A Computational View of C. Elegans Neural Circuit Learning, Utilization, and Plasticity

George Shaw
MAS.881 - Principles of Neuroengineering
Massachusetts Institute of Technology
Email: gshaw@media.mit.edu

***Abstract*—This paper describes a system for modeling neural circuits in the nematode C. Elegans, as well as several experiments in building and training such models.**

I. Introduction

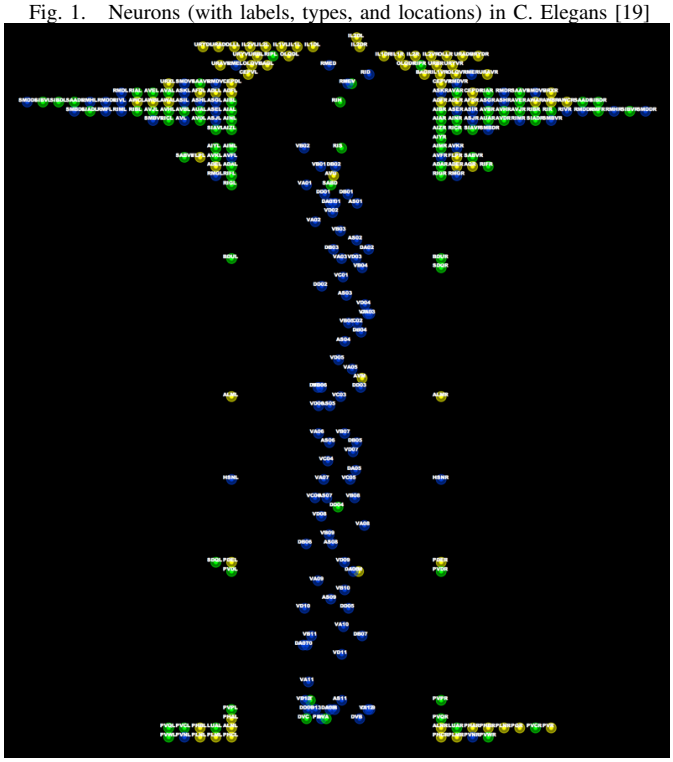
C. Elegans is a useful organism for studying how neural computation takes place, as its nervous system contains a limited number of neurons (302 in the hermaphrodite) whose connectivity is fully mapped and has been studied for many years. Furthermore, these neurons represent a variety of classes, 118 compared to only 5 in the mammalian cerebellum. These circuits include those for touch sensitivity, thermotaxis, chemotaxis, and smell [20]. Several sub-circuits of the overall network have been determined via observation and experimentation [19], [10], [2], [13], [7], [18], [8], [16], [13]. While there has been research, in particular computational modeling, into the workings of these sub-circuits in isolation [7], [16], little is known about how they interact with each other, how they perform in the larger neural environment, how they are selected originally, how they form in response to input, and how they might change in reaction to events in the worm's life. Additionally, much of the connectivity research into C. Elegans has been done on the basis of only a few individual organisms, so little is known about the variability of the worm's neural circuitry.

There is an ongoing interest in applying mathematical and computational models to neural computation in the hopes that new insights will emerge. Work such as [3] attempt to apply mathematical grounding (in particular probabilistic models) to the study of physical systems. Similarly, other forms of computational modeling seek to create abstract replicas of systems, incorporating as much empirical knowledge as possible. These *in silico* models can be evaluated by their predictive abilities, with outcomes verified by *in vivo* or *in vitro* observation. It is the hope that a good model will offer new insights into the workings of the system, due often to the transparency of the model - processes that are hidden in the real system become easily analyzed in a computational model.

Here I present a framework called C. Elegans Modeling System (CEMS) for applying common computational tools to some of the questions associated with C. Elegans' neural circuitry, and with neural computation in general. CEMS provides mechanisms for reading and writing neuronal data (names, positions, connectivity, etc.); a framework for representing an organism and its interactions with the environment; a framework for building, training, and testing arbitrary models of circuit formation and utilization; tools for analysis of various aspects of these models; and a visualization framework that supports visualization of empirical data as well as visualization of modeling and analysis.

As a test of this framework, and as a first step toward addressing research questions related to C. Elegans' circuitry, I build 3 different models of neural circuit formation of increasing complexity, and train and test these models thoroughly.

Fig. 1. Neurons (with labels, types, and locations) in C. Elegans [19]



II. Assumptions

I make several assumptions in undertaking the current work. It is important to note that these assumptions are not contradicted by current neurological knowledge, and serve to solidify the relationship of CEMS to empirical evidence as much as possible.

1. *The brain performs computation in a way that at some level resembles the computational notion of an artificial neural network.*

Empirical evidence supports the claim that computation in organisms is carried out in a manner resembling artificial neural networks [11]. There is a layer of input (or sensory) neurons, followed by one or more layers of hidden (or inter-) neurons, ending in a layer of output (motor) neurons. This models provides a non-linear mapping from input to output, with weights between connected nodes trained iteratively based on the quality of the network's output. The basic notation for such a network follows:

$$y_k(\mathbf{x},\mathbf{w}) = \phi(\sum_{j=1}^{M} w_{kj}^{(2)} \sigma(\sum_{i=1}^{D} w_{ji}^{(1)} x_i))$$

$$\sigma(a) = \frac{1}{1+e^{-a}}$$

It is likely that biological networks are not feed-forward, as most computational models are (and as the notation above implies). Neural network models that have been proposed include various types of Recurrent Neural Networks [4] such as LSTM [14], Hopfield Networks [15], and Deep Belief Networks [4].

This property of biological networks does not change the formulation of the problem presented here, as we simply wish to model the nodes that participate in such a network, not the functional properties of the network.

2. *Not all neurons are involved in each computation.*

It is generally accepted that for each discrete computation, some subset of available neurons play an active role. The neurons that take part in a given computational task collectively form the circuit associated with that task. It would be possible to model this sub-circuit property in a traditional neural network framework by setting the weights of the non-participating nodes to 0, however such a formulation becomes computationally intractable quickly and furthermore fails to model the variety of computational tasks undertaken by an organism even as simple as C. Elegans. Therefore, here I model only the *participation* of neurons in a discrete computation.

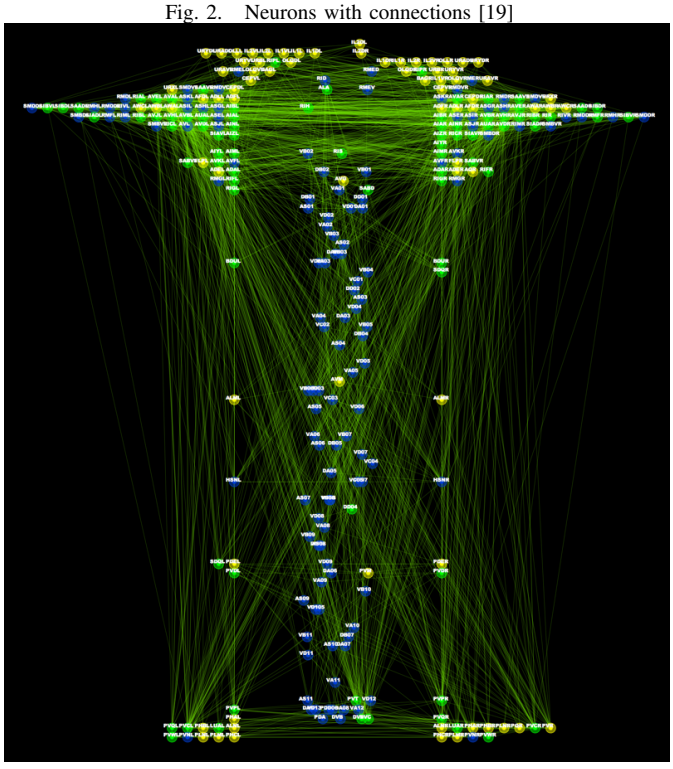
3. *Circuits tend to be partially invariant.*

Circuits are not purely random and are at least similar (if not identical) from computation to computation. They do, however, exhibit plasticity - when there is a change (such as loss of a critical neuron), the circuit re-forms and is again able to perform its computational task. These properties taken together imply the need for a dynamic system of circuit formation, rather than either a hardwired map of circuits, or a system whose properties are set once and then left alone.

4. *Given a sub-circuit of sufficient size, a set of weights exists to provide mapping from input to output.*

If we think of the neurons taking part in a given computation as a discrete neural network, we can assume that there is some set of weights between nodes that will map a given input to the desired output. The implication of this assumption is that we can model the functioning of the entire network by the formation of sub-circuits, and that if these circuits contain the appropriate input and output nodes, we will achieve the desired computation.

Fig. 2. Neurons with connections [19]



III. Implementation

CEMS was implemented entirely in Java and consists of approximately 5,000 lines of original code in 46 classes. Additionally, several open source libraries were utilized including Weka [12] for classifiers implementation, Apache Commons Math for various statistical operations, and SQLite for local database interaction. An outline of the CEMS architecture follows:

- World
Oversees creation of worms (either arbitrarily or by reproduction of existing worms), manages stimulus presentation to worms, oversees all statistics related to both individual worms and populations of worms, and manages system I/O.

– Worm
Accepts stimuli from world and manages activation of appropriate neurons in the brain (acts as bridge between world and nervous system), evaluates efficacy of a computation, and manages worm-specific data (i.e. overall health given worm's ability to move toward food).

— Brain
Manages per-computation circuit formation, oversees training of network, and provides a bridge from higher levels to neuron-level granularity.

—- Neural Graph
Consists of a set of Neural Nodes with connectivity defined by a weight matrix. This weight matrix can contain real-valued entries (i.e. in the case of the weighted graph formulation to be discussed later) or can be binary. In the case of a binary weight matrix, entries simply denote the *possibility* of a connection between neurons.

The Neural Graph also runs the per-computation circuit formation algorithm, and as such contains many of the parameters associated with the current model. Given start node(s), the Neural Graph generates a list of potential children based on empirical data given at system initialization, runs a model-dependent algorithm to determine which of these potential children should be added to the circuit, and then repeats with all children until the circuit is complete (completeness is reached when no new children are added at a given step).

—— Neural Node
Contains neuron-specific information such as name, class, and physical position in the worm's body. May or may not be involved in the determination of its own children in a given circuit, depending on the current model.

Basic program flow:

```
1. Generate worm(s)
2. At each time step:
   a. Present stimulus to worm(s)
   b. Provide feedback
   c. Log results for later analysis
3. Produce aggregate analysis
```

Detailed information flow:

```
1. Gather data from database
2. Generate list of neurons with associated data
        (label, class, type, position)
3. Generate binary connectivity matrix
4. Build worm
4a. Build neural graph
4b. Initialize model
5. Present stimulus to worm
6. Generate circuit
6a. Apply model parameters
7. Evaluate circuit
8. Pass evaluation to worm
8a. Generate training parameters (vector, weight)
9. Store model parameters
10. Repeat from (5)
11. Output statistics / visualizations
```

IV. Modeling

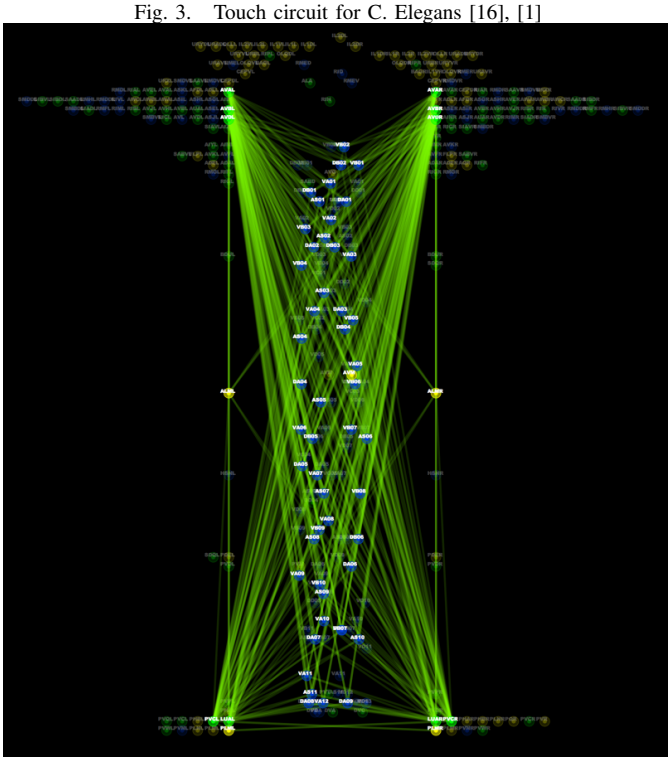
Computational modeling of complex biological systems is an active field, with a variety of ongoing research [9], [6]. Biological models generally fall into either the *integrationist* or *reductionist* categories. Integrationism seeks to explore a complex system by understanding the dynamic relationships between its components, with the workings of these parts essentially "abstracted away." A reductionist approach on the other hand seeks to understand the overall system via detailed modeling of its parts [17].

This work takes a decidedly integrationist approach. I seek to understand the principles at work in circuit formation and utilization without explicitly modeling the details of the components of the neural network. Understanding these components is the subject of much ongoing research, and as such is shaky ground on which to build the foundation of a computational model.

It is of utmost importance, however, that even the most high-level model not *contradict* scientific fact. Care must be taken in creating a model of this kind that we do not "play God" and impose rules and principles on the model that are implausible biologically. Strict adherence to this principle

ensures that a model from which results are derived can be refined and developed to finer levels of biological detail while maintaining the validity of those results. It is with this idea of biological plausibility in mind that many of the decisions in CEMS were made.

Fig. 3. Touch circuit for C. Elegans [16], [1]



A. Data

Several datasets of C. Elegans neural specifications and connectivity were evaluated. The data presented by [19] was deemed the most consistent and reliable, and so was utilized here. This data was made available electronically at [**?**] and was then converted into more efficient database structures, with no loss or modification of information. In addition to connectivity data, this dataset includes descriptions of neurons, classes, physical locations, and neuro-muscular junctions for all nonpharyngeal neurons except CANL and CANR, which have no synaptic connections. The total number of neurons included in the following models is therefore 280 (see Appendix IV. for a listing of neuron labels, types, classes, and longitudinal positions). Chemical synapses as well as electrical connections (gap junctions) were included and were given equal treatment in this work (since we look only at connectivity, no distinction need be made between the two types of synapse).

Data was verified by all available sources throughout the course of research, and in cases of incongruity, [19] took precedence. Additionally, [1] and [5] were referenced extensively as sources of information and intuition about the functioning of the network.

B. Research Questions

The following questions drove this work at a high level, providing meta-goals and focus.

1. How closely can computational tools replicate observed behavior in C. Elegans?
(Can we build a worm that passes the "Worm Turing Test?")

2. What can we learn about the biological mechanisms for computation by studying software that performs similar computation?

3. What can we learn about machine "intelligence" by studying how biological organisms compute?

C. Problem statement

Given the overall network of neurons, along with observed connectivity properties, design a model that learns functional sub-circuits, repeats those sub-circuits, and reacts appropriately when overall connectivity changes (i.e. ablation or damage).

Network definition:

$$N = \{n_0, n_1, ...n_k\}$$

Circuit definition:

$$C_t = \{\underline{n}_i, \underline{n}_j, \hat{\underline{n}}_k\};\ \underline{n}_i, \underline{n}_j, \hat{\underline{n}}_k \subseteq N$$

Optimization problem:

$$\alpha(\underline{n}_i; N) = \arg\max_{\underline{n}_j, \hat{\underline{n}}_k} C_t$$

Evaluation function:

$$E(C_t) = \theta_1(\hat{\underline{n}}_k) * \theta_2(|C_t|) * \theta_3(\hat{C}_t)$$

$$\theta_1(\hat{\underline{n}}_k) = \textit{reward}$$
$$\theta_2(|C_t|) = \textit{circuit size penalty}$$
$$\theta_3(\hat{C}_t) = \textit{circuit efficiency}$$

The formulation above is sensitive to the choice of evaluation function, therefore it is worth looking into this function in more detail. Following is a plot that shows the various terms $\theta_1, \theta_2, \theta_3$ as well as $E(C_t)$ plotted as a function of the total number of neurons present. The $E(C_t)$ plot implies that overall neuron count ($|C_t|$) increases linearly with network complexity ($\hat{C}_t$) and circuit efficacy ($\theta_1(\hat{\underline{n}}_k)$, which is measured as:

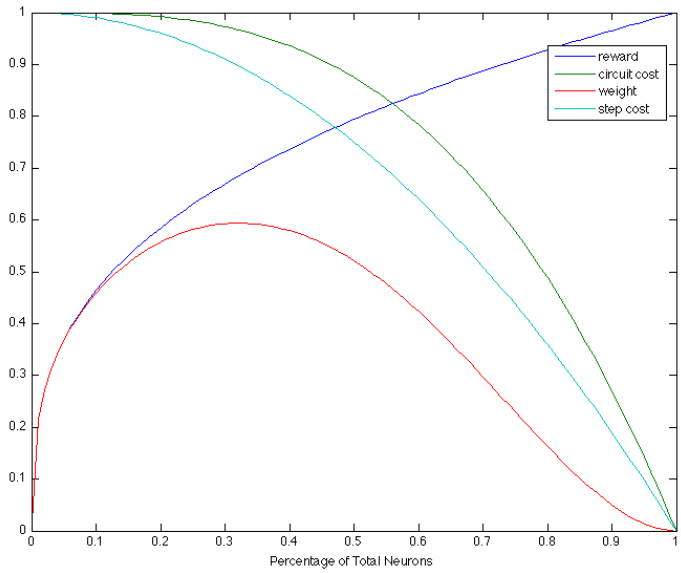
$$\frac{\text{relevant motor neurons in } C_t}{\text{relevant motor neurons in } N}$$

where "relevant" is dependent on the type of stimulus. For example, for the training undertaken in this work, I hope to

produce movement in the worm, which would result from the presence in C_t of neurons with appropriate neuro-muscular connections (50 in the case of C. Elegans).

Notice also in the following plot that $E(C_t)$ is maximized at approximately .25 of total neurons, which corresponds roughly to the number of neurons present in the actual circuit for touch sensitivity, providing empirical validation for the choice of evaluation function.

Fig. 4. Evaluation function



D. Training Procedure

In the following models, training and testing proceeded as follows:

5 worms were initialized with random parameters. Each worm was presented with 1 stimulus per time-step, with the choice of stimulus determined by the type of training undertaken (in some cases this choice was fixed, in others random). This proceeded for 500 time-steps, at which time the best worm was selected to reproduce. 5 new worms were created using parameters from the parent, with mutation (gaussian noise) added. After 500 more time-steps, the most fit worm was evaluated as seen in the following sections.

E. Random circuits

In this rudimentary model, we build circuits as follows:
For each node, for each potential child, flip a (fair) coin to determine connectivity.

For the C. Elegans network, there are 4,577 potential connections, which given the formulation above gives $E[|C_t|] = 2,288.5$ active connections for each circuit. Since a fully connected graph on n nodes contains $\frac{n(n-1)}{2}$ edges, we can determine the minimum number of neurons a random circuit is expected to contain:

$$2288.5 = \frac{n(n-1)}{2}$$
$$n^2 - n = 5477$$
$$n^2 > 5477$$
$$n > \sqrt{5477}$$
$$n > 74$$

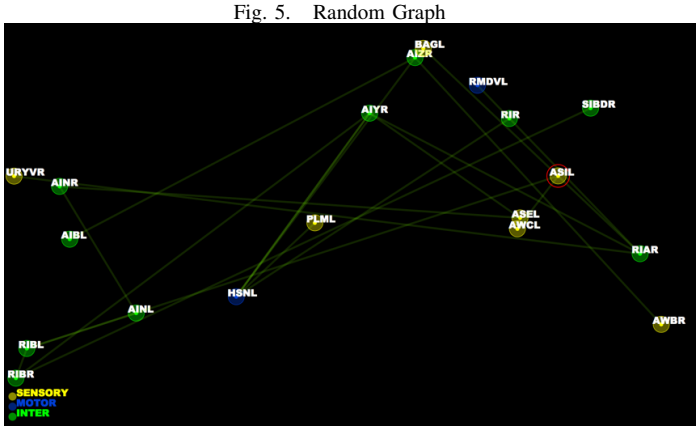
The maximum expected number of neurons is equal to the total available neurons, or 280. Since 74 is larger than the smallest known sub-circuit, we will need to apply a decay function to the circuit as it is formed to limit the total number of nodes in the network.

$$P(n_i \rightarrow n_j) = \frac{1}{2} - \delta(|C_t|)$$

Where $\delta(.)$ = decay function based on number of nodes already added to C_t.

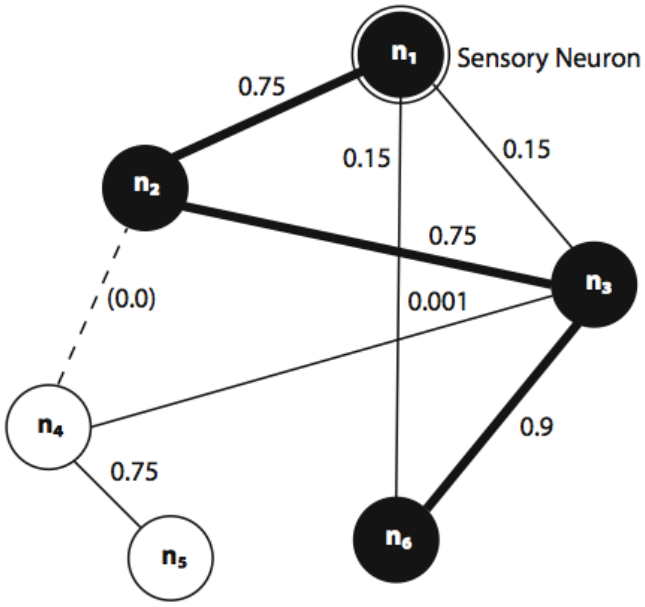
While this method produced some functional circuits, there were (as predicted) many problems. Most importantly, circuits were not repeatable - if a circuit proved beneficial there was no way to "save" it to use it again for a future computation. Furthermore, there were few such beneficial circuits formed. Below is an example of a circuit formed with this method.

Fig. 5. Random Graph



F. Weighted graph

In this model, circuits are formed as in the random graph above, except that the transition probabilities are trained iteratively based on evaluation of the previous circuit. Transition probabilities are initialized to some low value, with transitions associated with a successful circuit raised accordingly. As before, a decay function was required in order to limit circuit size. Below is a representation of the weighted graph structure.



The entries in the adjacency matrix A are given by:

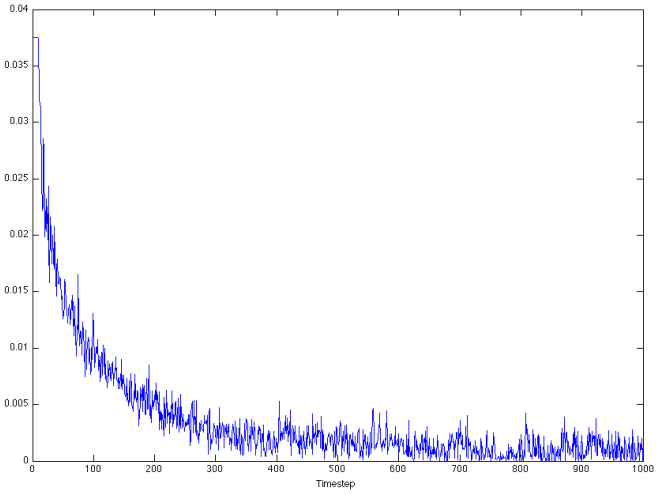
$$A : a_{ij} = P(n_i \rightarrow n_j)$$

Therefore, each potential connection c_t is evaluated according to:

$$P(c_t) = a_{ij} - \delta(|C_t|)$$

This method appeared to produce good results. Sample circuits evaluated during training appeared valid, and RMSE of weight matrices A_t, A_{t-1} at each time step t decreased, showing that the model was converging. Upon closer examination, however, I found that the model was converging toward $a_{ij} = 1 \forall i, j$. The decay function was essentially dictating formation of circuits, as all probabilities neared 1.

Fig. 6. Weighted Graph Training



G. Classifier-based Circuits

In this formulation, circuits are formed as follows. For each node, we generate a binary feature vector denoting the incoming connections to that node. This vector then has 280 entries, with all set to 0 except those that represent nodes connecting to the current node. We therefore have a feature vector representing the current node's parents. We use this vector with a set of Naive-Bayes classifiers to determine which of this node's potential children to connect to. The set of classifiers (4,577 required for C. Elegans) are trained based on evaluation of the resulting circuit. If the circuit gains positive evaluation, the classifiers are given the current circuit as a positive example. Likewise, if the circuit receives negative evaluation, classifiers are trained to suppress the current circuit by setting all transitions to 0 and presenting the ensuing non-circuit as a positive example.

Our adjacency matrix A is now a binary matrix denoting only potential connections (not weights):

$$A : \ a_{ij} \begin{cases} 1 & : P(n_i \rightarrow n_j) > 0 \\ 0 & : otherwise \end{cases}$$

The set of classifiers is described by:

$$\Phi_i(\underline{n}_i; \{\hat{\phi}_{ij}\}) = \underline{a}_i$$

$$\hat{\phi}_{ij}(\underline{n}_i; \theta, N(\mu; \sigma^2) \pm \epsilon) = a_{ij}$$

Note that gaussian noise is added to the base classification, along with a term ϵ that causes uncertain classifications to favor negative (unconnected) response. Each classifier $\hat{\phi}_{ij}$ outputs a class probability vector:

$$\underline{y}_{ij} = [P(a_{ij} = 0); P(a_{ij} = 1)].$$

This vector is then transformed as follows:

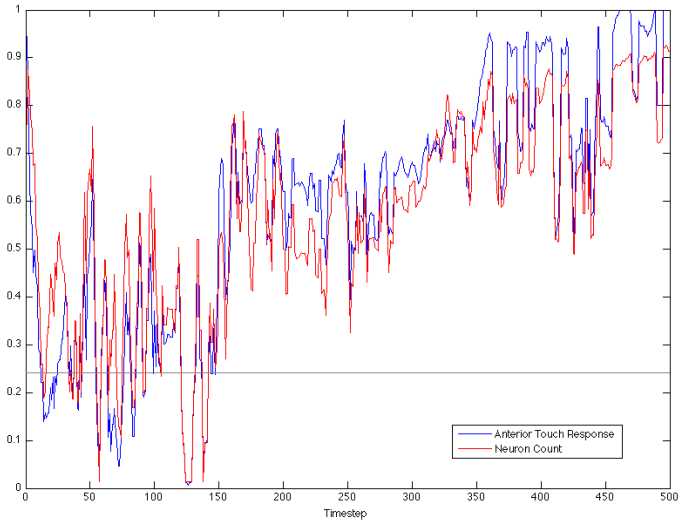
$$\underline{y}_{ij} = [P(a_{ij} = 0) + N(\mu; \sigma^2) + \epsilon; P(a_{ij} = 1) + N(\mu; \sigma^2) - \epsilon]$$

With final classification given by:

$$\hat{a}_{ij} = \arg\max \underline{y}_{ij}$$

This model was trained first by presenting only anterior touch stimulus. This corresponds to starting circuits with neurons ALML, ALMR, and AVM. Results of this training can be seen below:

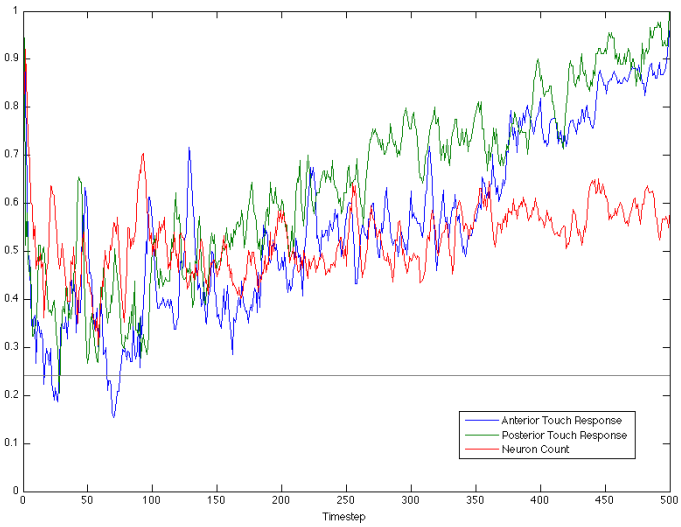
Fig. 7. Anterior Touch Training



This can be seen as unsuccessful training, as the efficacy of the circuit (measured by the ratio of relevant motor neurons in the circuit to total relevant motor neurons, depicted in blue) goes up only as total neuron count (depicted in red) goes up. The neuron count of the actual C. Elegans touch circuit (which contains all relevant motor neurons) is shown as a grey line here, at approximately .25 of total neurons.

Another training run, this time with stimulus alternating randomly between anterior and posterior touch, is shown below.

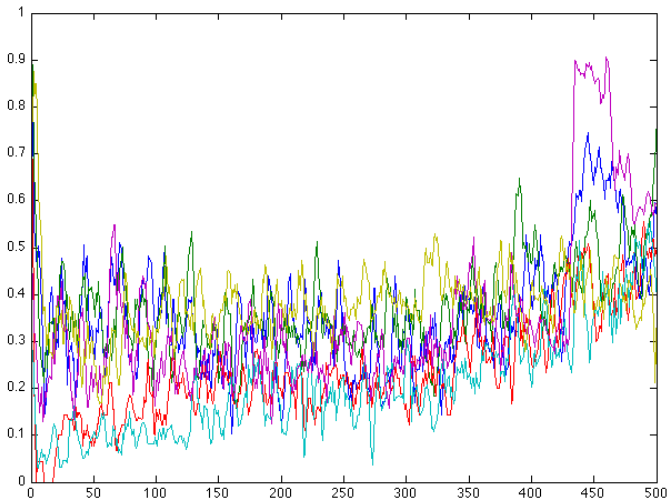
Fig. 8. Anterior/Posterior Touch Training



This is a more successful method of training, as circuit efficacy continues to rise for both anterior and posterior touch even after neuron count remains relatively stable. The circuits generated here are sub-optimal as compared to either the real touch circuit or to a graph-theoretical measure of optimality, but this method appears promising nonetheless.

The final round of training consisted of attempting to train all circuits associated with movement (anterior touch, posterior touch, olfactory, chemotaxis, and thermotaxis) simultaneously. Results are shown below.

Fig. 9. All Circuit Training



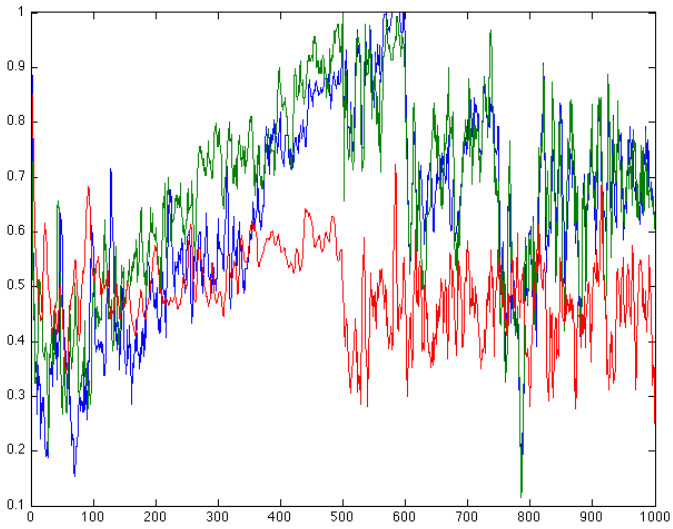
As can be seen in the figure, this model did not train well. Neuron count stayed low, but motor response never improved consistently. It is worth noting, however, that for a brief period near the end of the training cycle, response to heat and chemical stimuli (shown in magenta and blue respectively) rose significantly while neuron count stayed low. This implies that perhaps more training was necessary; the model may have been caught in a local maxima and simply needed to ”find” a way out via random trial and error.

H. Plasticity

As a measure of the plasticity of the model, the following procedure was undertaken:

The best worm was selected from the Touch Training procedure above (where anterior and posterior touch were trained together). 25 neurons were then deleted at random from the network, and training was allowed to progress for 250 time-steps. Then 25 more neurons were deleted, with training progressing for another 250 steps. Results are shown below.

Fig. 10. Plasticity Testing



As can be seen, after each set of neurons was deleted there was a period where motor response dropped. Response recovered, but never to the pre-deletion level. Overall, with 50 less neurons available, circuit sizes became smaller and, while motor response was hurt, it remained proportionately greater than circuit size. This can be seen as the model adapting to deletion of neurons, or successfully demonstrating plasticity.

V. Conclusions / Future Work

Software:
CEMS represents a robust, full featured test bed for circuit-level modeling of C. Elegans, as illustrated by the ease with which 3 very different models were implemented and tested in the CEMS framework. The code hierarchy mimics biological structure, with the varying layers of abstraction correlating closely with intuitive levels of detail in an actual organism. Integration with existing data sources and open-standard formats is robust, and allowances have been made for the future addition of finer detailed modeling. In particular, data structures are in place to allow a circuit, once formed, to be treated as an autonomous neural network consisting of the neurons in the circuit. As this is a generic data structure in CEMS, the neural network could be modeled in any number of different ways, including using an accepted neural modeling framework such as NEURON.

While optimization was not a focus of this work, some level of optimization was crucial in order to be able to use the system. Therefore, I present the following runtime statistics for the more complex classifier-based model:

Mean per-time-step time (seconds): 13
Standard deviation (seconds): 12.324
Mean per-worm time (seconds, 500 time-steps): 6,218 (103 minutes, 38 seconds)

The large variance in execution time is due to the variety of circuit sizes produced by the model. Smaller circuits take significantly less time to build, evaluate, and train than larger ones.

Modeling:
In some ways, the modeling work undertaken here was unsuccessful. In particular, I failed to find a viable model for circuit formation and utilization. It is possible, however, that the models proposed (in particular the classifier-based model) is very close to being viable. That model represents a biologically plausible per-neuron model of connectivity that results in circuits that would yield behavior reminiscent of actual worms, even if sub-optimally.

There are several extremely sensitive parameters present in this model that were not fully explored. The choice of per-worm optimization algorithm might be explored further with positive results. For example, the larval stage of C. Elegans might provide an environment for circuit training that is conducive to producing the types of circuits we hope to find. Or, perhaps training all touch circuits simultaneously was in fact the correct approach, given either more time or better tuned parameters. It is also possible that adding even more circuits to the training cycle (circuits for mating, electrosensory, others) would yield better results.

Possibly more importantly, the environment of the worm and its lifespan should be modeled more accurately. It is possible that the order or frequency with which stimuli are presented can account for large differences in how circuits are formed. Reproduction (and therefore mutation) are crucial to evolving ”good” biological structures, and these facets were modeled only very crudely here. The volume of worms generated and evaluated should also more closely mimic actual worms - here I work in generations of 5 worms, while in nature C. Elegans produces 300 offspring [20]. Finally, the male worm (which has slightly different neural topology) has not been studied here, and could provide additional insights or input to the model that could be beneficial.

To summarize, the software presented here provides an excellent framework in which to study C. Elegans neural circuitry. The two most immediate areas for development are per-worm model optimization techniques, and addition of more world-level detail (including reproduction, and more varied stimulus). As secondary areas for development, I would highlight increasing biological detail at the neuron level as the most important as this would aid in validating the approach taken here as well as the particular models utilized.

APPENDIX I
MAS.881 REQUIRED PROJECT POINTS

A. The need(s) (either clinical or basic-science) that it will fulfill

CEMS fills a void in the computational neuroscience community for a readily accessible, Java-based framework for modeling and exploration of properties of biological neural network topology. It also is, to my knowledge, the only such software focused exclusively on C. Elegans.

The models discussed here represent a small step toward understanding some basic principles of neural computation - namely how smaller sub-circuits, each able to be seen as an autonomous neural network, might work together to perform the variety of computational tasks necessary for even the simplest organisms to survive. The learning of such circuits, as well as the utilization and properties such as plasticity have not been studied in a computationally grounded way to date, and this work serves to fulfill that need.

Thus, this work fulfills basic-science needs in neuroscience, as well as computer science.

B. The specifications and designed properties of the neurotechnology

See section III. Implementation above.

C. The plan for testing it and assessing whether the specifications are met

Various testing was performed in order to ascertain the system's performance. In particular, a stated goal was to produce simulated C. Elegans nervous system configurations that matched empirical data. See section IV. Modeling above for full details of the testing protocol.

D. The funding plan to develop it (government grants, VC money, etc.)

No additional funding is required at this time.

E. The plan for dealing with any risk or obstacles encountered along the way

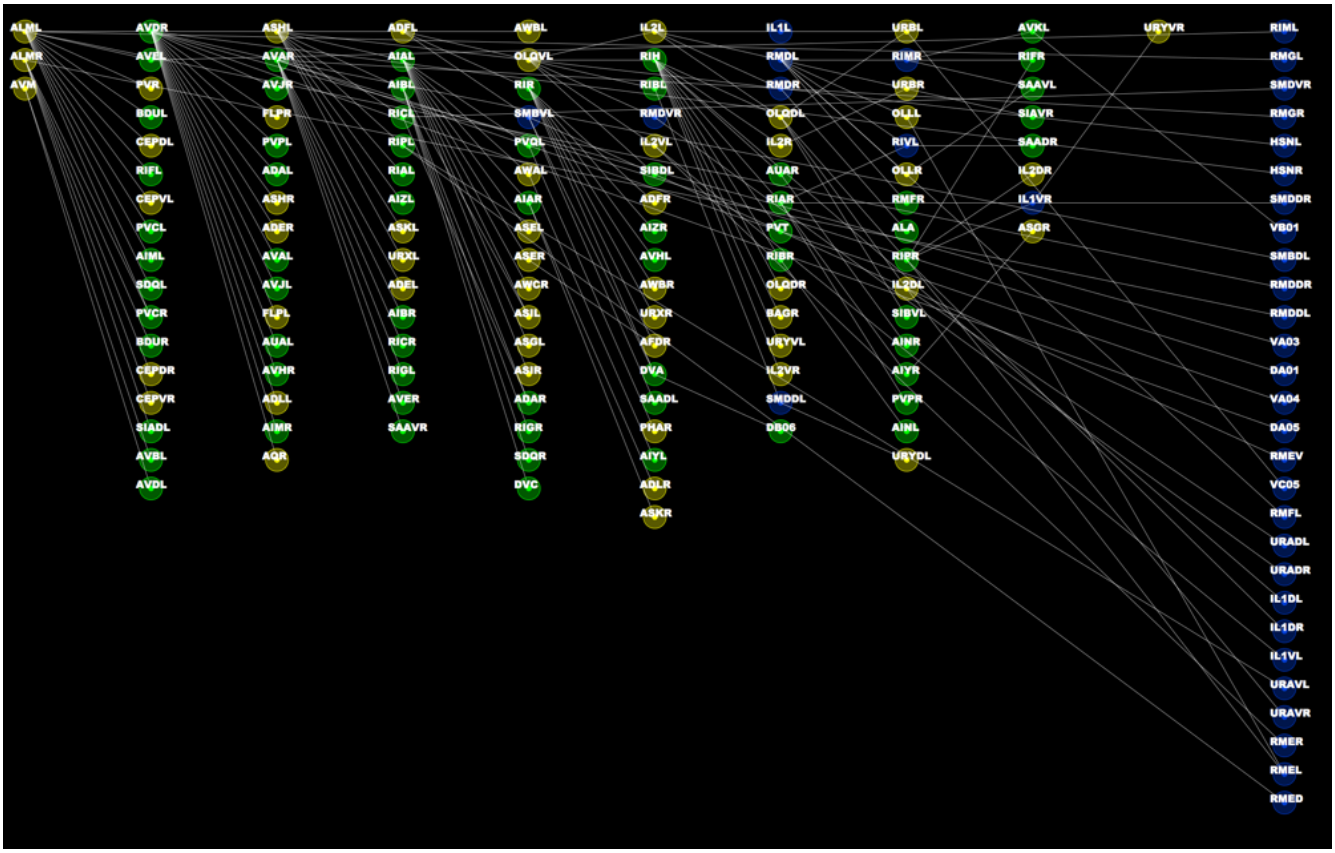
Risk is not relevant here. Obstacles were encountered in the course of development, and as development continues more obstacles are to be expected. This is the nature of software development, and many well documented strategies exist.

Obstacles that are particular to this system mainly involve the system's integration of empirical data. Should this data change, or new data be added, CEMS will need to incorporate these changes. Every effort has been made to make the software flexible and extensible enough to handle these situations should they arise.

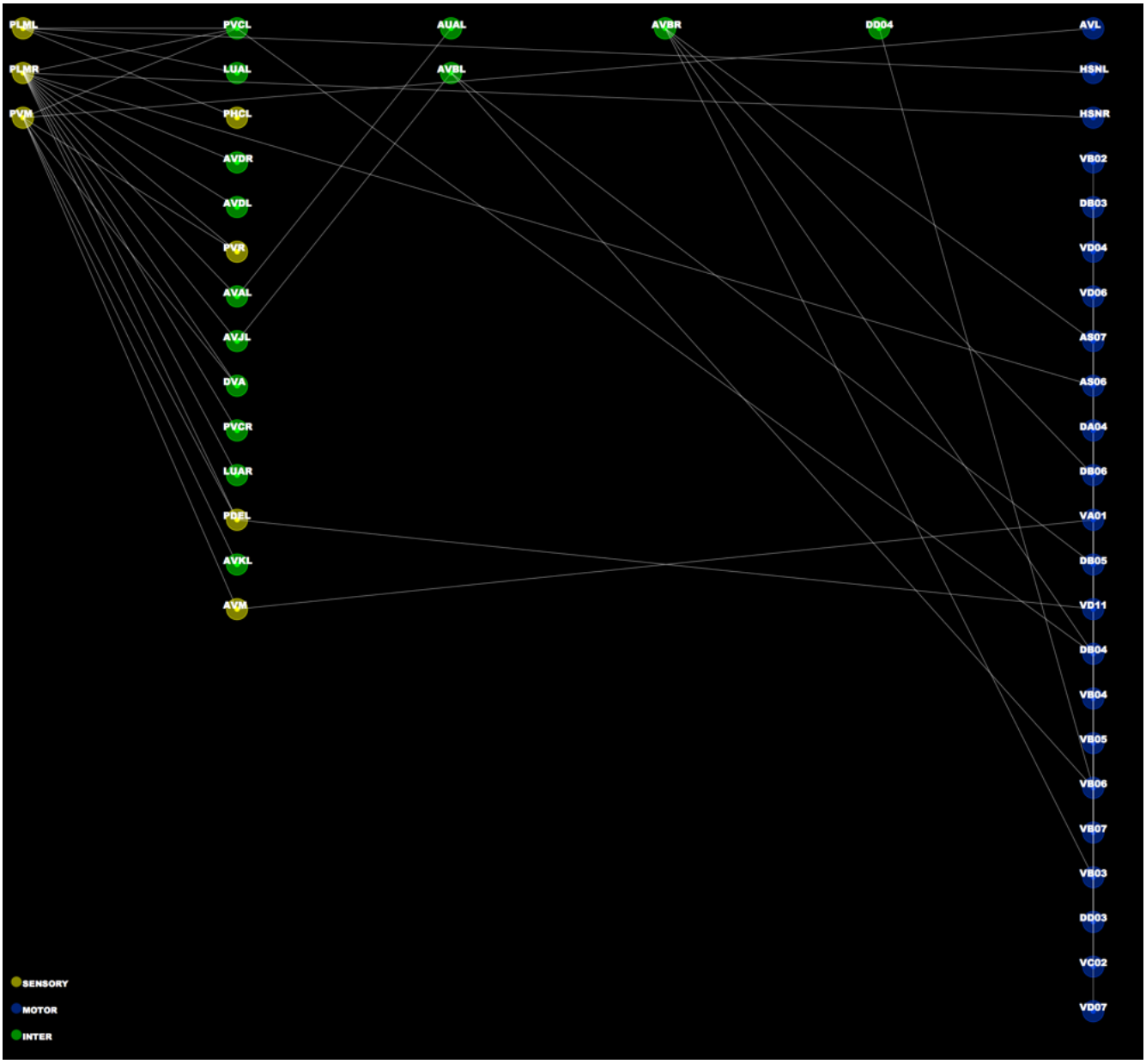
F. The timecourse of development and testing

Further development is expected to take place over the next 4-6 months. At that time, progress will be evaluated and continued work on the project will be re-assessed.

APPENDIX II
SAMPLE CIRCUIT (WEIGHTED GRAPH)



APPENDIX III
SAMPLE CIRCUIT (CLASSIFIER-BASED)



Appendix IV
Neuron Data [19]

ADAR (ADA): INTER - Position: 0.21
ADFL (ADF): SENSORY - Position: 0.13
ASHL (ASH): MOTOR - Position: 0.14
AVDR (AVD): INTER - Position: 0.16
PVQL (PVQ): INTER - Position: 0.82
ADEL (ADE): SENSORY - Position: 0.21
AIAL (AIA): INTER - Position: 0.15
AIBL (AIB): INTER - Position: 0.14
AIBR (AIB): INTER - Position: 0.15
AVAR (AVA): INTER - Position: 0.13
AVBL (AVB): INTER - Position: 0.15
AVBR (AVB): INTER - Position: 0.15
AVDL (AVD): INTER - Position: 0.16
AVEL (AVE): INTER - Position: 0.13
AVJR (AVJ): INTER - Position: 0.15
AWAL (AWA): SENSORY - Position: 0.14
FLPR (FLP): SENSORY - Position: 0.2
PVPL (PVP): INTER - Position: 0.8
PVR (PVR): SENSORY - Position: 0.82
RICL (RIC): INTER - Position: 0.16
RICR (RIC): INTER - Position: 0.17
RIML (RIM): MOTOR - Position: 0.15
RIPL (RIP): INTER - Position: 0.09
RMGL (RMG): MOTOR - Position: 0.22
SMDVR (SMD): MOTOR - Position: 0.13
ADAL (ADA): INTER - Position: 0.21
ADFR (ADF): SENSORY - Position: 0.14
ASHR (ASH): MOTOR - Position: 0.14
PVQR (PVQ): INTER - Position: 0.81
ADER (ADE): SENSORY - Position: 0.21
AIAR (AIA): INTER - Position: 0.16
AVAL (AVA): INTER - Position: 0.13
AVJL (AVJ): INTER - Position: 0.15
RIMR (RIM): MOTOR - Position: 0.16
RIPR (RIP): INTER - Position: 0.1
RIVR (RIV): MOTOR - Position: 0.15
RMGR (RMG): MOTOR - Position: 0.22
SMDVL (SMD): MOTOR - Position: 0.12
URBR (URB): SENSORY - Position: 0.1
AINL (AIN): INTER - Position: 0.16
AVKR (AVK): MOTOR - Position: 0.19
ALA (ALA): INTER - Position: 0.12
AVL (AVL): MOTOR - Position: 0.16
BDUL (BDU): INTER - Position: 0.31
CEPDL (CEP): SENSORY - Position: 0.12
FLPL (FLP): SENSORY - Position: 0.2
IL1L (IL01): MOTOR_SENSORY - Position: 0.08
IL2L (IL02): SENSORY - Position: 0.08
OLLL (OLL): SENSORY - Position: 0.08
RIAL (RIA): INTER - Position: 0.13
RIFL (RIF): INTER - Position: 0.22
RIGL (RIG): INTER - Position: 0.23
RIGR (RIG): INTER - Position: 0.22
RIH (RIH): INTER - Position: 0.14
RIVL (RIV): MOTOR - Position: 0.14
RMDL (RMD): MOTOR - Position: 0.13
RMHL (RMH): MOTOR - Position: 0.14
SIADR (SIA): INTER - Position: 0.16
SIBDR (SIB): INTER - Position: 0.14
SMBDR (SMB): MOTOR - Position: 0.17
URBL (URB): SENSORY - Position: 0.09
AVKL (AVK): INTER - Position: 0.2
ALNR (ALN): SENSORY - Position: 0.82
AVER (AVE): INTER - Position: 0.14
AVM (AVM): SENSORY - Position: 0.44
BDUR (BDU): INTER - Position: 0.31
CEPDR (CEP): SENSORY - Position: 0.13
IL2R (IL02): SENSORY - Position: 0.09
OLLR (OLL): SENSORY - Position: 0.09
RMDR (RMD): MOTOR - Position: 0.13
SAAVR (SAA): INTER - Position: 0.13
AIZL (AIZ): INTER - Position: 0.17
AUAL (AUA): INTER - Position: 0.15
AWBL (AWB): SENSORY - Position: 0.14
OLQVL (OLQ): SENSORY - Position: 0.1
RIR (RIR): INTER - Position: 0.15
SMBVL (SMB): MOTOR - Position: 0.16
AIYR (AIY): INTER - Position: 0.18
AIZR (AIZ): INTER - Position: 0.17
ASEL (ASE): MOTOR - Position: 0.15
AUAR (AUA): INTER - Position: 0.16
AVHL (AVH): INTER - Position: 0.15
AWAR (AWA): SENSORY - Position: 0.14
AWBR (AWB): SENSORY - Position: 0.14
PVPR (PVP): INTER - Position: 0.79
RIAR (RIA): INTER - Position: 0.13
SMBVR (SMB): MOTOR - Position: 0.16
URXR (URX): SENSORY - Position: 0.13
ADLR (ADL): SENSORY - Position: 0.14
ASER (ASE): MOTOR - Position: 0.15
AVHR (AVH): INTER - Position: 0.15
CEPVL (CEP): SENSORY - Position: 0.11
SDQR (SDQ): INTER - Position: 0.32
ADLL (ADL): SENSORY - Position: 0.13
AWCR (AWC): SENSORY - Position: 0.14
PVCL (PVC): INTER - Position: 0.82
AFDR (AFD): SENSORY - Position: 0.14
AINR (AIN): INTER - Position: 0.16
AIYL (AIY): INTER - Position: 0.19
AFDL (AFD): SENSORY - Position: 0.13
ASIL (ASI): MOTOR - Position: 0.14
AIML (AIM): INTER - Position: 0.19
ASGL (ASG): MOTOR - Position: 0.14
ASIR (ASI): MOTOR - Position: 0.15
ASKL (ASK): MOTOR - Position: 0.13
AWCL (AWC): SENSORY - Position: 0.14
HSNL (HSN): MOTOR - Position: 0.55
AIMR (AIM): INTER - Position: 0.19
ASGR (ASG): MOTOR - Position: 0.14
ASKR (ASK): MOTOR - Position: 0.13
RIFR (RIF): INTER - Position: 0.21
DVC (DVC): INTER - Position: 0.81
PVT (PVT): INTER - Position: 0.79
BAGL (BAG): SENSORY - Position: 0.1
HSNR (HSN): MOTOR - Position: 0.55
RIBR (RIB): INTER - Position: 0.15
SAADL (SAA): INTER - Position: 0.14
SAADR (SAA): INTER - Position: 0.14
SAAVL (SAA): INTER - Position: 0.12
SMDDR (SMD): MOTOR - Position: 0.15
DB01 (DB): MOTOR - Position: 0.24
RIS (RIS): INTER - Position: 0.19
RIBL (RIB): INTER - Position: 0.15
SDQL (SDQ): INTER - Position: 0.64
SMDDL (SMD): MOTOR - Position: 0.14
VB01 (VB): MOTOR - Position: 0.21
ALML (ALM): SENSORY - Position: 0.46
AVFL (AVF): MOTOR - Position: 0.2
AVFR (AVF): MOTOR - Position: 0.2
ASJR (ASJ): MOTOR - Position: 0.16
OLQDR (OLQ): SENSORY - Position: 0.1
PVNR (PVN): MOTOR - Position: 0.83
BAGR (BAG): SENSORY - Position: 0.11
RID (RID): MOTOR - Position: 0.11
DVA (DVA): INTER - Position: 0.81
SMBDL (SMB): MOTOR - Position: 0.15
VB02 (VB): MOTOR - Position: 0.19
PVCR (PVC): INTER - Position: 0.82
RMDDR (RMD): MOTOR - Position: 0.15
CEPVR (CEP): SENSORY - Position: 0.12
RMDDL (RMD): MOTOR - Position: 0.14
SIADL (SIA): INTER - Position: 0.15
RMHR (RMH): MOTOR - Position: 0.15
URXL (URX): SENSORY - Position: 0.12
VA03 (VA): MOTOR - Position: 0.31
VD02 (VD): MOTOR - Position: 0.26
DA01 (DA): MOTOR - Position: 0.25
VD01 (VD): MOTOR - Position: 0.25
DA02 (DA): MOTOR - Position: 0.3
VA04 (VA): MOTOR - Position: 0.37
DD01 (DD): MOTOR - Position: 0.24
VA05 (VA): MOTOR - Position: 0.43
DA03 (DA): MOTOR - Position: 0.37
VD03 (VD): MOTOR - Position: 0.31
AS05 (AS): MOTOR - Position: 0.47
DB03 (DB): MOTOR - Position: 0.3
VD04 (VD): MOTOR - Position: 0.36
AS04 (AS): MOTOR - Position: 0.4
VA07 (VA): MOTOR - Position: 0.55
DD02 (DD): MOTOR - Position: 0.34
VD05 (VD): MOTOR - Position: 0.42
DA05 (DA): MOTOR - Position: 0.54
VA08 (VA): MOTOR - Position: 0.6

PLMR (PLM): SENSORY - Position: 0.83
VD06 (VD): MOTOR - Position: 0.47
DVB (DVB): MOTOR - Position: 0.81
PDB (PDB): MOTOR - Position: 0.8
PDA (PDA): MOTOR - Position: 0.81
VA12 (VA): MOTOR - Position: 0.8
VD13 (VD): MOTOR - Position: 0.8
ASJL (ASJ): MOTOR - Position: 0.16
IL2DL (IL02): SENSORY - Position: 0.07
AQR (AQR): SENSORY - Position: 0.21
AS02 (AS): MOTOR - Position: 0.29
AS03 (AS): MOTOR - Position: 0.35
AS07 (AS): MOTOR - Position: 0.57
AS08 (AS): MOTOR - Position: 0.62
AS09 (AS): MOTOR - Position: 0.68
AS10 (AS): MOTOR - Position: 0.73
AS11 (AS): MOTOR - Position: 0.79
AS01 (AS): MOTOR - Position: 0.25
AS06 (AS): MOTOR - Position: 0.51
DA04 (DA): MOTOR - Position: 0.45
DA06 (DA): MOTOR - Position: 0.65
DA07 (DA): MOTOR - Position: 0.73
DB06 (DB): MOTOR - Position: 0.62
LUAL (LUA): INTER - Position: 0.82
SABD (SAB): INTER - Position: 0.23
SABVR (SAB): INTER - Position: 0.2
URYDL (URY): SENSORY - Position: 0.08
URYVR (URY): SENSORY - Position: 0.1
VA01 (VA): MOTOR - Position: 0.23
VA02 (VA): MOTOR - Position: 0.27
VA06 (VA): MOTOR - Position: 0.5
VA09 (VA): MOTOR - Position: 0.66
VA10 (VA): MOTOR - Position: 0.71
VA11 (VA): MOTOR - Position: 0.77
VB09 (VB): MOTOR - Position: 0.61
DA08 (DA): MOTOR - Position: 0.8
DA09 (DA): MOTOR - Position: 0.8
DB05 (DB): MOTOR - Position: 0.51
LUAR (LUA): INTER - Position: 0.82
PHBL (PHB): SENSORY - Position: 0.82
PHBR (PHB): SENSORY - Position: 0.82
PHCL (PHC): SENSORY - Position: 0.83
PQR (PQR): SENSORY - Position: 0.82
PVDL (PVD): INTER - Position: 0.65
PVDR (PVD): INTER - Position: 0.65
PVNL (PVN): MOTOR - Position: 0.83
SABVL (SAB): INTER - Position: 0.2
URYDR (URY): SENSORY - Position: 0.09
URYVL (URY): SENSORY - Position: 0.09
AVG (AVG): SENSORY - Position: 0.22
PDEL (PDE): SENSORY - Position: 0.64
PDER (PDE): SENSORY - Position: 0.64
PVWR (PVW): INTER - Position: 0.83
VD11 (VD): MOTOR - Position: 0.74
DB04 (DB): MOTOR - Position: 0.39
DB07 (DB): MOTOR - Position: 0.72
SIBVL (SIB): INTER - Position: 0.14
VB04 (VB): MOTOR - Position: 0.32
VB05 (VB): MOTOR - Position: 0.38
VB06 (VB): MOTOR - Position: 0.45
VB07 (VB): MOTOR - Position: 0.5
VB08 (VB): MOTOR - Position: 0.57
VB10 (VB): MOTOR - Position: 0.67
VB11 (VB): MOTOR - Position: 0.72
VC03 (VC): MOTOR - Position: 0.46
VC04 (VC): MOTOR - Position: 0.53
DB02 (DB): MOTOR - Position: 0.21
VB03 (VB): MOTOR - Position: 0.28
VD10 (VD): MOTOR - Position: 0.69
PHAL (PHA): SENSORY - Position: 0.81
RMDVR (RMD): MOTOR - Position: 0.12
RMEV (RME): MOTOR - Position: 0.12
IL1R (IL01): MOTOR_SENSORY - Position: 0.09
RMDVL (RMD): MOTOR - Position: 0.12
VC05 (VC): MOTOR - Position: 0.55
PHAR (PHA): SENSORY - Position: 0.82
PHCR (PHC): SENSORY - Position: 0.83
PVWL (PVW): INTER - Position: 0.83
PVM (PVM): SENSORY - Position: 0.65
RMFL (RMF): MOTOR - Position: 0.15
RMFR (RMF): INTER - Position: 0.15
SIAVR (SIA): INTER - Position: 0.17
DD06 (DD): MOTOR - Position: 0.8
VC01 (VC): MOTOR - Position: 0.33
VD12 (VD): MOTOR - Position: 0.79
ALMR (ALM): SENSORY - Position: 0.46
IL2VL (IL02): SENSORY - Position: 0.08
URADL (URA): MOTOR_SENSORY - Position: 0.08
URADR (URA): MOTOR_SENSORY - Position: 0.09
OLQDL (OLQ): SENSORY - Position: 0.09
IL1DL (IL01): MOTOR_SENSORY - Position: 0.08
IL1DR (IL01): MOTOR_SENSORY - Position: 0.09
IL2DR (IL02): SENSORY - Position: 0.08
IL1VL (IL01): MOTOR_SENSORY - Position: 0.08
SIAVL (SIA): INTER - Position: 0.17
URAVL (URA): MOTOR_SENSORY - Position: 0.1
OLQVR (OLQ): SENSORY - Position: 0.11
IL1VR (IL01): MOTOR_SENSORY - Position: 0.11
IL2VR (IL02): SENSORY - Position: 0.09
URAVR (URA): MOTOR_SENSORY - Position: 0.11
DD03 (DD): MOTOR - Position: 0.45
VC02 (VC): MOTOR - Position: 0.38
DD04 (DD): INTER - Position: 0.58
VD08 (VD): MOTOR - Position: 0.59
VD09 (VD): MOTOR - Position: 0.64
PLML (PLM): SENSORY - Position: 0.83
RMER (RME): MOTOR - Position: 0.11
RMEL (RME): MOTOR - Position: 0.1
RMED (RME): MOTOR - Position: 0.1
SIBVR (SIB): INTER - Position: 0.15
SIBDL (SIB): INTER - Position: 0.14
DD05 (DD): MOTOR - Position: 0.69
VD07 (VD): MOTOR - Position: 0.52
ALNL (ALN): SENSORY - Position: 0.82
PLNL (PLN): MOTOR_SENSORY - Position: 0.83
PLNR (PLN): SENSORY - Position: 0.82
VC06 (VC): MOTOR - Position: 0.57

REFERENCES

[1] The national center for biomedical ontology, c. elegans gross anatomy. http://bioportal.bioontology.org/visualize/40924/, 2009.

[2] P Alfonso. Optimally wired subnetwork determines neuroanatomy of caenorhabditis elegans. *Proceedings of the National Academy of Sciences of the United States of America*, 104(43):17180, 2007.

[3] JM Beck, WJ Ma, R Kiani, T Hanks, AK Churchland, J Roitman, MN Shadlen, PE Latham, and A Pouget. Probabilistic population codes for bayesian decision making. *Neuron*, 60(6):1142–1152, 2008.

[4] Y Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning, to appear*, 2009.

[5] Nikhil Bhatla. Worm web: C. elegans interactive neural network. http://wormweb.org/neuralnet, 2009.

[6] L Bortolussi and A Policriti. Modeling biological systems in stochastic concurrent constraint programming. *Constraints*, 13(1):66–90, 2008.

[7] A Cangelosi and D Parisi. A neural network model of caenorhabditis elegans: the circuit of touch sensitivity. *Neural processing letters*, 6(3):91–98, 1997.

[8] M Chalfie, JE Sulston, JG White, E Southgate, JN Thomson, and S Brenner. The neural circuit for touch sensitivity in caenorhabditis elegans. *Journal of Neuroscience*, 5(4):956, 1985.

[9] JL Coatrieux, J Bassingthwaighte, PJ Hunter, J van Beek, P Tracqui, M Viceconti, D Testi, F Taddei, S Martelli, and GJ Clapworthy. Special issue on the physiome and beyond. *Proceedings of the IEEE*, 94(4):671–677, 2006.

[10] CV Gabel, H Gabel, D Pavlichin, A Kao, DA Clark, and ADT Samuel. Neural circuits mediate electrosensory behavior in caenorhabditis elegans. *Journal of Neuroscience*, 27(28):7586, 2007.

[11] PA Getting. Emerging principles governing the operation of neural networks. *Annual Review of Neuroscience*, 12(1):185–204, 1989.

[12] M Hall, E Frank, G Holmes, B Pfahringer, P Reutemann, and IH Witten. The weka data mining software: An update.

[13] Takaaki Hirotsu and Yuichi Iino. Neural circuit-dependent odor adaptation in c. elegansis regulated by the ras-mapk pathway. *Genes to Cells*, 10(6):517–530, Jun 2005.

[14] S Hochreiter and J Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[15] JJ Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554, 1982.

[16] Y Iwasaki and S Gomi. Stochastic formulation for a partial neural circuit of c. elegans. *Bulletin of Mathematical Biology*, 66(4):727–743, 2004.

[17] P Kohl, D Noble, RL Winslow, and PJ Hunter. Computational modelling of biological systems: tools and visions. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, 358(1766):579–610, 2000.

[18] A Majewska and R Yuste. Topology of gap junction networks in c. elegans. *Journal of Theoretical Biology*, 212(2):155–167, 2001.

[19] LR Varshney, BL Chen, E Paniagua, DH Hall, and DB Chklovskii. Structural properties of the caenorhabditis elegans neuronal network. *Arxiv preprint arXiv:0907.2373*, 2009.

[20] J.G White, E Southgate, JN Thomson, and S Brenner. The structure of the nervous system of the nematode caenorhabditis elegans. *Philosophical Transactions of the Royal Society of London*, 314:1–340, Nov 1986.