# Analysis of Genetic Algorithms as a Tool for Optimising Neural Networks to Solve Reinforcement Learning Problems

Aasav Patel, Andre du Plessis, Frank Panton, Jamie Bell-Thomas and Phil Blecher
*University of Bristol*

***Abstract*—Reinforcement learning (RL) is a highly competitive field with many established algorithms, including Deep Q-Networks (DQN), Proximal Policy Optimisation (PPO), and Advantage Actor Critic (A2C) most notably. However, genetic algorithms (GAs) are a promising tool for finding optimal policies through neural network optimization. In this paper, we propose a modified GA for optimizing neural networks in RL tasks, which is evaluated on several OpenAI Gym environments. Our modifications include the addition of random agents and decay functions to maintain diversity and investigate the explore-exploit tradeoff, as well as exploring sparse rewards and fitness sharing techniques. We compare our final algorithm to traditional RL algorithms on LunarLander, CartPole, and MountainCar, demonstrating its competitive performance. Overall, our results suggest that our modified GA is a promising alternative for modern computing in RL tasks.**

## I. Introduction

Reinforcement learning (RL) is a type of machine learning where an agent learns to make decisions by interacting with an environment to maximize a reward signal [1]. RL has shown impressive results in solving complex problems such as game playing, robotics, and autonomous driving. However, designing an effective RL algorithm involves addressing several challenges, such as the exploration-exploitation trade-off, the stability of learning, and the scalability of the algorithm. One promising approach to addressing these challenges is deep reinforcement learning (DRL), which combines deep learning and RL to enable agents to make decisions based on features learnt from high-dimensional sensory input [2]. DRL has shown great promise in solving challenging problems that were previously considered intractable.

OpenAI's Gym is a popular toolkit for developing and comparing RL algorithms [3]. It includes a collection of simulation environments, each with its own observation space, action space, and reward signal, designed to help researchers and developers test and evaluate their RL algorithms. The environments range from simple classic control problems to more complex games and robotics tasks.

In this paper, genetic algorithms (GA) are explored as an alternative method to traditional RL when training a neural network to perform well in OpenAI Gym environments. GAs are a family of metaheuristic optimisation algorithms inspired by the process of natural selection and genetics [4]. By combining genetic operators such as mutation and crossover with fitness-based selection, GAs can search large and complex solution spaces effectively. Utilising a neural network to embody the solution offers a highly expressive and flexible depiction of solutions, which can increase the efficiency of the search for optimal solutions. The performance of this approach was evaluated on several OpenAI Gym environments and compared to other known RL algorithms.

## II. Literature Review

### A. Overview of RL Methods

The realm of RL can be summarised into model based and model free methods. Model free methods rely on the action of the agent to learn a policy whereas model based methods, model the environment internally to make decisions [5]. Model based learning relies on a complete understanding of the environment's state transition dynamics, which can be difficult to acquire. As a result, model based algorithms are less able to generalise as they require significant rebuilding or re-calibrating to the new environment. Model free learning benefits from a lack of an internal framework for calculating rewards which can aid in reducing the computational complexity of the algorithm [6]. Due to this, the focus of this research is model free RL problems.

Q-learning is a form of model free learning, with one of the more recent advances being Deepmind's Deep Q-Network (DQN) [7], a form of Q-learning that extracts features from complex environments. This showcases a key benefit of the model free method as it allows the same learning algorithm to be generalised to a variety of problems. A key drawback of these methods is an inability to deal with sparse rewards as is seen through the performance of the DQN on the Atari 2600 games [7], notably the poor performance on games with notoriously sparse rewards such as 'Montezuma's Revenge'. It is this area of model-free algorithms that provides an opportunity for exploration of solutions to combat challenges faced by DQN through the exploration of less documented RL techniques, such as evolutionary methods.

### B. Comparison and Critique of Evolutionary Methods

Generally, RL algorithms estimate value functions to solve tasks, however it has been shown that these estimations are not always required [8]. Some methods ignore the state space values and apply a number of static policies over a period of time. The policies obtaining the greatest reward being carried forward to the next generation of policies. These processes are known as Iterative Algorithms and include Evolutionary Algorithms (EA). The key advantages of EA's as an alternative to solving RL problems is the efficacy when dealing with incomplete or unreliable state information as the search is conducted directly in the solution space [9]. Furthermore, when the rewards within the environment are sparse the lack of feedback can result in the RL algorithm struggling to learn [10] due to insufficient effective exploration [11], whereas evolutionary algorithms with a higher diversity often explore a wider range of solutions. These factors show the benefits of exploring EAs, and as such, three algorithms have been selected for comparison below.

*a) Simulated annealing:* is an EA where agents attempt to find the optimal policy by iteratively changing the parameters of its policy and then evaluating this against a cost function [12]. If the new policy is better than the previous, the agent accepts this policy, and if it is worse it may still be accepted based on a probability which decreases over time. This encourages thorough exploration of the search space early on in the training to increase chances of finding true global optima.

*b) Particle swarm optimisation:* is an EA that creates independent agents to simultaneously and randomly attempt to find optimal solutions in the space [13]. At each iteration, individuals record their own optima and the swarm's global optimum is updated. Although the implementation is simple, the size of the search space greatly reduces the efficacy of this algorithm, especially when applied to DNN architectures [14].

*c) GA:* is a type of EA that is used to optimize policies selection by simulating the process of natural selection [4]. The algorithm produces an initial population of solutions and evaluates their effectiveness against a fitness function. The algorithm then applies multiple operations on the best policies to create a new iteration of improved policies.

### C. Specific Benefits of Genetic Algorithms

All the assessed iterative algorithms can be applied to RL problems and effectively optimise a neural network. However GA has been selected for development on OpenAI's Gym due to . Furthermore, studies have reported the effectiveness of the GA in achieving superior performance compared to other traditional RL methods such as Q-learning and SARSA. The GA algorithm has been widely applied to solve RL problems due to its ability to solve complex and non-linear functions. For instance, GA has outperformed RL algorithms in an autonomous driving situation in terms of learning efficiency in this dynamic situation [15].

Additionally, in a recent study by Such et al, who's GA was compared against traditional RL algorithms on a variety of Atari games. The results showed that the GA was able to equal and out perform DQN, A3C and evolutionary strategies across the spectrum of games [16]. This is promising for the implementation of the GA into the OpenAI Gym environments. These findings highlight the specific benefits of the GA in solving RL tasks, which include its ability to handle complex environments, its robustness to noise, and its ability to handle high-dimensional state spaces.

## III. Methodology

### A. Environments

Within the range of environments CartPole was chosen as an initial baseline because it is a well-known benchmark and relatively simple. Following this, the algorithm was tested on MountainCar, which has a sparse rewards system. Finally, LunarLander was used as a more complex environment to test the capabilities of our control system.

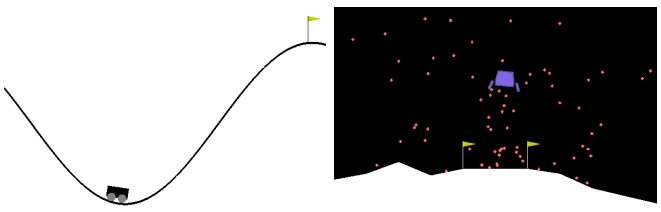

Fig. 1: MountainCar on the left and LunarLander on the right

### B. Encoding

Encoding is the process of attributing sets of values or variables to individuals in the population which can be manipulated and combined to produce new offspring. For RL tasks, this is utilised to interpret observations and predict the best action to take. A neural network is chosen as the encoding scheme, allowing for a more complex and adaptable representation of solutions.

A neural network is a computational model inspired by the structure and function of the brain, composed of interconnected nodes or neurons that process and transmit information. The architecture of the network plays a critical role in the final competency and complexity of the model. The input layer corresponds to the number of variables in the observation space, where each of these values can take any real value. The output layer contains nodes corresponding to the number of possible actions. In between these two layers, any number of hidden interconnected layers map the input to the output.

### C. Fitness Function

The fitness function evaluates the quality or suitability of an agent's solution to the problem. It assigns a fitness score to each individual, which is used to determine their probability of selection for the next generation. This function can be designed to optimize various objectives, such as maximizing particular parameters or finding the optimal combinations of parameters to achieve a specific outcome. Careful design of the fitness function is critical to the success of the algorithm, as it guides the search towards optimal solutions that meet the desired criteria.

Reward shaping is another option for improving the performance of agents in specific environments. It entails augmenting the reward sequence to allow greater logical links between game performance and reward [17]. It is difficult to implement in environments such as CartPole as these are already reward-dense. However, in a sparsely rewarded environment such as MountainCar or LunarLander, reward shaping would allow rewards to be linked to the exploration of the environment by the agent [18]. This would likely allow the agent to find the solution in a reasonable time.

### D. Selection

Selection involves picking individuals from a current population to be carried forward to the next generation. There are different approaches to selection, each with its unique benefits.

Deterministic methods like elitism choose the top-ranked individuals from a population. Meanwhile, probabilistic approaches such as tournaments and the roulette wheel add an element of randomness to the selection process, helping maintain genetic diversity and prevent premature convergence. Choosing the appropriate selection method may increase the chances of finding good solutions.

### E. Crossover

Crossover is the process of creating new solutions from the selected parent solutions. It is a key aspect of the GA in ensuring the diversity of solutions. This is achieved through sharing the genetic information of two preferred parent solutions. Partial solutions are combined which preserves good blocks of genes, leading to improved solutions. The improved offspring and the diversity generated through the large jumps in genetic information from parent to child help the algorithm escape local optima and converge towards a globally optimal solution.

The methods or crossover selected include single-point crossover, where a random splitting point is chosen and the

genomes of the parents are swapped at this point resulting in a hybrid offspring. Multipoint crossover, occurs similarly however the multiple splitting points are chosen. Finally, uniform crossover is where each gene of the offspring has been randomly carried forward from one of the parent genomes.

### F. Mutation

Mutation is a vital operation within the GA. It is used to preserve the genetic diversity of chromosomes within the population. Without mutation, the chromosome structure would converge and evolution would stagnate. Each gene within an agent's chromosome will have a mutation probability ($p_m$) associated with it. If an agent was selected from the previous generation, the mutation probability will be zero so a baseline model performance is guaranteed. Else, the mutation probability is set to a fixed global mutation rate defined in initialisation. This algorithm uses 4 main mutation methods: swap, scramble, inversion and random reset. Swap mutation interchanges individual genes. Scramble mutation will randomly rearrange blocks of successive genes. Inversion mutation will reverse the order of blocks of successive genes. Random reset mutation will change the value of a single gene to a random value within the range of the chromosome. This method creates two parameters to be explored, the first is the mutation rate and the second is the mutation type.

### G. Termination

There are two types of terminations in the GA for the game environment. This includes termination of the whole algorithm or termination states for each agent or member of the population in a certain generation. Both are conditions or criteria that have to be met to determine whether the agent or algorithm has run it's course. Some of these criteria would depend on the environment that the algorithm is trying to learn. For example, the criteria for agents in CartPole are as follows: 1) The pole falls below a certain angle, 2) The cart travels to the edge of the display, and 3) The game reaches 500 time steps. On the other hand, the criteria which terminates the whole genetic algorithm rather than the agents in each generation are more broadly applicable. The main examples are: 1) Termination after a certain number of generations, 2) Termination after a certain number of iteration or timescale, and 3) Termination after convergence or threshold.

## IV. Experimentation

### A. Varying population size

A major factor in how the GA performs is the population size of each generation (number of agents per generation). This parameter is important as it impacts the randomness and diversity of the generation. Generally, the higher the diversity the more it explores, meaning there is a higher chance of finding the optimal solution while reducing the probability of early convergence. However, as the population size is increased the time and memory complexity also increases.

Figure 2 shows that as the population size increases, not only is the GA performing better, but the variation in the agents is higher and there is less convergence. With all other factors being fixed this clearly shows having a larger population is more beneficial. However, there is a limit to how big this population can be and that is defined more specifically by


Fig. 2: Increasing population size shows a larger variety of agents and a higher average performance of each generation on the Cart Pole environment

the problem at hand. For example for simple games like CartPole (as shown in figure 2) a smaller population is better as increasing the population could lead to diminishing returns. However, for harder games like Mountain Car and Lunar Lander a larger population of about 50 performs better.

### B. Fitness Sharing

The fitness function has been developed to include fitness sharing. Fitness sharing is a form of Niching and reduces the likelihood of early convergence of the algorithm. This acts to improve the performance of the algorithm as it increases the diversity of the population and therefore improves the probability of a globally optimal solution being found [19]. The idea behind fitness sharing is that agents in a densely populated area of the solution space are penalised for their similarity to other agents. This penalty is the effective sharing of their fitness locally. This functionality provides two variable hyperparameters in the form of the maximum distance at which the sharing occurs and the factor by which the fitness of an agent is scaled.

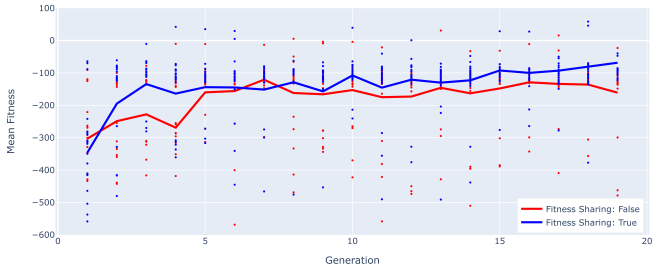


Fig. 3: Fitness sharing improves performance and prevents early convergence on the Lunar Lander environment.

Both sets of runs start similarly making quick improvements however, it can be seen that without fitness sharing, the mean fitness tends to level out and make little progress. This would signify an early convergence. With the inclusion of fitness sharing, the mean fitness continues to improve.

### C. Sparse Rewards

In Mountain Car, the car's engine is not strong enough to overcome gravity, so the agent must oscillate back and forth to

generate momentum to climb the hill. The agent only receives a reward when it reaches the finish point at the top, otherwise, it receives a penalty of -1 for every time step it isn't there.

To address the issue of sparse rewards, a novel fitness function incorporating exploration has been added to the GA. The fitness of an agent is determined by the amount of exploration of all observation dimensions, using a frontier-based approach. Specifically, a frontier is created for each observation dimension, and on entering a new state, the agent receives a reward proportional to the distance from the previous frontier. In the context of Mountain Car, the agent is rewarded for new positions and velocities. This is inspired by the general approach used in search algorithms [6], where new extreme states for each observation dimension are stored. An examination of the frontier developing over time, shown in Figure 4, demonstrates that the agent is explores the state space and the sparse reward adaptation works.

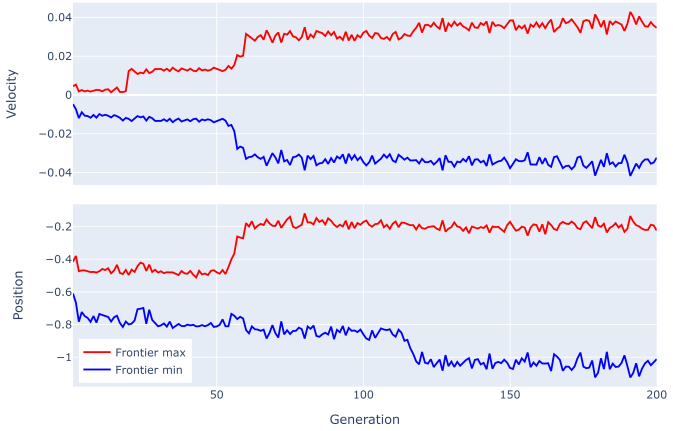


Fig. 4: Development of the frontier over generations. In the context of mountain car, this is an exploration of velocities and positions.

However, maintaining diversity in the population and preventing agents from falling into local minima pose significant challenges. To address this, the introduction of random agents is a possible solution as shown in section IV-G. This method has shown promising results and offers an alternative approach to solving reinforcement learning problems.

### D. Comparison of Selection techniques

A key feature of all selection techniques is the importance of the number of selected agents after each generation. This hyper parameter is defined before starting and impacts the convergence and performance of the population.

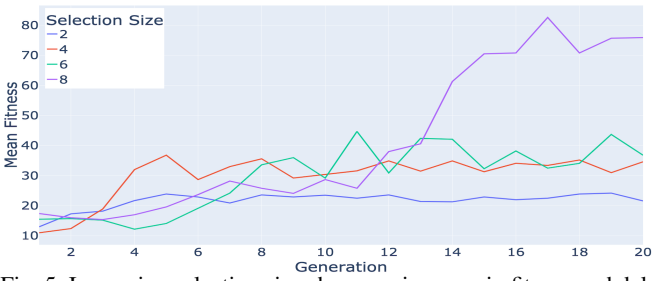


Fig. 5: Increasing selection size shows an increase in fitness and delay of convergence in the Cart Pole environment

Figure 5 shows a brief comparison of the impacts of selection size on the mean performance of agents in CartPole. All other parameters are controlled, specifically a population of 10 for 20 generations. Clearly, an increase in selection size shows both a delay in the convergence of the population and an overall increase in performance. This is expected as a greater genetic pool will reduce the risk of falling into local minima and is more likely to discover better solutions. Notably, the effect of different selection techniques was unclear for such small implementations, bar elitism which ensured generation on generation improvements.

### E. Comparison of Crossover techniques

Assessing the impact of crossover techniques, the results showed policy improvement with any type of crossover active, was far greater than one without. In the instances where the crossover rate was zero, minimal learning occurred due to the lack of diversity of agents.

Following this, a study on the impacts of varying crossover techniques and rates for the LunarLander problem was carried out. It demonstrated similar learning results for all methods of crossover and so a random selection of crossover techniques is used in the final algorithm. The results showed no distinct advantage to certain crossover rates and so a value of between 0.6 and 0.8 was used as suggested by Osaba et al [20] to allow for high variation in offspring but also carry forward a number of selected parents unedited.

### F. Comparison of Mutation techniques

*a) Varying Mutation Type:* The four mutation types adopted in this algorithm were introduced in Section III-F. Since the function of mutation is to maintain genetic diversity, the most important metric to consider is the cumulative fitness variance. Figure 6 shows this for the remaining permutations in the Lunar Lander gymnasium environment. The mutation rate was set to a low value $p_{\mathrm{m}} = 0.1$ as at higher values all chromosomes will undergo significant mutation regardless of the mutation type. The strongest in this regard was random reset mutation. This is likely due to the fact that random reset mutation is the only method that inserts new gene values into the chromosome as opposed to simply rearranging genes. However, a significant part of the variance of the random reset method is down to its ability to produce single outlying points that perform very poorly. For this reason, a combination of random reset and swap mutation will be used.

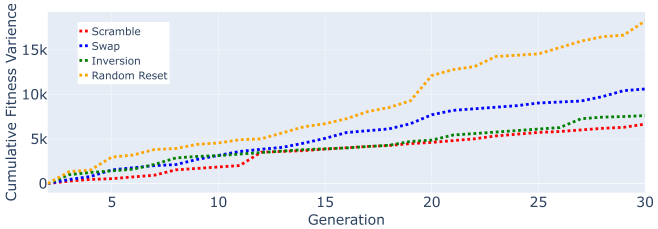


Fig. 6: Random reset mutation displayed the greatest cumulative agent variance over the course of the algorithm in the LunarLander environment

*b) Varying Mutation Rate:* The global mutation rate ($p_{\mathrm{m}}$) is the probability that non-selected genes will mutate between generations. Four values of $p_{\mathrm{m}}$ were tested for swap mutation. Figure 7 shows the average fitness score over the course of the algorithm.


Fig. 7: Variations in mutation rate ($p_{\mathrm{m}}$) showed that a rate of 0.2 produced the highest performance over the course of the algorithm for all the selected mutation types in the LunarLander environment.

The graphs illustrate the impact of mutation rate on the performance of the GA. As the mutation rate increases, the graphs become more erratic, indicating that genetic diversity is being introduced into the population through mutation. The graph at the lowest mutation rate demonstrates early convergence, while the graphs at the highest mutation rates show compromised performance due to the reduced quality of offspring produced by crossover. In contrast, a moderate mutation rate ($p_{\mathrm{m}} = 0.2$) appears to strike a balance between genetic diversity and the production of high-quality offspring. This is crucial, as the mutation process aims to prevent premature convergence by encouraging exploration of the solution space.

### G. Introduction of random agents

To mitigate premature convergence, random agents are inserted into the network to inspire exploration of the action space. After each generation, non parental agents have a probability of having their weightings randomised ($p_{\mathrm{r}}$). This is a function of the progress state such that it decreases from an initial probability ($p_{\mathrm{init}}$) to zero over the course of a run. Three decay methods were derived. The first is a linear decay, the second is exponential decay and the third is a Gaussian distribution function where the standard deviation ($\sigma$) is a constant representing the rate of decay. A fourth method was also used in which $p_{\mathrm{r}}$ does not decay. $p_{\mathrm{init}}$ was varied for each decay function. For the higher values ($p_{\mathrm{init}} = 0.8$), overall performance dropped across all methods. Furthermore, the Gaussian and exponential decay formulae unanimously outperformed the linear decay and fixed probability formulae. Figure 8 compares the remaining permutations.

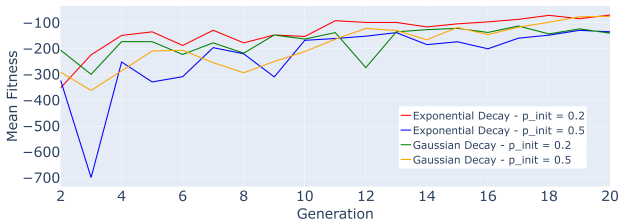


Fig. 8: The best performing functions were exponential and Gaussian decays with $p_{\mathrm{init}}$ values of 0.2 and 0.5 respectively.

Figure 8 verifies the work being done by the decay functions - demonstrating a decline in variation between generations as the algorithm progresses. The strongest performers were exponential where $p_{\mathrm{init}} = 0.2$ and Gaussian where $p_{\mathrm{init}} = 0.5$ This can be explained by the respective decay rates. The Gaussian function initially decays at a faster rate. This function supports a rapid exploration of the action space in the initial phases before quickly transitioning to exploitation for the remaining generations. When a low initial probability was used with this function, the exploration rate was too low, leading to premature convergence. The exponential function, with a more passive decay function, supports a gradual transition from exploration to exploitation. When the higher initial probability was used here, the higher exploration rate meant current solutions were prematurely abandons in favor of exploring new solutions. This can either lead to a delayed convergence. The Gaussian distribution creates a new hyper-parameter ($\sigma$) allowing for more flexibility in the decay rate of $p_{\mathrm{r}}$. This offers more control when deciding exploration and exploitation ratio. For these reasons a Gaussian distribution decay function with a $p_{\mathrm{init}}$ value of 0.5 will be carried forward.

### H. Evaluating genetic algorithm performance in Open AI Gym

*a) CartPole-v1:* The GA demonstrated its effectiveness on CartPole, it was able to confidently solve this environment in a few iterations and on some occasions faster than RL algorithms. This is because a population-based approach allows for more exploration of the search space, which is effective in discovering effective policies in simple games such as this with limited search spaces. There are few instances where the exploration is not effective in which case it can take longer to converge to a good policy, however, GA is generally a competitive method.

*b) MountainCar-v0:* Within the MountainCar environment, the GA demonstrated effective learning when sparse rewards were applied, showing increases in the algorithm's exploration into the observations space by finding new frontiers shown in figure 4. As such there were increases in rewards and therefore policy improvements alongside this. However, due to the greater complexity of the problem, a model was not developed that was effectively trained to reach the terminal states. However the incremental increases in the mean fitness are in-line with several RL algorithms in terms of policy iteration and the trend suggests it would develop comparable results over time, see V-A.

*c) LunarLander-v2:* The LunarLander environment is the most complex out of the tests due to its large number of action and observation dimensions. As a result, these data will be used for the conclusive comparison of the GA against other algorithms. This is discussed in more detail in the conclusion.

## V. Conclusion and Further Work

### A. Conclusion

*a) Comparison to Open RL Methods:* As seen in section IV-H, the face value performance of the GA on Gym environments is lower than that of methods developed by DeepMind and OpenAI. This is due to the difference in both development time and computational efficiency of those methods compared to the GA. However, this may not be a useful metric of comparison. By basing the comparison on the performance after a set number of time steps, a much more relevant metric can be calculated. The A2C, DQN and PPO algorithms used for comparison, all update their policy after each agent's episode. The GA only updates the policy after each generation. By making use of the parallelisation capabilities of GA (as explained below in paragraph V-B0a)

each policy update can be achieved at the same rate as the policy updates of the other algorithms but with the information gain from the entire population instead of just one agent.

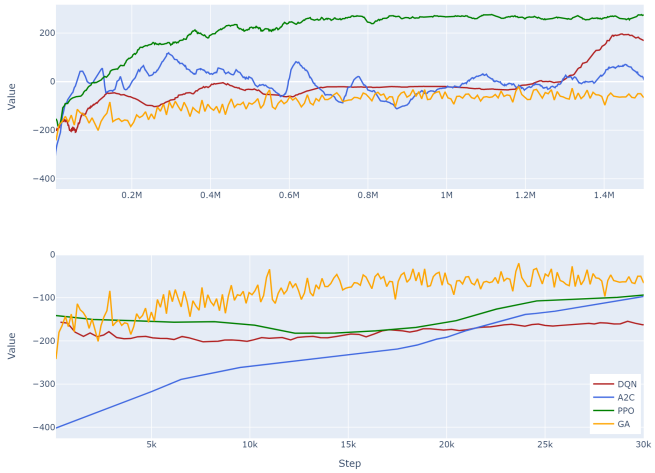


Fig. 9: Comparing the performance of GA against other methods using total time steps on the LunarLander environment. The top shows time steps taken by every agent sequentially. The bottom plot shows the timesteps run in parallel with mean values plotted

Figure 9 shows a parallel model for 50 agents (and therefore reducing the time steps by a factor of 50) the mean fitness of the agents increases at a much greater rate than that of the other algorithms. This comparison suggests that the performance of the GA can be competitive with existing methods only with greater processing power. There are many modern processors capable of this, therefore GA has the potential to outperform A2C, PPO and DQN due to their inability to parallelise.

### B. Further Work

*a) Parallelisation:* A major drawback of the GA is low sample efficiency, as progress is made through random actions. However, unlike traditional RL methods, GA does not require agents to be run sequentially. This means that parallelisation can be utilized to create great numbers of agents simultaneously, compensating for low efficiency. However, parallelisation comes with hardware requirements. To run so many versions of an agent, the user requires both multiple cores for processing and large volumes of memory to store agents. Without these resources, there may be serious limitations on the final results and capability of the model. Further work could therefore include the application of the algorithm on very large populations and large evolutionary periods, using greater computing power to determine whether it can be applied to complex environments where traditional methods also require longer learning periods.

*b) Improving Exploration vs Exploitation (EvsE):* A key feature of the GA is the number of hyper parameters available for change. Further development of this algorithm would benefit from research into the EvsE trade-off and enacting this via the decay of various parameters. The aim of this would be to encourage exploration of new states when starting a problem and increasing the exploitation as the learning progresses. In practice, decaying the effect of hyper-parameters and population size can aid to initially improve exploration and then minimise the solution space and allow the algorithm to converge. The truncation of episodes can also be used as a parameter during learning, as shorter episodes encourage explorative behaviour. By increasing the truncation length to the full episode during many generations, similar EvsE benefits can likely be achieved but more work is required to confirm these effects.

*c) Developing Architecture:* Experimentation with the architecture of the network was seen to make no discernible difference to the performance given the scale of this problem. As the scale increases, this factor will likely impact the convergence and so will be considered in future work. One method is to include the architecture as a gene base [21], optimising the architecture during training.

## References


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

[2] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[3] G. Brockman, "Openai gym," vol. abs/1606.01540, 2016.

[4] M. Mitchell, *An introduction to genetic algorithms.*, ser. Complex adaptive systems. Cambridge, MA, US: The MIT Press, 1996.

[5] M. Doody, M. M. H. Van Swieten, and S. G. Manohar, "Model-based learning retrospectively updates model-free values," *Scientific Reports*, vol. 12, no. 1, p. 2358, 2022.

[6] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and Ostrovski, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[8] C. Samuel, "Self-driving cars using genetic algorithm," *International Journal for Research in Applied Science and Engineering Technology*, vol. 8, pp. 508–511, 11 2020.

[9] D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette, "Evolutionary algorithms for reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 11, pp. 241–276, sep 1999.

[10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[11] S. Khadka and K. Tumer, "Evolution-guided policy gradient in reinforcement learning," 2018.

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.220.4598.671

[13] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948 vol.4.

[14] S. C. Nistor and G. Czibula, "IntelliSwAS: Optimizing deep neural network architectures using a particle swarm-based approach," *Expert Systems with Applications*, vol. 187, p. 115945, 2022.

[15] Z. Xiang, "A comparison of genetic algorithm and reinforcement learning for autonomous driving," p. 32, 2019.

[16] F. P. Such, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *CoRR*, vol. abs/1712.06567, 2017.

[17] Y. Hu, "Learning to utilize shaping rewards: A new approach of reward shaping," *CoRR*, vol. abs/2011.02669, 2020.

[18] S. Amin, M. Gomrokchi, H. Satija, H. van Hoof, and D. Precup, "A survey of exploration methods in reinforcement learning," *CoRR*, vol. abs/2109.00157, 2021.

[19] C. Fonseca and P. Fleming, "Genetic algorithms for multiobjective optimization: Formulation discussion and generalization," *the fifth Intl conference on Genetic Algorithms*, vol. 93, 02 1999.

[20] E. Osaba, R. Carballedo, and Díaz, "Crossover versus mutation: A comparative analysis of the evolutionary strategy of genetic algorithms applied to combinatorial optimization problems," *The Scientific World Journal*, 08 2014.

[21] M. A. J. Idrissi and Ramchoun, "Genetic algorithm for neural network architecture optimization," in *2016 3rd International Conference on Logistics Operations Management (GOL)*, 2016, pp. 1–4.