UNSUPERVISED AND REINFORCEMENT LEARNING

MASTER IN ARTIFICIAL INTELLIGENCE

Universitat Politècnica de Catalunya

ANALYSIS OF THE APPLICATION OF GENETIC ALGORITHMS TO REINFORCEMENT LEARNING

Albert Espín

April 16th, 2019

Abstract

Genetic Algorithms are an optimization technique capable of obtaining optimum or close-to-optimal results in many difficult search problems. As such, it was a natural research step to apply them in Reinforcement Learning problems, to try to find the best policy that an agent can take. Throughout decades, Genetic Algorithms and Reinforcement Learning have evolved in parallel: this paper examines this common history, starting with traditional approaches that serve as an alternative to popular Temporal Difference methods (for instance Q-Learning), with interesting capabilities such as large state generalization, aliasing robustness and adaptability to changing environments. The analysis of older approaches is key to understand the achievements of the next studied part, the recent Deep Neuroevolution model for Deep Reinforcement Learning, one of the state-of-the-art techniques in complex tasks such as playing Atari games, featuring a population of mutating neural networks with millions of parameters, stored in a compact seed-based representation. This algorithm produces faster and more accurate results than DQN, but is unable to outperform Rainbow in most tasks. Modifications in the Genetic Algorithm beyond vanilla design may lead to further improved performance.

Table Of Contents

1. Introduction

Reinforcement Learning (RL) is a machine learning paradigm in which an agent that is part of an environment tries to learn a behavior or policy (a set of actions to be performed in different situations, or states) that maximizes a notion of utility value, or cumulative reward (Sutton et al., 1998). The success of Reinforcement Learning (Dorigo et al., 1998) had been limited to problems with fully observable, low-dimensional states, until the advent of Deep Reinforcement Learning and particularly the Deep Q-Network algorithm (Mnih et al., 2015). This algorithm and a whole family of techniques based on it combine the traditional Reinforcement Learning theory with the usage of Deep Neural Networks that model complex sensory processing systems (such as image processing units), and enable agents to learn policies from high-dimensional perceptions, which makes them applicable to real-world problems, such as playing a variety of Atari games with human-level performance (Mnih et al., 2015).

An Evolutionary Algorithm (EA) is a type of algorithm in which a population of individuals representing candidate solutions to an optimization search problem undergo biological-inspired transformations (such as mutation, recombination and selection) that promote the promotion and refinement of individuals with a high value of fitness (a function related with the quality of a solution in the specific problem) throughout multiple generations, progressively leading to solutions closer to the optimum, until a convergence point (Holland, 1975). Different types of EAs exist, such as Genetic Algorithms (GAs), where the individuals, also called chromosomes, are represented as bit strings; Genetic Programming Algorithms (GPAs), where individuals are computer programs evolved to improve the performance at solving a specific task; or Evolutionary Strategies (ES), where solutions are real-valued vectors and the evolution process has a self-adaptive nature. EAs have been applied to many machine learning problems and obtained optimum or near-optimum results (Holland, 1986), which makes it reasonable to consider them as a possible approach to solve new problems.

Finding the best possible policy, a key step of Reinforcement Learning algorithms, is an optimization search problem, and therefore opts to be solved with Evolutionary Algorithms if the few applicability requirements of these algorithms are met. Particularly, there must be an appropriate representation of individuals based on the states of the search space, and a consistent fitness function (Moriarty et al., 1999).

Different types of Evolutionary Algorithms have been successfully applied to Reinforcement Learning, both in the traditional models (without using Deep Neural Networks) and in Deep Reinforcement Learning, where the usage of different types of EAs is common in the never-ending competition of researchers to design state-of-the-art algorithms for specific sets of problems, such as the Atari 2600 playing task. These works include the usage of Genetic Programming Algorithms of the Cartesian type (in which programs are indexed by grid coordinates, to react with different actions in different positions of a grid-based world) to evolve programs that play Atari games (Wilson et al., 2018) at a level that is competitive with the original DQN. Other approaches use Evolution Strategies to perform directed search in the space of parameters of the Deep Neural Networks, achieving similar performance to DQN but learning significantly faster (Conti et al., 2017, Salimans et al., 2017, Chrabaszcz et al., 2018). It is especially interesting the case of Deep Neuroevolution, the combination of Deep Neural Networks with Genetic Algorithms, that have gained a place among the state-of-the-art techniques applied to different Reinforcement Learning problems (Such et al., 2017), improving or rivaling with previous results in tasks such as Atari playing and humanoid locomotion simulations, using lower computation times than other alternatives. With these findings as motivation, this document focus on the application of Genetic Algorithms to Reinforcement Learning, first in the traditional Reinforcement Learning scenario in Chapter 2, to better understand the basics of this alliance of algorithms; next, the usage of Genetic Algorithms in Deep Reinforcement Learning is analyzed in Chapter 3, explaining the methods and strategies that allow to obtain state-of-the-art results.

2. Genetic Algorithms applied to traditional Reinforcement Learning

The application of Genetic Algorithms to traditional Reinforcement Learning was extensively analyzed in "Evolutionary algorithms for reinforcement learning", by Moriarty, Schultz and Grefenstette (Moriarty et al., 1999), the main work examined in this Chapter. Despite of the age of their study, the algorithms they described are still relevant in more recent research before the inception of Deep Reinforcement Learning (Weise, 2009), and it is also helpful to understand the later transition to Deep Reinforcement Learning (Agostinelli et al., 2018), especially when combining Deep Neural Networks with Genetic Algorithms.

An Evolutionary Algorithm for Reinforcement Learning (EARL) is a Reinforcement Learning algorithm that searches the best policy for a given problem using an Evolutionary Algorithm (Moriarty et al., 1999), usually a Genetic Algorithm. A simple EARL can be specified by defining the policy representation and its evaluation by means of the genetic fitness function. Some simple possibilities for both elements were formulated by EARL researchers (Moriarty et al., 1999) as follows, while more complex alternatives are explained later in this chapter. The easiest way to represent a set of candidate policies with a Genetic Algorithm is to use a single chromosome (an individual of the population) for each policy, with each gene (each conceptually-distinctive part of the chromosome) representing a state of the problem, and its particular allele (value of the gene) representing the action to take in that state, according to the policy. The fitness function must be able to express the long-term reward that an agent would get by following the policy, but the way to evaluate it is flexible, depending on the type of problem. In the case of episodic problems, it is common to compute the fitness function as the long-term reward of a single episode if the world is deterministic, or an average of trial rewards if it is non-deterministic. For non-episodic problems, where simulating finite trials is not suitable, it is more practical to compute the fitness function as the time required to achieve a certain accumulated reward or the value of that reward after a certain number of time steps.

There are different ways to represent policies beyond the simple state-action mapping per gene described in the previous paragraph. Namely, policies can be represented in rule-based notation or using neural networks. In both cases, they are especially used in scenarios with a large state space, that would make unfeasible having as many genes as possible states (Moriarty et al., 1999). In rule-based models, the first of which was the LS-1 system (Smith, 1983), a policy is represented as a set of condition-action mappings, where conditions are predicates that generalize different states, so that the same action can be taken in different states with some kind of determinant similar characteristic (Moriarty et al., 1999). A drawback of these systems is that conflict resolution may be needed to guarantee rule consistency, avoiding that conditions describing a same state lead to perform different actions. As mentioned, an alternative is to represent the policy as the set of weights of a neural network, to be optimized (Belew et al., 1991), with the fitness function being calculated as the reward obtained if the agent follows the policy: each input, generalizing a state, leads to activate an action in the output layer, according to the learned weights. The neural network representation has also been used in contexts without Genetic Algorithms, based on Temporal Differences (TD).

Even if the simplest way to represent policies is as single chromosomes, some algorithms consider that the whole population of chromosomes (or even more than one) forms a single policy, with each individual chromosome describing only part of the policy. One of the potential benefits of using a distributed representation of a policy is to optimize subtasks (policy components) separately, instead of trying to improve the whole task at once. This methodology allows to evolve each part of the main task of the policy in a more detailed way, and hypothetically produce a more refined final policy; another potential advantage, yet only be useful in some scenarios, is to exploit problem-specific information, such as the fact that mutually exclusive subtasks need to be learned separately (Moriarty et al., 1999). Since fitness is expressed as the reward obtained when performing a set of actions instead of the whole policy, it can be compared with multi-step TD techniques, while TD(0) exploits the extreme case of atomic subtasks, those consisting in performing a single action in a given state (Moriarty et al., 1999).

Like in the case of single-chromosome representation, a multi-chromosome policy can be represented a set of rules. A noteworthy approach in this field is the one defined in LCS system (Holland et al., 1978), where each chromosome is an if-then rule that can represent one or more states, establishing which action to take (Moriarty et al., 1999). The fittest rules are those that yield higher reward. Rules mutate and evolve competing to cover an increasing number of states of the policy while not decreasing the reward effectiveness of their actions. Other systems, such as Alecsys (Dorigo et al., 1998) use a combination of multiple LCS instances to divide the policy in different modules and improve the learning capability.

The neural network representation is also possible for a distributed policy. Some of the systems that follow this model keep multiple populations of neurons with fitness based on evaluating subsets of neurons (Potter et al., 1995). It is particularly interesting the case of the SANE system (Moriarty et al., 1996a), which uses not only a population of neurons but also a population of network blueprints: while the evolution in the first population is focused on individual neurons, considered as the building blocks of the genetic progress, the blueprint population consists in combinations of neurons that produce a greater reward, by means of the fitness function (Moriarty et al., 1999).

Other than the traditional genetic operators present in most Genetic Algorithms, such as mutation and crossover, the application of these techniques to Reinforcement Learning has brought new genetic operators. One of these operators appeared in a rule-based system with single-chromosome policy representation (Holland, 1986), consisting in the generation of a new chromosome in the population when none of the existing ones (representing rules) were able to cover the current state of the agent. The Samuel system (Grefenstette et al., 1990) showcases an alternative mutation operator that instead of changing gene values in a random way produces modifications based on previous experiences (actions performed in previously visited states akin to a new current one).

There are different factors that are considered as strengths of EARLs, with three notable advantages listed by EARL researchers as follows (Moriarty et al., 1999). Firstly, it is feasible to apply EARLs in problems with a large state space due to their ability to generalize, using representations that summarize multiple states for which the same action should be used, instead of explicitly working at the level of state-action pairs. Secondly, since EARLs evaluate policies in a global way, as a sequence of actions from a start to an end, they are less vulnerable than TD methods to problems with perceptual aliasing or ambiguous states. Whilst basic TD approaches focus on the reward of individual actions instead of considering the long-term performance of decisions, EARLs can have more perspective and penalize policies with the poorest performance. Lastly, unlike standard TD methods, EARLs are robust to changing environments, in which the moving target problem appears while searching the best policy, since the population can progressively adapt to the alterations: the fitness function will reward with greater chance of survival those policies that produce more reward in the current setting.

Despite of the mentioned benefits, EARLs also have a set of limitations (Moriarty et al., 1999). First, EARLs take longer time than other algorithms to learn a good policy in online settings, in which the agent has to learn a policy and, in parallel, perform the actions in the world. This situation happens because the policy of each chromosome in the population (which may be considerably large) must be evaluated to determine its fitness in each epoch (or generation) of the genetic algorithm, to be able to determine which are the fittest individuals, select parents and generate offspring. The second problem is derived from the fact that the representation of policies is selective, in the sense that those policies that produce high reward are preserved in the evolution process, while the less useful ones, as well as uncommon states, are eventually forgotten. Therefore, when a rare state is encountered again the agent does not have prior information of the most profitable action to take (according to the current policy). This issue does not happen in TD methods, that store information about each state-action pair every time that they are evaluted, no matter their frequency.

The EARL researchers (Moriarty et al., 1999) conclude that both EARL and TD approaches should be seen as complementary techniques instead of mutually exclusive. Furthermore, they explain that some of the previously mentioned systems have taken advantage of part of the benefits of both families of techniques, such as Alecsys, but also Samuel (Grefenstette et al., 1990), which borrows the TD(0)-like focus in each individual action by assigning a certain strength to each gene, computed from its contribution to the long-term reward, and used as a tie-breaker in situations in which different actions seem equally suitable at a given state. This feature allows the system to have a balance between short and long-term reward evaluation, a type of compromise that can also be achieved with multi-step TD methods.

3. Genetic Algorithms applied to Deep Reinforcement Learning

The main article analyzed in this Chapter is "Deep Neuroevolution: Genetic Algorithms are a competitive alternative for training deep neural networks for reinforcement learning" (Such et al., 2017), which presents state-of-the-art result in two common Reinforcement Learning tasks in the last years, playing Atari games and humanoid locomotion, using Deep Neural Networks evolved with Genetic Algorithms, a combination named as Deep Neuroevolution. Nevertheless, this Chapter also mentions other approaches of the Deep Reinforcement Learning field, to situate the works in their competing context.

The work on Deep Neuroevolution for Reinforcement Learning took place after other researchers had outperformed the results of the original Q-learning-based DQN algorithm (Mnih et al., 2015) using Evolutionary Strategies instead of backpropagation when learning the best policy (Conti et al., 2017, Salimans et al., 2017), achieving results rivaling those of backpropagation, and additionally being considerably faster. Despite of these innovations, Evolutionary Strategies have something in common with backpropagation: both approaches are gradient-based optimization techniques, which made researches raise the question on whether gradient-free methods, such as Genetic Algorithms, would be able to perform well in

Deep Reinforcement Learning tasks, such as playing Atari 2600 games, which lead to the design and implementation of the Deep Neuroevolution approach (Such et al., 2017).

In the Genetic Algorithm used in Deep Neuroevolution (Such et al., 2017), the population consists of a group of individuals, each of them representing a set of neural network weights, and the fitness function is the reward obtained by using those weights to make the neural activations react to the sensory input of each visited state with a certain action, thought of as the best choice. This representation of the population resembles the much older single-chromosome-policy network-weight models (Belew et al., 1991) explained in Chapter 2, with the distinction that the current architecture does not use Shallow Neural Networks, but Deep Neural Networks, where the total number of hidden layers and especially the quantity of neurons per layer is significantly higher. A fixed number of individuals with the highest achieved fitness (reward) are selected as parents for the next generation, and mutated with additive Gaussian noise. The absolute best individual of a generation is not simply the one with the highest base fitness, but the one among the best 10 that achieves the highest average reward throughout 30 additional experiments, to mitigate the influence of noisy experiences, that may represent a stroke of luck instead of a consistently solid strategy. An unchanged copy of that best individual is kept in the next generation, to guarantee that the best solution found so far is kept and subsequently exploited, while exploration is kept with the mutations that generate other modified individuals. The Genetic Algorithm of Deep Neuroevolution (Such et al., 2017) does not implement any kind of crossover (combining genetic information of multiple individuals to produce offspring), allegedly for simplicity, and updates the population for a fixed number of generations or until early stopping criteria is met, such as invariability of the best reward during a reasonably long number of generations.

One of the interesting innovation of Deep Neuroevolution is the way how individuals are stored. Representing each chromosome as a real-value vector with the values of every weight, as done in traditional approaches, would be extremely memory-intensive, taking into account that the networks are very large, of the order of 4 milions of parameters. A more efficient technique, that the researchers consider key in the feasibility of making the algorithm distributed, is to represent each individual as an initialization seed plus a list of the random seeds used in the mutations that an individual has undergone through generations, which allows to reconstruct the large set of weights from very compact information. The authors consider that this is the state-of-the-art when it comes to storing Deep Neural Networks evolved with Genetic Algorithms (Such et al., 2017).

The Deep Neuroevolution researchers compared the speed of running a distributed Deep Reinforcement Learning algorithm with Genetic Algorithms and Evolution Strategies based on Salimans' approach (Salimans et al., 2017), and found that the Genetic Algorithms technique is faster. The justification is related to two aspects that are necessary in Evolution Strategies and make them slower: firstly, ES needs to update the weight vector in each generation based on a complex average computation of the fitness of pseudo-offspring; secondly, to promote the diversification of policies batch normalization is required in ES, which implies performing more forward passes, while the random mutations in GAs are enough to keep diverse-enough policies in the tested settings (Such et al., 2017). According to the study, the GA-based DNN can be trained for the Atari task in roughly 4 hours in a single 48-CPU-core modern desktop, or about 1 hour if computation is distributed through 720 cores.

The experiments of Deep Neuroevolution's Genetic Algorithm (Such et al., 2017) in the Atari playing task attempt to be fair with the original DQN (Mnih et al., 2015), by using the same data preprocessing and convolution-based deep network architecture (the larger among the options presented in DQN). The results are comparable to DQN, A3C and ES algorithms in 3 games, lower than A3C in 4 games, while GA is better than any other algorithm up to date (at least to the knowledge of the authors) in the particular game of Skiing, even outperforming all variants of Rainbow (Hessel et al., 2017), the multi-improvement version of DQN considered to be its state-of-the-art successor. Nevertheless, Rainbow still performs better than GA in most tasks. The experiments also show that random-based search can outperform DQN in some games, such as Frostbite, and that GA is capable of finding better policies than the final DQN policy in just tens of generations for certain games, such as Skiing. It is specified that in many other games, however, GA performs worse, but without specifying which are those actual games. In humanoid locomotion, the GA method is almost as successful as ES, but the authors are surprised about the fact that it is not unable to outperform ES, taking into account the history of GA being apparently well-suited for robot control tasks (Such et al., 2017).

The algorithm presented so far is a vanilla Genetic Algorithm applied to Deep Neural Networks for Reinforcement Learning, but the Deep Neuroevolution authors also designed an alternative algorithm for especially deceptive problems, those in which it is very easy for agents to get stuck in local minima if focused on guiding search using a reward-based fitness function. This algorithm uses the Novelty Search (NS) strategy for Genetic Algorithms (Lehman et al., 2011a), in which the fitness is not measured as the reward but as the novelty measure of the individual (a policy in this case), computed as the average of the distance to a fixed number of top nearest neighbors (based on problem-specific similarity criteria). Therefore, this strategy promotes trying many new solutions, focusing on exploration rather than the exploitation of best-so-far policies. The algorithm performs remarkably well in the "Image Hard Maze", a problem of similar characteristics to the Atari games but specifically design to trap agents in local minima (Such et al., 2017). This algorithm is referred to as GS-NA in the paper, and should be seen as a secondary alternative to the main reward-based algorithm, that performs better in most of the problems. The authors encourage the research community to try to apply other classical improvements or variants of Genetic Algorithms in the Deep Reinforcement Learning context, to discover whether they perform better than vanilla GA implementations. After all, state-of-the-art DNN algorithms for Reinforcement Learning, such as Rainbow (Hessel et al., 2017), are the result of combining many improvements to the original algorithm.

4. Conclusions and Future work

The usage of Genetic Algorithms to learn the policy in Reinforcement Learning is a viable gradient-free alternative to the typically used gradient-based approaches, such as Q-Learning and Evolution Strategies. Twenty years ago, Genetic Algorithms were already an interesting option with some advantages over Temporal Difference methods, such as having the ability to scale up to large state spaces by generalizing states to rule-based representations or as or neural network weights. Genetic Algorithms proved to be more robust to perceptual aliasing due to their long-term computation of reward, and evolution made policies able to adapt to changing environments (Moriarty et al., 1999). Systems such as Samuel (Grefenstette et al., 1990) were able to incorporate some TD aspects in a Genetic Algorithms to take advantage of the immediate reward evaluation in circumstances where long-term estimation was ambiguous.

After the arrival of Deep Reinforcement Learning, Genetic Algorithms combined with Deep Neural Networks, also known as Deep Neuroevolution (Such et al., 2017), have been able to rival the results of Q-Learning-based approaches such as DQN (Mnih et al., 2015), and also Evolution Strategies (Salimans et al., 2017). In particular, Genetic Algorithms have obtained better results in tasks such as Atari 2600 playing, in a significantly lower time and space thanks to computational optimizations, such as a random-seed based storage of network weights. Nevertheless, the results of Deep Neuroevolution are not able to outperform the multi-improved DQN state-of-the-art successor Rainbow (Hessel et al., 2017) in the majority of the tested tasks. Since Deep Neuroevolution uses a vanilla Genetic Algorithm, it is reasonable to think that hypothetical modified versions using improved genetic techniques may be able to be more successful. Among the interesting techniques that future researches may test, Multi-dimensional Archive of Phenotypic Elites may be especially helpful, since it is an approach that maps the fittest candidate solutions to a multi-dimensional feature space to try to extract insights of which factors make them be the best, to refine the search in next iterations (Mouret et al., 2015). There is also extensive literature on design recommendations for Genetic Algorithms, both generic and applied to certain families of problems, so that their characteristics can be exploited in the optimization step, to maximize the algorithm's performance (Haupt et al., 2004).

5. References

[Agostinelli et al., 2018] F. Agostinelli, G. Hocquet, S. Singh, & P. Baldi, From Reinforcement Learning to Deep Reinforcement Learning: An Overview, in *Braverman Readings in Machine Learning, Key Ideas from Inception to Current State* (pp. 298-328), Springer, Cham, 2018.

[Barto et al., 1990] A. G. Barto, R. S. Sutton, & C. J. C. H. Watkins. Learning and sequential decision making, in *Learning and Computational Neuroscience*, 1990.

[Belew et al., 1991] R. K. Belew, J. McInerney, & N. N. Schraudolph, Evolving networks: Using the genetic algorithm with connectionist learning, in *Artificial Life II Reading, Addison-Wesley*, 1991.

[Chrabaszcz et al., 2018] P. Chrabaszcz, I. Loshchilov, and F. Hutter, Back to Basics: Benchmarking Canonical Evolution Strategies for Playing Atari, 2018.

[Choromanski et al., 2018] K. Choromanski, M. Rowland, V. Sindhwani, R. E. Turner, and A. Weller, Structured Evolution with Compact Architectures for Scalable Policy Optimization, in *35th International Conference on Machine Learning*, 2018.

[Conti et al., 2017] E. Conti, V. Madhavan, F. P. Such, J. Lehman, K. O. Stanley, and J. Clune, Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents, Dec. 2017.

[Dorigo et al., 1998] M. Dorigo, & M. Colombetti, Robot Shaping: An Experiment in Behavioral Engineering, *MIT Press*, 1998.

[Eiben et al., 2003] Eiben A. E., Smith J. E., and others, Introduction to evolutionary computing, *Springer*, vol. 53, 2003.

[Gangwani et al., 2018] T. Gangwani, C. Science, J. Peng, and C. Science, Policy Optimization by Genetic Distillation, in *ICLR 2018*, 2018.

[Green et al., 2019] M. C. Green, B. Sergent, P. Shandilya, and V. Kumar, Evolutionarily-Curated Curriculum Learning for Deep Reinforcement Learning Agents, Jan. 2019.

[Grefenstette et al., 1990] J. J. Grefenstette, C. L. Ramsey & A. C. Schultz, Learning sequential decision rules using simulation models and competition, *Machine Learning*, 5, 355-381, 1990.

[Haupt et al., 2004] R. L. Haupt & S. E. Haupt, Practical genetic algorithms, 2004.

[Hessel et al., 2017] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, Rainbow: Combining improvements in deep reinforcement learning, 2017.

[Holland, 1975] J. H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence, *University of Michigan Press*, 1975.

[Holland, 1986] J. H. Holland, Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems, in *Machine Learning: An Artificial Intelligence Approach*, Vol. 2, *Morgan Kaufmann*, 1986.

[Holland, 1992] J. H. Holland, Genetic algorithms, *Scientific american*, 1992.

[Holland et al., 1978] J. H. Holland, & J. S. Reitman, Cognitive systems based on adaptive algorithms, in *Pattern-Directed Inference Systems, Academic Press*, 1978.

[Lehman et al., 2011a] J. Lehman and K. O. Stanley, Abandoning objectives: Evolution through the search for novelty alone, *Evolutionary Computation*, 2011a.

[Mnih et al., 2015] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and others, Human-level control through deep reinforcement learning, *Nature*, 2015.

[Moriarty et al., 1999] D. E. Moriarty, A. C. Schultz, & J. J. Grefenstette, Evolutionary algorithms for reinforcement learning, in *Journal of Artificial Intelligence Research*, *11*, 241-276, 1999.

[Moriarty et al., 1996a] D. E. Moriarty & R. Miikkulainen. Efficient reinforcement learning throughsymbiotic evolution, *Machine Learning*, 22 , 11-32, 1996a.

[Mouret et al., 2015] J. Mouret and J. Clune, Illuminating search spaces by mapping elites, 2015.

[Niekum et al., 2010] S. Niekum, A. G. Barto, and L. Spector, Genetic Programming for Reward Function Search, *IEEE Trans. Auton. Ment. Dev.*, vol. 2, no. 2, pp. 83–90, Jun. 2010.

[Pathak et al., 2018] K. Pathak, & J. Kapila, Reinforcement Evolutionary Learning Method for self-learning, 2018.

[Potter et al., 1995] M. A. Potter, & K. A. De Jong, Evolving neural networks with collaborative species, in *Proceedings of the 1995 Summer Computer Simulation Conference*, 1995.

[Salimans et al., 2017] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, Evolution Strategies as a Scalable Alternative to Reinforcement Learning, Mar. 2017.

[Schultz et al., 1992] A. C. Schultz, & J. J. Grefenstette, Using a genetic algorithm to learn behaviors for autonomous vehicles, in *Proceedings of the AiAA Guidance, Navigation, and Control Conference,* 1992.

[Smith, 1983] S. F. Smith, Flexible learning of problem solving heuristics through adaptive search, in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pp. 422-425, *Morgan Kaufmann*, 1983.

[Such et al., 2017] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, & J. Clune, Deep Neuroevolution: Genetic Algorithms are a competitive alternative for training deep neural networks for reinforcement learning, 2017.

[Sutton et al., 1998] R. S. Sutton, & A. Barto, Reinforcement Learning: An Introduction, *MIT Press*, 1998.

[Watknis et al., 1992] C. J. Watkins, & P. Dayan, Q-learning, *Machine learning*, 1992.

[Weise, 2009] T. Weise, Global optimization algorithms-theory and application, *Self-Published Thomas Weise*, 2009.

[Wilson et al., 2018] D. G. Wilson, S. Cussat-Blanc, H. Luga, and J. F. Miller, Evolving simple programs for playing atari games, in *Proceedings of the Genetic and Evolutionary Computation Conference,* 2018, pp. 229–236.