Comparative Analysis
of Genetic Algorithms and Reinforcement Learning

Filippo Balzarini[1], Jason Kaxiras[1], Melvin Gode[1]

[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 the are several

2. Background

2.1. Reanforcment 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[4].

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[5].

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 [8]. 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 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. Reproduction is then the method in which the mixing of genes in the remaining population happens and gets passed to the offspring [2].

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 [1].

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 [7].

3.1.1. Reinforcement Learning

The reinforcement learning implementation is based on temporal difference learning [6], in particular Q-learning. The implementation takes inspiration on the work of *JackFurby* [3].

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 $< state, action >$, considering the two possible actions. 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 $\alpha = 0.1$
- 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

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 ($\mathrm{m\,s^{-2}}$)	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

You should try to show insight into what happened and why, and how things could have gone differently. If you have presented any background theory, try to tie it together with your results. How do they relate? If they differ, try to explain why. Even if things didn't work out as intended, a good discussion shows that you've understood what went wrong and how you could potentially overcome these obstacles.

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

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

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

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

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

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

[7] 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`.

[8] 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 **??** [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 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:

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

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

[ii] exaple usage of the varioref package

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

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).
$\vec{F} = \frac{\mathrm{d}\vec{p}}{\mathrm{d}t}$

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