



GENEREIT: generating multi-talented reinforcement learning agents

Aristotelis Lazaridis¹ · Ioannis Vlahavas¹

Received: 7 July 2022 / Accepted: 7 December 2022
© The Author(s) 2023

Abstract Creating an intelligent system that is able to generalize and reach human or above-human performance in a variety of tasks will be part of the crowning achievement of Artificial General Intelligence. However, even though many steps have been taken towards this direction, they have critical shortcomings that prevent the research community from drawing a clear path towards that goal, such as limited learning capacity of a model, sample-inefficiency or low overall performance. In this paper, we propose GENEREIT, a meta-Reinforcement Learning model in which a single Deep Reinforcement Learning agent (meta-learner) is able to produce high-performance agents (inner-learners) for solving different environments under a single training session, in a sample-efficient way, as shown by primary results in a set of various toy-like environments. This is partially due to the fixed subset selection strategy implementation that allows the meta-learner to focus on tuning specific traits of the generated agents rather than tuning them completely. This, combined with our system’s modular design for introducing higher levels in the meta-learning hierarchy, can also be potentially immune to catastrophic forgetting and provide ample learning capacity.

Keywords Deep reinforcement learning · Meta-learning · Neural networks · Games

1 Introduction

Reinforcement Learning is based on the concept of learning through rewards [1], which allows a system to develop a behavior that can solve tasks in optimal ways. It gained popularity when the research community implemented a Deep Q-Network (DQN) [2], the first stable Deep Reinforcement Learning model that combined the traditional Q-Learning technique with deep neural networks in a stable manner, and achieved human or superhuman performance when trained in several Atari games separately [3], followed by its application in various other domains [4–7]. Even though many improved models followed this success [8–10], most Deep RL models share a common limitation, which is the inability to perform well in multiple environments.

This deficiency is not related to a single component of the algorithms, but rather on a set of circumstances that are usually present when training a generalized model [11]. However, the more sophisticated issues that need to be addressed before diving into the lower-level technicalities are a model’s limited learning capacity, which is the model’s learning capacity (i.e., the maximum amount of “knowledge” or “skills” it can learn), sample inefficiency and low overall performance [12].

In this paper, we propose GENEREIT, which is a significant extension of the REIN-2 system [13] that can generalize and solve multiple tasks at once, with only a small trade-off in performance and sample-efficiency than what would be achieved by using the original REIN-2 model to solve a single environment. REIN-2 is an end-to-end meta-learning Deep RL system that tackles efficiently the aforementioned issues by learning how to generate high-performance and sample-efficient Deep RL agents that require no training on their own, also incorporating a Randomized Batch Vector (RBV) strategy.

✉ Aristotelis Lazaridis
arislaza@csd.auth.gr

¹ School of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece

Our approach is based on the assumption that the meta-learner of a REIN-2 model always has the same straightforward task of generating high-performance Deep RL agents, regardless of the different types of environments that these agents might have to solve; the only difference in this case is that the meta-learner has to distinguish the different environments itself. Subsequently, the learning capacity of the meta-learner can be used to a greater potential, thus can learn easier how to distinguish the different types of environments that the inner-learners have to solve.

The contribution of our paper is the introduction of a new method for creating generalized Deep RL agents that is based on a highly sample-efficient technique. Our proposed system has a high degree of extensibility, making it suitable for designing novel generalized systems.

A video demonstration of how GENEREIT works at a conceptual level has been prepared as well, and is available on YouTube¹ (unlisted video).

In the rest of the paper, we discuss related works and compare them with the nature of GENEREIT (Sect. 2), then we proceed with describing the problem formulation, system design and implementation (Sect. 3), and present the conducted experiments and results that indicate its performance (Sect. 4). The discussion of these results is given in Sect. 5, the conclusions we reach are given in Sect. 6 and the future scope is given in Sect. 7.

2 Related work

Generalization using Deep RL techniques proved to be a strenuous topic but with significant interest by the research community. In most cases, a generalization framework is used to host a Deep RL algorithm in order to improve its ability to generalize [12]. One notable example in this category is PopArt [14, 15], in which the authors focused on addressing the problem of having to learn different magnitudes of returns across different environments, as this is one component in Deep RL algorithms that leads to inductive bias [11]. Originally, the authors used IMPALA [16] as core Deep RL algorithm for PopArt, and experiments indicated the significant ability of the model to achieve satisfying performance when evaluated in all Atari games. Even though this is a highly-encouraging attempt at multi-tasking, we followed a different approach based on meta-learning that addresses a higher-level issue of generalization instead, which is the model's learning capacity, i.e., the maximum amount of "knowledge" or "skills" it can learn.

Our work shares many similarities with PathNet [17], where the authors proposed using a meta-learning

methodology based on transfer learning, and more particularly exploring different network architectures and finding an optimal one using Reinforcement Learning. However, there are two key differences of PathNet with GENEREIT: first, we propose Reinforcement Learning agents using their default network architecture instead of modifying it, and then incorporate an RBV strategy to select a subset of the network that will be used for solving the task directly. Secondly, our meta-learning scheme is based on an end-to-end Reinforcement Learning approach, i.e., we use a Deep RL formulation for solving the task and for the meta-learning mechanism, whereas in PathNet a genetic algorithm is used for to evaluate the fitness of agents.

Our methodology can also be viewed as an approach of having multiple workers controlled by a single entity, with the flow of information being distributed appropriately to increase learning efficiency. Modern implementations of Asynchronous Actor-Critic (A2C) [9, 18] and Proximal Policy Optimization (PPO) [10] also utilize the concept of having multiple workers so as to solve a given environment, however they use all workers within a single environment to solve it, whereas GENEREIT uses only one worker per environment/problem, with a single entity learning to solve all environments simultaneously.

3 Methodology

We developed a system that uses the same meta-learning approach as REIN-2 but is able to generalize efficiently in multiple environments. REIN-2 uses a Deep RL agent (the *meta-learner*) that has the role of generating an instance of other Deep RL agents (*inner-learner*) that is required to solve a specific task (i.e., *inner-environment*). Only an inner-learner interacts with the inner-environment; the meta-learner receives only the average performance of an inner-learner instance, evaluates it, and then generates a different inner-learner that is potentially better at solving the same task, based on this evaluation, formulating this way a different RL problem (*outer-environment*). It should be highlighted that during this procedure, the inner-learner is not trained at any way; only the meta-learner is trained in order to produce increasingly better inner-learners.

In this paper, we propose the use of multiple inner-learners instead of a single one, each solving a different task. The concept of training is represented visually in Fig. 1 and the GENEREIT framework is described in more detail in Sect. 3.2. The environments to be solved, as well as the inner and outer learning algorithms to be used, are given as input arguments in the framework. We make use of the stable-baselines library [19] for running our experiments with the different RL algorithms and environments.

¹ <https://www.youtube.com/watch?v=5448N64CQJE>.

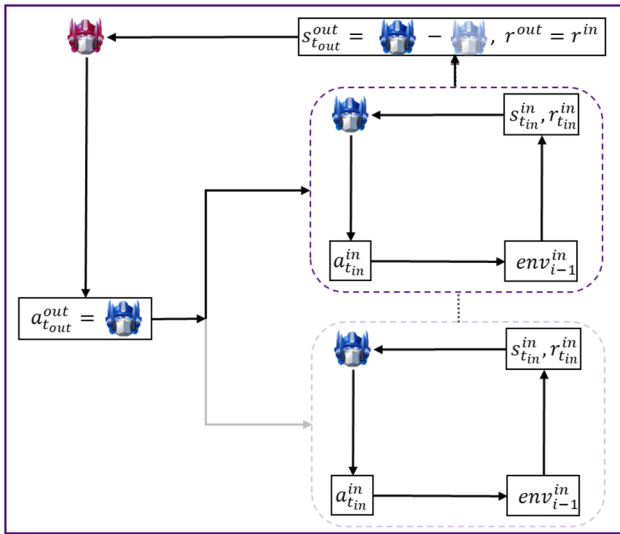




Fig. 1 The GENEREIT architecture. The  icon represents the meta-learner, while the  icon represents the inner-learners generated at every timestep t_{out}

In the next subsection we describe the problem formulation.

3.1 Problem formulation

Supposing there are n different inner-environments, let us define each inner-environment that is to be solved as P_i , which represents a Markov Decision Process (MDP) defined as (S_i, A_i, R_i, p_i) , where $i = 1, 2, \dots, n$ marks each different inner-environment. In this definition, S_i denotes the state space, A_i the action space, R_i the reward function, and p_i the probability transition function of each inner-environment i .

An inner-learner is a Reinforcement Learning model that is assigned to solve environment P_i , and is defined as $M_i(\theta_i)$, in which $\theta_i \in W^{k_i}$ denotes the set of parameters of the model, with W^{k_i} being a k_i -dimensional space of parameter values for the corresponding environment. For readability reasons, we shall denote the inner-learners as M_i .

The case of model M_i solving the problem P_i can be defined as optimizing objective J_i as follows:

$$\arg \max_{\theta_i} J_i(M_i(\theta_i), P_i) \tag{1}$$

Since the optimization problem that we address in this paper is the general case where all models M_i are assigned to solve their respective problems P_i , then our optimization problem is formulated as follows:

$$\begin{aligned} \arg \max_{\theta_1, \theta_2, \dots, \theta_n} & (J_1(M_1(\theta_1), P_1), J_2(M_2(\theta_2), P_2), \\ & \dots, J_n(M_n(\theta_n), P_n)) \end{aligned} \tag{2}$$

Additionally, let P_{out} denote the outer environment with which the meta-learner interacts. P_{out} is an MDP defined as the tuple $(S_{out}, A_{out}, R_{out}, p_{out})$, where $S_{out} = W^k$, $A_{out} = W^k$ and $R_{out} = R_{in}$ are the problem's state space, action space and reward function respectively, p_{out} denotes the probability transition function, and $k \leq \sum_i k_i$ is the dimension of the state/action spaces. The value of k depends on the strategy used during implementation and cannot exceed the total number of parameters of all inner-learners altogether. W^k can be thought of an expanded parameter space; in the general case where $k = \sum_i k_i$, each element $\theta \in W^k$ has dimension equal to the dimension of the concatenated elements $\theta_i \in W^{k_i}$, i.e., $\theta = (\theta_1, \theta_2, \dots, \theta_n)$, and when $k < \sum_i k_i$, then $\theta \in W^k$ represents a slice (i.e., subvector) of the concatenated parameter vector $(\theta_1, \theta_2, \dots, \theta_n)$, and in this case, the rest of the concatenated parameter vector is ignored during our proposed meta-learning process. Our strategy for selecting k is described in more detail in Subsection 3.2.

Let the meta-learner be a Reinforcement Learning model, represented as $M_{out}(\theta')$, where $\theta' \in H^l$ denotes the model's parameters, with H^l being an l -dimensional space of values for its parameters. Similarly as before, for purposes of simplification, we shall denote the meta-learner as M_{out} .

In each timestep, the meta-learner selects an action $a_{out} \in A_{out}$ through a function f in order to execute it in the outer environment, i.e., $f(M_{out}, P_{out}) = a_{out}$. We can divide this action vector into smaller pieces, i.e., $a_{out} = (a_1, a_2, \dots, a_n)$, where $a_i \in W^{k_i}$. For our purposes, let us assume that the action selection mechanism f in our problem can be modified to select each slice a_i of the whole action separately, as follows:

$$f = \{f_i \mid f_i(M_{out}, P_{out}) = a_i, \forall i = 1, 2, \dots, n\} \tag{3}$$

Now, since $A_{out} = W^k$, then $a_{out} \in W^k$ and $a_i \in W^{k_i}$, and since $\theta \in W^k$ and $\theta_i \in W^{k_i}$ as well, then action slices a_i can replace θ_i in the original optimization problem:

$$\begin{aligned} \arg \max_{\theta_1, \theta_2, \dots, \theta_n} & (J_1(M_1(\theta_1), P_1), J_2(M_2(\theta_2), P_2), \\ & \dots, J_n(M_n(\theta_n), P_n)) \\ = & \arg \max_{a_1, a_2, \dots, a_n} (J_1(M_1(a_1), P_1), J_2(M_2(a_2), P_2), \\ & \dots, J_n(M_n(a_n), P_n)) \\ = & \arg \max_{\theta'} (J_1(M_1(f_1(M_{out}(\theta'), P_{out})), P_1), \\ & \dots, J_n(M_n(f_n(M_{out}(\theta'), P_{out})), P_n)) \end{aligned} \tag{4}$$

Algorithm 1 GENEREIT Pseudocode

Input: Inner-learner model $M_i(\theta_i)$
 Meta-learner model $M_{out}(\theta')$
 Inner-environments $P_i : (S_i, A_i, R_i, p_i)$
 Outer-environment $P_{out} : (S_{out}, A_{out}, R_{out}, p_{out})$

Output: Trained meta-learner model

- 1: $i \leftarrow 1$
- 2: Initialize model parameters θ_i and θ'
- 3: **for** $t_{out} = 1:T_{out}$ **do**
- 4: $R_i = 0$
- 5: $a_{t_{out}} \leftarrow \text{GetActionMeta}(M_{out}(\theta'), s_{t_{out}}), s_{t_{out}} \in S_{out}$
- 6: $\theta_i \leftarrow a_t$
- 7: **for** $n = 1:N$ **do** ▷ Evaluate inner-learner in N episodes
- 8: $R_i \leftarrow \text{Evaluate}(M_i(\theta_i), P_i) + R_i$
- 9: **end for**
- 10: $R_{out} \leftarrow \frac{R_i}{N}$ ▷ Get inner-learner's average reward from N episodes
- 11: $\theta' \leftarrow \text{UpdateMeta}(M_{out}(\theta'), R_{out})$ ▷ Update meta-learner
- 12: $i \leftarrow (t_{out} \text{ div } k) + 1$ ▷ Proceed to next environment every k outer steps
- 13: **if** $0 \equiv t_{out} \text{ mod } k$ **then** ▷ Reset environment evaluation cycle
- 14: $i \leftarrow 1$
- 15: **end if**
- 16: **end for**

which is essentially the optimization of the meta-learner's parameters θ' :

$$\arg \max_{\theta'} J_i(M_i(f(M_{out}(\theta'), P_{out})), P_i) \quad (5)$$

Equation 4 shows how the original optimization objective of finding optimal parameters θ_i for each model M_i has been transformed into finding the optimal parameters θ' of the meta-learner M_{out} . Thus, the meta-learner is able to generate actions that correspond to optimized parameters θ of each inner-learner M_i that in turn solves each inner-environment.

3.2 GENEREIT system

The meta-learning concept of REIN-2 indicated that, instead of having a Deep RL agent learn how to behave optimally in an environment, a meta-learner can be trained to solve a different task, which is the generation of an optimal agent (inner-learner) that can in turn solve the original problem. By introducing more environments where the same type of inner-learner is used, the goal and complexity of the problem that the meta-learner has to solve remains the same, even though normally each inner-learner would have to develop a different type of optimal behavior.

The meta-learner is part of the top layer of our system, which is responsible for evaluating the performance scores of the different inner-learners, as well as generating new instances of them, in our case in the form of a set of neural network weights. This information transfer process is formulated as an RL problem, in which a conversion layer handles the proper processing of the contents sent by the inner-learners, and then forwards the action performed by

the meta-learner (i.e., the generation of new inner-learners) to the corresponding inner-environments.

The pseudo-algorithm of GENEREIT is presented in Algorithm 1. In each timestep in the outer environment, the meta-learner produces an inner-agent that corresponds to one of the inner-environments that are required to be solved. Each inner-learner is evaluated in that inner-environment, and its performance score defines the reward signal of the meta-learner. More precisely, the reward signal of an individual inner-learner during its evaluation process is defined as the average reward received from a number of episodes, so as to ensure that there is no other information leak related to the environment dynamics that could bias the meta-learner, as well as to avoid inaccuracies (e.g., a satisfying performance of an inner-learner that could be the result of stochasticity in the environment and not the inner-learner's skills).

The outer environment defines an action space for the meta-learner that depends on the type of the learning algorithm used for the inner-learners. An action of the meta-learner corresponds to an inner-agent, i.e., to the neural network values that define each that agent. In our approach, we used DQN [2] as the learning algorithm for all inner-learners in order to compare our findings with REIN-2. However, using the same model is not a restriction of our method, since with the appropriate configurations, different learning algorithms can be used for solving the different inner-environments.

The observation space used in the outer environment is similar as to that in the original REIN-2 implementation, which is defined as the space of differences between two consecutive inner-learners (i.e., the parameter space of the learning algorithm). In the generalized setting of

GENEREIT, this is expanded to include the differences of two consecutive inner-learners for all given environments.

Even though the Randomized Batch Vector (RBV) strategy introduced in REIN-2 can be used in our model so that only a fixed subset of an inner-learner's total network weights are tuned by the meta-learner (i.e., a fixed percentage of the total weights), we decided to randomly select a fixed (absolute) number of weights for all inner-learners instead. This choice was made in order to simplify the action/observation spaces. More particularly, if the RBV method was used, then the meta-learner would have to work with a different number of weights for each inner-learner due to their different network total sizes. In that case, the solution would be to concatenate each of these vectors into a long one that would constitute a complete and longer action/observation vector. This, however, would pose an extra difficulty to the meta-learner since he would be required to learn which parts of that vector correspond to the different inner-learners and environments. By selecting the same number of weights to tune for each inner-learner, no concatenation is necessary and the meta-learner can focus only on adjusting properly all available weights to generate an efficient inner-learner, regardless of the environment.

3.3 GENEREIT learner characteristics

The meta-learner perceives the state space $S \in \mathbb{R}^n$ from the outer environment, where n is the length of the vector that includes the network weight values of an inner-learner instance. In the more general case where a different learning algorithm is used for each inner-learner, n is the vector that contains the parameters of all algorithms. The action space of the meta-learner is defined as $A \in \mathbb{R}^n$, where an action vector contains the total of network weight values that are to be tuned and consequently realize an instance for each inner-learner.

Our framework works in a cyclic sequential fashion, which ultimately means that the meta-learner generates an inner-learner for a different inner-environment after k timesteps, where k is defined as a *meta-batch*, that is, the number of steps the meta-learner has to make in one environment before moving on to the next. This allows our model to be more robust than in the case of having all inner-learners deliver their results at once or having only one chance at interacting with one outer environment before moving to the next, since it improves its skills in noticing the specifics and dynamics of the given environment.

At timestep t , the meta-learner performs action \mathbf{a}_t that corresponds to an inner-learner that will be evaluated in the current inner-environment. Given l environments to be solved, at that same timestep the meta-learner observes state $s_t = \mathbf{a}_t - \mathbf{a}_{t-w}$, $w \in \{1, \dots, k\} \cup \{i \cdot k \mid i \in [1, l]\}$, that is, the

difference between the current and previous sets of weight values for that particular inner-learner.

The evaluation process consists of several runs of each inner-learner within its respective inner environment to get an average performance score. The number of runs is a hyperparameter that can be modified and acts as a trade-off between model accuracy and wall-clock time.

4 Experimental results

To evaluate our model, we tested its ability to generalize in a different number of environments. The sequential method of training the model in each environment allowed us to track separately the average rewards of the different inner-learners, and thus evaluate GENEREIT performance without using a separate evaluation process for each environment.

We compared our model directly with REIN-2, therefore we used the same architecture parameters and environments. More particularly, we selected the CartPole-v1 [20], Acrobot-v1 [21] and MountainCar-v0 [21] for the inner environments, and set A2C and PPO algorithms as the meta-learners, while DQN [2] was used as the algorithm for the inner-learners. Each experiment was compared against the performance of REIN-2 (PPO) and REIN-2 (A2C) in each respective environment. The sets of experiments conducted are presented in detail in Table 1.

At each outer timestep, the inner-learner interacted with the inner-environment for $N = 5$ episodes to extract an average performance score, and the number of weights that were tuned for the inner-learners was set to $n = 100$ (which is roughly the 1% of the total network weights for every inner-learner), in all experiments.

The meta-batch value for all experiments was set to $k = 16$. For lower values of k (e.g., 1), we noticed slight performance drops, and in particular, slower convergence speed in relation to peak performance. Also, for higher values of k , we did not notice and significant improvement in neither peak performance or convergence speed.

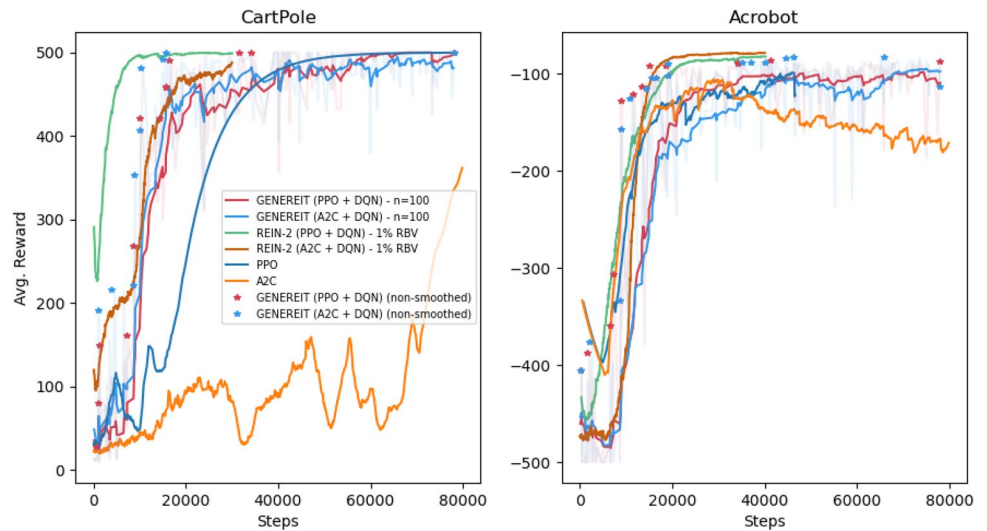
For all experiments, we performed a slight fine-tuning procedure for our model, and evaluated the average performance of the inner-learners per timestep of the meta-learner, per environment, against the reported results of REIN-2.

Additionally, we present the learning curves achieved by PPO and A2C in each environment during training, for comparison purposes. It should be noted that PPO and A2C were trained in each environment individually.

Initially, we trained our model to solve two environments (CartPole, Acrobot) simultaneously. Results indicate that not only generalization abilities are existent, but the sample-efficiency and high-performance skills from the

Table 1 Experiment configurations for evaluating GENEREIT

Set	Environments	Meta-learner	Inner-learner	Evaluation against
1	[CartPole-v1, Acrobot-v1]	PPO A2C	DQN	REIN-2 (PPO), REIN-2 (A2C)
2	[CartPole-v1, Acrobot-v1, MountainCar-v0]	PPO A2C	DQN	

Fig. 2 Performance of a single GENEREIT model trained on 2 environments (CartPole and Acrobot), compared to the performance of two REIN-2 models trained separately on the same environments

original REIN-2 model are maintained to a great degree (Fig. 2). It is also evident that both versions of GENEREIT achieve overall roughly equal or superior performance against PPO and A2C, which is another indicator of the models' property to maintain the strong performance even after being trained on generalization tasks.

Then we proceeded into feeding three environments (CartPole, Acrobot, MountainCar) to our system, which still managed to learn how to generate high-performance Deep RL agents (Fig. 3). It should be highlighted that neither convergence speed or peak performance was shriveled due to the higher number of environments that the meta-learner had to solve, compared to its performance in the single-environment case. This also applies when compared to PPO and A2C, which fail to learn faster than GENEREIT, even though they are trained on each environment individually.

5 Discussion

GENEREIT showed notable ability in learning to distinguish the different environments by simply using the average rewards received, as well as the differences between consecutive inner-learners that were generated. Therefore, the necessity to have a complete interaction with the task in hand was replaced with a method of using

compressed knowledge, subsequently allowing for greater learning capacity and avoiding issues such as catastrophic forgetting.

Experiments also showed that the tradeoff between performance of the REIN-2 implementation and our model's generalization skills is relatively low, but it could be potentially tackled with a slightly different state/action space representation, a different strategy for agent subset selection, or a modified data flow in relation to the meta-learning process, instead of the cyclic sequential method with meta-batches that we used.

The fact that GENEREIT managed to perform roughly as well as REIN-2 is highly encouraging, since it provides grounds for building a scaled-up generalization system that also has high performance. As such, there is plenty of space for further investigation with respect to the complete GENEREIT architecture, or specific configurations of it.

6 Conclusion

In this paper we presented an extension of the REIN-2 algorithm, namely GENEREIT, which allows the original methodology to be used in the multi-environment setting for the purpose of developing a generalized model. Our proposed architecture allows a Deep RL agent (the meta-learner) to generate other Deep RL agents (inner-learners) and assess

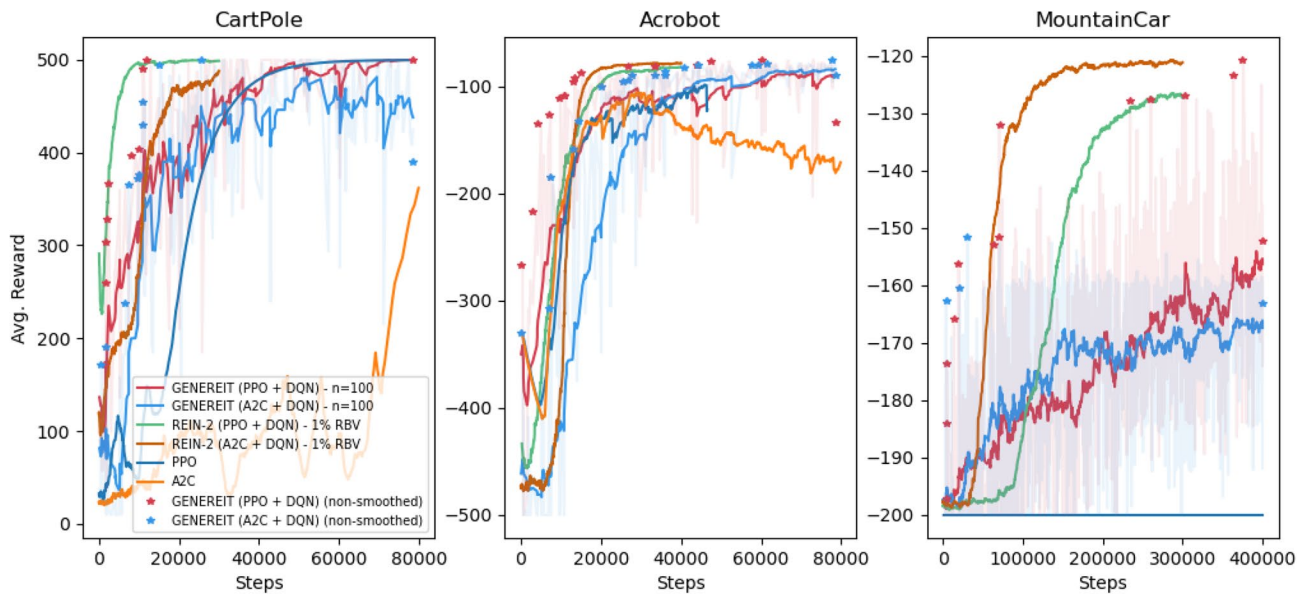


Fig. 3 Performance of a single GENEREIT model trained on 3 environments (CartPole, Acrobot and Mountain Car), compared to the performance of three REIN-2 models trained separately on the same environments. GENEREIT has a relatively small performance drop in

their performance in each given environment in a sequence. We trained and evaluated our model in two settings that included a different number of OpenAI Gym environments each time (2 and 3 environments), with the produced results being highly satisfactory. More particularly, after training, a single GENEREIT agent was able to perform optimally or near-optimally in all environments, while the performance drop in comparison to the original REIN-2 model was slight and provided the model with the required generalization skills. This was also evident when compared to PPO and A2C agents trained in each environment individually. Our proposed methodology is highly customizable and scalable, which is suitable for further experimentation in more complex environments as well.

7 Future scope

Our work allowed us to visit an unexplored space of meta-learning schemes. We implemented a promising model that can be further customized in various ways, such as by introducing more levels of hierarchy and integrating multiple meta-learners. This would potentially increase even more the learning capacity of the whole architecture and maintain the generalization abilities as well, plus allow learning in highly complex environments. Additionally, using a variety of learning algorithms for the different environments is an option that should provide more insights regarding the model's abilities.

terms of convergence speed to the optimal (or near-optimal) solutions as reached by REIN-2, which could be considered as the trade-off for its generalization abilities

Acknowledgements The authors would like to thank Christos Perchanidis for his help in preparing the video demonstration for the GENEREIT system.

Funding Open access funding provided by HEAL-Link Greece.

Declarations

Competing interests The authors have no competing interests to declare that are relevant to the content of this article.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Sutton RS, Barto AG (2018) Reinforcement learning: An introduction. MIT press, Cambridge
2. Mnih V et al (2015) Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533
3. Bellemare MG, Naddaf Y, Veness J, Bowling M (2013) The arcade learning environment: An evaluation platform for general agents. *J Artif Intell Res* 47:253–279

4. Rani G, Pandey U, Wagde AA, Dhaka VS (2022) A deep reinforcement learning technique for bug detection in video games. *Int J Info Technol* 1–13
5. Khurana S, Upadhayaya S (2020) Spectrum management in cognitive radio ad-hoc network using q-learning. *Int J Info Technol* 12(2):599–604
6. Rajyaguru V, Vithalani C, Thanki R (2020) A literature review: various learning techniques and its applications for eye disease identification using retinal images. *Int J Info Technol* 1–12
7. Saini M, Sharma K, Doriya R (2022) An empirical analysis of cloud based robotics: challenges and applications. *Int J Info Technol* 14(2):801–810
8. Hessel M et al. (2018) Rainbow: Combining improvements in deep reinforcement learning
9. Mnih V, et al. (2016) Asynchronous methods for deep reinforcement learning, 1928–1937. PMLR
10. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv preprint: [arXiv:1707.06347](https://arxiv.org/abs/1707.06347)
11. Hessel M, van Hasselt H, Modayil J, Silver D (2019) On inductive biases in deep reinforcement learning. arXiv preprint: [arXiv:1907.02908](https://arxiv.org/abs/1907.02908)
12. Lazaridis A, Fachantidis A, Vlahavas I (2020) Deep reinforcement learning: A state-of-the-art walkthrough. *J Artif Intell Res* 69:1421–1471
13. Lazaridis A, Vlahavas I (2022) Rein-2: Giving birth to prepared reinforcement learning agents using reinforcement learning agents. *Neurocomputing*
14. Hessel M et al. (2019) Multi-task deep reinforcement learning with popart vol. 33, 3796–3803
15. van Hasselt HP, Guez A, Hessel M, Mnih V, Silver D (2016) Learning values across many orders of magnitude. *Adv Neural Inf Process Syst* 29:4287–4295
16. Espeholt L et al. (2018) Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, 1407–1416 PMLR
17. Fernando C et al. (2017) Pathnet: Evolution channels gradient descent in super neural networks. arXiv preprint: [arXiv:1701.08734](https://arxiv.org/abs/1701.08734)
18. Bhatnagar S, Sutton RS, Ghavamzadeh M, Lee M (2009) Natural actor-critic algorithms. *Automatica* 45(11):2471–2482
19. Hill A (2018) et al. Stable baselines. github repository
20. Barto AG, Sutton RS, Anderson CW (1983) Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans Syst Man Cybernet* 5:834–846
21. Geramifard A, Dann C, Klein RH, Dabney W, How JP (2015) RLpy: a value-function-based reinforcement learning framework for education and research. *J Mach Learn Res* 16(1):1573–1578