] is applied to all layers. The target networks are soft-updated with $\tau = 0.001$. The critic is trained with a learning rate of 10^{-3} while the actor uses a learning rate of 10^{-4}. Both actor and critic are updated using the Adam optimizer [46] with batch sizes of 128. The critic is regularized using an $L2$ penalty with 10^{-2}. The replay buffer holds $100\,\mathrm{K}$ state transitions and $\gamma = 0.99$ is used. Each observation dimension is normalized by an online estimate of the mean and variance. For parameter space noise with DDPG, we adaptively scale the noise to be comparable to the respective action space noise (see Section C.2). For dense environments, we use action space noise with $\sigma = 0.2$ (and a comparable adaptive noise scale). Sparse environments use an action space noise with $\sigma = 0.6$ (and a comparable adaptive noise scale).

TRPO uses a step size of $\delta_{\mathrm{KL}} = 0.01$, a policy network of 2 hidden layers with 32 $\tanh$ units for the nonlocomotion tasks, and 2 hidden layers of 64 $\tanh$ units for the locomotion tasks. The Hessian calculation is subsampled with a factor of 0.1, $\gamma = 0.99$, and the batch size per epoch is set to $5\,\mathrm{K}$ timesteps. The baseline is a learned linear transformation of the observations.

The following environments from OpenAI Gym[5] [30] are used:

- *HalfCheetah* ($\mathcal{S} \subset \mathbb{R}^{17}$, $\mathcal{A} \subset \mathbb{R}^6$),
- *Hopper* ($\mathcal{S} \subset \mathbb{R}^{11}$, $\mathcal{A} \subset \mathbb{R}^3$),
- *InvertedDoublePendulum* ($\mathcal{S} \subset \mathbb{R}^{11}$, $\mathcal{A} \subset \mathbb{R}$),
- *InvertedPendulum* ($\mathcal{S} \subset \mathbb{R}^4$, $\mathcal{A} \subset \mathbb{R}$),
- *Reacher* ($\mathcal{S} \subset \mathbb{R}^{11}$, $\mathcal{A} \subset \mathbb{R}^2$),
- *Swimmer* ($\mathcal{S} \subset \mathbb{R}^8$, $\mathcal{A} \subset \mathbb{R}^2$), and
- *Walker2D* ($\mathcal{S} \subset \mathbb{R}^{17}$, $\mathcal{A} \subset \mathbb{R}^6$).

[5] `https://github.com/openai/gym`

For the sparse tasks, we use the following environments from rllab[6] [31], modified as described by [3]:

- *SparseCartpoleSwingup* ($\mathcal{S} \subset \mathbb{R}^4$, $\mathcal{A} \subset \mathbb{R}$), which only yields a reward if the paddle is raised above a given threshold,
- *SparseHalfCheetah* ($\mathcal{S} \subset \mathbb{R}^{17}$, $\mathcal{A} \subset \mathbb{R}^6$), which only yields a reward if the agent crosses a distance threshold,
- *SparseMountainCar* ($\mathcal{S} \subset \mathbb{R}^2$, $\mathcal{A} \subset \mathbb{R}$), which only yields a reward if the agent drives up the hill,
- *SparseDoublePendulum* ($\mathcal{S} \subset \mathbb{R}^6$, $\mathcal{A} \subset \mathbb{R}$), which only yields a reward if the agent reaches the upright position, and
- *SwimmerGather* ($\mathcal{S} \subset \mathbb{R}^{33}$, $\mathcal{A} \subset \mathbb{R}^2$), which yields a positive or negative reward upon reaching targets.

A.3 Chain Environment

We follow the state encoding proposed by [7] and use $\phi(s_t) = (\mathbb{1}\{x \leq s_t\})$ as the observation, where $\mathbb{1}$ denotes the indicator function. DQN is used with a very simple network to approximate the Q-value function that consists of 2 hidden layers with 16 ReLU units. Layer normalization [22] is used for all hidden layers before applying the nonlinearity. Each agent is then trained for up to $2\,\mathrm{K}$ episodes. The chain length N is varied and for each N three different seeds are trained and evaluated. After each episode, the performance of the current policy is evaluated by sampling a trajectory with noise disabled (in the case of bootstrapped DQN, majority voting over all heads is performed). The problem is considered solved if one hundred subsequent trajectories achieve the optimal episode return. Figure 6 depicts the environment.

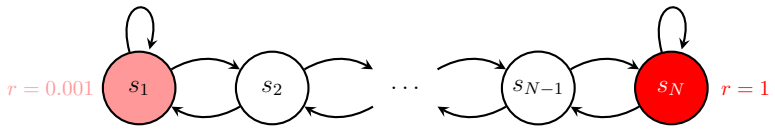

Figure 6: Simple and scalable environment to test for exploratory behavior [7].

We compare adaptive parameter space noise DQN, bootstrapped DQN [7] (with $K = 20$ heads and Bernoulli masking with $p = 0.5$), and ϵ-greedy DQN (with ϵ linearly annealed from 1.0 to 0.1 over the first one hundred episodes). For adaptive parameter space noise, we only use a single head and perturb Q directly, which works well in this setting. Parameter space noise is adaptively scaled so that $\delta \approx 0.05$. In all cases, $\gamma = 0.999$, the replay buffer holds $100\,\mathrm{K}$ state transitions, learning starts after 5 initial episodes, the target network is updated every 100 timesteps, and the network is trained using the Adam optimizer [46] with a learning rate of 10^{-3} and a batch size of 32.

B Parameter Space Noise for On-policy Methods

Policy gradient methods optimize $\mathbb{E}_{\tau\sim(\pi,p)}[R(\tau)]$. Given a stochastic policy $\pi_\theta(a|s)$ with $\theta \sim \mathcal{N}(\phi, \Sigma)$, the expected return can be expanded using likelihood ratios and the reparametrization trick [24] as

$$\nabla_{\phi,\Sigma}\mathbb{E}_\tau[R(\tau)] = \nabla_{\phi,\Sigma}\mathbb{E}_{\theta\sim\mathcal{N}(\phi,\Sigma)}\left[\sum_\tau p(\tau|\theta)R(\tau)\right] \tag{3}$$

$$= \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}\nabla_{\phi,\Sigma}\left[\sum_\tau p(\tau|\phi + \epsilon\Sigma^{\frac{1}{2}})R(\tau)\right] \tag{4}$$

$$= \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I),\tau}\left[\sum_{t=0}^{T-1}\nabla_{\phi,\Sigma}\log\pi(a_t|s_t;\phi + \epsilon\Sigma^{\frac{1}{2}})R_t(\tau)\right] \tag{5}$$

[6] `https://github.com/openai/rllab`

$$\approx \frac{1}{N} \sum_{\epsilon^i, \tau^i} \left[\sum_{t=0}^{T-1} \nabla_{\phi,\Sigma} \log \pi(a_t | s_t; \phi + \epsilon^i \Sigma^{\frac{1}{2}}) R_t(\tau^i) \right] \tag{6}$$

for N samples $\epsilon^i \sim \mathcal{N}(0, I)$ and $\tau^i \sim (\pi_{\phi+\epsilon^i\Sigma^{\frac{1}{2}}}, p)$, with $R_t(\tau^i) = \sum_{t'=t}^{T} \gamma^{t'-t} r^i_{t'}$. This also allows us to subtract a variance-reducing baseline b^i_t, leading to

$$\nabla_{\phi,\Sigma} \mathbb{E}_\tau[R(\tau)] \approx \frac{1}{N} \sum_{\epsilon^i, \tau^i} \left[\sum_{t=0}^{T-1} \nabla_{\phi,\Sigma} \log \pi(a_t | s_t; \phi + \epsilon^i \Sigma^{\frac{1}{2}}) (R_t(\tau^i) - b^i_t) \right]. \tag{7}$$

In our case, we set $\Sigma := \sigma^2 I$ and use our proposed adaption method to re-scale as appropriate.

C Adaptive Scaling

Parameter space noise requires us to pick a suitable scale σ. This can be problematic since the scale will highly depend on the specific network architecture, and is likely to vary over time as parameters become more sensitive as learning progresses. Additionally, while it is easy to intuitively grasp the scale of action space noise, it is far harder to understand the scale in parameter space.

We propose a simple solution that resolves all aforementioned limitations in an easy and straightforward way. This is achieved by adapting the scale of the parameter space noise over time, thus using a time-varying scale σ_k. Furthermore, σ_k is related to the action space variance that it induces, and updated accordingly. Concretely, we use the following simple heuristic to update σ_k every K timesteps:

$$\sigma_{k+1} = \begin{cases} \alpha \sigma_k, & \text{if } d(\pi, \widetilde{\pi}) < \delta \\ \frac{1}{\alpha} \sigma_k, & \text{otherwise,} \end{cases} \tag{8}$$

where $d(\cdot, \cdot)$ denotes some distance between the non-perturbed and perturbed policy (thus measuring in action space), $\alpha \in \mathbb{R}_{>0}$ is used to rescale σ_k, and $\delta \in \mathbb{R}_{>0}$ denotes some threshold value. This idea is based on the Levenberg-Marquardt heuristic [47]. The concrete distance measure and appropriate choice of δ depends on the policy representation. In the following sections, we outline our choice of $d(\cdot, \cdot)$ for methods that do (DDPG and TRPO) and do not (DQN) use behavioral policies. In our experiments, we always use $\alpha = 1.01$.

C.1 A Distance Measure for DQN

For DQN, the policy is defined implicitly by the Q-value function. Unfortunately, this means that a naïve distance measure between Q and $\widetilde{Q}$ has pitfalls. For example, assume that the perturbed policy has only changed the bias of the final layer, thus adding a constant value to each action's Q-value. In this case, a naïve distance measure like the norm $\|Q - \widetilde{Q}\|_2$ would be nonzero, although the policies π and $\widetilde{\pi}$ (implied by Q and $\widetilde{Q}$, respectively) are exactly equal. This equally applies to the case where DQN as two heads, one for Q and one for π.

We therefore use a probabilistic formulation[7] for both the non-perturbed and perturbed policies: $\pi, \widetilde{\pi} : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ by applying the softmax function over predicted Q values: $\pi(s) = \exp Q_i(s) / \sum_i \exp Q_i(s)$, where $Q_i(\cdot)$ denotes the Q-value of the i-th action. $\widetilde{\pi}$ is defined analogously but uses the perturbed $\widetilde{Q}$ instead (or the perturbed head for π). Using this probabilistic formulation of the policies, we can now measure the distance in action space:

$$d(\pi, \widetilde{\pi}) = D_{\mathrm{KL}}(\pi \parallel \widetilde{\pi}), \tag{9}$$

where $D_{\mathrm{KL}}(\cdot \parallel \cdot)$ denotes the Kullback-Leibler (KL) divergence. This formulation effectively normalizes the Q-values and therefore does not suffer from the problem previously outlined.

We can further relate this distance measure to ϵ-greedy action space noise, which allows us to fairly compare the two approaches and also avoids the need to pick an additional hyperparameter δ. More

[7] It is important to note that we use this probabilistic formulation only for the sake of defining a well-behaved distance measure. The actual policy used for rollouts is still deterministic.

concretely, the KL divergence between a greedy policy $\pi(s,a) = 1$ for $a = \text{argmax}_{a'} Q(s,a')$ and $\pi(s,a) = 0$ otherwise and an ϵ-greedy policy $\widehat{\pi}(s,a) = 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|}$ for $a = \text{argmax}_{a'} Q(s,a')$ and $\widehat{\pi}(s,a) = \frac{\epsilon}{|\mathcal{A}|}$ otherwise is $D_{\text{KL}}(\pi \parallel \widehat{\pi}) = -\log(1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|})$, where $|\mathcal{A}|$ denotes the number of actions (this follows immediately from the definition of the KL divergence for discrete probability distributions). We can use this distance measure to relate action space noise and parameter space noise to have similar distances, by adaptively scaling σ so that it matches the KL divergence between greedy and ϵ-greedy policy, thus setting $\delta := -\log(1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|})$.

C.2 A Distance Measure for DDPG

For DDPG, we relate noise induced by parameter space perturbations to noise induced by additive Gaussian noise. To do so, we use the following distance measure between the non-perturbed and perturbed policy:

$$d(\pi, \widetilde{\pi}) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_s \left[(\pi(s)_i - \widetilde{\pi}(s)_i)^2 \right]}, \tag{10}$$

where $\mathbb{E}_s[\cdot]$ is estimated from a batch of states from the replay buffer and N denotes the dimension of the action space (i.e. $\mathcal{A} \subset \mathbb{R}^N$). It is easy to show that $d(\pi, \pi + \mathcal{N}(0, \sigma^2 I)) = \sigma$. Setting $\delta := \sigma$ as the adaptive parameter space threshold thus results in effective action space noise that has the same standard deviation as regular Gaussian action space noise.

C.3 A Distance Measure for TRPO

In order to scale the noise for TRPO, we adapt the sampled noise vectors $\epsilon\sigma$ by computing a natural step $H^{-1}\epsilon\sigma$. We essentially compute a trust region around the noise direction to ensure that the perturbed policy $\widetilde{\pi}$ remains sufficiently close to the non-perturbed version via

$$E_{s \sim \rho_{\widetilde{\theta}}}[D_{\text{KL}}(\pi_{\widetilde{\theta}}(\cdot|s) \| \pi_\theta(\cdot|s))] \leq \delta_{\text{KL}}.$$

Concretely, this is computed through the conjugate gradient algorithm, combined with a line search along the noise direction to ensure constraint conformation, as described in Appendix C of [15].

D Additional Results on ALE

Figure 7 provide the learning curves for all 21 Atari games.

Table 1 compares the final performance of ES after $1\,000\,\text{M}$ frames to the final performance of DQN with ϵ-greedy exploration and parameter space noise exploration after $40\,\text{M}$ frames. In all cases, the performance is estimated by running 10 episodes with exploration disabled. We use the numbers reported by [9] for ES and report the median return across three seeds for DQN.

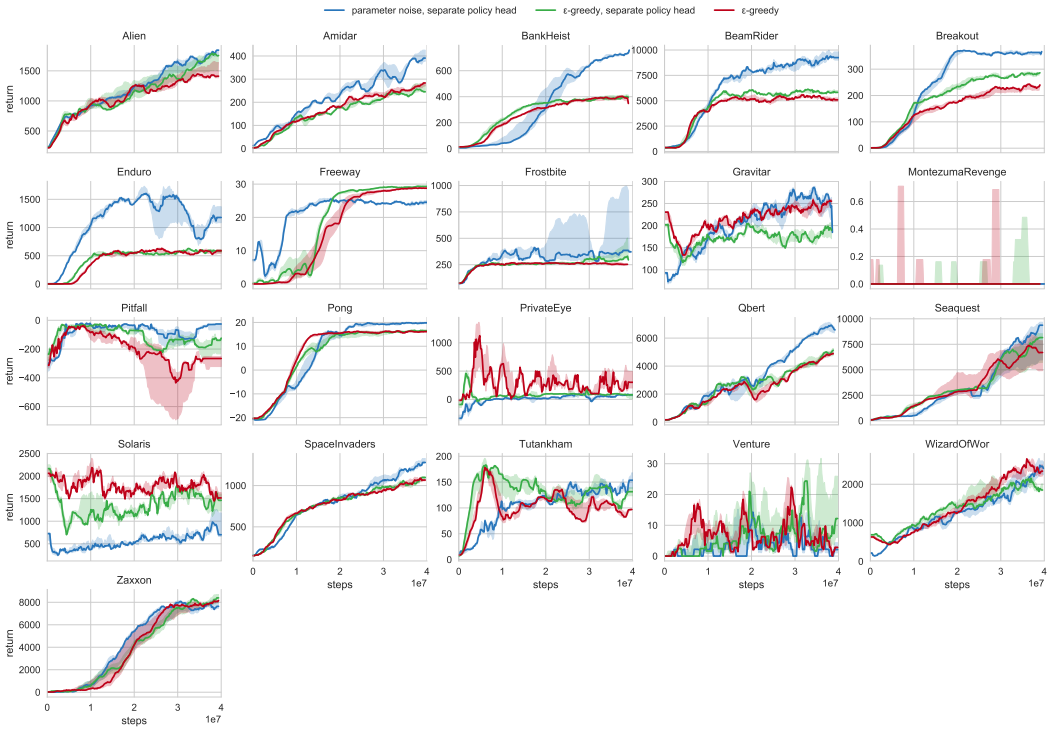

Figure 7: Median DQN returns for all ALE environment plotted over training steps.

Table 1: Performance comparison between Evolution Strategies (ES) as reported by [9], DQN with ϵ-greedy, and DQN with parameter space noise (this paper). ES was trained on $1\,000$ M, while DQN was trained on only 40 M frames.

Game	ES	DQN w/ ϵ-greedy	DQN w/ param noise
Alien	994.0	1535.0	**2070.0**
Amidar	112.0	281.0	**403.5**
BankHeist	225.0	510.0	**805.0**
BeamRider	744.0	**8184.0**	7884.0
Breakout	9.5	**406.0**	390.5
Enduro	95.0	1094	**1672.5**
Freeway	31.0	**32.0**	31.5
Frostbite	370.0	250.0	**1310.0**
Gravitar	**805.0**	300.0	250.0
MontezumaRevenge	**0.0**	**0.0**	**0.0**
Pitfall	**0.0**	-73.0	-100.0
Pong	**21.0**	**21.0**	20.0
PrivateEye	100.0	**133.0**	100.0
Qbert	147.5	**7625.0**	7525.0
Seaquest	1390.0	8335.0	**8920.0**
Solaris	**2090.0**	720.0	400.0
SpaceInvaders	678.5	1000.0	**1205.0**
Tutankham	130.3	109.5	**181.0**
Venture	**760.0**	0	0
WizardOfWor	**3480.0**	2350.0	1850.0
Zaxxon	6380.0	**8100.0**	8050.0

E Additional Results on Continuous Control with Shaped Rewards

For completeness, we provide the plots for all evaluated environments with dense rewards. The results are depicted in Figure 8.

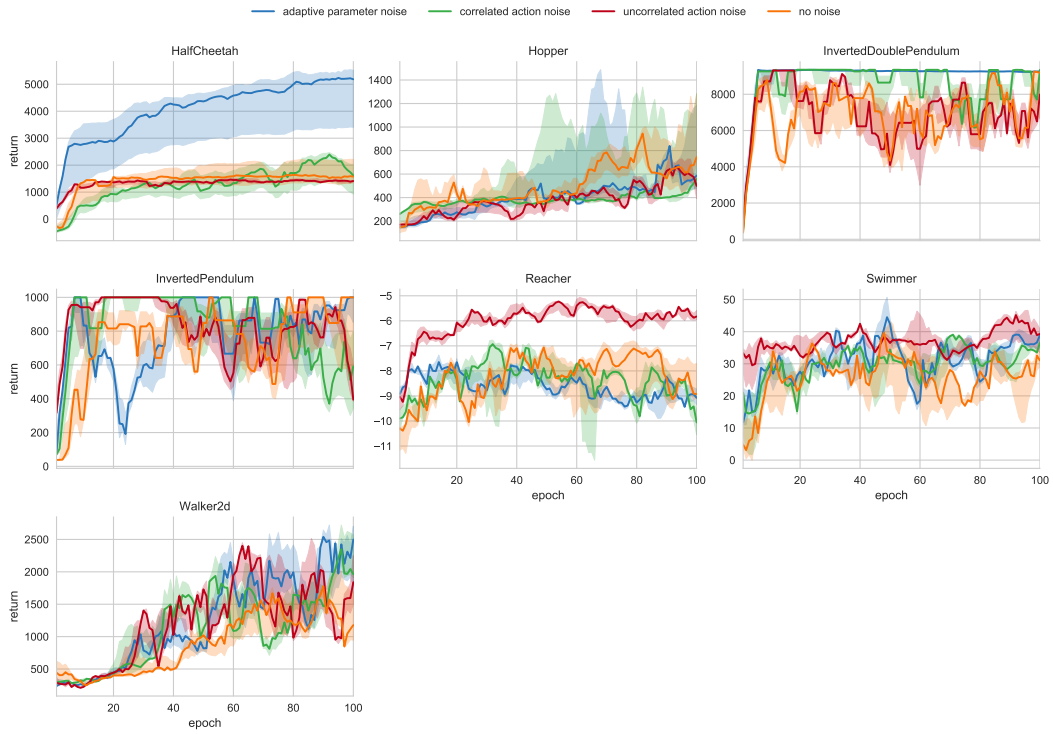

Figure 8: Median DDPG returns for all evaluated environments with dense rewards plotted over epochs.

The results for *InvertedPendulum* and *InvertedDoublePendulum* are very noisy due to the fact that a small change in policy can easily degrade performance significantly, and thus hard to read. Interestingly, adaptive parameter space noise achieves the most stable performance on *InvertedDoublePendulum*. Overall, performance is comparable to other exploration approaches. Again, no noise in either the action nor the parameter space achieves comparable results, indicating that these environments combined with DDPG are not well-suited to test for exploration.

F Additional Results on Continuous Control with Sparse Rewards

The performance of TRPO with noise scaled according to the parameter curvature, as defined in Section C.3 is shown in Figure 9. The TRPO baseline uses only action noise by using a policy network that outputs the mean of a Gaussian distribution, while the variance is learned. These results show that adding parameter space noise aids in either learning much more consistently on these challenging sparse environments.

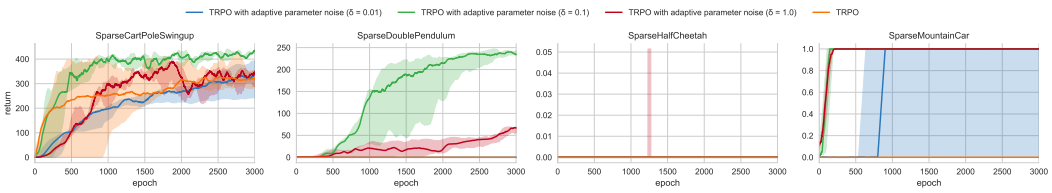

Figure 9: Median TRPO returns with three different environments with sparse rewards plotted over epochs.