Reinforcement Learning using Simple Evolutionary Strategies

Ashkan Alami, ashkan.alami@studenti.unitn.it, 225087

***Abstract*—This report discusses how simple evolutionary strategies, can be used for solving reinforcement learning problems. we start by implementing an agent and evolutionary strategies. we train and optimize the agent actions by evolutionary strategies, to get the best reward in the problem environment.**

***Index Terms*—evolutionary strategies, reinforcement learning**

I. Introduction

In reinforcement learning, developers create a method of rewarding desired behaviors and punishing negative behaviors. This method assigns positive values to the desired actions to encourage the agent and negative values to undesired behaviors. This programs the agent to seek long-term and maximum overall reward to achieve an optimal solution.[1] The agent is placed in an environment that has different states. In each state, the agent chooses the best action, and this action changes the agent's state to another state and rewards it. In each episode, the agent starts from the initial state and tries to gather more rewards until the end of the episode. In other words, the agent takes a state as input, and action is its output, and its goal is gathering the most rewards in the environment, and optimizing its action to reach the best reward at the end of each episode. Researchers use deep models to choose the best action in complex environments. This approach is called deep reinforcement learning and it shows a promising result in these types of problems. However, it is not easy to use deep models in reinforcement learning. Despite supervised learning, in reinforcement learning, there are no true values that can use them in backpropagation, and train the model. One of the strategy that researchers recently use for training the deep model in these problems is evolutionary strategies. In this method, they find the best parameters in the deep model to receive the optimum reward at the end of each episode. In this work, we implemented simple evolutionary strategies algorithms and used them for training an agent for a reinforcement learning problem.

II. Approach

For creating the reinforcement learning problem we used openai gym library. The gym is a toolkit for developing and comparing reinforcement learning algorithms [2]. cartpole-v1 is chosen for the environment. The "cartpole" agent is a reverse pendulum where the "cart" is trying to balance the "pole" vertically, with a little shift of the angle.[3] Agent observes 4 values in each state and chooses between 2 actions moving right or left. For implementing the deep model we used Pytorch library. PyTorch is an open source machine learning framework[4]. Moreover, we implement 8 simple uncorrelated evolutionary algorithms. Static, adaptive: 1/5 rule, self-adaptive with single σ and Self-adaptive with multiple σ for (μ, λ)-ES and (μ + λ)-ES. in (μ, λ)-ES algorithms next generation only chosen from offspring, but in (μ + λ)-ES algorithms if parents have a better result, can come to the next generation.

Static: In this method σ is fixed and does not change. User chooses its value. Parameters updated as following formula.

$$x_i' = x_i + \sigma \times N(0,1) \tag{1}$$

Adaptive: 1/5 rule: Parameters updated as Static method. However, σ dose not stay fix in this method. It is gets updated is following:

$$\sigma' = \sigma / c \;\; if \;\; p_s > 1/5 \tag{2}$$

$$\sigma' = \sigma \times c \;\; if \;\; p_s < 1/5 \tag{3}$$

$$\sigma' = \sigma \;\; if \;\; p_s = 1/5 \tag{4}$$

We define p_s as the number of last k iterations that have better rewards compare to last iteration. If p_s is more than one-fifth of k then we change $sigma$ as equation 2, if it is less we follow the equation 3, and if they are equal, equation 3. As a default k is 5 and c is 0.8.

Self-adaptive with single σ: In this method σ and parameters updated as following:

$$\sigma' = \sigma \times \exp(\tau \times N(01)) \tag{5}$$

$$x_i' = x_i + \sigma' \times N(0,1) \tag{6}$$

Where τ is learning rate and $\tau \propto 1.0/sqrt(n)$.

Self-adaptive with multiple σ : In this method we have multiple σ. each parameter has its σ. Parameter and σ updated as following.

$$\sigma_i' = \sigma_i \times \exp(\tau' \times N(0,1)) + \tau \times N(0,1)) \tag{7}$$

$$x_i' = x_i + \sigma_i' \times N(0,1) \tag{8}$$

τ' is global learning rate and $\tau' \propto 1.0/sqrt(2n)$ and τ is coordinate-wise learning rate and $\tau \propto 1.0/sqrt(2 \times sqrt(n))$.

III. Result and Discussion

In each iteration, we save the average reward of the population and also, the best reward. We changed the max reward of the cartpole-v1 environment to 200. As we can see in the figures the best reward in (μ + λ)-ES methods are a quit less oscillating than (μ, λ)-ES . Since we keep parents and if parents have a better reward rather than offsprings we do not lose them in the next generation. Between different methods, we believe adaptive σ models work better than static, they usually reach the max reward faster, and they are more stable after that. Ultimately, all methods solve the problem, and the agent act in the best way to take the best result, however, there are some drawback and issues that we faced, we are going to explain them and suggest some solutions for them.

In the beginning, the average reward was only considered as a parameter to show how the model is improving, however, in most cases and different runs it did not reach and merge to the maximum reward. After studying the best reward in each generation, we understand that since the model in each iteration explores the space around the previous parameters, when the model gets reaches the best result, still in each generation after that create some worse parameters that do not work as well as the best result and these parameters decrease the average reward. This is the reason why usually the average reward does not reach the best reward. For increasing the average reward and encouraging all parameters in a generation to reach the best result we should decrease the exploring space when they get close to the max reward. To do this, we should decrease the σ, which happens with adaptive models. However for getting a better result, we should define the initial parameters due to the problem (for example model architecture, parent size, etc), and after a lot of iteration, each generation member should get the max reward.

Another interesting fact that we faced, is the best reward also can decrease in the next generation, even in (μ + λ)-ES models. By checking the best result in each generation, we can see some individuals reach the best rewards in about 50 iterations, however, in most cases next generations produce the worse best reward and the best reward oscillation to getting better and worse. Since in (μ + λ)-ES models we can keep the best model and transfer it to the next generation, we did not expect to best reward decrease in the next generations. After studying it we understand that in Reinforcement Learning problems environment can change each time, and the agent starts with different initial states. Therefore it means if a model can solve a problem one time, it does not mean the model can solve this problem for a second time with another initial state. For cartpole-v1 problem, it is not so obvious and the model usually reaches the best result in the first iterations. We check the model in a more complex LunarLander-v2 environment and this issue was recognizable. When we checked the best model in this environment which got the reward of about 280 (above 200 points means this problem is solved) and model failed in multiple tests and only solved the problem in a few tests. For solving this issue we suggest each model solve the problem couple of times with the same parameters and consider the average of these rewards as the final reward for these parameters. It goes without saying that this solution is computationally expensive however can produce a more accurate result.

For more exploring the results, in the static method, we can see average reward with smaller σ oscillating less than average reward with bigger σ. also, a bigger population helps the model to find and coverage to the best reward, however, it is computationally expensive.

Unfortunately, due to the stochastic nature of Reinforcement Learning problems, we believe, there are not the best problems for comparing the different ES methods and seeing each one how moves forward the optimum in the same problem. If we use the same method for the same problem two times, we can see in both times model merge to the best reward, but with different optimizing paths. The best way to compare them in these problems is using the optimizing methods several times to approximately understand how there are working. The result that we mention about the difference in (μ, λ)-ES and (μ + λ)-ES method and effect of σ are driven with several runs and checking the results. With just comparing results from two methods we do not think can drive an accurate comparison.

IV. Conclusion

In this work, we implement a few simple uncorrelated ES methods and use them for solving a reinforcement learning problem. These simple methods showed a promising result. There optimize our deep model, the model can do the best action in each state of the environment to at the end of the episode gather the most rewards. Due to the stochastic and tricky nature of reinforcement learning problems, it is hard to drive a solid conclusion about our different methods. however, we observed that (μ + λ)-ES models work better than (μ, λ)-ES models. Also, for reaching better results it is better to use adaptive models.

References

[1] J. M. Carew, "What is reinforcement learning? A comprehensive overview," SearchEnterpriseAI, 29-Mar-2021. [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/reinforcement-learning.

[2] OpenAI, "A toolkit for developing and comparing reinforcement learning algorithms," Gym. [Online]. Available: https://gym.openai.com/.

[3] Fakhry, A. (2020, November 13). Using Q-learning for OpenAI's cartpole-V1. Medium.Retrieved from https://medium.com/swlh/using-q-learning-for-openais-cartpole-v1-4a216ef237df

[4] Wikimedia Foundation. (2022, February 6). Pytorch. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/PyTorch

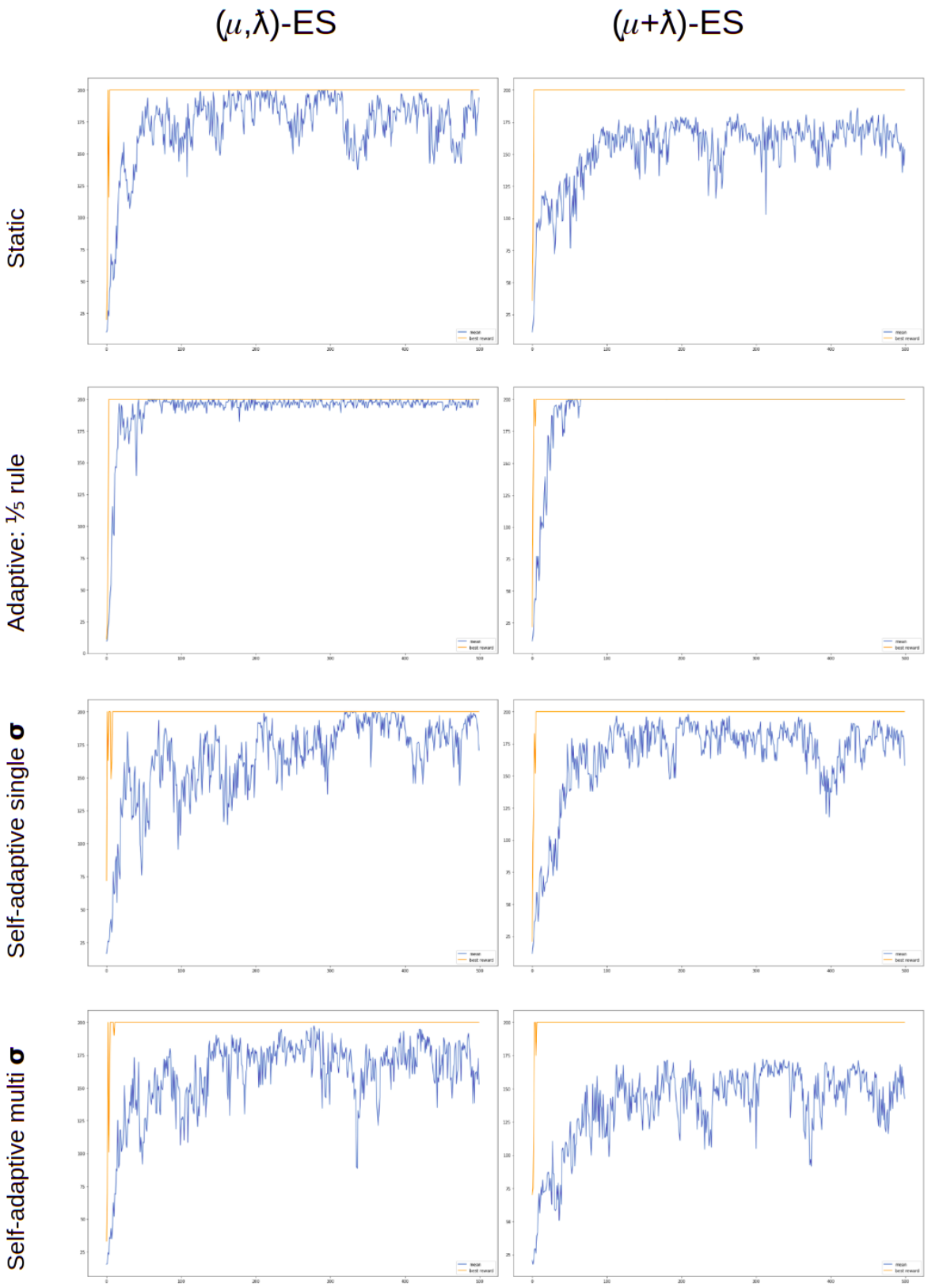

Fig. 1. These are the results of learning for 500 iterations, the population size of 50, and the parent size of 5. Orange line shows the best reward and blue line average of the reward in that iteration. As the plot is shown in the first iteration best rewards reach the maximum reward.

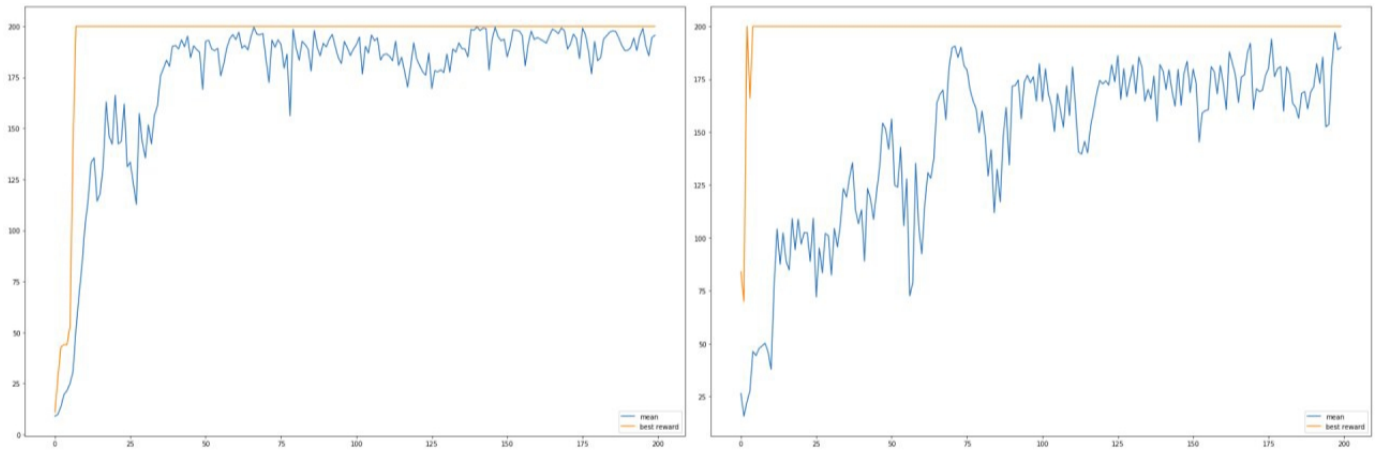

Fig. 2. Result of the same static model with different σ, left plot σ is 0.005 and right plot σ is 1.