Brainiac+: Evolving Multiple Variables within the Brain of a Quadruped

Jordan Donovan
University of Vermont
Burlington, VT
jdonova6@uvm.edu

1 PROBLEM STATEMENT

This work focused on evolving the neural network controller, or "brain", of a quadruped robot created during the exercises for the ludobots online course[1]. At each stage in this work, we chose a variable of the controller and let the evolutionary algorithm manipulate and mutate that variable. This allowed for evolution to search in a larger search space since each new variable supplied to the algorithm provided a new dimension to this space.

Initially evolution was allowed only to mutate the weights of the neural network controller, meaning that the architecture itself was constant and did not change over evolutionary time. This architecture involved 9 touch-sensor neurons (attached to each portion of leg and to the torso), 1 hidden neuron, and 8 motor neurons (attached to each joint in each leg). This evolutionary algorithm is the baseline ("A") in our comparison ("AB") tests.

The Brainiac+ version of the evolutionary algorithm was allowed to manipulate several variables within the neural network. These variables included the architecture via hidden neurons and the weights of all synapses including recurrent and self-connecting synapses. The different types of synapses included feed-forward synapses between sensor->hidden neurons and hidden-motor neurons as well as recurrent and self-connecting synapses between hidden->hidden neurons. The algorithm was also allowed to mutate the activation, or threshold, function for each of the hidden neurons in the network.

The hypothesis here was that if evolution was allowed to search the space provided by these new variables, it would be able to optimize for a controller that had a higher capacity for success. The goal was to see if it could find a more powerful and

[1]https://reddit.com/r/ludobots

CS 206: Final Projects, May. 10, 2022, UVM

.

higher-performing controller for the quadruped robot. The performance was a measure of how far the robot traveled in a single specified direction given a specific amount of time.

2 METHODOLOGY

The process of implementation for this work involved four main phases, with the first three of these supplying new variables to the evolutionary algorithm, and the fourth running several iterations of tests.

In the first phase a new variable supplied to the evolutionary algorithm was the number of hidden neurons. The mutations added during this stage of the process allowed for the evolutionary algorithm to add hidden neurons. These hidden neurons had randomly-weighted incoming synapses (from each sensor neuron) and outgoing synapses (to each motor neuron). In addition, self-connecting synapses were created for every hidden neuron with random weight. All of these synapses had equal probability of being mutated in the mutation function.

The second variable supplied to the evolutionary algorithm was the addition of recurrent connections and thus the ability to mutate these connections. This was chosen because many times recurrent connections can lead to memory within the neural network which can increase performance. Recurrent connections with random weights were added from every hidden neuron to every other hidden neuron. The mutation function was then altered to give equal probability to all synapses in the network of being modified and a decreasing chance of the addition of hidden neurons over time.

The last variable supplied to the algorithm was the ability to mutate the activation, or threshold, functions for each of the hidden neurons. This involved manually adding functions to the pyrosim library. The functions that evolution was allowed to select included: tanh (default), ReLu, Leaky ReLu, sigmoid, and sin. These are some of the most-commonly utilized activation functions within the field of deep learning.

The fourth and final phase of the implementation involved several iterations of running each of the algorithms for comparison. The first step in this phase was to implement logging. This work used the wandb[2] python library to log data. Doing so, allowed for comparing runs across time and with different implementations quite easily. The experiments were conducted with a population

[2]wandb.ai

size of 50 and a generational budget of 200 and repeated for 10 runs.

3 RESULTS

The main result of this work shows two comparisons ("AB tests") of the quadruped with the baseline controller described in the previous section ("A") and the "Brainiac+" controller with additional, evolvable variables of hidden nodes, recurrent and self-connecting synapses, and activation functions ("B") with a fitness corresponding to displacement along the y-axis into the screen (negative direction).

Figure 1 plots the average fitness of the best fit individual at each generation for each type of controller over 10 runs with standard deviation shading. In the case of the baseline controller, one can see evolution occurs with fitness increasing (negatively) throughout generational time. It is also apparent that evolution occurs in the case of the "Brainiac+" controller showing a similar trend. By comparing these two curves, we can see that the controller with additional variables supplied outperforms the baseline controller in regards to this particular metric.

Figure 2 plots a different, but similar, metric: the average of the average fitness of the population at each generation for each type of controller over the same 10 runs as above with standard deviation shading. This plot again shows the "Brainiac+" controller outperforming the baseline controller. This plot also shows us that the averages for best fitness in each generation tend to be just about double the averages for average fitness of the population. Another observation of the occurrence of evolution resides within this plot where the initial population can be seen as a random instantiation and the final population as the evolved population where the evolved population outperforms this random, initial population significantly.

The behaviors of the robots evolved by each of these controllers were also unique to each other. The evolved baseline controller produced a quadruped with little regularity to its movements and slow progress, whereas the evolved "Brainiac+" controller produced a quadruped with more regularity and cyclic behavior to its leg movements that traveled much more quickly.

4 DISCUSSION

This work required several design decisions and resolutions to difficulties. The A/B variants were chosen purposefully to show that the changes to the controller had positive effects on fitness. The

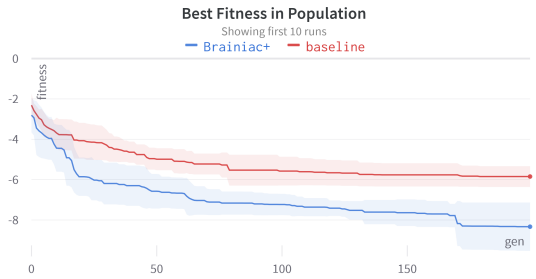

Figure 1: Average of highest fit individual for each generation across 10 runs for each controller type.

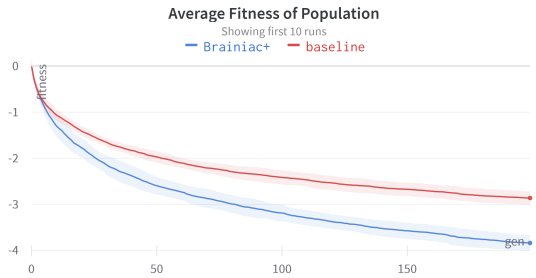

Figure 2: Average of the average population fitness for each generation across 10 runs for each controller type.

phases, or deliverables, described were intentional in that the researcher wanted to experiment not only with a hard-coded neural network architecture but also with an architecture that evolution was allowed to grow and change over generational time. Additionally, the mutation function that was implemented involved several difficult design choices regarding the probability of each mutation. Each synapse is given equal probability, but the chance to add a hidden neuron or to change the activation function are added separately and are proportional to the number of hidden neurons that exist in the individual controller in question. The probability of a hidden neuron being added decreases as the number of hidden neurons grows, but the probability of mutating an activation function for a hidden neuron increases as this number grows.

Another aspect of this work was editing the pyrosim library to include new functionality for the activation functions. This is simple once one is familiar with the library, but requires some knowledge of pyrosim to begin.

If more time were given to work in this area, the author would encourage a more incremental approach to growing the neural network since the final version of this work adds several, randomly-weighted synapses upon adding a hidden neuron. One can also imagine that altering the mutation of synapses to include more diverse sets of weights would be beneficial as well. Both of these changes should be relatively simple to perform. However they might be time-consuming to implement well. In a more global view, implementing CPPNs within the evolution of controllers or as the controllers themselves could be an interesting direction to take this work. This would most likely involve several changes to the pyrosim neural network

library or at the very least a bit conversion code to communicate with pyrosim.