Finding a balance between reinforcement and evolution

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

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

May, 2024

Abstract

1. Introduction

In the field of Machine learning, several methods can be used to solve the same problem. For example, trying to find the local minima using evolutionary algorithms or supervised learning in the form of backpropagation. 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 interesting 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, in reinforcement learning the agent engages a dynamic and evolving environment by taking actions that affect it to accomplish a specific job. On the other hand, we have evolutionary algorithms that employ evolutionary principles for automated and concurrent problem-solving by drawing inspiration from populations of interacting organisms. Despite their apparent dissimilarities, RL and GA 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 [**drugan2019reinforcement**].

This paper focuses on comparing Reinforcement learning and Genetic Algorthms by having them balance a cartpole in 500 moves. More specifically it is a problem in nonlinear dynamics where an inverted pendulum is balancing in a cart. The aim or final goal of both RL and GA are to keep it the system balanced until they run out of moves. the environment will be described in more detail under the environment part.

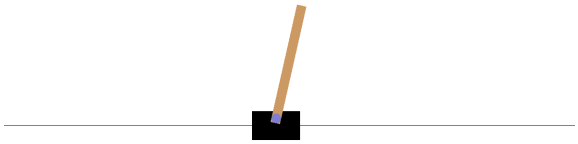

Figure 1: the cartpole in 2D graphics

Other related research on this topic includes for [**drugan2019reinforcement**] which focuses on a comprehensive overview of recent trends in the field rather than comparisons of subclasses of algorithms or particular aspects of RL and GA. Several works focus on combining these two methods for machine learning by either using GA to train RL or vice versa such as [**eiben2007reinforcement**] where the authors try to use Reinforcement learning to tune the parameters of GA. Papers such as [**khadka2018evolution** explore the opposite combination of training RL using GA . It is important to mention that the implementation of the reinforcement learning algorithm that is used is based on the work of *JackFurby* [**JackFurbyCartPole**].

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 environment. In other words programming an agent through rewards and punishments rather than how to specifically solve the task itself[**kaelbling1996reinforcement**] as depicted in figure 2.

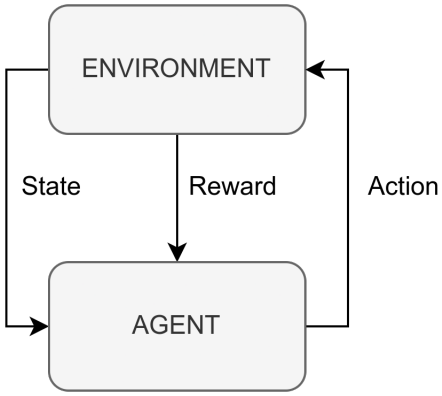

Figure 2: Graph representing reinforcement learning

The first concept crucial for reinforcement learning is the *reward function* which is objective feedback from the environment. 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 environment. The sole objective of the agent is then to maximize this reward[**sutton1999reinforcement**].

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 consideration when taking an action.

2.1.1. Temporal difference learning

A central class of methods reinforcement learning is *temporal difference learning*. It refers to a class of methods which the learning is based on the difference between temporally successive predictions. It aims to adjust the learner's current expectation for the present input pattern so that it more accuratly aligns with the subsequent prediction at the following time step. Unlike Monte carlo methods and other methods in temporal learning updates its estimated value function at every step. [**tesauro1995temporal**].

In temporal learning there are several subcmethods or rather algorithms such as SARSA ,Q learning, TD-Lambda and more [**eiben2007reinforcement**].

2.1.2. Q learning

Q learning is an algorithm where the environment can be constituted by a controlled Markov process where the agent is controlling it [**watkins1992q**]. 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.

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

Equation 1 is the *value* or *update* equation which is responsible for mapping the different states based on their estimated long term reward in Q learning. 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 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 controls how many steps into the future the agent will try ot 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 determins 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 [**holland1992genetic**].

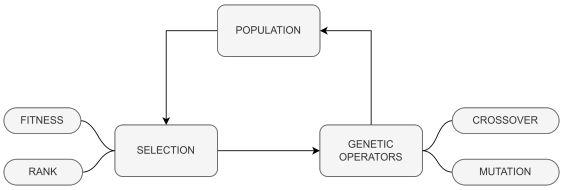

Figure 3: visual representation of genetic algorithms

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 remaining individuals will mix their bit strings to produce individuals for the next generation. These steps will be

continually done fore some number of generations [**forrest1996genetic**].

It is important 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 [**du2018elitism**].

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.

move this in background!

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 [**towers_gymnasium_2023**].
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 [**sutton1998temporal**], in particular Q-learning. The implementation takes inspiration from the work of *JackFurby* [**JackFurbyCartPole**].

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.

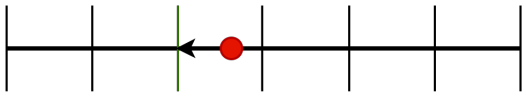

Figure 4: Representation of the state discretization technique, considering an element of the 4D-state, s_i, the red dot is the real value of s_i, this is discretized to the nearest leftwise discrete state.

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}	20000
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

3.2.1. 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 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.

The second encoding takes inspiration from Reinforcement Learning *Q-table*. In this approach, the focus is not to predict every action individually but instead use GA to assign values to state-actions pairs then select the action that referes to the observed state.
The discretization techninque mirrors that utilized in the reinforcement learning implementation and briefly outlined in Figure 4.
Here, mutation is performed by swapping the action of a given state with a probability determined by the *Mutation rate*.
In our investigation into various genotype options for addressing the pole-balancing problem, we consciously opted not to explore a solution based on NEAT[**6790655**] networks. Our decision stemmed from the inherently simplistic nature of the environment. Given our primary objective of comparing genetic algorithm (GA) approaches with reinforcement learning (RL) agents, we excluded the possibility to employ a NEAT network for such environment.

Ensuring a fair and transparent comparison between GA and RL methods is the main objective. Introducing unnecessary complexity through a NEAT network could potentially obscure the true comparative performance of the two approaches. Thus, we chose simpler solutions, aligning more closely with typical RL agent implementations. This approach facilitates a clearer evaluation of the relative effectiveness and efficiency of GA and RL algorithms in the pole-balancing task.

3.2.2. Parameters

The GA parameters can be found in the following table :

Table 2: Parameters used in the GA implementation

Genotype	Q-table
Population Size	100
Generations	200
Selection	Fitness
Mutation Rate	0.005
Crossover	one-point
Elitism	2

The one-point crossover method has been adopted, where the <state, action> pair is divided precisely at its midpoint. Consequently, for the first offspring, the initial portion of the table inherits traits from the first parent, while the latter part derives from the second parent. Conversely, the second offspring exhibits the reverse pattern, inheriting the initial traits from the second parent and the latter traits from the first.

4. Results

4.1. Training comparison

The training phase of a Reinforcement Learning agent and a Genetic Algorithm are fundamentally different. Therefore, we had to find a way to harmonize the training data of the two methods in order to compare them.
Our first idea was to consider RL episodes the same as GA individuals and aggregate the RL performances to match the number of generations used for GA. For example if we had a population size of k for our Genetic Algorithm, we would take the max and mean of every last k Reinforcement Learning episodes to compare them with each GA generation.
Due to the inherent difference between GA which performs a form of parallel search and RL which iteratively improves each episode, we decided to not alter the training data at all and take a more empirical approach. The comparison we ended up using is thus simply tracking the training time (in seconds) and plotting the performances achieved over time. (Of course both algorithms need to be ran on the same machine for a fair comparison).
Let us take a look at both average and best results achieved over training time for our two methods.

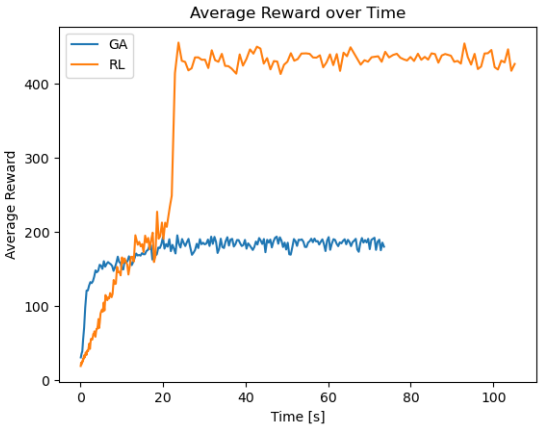

Figure 5: placeholer

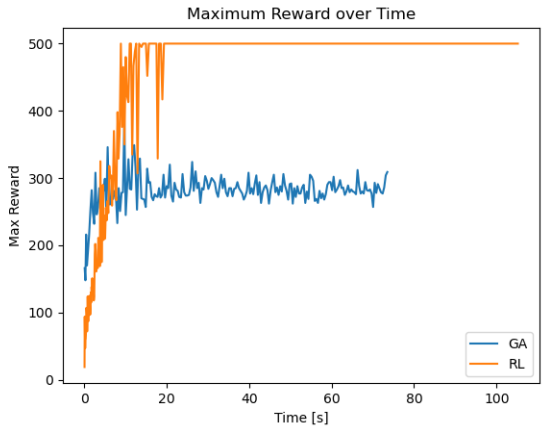

Figure 6: placeholer

As can be seen in the last two figures, the first takeaway is that RL scores much better than GA. One interesting point though, is that GA performances seem to grow quicker than their RL counterpart at the beginning of training. This can be explained by the fact that GA performs parallel search and that the genotype can be modified much more in one crossover than RL's Q-table in an episode. These two factors create more variety in the achieved solutions and by retaining the best out of them, it is easy to see how we might arrive earlier to a viable solution.
However, the other side of this described coin is that, with less stability provided by GA, comes a harder time in making specific and iterative changes to a solution. When looking at figure 5, it is very interesting to note these very similar performances between the two algorithms at 10 to 20 seconds of training, before RL jumps much higher while GA keeps plateau-ing at the same level. An explanation to the phenomenon might be that RL iteratively tweeks the same solution to perform better which allows to fix encountered problems. Meanwhile, GA might provide too much change over each generation to slowly improve an existing solution further, explaining the flattening out of performance.

4.2. final model comparison

The second approach used to compare the obtained results is based on the final models' observations.

Figure 7 shows the difference between the two obtained state-action tables of the final reinforcement learning agent and one of the individuals from the last generation of the genetic algorithm.

Since the *Q-Table* does not provide an exact chosen action given a state, the compared state-action table of the reinforcement learning agent used in Figure 7 is obtained by selecting the action a of the state s as $a = argmax_{a_i}(Q(s, a_i))$.

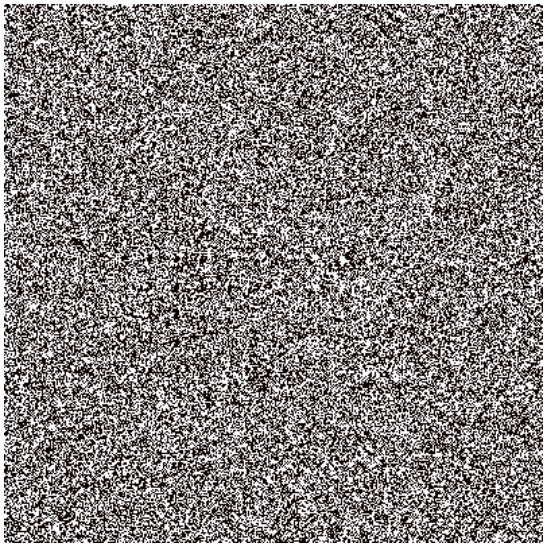

Figure 7: Difference between the two state-action tables obtained using GA and RL. The black pixels represent the states where the chosen action is the same, while white pixels represent a different action choice. The x-axis contains all the possible pairs of the first two elements of the state's 4D-vector, (s_1, s_2), and the y-axis contains all the possible pairs of the last two elements, (s_3, s_4).

Looking into Figure 7, it is possible to observe how the actions selected by the two algorithms differ in various states. However, some wide areas where the action taken by both algorithms is the same can be spotted, which could be referred to as sensitive states where considering a different choice could lead the pole to fall down.

Figure 8 shows the final testing results of the two methods. As already showed by the traning comparison in both figure 5 and figure 6, The reinforcement learning agent showed a better generalization ability and more understanding of the environment, leading to obtain better results.
The testing has been conducted as follow: the environment has been seeded with ten different seeds and for every execution the same environment has been provided to both the algorithm. The reinforcement learning agent has been tested using the obtained *q-table* from the training, the genetic algorithm population is the last generation of the training performed.
Every individual of the GA population have been tested for all the tests, and the best one has been considered for the comparison, during the testing of the population, no particular difference have been noticed between the reward obtained by the individuals.

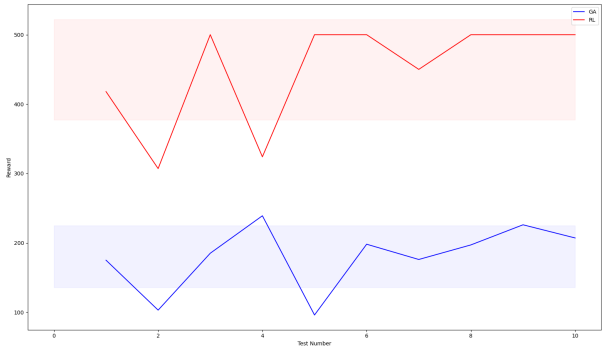

Figure 8: The plot shows the results obtained testing the two models on the same test set. The x-axis represents the number of the test set, the y-axis the number of steps the model was able to take before reaching the goal. The blue line represents the results obtained using the model trained with the GA, the red line the results obtained using the model trained with the RL.

Observing the results, some differences regarding the variance can be noticed: the GA population seem to obtain more consistent results in different tests, but due the fact that the testing is performed on the entire generation this could be related to the fact that every time the best individual is extracted.

To describe the differences between the two gifs below here :)

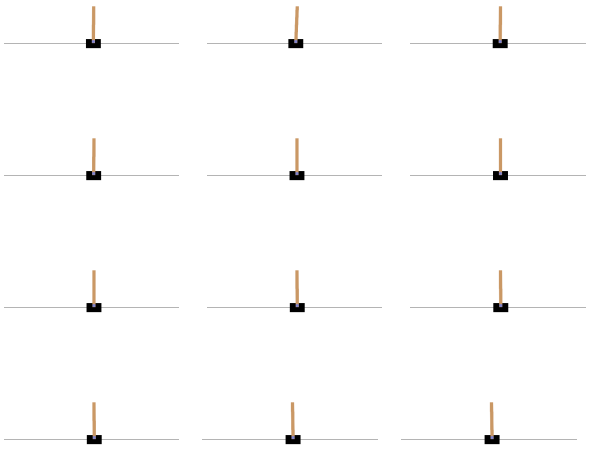

Figure 9: Frames RL

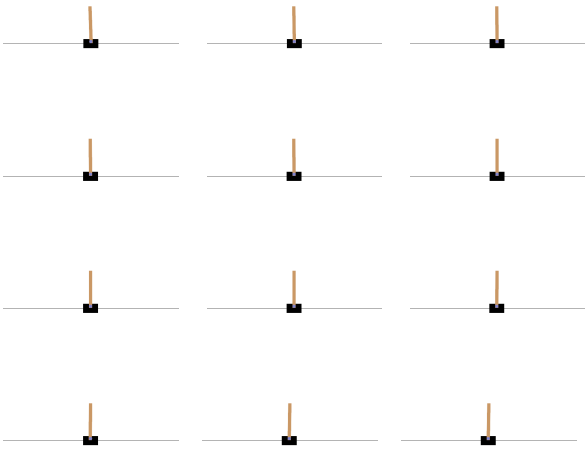

Figure 10: Frames GA

5. Related Works

6. Discussion

Originally, when only trying the action-by-action approach in GA, observations led to one-sided results due to the dependency of the genotype on the initial state, leading to poor performances for GA implementation and no real generalization capacities. However, it was very interesting to see that the Genetic Algorithm performed much better when combining it with features from Reinforcement Learning - namely the *Q-table*.

The obtained results could be possible thanks to the *state discretization*, which allowed us to use the aforementioned tabular approach. Without the state discretization approach, different and more elaborate directions should have been considered. A discussed alternative could be exploring the path of function approximation, which is a very common approach in continuous state problems in RL, but no further experiments have been conducted in this direction due to the complex adaptations of this technique to the GA algorithm.

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).