Proximal Distilled Evolutionary Reinforcement Learning

Cristian Bodnar, Ben Day, Pietro Lió
Department of Computer Science & Technology
University of Cambridge
Cambridge, United Kingdom
cb2015@cam.ac.uk

Abstract

Reinforcement Learning (RL) has achieved impressive performance in many complex environments due to the integration with Deep Neural Networks (DNNs). At the same time, Genetic Algorithms (GAs), often seen as a competing approach to RL, had limited success in scaling up to the DNNs required to solve challenging tasks. Contrary to this dichotomic view, in the physical world, evolution and learning are complementary processes that continuously interact. The recently proposed Evolutionary Reinforcement Learning (ERL) framework has demonstrated mutual benefits to performance when combining the two methods. However, ERL has not fully addressed the scalability problem of GAs. In this paper, we show that this problem is rooted in an unfortunate combination of a simple genetic encoding for DNNs and the use of traditional biologically-inspired variation operators. When applied to these encodings, the standard operators are destructive and cause catastrophic forgetting of the traits the networks acquired. We propose a novel algorithm called Proximal Distilled Evolutionary Reinforcement Learning (PDERL) that is characterised by a hierarchical integration between evolution and learning. The main innovation of PDERL is the use of learning-based variation operators that compensate for the simplicity of the genetic representation. Unlike traditional operators, our proposals meet the functional requirements of variation operators when applied on directly-encoded DNNs. We evaluate PDERL in five robot locomotion settings from the OpenAI gym. Our method outperforms ERL, as well as two state-of-the-art RL algorithms, PPO and TD3, in all tested environments.

Introduction

The field of Reinforcement Learning (RL) has recently achieved great success by producing artificial agents that can master the game of Go (Silver et al. 2017), play Atari games (Mnih et al. 2015) or control robots to perform complex tasks such as grasping objects (Andrychowicz et al. 2017) or running (Lillicrap et al. 2015). Most of this success is caused by the combination of RL with Deep Learning (Goodfellow, Bengio, and Courville 2016), generically called Deep Reinforcement Learning (DRL).

At the same time, Genetic Algorithms (GAs), usually seen as a competing approach to RL, have achieved limited suc-

cess in evolving DNN-based control policies for complex environments. Though previous work has shown GAs to be competitive with other DRL algorithms in discrete environments (Such et al. 2017), they are still significantly less sample efficient than a simple method like Deep Q-Learning (Mnih et al. 2015). Moreover, in complex robotic environments with large continuous state and action spaces, where environment interactions are costly, their sample inefficiency is even more acute (Khadka and Tumer 2018), (Such et al. 2017).

However, in the physical world, evolution and learning interact in subtle ways. Perhaps, the most famous product of this interaction is the Baldwin effect (Simpson 1953), which explains how the genotype can assimilate learnt behaviours over the course of many generations. A more spectacular byproduct of this interplay, which has received more attention in recent years, is the epigenetic inheritance of learnt traits (Dias and Ressler 2013).

Despite these exciting intricacies of learning and evolution, the two have almost always received separate treatment in the field of AI. Though they have been analysed together in computational simulations multiple times (Hinton and Nowlan 1987), (Ackley and Littman 1992), (Suzuki and Arita 2004), they have rarely been combined to produce novel algorithms with direct applicability. This is surprising given that nature has always been a great source of inspiration for AI (Floreano and Mattiussi 2008).

For the first time, Khadka and Tumer (2018) have recently demonstrated on robot locomotion tasks the practical benefits of merging the two approaches in their Evolutionary Reinforcement Learning (ERL) framework. ERL uses an RL-based agent alongside a genetically evolved population, with a transfer of information between the two. However, ERL has not fully addressed the scalability problem of GAs. While the gradient information from the RL agent can significantly speed up the evolutionary search, the population of ERL is evolved using traditional variation operators. Paired with directly encoded DNNs, which is the most common genetic representation in use, we show that these operators are destructive.

This paper brings the following contributions:

- Demonstrates the negative side-effects in RL of the traditional genetic operators when applied to directly encoded DNNs.

- Proposes two novel genetic operators based on backpropagation. These operators do not cause catastrophic forgetting in combination with simple DNN representations.

- Integrates these operators as part of a novel framework called Proximal Distilled Evolutionary Reinforcement Learning (PDERL) that uses a hierarchy of interactions between evolution and learning.

- Shows that PDERL outperforms ERL, PPO (Schulman et al. 2017) and TD3 (Fujimoto, van Hoof, and Meger 2018) in five robot locomotion environments from the OpenAI gym (Brockman et al. 2016).

Background

This section introduces the Evolutionary Reinforcement Learning (ERL) algorithm and the genetic operators it uses.

Evolutionary Reinforcement Learning

The proposed methods build upon the ERL framework introduced by Khadka and Tumer (2018). In this framework, a population of policies is evolved using GAs. The fitness of the policies in the population is based on the cumulative total reward obtained over a given number of evaluation rounds. Alongside the population, an actor-critic agent based on DDPG (Lillicrap et al. 2015) is trained via RL. The RL agent and the population synchronise periodically to establish a bidirectional transfer of information.

The first type of synchronisation in ERL, from the RL agent to the genetic population, is meant to speed up the evolutionary search process. This synchronisation step clones the actor of the RL agent into the population every few generations to transfer the policy gradient information. The synchronisation period, ω, is a hyperparameter that controls the rate of information flowing from the RL agent to the population.

The second type of synchronisation consists of a reverse information flow coming from the population to the RL agent. The actors in the population collect experiences from which the RL agent can learn off-policy. All the transitions coming from rollouts in the population are added to the replay buffer of the DDPG agent. The population experiences can be seen as being generated by evolution-guided parameter space noise (Plappert et al. 2018).

Genetic encoding and variation operators

The policies in the ERL population are represented by neural networks with a direct encoding. In this common genetic representation, the weights of a network are recorded as a list of real numbers. The ordering of the list is arbitrary but consistent across the population. As we will show, applying the usual biologically inspired variation operators on this representation can produce destructive behaviour modifications.

In the physical world, mutations and crossovers rarely have catastrophic phenotypic effects because the phenotype is protected by the complex layers of physical, biological and chemical processes that translate the DNA. In a direct genetic encoding, the protective layers of translation are absent because the representation is so simple and immediate.

As such, the biologically inspired variation operators commonly found in the literature, including ERL, do not have the desired functionality when paired with a direct encoding. Ideally, crossovers should combine the best behaviours of the two parents. At the same time, mutations should produce only a slight variation in the behaviour of the parent, ensuring that the offspring inherits it to a significant extent. However, because DNNs are sensitive to small modifications of the weights (the genes in a direct encoding), these operators typically cause catastrophic forgetting of the parental behaviours.

ERL evolves the population using two variation operators commonly used for list-based representations: n-point crossovers and Gaussian mutations (Eiben and Smith 2015, p. 49-79). n-point crossovers produce an offspring policy by randomly exchanging segments of the lists of weights belonging to the two parents, where n endpoints determine the segments. ERL uses a version of the operator where the unit-segments are rows of the dense layer matrices, ensuring that an offspring receives nodes as they appear in the parents rather than splicing the weights of nodes together. The resulting child policy matrices contain a mix of rows (nodes) coming from the matrices (layers) of both parents. This is intended to produce functional consistency across generations.

However, the lack of an inherent node ordering in DNNs means that hidden representations need not be consistent over the population and as such the input to a node may not be consistent from parent to offspring, creating the possibility for destructive interference. This can cause the child policy to diverge from that of the parents, as we will demonstrate. Similarly, the damaging effects of adding Gaussian noise to the parameters of a DNN have been discussed at great length by Lehman et al. (2018). A common approach to containing these issues, employed by ERL, is to mutate only a fraction of the weights. Nevertheless, these mutations are still destructive. Furthermore, evolving only a small number of weights can slow down the evolutionary search for better policies.

Method

This section introduces our proposed learning-based genetic operators and describes how they are integrated with ERL.

The genetic memory

A significant problem of the population from ERL is that it does not directly exploit the individual experiences collected by the actors in the population. The population only benefits indirectly, through the RL agent, which uses them to learn and improve. The individual experiences of the agents are an essential aspect of the new operators we introduce in the next sections and, therefore, the agents also need a place to store them.

The first modification we make to ERL is to equip the members of the population, and the RL agent, with a small personal replay buffer containing their most recent experiences, at the expense of a marginally increased memory footprint. Depending on its capacity κ, the buffer can also include experiences of their ancestors. Because the transitions

in the buffer can span over multiple generations, we refer to this personal replay buffer of each agent as the *genetic memory*. When the policies interact with the environment, they not only store their experiences in DDPG's replay buffer as in ERL but also in their *genetic memory*.

The ancestral experiences in the genetic memory are introduced through the variation operators. A mutated child policy inherits the genetic memory of the parent entirely. During crossover, the buffer is only partially inherited. The crossover offspring fills its buffer with the most recent half of transitions coming from each of the two parents' genetic memories.

Q-filtered distillation crossovers

In this section, we propose a *Q-filtered behaviour distillation crossover* that selectively merges the behaviour of two parent policies into a child policy. Unlike n-point crossovers, this operator acts in the phenotype space, and not in parameter space.
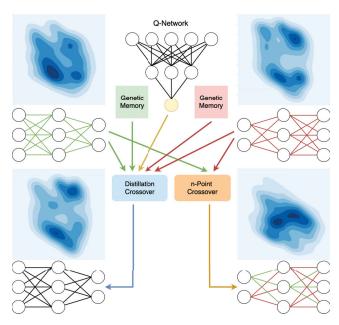

Figure 1: Q-filtered distillation crossover compared to n-point crossover. The contour plots represent the state visitation distributions of the agents (i.e. the average amount of time the agent spends in each state) for the first two state dimensions of an environment. The plot is generated by fitting a Gaussian kernel density model over the states collected over many episodes. These distributions show how Q-filtered distillation crossover selectively merges the behaviours of the two parents by inheriting from the shapes of both parent distributions. In contrast, the modes of the state visitation distribution obtained by the traditional crossover are mostly disjoint from the modes of the parent distributions.

For a pair of parent actors from the population, the crossover operation works as follows. A new agent with an initially empty associated genetic memory is created. The

genetic memory is filled in equal proportions with the latest transitions coming from the genetic memories of the two parents. The child agent is then trained via a form of Imitation Learning (Osa et al. 2018) to selectively imitate the actions the parents would take in the states from the newly created genetic memory. Equivalently, this process can be seen as a more general type of policy distillation (Rusu et al. 2016) process since it aims to "distil" the behaviour of the parents into the child policy.

Algorithm 1: Distillation Crossover

Input : Parent policies μ_x, μ_y with memory R_x, R_y
Output: Child policy μ_z with memory empty R_z
1 Add latest $\frac{\kappa}{2}$ transitions from R_x to R_z
2 Add latest $\frac{\kappa}{2}$ transitions from R_y to R_z
3 Shuffle the transitions in R_z
4 Randomly initialise the weights $\boldsymbol{\theta}_z$ of μ_z with the weights of one of the parents
5 **for** $e \leftarrow 1$ **to** *epochs* **do**
6   **for** $i \leftarrow 1$ **to** κ/N_C **do**
7   Sample state batch of size N_C from R_z
8   Optimise $\boldsymbol{\theta}_z$ to minimise $L(C)$ using SGD
9   **end**
10 **end**

Unlike the conventional policy distillation proposed by Rusu et al. (2016), two parent networks are involved, not one. This introduces the problem of divergent behaviours. The two parent policies can take radically different actions in identical or similar states. The problem is how the child policy should decide whom to imitate in each state. The key observation of the proposed method is that the critic of the RL agent already knows the values of certain states and actions. Therefore, it can be used to select which actions should be followed in a principled and globally consistent manner. We propose the following Q-filtered behaviour cloning loss to train the child policy:

$$
\begin{aligned}
L(C) = &\sum_{i}^{N_C} \|\mu_z(s_i) - \mu_x(s_i)\|^2 \, \mathbb{I}_{Q(s_i, \mu_x(s_i)) > Q(s_i, \mu_y(s_i))} \\
+ &\sum_{j}^{N_C} \|\mu_z(s_j) - \mu_y(s_j)\|^2 \, \mathbb{I}_{Q(s_j, \mu_y(s_j)) > Q(s_j, \mu_x(s_j))} \\
+ &\frac{1}{N_C} \sum_{k}^{N_C} \|\mu_z(s_k)\|^2,
\end{aligned}
$$

where the sum is taken over a batch of size N_C sampled from the genetic memories of the two parent agents. μ_x and μ_y represent the deterministic parent policies, while μ_z is the deterministic policy of the child agent.

The indicator function $\mathbb{I}$ uses the Q-Network of the RL agent to decide which parent takes the best action in each state. The child policy is trained to imitate those actions by minimising the first two terms. The final term is an L_2 regularisation that prevents the outputs from saturating

the hyperbolic tangent activation. Figure 1 contains a diagram comparing this new crossover with the ERL n-point crossover. We refer to ERL with the distillation crossover as Distilled Evolutionary Reinforcement Learning (DERL).

We note that while this operator is indeed more computationally intensive, a small number of training epochs over the relatively small genetic memory suffices. Additionally, we expect a distributed implementation of our method to compensate for the incurred wall clock time penalties. We leave this endeavour for future work.

Parent selection mechanism

An interesting question is how parents should be selected for this crossover. A general approach is to define a mating score function $m : \Pi \times \Pi \to \mathbb{R}$ that takes as input two policies and provides a score. The pairs with higher scores are more likely to be selected. Similarly to Gangwani and Peng (2018), we distinguish two ways of computing the score: greedy and distance-based.

Greedy The score $m(\mu_x, \mu_y) = f(\mu_x) + f(\mu_y)$ can be greedily determined by the sum of the fitness of the two parents. This type of selection generally increases the stability of the population and makes it unlikely that good individuals are not selected.

Distance based The score $m(\mu_x, \mu_y) = d_\Pi(\mu_x, \mu_y)$ can be computed using a distance metric in the space of all possible policies. "Different" policies are more likely to be selected for mating. The exact notion of "different" depends on the precise form of the distance metric d_Π. Here, we propose a distance metric in the behaviour space of the two policies that takes the form:

$$d_\Pi(\mu_x, \mu_y) = \mathbb{E}_{\mathbf{x} \sim \rho_x}[\|\mu_x(\mathbf{x}) - \mu_y(\mathbf{x})\|^2]$$
$$+ \mathbb{E}_{\mathbf{x} \sim \rho_y}[\|\mu_x(\mathbf{x}) - \mu_y(\mathbf{x})\|^2],$$

where ρ_x and ρ_y are the state-visitation distributions of the two agents.

This distance metric measures the expected difference in the actions taken by the two parent policies over states coming from a mixture of their state visitation distributions. This expectation is in practice stochastically approximated by sampling a large batch from the genetic memories of the two agents. This strategy biases the introduction of novel behaviours into the population at the expense of stability as the probability that fit individuals are not selected is increased.

Proximal mutations

As showed by Lehman et al. (2018), Gaussian mutations can have catastrophic consequences on the behaviour of an agent. In fact, the stability of the policy update is a problem even for gradient descent approaches, where an inappropriate step size can have unpredictable consequences in the performance landscape. Methods like PPO (Schulman et al. 2017) are remarkably stable by minimising an auxiliary KL divergence term that keeps the behaviour of the new policy close to the old one.

Based on these motivations, we integrate the safe mutation operator SM-G-SUM that has been proposed by
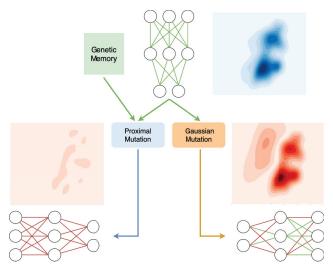

Figure 2: Proximal mutations compared to Gaussian mutations. The blue contour plot shows the state visitation distribution of the parent policy. The red contour plots show the difference between the distribution of the children and that of the parent. The difference plots are generated by taking the normalised difference between the parent and child probability densities. The behaviour of the policy obtained by proximal mutation is a small perturbative adjustment to the parent behaviour. In contrast, the traditional mutation produces a divergent behaviour, even though it modifies only a fraction of the weights (shown in red).

Algorithm 2: Proximal Mutation

    Input : Parent policy μ_x with memory R_x
    Output: Child policy μ_y with memory R_y
1 Initialise $R_y \leftarrow R_x$ and $\mu_y \leftarrow \mu_x$
2 Sample state batch of size N_M from R_x
3 Compute $\boldsymbol{s}$ on the batch samples s_i as in Equation 1.
4 Mutate $\boldsymbol{\theta}_y \leftarrow \boldsymbol{\theta}_y + \frac{\mathbf{x}}{\boldsymbol{s}}, \mathbf{x} \sim \mathcal{N}(\mathbf{0}, \sigma \boldsymbol{I})$

Lehman et al. (2018) with the genetic memory of the population. This operator uses the gradient of each dimension of the output action over a batch of N_M transitions from the genetic memory to compute the sensitivity $\boldsymbol{s}$ of the actions to weight perturbations:

$$\boldsymbol{s} = \sqrt{\sum_k^{|\mathcal{A}|} \Big(\sum_i^{N_M} \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s_i)_k \Big)^2} \tag{1}$$

The sensitivity is then used to scale the Gaussian perturbation of each weight accordingly by $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \frac{\mathbf{x}}{\boldsymbol{s}}$, with $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \sigma \boldsymbol{I})$, where σ is a mutation magnitude hyperparameter. The resulting operator produces child policies that are in the *proximity* of their parent's behaviour. Therefore, we refer to this operator as a *proximal mutation* (Figure 2), and the version of ERL using it as Proximal Evolutionary Reinforcement Learning (PERL).

While the proximal mutations do not explicitly use learning, they rely on the capacity of the policies to learn, or in

other words, to be differentiable. Without this property, these behaviour sensitivities to the parameter perturbations cannot be computed analytically.

Integration

The full benefits of the newly introduced operators are realised when they are used together. The Q-filtered distillation crossover increases the stability of the population and drives the agents towards regions of the state-action space with higher Q-values. The proximal mutations improve the exploration of the population and its ability to discover better policies. As will be seen in the evaluation section, the operators complement each other. We refer to their dual integration with ERL as Proximal Distilled Evolutionary Reinforcement Learning (PDERL).
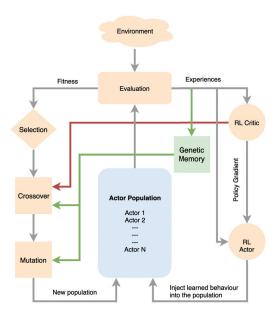

Figure 3: A high-level view of PDERL. The new components and interactions are drawn in green and red. In PDERL, there is a higher flow of information from the individual experiences and learning (right) to the population (left) than in ERL.

Ultimately, PDERL contains a hierarchy of interactions between learning and evolution. A high-level interaction is realised through the information exchange between the population and the RL agent. The newly introduced operators add a lower layer of interaction, at the level of the genetic operators. A diagram of PDERL is given by Figure 3.

Evaluation

This section evaluates the performance of the proposed methods, and also takes a closer look at the behaviour of the proposed operators.

Experimental setup

The architecture of the policy and critic networks is identical to ERL. Those hyperparameters that are shared with ERL have the same values as those reported by Khadka and Tumer (2018), with a few exceptions. For Walker2D, the synchronisation rate ω was decreased from 10 to 1 to allow a higher information flow from the RL agent to the population. In the same environment, the number of evaluations ξ was increased from 3 to 5 because of the high total reward variance across episodes. Finally, the fraction of elites in the Hopper and Ant environments was reduced from 0.3 to 0.2. Generally, a higher number of elites increases the stability of the population, but the stability gained through the new operators makes higher values of this parameter unnecessary.

For the PDERL specific hyperparameters, we performed little tuning due to the limited computational resources. In what follows we report the chosen values alongside the values that were considered. The crossover and mutation batch sizes are $N_C = 128$ and $N_M = 256$ (searched over $64, 128, 256$). The genetic memory has a capacity of $\kappa = 8k$ transitions ($2k, 4k, 8k, 10k$). The learning rate for the distillation crossover is 10^{-3} ($10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$), and the child policy is trained for 12 epochs ($4, 8, 12, 16$) . All the training procedures use the Adam optimiser. Greedy parent selection is used unless otherwise indicated. As in ERL, the population is formed of $k = 10$ actors.

When reporting the results, we use the official implementations for ERL[1] and TD3[2], and the OpenAI Baselines[3] implementation for PPO. Our code is publicly available at https://github.com/crisbodnar/pderl.

Performance evaluation

This section evaluates the mean reward obtained by the newly proposed methods as a function of the number of environment frames experienced. The results are reported across five random seeds. Figure 4 shows the mean reward and the standard deviation obtained by all algorithms on five MuJoCo (Todorov, Erez, and Tassa 2012) environments.

While PERL and DERL bring improvements across multiple environments, they do not perform well across all of them. PERL is effective in stable environments like HalfCheetah and Hopper, where the total reward has low variance over multiple rollouts. At the same time, DERL is more useful in unstable environments like Walker2d and Ant since it drives the population towards regions with higher Q values. In contrast, PDERL performs consistently well across all the settings, demonstrating that the newly introduced operators are complementary. PDERL significantly outperforms ERL and PPO across all environments and, despite being generally less sample efficient than TD3, it catches up eventually. Ultimately, PDERL significantly outperforms TD3 on Swimmer, HalfCheetah and Ant, and marginally on Hopper and Walker2d.

Table 1 reports the final reward statistics for all the tested models and environments. Side by side videos of ERL and PDERL running on simulated robots can be found at https://youtu.be/7OGDom1y2YM. The following subsections take

---

[1]https://github.com/ShawK91/erl_paper_nips18
[2]https://github.com/sfujim/TD3
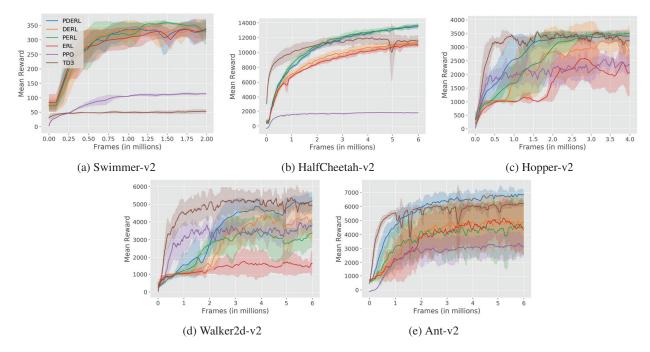[3]https://github.com/openai/baselines/

Figure 4: The mean reward obtained on Swimmer (a), HalfCheetah (b), Hopper (c), Walker2d (d) and Ant (e). PDERL outperform ERL, PPO and TD3 on all the environments.

a closer look at the newly introduced operators and offer a justification for the improvements achieved by PDERL.

Environment	Metric	TD3	PPO	ERL	PERL	DERL	PDERL
Swimmer	Mean	53	113	334	327	333	**337**
	Std.	26	3	20	26	6	**12**
	Median	51	114	346	354	338	**348**
HalfCheetah	Mean	11534	1810	10963	**13668**	11362	13522
	Std.	713	28	225	**236**	358	287
	Median	11334	1810	11025	**13625**	11609	13553
Hopper	Mean	3231	2348	2049	**3497**	2869	3397
	Std.	213	342	841	**63**	920	202
	Median	3282	2484	1807	**3501**	3446	3400
Walker2D	Mean	4925	3816	1666	3364	4050	**5184**
	Std.	476	413	737	818	1170	**477**
	Median	5190	3636	1384	3804	4491	**5333**
Ant	Mean	6212	3151	4330	4528	4911	**6845**
	Std.	216	686	1806	2003	1920	**407**
	Median	6121	3337	5164	3331	5693	**6948**

Table 1: Final performance in all environments. The result with the highest mean is shown in bold. PERL marginally outperforms PDERL on two environments, but PDERL consistently performs well across all environments.

Crossover evaluation

A good indicator for the quality of a crossover operator is the fitness of the offspring compared to that of the parents. Figure 5 plots this metric for ten randomly chosen pairs of parents in the Ant environment. Each group of bars gives the fitness of the two parents and the policies obtained by the two types of crossovers. All these values are normalised by the fitness of the first parent. The performance of the child
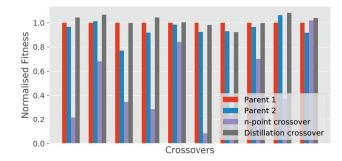

Figure 5: Normalised crossover performance on the Ant environment. The distillation crossover achieves higher fitness than the n-point crossover. Fitness is relative to Parent 1 in each group.

obtained via an n-point crossover regularly falls below 40% the fitness of the best parent. At the same time, the fitness of the policies obtained by distillation is generally at least as good as that of the parents.

The state visitation distributions of the parents and children offer a clearer picture of the two operators. Figure 6 shows these distributions for a sample crossover in the Ant environment. The n-point crossover produces a behaviour that diverges from that of the parents. In contrast, the Q-filtered distillation crossover generates a policy whose behaviour contains the best traits of the parent behaviours. The new operator implicitly drives each new generation in the population towards regions with higher Q values.
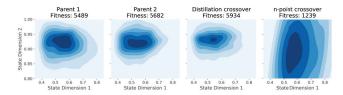
Figure 6: This figure shows the state visitation distributions for the distillation crossover and the n-point crossover. Unlike the n-point crossover, the distillation crossover produces policies that selectively merge the behaviour of the parents.
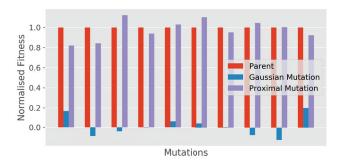
Mutation evaluation

Figure 7: Normalised mutation performance on the Ant environment. The proximal mutations obtain significantly higher fitness than the Gaussian mutations. Fitness is relative to the Parent in each group.

Figure 7 shows the fitness of the children obtained by the two types of mutation for ten randomly selected parents on the Ant environment. Most Gaussian mutations produce child policies with fitness that is either negative or close to zero. At the same time, the proximal mutations create individuals that often surpass the fitness of the parents.
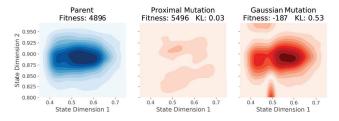

Figure 8: As before, the blue contours represent the state visitation distribution of the parent, whereas the red ones represent the difference. The child obtained by proximal mutation inherits the behaviour of the parent to a large degree and obtains a 600 fitness boost. The behaviour obtained by Gaussian mutation is entirely different from that of the parent. The KL divergence between the parent and child distributions (0.03 and 0.53) quantitatively confirm this.

As in the previous section, the analysis of the state visitation distribution of the policies reveals the destructive be-

haviour of the Gaussian mutations. The contours of these distributions for a sample mutation are given in Figure 8. The policy mutated by additive Gaussian noise completely diverges from the behaviour of the parent. This sudden change in behaviour causes catastrophic forgetting, and the new offspring falls in performance to a total reward of -187. In contrast, the proximal mutation generates only a subtle change in the state visitation distribution. The offspring thus obtained inherits to a great extent the behaviour of the parent, and achieves a significantly higher total reward of 5496.

Related work

This paper is part of an emerging direction of research attempting to merge Evolutionary Algorithms and Deep Reinforcement Learning: Khadka and Tumer (2018), Pourchot and Sigaud (2019), Gangwani and Peng (2018), Khadka et al. (2019).

Most closely related are the papers of Lehman et al. (2018) and Gangwani and Peng (2018). Both of these works address the destructive behaviours of classic variation operators. Lehman et al. (2018) focus exclusively on safe mutations, and one of their proposed operators is directly employed in the proximal mutations. However, their paper is lacking a treatment of crossovers and the integration with learning explored here. The methods of Gangwani and Peng (2018) are focused exclusively on safe operators for stochastic policies, while the methods proposed in this work can be applied to stochastic and deterministic policies alike. The closest aspect of their work is that they also introduce a crossover operator with the goal of merging the behaviour of two agents. Their solution reduces the problem to the traditional single parent distillation problem using a maximum-likelihood approach to combine the behaviours of the two parents. They also propose a mutation operator based on gradient ascent using policy gradient methods. However, this deprives their method of the benefits of derivative-free optimisation such as the robustness to local optima.

Discussion

The ERL framework demonstrates that genetic algorithms can be scaled to DNNs when combined with learning methods. In this paper we have proposed the PDERL extension and shown that performance is further improved with a hierarchical integration of learning and evolution. While maintaining a bi-directional flow of information between the population and RL agent, our method also uses learning within the genetic operators which, unlike traditional implementations, produce the desired functionality when applied to directly encoded DNNs. Finally, we show that PDERL outperforms ERL, PPO and TD3 in all tested environments.

Many exciting directions for future research remain, as discussed in the text. An immediate extension would be to develop a distributed version able to exploit larger and more diverse populations. Better management of the inherited genetic memories may yield efficiency gains by prioritising key experiences. Lastly, we note the potential for using learning algorithms at the level of selection operators.

References

Ackley, D., and Littman, M. 1992. Interactions between learning and evolution. In Langton, C. G.; Taylor, C.; Farmer, C. D.; and S., R., eds., *Artificial Life II, SFI Studies in the Sciences of Complexity*, volume X. Reading, MA, USA: Addison-Wesley. 487–509.

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, P.; and Zaremba, W. 2017. Hindsight experience replay. In *NIPS*.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym.

Dias, B. G., and Ressler, K. J. 2013. Parental olfactory experience influences behavior and neural structure in subsequent generations. *Nature Neuroscience* 17:89 EP –.

Eiben, A. E., and Smith, J. E. 2015. *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd edition.

Floreano, D., and Mattiussi, C. 2008. *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. The MIT Press.

Fujimoto, S.; van Hoof, H.; and Meger, D. 2018. Addressing function approximation error in actor-critic methods. In *ICML*.

Gangwani, T., and Peng, J. 2018. Policy optimization by genetic distillation. In *ICLR*.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Hinton, G. E., and Nowlan, S. J. 1987. How learning can guide evolution. *Complex Systems* 1.

Khadka, S., and Tumer, K. 2018. Evolution-guided policy gradient in reinforcement learning. In *NeurIPS*.

Khadka, S.; Majumdar, S.; Nassar, T.; Dwiel, Z.; Tumer, E.; Miret, S.; Liu, Y.; and Tumer, K. 2019. Collaborative evolutionary reinforcement learning. In Chaudhuri, K., and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 3341–3350. Long Beach, California, USA: PMLR.

Lehman, J.; Chen, J.; Clune, J.; and Stanley, K. O. 2018. Safe mutations for deep and recurrent neural networks through output gradients. In *GECCO*.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518:529–533.

Osa, T.; Pajarinen, J.; Neumann, G.; Bagnell, J.; Abbeel, P.; and Peters, J. 2018. An algorithmic perspective on imitation learning. *Foundations and Trends in Robotics* 7(1-2):1–179.

Plappert, M.; Houthooft, R.; Dhariwal, P.; Sidor, S.; Chen, R. Y.; Chen, X.; Asfour, T.; Abbeel, P.; and Andrychowicz, M. 2018. Parameter space noise for exploration. *CoRR* abs/1706.01905.

Pourchot, A., and Sigaud, O. 2019. Cem-rl: Combining evolutionary and gradient-based methods for policy search. *CoRR* abs/1810.01222.

Rusu, A. A.; Colmenarejo, S. G.; Çaglar Gülçehre; Desjardins, G.; Kirkpatrick, J.; Pascanu, R.; Mnih, V.; Kavukcuoglu, K.; and Hadsell, R. 2016. Policy distillation. *CoRR* abs/1511.06295.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *CoRR* abs/1707.06347.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L. R.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T. P.; Hui, F. F. C.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the game of go without human knowledge. *Nature* 550:354–359.

Simpson, G. G. 1953. The baldwin effect. *Evolution* 7(2):110–117.

Such, F. P.; Madhavan, V.; Conti, E.; Lehman, J.; Stanley, K. O.; and Clune, J. 2017. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *ArXiv* abs/1712.06567.

Suzuki, R., and Arita, T. 2004. Interactions between learning and evolution: the outstanding strategy generated by the baldwin effect. *Bio Systems* 77 1-3:57–71.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033.