EDM-DRL: Toward Stable Reinforcement Learning through Ensembled Directed Mutation

Michael H. Prince[1,2], Andrew J. McGehee[1,3], and Daniel R. Tauritz[1,4]

[1] BioAI Research Group, Auburn University, Auburn AL 36830, USA
[2] mhp0009@auburn.edu
[3] ajm0045@auburn.edu
[4] dtauritz@acm.org

Abstract. Deep reinforcement learning (DRL) has experienced tremendous growth in the past few years. However, training stability of agents continues to be an open research question. Here, the authors present Ensembled Directed Mutation of Deep Reinforcement Learning (EDM-DRL) - a hybridization of evolutionary computing (EC), ensemble learning, and DRL methods as a means of mitigating training instability in DRL agents. We show that our method trains more consistently than synchronous Advantage Actor Critic (A2C). We also show that by employing our novel mutation and ensemble methods, performance of DRL agents can be improved during test time without sacrificing training stability. Further, though a similar number of time steps are used, we show that the EDM-DRL algorithm uses a mere 1% or less of the network parameter updates used in A2C. Finally, we conduct an ablation study to identify components within the EDM-DRL algorithm responsible for highest contribution. Code and experimental logs are available at: https://github.com/Linked-Liszt/EDM-DRL

Keywords: Deep Reinforcement Learning · Evolutionary Algorithms · Ensembles · Game Playing

1 Introduction

Deep reinforcement learning (DRL) has shown significant improvement over the last several years, proving its ability to competently navigate complex, high dimensional problem spaces [1]. Despite the progress, the training stability of RL agents still presents unique challenges for the field. Without the ability to train RL agents in a stable manner, the technology can only be realized *in silico* as it is too unsafe or unreliable to be applied in practical domains. Fortunately, several methods such as double Q-learning [26], prioritized experience replay [19], dueling networks [27], distributional RL [2], asynchronous advantage actor-critic (A3C) [15], multi-step bootstrapping [23] [24], and Rainbow [11], which combines several of the previous, have been developed to address the issue of training instability. Though these methods have demonstrated state of the art, reproducible results, DRL training stability remains highly sensitive to choices of hyper-parameters, random seeds, and reward function design.

Drawing inspiration from human intelligence, the only existence proof for general intelligence, the authors hypothesize that the combination of evolution and learning may improve the state of DRL stability. Within the field of evolutionary computing (EC), several hybrid methods have been developed in order to combine the unique benefits of evolution and learning. Further, since RL problems typically require both exploration of a space and optimization of the expected reward within that space, they can certainly benefit from exploration methods which are not gradient-based, as in evolutionary algorithms (EAs). In addition, the population-based approach of EAs may offer stability to RL algorithms since, with an ensemble, various members may be able to cover each others' weaknesses at inference time. Therefore, this work explores the following research question:

How does the hybridization of EAs and DRL affect the training stability of DRL agents?

With this abstract research question in mind, the authors engage the following sub-topics in particular:

- Methods of integrating EAs and DRL which include both mutation and learning
- Comparison of ensemble methods for action selection in EDM-DRL
- Analysis of the stability of EDM-DRL as compared to its ablated components
- Analysis of stability of EDM-DRL as compared to an A2C baseline.

2 Background

RL is an area of machine learning which places agents inside of environments. These agents learn to maximize a reward signal by interacting with the environment. This work focuses on a subset of RL, referred to as DRL, which integrates deep learning and neural networks into the RL paradigm. RL and DRL have both shown much promise when applied to challenging problems in manufacturing [14] and finance [5]. DRL, in particular, has recently shown outstanding success, capable of exceeding human-level performance in the game of Go [22].

There have been significant efforts and contributions to improving the stability of RL agents during training in recent decades. Perhaps the earliest example comes from Sutton et al. in their introductory works to RL in 1988 and 1998 [23] [24]. These works introduced critical concepts to the field of RL like temporal difference (TD) learning and multi-step bootstrapping for reward targets. Both have been shown to provide stability to agents under certain learning and policy conditions. The interested reader may find the more recent work by Sutton et al. an informative review on stability conditions for various forms of TD [25].

Another notable improvement, inspired by concepts in neuroscience, was the development of an experience replay mechanism in 2013 by Mnih et al. [16]. Prior to this work, DRL was unable to learn control policies directly from high

dimensional sensory input such as the raw RGB pixels of an image. The addition of an experience replay buffer allows agents to uniform randomly sample and learn from experiences within a predefined history size, smoothing the reward distributions over previous behaviors.

An improvement to this work was introduced in 2015 by Schaul et al. in which the sampling of previous experiences is prioritized proportional to the TD error δ, which roughly denotes how "surprising" an action is to the agent [19]. The probability of sampling a transition is defined in Eq. 1, where $p_i > 0$ is the priority of the transition given by $|\delta_i|$, k is the transition history size, and the exponent α controls the intensity of prioritization (with $\alpha = 0$ corresponding to uniform random sampling).

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{1}$$

Both the introduction of experience replay and prioritized experience replay have been shown to contribute significantly to the stability of RL agents during training, especially those learning from noisy reward signals. A further contribution made in 2015 by Mnih et al. was the introduction of a dual architecture (not to be confused with double Q-learning or dueling networks) where the "target" weights of a network are only periodically updated from the "online" weights, effectively avoiding overfitting to an unstable reward target [17].

Drawing inspiration from a single network with multiple streams, Wang et al. introduced the dueling network architecture in 2015 [27]. In the dueling architecture, the two streams which are aggregated together are the value stream and the advantage stream, with the prior estimating the quality (Q) of a particular state and the latter estimating the advantage gain of a particular state-dependent action. This combination of differing goals was shown to moderately improve the stability during optimization of the network.

Further improvements were introduced with the innovation of double Q-learning, first proposed in 2010 and generalized for DRL models in 2016 by van Hasselt et al. [9,26]. Double Q-learning addresses the issue of overly optimistic value estimation in Q-learning models by decoupling the evaluation and selection of actions. Two Q functions Q^A and Q^B are used to evaluate the quality of state-action pairs. However, instead of updating Q^A with Eq. 2 as in traditional Q-learning, and without loss of generality, double Q-learning uses $Q^B(s', a*)$ where Q^B is updated on a different sample of experiences than Q^A.

$$Q^A(s', a^*) = \max_a \; Q^A(s', a) \tag{2}$$

Another significant addition to the stability of RL agents during training was the work by Mnih et al. in 2016 on A3C [15]. A3C is in some ways similar to other multiple stream approaches. The actor is a policy based mechanism and the critic is a value based mechanism. However, the value estimations from the critic are not aggregated with other estimations in any way (as in dueling networks) nor used to update the actor's policy (as in double Q-learning). Instead the policy is updated directly through a policy gradient which aims to maximize

the value estimations of the critic, which is itself independently learned. This has been shown to stabilize training of agents without the need for experience replay, allowing for on policy learning methods. In 2017, Wu et al. [28] discovered that a synchronous version of A3C, dubbed A2C, provided even more stable and efficient policy updates by waiting for all actors to finish episode roll-outs before performing the policy update. For this reason, A2C was selected as the DRL stability baseline.

More improvements to the performance of RL agents came from Bellemare et al. in their work on distributional RL in 2017 [2]. Distributional RL attempts to model expectation of reward as a discrete parameterized probability distribution. The work gives strong evidence of improvements to stability during training, especially in the case where reward signals do not follow stationary distributions. Another work in 2017 that is of note is the work by Fortunato et al. which introduced noisy networks [6]. While the work does not necessarily directly improve the stability of RL agents, it encourages exploration in tasks which require long chains of actions prior to the first reward signal without destabilizing the agent by adding intrinsic motivation signals.

In 2018, an intriguing paper by Hessel et al. was published in which several research scientists from DeepMind conducted an ablation study of several of the previously described contributions to RL [11]. Termed Rainbow, the combined algorithm was shown to be stable, generalizable, and sample efficient. Further, the work offers highly valuable insight into the individual contributions of each method to specific tasks within the Atari suite of games. An elegantly simple idea, the work strengthens the adage that the whole is greater than the sum of its parts.

Within the realm of EC, Covariance Matrix Adaptation Evolutionary Strategies (CMA-ES) has demonstrated state of the art results when optimizing functions with several local extrema and high multi-modality (e.g., shifted rastrigin) [8]. CMA-ES uses information derived from the population to dynamically adapt the distribution function used in mutation. By controlling both the step size and the "direction" of mutation, CMA-ES can more efficiently explore the problem space. Further, since CMA-ES is a population-based method with stochastic mutation, it may more easily escape deceptive gradients or suboptimal local extrema. This ability to navigate deceptive function landscapes may lend itself well to exploring highly noisy or deceptive reward landscapes in modern DRL problems.

Though the mentioned RL works all contribute significantly to the stability of RL agents, all are contributing ideas taken from mathematics, statistics, neuroscience, or deep learning. In comparison, relatively little research has been conducted on employing ideas from population dynamics, evolution, or EC to stabilize DRL training. Additionally little investigation has been done into ensembling populations in an RL setting. Some hybrid methods combining elements of EC and DRL exist [12], [18], [13]. However, such hybrids often merely implement existing techniques like conventional neuroevolution or Lamarckian and Baldwinian learning. Further, all the discussed related work focuses on the per-

formance of the agent rather than having training stability as the goal. This work addresses this deficit by employing a novel combination of some of the addressed techniques with evolutionary methods and analyzing the training stability of the agents as compared against the ablated components.

3 Training Procedure

Here we propose a close integration between EC and deep learning paradigms. Our method exploits beneficial techniques common to the deep learning world, namely a shared feature extractor, as well as novel exploration techniques drawn from EC. We also exploit population-based methods by ensembling all actors at inference time during testing. This learning method is detailed in Algorithm 1 with the next sections detailing its individual components and methods.

Algorithm 1 Learning Procedure

$Actors = \textsc{InitializeActors}()$
$Critic = \textsc{InitializeCritic}()$
$FeatureExtractor = \textsc{InitializeFE}()$
$IsUnsolved = True$
$Timesteps = 0$
while $Timesteps < MaxTimesteps$ **and** $IsUnsolved$ **do**
  $Experiences = []$
  $Fitnesses = []$
  for $i, Actor \in \textsc{Enumerate}(\textsc{Actors})$ **do**
    $Experiences[i],\ Fitnesses[i],\ NumSteps = \textsc{Environment}(Actor)$
    $Timesteps = Timesteps + NumSteps$
  end for
  $GradsFE,\ GradsActors,\ GradsCritic = \textsc{Backpropagation}(Experiences)$
  $FeatureExtractor = FeactureExtractor - GradsFE$
  $Critic = Critic - GradsCritic$
  $Actors = \textsc{Evolve}(Actors,\ GradsActors,\ Fitnesses)$
  $IsUnsolved = \textsc{TestEnsemble}(Actors,\ FeatureExtractor)$
end while

3.1 Controller Structure

Our novel, hybrid controller structure is a fully connected neural network composed of the previously mentioned feature extractor, a critic network, and a population of actor networks. In contrast to other EC and RL hybrid methods, the critic network is not separate from the actor network. The model takes in a state vector representing the environment at a given time which is passed to the feature extractor for learning abstract representations. The feature extractor is a set of shared layers within the critic neural network and *all* actor networks

within the population. As such, we explicitly show the separation of the critic, the feature extractor, actors, and their respective gradients in Algorithm 1. The representations learned by these layers are then passed to the population of value based actor networks, which estimate the expected reward for a given state-action pair, and the policy based critic network, which assesses the quality of the state. Finally, data from all actors, even agents which do not survive to the next generation, and data from the critic network are back-propagated through the respective networks and the shared feature extractor.

In the backward pass of the model, each actor of the generation is given a single episode to sample the environment. All the experiences are batched into a singular network update, similar to A2C. During the update, the advantage stream, or the difference between the observed value and the critic value, is back-propagated through the value network. Then, using the TD error, the critic network is updated to better predict future transitions. The accumulated gradients are then back-propagated through the feature extractor. Entropy regularization [15] is applied to encourage convergence over time.

3.2 Ensembling

Since the EDM-DRL algorithm makes use of a population of actors, it's possible to ensemble them together during test time. This ensemble is applied on line 14 of Algorithm 1. We try several ensembling methods which are influenced by the fitness of the actors. Here we define fitness as the sum of the rewards in a given episode. The first and simplest method is to allow the 1-elite actor - the member that performed best during its training episode - to select all actions during evaluation. Intuitively, this makes the assumption that reward is a near perfect representation of true performance. Alternatively, we consider modeling fitness as a probability mass function across the population by applying a softmax. The resulting action is chosen proportional to the fitness probability masses. This is a weaker assumption than the previous method, but the fittest individuals still generally choose the actions. Finally, the third method we consider is a weighted voting mechanism. Actors are assigned a number of votes proportional to their fitness. In this case, lower fitness individuals are able to overwhelm the decisions of more fit members if they are in agreement. A visualization of the weighted vote mechanism can be seen in Fig. 1.

3.3 Initialization and Termination

The parameters of every network layer, including the feature extractor, actors, and the critic, are initialized using Xavier initialization [7]. This is defined as $\mathcal{U}(-\sqrt{x}, \sqrt{x})$ where x is the number of input parameters. This type of initialization attempts to preserve the scale of the input and output activation between each layer. Each actor is initialized independently in order to encourage diversity in the population.

The algorithm is terminated on one of two conditions: solving the environment or 100k time steps, whichever is first. In the environment implementations

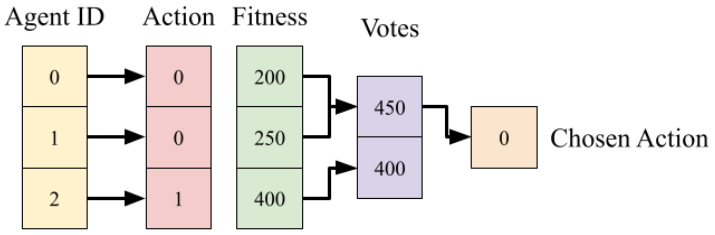

Fig. 1. The "weighted vote" mechanism. In a discrete environment, each actor selects an action. Their finesseses are accumulated. The action with the most votes is taken.

used in this work, the OpenAI gyms [4], each environment has different conditions to determine when it is "solved." In order to ensure robustness, a typical definition of "solved" is achieving a given mean reward μ across k episodes. If 100k time steps pass, the algorithm is terminated and the run is marked as unsolved. This is intentionally a significantly longer evaluation time than is needed to solve the environment as we wish to analyze failure and abnormal cases.

4 Evolution Procedure

The evolution procedure, stated in line 16 of Algorithm 1, creates the next generation of actor networks. With access to the gradients produced by backpropagation, we present a hybrid mutation method and a novel method, coined "repopulation", which replaces traditional parent and survivor selection. The base EDM-DRL method employs mutation as its sole variation operator. In a separate ablation, we experiment with adding a recombination method. In general, all the described methods try to combine and capitalize on the benefits of EC and gradient decent. They tend to be more greedy than typical EC methods through the use of gradients, but less greedy than typical DRL approaches through the use of stochastic methods and population dynamics.

Algorithm 2 Evolution Procedure

1: $Population.sort(Fitnesses)$
2: $NewPopulation = []$
3: $NewPopulation.append(Population[0])$
4: **for** $ParentIdx$, $ChildCount \in Enumerate(Repopulation[])$ **do**
5: **for** $i \in Range(ChildCount)$ **do**
6: $Child = DirectedMutation(Population[ParentIdx], Grads[ParentIdx])$
7: $NewPopulation.append(Child)$
8: **end for**
9: **end for**

4.1 Repopulation

We introduce a novel method, called repopulation, to create the next generation, replacing the canonical parent selection and survival selection methods. Capitalizing on the accuracy of the gradient step, we chose to systemically mutate the best performing population members. In each generation, parents produce a pre-defined number of offspring. For example, with repopulation parameters of $[p_1, p_2, \ldots, p_m]$ with the sum of p_1 through p_m equal to μ, the fittest parent produces p_1 offspring, the next produces p_2, and so on. This method, as opposed to a more traditional mating and survival selection, will cause a more greedy selection of parents which reproduce when weighted heavily on the first few members. We also add the option to guarantee *1-elite*, which is beneficial in the case that a generation G_{t+1} performs worse than generation G_t, as the 1-elite is neither mutated nor updated via the gradients. However, the 1-elite individual is re-evaluated with every new generation of offspring. This system creates a generational (μ, λ) population management model where the population size will remain static since $\mu = \lambda$. The choice of how many offspring each rank of parent is allowed to produce was hand-tuned in order to heavily bias offspring toward the most elite parents.

4.2 Mutation

Since we have access to the gradients produced by back-propagation, the mutation operator can be augmented with this information. We will call this "directed mutation." In contrast to random mutation, directed mutation is intentionally skewed to follow the previous gradient step. For each parameter in the mutated layers, a random learning rate is chosen from a defined interval, and Gaussian noise is applied to the gradient step. In addition, the size of the mutation is scaled to the size of the gradient step. In order to encourage convergence, this learning rate is multiplied by a decay rate at the end of every generation. Drawing inspiration from CMA-ES, this approach leverages the strengths of both gradients and population based methods, albeit differently. Instead of using the covariance of a population, the mutation distribution is informed by the analytical gradient. Further, rather than disallowing large gradient steps, as in trust region policy optimization and proximal policy optimization methods, we accomplish similar stability during exploration by relying on the diversity of exploration within a population [20,21]. In other words, individuals may explore unsafe regions of the reward space to their detriment, yet other members of the population may remain intact. A pseudo-code implementation of directed mutation is included in Algorithm 3. For a full visual representation, see Fig. 2.

4.3 Recombination Ablation

We also experiment with two forms of parameter-space recombination in a separate ablation, with some modification of the repopulation system. For both types of recombination, two parents are selected, and used to create a single

Algorithm 3 Directed Mutation

$lr = \mathcal{U}(lr\_low,\ lr\_high)$
for $param,\ grad \in \{Parameters,\ Gradients\}$ **do**
  $\mu = param - grad$
  $\sigma = mutation\_scale \cdot |param - \mu|$
  $param = x \sim \mathcal{N}(\mu, \sigma)$
end for

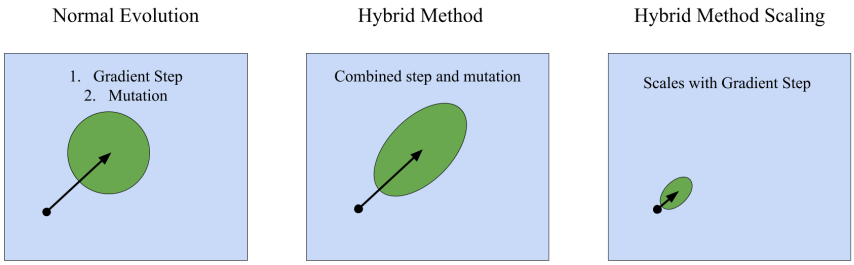

Fig. 2. Left: conventional mutation - parameters are moved randomly following a gradient step; Middle: hybrid "directed" mutation - the mutation is skewed to follow the learning step; Right: hybrid "scaled" mutation - same as the directed method but scaled to a percentage of the gradient step's magnitude.

child. The first method averages parameters from the two parent networks. The second form of recombination randomly selects individual parameters from the parents to build a full parameter set. In both cases, the parents' gradients are applied before the parameters are crossed over.

In the recombination experiments, a child actor is created through either recombination or mutation. Like the repopulation method, we greedily select the best performing members for use in recombination. In the configuration [α mutation, β (γ) recombination], the first α members are created by mutation from the best fitness member. The next β members are created through combinations of the top γ population members, denoted in parenthesis. Finally, similar to the base repopulation method, the elite member is held for the next generation.

5 Experimental Setup

This work conducts three experiments. The first experiment compares the EDM-DRL algorithm against an established baseline, the second analyzes the benefits of ensembling during test time, and the final offers an ablation study of common combinations of mutation and recombination. All experiments share the following general setup.

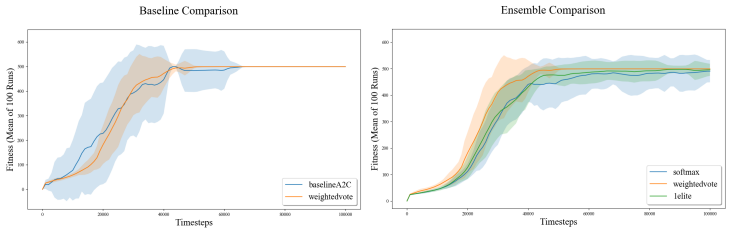

Fig. 3. On the left, EDM-DRL compared to the baseline A2C. On the right, a comparison of ensemble methods against the 1-elite method.

5.1 Training Environment

We use the "Cartpole-V1" environment, a well known and maturely tested environment from OpenAI gyms [4]. OpenAI gyms provide a common API and stable environments to help promote rapid development and fair comparison for RL algorithms. The goal of the cartpole environment is to balance a hinged pole affixed to a cart which may undergo 2D changes in motion. The agent is allowed to push the cart one unit left or right in a given time step. The agent receives a reward of positive one for each time step that the pole remains "balanced" and the cart remains within the frame. Here, "balanced" is defined as remaining within fifteen degrees of vertical center. The environment is initialized randomly with a slight tilt on the pole and offset on the cart. Version 1 of cartpole is the more challenging version as the agent must keep the pole upright and the cart on screen for 500 time steps. As mentioned previously, the definition of "solved" for this particular environment is to achieve a mean score of 475 across 100 episodes. Though this environment is certainly a toy problem, easily solved by linear, random search algorithms, it stands as a fair comparison for various agents and controllers as evidenced by its use throughout the field[1][10].

5.2 Measuring Performance and Stability

For each run, we attempt to sample the environment every 2000 time steps during training. Here a sample is defined as the mean performance of a particular agent over 100 episodes. In the A2C baseline, we are able to directly sample scores every 2000 time steps. However, due to the requirement of completing an episode prior to fitness evaluation, we are only able to sample individuals' performances following each generation. To directly and fairly compare the progress of each run at a set point, we use linear interpolation between surrounding sample points.

For each configuration, 30 unseeded runs are conducted. This gives us enough samples to compare the variation in the learning curves of each class of agent. During each run, we track the following metrics: number of runs solved in under 100k time steps, time steps until solved μ, time steps until solved σ, generations until solved μ, generations until solved σ, and the cross-run reward σ. Since this work focuses primarily on training stability, the cross-run reward standard

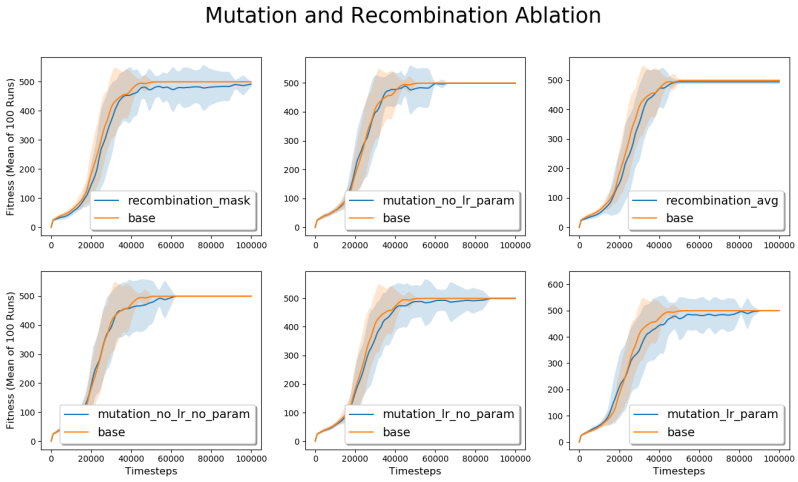

Fig. 4. Ablation of mutation and recombination methods. For readability, each ablation is shown against a base algorithm with no mutation or recombination.

deviation is of particular interest. Visually, the mean performance corresponds to the plotted line and the standard deviation corresponds to the shaded regions in Fig. 3 and Fig. 4. In addition to this variation, we also consider the standard deviation in time steps used to solve the environment as related to the agent's stability.

Since each of the configurations contain 30 runs, they can be statistically compared. In order to gauge performance, we compare time steps used to solve the environment for each run. Then we conduct an F-test to determine if the variance of each of the sets may be assumed to be equal. The logs were manually examined and were determined to follow a normal distribution. Following this, we perform a two-tailed T-test assuming the appropriate relation of variances, equal or unequal. We use a confidence level of 0.95 to determine if the differences in means are significant.

Strategy parameters are tuned using a combination of random search and hand-tuning. The parameters used in the following experiments are listed in Table 1. For each configuration, all strategy parameters remain static except for the one being ablated. For the A2C baseline, we used the hyper-parameters suggested in their respective works.

5.3 Experiments

Our algorithm draws inspiration from A2C, so it is a logical choice for benchmarking. The baseline is compared against the best found configuration of the EDM-DRL algorithm.

Then, we examine the benefits gained from allowing an ensemble approach for action selection during evaluation. We compared the softmax and weighted

Table 1. EDM-DRL Strategy Parameters

Parameter	Value
Base Parameters	
Population Size	10
Repopulation	[5,3,1] (+1 elite)
Evolutionary Learning Rate	5e-5
Evolutionary Learning Rate Decay	0.98 (per generation)
Optimizer	ADAM
Optimizer Learning Rate	3e-2
Value Coefficient	0.5
Entropy Coefficient	0.05
Shared Layers	1 (size=128)
Actor Layers	2 (size=128)
Value Layers	1 (size=128)
Activations	ReLU
Mutation Ablation Parameters	
Learning Rate (high bound, low bound)	[4e-5, 9e-5]
Mutation Scale	0.1
Recombination Ablation Parameters	
Repopulation	[5 mutation , 4 (3) recomb] (+1 elite)

vote ensemble methods against a more traditional 1-elite approach. All employ their respective best found configuration.

Finally, we conduct an ablation study concerning two critical sections of the mutation and recombination algorithms. The added elements include: random selection of learning rates, application of Gaussian noise to gradient steps, mean recombination, and masking recombination. Each element is added in isolation.

6 Results & Discussion

Here we highlight the results from each of the discussed experiments. The source code and raw data for all experiments is included as supplementary material.[5] For each of the experiments we cover statistical results and graph relevant comparisons. Table 2 compares data from all of the experiments.

6.1 Comparison to A2C

As shown through statistical analysis, the A2C baseline was able to achieve solutions faster than the best configuration of our algorithm ($p = 0.0015$). However, the EDM-DRL algorithm is far more stable in training in general, as shown by the tighter variance, represented by the shaded areas in Fig. 3.

[5] https://github.com/Linked-Liszt/EDM-DRL

Table 2. The observed results of the experiments

Algorithm	Runs	Training σ	Gen.s μ	Gen.s σ	Frames μ	Frames σ
baseline_A2C	**30/30**	119.7	N/A	N/A	**24,560**	10,402
weighted_vote	**30/30**	52.7	**62**	11.23	32,287	**6,627**
1elite	**30/30**	**40.4**	67	**7.9**	44,962	10,135
softmax	29/30	60.5	73	15	44,700	13,115
Ablations						
mutation_lr_param	30/30	66.5	62	12.5	37,560	15,384
mutation_lr_no_param	30/30	**57.1**	65.7	14.9	33,376	6,831
mutation_no_lr_param	30/30	57.3	62	15.6	**31,694**	**6,066**
mutation_none	30/30	62.9	**60**	**10.1**	33,635	9,594
mean_mating	30/30	76.6	67.9	11.0	34,093	7,134
mask_mating	30/30	61.6	67.93	13.6	36,525	15,067

Best results in each category are bolded

6.2 Ensemble Comparison

We are able to show that ensembling is an effective strategy for action selection. The weighted voting mechanism in particular outperforms the non-ensembling 1-elite strategy by a statistically significant margin ($p = 5.3e{-}7$) in frames used to solve the environment. This can be clearly seen in the steeper slope and greater final value of the time steps until solved μ metric, which is visualized in Fig. 3. However, the softmax ensemble method proved to be neither significantly better nor worse than the 1-elite strategy. Though these results are surprising, we speculate that the success of the weighted vote mechanism is due to the natural tendencies of a population to regress to the mean. While weaker individuals alone are less fit than a single strong individual, the combined strength of the population is able to compensate when they are in agreement.

6.3 Ablation Study

Our ablation study shows that mutation and recombination, in the current forms proposed in this work, had little positive effect on the overall performance or stability. Statistically, none of the ablated agents were better or worse than the base agent by a noteworthy margin. In particular, the mean recombination strategy is remarkably similar to the unmodified base agent. So, it stands to reason that this approach would be the least beneficial to training stability. We suspect that mutation and recombination actually injected further noise into the learning environment while not significantly destabilizing the agent. A pair-wise comparison of each ablation against an unmodified base agent is laid out in Fig. 4.

7 Conclusion

The authors have shown that using evolutionary methods may be used to create more stable DRL agents. While the ablated mutation and recombination elements of the EDM-DRL algorithm yield subpar performance, the population-based approach and repopulation mechanism detailed in this work seem to greatly benefit the stability of the EDM-DRL agent. In particular, this improvement is evidenced by the tighter standard deviation of rewards received across 30 unique runs as compared with the A2C baseline.

We observe that our algorithm is able to solve the CartPole environment using one parameter update for every 100 (up to 500 in some cases) parameter updates used in the A2C baseline. In A2C, the algorithm updates the network parameters at every time step. In contrast, EDM-DRL updates parameters only once per generation. We further demonstrate the benefits of having an ensemble of co-trained population members. The weighted voting mechanism is able to consistently perform statistically better than the 1-elite member of the population. In conventional DRL, ensembling methods are typically not practical due to the sampling inefficiency of each individual. This method mitigates such inefficiencies.

The stability of DRL agents during training remains a challenging research area for the AI community. Combined efforts over years of work have produced impressive engineering solutions which have incrementally yet profoundly improved this issue. Though in its early stages, the EDM-DRL algorithm stands to further improve upon the stability of DRL agents by hybridizing previous contributions with ideas drawn from EC.

8 Future Work

While we have interesting early results, this by no means is a comprehensive examination of the benefits that EC may offer DRL agents in terms of training stability. For example, further research may make use of more complex neural network components or architectures. The current implementation is built entirely from fully connected layers. In complex environments, particularly those which rely on raw RGB pixel input, convolutional layers would likely serve as more effective feature extractors. Memory-based units like recurrent neural networks and long-short term memory layers may also be candidates for hybridizing EC and DRL for time-series problems.

Some of the next logical steps for this work would be to explore the more complex gym environments such as the Arcade Learning Environment [3]. Further, we have observed that the discussed EDM-DRL algorithm uses significantly fewer parameter updates (by orders of magnitude) than the A2C baseline. Interesting future work may investigate whether the population based approach enables the EDM-DRL algorithm to take modified gradient steps which act as shortcuts along a purely gradient-based optimization path way.

To this end, an interesting area of research that is highly relevant to DRL is the investigation of unique exploration methods. Since DRL relies heavily

on exploring the environment to gather data concerning the reward landscape, novelty search is a well suited contribution from EC which may benefit DRL agents. Particularly, the authors suggest that research into the effects novelty search may have on training stability, as opposed to performance, is worth future investigation.

Acknowledgment

The authors thank GitHub user jankrepl whose repository contained an implementation and hyper-parameter set of A2C which was critical to the development of this work. Further, the authors thank the students of Dr. Tauritz' research methods class, who served roles in reviewing and critiquing our methods.

References

1. Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: Deep Reinforcement Learning: A Brief Survey. IEEE Signal Processing Magazine **34**(6), 26–38 (2017). https://doi.org/10.1109/MSP.2017.2743240
2. Bellemare, M.G., Dabney, W., Munos, R.: A Distributional Perspective on Reinforcement Learning. In: International Conference on Machine Learning. pp. 449–458. PMLR (2017)
3. Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The Arcade Learning Environment: An Evaluation Platform for General Agents. Journal of Artificial Intelligence Research **47**, 253–279 (2013). https://doi.org/10.1613/jair.3912
4. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym. arXiv preprint arXiv:1606.01540 (2016)
5. Deng, Y., Bao, F., Kong, Y., Ren, Z., Dai, Q.: Deep Direct Reinforcement Learning for Financial Signal Representation and Trading. IEEE transactions on neural networks and learning systems **28**(3), 653–664 (2016). https://doi.org/10.1109/TNNLS.2016.2522401
6. Fortunato, M., Azar, M.G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al.: Noisy Networks for Exploration. arXiv preprint arXiv:1706.10295 (2017)
7. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. pp. 249–256 (2010)
8. Hansen, N., Ostermeier, A.: Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation. In: Proceedings of IEEE international conference on evolutionary computation. pp. 312–317. IEEE (1996). https://doi.org/10.1109/ICEC.1996.542381
9. Hasselt, H.V.: Double Q-learning. In: Advances in Neural Information Processing Systems (NIPS). pp. 2613–2621 (2010)
10. Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D.: Deep Reinforcement Learning that Matters. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 32 (2018)
11. Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D.: Rainbow: Combining Improvements in Deep Reinforcement Learning. In: Thirty-Second AAAI Conference on Artificial Intelligence. pp. 3215–3222 (2018)

12. Khadka, S., Majumdar, S., Nassar, T., Dwiel, Z., Tumer, E., Miret, S., Liu, Y., Tumer, K.: Collaborative Evolutionary Reinforcement Learning. In: International Conference on Machine Learning. pp. 3341–3350. PMLR (2019)
13. Khadka, S., Tumer, K.: Evolution-Guided Policy Gradient in Reinforcement Learning. In: Advances in Neural Information Processing Systems (NIPS). pp. 1188–1200 (2018)
14. Li, Y., Fadda, E., Manerba, D., Tadei, R., Terzo, O.: Reinforcement Learning Algorithms for Online Single-Machine Scheduling. In: 15th Conference on Computer Science and Information Systems (FedCSIS 2020). pp. 277–283. IEEE (2020)
15. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous Methods for Deep Reinforcement Learning. In: International conference on machine learning. pp. 1928–1937 (2016)
16. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with Deep Reinforcement Learning. arXiv preprint arXiv:1312.5602 (2013)
17. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236
18. Pourchot, A., Sigaud, O.: CEM-RL: Combining Evolutionary and Gradient-Based Methods for Policy Search. arXiv preprint arXiv:1810.01222 (2018)
19. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized Experience Replay. arXiv preprint arXiv:1511.05952 (2015)
20. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust Region Policy Optimization. In: International conference on machine learning. pp. 1889–1897 (2015)
21. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347 (2017)
22. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016). https://doi.org/10.1038/nature16961
23. Sutton, R.S.: Learning to Predict by the Methods of Temporal Differences. Machine Learning **3**(1), 9–44 (1988)
24. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
25. Sutton, R.S., Mahmood, A.R., White, M.: An Emphatic Approach to the Problem of Off-policy Temporal-Difference Learning. The Journal of Machine Learning Research **17**(1), 2603–2631 (2016)
26. Van Hasselt, H., Guez, A., Silver, D.: Deep Reinforcement Learning with Double Q-Learning. In: Thirtieth AAAI conference on artificial intelligence (2016)
27. Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., De Freitas, N.: Dueling Network Architectures for Deep Reinforcement Learning. arXiv preprint arXiv:1511.06581 (2015)
28. Wu, Y., Mansimov, E., Grosse, R.B., Liao, S., Ba, J.: Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. In: Advances in neural information processing systems (NIPS). pp. 5279–5288 (2017)