CS275 Project Report: Survival Strategies of Bacteria with Evolving Algorithms

Yuxi Zhang,* Kainan Wang,* and Tao Yuan*

University of California, Los Angeles

E-mail: yuxi.zhang@cs.ucla.edu; knwang@cs.ucla.edu; taoyuan@ucla.edu

Abstract

The goal of this project is to train an AI that can learn survival strategies from the environment. We carefully design an environment for bacteria that does not have trivial strategy to survive. Our training process starts from simple strategies and we apply the trained model as input for the next round to keep it evolving. We adjust the environment with different parameters and observe how it changes the behaviours of bacteria.

Introduction

In the past, people tried hard to design strategies for artificial life. However, most environment are too complex to fit in single strategy and when we apply multiple strategies, it very hard to find the trade-off. Anther strategy is searching, whose time complexity grows too fast for the environment size. Nowadays, we only need to set up the for the environment and define a function as a goal, we can train a model to make those decisions. As a course project, we want to know more about how it works. In the first part of this report, we discuss the environment setting, a square world of bacteria. We assume that all bacteria are

in shape of circle and bigger one win when two touch with each other. Moreover, we add one more attribute to bacteria as energy. The energy attribute helps to discourage bacteria from being too large. In the simulation and training part, we apply different strategies to different bacteria to keep diversity of the environment. For the smart strategy modeled by neural network, we utilize decisions made by bacteria with longest life as input for training. And after one round of training, we can use the trained model for the next round. We also design a web interface to display the simulation result. Before testing, we made some possible strategies to see if neural network favors one of them. During the testing phase, We adjust different parameters of the environment and train the model with different settings of strategy distribution. We apply HTML5 SVG technique for visualization and create a demo to show how the training and testing works.

Environment Settings

Motivated by agar.io,[1] we create a world of bacteria, which be considered as a unit square. Bacteria are circles born at random places in the world. The radius of bacteria follows geometric distribution. When two bacteria meet with each other, the big one devours the small one and becomes bigger. The size of merged bacteria is the sum of its components. When we discuss the relation between two bacteria, the bigger one is called predator and the smaller one prey.

Also, each bacterium has a maximum speed. Smaller bacteria move faster than bigger ones so that they have chances to avoid being chased. When a bacterium hit the border of the square, it will be reflected like a light ray. As a result, the bigger bacteria may catch its prey by chasing it to the corner of the square.

However, there is a naive and strong strategy for the environment above. In most cases, the optimal strategy for a bacterium is to chase its closest prey and avoid being too close to its predator. Once a bacterium becomes large enough, no one else can threaten its life. To

resolve this problem and make the world more practical, we introduce the notion of energy. Bacteria need positive energy to move and eat. When a predator catch a prey, the predator immediately get energy proportional to the prey's size. For each tick of simulation, every bacterium lost certain amount of energy proportional its size. After a bacterium has zero or negative energy, it dies and get removed from the square. Once a bacterium becomes too large, it can not find enough food to catch up with its energy lost.

So, the state of a bacterium B_i at each time frame τ can be represented by:

$$\Phi_i^\tau = \{x_i^\tau, y_i^\tau, r_i^\tau, \overrightarrow{v_i^\tau}, E_i^\tau\}$$

x_i^τ, y_i^τ is the position of b_i, r_i^τ is the radius of b_i. v_i^τ is the velocity and we have:

$$\overrightarrow{v_i^\tau} = |v_i^\tau| * \overrightarrow{d_i^\tau}$$
$$|v_i^\tau| \propto \frac{1}{r_i^{\tau 2}}$$

E_i^τ is the energy, and we have:

$$E_i^{\tau_0} \propto r_i^{\tau_0 2}$$
$$\Delta E_i^\tau = \alpha E_i^{\tau-1}, \alpha = 0.05$$

if two bactera B_i, B_j collapse at time τ, and $r_j^\tau < r_i^\tau$, the bigger bacterium will gain the size of the smaller one. B_j will disappear and B_i will be updated:

$$r_i^{\tau+1} = \sqrt{r_i^{\tau 2} + r_j^{\tau 2}}$$
$$E_i^{\tau+1} = E_i^{\tau} + r_j^{\tau 2}$$

Simulation

The simulation system includes two parts, one for bacterium and the other for environment. The bacterium maintain the status of each bacteria. It can also choose where to go given condition of its nearest neighbours. The environment module simulate how bacteria interact with each other. It simulate actions of bacteria tick by tick. When some event happens (die, eat), it update the status of correspondent bacteria. Moreover, it needs to record the history of simulation for training and visualization

Learning

In order to make the bacteria move more intelligently, we apply neural network algorithm to train the bacteria. FANN[2] is chosen as our learning library for training and testing.

In the first step, all the bacteria are very simple and naive. They either move randomly or just flee away from their nearest enemy which is larger than them. Based on this strategy, we simulate this environment for thousands of ticks and pick top 10% bacteria with longest lives. We think those top 10% bacteria behave smartly, and choose their movements as training example.

In the next steps, since we already have a trained neural network, we design a iterative training. In other words, we utilize the current trained neural network to generate new simulation data. In new simulations, one third of the bacteria move randomly, one third

of the bacteria take naive strategy, and one third of the bacteria adopt neural network strategy. Note that in order to avoid over fitting problem, we only choose part of bacteria to "act smartly" and other bacteria still follow random or naive strategy.
For each movement, we choose the following data as input.

- radius of the bacterium
- energy of the bacterium
- radius of neighbours
- position of neighbours

We have some intuitions about those attributes. First, each bacterium cares more about its neighbours. In most cases, they tend to chase prey and avoid predators. However, since bigger bacterium tends to die quickly, they are not willing to eat until they are hungry.
The output should simple contain the direction information of the bacterium's moving, since we set the moving speed is a function of its radius. We initially take the degree value $\theta \in [-\pi, \pi]$, however, we later found that there is a problem in this design. For example, the degree close to π and the degree close to $-\pi$ should indicate similar direction, but they have large training errors in the neural network, since their numerical values do differ a lot. The way to solve this problem is to take two values, $\cos\theta$ and $\sin\theta$ to describe the direction. Due to their periodic and continuous properties, any close degree should have similar cos and sin values, while the degree value itself sometimes may differ a lot. Once we have the cos and sin value, we can get the degree value by

$$\theta = \arctan\frac{\cos\theta}{\sin\theta} \tag{1}$$

which is a one-to-one function.

Visualization

The visualization is implemented with HTML5 SVG technique, and can be run directly in the browser. The data are first produced by our simulation engine, and then read by javascript to visulize the result. In this section, we will introduce the pipeline of the visualization and tools used in our project.

Tools

Jinja2[3] is a modern and designer-friendly templating language for Python. The bacteria states data produced by simulation engine are read by Python and translated into javascript variables using Jinja2.

Two.js[4] are used for visualizing the result. It is a two-dimensional drawing api geared towards modern web browsers. It is renderer agnostic enabling the same api to draw in multiple contexts: svg, canvas, and webgl.

Bacteria Visualization

Bacteria are represented by circles. Random colors are assigned to identify different bacteria. A red ring around a bacterium represents the energy ratio of that bacterium. Lighter ring color means lower energy ratio. Figure 1 shows two bacteria with different energy ratio.

Result

In the simulation, we generated 40 bacteria at the beginning, and then generate one bacteria every 10 ticks. We run a whole simulation for 1000 ticks.

The environment is a 60 by 60 square. The base size of a bacterium is 1, and the actual size follow the geometry distribution as stated before. The base speed is 1, and it's proportion to the inverse of its radius.

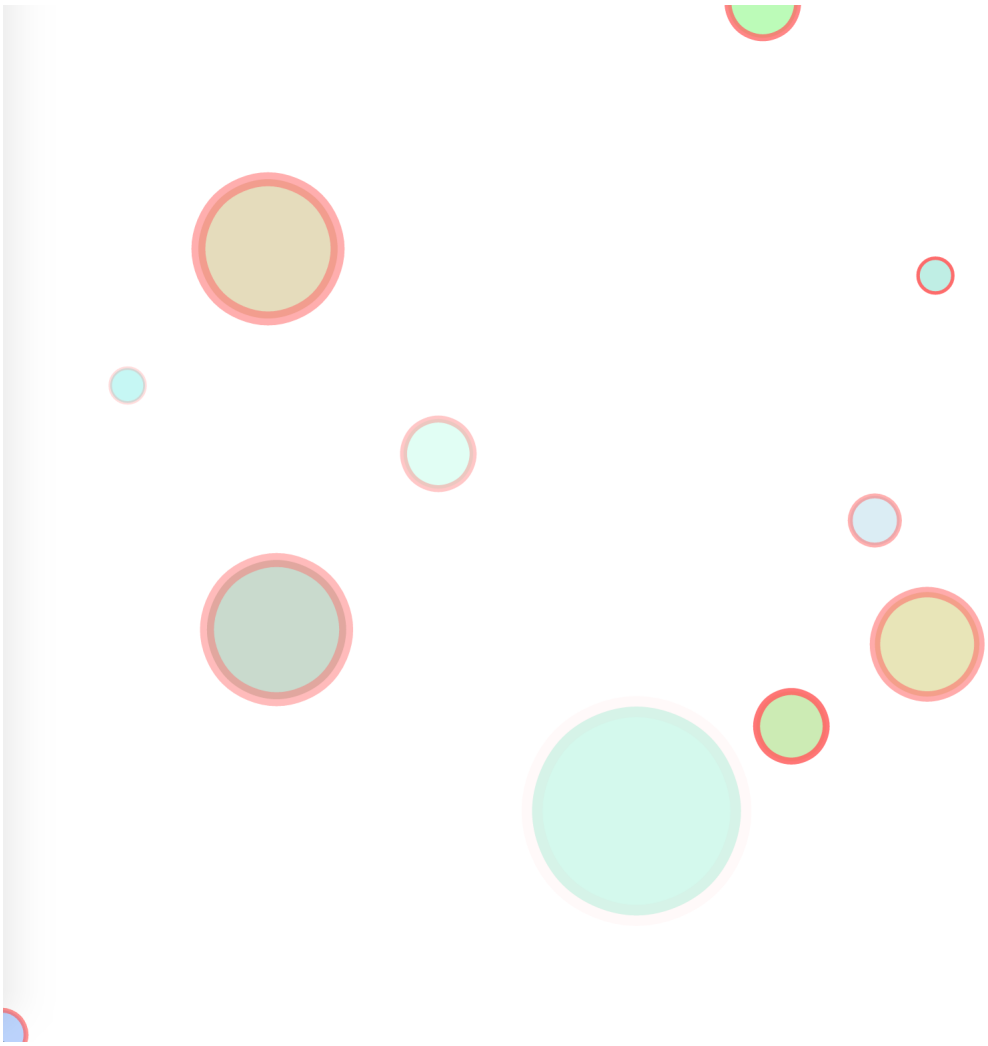

Figure 1: Bacteria example

Assume a bacterium's radius is r. The initial energy of a bacterium is proportion to r^2, while the energy consuming speed is proportion to r^3, which indicates that the larger bacterium consuming faster. Once a bacterium devouring another, it gets all the prey's initial energy. For the neural network, the input dimension is 42, including its own raidus and energy, as well as 10 nearest neightbour's information (position X, position Y, radius, energy). The output dimension is 2, just including $\cos\theta$ and $\sin\theta$ as previously stated. We config the neural network with 3 layers and 10 hidden neurons. The maximum iteration times is 1000. We iteratively train the neural network for 10 times, and record each bacterium's age when it's die, as well as its moving strategy. Then we calculate the means and the variances of each strategy in each iteration. Figure 2 shows the means in each iteration. From the figure, we can see that the neural-network-based moving strategy performs better than the other two, and has an increasing tendency. This makes sense because as training more iterations, neural network should perform better and better.

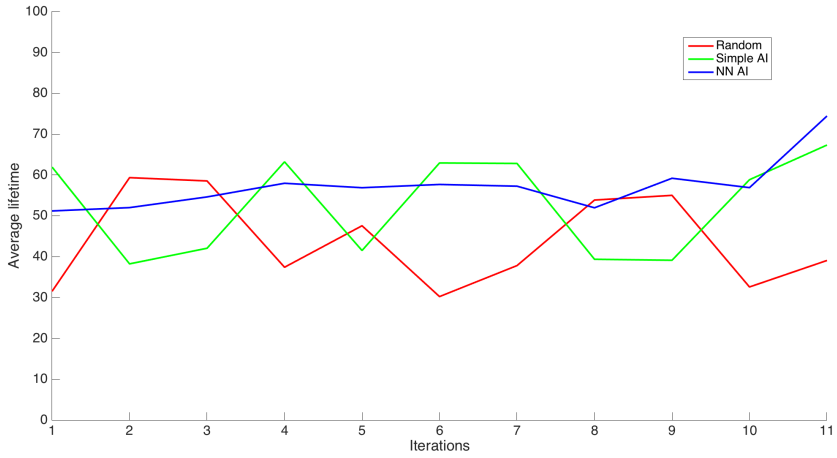

Figure 2: Average lifetime of different moving strategies. Red line represents random move, green line represent simple flee-away moving strategy, and blue line represents neural-network-based strategy.

Figure 3 shows the variances in each iteration of different strategy. From the figure we can see that although NN-based strategy performs better on average living time, its variance does not make much difference from the other two. Variances of all the three are fluctuating

around 1000. This means neural network strategy cannot lead to a more stable result.

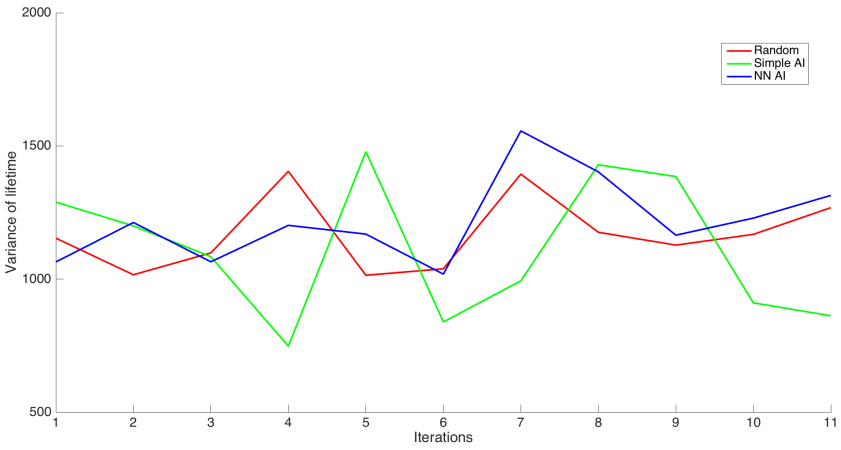

Figure 3: Variances of lifetime of different moving strategies. Red line represents random move, green line represent simple flee-away moving strategy, and blue line represents neural-network-based strategy.

References

(1) AGAR.IO. `http://agar.io/`.

(2) Fast Artificial Neural Network Library. `http://leenissen.dk/fann/wp/`.

(3) Jinja2. `http://jinja.pocoo.org/`.

(4) Two.js - jonobr1. `https://jonobr1.github.io/two.js/`.