

Comparative Analysis of Genetic Algorithms and Reinforcement Learning

Filippo Balzarini¹, Jason Kaxiras¹, Melvin Gode¹

^aDepartment 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: $R = (3.05 \pm 0.02) \cdot 10^{-6} \text{ kg/s}^2$

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 specifically 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 reinforcement 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. Reinforcement Learning

Reinforcement 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 specifically solve the task itself[6] as depicted in figure ??.

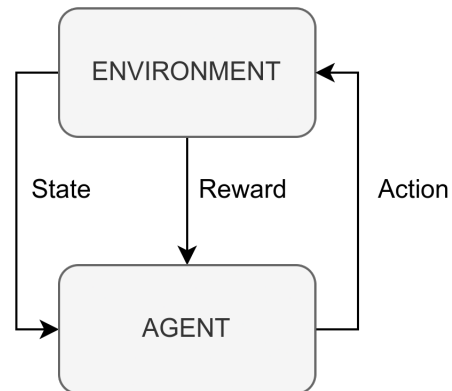


Figure 1: Graph representing reinforcement learning

The first concept crucial for reinforcement 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 beneficial 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 important is the *value* or *update* equation which is responsible for mapping the different states based 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 environment 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 accumulated by the next state action and updates towards that as shown in the equation below.

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

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 important variable here is γ which represents the discount factor. This is used to limit the Markov chain to a limited finite number so they don't end up infinite. This controls how many steps into the future the agent will try to estimate.

2.2. Genetic Algorithms

Genetic 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 determines which individuals get to survive by some test of fitness. After the best fitted are selected 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].

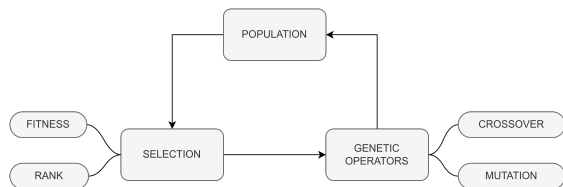


Figure 2: visual representation of genetic algorithms

By starting with a population of individuals which are created randomly we have an initial population with variation amongst the individuals. The DNA which is essentially the code of the gene can be represented by a string of bits. These string bits can be thought as potential solutions to the problem.

Due to the variation in the population some individuals will be better *fit* which then will be selected to remain. In the final stage the remaining individuals will mix their bit strings to produce individuals for the next generation. These steps will be continually done for some number of generations [3].

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

3. Method

To evaluate the performance of Reinforcement Learning and Genetic Algorithms, experiments have been conducted on different environments: Cart Pole and

Below, we describe the environments and the implementation used in the experiments.

3.1. 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].

3.1.1. Reinforcement Learning

The reinforcement learning implementation is based on temporal difference 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 40 possible pairs of $\langle \text{state}, \text{action} \rangle$, considering the two possible actions. Once an action is performed, the state selected is the first larger than the observed state.

The parameters used in the experiments are the following:

- Learning rate $\alpha = 0.1$
- Discount factor $\gamma = 0.95$
- Number of episodes $n_{ep} = 1000$
- 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.
- Penalty factor $PF = -375$

3.1.2. 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.

The first method we used is a very naive implementation that can be applied to a very large variety of problems with GA : representing individuals with the vector of all actions they will perform in order. Thus, `genotype[i]` corresponds to move number i performed by the corresponding individual. In this approach, mutation is performed by changing an action in the genotype from left to right or from right to left.

Since this particular genotype doesn't generalize well to the random initialization of the position of the pole, we have also tried this in fixed conditions, by seeding the environment to always start in the same place.

Second, we tried another GA implementation, relying on what we already were using in our Reinforcement Learning part, that is the *Q-table*. In this approach, we don't try to predict every action one by one as we did previously but instead use GA to assign values to state-actions pairs in our *Q-table* and then act greedily by selecting the best action for every state we encounter. Here, mutation is performed by adding some random noise to the *Q-table* values.

Parameters

The GA parameters can be found in the following

	enotype
	Population Size
	Generations
	Selection
	Mutation Rate
table :	Crossover
	<i>Q-table</i>
	50
	200
	Rank
	0.05
	One-point

4. Results

The results section can be combined with the discussion if appropriate. In case of many sub-experiments where the results are vaguely related or unrelated, it would be appropriate to combine the results and discussion. This way you have the information related to each sub-experiment gathered in one place.

Provide uncertainties for the results, but don't discuss it. Do not involve personal opinions, just present the cold hard results in form of numbers, tables, graphs and some sentences.

?? shows a nice table with comma alignment.

Table 1: Table with comma alignment.

m (kg)	a (m s ⁻²)	F (N)
1.2	10.1	12
2.44	6.92	16.88
10	1.0	10
8.2	1.1	9.0
100	1	100

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 Learning -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 aforementioned 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. takkseingar) 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

- ¹M. M. Drugan, “Reinforcement learning versus evolutionary computation: a survey on hybrid algorithms”, *Swarm and evolutionary computation* **44**, 228–246 (2019).
- ²H. Du, Z. Wang, W. Zhan, and J. Guo, “Elitism and distance strategy for selection of evolutionary algorithms”, *IEEE Access* **6**, 44531–44541 (2018).
- ³S. Forrest, “Genetic algorithms”, *ACM computing surveys (CSUR)* **28**, 77–80 (1996).
- ⁴J. H. Holland, “Genetic algorithms”, *Scientific american* **267**, 66–73 (1992).
- ⁵JackFurby, *Cartpole-v0, github repository*, (2019) <https://github.com/JackFurby/CartPole-v0> (visited on 04/25/2024).
- ⁶L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: a survey”, *Journal of artificial intelligence research* **4**, 237–285 (1996).
- ⁷R. S. Sutton, A. G. Barto, et al., “Reinforcement learning”, in, Vol. 11, 1 (1999), pp. 126–134.
- ⁸R. S. Sutton and A. G. Barto, “Temporal-difference learning”, in *Reinforcement learning: an introduction* (MIT Press, Cambridge, MA, 1998), pp. 32–64.
- ⁹M. 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](https://doi.org/10.5281/zenodo.8127026).
- ¹⁰C. 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 ?? ⁱⁱ found in ??:

Listing 1: Example from external file

The code from ?? 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.

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

ⁱRule of thumb: short code goes in the appendix, long code goes in a separate file

ⁱⁱexample usage of the `varioref` package

ⁱⁱⁱNote that there is **no** commands: `\appendixfootnotemark` and `\appendixfootnotetext`

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.

more about the theorem environment in the [Overleaf documentation](https://www.overleaf.com/learn/latex/Theorems_and_proofs) using this link: https://www.overleaf.com/learn/latex/Theorems_and_proofs.

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
<p>Here is an infobox. You can also write math inside it:</p> $3x + 5y = 6z^2$	

Here is a reference to the infobox: ???. 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).

$$\vec{F} = \frac{d\vec{p}}{dt}$$

You can reference the theorem environment: See ??. I also added a Norwegian 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