Artificial Neural Network Simulation

Aishwarya, Amritanshu, Anand, Matthew, Nayana

May 4, 2015

Goal

Artificial Neural Networks (ANNs) are often implemented in a very simple way, modeled simply as a network of nodes with associated double values representing a propogating signal. This model is abstracted from the biology by reducing the nodes to indices in matrix operations, which carry out the computations of the simulation. The value of this approach is to create extremely large, efficient neural networks. Our goal is to produce a smaller, more biologically accurate ANN designed with visual simulation in mind.

We will use an object oriented MVC architecture in Java with classes for neurons and synapses rather than a linear algebra computational engine. Additionally, we will model multiple neural modulators, specifically using gas dispersion. Then, we will use a genetic training algorithm rather than a more mathematical backpropagation method. Finally, we will develop a user interface which gives visual cues to the user to improve intuition about the operation of the network.

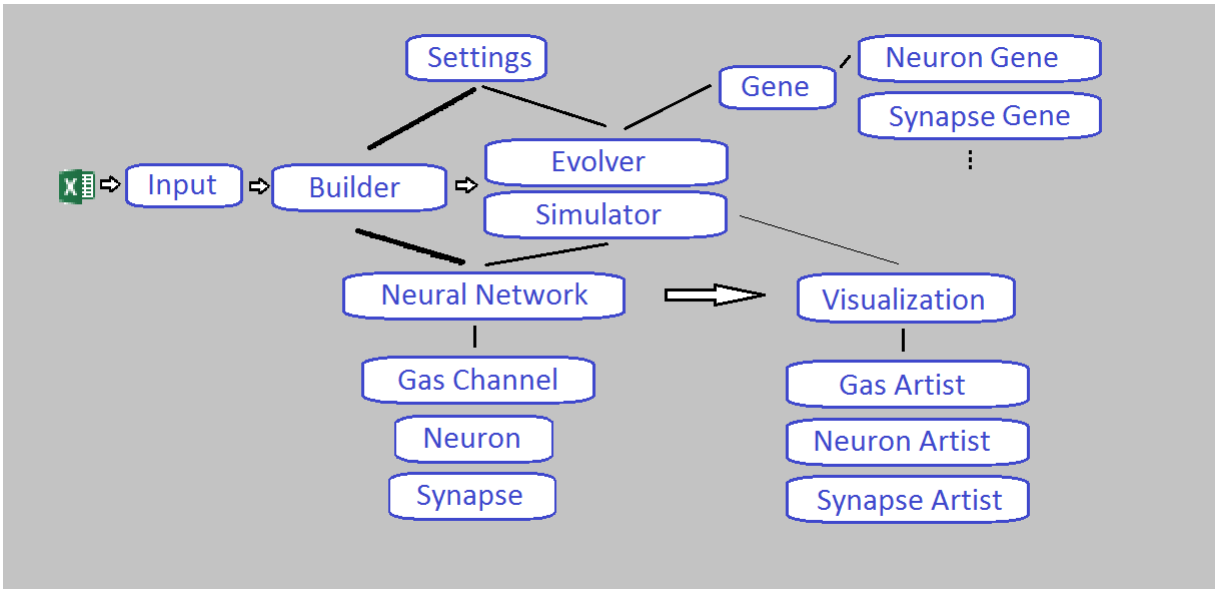
Milestone Overview

- Week 1: Neurons with synaptic connections.
- Weeks 2 and 3: Visualizations of the basic network.
- Weeks 3 and 4: Hebbian learning, gas emission, and gas activation. Additional visualization features.
- Week 5: Receptors, gas modulation, and visualizations of gas concentrations.
- Week 6: Genetic enconding of the network, and a genetic learning algorithm.

Architecture Overview

The design employs an MVC pattern for the neural network simulation with an additional frame for user selection and initialization of the simulation or the evolution procedure.

The simulator employs a timer and has two modes, one in which it runs an input signal all the way through the neural network in each time tick (only useful for feed forward networks) and another in which it takes new input at each time tick and runs the inputs through in a more "real time" fashion (required for recurrent neural networks. Similarly, there is a distinct visualization for each kind of network. The recurrent network collects and modifies the color of neurons with sufficiently high concentrations at each time step, showing the signal propagating in time. The feed forward network traces the signal through the entire network, displaying all of the neurons activated along a given input's path through the network. The network evolver makes use of a display-less simulation mode for testing networks but also has access to the timer method for displaying networks with the top fitness. Its Evolver class uses a network to initialize sets of random Genes that represent the network, a family of genomes. Then it maintains the population of genomes and sends networks one by one to a FitnessComputer class for simulation and evaluation. Then the Evolver produces a new generation of copied, blended, and mutated genomes from the fittest genomes.



Milestone Detail

Synaptic Connections

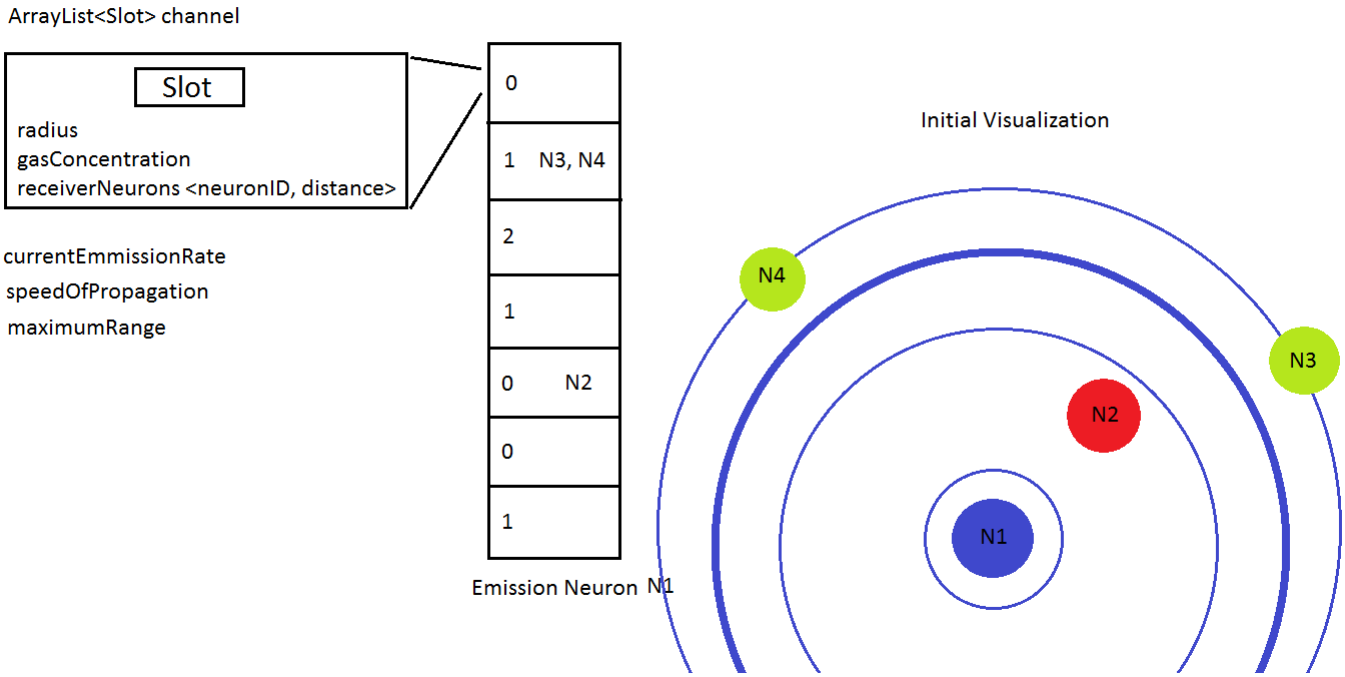
In the first week, we developed the skeleton of the model. The Neuron, Synapse, and NeuralNetwork classes are built and populated with data from an Excel workbook.

MVC

In the second week, we added on a set of classes that take the NeuralNetwork and paint it into a panel. We made a second way to simulate, where input propogates in a time-dependent way (so that prior signals could affect the processing of subsequent signals), and each of our two simulation methods have distinct display classes. We also built a set of utility classes for painting each sort of component, so that new display algorithms can easily be built from existing component display methods.

Gas Emission

Our third milestone was to begin to make our networks into GasNets, by having neurons emit gas, tracking the emissions, painting them, and having neurons which could receive and activate on gas signals.



Hebbian Learning

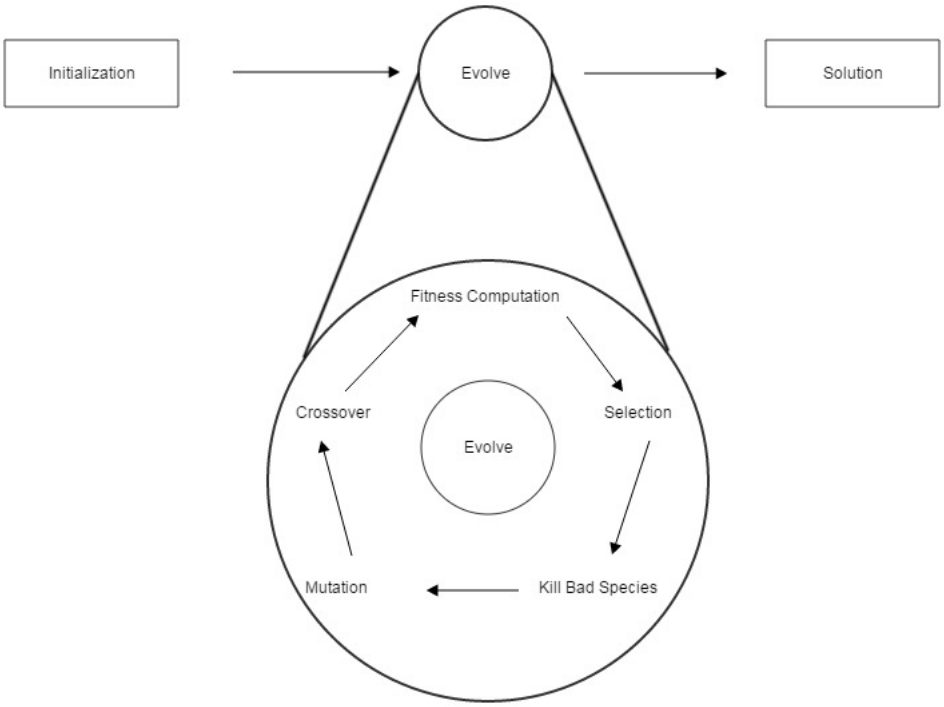
"Neurons which fire together, wire together" is the mnemonic for Hebbian learning, in which our Synapse classes track the activations of the neurons they connect and adjust the strength of their signal transmission accordingly. The function which adjusts the synaptic weight of the synapse when it learns also relies on a parameter called plasticity.

Gas Modulation

Another component of all GasNets that we added in our fifth milestone is gas modulation. Specifically, we needed to have different gases which not only could activate some Neurons, but modify the synaptic activation of others and also control the learning of synapses. To do this, we added a Receptor class which stores user-created modulation functions and the concentrations of any gases present at the Neuron to which the Receptor is attached. This structure allows different neurons to respond differently to multiple gases and gas mixtures.

Genetic Learning

Finally, we created a set of Gene classes for the components of our model so that they could be encoded in a genome structure, a list of genes which cointain the algorithms to mutate themselves and write their genetic code into networks. With this genetic structure in place, we built algorithms which would breed new generations of genomes form a starting population, evolving the population through selecting by a fitness value, calculated using the error in the network's output relative to a user-generated set of input/expected output pairs. To allow for improvement beyond the initial population's genetic code, we have written algorithms to crossover and mutate genomes.



Next Steps

The client's goal is to have a somewhat user-friendly research tool which can be extended to build additional networks and will scale to display them in a clear way. Therefore a big part of our last few weeks was the clarification of requirements the client had for extensibility and the development of clear documentation that suited the clients needs for taking over as both user and developer of our code base. This also drove our development of proofs of concept for each milestone. These will serve as cases for our client to understand our code and debug future changes.

Conclusions

Our team operated as a learning group with small weekly tasks and a miniature project to confirm completion of the task. It was a dynamic environment in which we tried to share and change responsibilities every week. We found that a small amount of work to establish a structure and monitor its activity can be enough to solve a vast array of problems, both with artificial neural networks and in our team.