EE304 Project

Matthew Feldman, Jacob Baldwin

June 8, 2016

Abstract

In this project, we created a simple world populated with creatures and food. The goal of the creatures is to survive as long as possible, which involves eating the most food and avoiding the most predators. In order to control the decision logic of the creatures, we used Leaky-Integrate-and-Fire (LIF) neurons in neuromorphic networks. We analyzed the effectiveness of different neuromorphic parameters, namely number of neurons in the networks and synaptic time constants of the neurons, to see which creatures perform the best. We also experimented with the Basal Ganglia node in Nengo to help the creatures make stable, reasonable decisions based on input stimuli. We found that there is a trade-off between brain complexity and survivability, as the more complex brains may make better decisions but they do so at the expense of high energy consumption rates. Similarly, creatures with small synaptic time constants generally performed better because they were able to make quick, impulsive decisions while they roamed. However, the creatures with extremely low synaptic time constants and a small number of neurons behaved more erratically and would probably not have done as well if food were harder to find in the environment.

1 Introduction

In the University of Florida Museum of Natural History, there used to be an interactive exhibit that taught the idea of allopatric speciation by allowing visitors to divide a pool of fish into different groups and watch them evolve separately. The fish swim around looking for food and prey, while avoiding predators. The exhibit and creatures' behavior was originally programmed in Processing by Ian Elsner, a museum exhibit contractor at Richard Lewis Media Group, but we thought it would be an appropriate place to integrate neuromorphic systems.

The way the game works is that there are creatures roaming around and food spawning periodically at random. The creatures have antennae sensors to detect what is in front of them and will seek food and prey but avoid predators. A creature interprets a predator as another creature that is significantly larger than itself and a prey as one that is significantly smaller. The creatures shrink continually as they roam, and grow when they eat food or prey. If they shrink too small, they become "starved" and their body turns gray and stationary until a creature comes along and consumes it. Creatures respawn at random after being eaten. Figure 1 shows a screenshot of our game, along with the diagnostic tools we used to develop it.

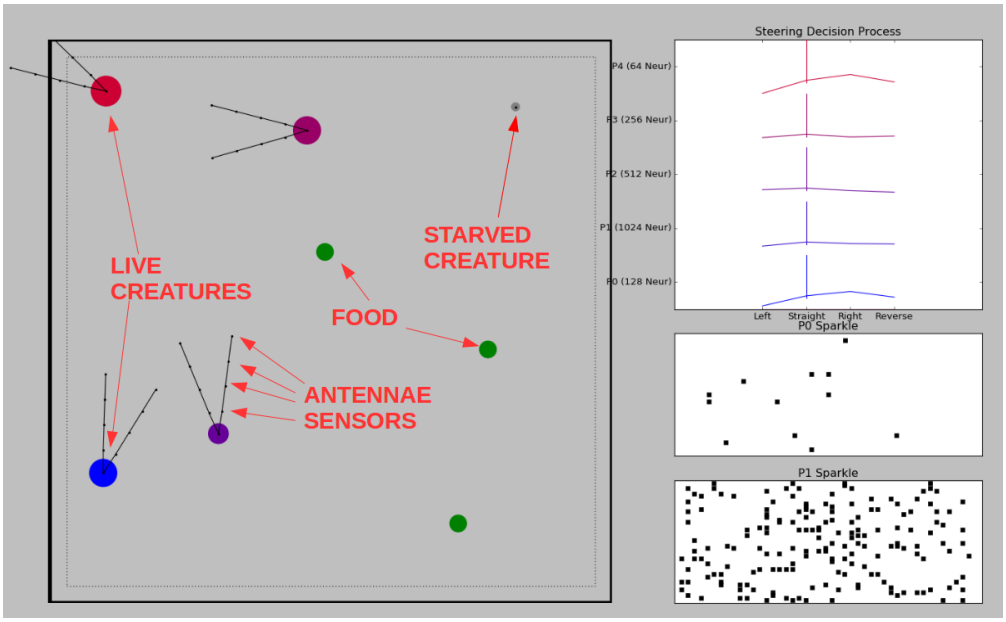

Figure 1: Screenshot of the game.

We downloaded an old prototype of the code and reimplemented it in Python. The old version of this code used many nested if-then-else statements to control the creatures' behavior, which inherently lacks any sensor noise that real creatures would have, delay between sensing and decision-making, and the penalty for crunching more data to make a decision. By using a neuromorphic system, we wanted to be able to explore these tradeoffs with respect to system architectures that use Leaky-Integrate-and-Fire (LIF) neurons to see how we can create the fittest creature for this particular task and environment.

We also wanted to experiment with the Basal Ganglia, which is a brain structure that can take various input stimuli and decide on which action to take such that voluntary behavior can be performed smoothly. There is a Nengo object that simulates this behavior, which we can plug directly into our network to make the final call on which action creatures should take.

2 Technical Approach

We started by implementing the game in Python, carefully planting the hooks we need to drive the creatures with neuromorphic systems. In order to make it easy to plug-and-play various neuromorphic parameters, such as decision logic, synapse time constants, and ensemble architectures, we divided the program into various modular components.

First, we created the physics engine that takes care of the environment and "hard" interactions, such as bumping into walls, eating food and prey, and determining what item is physically touching a creature's sensor. This engine was inspired by the museum exhibit and was a straightforward translation of the original Processing code into Python so that we could plug in Nengo.

Next, we set up all of the framework to allow the creatures to sense and make decisions with Nengo objects. This involved creating a sim object but manually updating the stimulus for each Nengo.node sensor in real time.

Then, we set up the animation framework using the matplotlib.animation library. Inside the animate method of this library, we included all of the physics to update the creatures from frame to frame, including updates to angular and linear acceleration, total Nengo spike counts, calorie burning and consumption, and creature decision-making. We also implemented other useful hooks into the program, such as decoders for Basal Ganglia outputs, neural populations' stimuli, and sparkle plots. Throughout a run, the program tracks various events that help us analyze the performance of the various creatures, such as calories consumed, number of times eaten, and number of times starved.

Finally, we abstracted away all of the Python details by importing a config file that allows us to change any property of the simulation, including neuromorphic parameters (number of neurons per creature, synapse time constants, the Basal Ganglia decision functions, etc) and physical properties (size of the field, scarcity of food, Ecological efficiency, rate of starvation, geometry of antennae, etc).

With this framework set up, we were easily able to compare the performance of different creature configurations. After a simulation is cancelled by the user, a table displaying all of the events is shown. We also maintained the original version of the program with boolean logic in the "brain," which we could use as a reference for what creature behavior is reasonable.

We chose to have a simple network with two nodes that independently accumulate left- and right- stimuli, and then connect both of these ensembles to a Basal Ganglia node with a recommendation function. Each antenna node is a 3-dimensional node that reports the properties `Edible_Item`, `Predator`, and `Wall`. The connection between the accumulator ensembles and the Basal Ganglia computes recommendation weights based on the desirability of objects detected by each antenna for each of four actions `Turn_Left`, `Straight`, `Turn_Right`, and `Reverse`. Figure 2 shows this architecture beside a creature to give a sense of the physical and conceptual geometry.

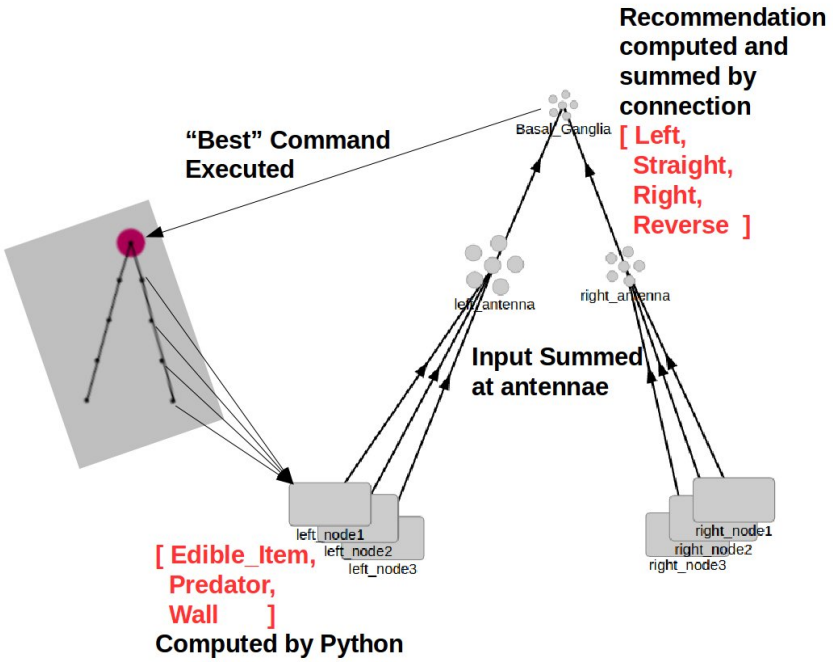

Figure 2: Nengo architecture alongside screenshot of creature.

The Basal Ganglia node in Nengo consists of five ensembles, which were designed to mimic the way a real Basal Ganglia works. Namely, these populations are:

- Striatal D1 dopamine-receptor neurons (strD1)
- Striatal D2 dopamine-receptor neurons (strD2)
- Subthalamic nucleus (stn)
- Globus pallidus internus / substantia nigra reticulata (gpi)
- Globus pallidus externus (gpe)

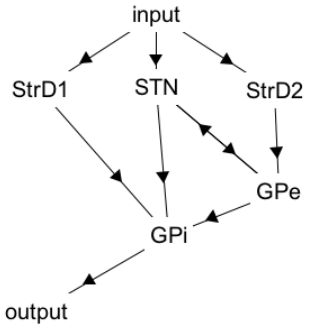

Figure 3: Basal Ganglia architecture that is included in the Nengo package.

Figure 3 shows the way these populations are connected. This network simulates both the direct (excitatory) and indirect (inhibitory) pathways of a biological Basal Ganglia. The advantage of using this node over a one simple neuron population to sum the utilitiy functions and decide on an action is that this node offers both nonlinear signal conditioning and action stability. The node drives the best action at a value of 0 and the rest are forced negative, and the various populations provide more complex filtering and therefore more stable decisions by forcing the "winning" action to high for more time before it is chosen.

Finally, the equations that we decided were reasonable ways of transforming antennae data into action recommendations is shown below. Each node in an antenna reports either a 1 if the item was detected and a 0 if not. These values get summed up and are the input array, x, to this function. We then multiply each element by a pre-configured gain as a form of affinity and aversion to these objects. We chose to have the creature mainly reverse when there is a predator, turn right when there is a wall, and seek anything edible. In this snippet, "food" refers to starved creature bodies, food, or prey equally.

```
def fcn_left(x):
    food = x[0] * food_gain
    predator = x[1] * predator_gain
    wall = x[2] * wall_gain
    # Recommendation = [left, straight, right, reverse]
    recommendation = [food - predator - wall,
                      no_action,
                      predator + .5*wall,
                      backup_gain*predator]
    return recommendation


```

```
def fcn_right(x):
    food = x[0] * food_gain
    predator = x[1] * predator_gain
    wall = x[2] * wall_gain
    # Recommendation = [left, straight, right, reverse]
    recommendation = [predator - wall,
                      no_action,
                      food - predator + wall,
                      backup_gain*predator]
    return recommendation
```

Listing 1: Decision functions

Because the goal of our project is not to perform rigorous calculations or to achieve any particular level of accuracy, we will measure the quality of our approach in two ways. First, since our goal was to implement a museum exhibit, we will judge our approach by watching the running simulation, to see that creatures are behaving more or less as we would expect. Second, since another goal of our project is to explore trade-offs in neural systems, we would judge our approach on how well it allows us to vary neural configurations and observe the effects of these changes. Our results show that our approach achieves these two goals reasonably well.

3 Results

The most important result of our project is the running simulation. The creatures generally behave as expected. They move towards food and away from predators and walls. Creatures with smaller numbers of neurons make decisions that change frequently and are more whimsical, which makes sense, because their calculations will have more variability. These quick-changing decisions result in a somewhat jerky movement of the creature. Similarly, decreasing the synapse time constant makes creatures' decisions fluctuate constantly. Increasing synapse time constants smooths out this behavior, but if they are made too large, the creatures never react in time to successfully track targets, so they generally wind up moving in circles. All this behavior makes sense based on our understanding of neuromorphic systems, so we believe that our system successfully implements the exhibit using neuromorphic systems, which was the primary goal of our project.

After a simulation has been observed, we looked of the summary data describing how well each creature did. We performed experiments comparing different numbers of neurons and synapse time constants. Results of various runs are shown in Figure 4. The fields that we keep track of are:

- **Neurons** - number of neurons in each antenna ensemble. The total number of neurons for the creature is therefore $2 * Neurons + BasalGanglia$
- **Synapse Tau** - time constant of the connections between each antenna ensemble and the basal ganglia.
- **Survival Time** - total time in the simulation each creature spent alive. This number increments on every frame unless if the creature was recently eaten or is in the starved state.
- **Calories Consumed** - the amount of food eaten by each creature, including prey that it ate. This is a measure of how effectively the creature found food.
- **Times Eaten / Starved** - how often a creature starved to death or was eaten by a larger creature.

ID	Neurons	Synapse Tau	Survival Time	Calories Consumed	Times Eaten	Times Starved
Creature 0	64	0.005	4253	888	0	0
Creature 1	128	0.005	4153	436	2	0
Creature 2	256	0.005	4103	325	3	0
Creature 3	512	0.005	3857	120	8	0
Creature 4	1024	0.005	3232	92	11	1
Creature 5	2048	0.005	1779	30	5	4

ID	Neurons	Synapse Tau	Survival Time	Calories Consumed	Times Eaten	Times Starved
Creature 0	128	0.001	3598	845	2	0
Creature 1	128	0.005	3596	572	2	0
Creature 2	128	0.01	3348	270	7	0
Creature 3	128	0.02	2950	105	15	0
Creature 4	128	0.05	3196	134	10	0
Creature 5	128	0.1	3196	141	10	0

ID	Neurons	Synapse Tau	Survival Time	Calories Consumed	Times Eaten	Times Starved
Creature 0	64	0.001	7353	299	25	0
Creature 1	64	0.005	8599	2094	0	0
Creature 2	64	0.01	8101	554	10	0
Creature 3	128	0.001	7901	628	14	0
Creature 4	128	0.005	7803	839	16	0
Creature 5	128	0.01	7405	452	24	0

Figure 4: Creature summary output for various numbers of neurons and taus

Based on this data, one can make a few conclusions. First, creatures with small numbers of neurons perform significantly better than creatures with large numbers of neurons. They survive longer, find more food, get eaten less, and starve less. This presumably happens because the energy saved by having fewer neurons greatly outweighs the reduced accuracy of calculations. Second, creatures with small synapse time constants tend to survive better than creatures with large synapse time constants. This is presumably because they can react more quickly, and this benefit again outweighs the loss of accuracy caused by less filtering. These results might differ if the calculations the creatures were performing were more complex. In this simulation, speed and energy efficiency is much more important than accuracy.

Results like these, and the conclusions based on them, show that our system can be used to analyze trade-offs in neuromorphic systems based on real life. Many more configurations could be changed, and the simulation could be modified to model different situations, but our results show that our project could be used as a foundation for such investigation.

4 Discussion

The original goal was to reimplement the museum exhibit with neuromorphic brains, rather than boolean brains, but the process of translating it into Python and providing simple hooks ended up transforming into something that resembled the populor MMO, Agar.io. We received some feedback that it would have been interesting to interface the Nengo behavior with Javascript to create an AI that would actually play this game with other people (or AI) online in real-time.

While we eventually figured out how to run Nengo simulations in real-time inside of our program, it was surprisingly convoluted to do this. Througout the course, we became familiar with running simulations for fixed durations, and then analyzing the data all at once afterwards, but we needed to dig into the Nengo source code to figure out how to run in real time.

The original game was intended to only show how creatures can evolve separately when geo-

graphically isolated, the game that we created also showed a toy example of the trade-offs between complexity and survivability. While the real world is many orders of magnitude more complex than this simulation, we were surprised that there actually seemed to be an optimal number of neurons in-between the two extremes that created the best creature.

If we had more time, it would have been interesting to change the way the game is played. One of the problems with testing various neuromorphic parameters was that food was probably too easy to find without much "thought" by the creatures, giving the advantage to creatures who wander around aimlessly because they have too few neurons to properly drive the Basal Ganglia. There was a wide array of other parameters that we could have played with and that would have greatly changed our results. For example, we could have added more sensors on the creatures' antennae, made the antennae longer, or changed the affinity-aversion parameters to make the creatures more afraid of predators and more greedy for food. By exploring these parameters, it would be interesting to watch how the Basal Ganglia grapples with recklessly seeking food versus having a strong phobia of predators.

Also, by reading further into the Nengo documentation, we realized that the Basal Ganglia was intended to be used in tandem with a Thalamus. In a biological brain, these two structures work together to choose actions of high utility and then inhibit all of the other actions. We followed a Nengo example on using the Basal Ganglia on its own, but it used a simple winner-take-all probe to extract a decision from the node. If we had used a Thalamus, this node would have taken care of suppressing all of the actions except for the one with highest utility, and therefore giving the creatures fully neuromorphic controllers from sensory input to decision output.

Additionally, If we had more time, we would have continued to prune the parameters for all of the creatures to create a sense of "offspring" that fuse together the best of both worlds between competing creatures to continually create better creatures. We also could have used machine learning techniques on the fly to permit the creatures to change their own parameters each time they die to let them converge on a more optimal configuration. If we had a version of the program that we could run on a cluster without animation to collect larger amounts of data more quickly, this would have helped us draw stronger conclusions and provide more feedback for how to optimize the creatures.