Comparative Analysis of Genetic Algorithms and Reinforcement Learning

Filippo Balzarini[a], Jason Kaxiras[a], Melvin Gode[a]

[a]*Department of Computer Science, Uppsala University, Uppsala, Sweden*

May, 2024

Abstract

The abstract should be *very* brief, two or three sentences may be enough. It must answer the following questions, however: 1. What did you do (What did you measure)? 2. How did you do it (which method)? 3. What did you discover (what was the results of the experiment)? Results in form of numbers should be accompanied by an error: $R = (3.05 \pm 0.02) \cdot 10^{-6}\,\mathrm{kg/s^2}$

1. Introduction

In the feild of Machine learning there are several methods which can be used to solve the same problem. For example trying to find the local minima using eveolutionary algorithms or supervised learning in the form of back propagation. Both come with advantages and disadvantages depending on if we are after precision or just want to find the local minima fast, but also depending on the nature of our problem.

More specificaly intresssting is the comparison of the performance of Genetic Algorithms (GA) and Reinforcement Learning (RL) techniques in the context of a game environment. On one hand reinforcment learning the agent engages dynamic and evolving enviroment by taking actions that affect it in order to acomplish a specific job. On the other hand we have evolutionary algorithms which employ evolutionary princibles for automated and concurrent problem-solving by drawing inspiration from populations of interacting organisms. Despite their apparent dissimilarities, RL and EC both tackle the same fundamental issue: optimizing a function. This entails maximizing an agent's reward in RL and the fitness function in evolutionary algorithms, respectively, particularly in environments where the parameters may be unknown [1].

2. Background

2.1. Reinforcment Learning

Reinforcment learning is defined as the problem that an agent tries to solve by learning behaviour through trial and error with its enviroment. In other words programming an agent through rewards and punishments rather than how to specificaly solve the task itself[6] as depicted in figure 1.

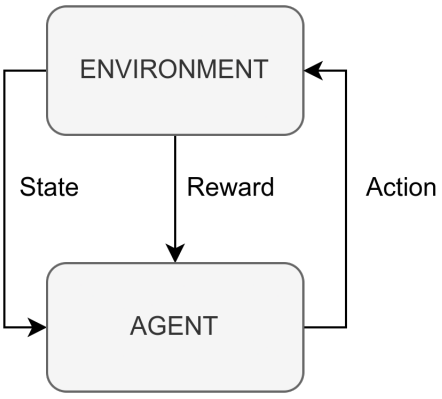

Figure 1: Graph representing reinforcment learning

The first concept crucial for reinforcment learning is the *reward function* which is objective feedback from the enviroment. It is usually scalar values that are associate with state action pairs. High rewards are usually associate by state-action pairs which benificial for the agent to be situated in whereas negative rewards would then be disadvantageous states or *hazardous* for the agent to be in. Essentially what is good and bad for the agent in the enviroment. The sole objective of the agent is then to maximize this reward[7].

Naturally we have to define *state* and *action*, which compared to the rest of the concepts have a very general definition. That being the latter is a descision of some sort and the former a factor that has to be taken into considaration when taking an action.

Equally importand is the *value* or *update* equation which is responsible for mapping the different states bassed on their estimated long term reward. There can be several algorithms for how to update the values but we use Q learning in this report. Q learning is an algorithm where the enviroment can be constituted by a controlled Markov process where the agent is controlling it [10]. The agent chooses an action and accordingly gets rewarded for it. Q-learning uses the Markov chains to calculate the max reward that can be accumilated by the next state action and updates towards that as shown in the equation below.

$$Q(s,a) := Q(s,a) + \eta[r + \gamma \max Q(s',a') - Q(s,a)] \tag{1}$$

Here $Q(s,a)$ is the current state of the agent r is the reward, η is the learning rate and $Q(s',a')$ is the next state. An importnad variable here is γ which represent the discount factor. This is used to limit the Markov chain to a limited finite number so they don't end up infinite. This controlls how many steps into the future the agent will try ot estimate.

2.2. Genetic Algorithms

Genteic algorithms are computational models based on the concept of evolution as seen in biology. Similarly to how organisms evolve by natural selection and sexual reproduction, programs can also simulate these processes and behave in a similar fashion to organisms in order to solve a specific problem. In a general sense natural selection is the process which determins which individuals get to survive by some test of fitness. After the best fitted are selelcted the creation or reproduction of the next generation starts. Reproduction is then the method in which the mixing of genes in the remaining population happens and gets passed to the offspring [4].

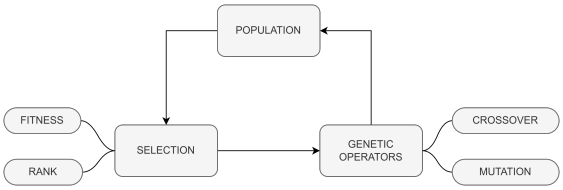

Figure 2: visual representation of genetic algorithms

By starting with a population of individuals wich are creaated randomly we have an initial population with variation amongst the individuals. The DNA wich is essentialy the code of the gene can be represented by a string of bits. These string bits can thought as potential solutions to the problem. Due to the variation in the population some individuals vill be better *fit* which then will be selected to remain. In the final stage the remaiining individuals will mix their bit strings to produce individuals for the next generation. These steps will be continually done fore some number of generations [3].

It is importnad however that we reduce the genetic drift and keep track of hte best solutions that have been produced by the previous generation. To do that we employ a method called Elitism. In elitism compared with traditional reproductiion the most fitting individual are copied to the next generation without any alteration. In that way the best solution of each generation is alway preserved and adds selective preassure and improve convergence speed [2].

3. Method

To evaluate the performance of Reinforcement Learning and Genetic Algorithms, experiments have been conducted on the simple but effective environment of the cart pole.

Cart Pole is a classic control problem in reinforcement learning. The goal is to balance a pole on a cart that can move left or right.

The state space is four-dimensional, consisting of the cart position, cart velocity, pole angle, and pole angular velocity $[p, v, \alpha, \omega]$.

The action space is discrete, with two possible actions: move left or move right.

The reward is 1 for every time step the pole is balanced.

The goal is to balance the pole for as long as possible, with a limit of 500 actions.

The environment, called *CartPole-v1*, is implemented in Python using the Gymnasium library [9].

Below described implementations have been trained using the same enviromnemt and ensuring that in every iteration the starting point is the same between two methods, but different from the previous iteration, to ensure that the comparison can be evaluated without considering the stochastic nature of the training.

3.1. Reinforcement Learning

The reinforcement learning implementation is based on temporal differenc learning [8], in particular Q-learning. The implementation takes inspiration on the work of *JackFurby* [5].

The *Q-table* is represented by the discretization of the continuous 4-dimensional state vector in 20 even intervals for every dimension of the vector leading to 160000 possible pairs of $< state, action >$, considering the two possible actions.

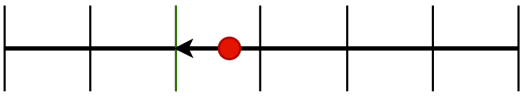

Figure 3: Representation of the state discretization technique, considering an element of the 4D-state, s_i, the red dot is the real value of s_i, this is discretized to the nearest leftwise discrete state.

Once an action is performed, the state selected is the first larger that the observed state.
The parameters used in the experiments are the following:

Learning rate α	0.1
Discount factor γ	1
Number of episodes n_{ep}	100
Exploration rate ε	variable
Mutation Rate	0.05
Penalty factor PF	-375

Table 1: Parameters used in the RL implementation. The exploration rate ε starts with $\varepsilon(0) = 1$ and decays by $\varepsilon(t) = \varepsilon(t-1) - \frac{1}{\frac{n_{ep}}{2}-1}$, every episode, stopping after $\frac{n_{ep}}{2}$ episodes.

3.2. Genetic Algorithms

Genotype
Since Genetic Algorithms can be very different depending on the genotype chosen to represent individuals, we have tried several different implementation of GA, varying the used genotype.

3.2.1. Encoding the sequence of actions

The first method used is a very naive implementation that can be applied to a very large variety of problems with GA : reprensenting individuals with the vector of all actions they will perform in order. Thus, *i-th* character of the genotype of an individual j corresponds to the *i-th* action performed by the corresponding individual. In this apporach, mutation is performed by switching an action in the genotype from left to right or from right to left with a probability given by the *Mutation rate* for every action i inside the genotype.
Since this particular genotype does not generalize well with the random initialization of the starting position of the pole. Due its intrinisc dependecy with the initial state, fixed starting conditions should be applied to effectively train in a meaningful way this genotype, by seeding the environment to always start in the same place, but this leads to a scarse ability of generalization, since the training is valid just for a determinated starting position of the pole.

All those considerations lead to the decision of evaluating other encodings for the final implementation.

3.2.2. Encoding a state-action table

The second encoding takes inspiration from Reinforcement Learning *Q-table*. In this approach, the focus is not to predict every action one by one but instead use GA to assign values to state-actions pairs then select the action that referes to the observed state.
The discretization techninque is the same used in the reinforcement learning implementation and briefly described in figure 3.
Here, mutation is performed by swapping the action of a given state with a probability given by the *Mutation rate*.

Parameters
The GA parameters can be found in the following table :

Table 2: Parameters used in the GA implementation

Genotype	Q-table
Population Size	50
Generations	200
Selection	Rank
Mutation Rate	0.05
Crossover	Uniform

4. Results

4.1. Training comparison

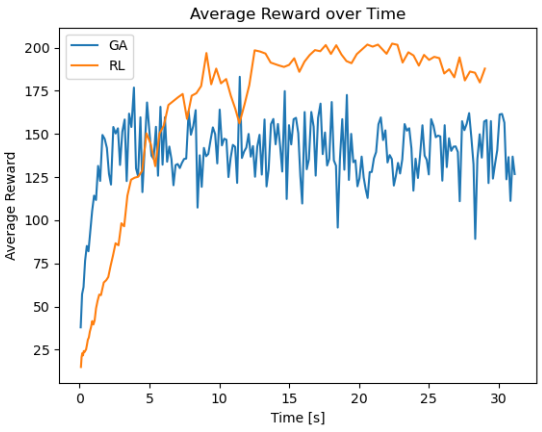

Figure 4: placeholer

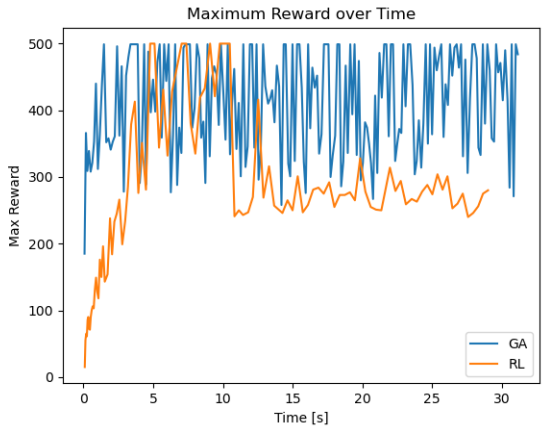

Figure 5: placeholer

4.2. final model comparison

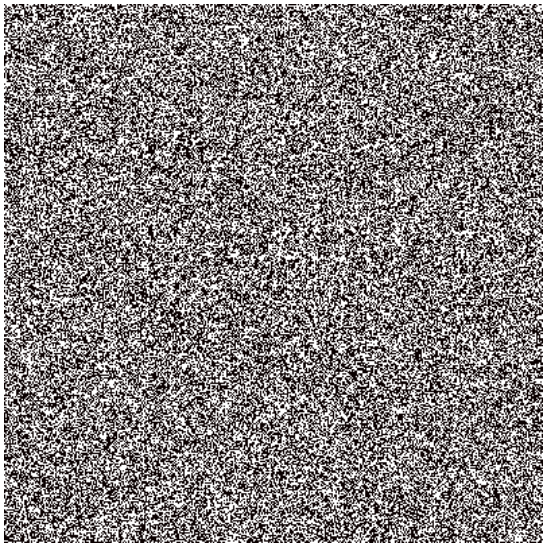

Figure 6: Difference between the two state action tables obtained using GA and RL. The black pixels represents the states where choosen action is the same, white pixels represent a different action's choice. The x-axis contains all the possible pairs of the first two elements of the state's 4D-vector, (s_1, s_2), y-axis all the possible pairs of the last two elements (s_3, s_4).

5. Related Works

6. Discussion

Originally, when only trying the action by action approach in GA, we were scared that our results might be very one sided, with very poor performances for GA and no real generalization capacities. However, it was very interesting to see that Genetic Algorithm performed much better when combining it with features from Reinforcement Learing -namely the *Q-table*.

It is still important to keep in mind that a specific thing we used in this project was the *state discretization*, which allowed us to use the affordmentioned tabular approach. Without this, we would have had to take a different and more elaborate direction. In particular, we thought of exploring the path of function approximation which is a very common approach in continuous state problems in RL, but we could not see simple adaptations to bring it to GA.

7. Conclusion

The conclusion should summarise your main results and main points from the discussion. A rule of thumb is to not present any new information (information not found in the results or discussion).

Acknowledgements

Acknowledgements (nb. takksigelser, nn. takkseiingar) is not a requirement in a laboratory report. However, it is used in most scientific articles. It looks more professional and adds some "extra spice" to your report. Here is an example:

The authors would like to thank Dr. Ola Normann at the University of Oslo for assistance with the SIMS-analysis and Dr. Kari Normann at NTNU for fruitful discussions and support concerning melt spinning of silicon. This work was financially supported by the Norwegian research council and the Norwegian PhD Network on Nanotechnology for Microsystems.

References

[1]M. M. Drugan, "Reinforcement learning versus evolutionary computation: a survey on hybrid algorithms", Swarm and evolutionary computation **44**, 228–246 (2019).

[2]H. Du, Z. Wang, W. Zhan, and J. Guo, "Elitism and distance strategy for selection of evolutionary algorithms", IEEE Access **6**, 44531–44541 (2018).

[3]S. Forrest, "Genetic algorithms", ACM computing surveys (CSUR) **28**, 77–80 (1996).

[4]J. H. Holland, "Genetic algorithms", Scientific american **267**, 66–73 (1992).

[5]JackFurby, *Cartpole-v0, github repository*, (2019) https://github.com/JackFurby/CartPole-v0 (visited on 04/25/2024).

[6]L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey", Journal of artificial intelligence research **4**, 237–285 (1996).

[7]R. S. Sutton, A. G. Barto, et al., "Reinforcement learning", in, Vol. 11, 1 (1999), pp. 126–134.

[8]R. S. Sutton and A. G. Barto, "Temporal-difference learning", in *Reinforcement learning: an introduction* (MIT Press, Cambridge, MA, 1998), pp. 32–64.

[9]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, 10.5281/zenodo.8127026.

[10]C. J. Watkins and P. Dayan, "Q-learning", Machine learning **8**, 279–292 (1992).

Appendix A. Additional Information

You can use the appendix to include information that is relevant, but does not belong in the report. In most cases however, the appendix can be omitted and isn't necessary.

Appendix A.1. Python code

If you used python code to process data, you can include the code (or a shorter version of it) in the appendix. Usually, however, it is better to hand in a separate file containing your code together with the report. [i]

Below is a simple example of some code used to calculate the values for the circuit in **??** on page ?? [ii] found in **??**:

Listing 1: Example from external file

The code from listing 1 was displayed using a `.py` file. Since the lines are not numbered in this code example, you can copy and paste the code from the PDF into python without many issues (does however need to correct indents).

I would advice against using the `lstlisting` package to display code, as this introduces many unnecessary problems when trying to copy-paste the code.

[i]Rule of thumb: short code goes in the appendix, long code goes in a separate file

[ii]exaple usage of the varioref package

Appendix B. Appendix footnotes

This template has a separate roman numeral footnote system for the appendix. You can chose to use this or normal footnotes in the appendix. Use the command `\appendixfootnote{text}`[iii] to get a (lower case) roman numerical footnote. I added this footnote system because I thought it would be nice to have a separate footnote system for the appendix, since this section is in some ways separate from the rest of the document.

Appendix C. Boxes

This template also include two box environments to highlight text. I will showcase these in the two next subsections.

Appendix C.1. Info Box

The first environment is named `infobox` and is numbered, which allows for references to the box. You can also change the title of the box as well as the colours. To change the colour use the command `\SetInfoBoxBgColor{}` (changes background colour) and `\SetInfoBoxFrameColor{NTNU_blue}` (changes frame colour). The default colours are a light blue background and a darker blue frame. Here is an example:

Infobox **Appendix C-1**

Here is an infobox. You can also write math inside it:

$$3x + 5y = 6z^2$$

Here is a reference to the infobox: box Appendix C-1. Notice that the structure of the infobox numbering is (section number)-(box number). The first infobox in section 2 thus has the reference 2-1.

Appendix C.2. Simple Box

The second environment is just a coloured box with no number or title. This can be used just to highlight text.

I also added a theorem environment `Sclaw` that may prove useful.

Scientific law 1 (Newton's 2. law).
$\vec{F} = \frac{\mathrm{d}\vec{p}}{\mathrm{d}t}$

You can reference the theorem environment: See scientific law 1. I also added a Norwegian

[iii]Note that there is **no** commands: \appendixfootnotemark and \appendixfootnotetext

version of the environment: `naturlov`. Let us change the colour of the next box to blue using `\SetSimpleBoxColor{bg_blue}`.

> To create your own theorem environment, use the command `\newtheorem{}{}[]`.

You must use the `newtheorem` command before `\begin{document}` (the preamble). You can read more about the theorem environment in the Overleaf documentation using this link: https://www.overleaf.com/learn/latex/Theorems_and_proofs.