Genetic Algorithms and Neural Agents

Angel Macwan
MSc Data Science
Amity University Gurgaon, Haryana

Abstract

Inspired by Charles Darwin's Theory of Natural Evolution, Genetic Algorithms uses a heuristic approach that reflects the process of natural selection. Each member of a generation will have their individual DNA object which in this case will be a neural network. Weights, biases, and outputs of these objects will act as genetic material which will be used to form a new population. Another method of training an agent is by using Imitation Learning (also called Behavior Cloning). Humans often learn how to perform tasks via imitation: they observe others perform a task, and then very quickly infer the appropriate actions to take based on their observations. Extending this paradigm to autonomous agents allows us to train these autonomous agents to imitate human behavior by training them on human-generated data.

Contents

List of Figures

1 Proposed Work

Two methods of training an agent have been implemented here: Genetic Algorithm and Imitation Learning. The goal of these algorithms is to test and generate a training system that can be used to train hardware instances of robots by various methods.

2 Genetic Algorithms

Genetic Algorithms are heuristic algorithms that are used to evolve smart agents from a seemingly random population. Darwinian Natural Selection proposes three core principles: Heredity, Variation, and Selection.

Heredity: If creatures live long enough to reproduce, then their traits are passed down to their children in the next generation.

Variation: There must be a variety of traits present in the population or a means with which to introduce variation. Without any variety in the population, the children will always be identical to the parents and to each other.

Selection: This is typically referred to as "survival of the fittest." A DNA or agent with a higher fitness will have a higher chance to pass down its gene to the next generation.

The process of natural selection starts with the selection of the fittest individuals from a population. They produce offspring which inherit the characteristics of the parents and will be added to the next generation. If parents have better fitness, their offspring will be better than their parents and have a better chance of surviving. This process keeps on iterating and in the end, a generation with the fittest individuals will be found.

LISTING 1. Pseudo code

```
START
Generate initial population
Compute fitness
REPEAT
    Selection
    Crossover
    Mutation
    Compute fitness
UNTIL population has converged
STOP
```

2.1 Architecture

The implementation of a genetic algorithm uses a Neural Network as a brain/DNA object. There are a total of 16 agents in each generation. A new generation is created when all the existing members of a population die. A higher population density per generation will result in more variety leading to faster learning but it might also slow down the simulation as drawing each agent and executing their neural network will be a CPU expensive task.

Agents in a population are represented by a blue circle (representing car agent). There are obstacles on the road represented by red circles. This obstacles are spawned every 10 frames and are removed from the list of active obstacles as soon as it hits the end boundary (0 on X axis). The goal of the agents is to dodge all obstacles and survive the longest. The fitness function is simply defined as the number of frames an agent survives without hitting an obstacle.

This Neural Network used as DNA/Brain is a shallow network with 5 inputs and 2 outputs. The 5 inputs are the X and Y locations of the agent itself, the X and Y location of the nearest obstacle, and the distance to the nearest obstacle. The outputs are the probability of going left or right in order to dodge the obstacle. Crashing into an obstacle will eliminate the agent.

Once all agents of a population are eliminated a gene pool is created with all agent objects in it. The chance of a DNA from an agent being picked for the next generation depends on how well it performed on the current run. A static mutation rate of 0.1 or 10% is used, i.e there is 10% chance that a selected weight or input will be altered.

The figure below shows the neural architecture and visual representation of the simulation.

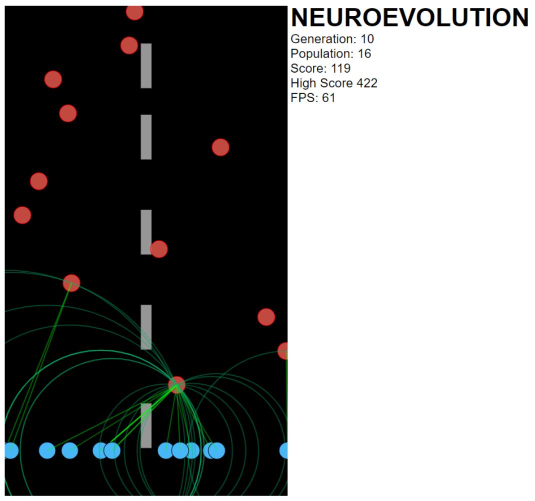

FIGURE 1. Visual representation of the simulation

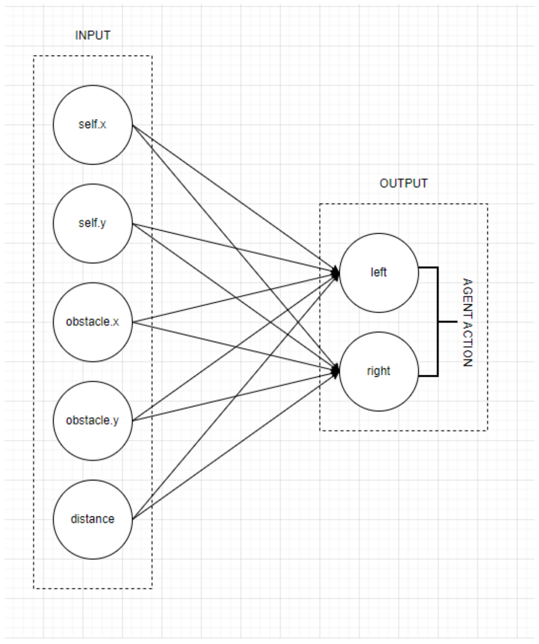

FIGURE 2. Neural Architecture

In about 10 generations the agents were able to capture the overall gist of the simulation and started avoiding obstacles. The score kept rising with each passing generation as agents keep learning from past experiences that were passed down from their ancestors.

This is a very effective method of training an agent to perform complex actions when training data is not available. Often times these algorithms will produce behaviors that were previously unknown were possible.

One of the examples would be from a paper by OpenAI called ‘Emergent Tool Use from Multi-Agent Interaction’ in which agents were able to exploit a bug in the physics engine to come up with strategies that were previously unknown. However, these algorithms are notorious for their long training time, especially in complex environments including a variety of inputs and a large action space.

3 Imitation Learning (Behavior Cloning)

Imitation Learning remains one of the simplest machine learning methods to acquire robotic skills in the real world. It is a method by which human sub-cognitive skills can be captured and reproduced in a computer program. As the human subject performs the skill, his or her actions are recorded along with the situation that gave rise to the action.

A log of these records is used as input to a learning program. The learning program outputs a set of rules that reproduce the skilled behavior. This method can be used to construct automatic control systems for complex tasks for which classical control theory is inadequate.

3.1 Architecture

An autonomous car is used as the agent for this example. The simulator used is the Udemy Car Simulator which provides a socket API to communicate (capture data and send inputs) with the simulator. The Udemy Car Simulator is a simple environment with a car agent on a track. The inputs to the agent are throttle and steering angle. The throttle controls the acceleration of the car and the steering angle steers the car with a steering limit of 30 degrees in both directions.

The simulator has 3 camera sensors that output 1 image each. These images are augmented to generate data for the left, center, and right steer. This output data is collected as JPEG and a CVS is generated that contains metadata such as steering angle and throttle. A total of 14,484 data records were generated by manually operating the agent. Generated data is then balanced, augmented, and used to train a machine learning model.

The controller script also hosts API endpoints to transmit information about the model. This includes outputs that the model generates after inference and current status of the simulator (online or offline).A separate p5js sketch connects to this endpoints and logs all required information visually. This helps with debugging and sending commands to the simulator directly from the front-end application without disrupting the controller script and breaking the connection.

The simulator outputs a single image when in autonomous mode. This image is fetched by the controller script and is used as inference for the model. The controller model (ML Model) comprises two convolution layers and three dense layers. The Machine Learning Model is trained on 10 Epochs. Amount of data required to train this model is relatively large (about 239 MB), this data has to be loaded and augmented in the training script. Loading all data at once leads to extremely high memory usage as there will be multiple augmented copies of data in memory at the same time. In order to overcome memory bottleneck, data is loaded in batches using a generator function with a batch size of 256.

FIGURE 3. Simulator Output

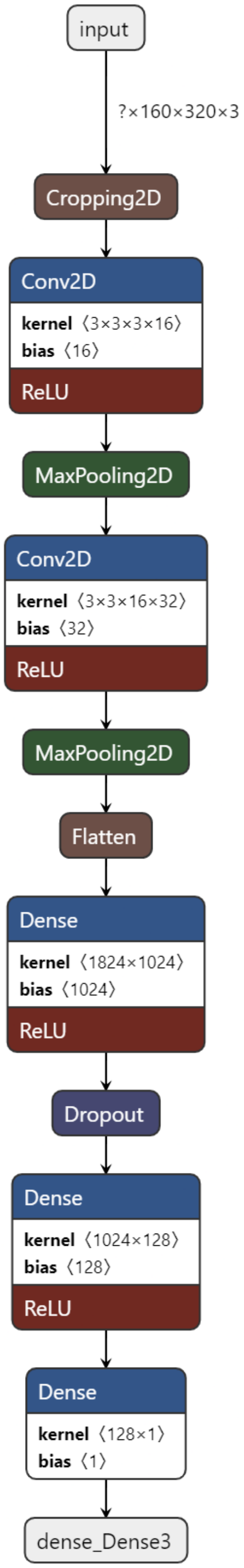

FIGURE 4. NN Architecture for Autonomous Agent

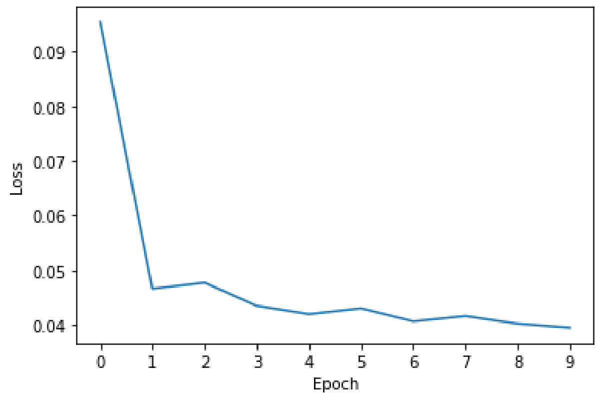

FIGURE 5. Imitation Learning Training Graph

The output is a single float value that represents the steering angle. This steering angle is transmitted over sockets to the simulator via the socket.io module in node JS. The simulator also expects a throttle value which can be calculated by getting the current velocity (which can be obtained from simulator API) and maximum velocity that has been hard-coded in the controller script.

For this demonstration, the value of maximum velocity is set to 30 (which is the maximum speed at which the simulated car can travel).

$$throttle = 1 - currentSpeed/maxSpeed$$

By using the mentioned architecture the agent was easily able to learn the landscape and rules to navigate it. However, a drawback of this approach is a generalized model is really hard to produce. The model trained in a particular environment will only operate in that given ecosystem. It will have a hard time adapting at all to other environments.

4 Conclusion

The goal of this study was to go over two most widely used algorithms for training autonomous agents. Genetic Algorithms are very useful when training data is not available or the final outcome of a system is undetermined. Genetic algorithms can often produce surprising behaviors that leads to the solution in a manner that was previously unknown. For example bugs in a physics engine was used as exploit in a study conducted by OpenAI titled 'Emergent Tool Use from Multi-Agent Interaction'

Methods such as Imitation Learning and Behavior Cloning are much more effective if enough data is available and the desired outcome is predetermined. Behaviour and movement of a human can be recorded and simply transferred over to a simulation or a robot if the environment is static. In case of a dynamic and rapidly changing environment more data is required covering almost all conditions.

5 References

- towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3
- The Nature of Code by Daniel Shiffman
- openai.com/blog/emergent-tool-use/
- carla.org/
- proceedings.mlr.press/v78/dosovitskiy17a/dosovitskiy17a.pdf
- proceedings.mlr.press/v164/florence22a/florence22a.pdf
- p5js.org (p5.js is a JavaScript library for creative coding, with a focus on making coding accessible and inclusive for artists, designers, educators, beginners, and anyone else)