A Study of the Impact of Evolutionary Strategies on Performance of Reinforcement Learning Autonomous Agents

Ventseslav Shopov, Vanya Markova

Institute of Robotics
Bulgarian Academy of Sciences
Bulgaria
Email: vkshopov@yahoo.com, markovavanya@yahoo.com

***Abstract*—Algorithms for evolutionary computation, are applied in reinforcement learning autonomous agent to discover high-performing reinforcement-learning policies. Evolutionary reinforcement-learning approaches allow the agent to find good representations and cope with partial environment observability. We have compared the performance of classic reinforcement learning and evolutionary augmented autonomous agent in area of sequential games.**

***Index Terms*—Autonomous Agents; Deep Reinforcement Learning; Evolutionary Computation.**

I. Introduction

In recent years, the accuracy of machine learning methods has improved significantly thanks to progress in Reinforcement Learning and Deep Learning. Reinforcement Learning (RL) is a scientific area where the main topics are the autonomous learning and acting of an agent. So, agent is an environmental interacting autonomous entity, studying optimal politics for sequential decision-making in a wide range of areas both in the natural and social sciences [1]–[3].

The consistent structure in the modeling of different games has been dealt with in several theoretical and experimental studies [4] [5], which sets out strategies that summarize the results that would arise if cooperation or competition were chosen as one-off long play strategies.

Deep learning predominates in Reinforcement Learning over the past few years in games, robotics, natural language processing, and so on. We have seen breakthroughs like a deep Q-networks, AlphaGo, asynchronous methods and many others [6]–[8]. The integration of Reinforcement Learning and Neural Networks has a long history [9] and recent exciting achievements of deep learning [10] [11] leads to great benefits in areas of big data and data science. However, for several reward functions in RL, employing greedy optimization for a reward without any incentive mechanism can lead to sticking to a local minimum [12].

Evolutionary strategies are an approach that helps to find global minimums. A comprehensive overview of different Evolutionary Strategies techniques in the field of machine learning is given in the [13]. Several studies have been done so far [14] [15], however most of them consider the ES as an alternative to RL. If the agent greedily takes actions that maximize reward, the training data for the algorithm will be limited and it may not discover alternate strategies with larger payoffs. In this article we study, how evolutionary strategies could help us help to avoid getting into a local minimum in Reinforcement Learning training.

In our study, we combine Evolutionary Strategies as they were described in [13] and Deep Q-Networks [6]–[8] in Reinforcement Learning to explore the applicability and effectiveness of the agent learning in the field of Sequential Games. The main purpose of this study is to show that by guided exploration trough Evolutionary Strategies, the convergence of the learning process is faster. So the hypothesis in this study is that Deep Reinforcement Learning with Evolutionary Strategies exploration is more effective than Deep Reinforcement Learning with e-greedy exploration strategy.

This paper is organised as following, in Section2 we briefly describe some the underlying theory of Reinforcement Learning, Q-Learning and Deep Q Networks. In addition, we present the implementation of our approach. In Section 3 of our paper we describe the experiments and gather evidence to support our hypothesis.

II. Methods and Materials

A. Theory

1) Reinforcement Learning: To solve sequential decision-making problems, the agent should learn about the optimal value of each action, defined as the expected amount of future rewards when taking this action and following the optimal policy afterwards. Under a given policyπ , the true value of an action a in a state s is

$$Q_\pi(s;a) = E[R_1 + \gamma R_2 + \ldots | S_0 = s; A_0 = a;] \quad (1)$$

where $r \in [0;1]$ is a discount factor which trades off the importance of immediate and later rewards. The optimal value is then $Q_{\pi*}(s;a) = maxQ(s;a)$. An optimal policy can be easily learned from the optimal values by selecting in every state the highest valued action.

 ISBN: 978-1-61208-634-7

2) Q-Learning: The optimal action values can be derived through Q-learning [16] [17], a form of time learning. The real problems are too large to learn all the values of action in all states separately. Instead, we can learn a parametric value $Q(s; a; q_t)$. In this way, Q-learning values update the parameters after taking action A_t at S_t and observing the immediate reward R_{t+1} so that the resulting state S_{t+1} is then

$$q_{t+1} = q_t + \alpha(Y_t^Q - Q(S_t; A_t; q_t))\nabla_{q_t} Q(S_t; A_t; q_t) \quad (2)$$

where q is a scalar value and the target Y_t^Q is defined as

$$Y_t^Q = R_{t+1} + \gamma \max_a Q(S_{t+1}; a; q_t) \quad (3)$$

Updating the current value $Q(S_t; A_t; q_t)$ towards a target value Y_t^Q the agent apply stochastic gradient descent approach.

3) Deep Q Networks: Deep Q networks (DQN) are multilayered neural networks.These networks for a given state s outputs not a single action but a vector of action values $Q(s; a; q)$, where θ are the parameters of the network. If an action space containing m actions and state space is a n-dimensional vector, the neural network maps R^n to R^m. In addition in Deep Q networks there are target network [7], with parameters θ^-. This additional network is the same as the original network except that its parameters are copied every τ steps from the online network, so that then $\theta_t^- = t$, and are not changed on all other steps. So, the target used by DQN is then

$$Y_t^{DQN} = R_{t+1} + \gamma \max_a Q(S_{t+1}; a; \theta_t) \quad (4)$$

4) Double Q-learning: The max operator in standard Q-learning and DQN, in 2 and 4, uses the same values both to select and to evaluate an action. To prevent this overoptimistic value estimation we can decouple the selection from the evaluation. This is the idea behind Double Q-learning [18]. In the original Double Q-learning algorithm, two value functions are learned by assigning each experience randomly to update one of the two value functions, such that there are two sets of weights, and 0. For each update, one set of weights is used to determine the greedy policy and the other to determine its value. For a clear comparison, we can first untangle the selection and evaluation in Q-learning and rewrite its target 4 as

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, max_a Q(S_{t+1}; a; q_t); q_t) \quad (5)$$

The Double Q-learning error can then be written as

$$Y_t^{DoubleQ} = R_{t+1} + \gamma Q(S_{t+1}, max_a Q(S_{t+1}; a; q_t); q_t) \quad (6)$$

5) Evolution Strategies: If the action values contain random errors uniformly distributed in an interval $[-\epsilon, epsilon]$ then each target is overestimated up to $\gamma\epsilon\frac{m-1}{m+1}$, where m is the number of actions [19]. This could leads to local optima. So, we need a new approach for achieving the exploration strategy that will lead us to a global optima. Such kind of algorithms are Evolution strategies.

Evolution strategies (ES) are a class of black box optimization algorithms inspired by natural evolution [20]. At every iteration (generation), a population of parameter vectors (genomes) is perturbed (mutated) and, optionally, recombined (merged) via crossover. The reward (fitness) of each resultant offspring is then evaluated according to some objective function. Some form of selection then ensures that individuals with higher reward tend to produce the individuals in the next generation, and the cycle repeats.

Recent work from OpenAI outlines a version of NES applied to standard RL benchmark problems [14]. We will refer to this variant simply as ES going forward. In their work, a fitness function $f()$ represents the stochastic reward experienced over a full episode of agent interaction, where θ is the parameters of a policy π.

$$\nabla_\phi E_{\theta\sim\phi}[f(\theta)] = \frac{1}{n}\sum_{i=1}^{n} f(\theta_t^i)\nabla_\phi \log p_\phi(\theta_t^i) \quad (7)$$

where n is the number of samples estimated per generation. The sample parameters in the neighborhood of t and determines the direction in which t must move to improve the expected reward. Instead of the baseline, the Evolutionary Strategy relies on a large number of samples n to reduce the variance of the gradient estimate. To avoid bias in the optimization process due to large scale of reward between domains, we follow the approach of [14] and rank-normalize $f(\theta_t^i)$ before taking the weighted sum.

Optimizing for reward only can often lead an agent to local optima. NS, however, avoids deception in the reward signal by ignoring reward altogether. Inspired by natures drive towards diversity, NS encourages policies to engage in notably different behaviors than those previously seen.

The algorithm encourages different behaviors by computing the novelty of the current policy with respect to previously generated policies and then encourages the population distribution to move towards areas of parameter space with high novelty. NS outperforms reward-based methods in maze and biped walking domains, which possess deceptive reward signals that attract agents to local optima [12].

Optimization if is performed only regarding the reward can lead the agent to local optima. With ensuring more exploration trough Novelty Search, however, avoids deception in signal rewards, ignoring the overall reward. Inspired by nature's desire for diversity, the novelty search approach encourages politics to behave differently than those that have been seen before. The algorithm encourages different types of behavior, calculating the novelty of the current policy for newly created samples, and then encouraging the population. We use the ES optimization to compute and follow the gradient of expected novelty [21]. Given an archive A and sampled parameters $\theta_t = \theta_{t+i}$, the gradient estimate can be computed:

$$\nabla_\phi E_{\theta\sim N(0,I)}[N(\theta_t + \sigma\epsilon, A)|A] = \frac{1}{n\sigma}\sum_{i=1}^{n} N(\theta_t^i, A)\epsilon_i \quad (8)$$

The gradient estimate shows how to change the parameters of our current policy increase the average novelty in the

 ISBN: 978-1-61208-634-7

distribution of our parameters. We determine the progress of the gradient of A at the beginning of an iteration and is updated only when iteration is at the end. We add only a behavioral characteristic corresponding to each parameter vector, since adding these for each sample would cause the archive to be sipped and delay the calculation of the closest neighbors. To encourage extra diversity and reap the benefits of population surveys, we create two populations of agents that we will call followers as pursuers. Each agent, characterized by a unique identification number, is rewarded as being different from all previous agents in the archive (ancestors, other agents and ancestors of other agents). So, we have numerous agents in both populations, but we have not done a thorough analysis of how this variable parameter influences efficiency in different areas.

The choice of M depends on the domain, and that identifying which action is a beneficial for future research. We initialise the arbitrary parameters of M and, at each iteration, select one for updating. For our experiments, we choose which one to move from a discrete probability distribution as a function of the novelty of m. In particular, for each iteration of agent parameters, the probability of each being selected $P(m)$ normalized by the sum of novelty in all policies is calculated [21]:

$$P(\theta^m) = \frac{N(\theta^m, A)}{\sum_{j=1}^{M} N(\theta^j, A)} \tag{9}$$

After selecting a certain individual from the population, we compute the gradient of expected novelty with respect to current parameter vector, and perform an update step accordingly as it :

$$\theta_{t+1}^m \leftarrow \theta_t^m + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} N(\theta_t^i, A)\epsilon_i \tag{10}$$

Where n is the number of sampled perturbations to m_t , is the step size, and $i; mi = mt + i$, where $\theta_i \in N(0; I)$. Once the current parameter vector is updated, $b(mt + 1)$ is computed and added to the shared archive A.

B. Implementation:

We generate a discrete map with predefined dimensions. Then randomly place obstacles on the map. The next stage generates two lists: one with persecutors and one with prey. We study the influence of the number of pursuers and prey on the speed of learning learning. We also study the effect of the number of obstacles on the speed of learning. And also we study the influence of the amount of reward on the "elusive" movements of pursuers on the speed of training.

In our case, a group of predators pursues a group of victims (evaders). Since in the classic Pursue-evasion process, we study our problem as an MDP task. All members of both groups act after all members of the other group have committed their actions. Therefore, we could describe our approach as a classical sequential game. The pursuers and evaders have a short range of views, so they must move continuously. We determine the stochastic behavior of both groups imposing some additional rules. With a small probability α_{evader} will miss the opportunity to leave unnoticed and will give some handicap to the pursuer. On the other hand, the pursuer with a low probability $\alpha_{pursuer}$ "will lose" the evader from the site and, thus, give the prey a chance to evade.

In general, predators have a small negative reward for every "empty" step, and the victim has a small positive reward for every "evasion". If the pursuer "catches" prey, her reward is significantly increased (by almost two orders of magnitude), and the victim's reward will be reduced by the same amount. Groups are implemented on two lists: one for predators and one for evaders. When pursuers catch their victim a new prey is generated in a random place on the map, but out of the field of view of the pursuers.

In our case, a group of predators pursues a group of victims (malefactors) Classical reinforcement training consists in finding the best policy for the entire area with high details. Our approach is based on the following:

- Classical reinforcement training with e-greedy
- Reinforcement Learning with Evolutionary Strategy

III. Experiments and results

We gather evidence to support the hypothesis that using the Evolutionary Strategy will significantly speed up the training process of Agent's Reinforcement Learning. Hence, we claim that building of Multi Agent Deep Reinforcement Learning with Evolutionary Strategies is more efficient than Classical reinforcement training with e-greedy.

We perform the following experiment: For a given, map we should find an optimal autonomous agent group behaviour. The map is described by its size nxn and complexity rate R_c.

We have two methods: Multi Agent Classic Reinforcement Learning with e-greedy exploration and Multi Agent Deep Reinforcement Learning with with Evolutionary Strategies. And two cases:

- case study I - we have the map with almost no obstacles
- case study II - we have map with big amount of obstacles

We do the following task: for a given map we need to find optimal behaviour of pursuers. The agent's task is to travel on a chased the maximum preys for given amount of time. The environment is represented as a two-dimensional obstacle map. The map is described by its size nxn and the rate of complexity Rc.

We have two cases: in the one case, we have a small number of obstacles, which leads to a low probability of stranding the pursuers. Thus, even at low e-greedy values, the learning process should quickly reach the maximum reward. On the other hand, we investigate the behavior of agents pursuing the preys in a map with a large number of obstacles. In this case, if a greedy strategy is used, it is highly probable that the learning process will get stuck at a local minimum.

In both cases, the number of hunters and loot is the same, arguing that the only difference is the number of obstacles. Hence, the difference between the two experiments is the probability of falling into a local minimum. From Figure

 ISBN: 978-1-61208-634-7

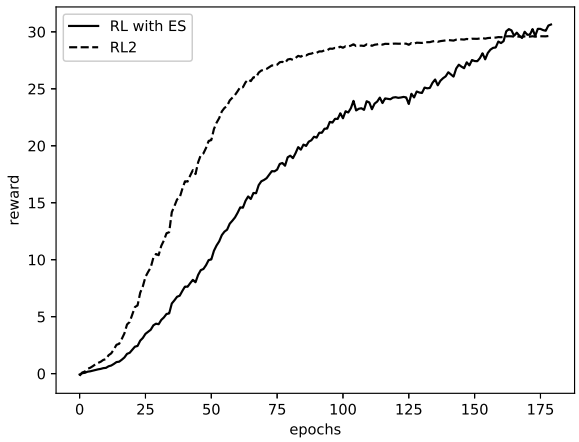

Fig. 1. We study the performance of Reinforcement Learning with Evolutionary Strategies(RL with ES) and classic Reinforcement Learning on map with low number of obstacles.

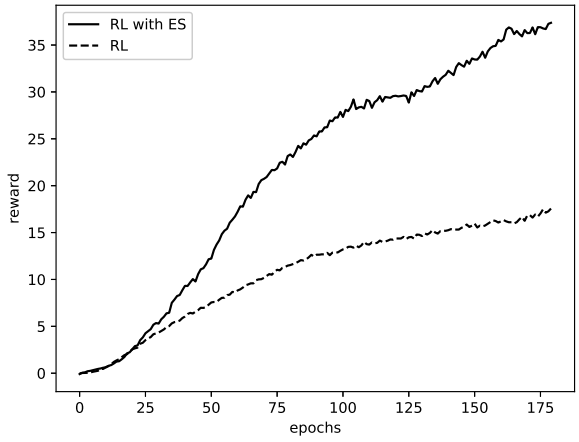

Fig. 2. We study the performance of Reinforcement Learning with Evolutionary Strategies(RL with ES) and classic Reinforcement Learning on map with big amount of obstacles.

1, it is believed that in maps with a low probability of trapping the pursuers and respectively a low probability of getting into a local minimum, the classic approach is better. However, in the event of a large number of obstacles (see Figure 2), the probability of jamming between the obstacles leads to a significantly higher probability of falling into a local minimum. In this case, the approach with evolutionary strategies is significantly better.

IV. Conclusion

The impact of different factors for building of Multi Agent behaviour is discussed in this paper. Two different approaches are presented: Multi Agent Reinforcement Learning (MARL) and Multi Agent Deep Reinforcement Learning (MADRL). The impact of four factors on Reinforcement Learning performance has studied. As a measure of performance is used the summary reward. In all case studies, the Multi Agent Deep Reinforcement Learning demonstrate significantly better performance than Multi Agent Reinforcement Learning.

References

[1] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press Cambridge, 1998, vol. 1, no. 1.

[2] C. Szepesvári, "Algorithms for reinforcement learning," Synthesis lectures on artificial intelligence and machine learning, vol. 4, no. 1, 2010, pp. 1–103.

[3] D. P. Bertsekas, Dynamic programming and optimal control 3rd edition, volume II. Belmont, MA: Athena Scientific, 2011.

[4] W. E. Walsh, R. Das, G. Tesauro, and J. O. Kephart, "Analyzing complex strategic interactions in multi-agent systems," in In AAAI-03 Workshop on Game Theoretic and Decision Theoretic Agents, 2002, pp. 109–118.

[5] M. P. Wellman, J. Estelle, S. Singh, Y. Vorobeychik, C. Kiekintveld, and V. Soni, "Strategic interactions in a supply chain game," Computational Intelligence, vol. 21, no. 1, 2005, pp. 1–26.

[6] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in International Conference on Machine Learning, 2016, pp. 1928–1937.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, 2015, pp. 529–538.

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al., "Mastering the game of go with deep neural networks and tree search," nature, vol. 529, no. 7587, 2016, pp. 484–489.

[9] J. Schmidhuber, "Deep learning in neural networks: An overview," Neural networks, vol. 61, 2015, pp. 85–117.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," nature, vol. 521, no. 7553, 2015, pp. 436–442.

[11] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, Deep learning. MIT press Cambridge, 2016, vol. 1.

[12] J. Lehman and K. O. Stanley, "Novelty search and the problem with objectives," in Genetic programming theory and practice IX. Springer, 2011, pp. 37–56.

[13] D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette, "Evolutionary algorithms for reinforcement learning," Journal of Artificial Intelligence Research, vol. 11, 1999, pp. 241–276.

[14] T. Salimans, J. Ho, X. Chen, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," arXiv preprint arXiv:1703.03864, 2017.

[15] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," arXiv preprint arXiv:1712.06567, 2017.

[16] C. J. Watkins and P. Dayan, "Q-learning," Machine learning, vol. 8, no. 3-4, 1992, pp. 279–292.

[17] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, 1989.

[18] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning." in AAAI, vol. 16, 2016, pp. 2094–2100.

[19] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," in Proceedings of the 1993 Connectionist Models Summer School. Hillsdale, NJ. Lawrence Erlbaum, 1993, pp. 255–264.

[20] I. Rechenberg, "Evolutionsstrategien," in Simulationsmethoden in der Medizin und Biologie. Springer, 1978, pp. 83–114.

[21] E. Conti, V. Madhavan, F. P. Such, J. Lehman, K. O. Stanley, and J. Clune, "Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents," arXiv preprint arXiv:1712.06560, 2017.

 ISBN: 978-1-61208-634-7