Genetic algorithm for robot planning in the BipedalWalker-v2 environment

Leandro Couto Medeiros - Universidade Federal de Viçosa

12/05/2019

1 Introduction

1.1 Genetic Algorithm

Genetic algorithms (GA) are a group of computational models used in operational research and computer science that mimics evolution to solve search and optimization problems. As the theory of evolution dictates, heritage characteristics and patterns of biological beings are passed through successive generations by reproduction. Inherited from genes, these characteristics reflect the exposure of a population to an environment. In order to better fit this enviroment, natural selection is the key mechanism of evolution that is responsible for these heritable traits.

Most techniques used in GA are similar to genetic variations observed in evolution. The most common ones are selection, crossover and mutation.

Given an initial population (usually randomized), the **selection** process is responsible for choosing which chromosomes from the population are most suited for breeding in the crossover phase. Because GAs are heavily customizable (that is, it can be modeled in many different ways), how one can model the selection is generally depended on the problem. One strategy that is commonly used is to apply a probability of being chosen to each of the cromosomes. If a chromosome is more fit to the enviroment, then it will receive a higher probability. On the other hand, a chromosome that is not suited is doomed to be extinct by natural selection and therefore will receive a lower probability.

The **crossover** stage is in charge of the reproduction between two selected chromosomes. Common techniques of crossover are swapping a certain quantity of genes between the chromosomes. How this swap is done is again highly relied on the problem. Domain-specific knowledge can also be used in this stage, because a smart choice of genes being chosen can generate fruitful individuals for the next generation.

Lastly, the **mutation** phase is very important because it is used to maintain genetic diversity in the population. Genetic diversity is crucial because it prepares the population for environmental changes. This way, if some drastic changes occurs in the environment, the population may have a higher chance of surviving due to these mutated genes. In GAs, mutations are commonly done by changing some genes inside the chromosome. The choice of mutating every chromosome or not is also tied to the problem.

1.2 OpenAI Gym

OpenAI Gym (BROCKMAN et al., 2016) is a toolkit developed by OpenAI that allows users to implement artificial intelligence algorithms in simple environments. It has an open-source interface meaning that students and researchers need only to implement the learning algorithm and not the environment in itself. It also has support for numerical computation libraries such as TensorFlow.

2 Methodology

In this report, it was used the *BipedalWalker-v2* [1] environment. It is given as input to the genetic algorithm a vector of four real values between -1 and 1 that represents an action the robot takes in the environment. Even though it is known what these four values mean (they represent the $\frac{Torque}{Velocity}$ of the hips and knees of the robot), this work does not take

[1]http://gym.openai.com/envs/BipedalWalker-v2/

into consideration domain-specific knowledge and therefore, the GA implemented learns a sequence of actions by just experimenting these four values in the environment and then applying the genetic algorithm phases given the reward received after executing an action.

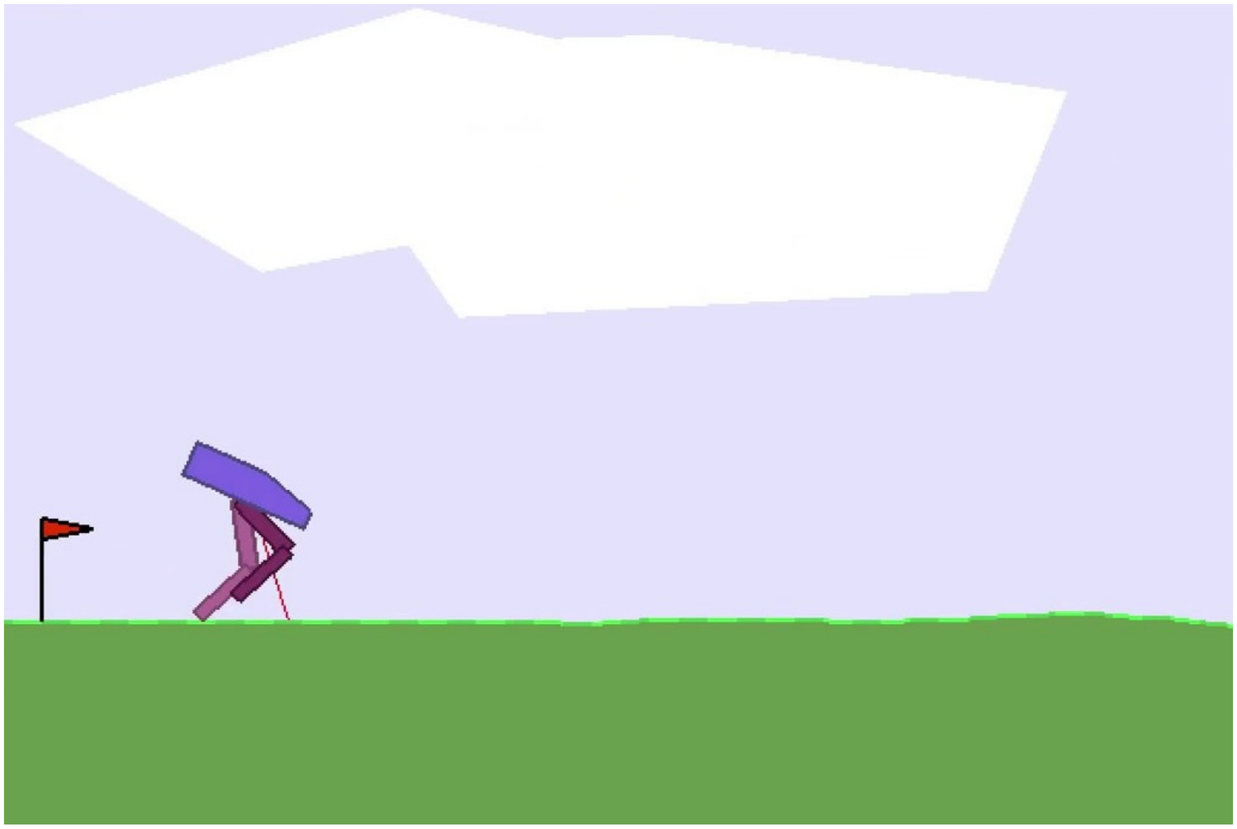

Figure 1: BipedalWalker-v2 environment from OpenAI Gym

For this problem, a chromosome is represented as a list of actions. It was modeled this way because since the objective is to make the robot walk, it is more natural to represent a walk as a sequence of actions as opposed to a single one. The length of this list was defined empirically as 40. As expected, the population is a list of chromosomes. In the implementation, the population has 100 chromosomes.

Initially, all actions from the chromosomes are randomized. Then, for each generation (specified in this project as 100) each chromosome goes in the **playout** stage: the chromosome applies their list of actions in order for $\frac{500}{40}$ times and sums all the rewards received by each action applied. **This sum represents the chromosome's fitness value**.

After the playout, it is calculated the probabilities of each chromosome to be selected. The way this is implemented is as it follows: find the minimum fitness value of all chromosomes, then sum the modulus of this value to all fitness values and then sum one. This is made in order to maintain all fitness values not negative, facilitating the probability calculation. It is necessary to sum one at the end to avoid any new fitness value to not be 0 (and therefore, receiving a 0% chance of being selected). After this step, it is easy to get a probability for each chromosome given the new fitness value: keep track of past fitness values and divide by the new sum of the fitness values. For example, for a fitness value list $[4, 3, 2, 1, 0, -1]$, the new list will be $[6, 5, 4, 3, 2, 1]$. Then, the new distribution probability list will be $[\frac{6}{21}, \frac{11}{21}, \frac{15}{21}, \frac{18}{21}, \frac{20}{21}, \frac{21}{21}] =$ [0.28, 0.52, 0.71, 0.85, 0.95, 1.00]. This way, it is generated a random real number between 0 and 1. If it is 0.15, then the first chromosome will be selected. If 0.72 is generated, then the third chromosome will be selected.

After this step, it is calculated the **elitism** of this population. An elite of a population is simply the best chromosomes in terms of their fitness value. In the algorithm, it is used a 50% of the population for the elite. This means that the best 50% of the population are guaranteed to proceed to the next generation.

Given the elite, the algorithm enters the **crossover** stage. It was implemented two different crossovers that from now on will be referred as **crossover 1** and **crossover 2**. Crossover 1 will sample two chromosomes (i.e., the selection phase) from the whole population whether crossover 2 will sample two chromosomes only between the elite. Note that in crossover 2 the algorithm is effectively discarding the worst half of the population while in crossover 1, the worst half will still be used for sampling. For both of them, the actual crossover procedure is very simple: if a chromosome A is having a crossover with another chromosome B, A will keep the first 75% of its actions list and the remaining 25% will be taken from the first 25% of the actions list of B. Then the same procedure will be done with B doing a crossover with A. This way we guarantee that the population will keep the same size between generations.

After the crossover, the population goes through the **mutation** stage. The mutation used in this implementation is a simple one: The algorithm iterates through all chromosomes, there is a chance of 50% of a chromosome to be mutated. If it is, then 1 random action of the chromosome will be chosen to be reset to a completely random one. As explained before, mutations are important because they help prevent the algorithm from falling to a local maxima.

3 Results and Conclusion

Below it is shown the performance of the developed genetic algorithm for the *BipedalWalker-v2* environment. As previously stated, the fitness function used is the sum of all the rewards received by all the actions present in a chromosome. For our particular experiment, it is the sum of 40 actions applied $\frac{500}{40}$ times. The x-axis represents the number of generations, with a value of 100 in this implementation. The graph in the left represents our algorithm using **crossover 1** whereas the graph in the right was tested using the **crossover 2** implementation.

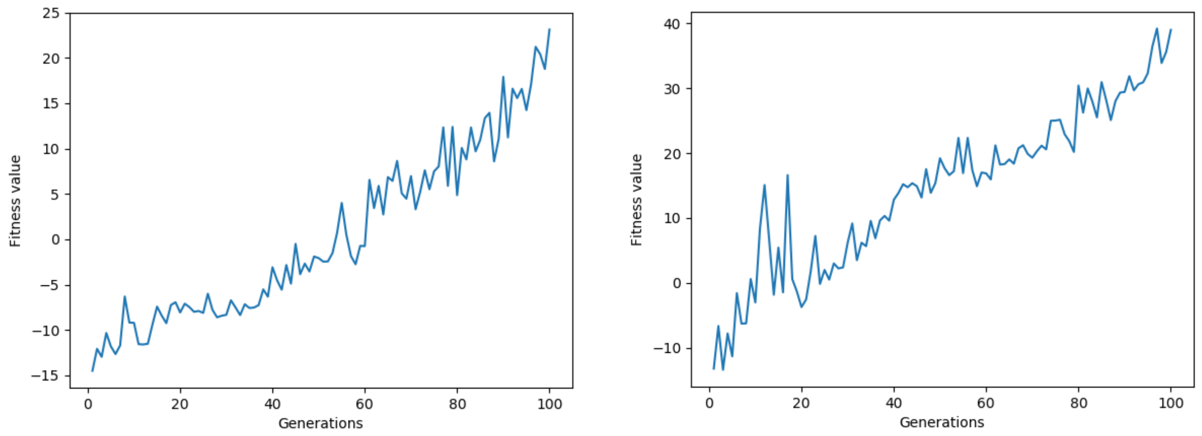

Figure 2: Learning curve for the genetic algorithm using crossover 1 and 2, respectively

As it can be noted, the best chromosome from the crossover 1 experiment which selects chromosomes from all the population reaches a fitness value of 23.12 after 100 generations. On the other side, crossover 2 reaches 38.97 of fitness value also in 100 generations.

For this particular environment, it can be inferred that a more greedy approach (crossover 2 implementation selects chromosomes only from the elite) works better. A possible explanation for this is that because the environment and the action modelling are so simple (only four real values), a good (high fitness value) chromosome bred with another good chromosome is likely to birth an also fit offspring. Note that the environment rarely changes in this framework (only slightly variances on the ground structure), so biasing the population of the next generation based only on the elite will likely generate a steady curving learning experience for the robot. It is conjectured that crossover 1 model would also reach higher values of the fitness function. However, since it applies a probability for all of the population, several bad chromosomes would still be chosen for many generations to come.

This project was undertaken to design a genetic algorithm for the *BipedalWalker-v2* environment and to evaluate its learning curve of the best chromosome over the generations. Noticeably from Figure 2, for both models the genetic algorithm developed fulfilled the purpose. This report and the code for this experiment can be found here: ⟨https://github.com/leandrocouto/genetic-bipedal-walker-v2⟩.

For future work, it would be interesting to investigate more the hyperparameters of the GA developed and to test different ways of doing the crossover and mutation phase. A future challenge for a genetic algorithm would be applied to another environment called *BipedalWalkerHardcore-v2* [2], where the environment is extremely inconsistent. A greedy approach for the crossover stage is likely to fail and therefore, a smart way of assessing this environment is expected.

[2]https://gym.openai.com/envs/BipedalWalkerHardcore-v2/

References

BROCKMAN, G. et al. **OpenAI Gym**. 2016.