Generation of learning: A comparative study between Genetic Algorithm and Reinforced Learning Algorithm

Thomas van der Sterren

University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark
thvan23@student.sdu.dk

Abstract. Within this comparative study, two well known Artificial Intelligent (AI) algorithms are implemented. It explores the key difference of the application of a Genetic Algorithm (GA) and a Reinforcement Learning Algorithm (RL). The analysis discussed within this paper reveals a significant difference within learning time required. This comes to show that the Artificial Intelligent algorithm to which is used is very depended on the environment to which it is deployed.
Keywords - AI, Genetic Algorithm, Reinforcement Learning, Gymnasium

1 Introduction

The presence of Artificial Intelligence (AI) in modern technology is undeniable. From its use in everyday consumer products to its application in cutting-edge robotics, AI is rapidly transforming various aspects of our lives. A lot of development in AI has shown its use in a lot of fields, such as automotive, robotics or even household appliances. As seen within Tesla's self driving vehicles, the motion control and obstacle avoidance of the Boston Dynamics Spot Robot Dog, or the Smart Fridge developed by Samsung. However, selecting the fitting AI for a specific job can be challenging but is crucial for its success. Within this study two different algorithms have been tested to see what the difference in the result to its environment is. The two algorithms discussed are the NEAT Algorithm from the class Genetic Algorithms and the A2C algorithm for Reinforcement Learning. This study aims to identify some of the differences that are presented when using two different algorithms for the same task.

2 Background

2.1 AI

Artificial Intelligent is a current field of research where the goal is to make machines able to reason the same way, or better, than animals can. Although this is already widely used in different applications, the advancements are still very pronounced. This study mainly focuses on two types of algorithms, being the Genetic Algorithm and the Reinforcement Learning Algorithm.

2.2 Genetic Algorithm

A GA takes after the Darwinian evolutionary theory. Where only the best survives and gets to procreate. in figure 1, a schematic overview is presented on how a GA works in principle. As can be seen, a population is created. In the case of this study, the population consists of neural networks trying to solve the challenge inside the environment. After each actor has completed the challenge, it receives a score (based on its rewards). The actors that performed the best will get to mix parts of their network with those of others to create offspring. This creates a second generation of actors in the population. They in turn will also perform the challenge and create offspring. This continues for a set amount of generations. To make sure that the generations still have new finding opportunities, it is possible to introduce random mutation. This part will randomly adjust some parts of the network to try and produce a new genome which has greater success.

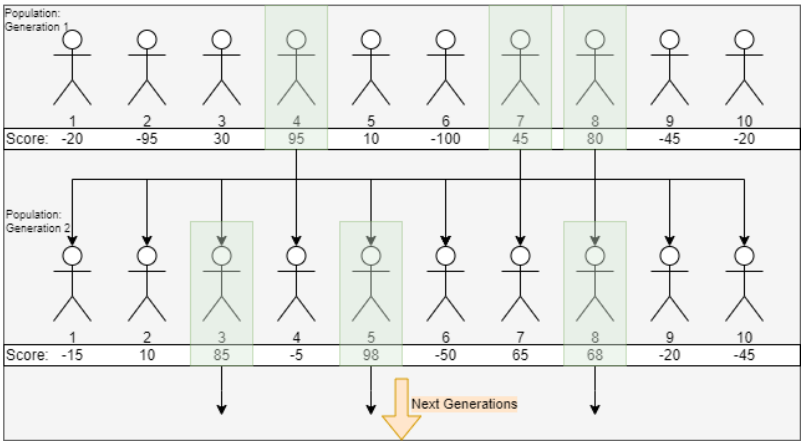

Fig. 1. Simple Schematics overview of a Genetic Algorithm.

Within GA there are a few different subsets. Within this study NEAT is used, which stands for NeuroEvolution of Augmenting Topologies.

NEAT NeuroEvolution of Augmenting Topologies is a subsets of the Genetic Algorithm, where next to the weights of the neural network, also the topology of the Artificial Neural Network evolves. Next to the topological improvement, NEAT also improves on the concept by preserving innovation. Something GAs often struggle with is phenomenon called "destructive crossover." This is when two parents merge, and 'breed' an offspring without their good quality. NEAT improves on this using a "historical markup." Using this, NEAT keeps track of the improving qualities of parents, to be further improved on. [1]

2.3 Reinforcement Learning

Reinforcement learning (RL) is an interesting field of Machine Learning developed based on how animals learn. Unlike supervised learning, where the algorithm is forced to learn with a dataset, reinforcement learning is free to 'roam' the environment. It does not have a predefined set of instructions but instead gets rewards or penalties based on actions it takes within the environment. This means that through trail and error, the agent can become more sophisticated as it learns.

A2C Advantage Actor-Critic (A2C) is a specific type of Reinforcement Learning algorithm which aims to create a high variance in learning. The learning algorithm consists of two parts, an Actor and a Critic. The Actor takes actions in the environment, while the Critic the quality of the action and compares this with the average of a similar situation. In essence, the Critic evaluates how much better or worse the action is compared to the usual outcome. By using a critic in the loop, it allows the learner to be better aware of the learning growth, and reduce the odds of learning something based on 'getting lucky'. [2]

3 Method

This section will give an overview how the experiment between the two AI types have been designed and performed.

3.1 The Environment

To start off, it is important to get an understanding of the environment that the two AIs will be facing. For this study, the Lunar Lander of OpenAI's Gymnasium was used. [3] The environment, displayed in Figure 2, consists of a small controllable blue lander, and a lunar surface. The goal for the AI is to land the lander on the surface between the two flags softly, so the legs of the lander don't break. The AI can take one of four actions every step. The four actions are; do nothing, fire left roll thruster, fire right roll thruster, fire main thruster. As the name suggests, the roll thrusters will rotate the lander, where as the main thruster will help it slow its decent down. For the AIs to know what action to take, they must be able to observe the environment. This is done using the 'Observation Space'. The environment, upon every step, returns observations from its current state. In this case it returns the following; The lander's X-Y coordinate, its angle, its linear and angular velocity, and two booleans representing whether the leg has contact with the ground.

To know whether the AI is acting the way it is intended, the environment also returns rewards. Rewards are either positive or negative, depending on whether the AI is advancing towards its goal. The rewards for this environment consist of; increase or decrease when getting closer to the landing-pad, increase or decrease for the speed of the lander, decrease for tilting, increase for landing, and decrease for engine use.

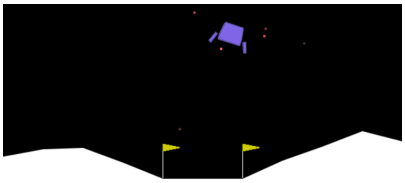

Fig. 2. The Lunar Lander environment from OpenAI's Gymnasium.

3.2 Creating the AI

The two types of AIs were both created with the use of libraries that support it. The data was collected using Tensorboard, allowing to be exported and further analyzed.

All the code for this project can be viewed and downloaded from the repository located on Github. [4]

GA: NEAT The NEAT code was implemented using the NEAT-Python library [1]. Where the configurations were set to have a population of 75. Each actor in the gene pool would have 30 attempts. By increasing the attempts, it becomes less likely that an actor succeeded because of a single lucky start, or failed because of one unlucky start. Thus increasing the attempts an actor can make will increase the likelihood that it is based on skill. Next to running the code responsible for generating the ANN, it also uses the TensorboardX[5] library to report the total reward of each episode to a log. This log can then be viewed on the Tensorboard.[6]

RL: A2C To learn an ANN using Reinforcement Learning, the python library Stable Baseline3 was used [7]. This library easily wraps a learning algorithm with ANN, and connects this to an environment. Within the python code callbacks can be specified, which in this case, were used to write data to the Tensorboard [6] using TensorboardX.[5]

3.3 Data Collection

Now that the AIs have been created, and a method of gathering data has been implemented, it is time to have the AIs train. To train the AIs they each will train for 90 minutes. After these 90 minutes they will preform a visualization that a human controller can see. This will be a good indicator whether the AI has functioned towards a desired outcome.

Since the AIs start with a random value, three tests will be done with each. This data will then be plotted side-by-side to get a good comparison view. As both AIs will be trained using the same computational system, the comparing results should not be conflicting based on hardware, however, it is key to note

that using a more powerful system would yield a faster learning speed. This is due to the usage of the CPU and calculations during the learning phase. Throughout this test, an AMD® Ryzen 7 4800h was used with 16GB of RAM.

4 Results

Within this section, a review of the results gathered from the two different AIs, is described. The runs lasted 90 minutes, and were performed 3 times per AI. The derived results can also be viewed within the repository linked to this study. [4]

The metric that was decided to be studied was the mean reward over time of each episode. Of each algorithm the average of their three runs was used to see if there was a significant difference. The three different runs for A2C and NEAT are displayed in figure 3 (1) and figure 3 (2) respectively.

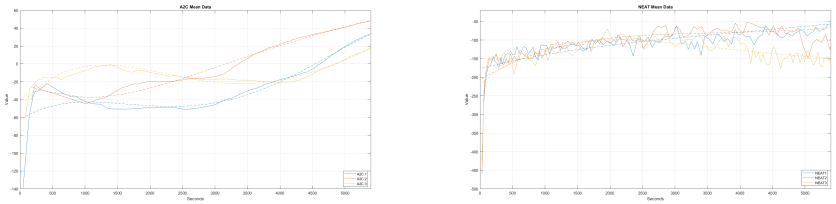

Fig. 3. A2C mean data (1) and NEAT mean data (2).

Though the mean of the NEAT Algorithm seems low, it actually has fairly high performance as well. Looking at figure 4, a very good run has been recorded. This is the best performing net from the second trail in the NEAT algorithm. It is crucial to be aware of this, as when deploying such AI, one would only use the best performer.

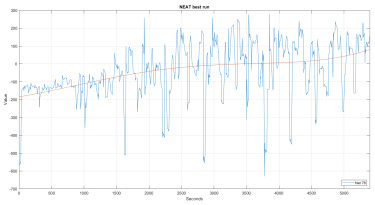

Fig. 4. Top net of second run NEAT.

The statistical method used to compare the data was a Mann-Whitney U test, which returns a p-value of the data and came to the conclusion that there was a significant difference with p being 3.44×10^{-265} and thus smaller then 0.05. This large difference is also visible in the graph shown in figure 5 below.

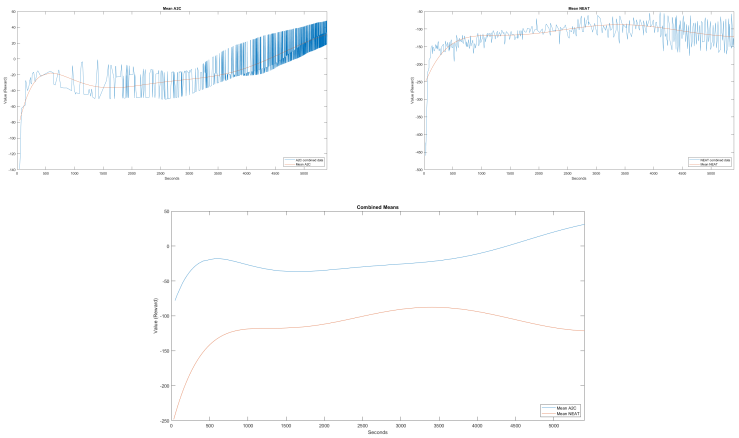

Fig. 5. The mean of all runs for A2C (1), NEAT (2) and plotted together (3).

After the AIs trained, a visual interface would start for the human operator. As can be seen in figure 6 the lander successfully landed. This end state was observed within both networks under a multitude of trials.

Fig. 6. The successful landed lander, visualized within the environment.

5 Discussion

As stated in the results section, a significant difference can be seen between the results of the means of the two AI's. Looking back at the main differences between how the two algorithms learn it is logical. Where one algorithm focuses on the improvements of the weights using a critic, the other algorithm focuses on optimizing the topological arrangement of the nodes and layers using evolutionary theories.

Although NEAT has the benefit of being more exploratory, it comes at a cost. As can be seen from the results, the final mean rewards don't increase fast after the initial start. However looking back at the spread of the data within the NEAT algorithm compared to that of the A2C, it becomes apparent that the NEAT algorithm has a larger exploration element. Leading some networks within the NEAT learning to exceed the reward compared to that of the A2C. Thus, the exploring could be beneficial when learning in more complex environments or when learning for a larger time. This in contrary to a Reinforcement Learning algorithm such as A2C, as once the actor gets better at finding the rewards, it will focus less on exploring, and thus risking getting stuck in a local minimum. [8]

Future studies could identify if there is potential benefit of overlapping the two algorithms to use each others benefits and strengthening the Artificial Neural Network. Allowing the exploration and adaptation of the topological network of the Artificial Neural Network, while maintaining the strong learning technique of the A2C learning Algorithm.

6 Conclusion

Within this study, two AI's were generated using two different algorithms. The AI's would have to land a lander in the OpenAI gymnasium Lunar Lander environment. The two algorithms that were investigate were the Genetic Algorithm called NEAT, and the Reinforcement Learner, A2C. While both methods created an ANN that was competent and able to interact with the environment, the time it took to train the network to gain a certain reward level would vary. In the training, A2C focused on training the weights within a fixed architecture, while the NEAT focuses on evolving the topology of the architecture itself. Looking at the difference in results, one could say that the topological structure of an ANN has less influence on its quality then the weights of a structure, however that would require further investigation.

A possible further investigation could be to combine the two algorithms to essentially create a changing topological ANN while still training them using a form of reinforcement learning.

Overall this study does highlight the importance of fitting a type of algorithms for the task, where a Genetic Algorithm might excel, would not necessarily be a good place to implement a reinforced learning algorithm. The same goes the other way, where a reinforcement learning algorithm might excel over a genetic algorithm.

7 Acknowledgment

During the study a couple of people were paramount in the brainstorming phase. I would like to take this moment to thank these brilliant minds for their support and contribution as a discussion partner. For starters the professor, Xiaofeng Xiong of the subject, giving us the freedom to develop within a chosen environment, whilst still providing examples in the case we would get stuck. This freedom allowed a more in-depth research into the different possible AIs and training environments giving a better sense of the opportunities and possibilities out there. Additionally, I would like to thank Angel Manuel Montejo Quesada. An excellent colleague student with whom discussion on the process led to a better result.

References

1. A. McIntyre, M. Kallada, C. G. Miguel, C. Feher de Silva, and M. L. Netto, "neat-python."
2. V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016.
3. M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," Mar. 2023.
4. T. van der Sterren, "Generation of learning." `https://github.com/blatv/SDU_GA-vs-RL`.
5. lanpa et al., "tensorboardx," 2017.
6. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.
7. A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.
8. "Escaping local minima in deep reinforcement learning for video summarization," 2023.