Using Deep Neuroevolution to train Deep Reinforcement Learning Agents

Muhammad Salik Nadeem
Faculty of Science
Ontario Tech University
Oshawa, Ontario
muhammadsalik.nadeem@ontariotechu.net

***Abstract*—In this project I have implemented a Deep Genetic Algorithm (GA) to train the neural network of a reinforcement learning agent. The algorithm shows promising results for DL Agent training as it converges to optimum performance in very few generations. Traditionally, Neural Networks are trained using a gradient based back propagation step. This approach evolves the weights of the network using a simple, gradient-free, population based GA. The experiments were performed on relatively simple examples which takes just a few minutes to train but the underlying algorithm is capable of running on Deep Convolutional Networks with millions of parameters as well.**

***Index Terms*—Genetic Algorithms, Reinforcement Learning, Deep Learning, Neural Networks**

I. Introduction

Reinforcement learning (RL) is a class of Machine Learning algorithms (Figure.1) which is different from supervised and unsupervised learning. In supervised learning, we have a set of richly labelled data, where for each training example, a correct prediction or label is already defined. These labelled examples are used as a metric for directly evaluating supervised learning algorithms. Unsupervised learning is about finding patterns and structures in data. It requires no labels.

Reinforcement learning on the other hand, works on a single scalar value which generally quantifies how good an agent's most recent action is in an environment. It is the sum of rewards from all actions taken by the agent over time in that environment. This number determines how adept the agent is in that environment. It is generally very hard to directly correlate this scalar reward value with a specific sequence of actions which will culminate in that reward value.

II. Background

A. Reinforcement Learning

Reinforcement Learning is a very broad area in Machine Learning. Like with supervised learning, RL has also seen massive growth and popularity when paired with Deep Learning models. The representational power of Deep Neural Networks has made it possible to train very complex RL agents which can learn from just pixel values and achieve remarkable results.

RL algorithms can be broadly categorized in two major categories, model-based and model-free. Regardless of the type all RL environments must be modelled as a Markov Decision Process (MDP) (Figure.2). This MDP formalizes how an agent interacts with an environment. Every RL algorithm also has some core elements such as Policy, Reward Signal and a Value function. Lastly there is a fourth optional element which is the Environment Model, which depends on weather the problem is model-based or model-free.

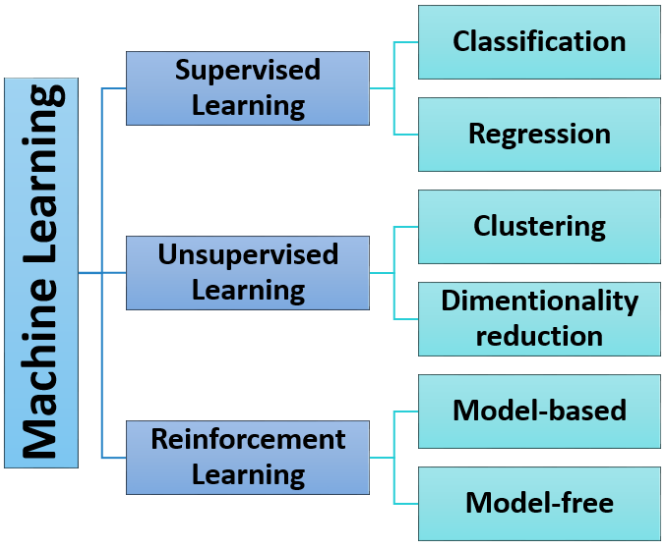

Fig. 1. Types of Machine Learning

- A **Policy** refers to the part of an agent directly responsible for deciding on the action to take. It can be an approximator that is learned (i.e using a neural network), or a program that is made to follow a set of rules or heuristics based on the state of the environment.
- A **Reward Signal** is a scalar value the agent receives every time step after taking an action. It is a function of both the current state, as well as the selected action, and it quantifies how good the agent's last action was.
- A **Value Function** is used to evaluate the long-term behaviour of an agent in an environment. Given a policy and a starting state for an environment, the value function can be used to predict the long-term total reward of the given policy in that environment.
- An **Environment Model** is only used by some classes

of RL algorithms. It is a component that can simulate or approximate the environment an agent is acting in. If an agent has an existing environment model it can make use of, the model can be leveraged to predict an environment's subsequent state, and a reward value, given the current state, and an action. This is powerful because this environment model can be used by the agent for planning before it takes an action in the actual environment.

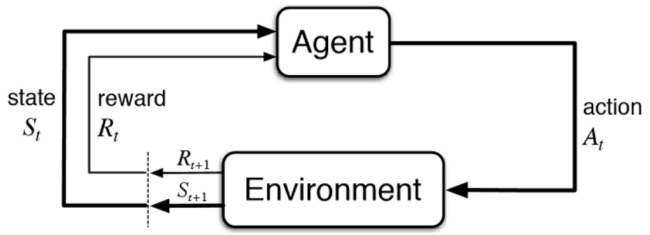

Fig. 2. A Markov Decision Process. Formal representation of how an agent interacts with the environment.

B. Deep Reinforcement Learning

Deep Reinforcement Learning (Deep RL) [1] is a recent area which leverages the principles and tools of Deep Learning and applies them on challenging problems in RL [2]–[4]. Deep RL has been able to tackle problems with a wide range of complex decision making tasks that were previously not solvable for a machine. This combination of RL with Deep Learning, is most useful in problems with high dimensional state-space. Previous RL approaches had a difficult design issue in the choice of features [5], [6]. Deep RL is able to learn problems through pixel values using Convolutional Neural Networks which has lead to smart AI systems capable of playing games at beyond human level expertise [7]. Deep RL has seen tremendous success in the last few years with case studies such as the first AI GO player beating the human world champion [8]. Deep RL was also applied to play popular competitive e-Sports games such as Dota 2 [9] and Starcraft 2 [10].

C. Genetic Algorithms

Genetic Algorithms (GA) belong to a category of the Meta-heuristic search family of algorithms. GA was invented by John Holland and developed this idea in his book "Adaptation in natural and artificial systems" in the year 1975 [11]. Holland proposed GA as a heuristic method based on "Survival of the fittest". GA was discovered as a useful tool for search and optimization problems [12]. GAs use principles of natural selection to search for optimal solutions in a search space. They have shown to be very efficient even when the search space is very large, complex or even high dimensional. There is a population of individuals, only the best candidates are chosen to reproduce using some genetic operations and the poor solutions are discarded.

III. Related Work

The field of Deep RL is a relatively new one. It is less than a decade old. There are three broad families of algorithms which have shown promise on RL problems:

- **Q-learning methods**
- **Policy gradient methods**
- **Evolution Strategies (ES)**

Q-Learning methods try to approximate the optimal Q function value with DNNs, generating policies that for a given state chose an action which tries to maximize the Q value. DQN is an example of this approach [7]. Policy gradient methods directly encode the action probability response to a given state input through DNN representation. Asynchronous Advantage Actor-Critic (A3C) framework [13], trust region policy optimization (TRPO) [14], Proximal Policy Optimization Algorithms [15] and deep deterministic policy gradient (DDPG) [16] are all examples of policy gradient based algorithms. Evolutionary strategies try to optimize the parameters of a DNN in order to maximize the output cumulative reward. Salimans et al [17] developed ES-based algorithms using massive parallelization to make them scalable and competitive to Q-learning and Policy Gradient based approaches. ES provide a competitive alternative to traditional RL algorithms and only require a fraction of the time to train.

All of these approaches above have an element of gradient calculation or approximation which makes them computationally expensive. Q-Learning approaches calculate the gradient loss of the Q-function via back-propagation calculation on the DNN. Policy gradient based approaches samples behaviours stochastic-ally from the current policy and then calculate gradients to improve the performance. ES uses an approximation similar to finite difference to optimize the algorithm which is an approximate to gradient calculation.

The method proposed in this paper is independent of any gradient calculations. Only Gaussian perturbations are applied to the parameters space to optimize the results. This drastically decreases the computational complexity of the algorithm and achieves good results in very short times.

IV. Methodology

For this project I have implemented a Genetic Algorithm based approach to train the parameters of Deep RL models. The GA implementation can be used to update the parameters of a simple feed forward Neural Network without the need for the back propagation algorithm, meaning it is gradient free. The Network layers can be Convolutional or Fully Connected layers. The model is built for RL agents based on the Open AI Gym environments in Python programming language. Open AI Gym was chosen for its relatively straight forward configuration, support for python language and its stable and diverse set of environments to test various RL and Deep RL approaches.

A. A Simple Genetic Algorithm

For the core optimization algorithm, I have used the simple Genetic Algorithm as described by Such et al. [18]. The GA

has a population of N individuals. Each individual chromosome is a set of all the parameters of a DNN, also called the parameters vector θ. At every generation, each θ_i is evaluated by running the environment with that NN, which produces a reward or a fitness score $F(\theta_i)$. This is our fitness function. Because of the stochastic nature of the RL environment we run this experiment x number of times (for my experiments I used $x = 3$) and the average score is taken for these runs to get the fitness value of that parameter vector θ_i. Once we have the fitness of all the individuals we apply a truncation selection where we only select the top T individuals. We then only used these T individuals to produce the next generation of children. For the reproduction process only the following mutation operation is used: A parent is selected from the set T of top individuals with replacement and is mutated by applying an addative Gaussian noise to the parameter vector such that $\theta' = \theta + \sigma\epsilon$ where $\epsilon \sim \mathcal{N}(0, I)$. This process is repeated $N-1$ times. The lat N^th individual is the individual with the best fitness function which is used from the previous generation. This technique is called elitism. In order to ensure reliable optimization, the top T individuals are evaluated an additional $x = 3$ number of times. The one with the highest score is called the elite individual in that generation.

Algorithm 1 Simple Genetic Algorithm

Input: mutation function ψ, population size N, number of selected individuals T, policy initialization routine ϕ, fitness function F.
for $g = 1, 2..., G$ generations **do**
  for $i = 1, ..., N-1$ in next generation's population **do**
    if $g = 1$ **then**
      $\mathcal{P}_i^{g=1} = \phi(\mathcal{N}(0, I))$ {initialize random DNN}
    else
      $k = \text{uniformRandom}(1, T)$ {select parent}
      $\mathcal{P}_i^g = \psi(\mathcal{P}_k^{g-1})$ {mutate parent}
    end if
    Evaluate $F_i = F(\mathcal{P}_i^g)$
  end for
  Sort $\mathcal{P}_i^g$ with descending order by F_i
  if $g = 1$ **then**
    Set Elite Candidates $C \leftarrow \mathcal{P}_{1...10}^{g=1}$
  else
    Set Elite Candidates $C \leftarrow \mathcal{P}_{1...9}^g \cup \{\text{Elite}\}$
  end if
  Set Elite $\leftarrow \arg\max_{\theta \in C} \frac{1}{30} \sum_{j=1}^{30} F(\theta)$
  $\mathcal{P}^g \leftarrow [\text{Elite}, \mathcal{P}^g - \{\text{Elite}\}]$ {only include elite once}
end for
Return: Elite

Fig. 3. This is the original author's algorithm for the simple GA used in my project.

B. Deep Neural Network Representation

The key architecture used in the learning process is a simple fully connected feed forward neural network. For the experiments performed mostly, just a single hidden layer was enough to achieve good results. The number of neurons in the hidden layer were adjusted based on empirical results. The inputs to the DNN are the observation variables of an environment at a given state. The DNN outputs the appropriate action for the agent in order to maximize the overall cumulative reward. We only use the weights and bias parameters of the neural network in our GA. No back probagation step is needed to train the model. The parameters are updated by the GA.

C. Environment

The environment used for all the experiments conducted is Open AI Gym [19]. It is a very comprehensive toolkit for RL research. It includes many diverse set of benchmark problems and tasks built using a standard interface. Specifically three environments from Gym were used in the following experiments namely Cart Pole, Mountain Car and Pendulum.

V. Experiment

All the experiments were performed on a laptop using Jupyter Notebook through WSL Linux subsystem. GPU computing was not used, still the generations only took several minutes to compute on the CPU. The dependencies for the project include the following libraries: numpy, pytorch and gym. All of these can be installed using pip or anaconda. Details on each individual environment used are provided below.

A. Cart Pole

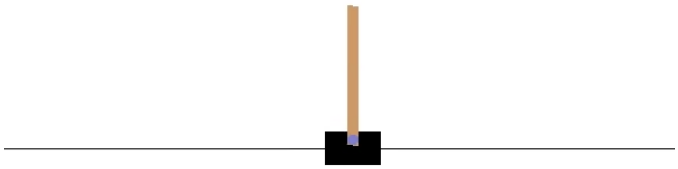

Fig. 4. Cart Pole v0

a) The Problem: A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is

more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

b) The Model: The neural network used is a single hidden layer, fully connected neural network. The input layer has 4 parameters and the output layer has 2 parameters; these correspond to the environment observation and actions respectively. The hidden layer used consists of 128 neurons. The total parameters for the network are $\theta = 898$, which is the genotype of the GA.

TABLE I
CART POLE: OBSERVATION (INPUTS OF NEURAL NETWORK)

Num	*Observation*	*Min*	*Max*
0	Cart Position	-2.4	2.4
1	Cart Velocity	$-\infty$	∞
2	Pole Angle	$\sim -41.8°$	$\sim 41.8°$
3	Pole Velocity At Tip	$-\infty$	∞

TABLE II
CART POLE: ACTIONS (OUTPUT OF NEURAL NETWORK)

Num	*Action*
0	Push cart to the left
1	Push cart to the right

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 128]             640
              ReLU-2                [-1, 128]               0
            Linear-3                  [-1, 2]             258
           Softmax-4                  [-1, 2]               0
================================================================
Total params: 898
Trainable params: 898
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.01
----------------------------------------------------------------
```

Fig. 5. Cart Pole Neural Network Model

B. Mountain Car

a) The Problem: Get an under powered car to the top of a hill (top = 0.5 position). There are 2 observations of position and velocity available and there are 3 possible actions no push, push left and push right. -1 for each time step, until the goal position of 0.5 is reached. There is no penalty for climbing the left hill, which upon reached acts as a wall. The environment starts from a random position -0.6 to -0.4 with no velocity.

b) The Model: The neural network used is a single hidden layer, fully connected neural network. The input layer has 2 parameters and the output layer has 3 parameters; these correspond to the environment observation and actions respectively. The hidden layer used consists of 256 neurons. The total parameters for the network are $\theta = 1539$, which is the genotype of the GA.

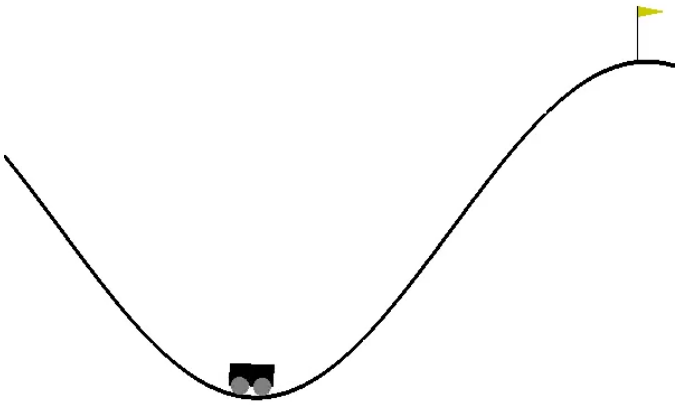

Fig. 6. Mountain Car v0

TABLE III
MOUNTAIN CAR: OBSERVATION (INPUTS OF NEURAL NETWORK)

Num	*Observation*	*Min*	*Max*
0	Position	-1.2	0.6
1	Velocity	-0.07	0.07

C. Pendulum

a) The Problem: Try to keep a friction-less pendulum standing up. The precise equation for reward: $-(theta^2 + 0.1 * theta_d t^2 + 0.001 * action^2)$. Theta is normalized between -pi and pi. Therefore, the lowest cost is $-(pi^2 + 0.1 * 8^2 + 0.001 * 2^2) = -16.2736044$, and the highest cost is 0. In essence, the goal is to remain at zero angle (vertical), with the least rotational velocity, and the least effort.

b) The Model: The neural network used is a single hidden layer, fully connected neural network. The input layer has 3 parameters and the output layer has 1 parameter; these correspond to the environment observation and actions respectively. The hidden layer used consists of 256 neurons. The total parameters for the network are $\theta = 1281$, which is the genotype of the GA. The output generates a regression of values from -2.0 to 2.0

TABLE IV
MOUNTAIN CAR: ACTIONS (OUTPUT OF NEURAL NETWORK)

Num	*Action*
0	Push left
1	no Push
2	Push right

TABLE V
PENDULUM: OBSERVATION (INPUTS OF NEURAL NETWORK)

Num	*Observation*	*Min*	*Max*
0	cos(theta)	-1.0	1.0
1	sin(theta)	-1.0	1.0
2	theta dot	-8.0	8.0

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 256]             768
              ReLU-2                [-1, 256]               0
            Linear-3                  [-1, 3]             771
           Softmax-4                  [-1, 3]               0
================================================================
Total params: 1,539
Trainable params: 1,539
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.01
Estimated Total Size (MB): 0.01
----------------------------------------------------------------
```

Fig. 7. Mountain Car Neural Network Model

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 256]           1,024
              ReLU-2                [-1, 256]               0
            Linear-3                  [-1, 1]             257
         Threshold-4                  [-1, 1]               0
================================================================
Total params: 1,281
Trainable params: 1,281
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.01
----------------------------------------------------------------
```

Fig. 9. Pendulum Neural Network Model

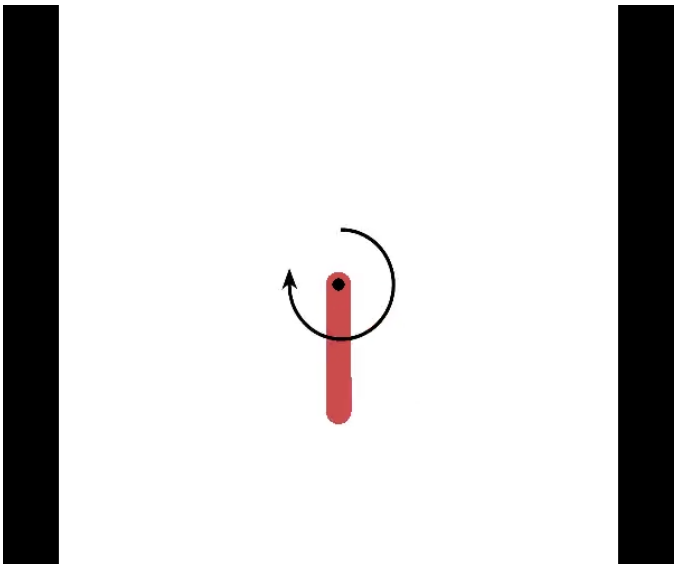

Fig. 8. Pendulum v0

VI. Results

All experiments are performed with maximum generations $G = 10$. At the end the candidate with the highest fitness value is used to generate the test render of the experiments. The final renders of all three experiments are attached with this report.

A. Cart Pole

Cart Pole experiment can run for infinite time. The experiment works very well with the simple GA-based algorithm which usually produces good candidates within the first 3 to 5 generations. The trained network can run for long periods without failing. It achieves perfect result. The model was trained on an episode length, $E = 5000$ to show the effectiveness of the trained model. This result runs the Cart Pole for about 2 minutes without falling. The GA manages to find multiple solutions to solve this problem.

TABLE VI
Pendulum: Actions (output of Neural Network)

Num	*Action*	*Min*	*Max*
0	Joint effort	−2.0	2.0

B. Mountain Car

This was a difficult problem for the type of GA-based approach implemented. Even though the algorithm managed to find a solution which can reach the goal in under 200 episodes, this problem is not well suited for GA. The environment keeps decreasing the score for each frame the goal is not achieved and the score is locked when the target is achieved. This kind of reward function in which there is no direct maximization or minimization step at each frame is difficult to converge. Yet the exploratory nature of the mutation function manages to find a solution which can solve the given problem.

C. Pendulum

This problem also showed good results with the GA. Since this problem can run infinitely, we can train the network to try and minimize the loss to the reward function by keeping the pendulum in the upright position for as long as possible. The goal is to apply the least amount of action to keep the pendulum upright and GA shows good results with this problem.

VII. Conclusion

In conclusion, I implemented a very simple Genetic Algorithm on a Deep Reinforcement Learning environment to produce quality results. 3 environments were used for experimentation and the GA performed very well with two out of the three problems. For the third one, the GA was still able to solve it but more work can be done to improve this basic GA to avoid getting stuck in such local optima states where the reward function is a bit deceptive. The exploration component of the mutation function still managed to find a solution to the problem but more work needs to be done to balance the use of exploration and exploitation when searching for solutions.

VIII. Future Work

The GA implemented is very modular and can be applied to similar problems to gain more insights into the effect of different types of environments and reward functions. The code can be easily augmented to operate with CNNs which opens up avenues for further experimentation with more complex problems and much lager state spaces based on pixel values. The GA is a very simple baseline version. Experimentation can be done to optimize this GA further to produce better results. We can replace the GA with more complex algorithms such as DE or CMA-ES and perform studies to gauge their effectiveness in this domain of Deep RL.

Acknowledgment

Thanks to Professor Shahryar Rahnamayan for his guidance and knowledge.

References

[1] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *CoRR*, vol. abs/1811.12560, 2018. [Online]. Available: http://arxiv.org/abs/1811.12560

[2] I. J. Goodfellow, "NIPS 2016 tutorial: Generative adversarial networks," *CoRR*, vol. abs/1701.00160, 2017. [Online]. Available: http://arxiv.org/abs/1701.00160

[3] J. Schmidhuber, "Deep learning in neural networks: An overview," *CoRR*, vol. abs/1404.7828, 2014. [Online]. Available: http://arxiv.org/abs/1404.7828

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[5] R. Munos and A. Moore, "Variable resolution discretization in optimal control," *Machine Learning*, vol. 49, no. 2, pp. 291–323, Nov 2002. [Online]. Available: https://doi.org/10.1023/A:1017992615625

[6] M. Bellemare, J. Veness, and M. Bowling, "Bayesian learning of recursively factored environments," in *International Conference on Machine Learning*, 2013, pp. 1211–1219.

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

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016. [Online]. Available: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

[9] OpenAI, C. Berner, G. Brockman, and B. Chan, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint*, 2019.

[10] K. Arulkumaran, A. Cully, and J. Togelius, "Alphastar: An evolutionary computation perspective," *CoRR*, vol. abs/1902.01724, 2019. [Online]. Available: http://arxiv.org/abs/1902.01724

[11] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[12] S. Sivanandam and S. Deepa, "Genetic algorithms," in *Introduction to genetic algorithms*. Springer, 2008, pp. 15–37.

[13] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016.

[14] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[16] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[17] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.

[18] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv preprint arXiv:1712.06567*, 2017.

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540