Aviva Blonder

Evolving Continuous-Time Recurrent Neural Networks to Solve a Minimally Cognitive Task

Introduction

All organisms that exist and have ever existed arose through evolution. Their traits are the result of mutations that rose to prominence in the population through a combination of natural selection and genetic drift. Neurological structures and the resulting cognitive capacities are no exception. Cognition does not only emerge through learning in the lifetimes of individual organisms, but also evolves in populations over the course of generations. Artificial cognitive systems can evolve as well. Evolutionary algorithms can be used to find the optimal parameters to solve a particular task in a given environment, similarly to how natural selection favors organisms who are better suited to their environment.

Evolutionary algorithms have been used to solve many tasks of varying complexity. Beer (2003, 2015) has thoroughly analyzed how neural networks can be evolved to solve minimally cognitive tasks. He uses continuous-time recurrent neural networks (Beer 1995). They are recurrent in that activation does not just flow from the input layer to the output layer, there are also connections between neurons in the same layer, including connections from neurons to themselves. This enables the network's response to one stimulus to influence its response to the next over time. This is magnified by the continuous-time component. The activation of a given neuron is equal to the sum of its inputs, as in a conventional feed-forward neural network, plus whatever remains of its activation from the previous time step, as modified by a leak term. This persistence of information over time enables continuous-time recurrent neural networks to exhibit complex behavior despite their simple structure.

One of the problems Beer evolved continuous-time recurrent neural networks to solve is a simple categorical perception task. The neural networks controlled a single agent on a linear track. It was able to detect the approach of varying objects from "above" and had to catch some while avoiding others. I have implemented a simplified version of this task and used NumPy to create continuous-time recurrent neural networks that I have evolved to be capable of solving it. In my version of Beer's task, the agent is a horizontal bar composed of five "shadow sensors" that can detect whether or not something is directly above the agent. Agents are evolved to catch as much of falling objects as possible. The most successful agents' genomes are then recombined and mutated to make the next generation. I used an evolutionary algorithm to optimize the weight, bias, gain, and leakage - or time constant - parameters for each neuron. I ran several iterations to find the optimal parameters for the evolutionary algorithm, including mutation rate, crossover rate, and survival rate. I then ran the evolutionary algorithm repeatedly with the optimal parameters and monitored the results for various random seeds.

Methods

My implementation is composed of three classes, all written in the Python programming language.

The World class implements the minimally-cognitive task I have evolved the continuous-time recurrent neural networks to solve. It is composed of a 2-dimensional grid world with a default size of 15 rows and columns, though the user can change the height and width of the grid to suit their needs. Falling objects are simulated as ones on a grid of zeros. Each tick, all of the objects on the grid fall one row. New objects are created on the top row at an interval designated by the user. For all of my tests, I generated new objects every 5 ticks. I simplified the task by making all falling objects take up exactly four adjacent squares on the grid and generated all of them in the same location. The agent was rewarded for catching falling objects by being in the same column of the grid when the objects reached the last row and were removed. The agent gained one point for every square of an object that it successfully caught.

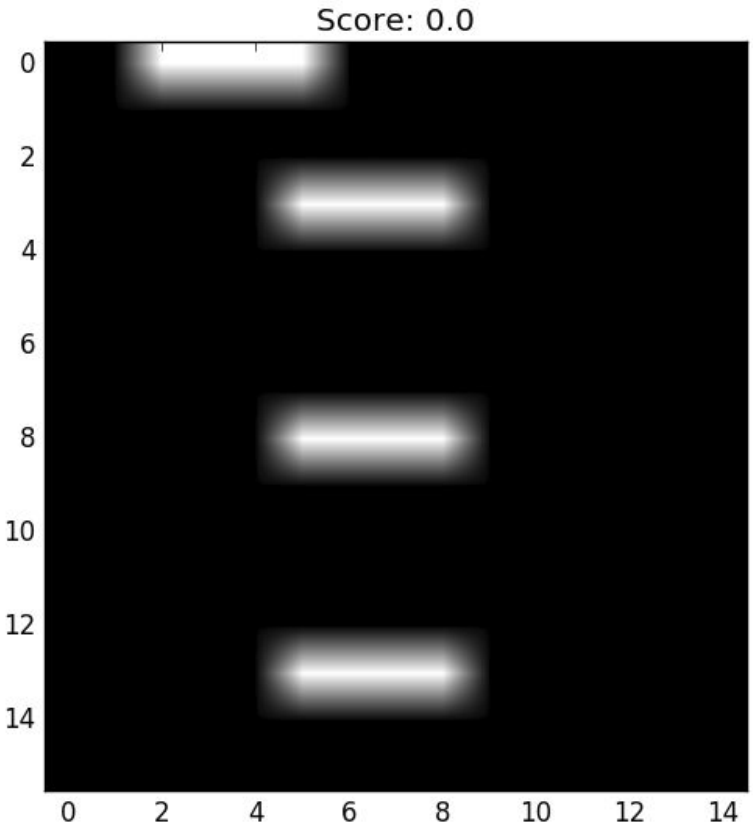

Figure 1. A visualization of the grid world generated in my implementation of Beer's minimally cognitive task. The black squares are empty. The five white squares in row 0 (at the top of the

image) represent the agent's location. Its shadow sensors are activated by the presence of objects in the same column. The three white bars, each 4 squares long, represent falling objects. In this visual representation, they are falling "up" towards the agent. The agent's current score - the number of object-squares it has successfully caught - is displayed above the grid.

I implemented the continuous-time recurrent neural network in the Agent class. The neural network is composed of several NumPy arrays that I combined using various matrix operations. The agents' genomes consist of two "chromosomes;" their weights and other parameters, each of which is a three-dimensional NumPy array. The weight-array alternates between matrices of weights from a layer to itself, and weights from one layer to the next. The other parameters are in another NumPy array, organized by layer, neuron, and parameter type. This ensures that a given neuron's parameters are unlikely to be split up in crossover, but can be indexed by type for use in calculations. All of the dimensions in a NumPy array must be consistent in length, but not all of the neural network's layers necessarily contained the same number of neurons, so I filled in unused parameters for nonexistent neurons with zeros. This had the added advantage of creating an analog to non-coding regions that are unaffected by natural selection. I enabled the user to designate the number of layers in the neural network and the number of neurons in each. I consistently used a neural network with 5 input neurons, 3 hidden neurons, and two output neurons. Each input neuron represents one shadow sensor that receives an external input of 1 if there is an object in the same column and an external input of 0 otherwise. The output neurons represented a decision to move to the left and a decision to move to the right respectively. The agent's actual action was determined by taking a softmax over the activations of the two output neurons. To calculate the activation of each neuron, I translated the activation function in Beer 1995 into a matrix equation (Equation 1).

$$\boldsymbol{x}_l = (-\boldsymbol{x}_l + \boldsymbol{w}_{l,l} \cdot \sigma(\boldsymbol{x}_l) + \boldsymbol{w}_{l\text{-}1,l} \cdot \sigma(\boldsymbol{x}_{l\text{-}1}) + \boldsymbol{b}_l + \boldsymbol{I}_l)/\tau_l$$

Equation 1. The activation function for layer l of a continuous-time recurrent neural network where $\boldsymbol{x}_l$ is the activation of layer l, $\boldsymbol{w}_{l,l'}$ is the weight matrix from layer l to layer l', $\boldsymbol{b}_l$ is the array of biases to layer l, $\boldsymbol{I}_l$ is the array of external inputs to layer l, τ_l is the array of time constant parameters, all greater than 0, for the neurons in layer l, and $\sigma(\boldsymbol{x})$ is the sigmoid function of each element of $\boldsymbol{x}_l$ (Equation 2).

$$1/(1+e^{-xg})$$

Equation 2. The sigmoid function for the activation of neuron x with gain parameter g, which is greater than 0.

I implemented an evolutionary algorithm in the Evol class. It generates a population of agents with the same neural network structure and a random genome of values between -1 and 1. The user designates the population size and neural network structure. The population can then be evolved for any number of generations. Each generation, every agent in the population spends the user designated number of ticks in the world defined above. The user determines how many agents "survive" each generation to breed, only those with the highest scores are recombined to have offspring. Then a new population of the same size is generated. Each new individual is the offspring of two of the individuals who survived from the previous generation. Its parents can also both be the same individual, representing asexual reproduction or reproduction between individuals with very similar genotypes. The parent genomes are recombined by choosing a user provided number of random points in each chromosome where the child genome switches between parental genomes. Then a user provided number of mutations are introduced. Each mutation happens only at a single allele, whose value is randomly incremented by some value from -.5 to .5. Once the new population is the same size as the original population, it is evaluated on the world and the cycle begins again.

I ran the evolutionary algorithm for 50 generations, on populations of 50 individuals, for 50 ticks per run of the world, with various values of the evolution parameters to determine which was best suited to solving the task. I evaluated a range of values of mutation rate, crossover rate, and survival rate based on which produced the highest average score over the entire population in all 50 generations. This enabled me to find optimal parameter values, which I then used to evolve several populations with different random seeds.

Results

I first evolved populations with different survival rates (Figure 2) - the number of individuals who were able to reproduce and pass their genes on to the next generation. I evaluated survival rates between 2 and 50, and found that the highest average scores resulted from survival rates between 4 and 17. Based on these results, I settled on a standard survival rate of 4.

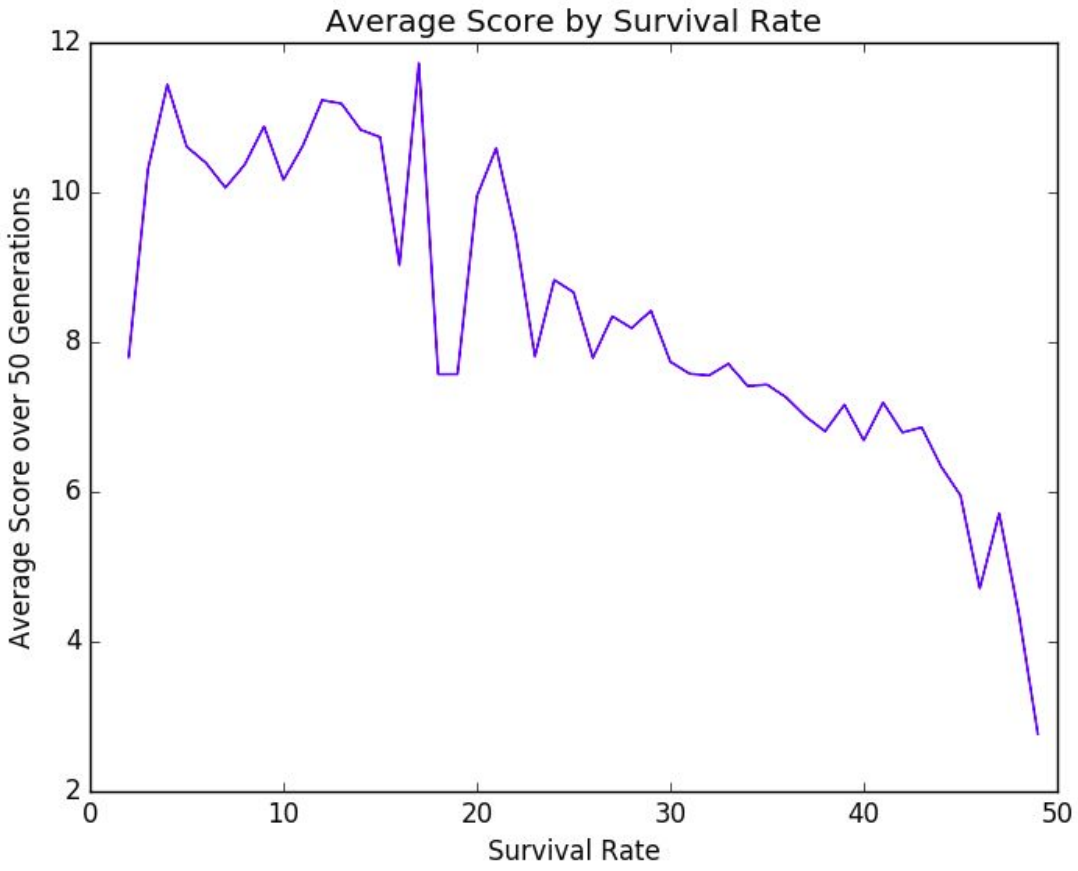

Figure 2. The average score across all populations of 50 individuals in 50 generations with survival rate from 2 to 50, no crossover between individuals' parents' chromosomes, and 1 mutation per chromosome.

I then evaluated the effect of crossover rates between 0 and 5 crossover points per chromosome (Figure 3). I held the survival rate at 4, as determined in the previous test, and set the mutation rate to 0. A crossover rate of 5 produced the highest average score. Interestingly, crossover rates of 1 and 4 performed much worse than the other values I evaluated. To my

surprise, larger crossover rates tended to perform better, which suggests that they introduce more variation which increases the opportunity for better performing genomes to arise by chance and be selected for.

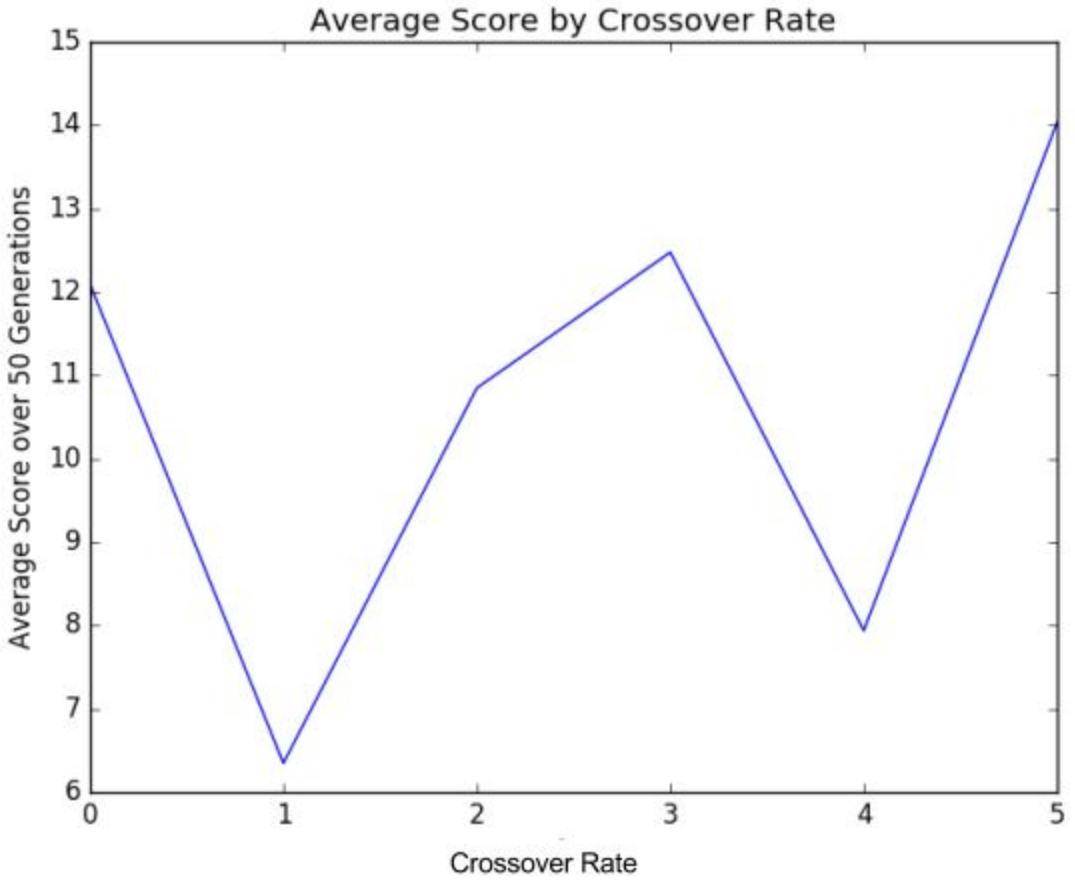

Figure 3. The average score across all populations of 50 individuals in 50 generations with crossover rate from 0 to 5, the 4 highest-scoring individuals surviving to reproduce per generation, and 1 mutation per chromosome.

I also evaluated the effect of mutation rates from 0 to 9. I again held the survival rate at 4 and used two different crossover rates; 0 (Figure 4) and the previously determined optimum, 5 (Figure 5). I found that without any crossover, the optimal mutation rate was also 0 and average score tended to decrease as mutation rate increased. However, with a crossover rate of 5, the optimal mutation rate increased to 2, though again the average score tended to decrease as mutation rate increased.

Figure 4. The average score across all populations of 50 individuals in 50 generations with mutation rate from 0 to 9, the 4 highest-scoring individuals surviving to reproduce per generation, and no crossover between individuals' parents' chromosomes.

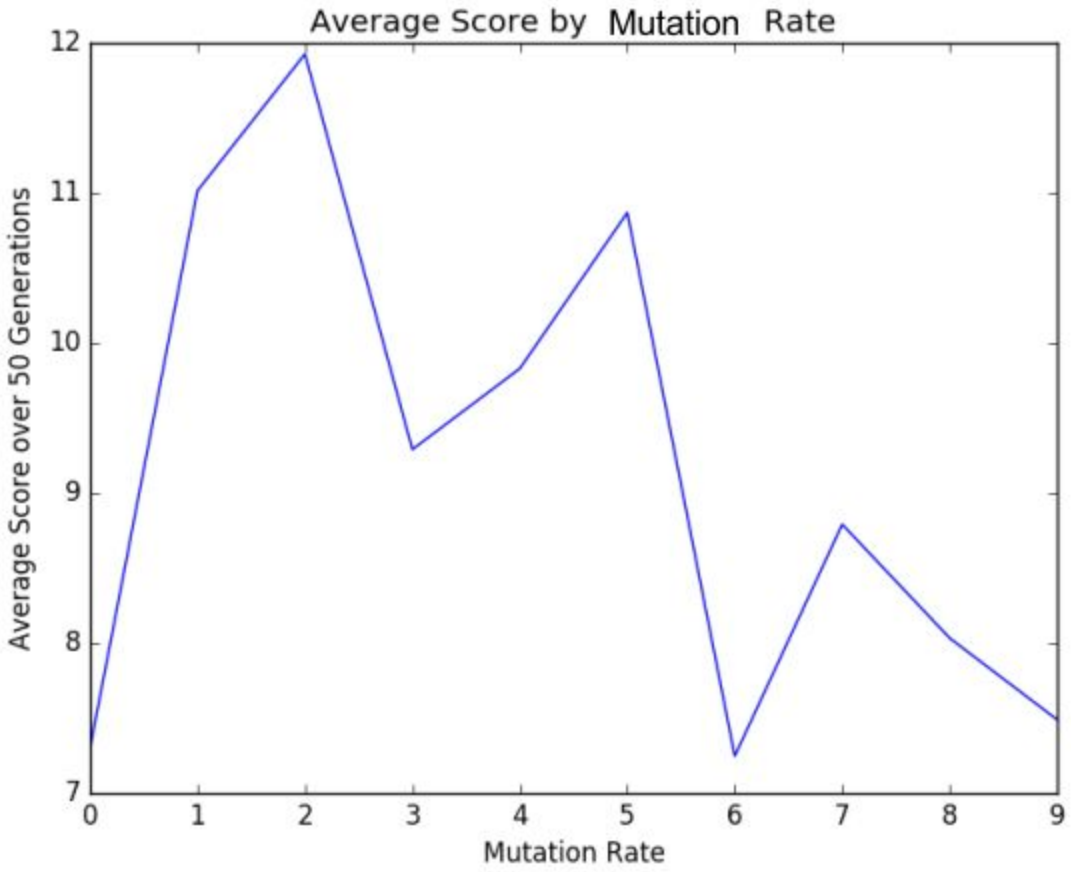

Figure 5. The average score across all populations of 50 individuals in 50 generations with mutation rate from 0 to 9, the 4 highest-scoring individuals surviving to reproduce per generation, and 5 points of crossover per chromosome.

Once I found the optimal parameters, I used them to evolve five different populations defined by five different random seeds and monitored their evolution over 50 generations (Figure 6 - 11). I found that though their performance varied across random seeds, in most of the populations there was marked improvement over the course of 50 generations, indicating that the populations evolved successfully in that time to achieve higher scores in the minimally cognitive task. Most of the populations experienced a large jump in average score in the first few generations and then the average score increased more gradually (Figure 6 - 9). However, of particular note is the steady, if noisy, increase in average score over generation in one population (Figure 10). This stands in contrast with the very small, if not total lack of increase in average score in another population (Figure 11). This indicates that random chance has a large influence on whether a population evolves to perform better at a task beyond the influence of parameter values.

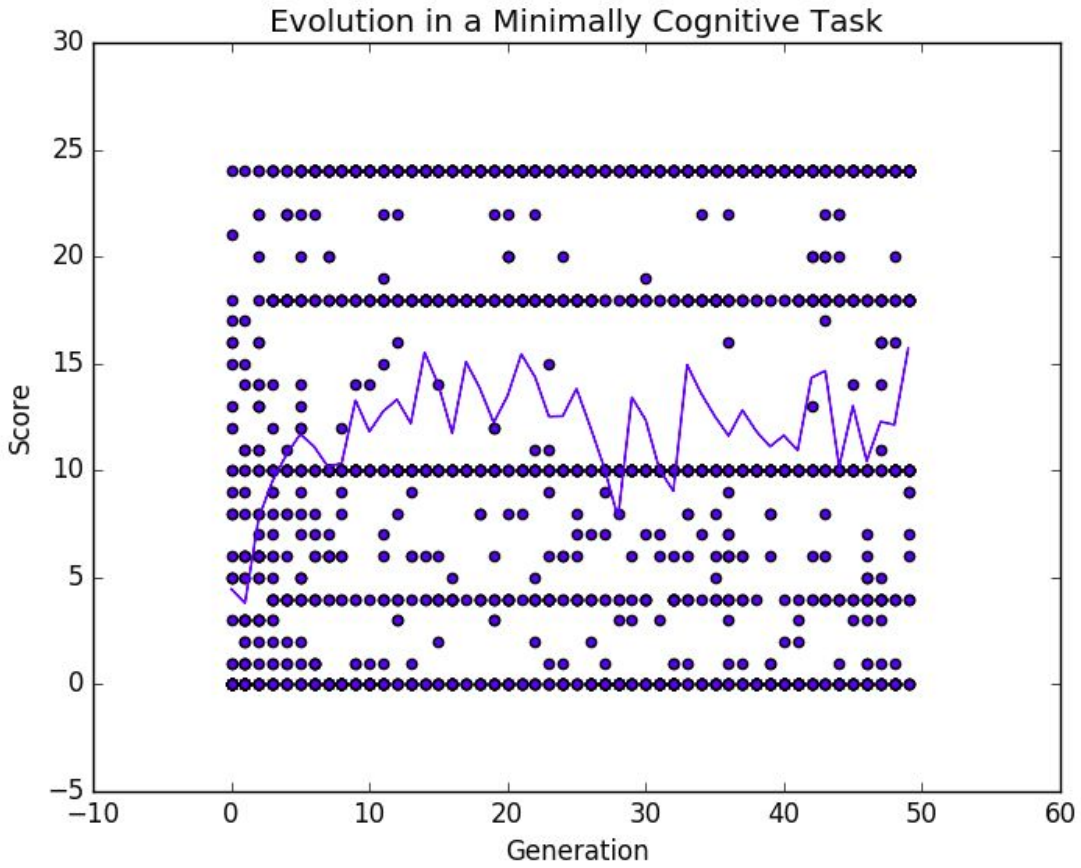

Figure 6. The scores of individual agents in each generation. represented by blue dots, and the average scores of all agents in each generation, represented by the blue line, with mutation rate of 2, crossover rate of 5, survival rate of 4, and random seed of 0.

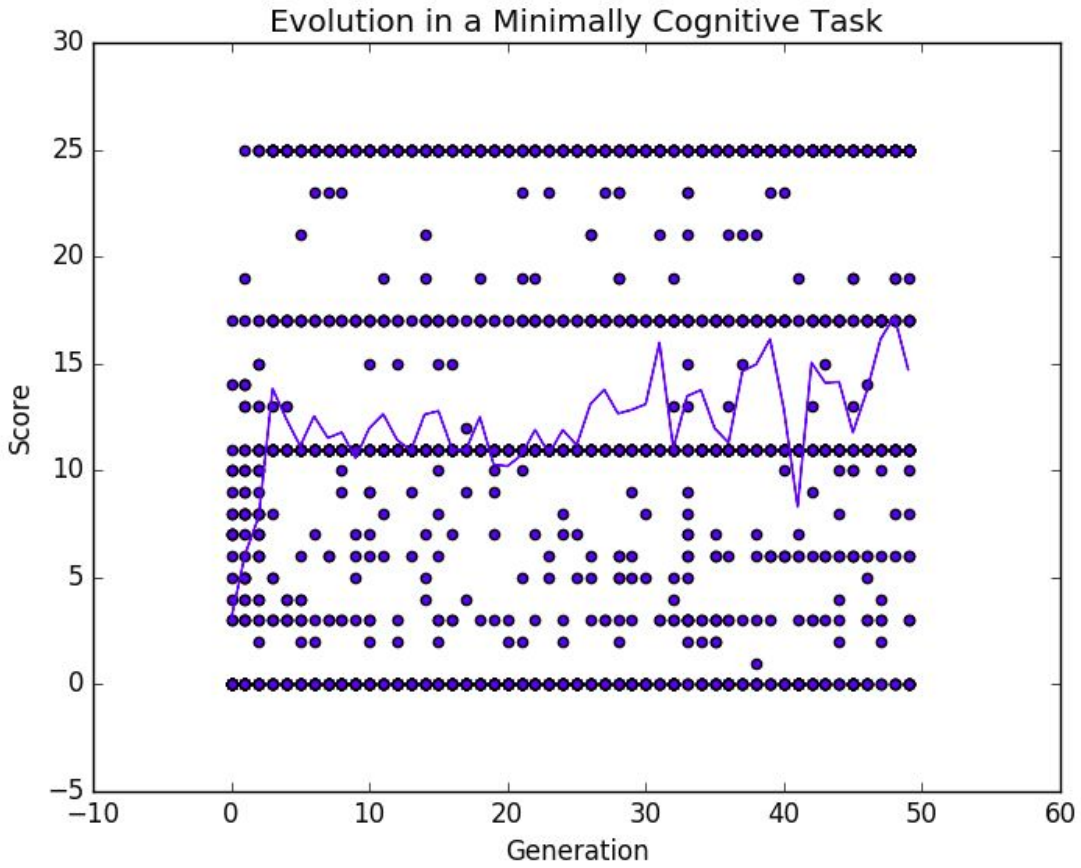

Figure 7. The scores of individual agents in each generation. represented by blue dots, and the average scores of all agents in each generation, represented by the blue line, with mutation rate of 2, crossover rate of 5, survival rate of 4, and random seed of 1.

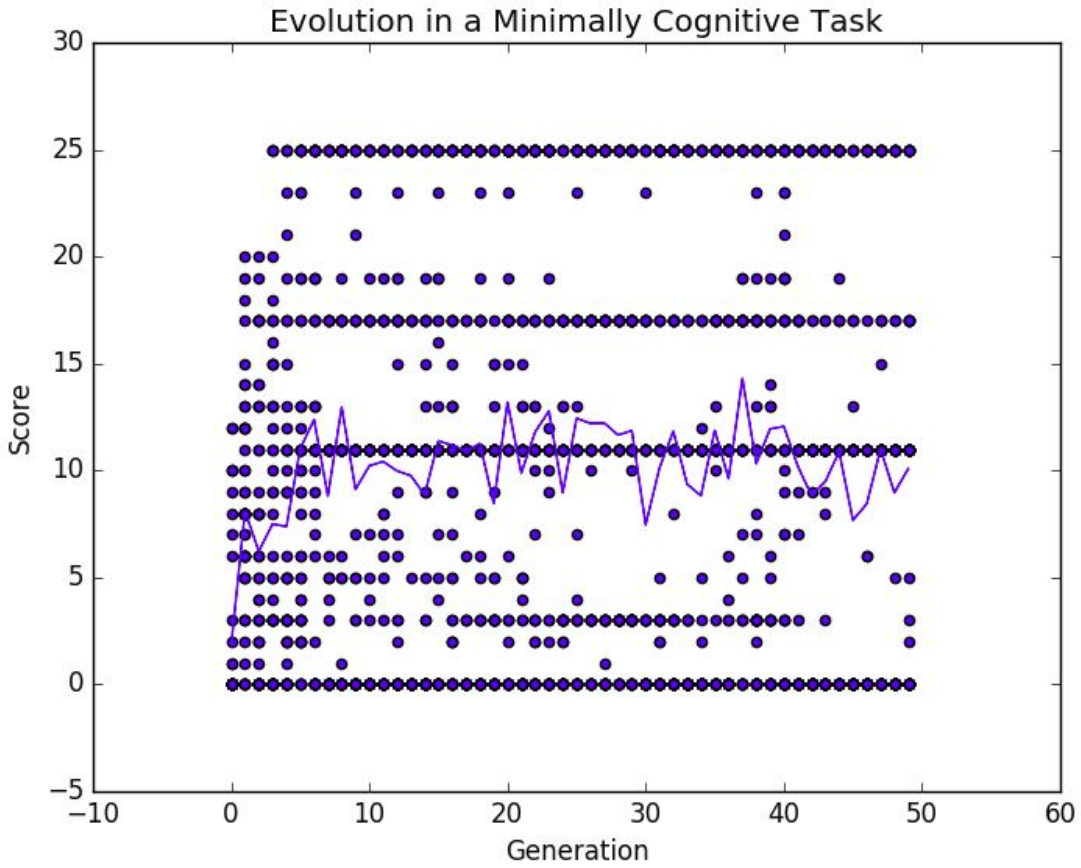

Figure 8. The scores of individual agents in each generation. represented by blue dots, and the average scores of all agents in each generation, represented by the blue line, with mutation rate of 2, crossover rate of 5, survival rate of 4, and random seed of 2.

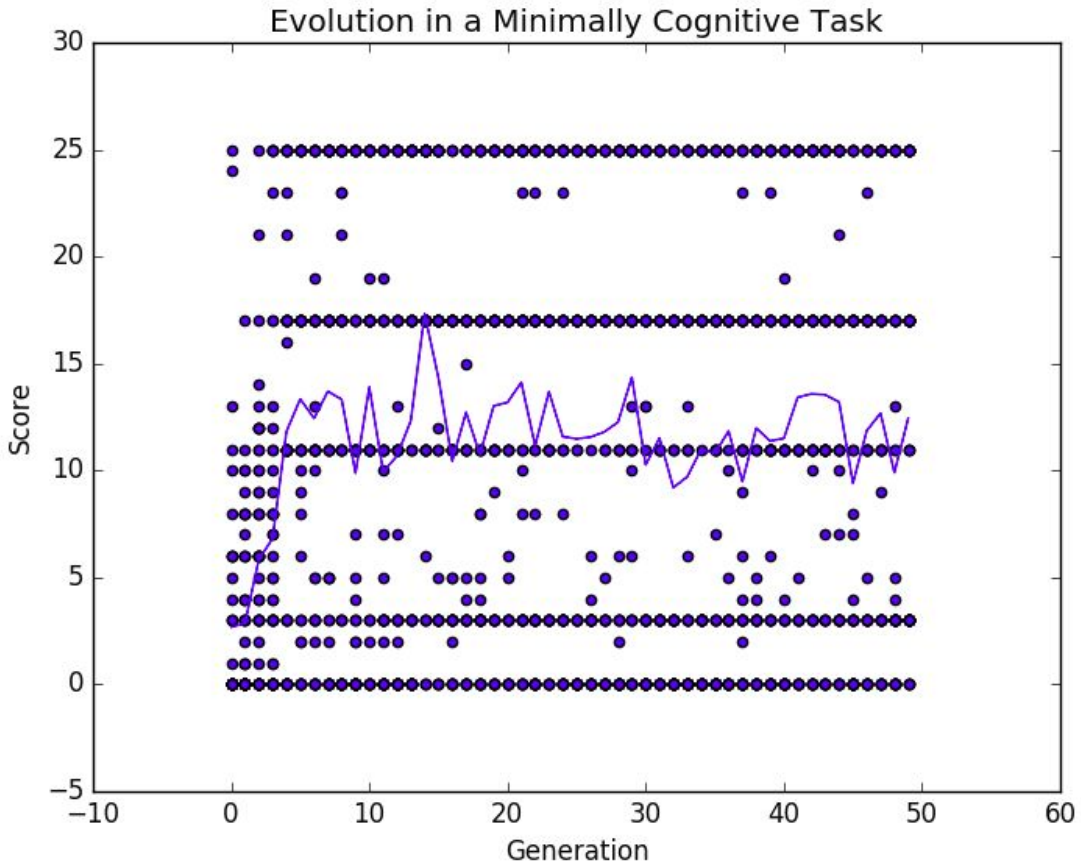

Figure 9. The scores of individual agents in each generation. represented by blue dots, and the average scores of all agents in each generation, represented by the blue line, with mutation rate of 2, crossover rate of 5, survival rate of 4, and random seed of 3.

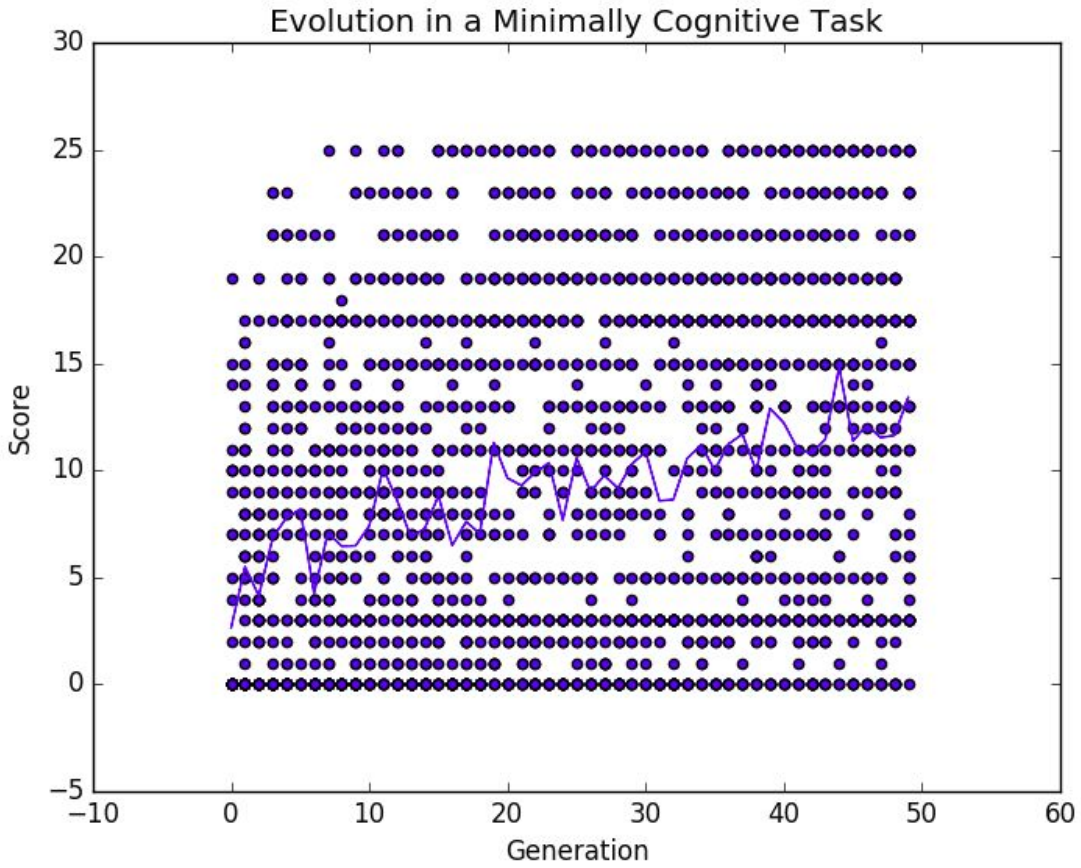

Figure 10. The scores of individual agents in each generation. represented by blue dots, and the average scores of all agents in each generation, represented by the blue line, with mutation rate of 2, crossover rate of 5, survival rate of 4, and random seed of 4.

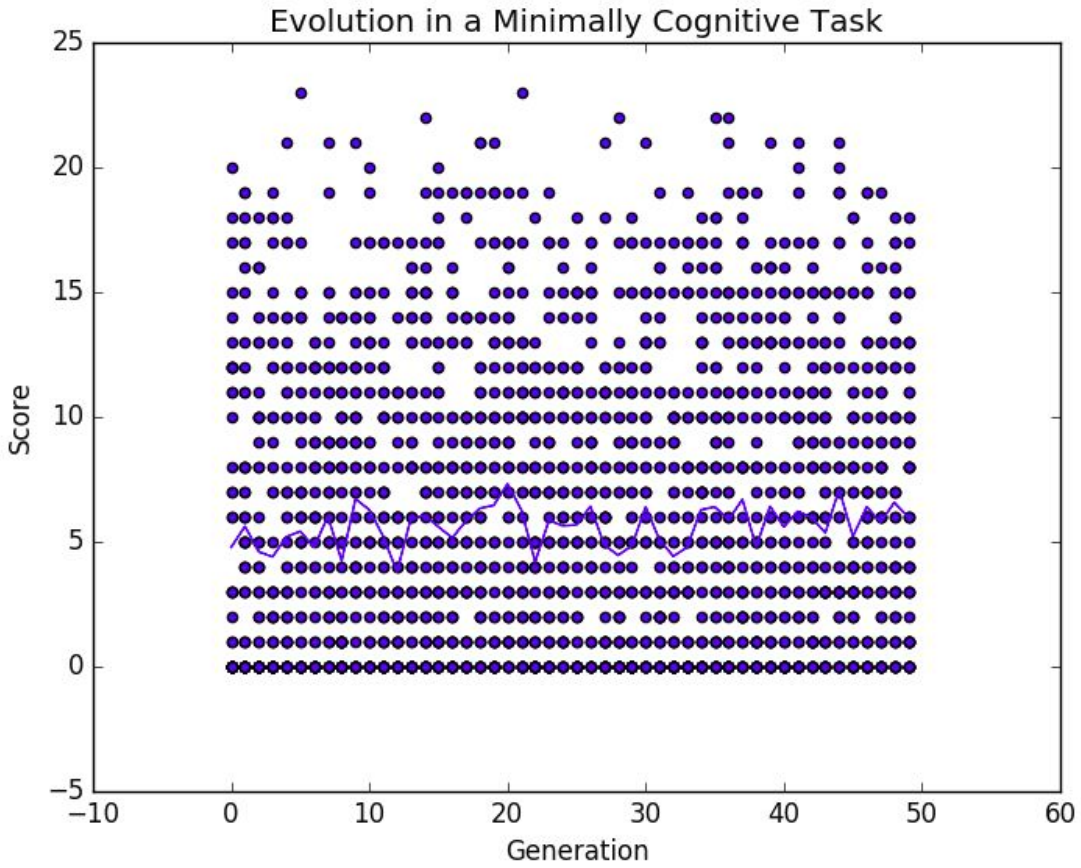

Figure 11. The scores of individual agents in each generation. represented by blue dots, and the average scores of all agents in each generation, represented by the blue line, with mutation rate of 2, crossover rate of 5, survival rate of 4, and random seed of 5.

Conclusion

I successfully evolved continuous-time recurrent neural networks to solve a simplified variant of Beer's minimally cognitive task (2003, 2015). Over the course of 50 generations, populations of 50 individuals improved their average performance at the task. Furthermore, I evolved several populations with different survival, mutation, and crossover rates to determine the optimal parameter values for evolution. I found that populations achieve the highest average score over 50 generations with a relatively high crossover rate, of 5 crossover points per chromosome, and 2 mutations. This increases the diversity of each child population, while limiting the potentially disruptive influence of mutations, possibly increasing the chances of reaching a local optimum in performance.

I used a highly simplified version of Beer's minimally cognitive task to increase the likelihood of evolving a successful strategy that remains stable over generations despite crossover and mutations. If I were to continue this project, I would first extend the task to include objects that start at different locations and vary in size. To further increase the difficulty of the problem, I could make it so that the agent must catch the entirety of an object for it to increase its score. I would also like to include an object discrimination aspect to the task, where the agent must distinguish between objects, based on size for example, to determine which to catch and which to avoid.

In actuality, organisms do not only modify their behavior as populations through evolution, but also through individual learning. In the future I hope to implement an evolutionary algorithm that selects for agents' neural network structures and initial weights, and then allows for the weights to be further tuned through learning. I hypothesize that using evolutionary algorithms to select between agents capable of unsupervised, Hebbian learning would most closely simulate how evolution and learning shape the nervous systems of living organisms in ways that learning and evolutionary algorithms on their own cannot.

References

Beer, R. D. (1995). On the Dynamics of Small Continuous-Time Recurrent Neural Networks. *Adaptive Behavior, 3*(4), 469-509. http://journals.sagepub.com/doi/pdf/10.1177/105971239500300405

Beer, R. D. (2003). The Dynamics of Active Categorical Perception in an Evolved Model Agent. *Adaptive Behavior, 11*(4), 209-243.

Beer, R. D. (2015). Characterizing Autopoiesis in the Game of Life. *Artificial Life, 21*, 1-19.