Deep Learning with Evolutionary Strategies for Building Autonomous Agents Behaviour

Ventseslav Shopov, Vanya Markova

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

***Abstract*—In this study, we will consider the construction of the behaviour of an autonomous agent in an environment that has many traps and a large number of obstacles. Such environments require the agent to build a policy that will lead them to the goal as quickly as possible. As a working basis, we use Reinforcement Learning and apply approaches from the field of random search and Evolutionary Strategies.**

Keywords–Autonomous Agents; Reinforcement Learning; Evolutionary Strategies.

I. INTRODUCTION

Reinforcement Learning (RL) is a scientific area where the main topics is agent training without supervision. Thus, an agent is an autonomous subject who learns and makes decisions independently. Such an agent, through interactions with the environment, finds the optimal policy for consistent decision making [1]–[3].

Deep learning prevails in the areas of studying natural language, recognising objects in the pictures of classification in multidimensional cases. Q-nets, AlphaGo, asynchronous methods and many others are examples of successful Deep Learning applications [4]–[8]. Deep learning leads to great benefits in areas of big data and data science. However, there are cases in which employing greedy optimisation for a reward can lead to sticking to a local minimum or suffer of slow converging [9].

Evolutionary Strategies (ES) are an approach that helps to find global minimums. A comprehensive overview of different ES techniques in the field of machine learning is given in [10]. Several studies have been done so far [11] [12], however most of them consider the ES as an alternative to RL.

In our study, we combine ES as they were described in [10] and Deep Q-Networks [4]–[6] in Reinforcement Learning to explore the applicability and effectiveness of the agent learning in the field of Sequential Games. At the moment, many specific methods of gradient descent have been proposed, but they all assume that the gradient behaves well: there are no cliffs where it increases abruptly, or a plateau where it vanishes. The first problem can be dealt with using the gradient clipping, but the second is more challenging.

The main objective of this study is to compare the performance of classical optimisation methods and ES as well as to verify how these algorithms affect learning speed. Thus, the hypothesis in this study is to compare the behaviour of gradient optimisation algorithms and algorithms for ES.

This paper is organised as follows: in Section 2, we briefly describe some basic theories of learning in the field of reinforcement, Deep Learning, and ES. In addition, we present the implementation of our approach. In Section 3 of our article, we describe the experiments and collect evidence to support our hypothesis. We conclude the work in Section 4.

II. METHODS AND MATERIALS

A. Theory

1) Autonomous Agent Behaviour: Information about past and current states of the agent and environment allows agents to evaluate their own progress. In reinforcement training, an agent builds up policies based on progress. The policy determines the reaction of the agent to the state of the environment. Through RL, the agent builds such policies that will achieve the goal with the maximum benefit for the agent.

So, if we describe the states of the agent and environment as a time series, then the task of making efficient plans will be significantly aided if the agent could forecast the future with desirable accuracy. An n-tipple (vector) is a result of one cycle of the work of the agent. It consists of the parameters of the behaviour of the agent: $b(b1, b2, \ldots, bn)$. The data from environment are collected and transformed into time series in the knowledge base of the agent.

2) Markov Decision Process: We formulate the transfer learning problem in sequential decision making domains using the following framework of Markov Decision Process. We use the following definition of Markov Decision Process (MDP) as a 5-tuple

$$< S, A, P, R, \gamma > \quad (1)$$

where the set of states, set of actions, transition function and reward function are described. $P : S \times A \rightarrow \Pi(S)$ is a transition function that maps the probability of moving to a new state given an action and the current state,

$$R : S \times A \rightarrow R \quad (2)$$

is a reward function that gives the immediate reward of taking an action in a given state. $\gamma \in [0, 1)$ is the discount factor. The MDP of the agent is described in (1), where S is the set of states, A is the set of actions, P is transition function and R is a reward function.

 ISBN: 978-1-61208-712-2

3) 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 policy π , the true value of an action a in a state s is

$$Q_\pi(s;a) = E[R_1 + \gamma R_2 + \ldots | S_0 = s; A_0 = a;] \quad (3)$$

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.

4) Q-Learning: The optimal action values can be derived through Q-learning [13] [14], 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 observe the immediate reward R_{t+1} so that the resulting state S_{t+1} is then

$$q_{t+1} = q_t + \alpha(Y_t^Q - Q(S_t;A_t;q_t))\nabla_{q_t} Q(S_t;A_t;q_t) \quad (4)$$

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

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

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

5) Deep Q Networks: Deep Q Networks (DQN) are multi-layered 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 [5], 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

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

6) Double Q-learning: The max operator in standard Q-learning and DQN, in 4 and 6, 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 [15]. 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 as

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, max_a Q(S_{t+1};a;q_t);q_t) \quad (7)$$

The Double Q-learning error can then be written as

$$Y_t^{DoubleQ} = R_{t+1} + \gamma Q(S_{t+1}, max_a Q(S_{t+1};a;q_t);q_t) \quad (8)$$

7) 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 [16]. 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 ES.

ES are a class of black box optimisation algorithms inspired by natural evolution [17]. 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 [11]. 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.

$$\nabla_\phi E_{\theta\sim\phi}[f(\theta)] = \frac{1}{n}\sum_{i=1}^{n} f(\theta_t^i)\nabla_\phi \log p_\phi(\theta_t^i) \quad (9)$$

where n is the number of samples estimated per generation. The sample parameters in the neighbourhood of t and determines the direction in which t must move to improve the expected reward. Instead of the baseline, the ES relies on a large number of samples n to reduce the variance of the gradient estimate. To avoid bias in the optimisation process due to large scale of reward between domains, we follow the approach of [11] and rank-normalise $f(\theta_t^i)$ before taking the weighted sum.

B. Implementation

The idea is quite simple. With a standard gradient descent, at each step we look at the inclination of the surface on which we are located and move in the direction of the greatest gradient. In ES, we fire a nearby neighbourhood with points where we can supposedly move, and move in the direction where most points with the greatest height difference fall (and the farther the point, the more weight is attached to it).

In the case of a piecewise-step function, the resulting estimate will represent the gradient of the smoothed function without having to calculate the specific values of this function at each point. Also in the case when the loss function depends on the discrete parameters, it can be shown that the estimate remains valid, since in the proof one can interchange the order of taking the expectation. Which is often not possible for ordinary Stochastic Gradient Descent (SGD).

$$\mathbb{E}_\epsilon \epsilon E(\theta+\epsilon) = \mathbb{E}_\epsilon \epsilon \mathbb{E}_x E(\theta+\epsilon, x) = \mathbb{E}_x \mathbb{E}_\epsilon \epsilon E(\theta+\epsilon, x) \quad (10)$$

The greater the sigma distribution, the less the local structure of the function manifests itself. When the sampling algorithm is too large, the optimisation algorithm does not show narrow minima and hollows, from which one can go from one good state to another. If it is too small, the gradient descent may not start if the initialisation point was chosen unsuccessfully.

 ISBN: 978-1-61208-712-2

Sampling makes noise in the gradient calculation, which makes learning more sustainable. Just like dropout in learning neural networks in the usual way. ES does not depend on frame-skip with RL. Also ES allow learning more easily than regular SGD, when a large amount of time can pass between an action in RL and a positive response, and in noisy conditions, when it is not clear which change helped improve the result. What are the disadvantages? The computation per episode is slower than in SGD. And the final results are not significantly better. Noise in gradients - even with one-dimensional optimisation, are slightly unstable.

The RL algorithm can query the environment by sending it the suggested policy π. The model then selects a random variable e, independent of the past, and generates a vector from the system in accordance with the policy π and randomness e.And then the model returns to the our algorithm a sequence of states, actions, and rewards (s, a, r), which represent a vector generated from the system in accordance with the policy π. In this scheme, one request is called an episode. The purpose of the RL algorithms is to approximate the solution of problem by making as few calls as possible to the medium.

III. Experiments and Results

For the purposes of our research, we do the following : we look at a stochastic single player game that strives to maximise its winnings. The game is a 2d map in which the player must reach a certain goal by avoiding certain traps. The reward in the target is 100 and the reward in the trap is -100. For each idling, the player receives a -5. The game has a stochastic policy because the probability of going to the next scheduled state is 0.9 and with probability 0.025 the agent will either end up in one of the neighbours to the current state or will remain in the current state. In this way, an odometric error or a real agent monitoring error is modelled. If the agent made a transition to an obstacle or out of the map, we see this as a ”collision”. Upon collision, the agent returns, returning to the current state and receiving a -10 reward. The agent performs one episode until it reaches a terminal state or by making a number of steps larger than the size of the environment.

Maps are rated by many parameters as: size, size of hurdles, trap to size ratios, and reward ratios to size. The latter is always inversely proportional to the size of the map. We are looking at a couple of specially made maps:

- map with minimal obstacles and traps. This map is a virtually ideal playing field. The likelihood of collision or the agent becoming trapped is minimal. Depending on the ratio of the reward to the size, the agent is favourably trained in small-sized maps.
- map with a significant number of obstacles and traps. In this case, we have an obstacle to size ratio of 0.2 and trap ratios to the size of 0.2. On this map, the total return is less than the first. However, obstacles and traps are selected so that there are no conditions for occurrence of local minima.
- map with a significant number of obstacles and traps designed to generate a local minimum. This map has the same ratio of obstacles and traps as in the previous case, but here the goal is surrounded by traps and obstacles. We have done this arbitrarily in order to check how our policy optimisation methods will behave in such a situation.

We create a model of the RL problem in a way similar to the one in [18]. This model allows us to get an estimate of the information our agent can extract from the environment. The training agent generates policy and applies it to the environment. In fact, the agent uses this policy for an episode. In addition, the agent generates a random magnitude that I apply to policy parameters. This magnitude is different and independent for each step. In this way, the environment generates a vector with the responses to the proposed policy for each step.

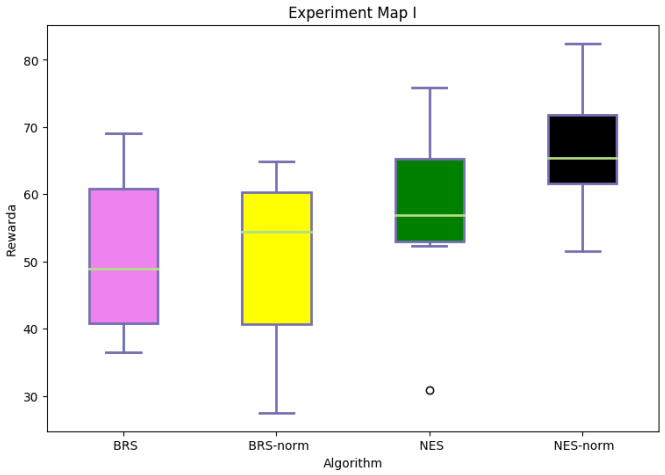

Figure 1. We study the performance of the algorithms in simple map.

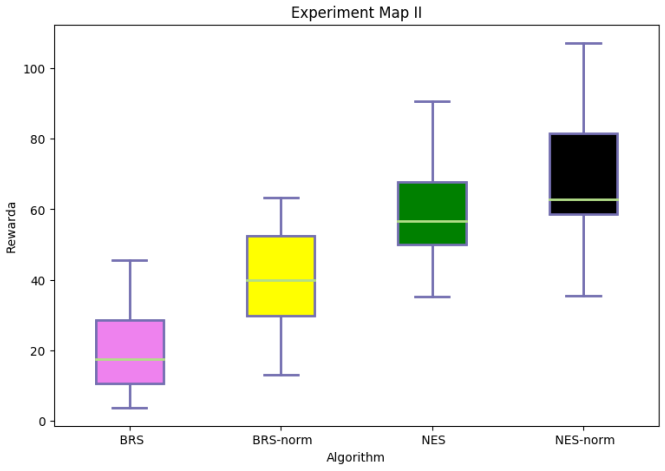

Figure 2. We study the impact of higher number of obstacles and traps.

This vector has the form $< s, a, r >$ where c is the current state, and the action a r is the reward. This vector is recorded for each episode. The optimisation method should change the policy parameters depending on what reward is awarded at each episode step. Through this model, we get the opportunity to generate queries to the environment and get vectors with all of the agent’s trajectories for each episode.

From a practical point of view, the agent strives to obtain a policy whereby the overall return is maximum. Creating a stop criterion is not a trivial task especially if we have a stochastic pattern of behaviour. Fluctuations in rewards as a result of

 ISBN: 978-1-61208-712-2

unfavourable coincidence of random events lead to a significant volatility of the overall return. However, in our research we are are primarily interested in the comparative characteristics of the two policy optimisation approaches. Therefore, we will ignore the convergence criterion and set a final number of epochs as a measure of completing the training.

We compare the following four algorithms: BRS, BRS-norm, Natural Evolutionary Strategies (NES) and NES-norm. BRS and BRS-norm differ only in that the initial initialisation of the BRS-norm parameters is normalised according to the maximum and minimum reward. The same applies to NES and NES-norm.

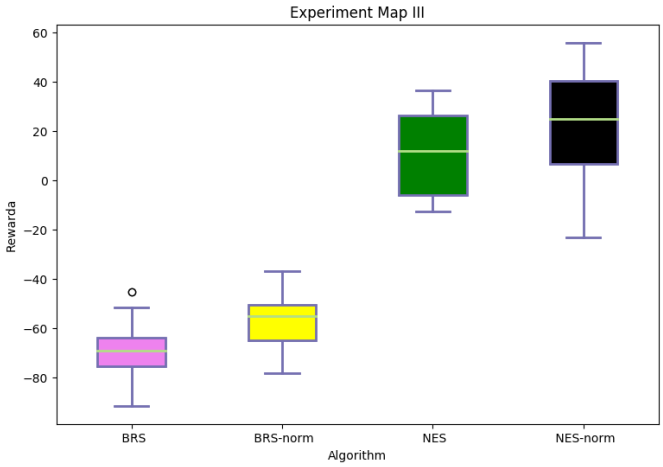

Figure 3. We see the impact of ”local minima” environment.

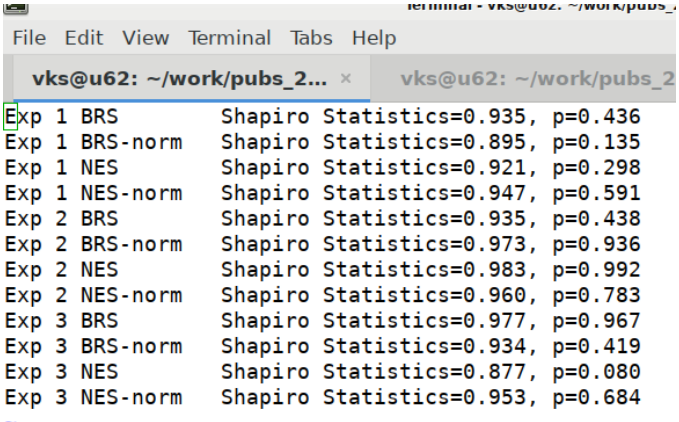

Figure 4. The Shapiro tests shows that all results have normal distribution.

The results of first experiment can be seen in Figure 1. One can see that under favourable conditions the cumulative reward after training does not differ significantly. The results of second experiment are shown in Figure 2. Here we can see that NES and NES-norm have a higher median reward, but their dispersion also is higher. It is only in the third experiment (Figure 3) that we see the superiority of ES. It seems that Basic Random Search (BRS) and BRS-norm are stuck in the local minima. NES algorithms perform much better although they show higher volatility.

From Shapiro's tests (result shown in Figure 4), it can be seen that as the complexity of the environment increases, the volatility of the solutions increases. BRS and BRS-norm demonstrate more stable but significantly lower performance, while NES and NES-norm achieve a higher overall return but at the expense of increased volatility.

IV. Conclusion and Future Work

In this study, we looked at building an autonomous agent's behaviour in an environment that has both rewards and traps. Such environments require agents to build a policy that leads them as quickly as possible to the goal. On the other hand, the agent should ”avoid” traps especially in the case of a stochastic policy of movement.

As a working framework, we used Reinforcement Learning. We compared approaches from the field of random search and Evolutionary Strategies. Experiments have shown that methods based on an evolutionary approach show better results when the environment is more complex. Especially important is the superiority of Evolutionary Strategies in cases where the environment has local minima.

References

[1] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press Cambridge, 1998.

[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] V. Mnih et al., “Asynchronous methods for deep reinforcement learning,” in International Conference on Machine Learning, 2016, pp. 1928–1937.

[5] ——, “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, 2015, pp. 529–538.

[6] D. Silver et al., “Mastering the game of go with deep neural networks and tree search,” nature, vol. 529, no. 7587, 2016, pp. 484–489.

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

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

[9] 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.

[10] 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.

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

[12] F. P. Such et al., “Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning,” arXiv preprint arXiv:1712.06567, 2017.

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

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

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

[16] 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.

[17] I. Rechenberg, “Evolutionsstrategien,” in Simulationsmethoden in der Medizin und Biologie. Springer, 1978, pp. 83–114.

[18] H. Mania, A. Guy, and B. Recht, “Simple random search provides a competitive approach to reinforcement learning,” arXiv preprint arXiv:1803.07055, 2018.

 ISBN: 978-1-61208-712-2