A Visualized Neural Simulator through WebGME

Zhengzhong Liang
ECE University of Arizona
1230 E. Speedway Blvd
zhengzhongliang@email.arizona.edu

Jonathan Sprinkle
ECE University of Arizona
1230 E. Speedway Blvd

Abstract

In this paper, a visualized neural simulator is proposed. The paper begins with the neuroscience and mathematical foundations of this visualized neural simulator. Then the paper discussed the meta model design and interpreter design of this simulator. Next, an example model created by this simulator is shown. Finally, some discussion of the advantages and disadvantages of this simulator is provided.

1. Introduction

1.1. Background

Our brain consists of about 10 billion neurons. Each neuron has around 7000 synapses connected with other neurons in average, thus forming an incredibly complicated network in our brain. Although the behavior of each neuron is quite simple, the huge network that all neurons form can achieve amazing cognitive tasks. Thus, studying the network of brain has always been a hot-spot of research.

However, since neural network is a non-linear dynamical system, it is nearly impossible to use analytical method to study the overall behavior of the whole network in brain. Instead, we tend to turn to computational methods for help. That is to build simulation model to study the behavior of neural network Language.

1.2. Related Work

Many tools aiming at neural network simulation have been proposed. [1] proposed a neural network simulator which enables user to build their own neural networks. However, the construction is through command, not through visualized graphs. User will have to write code to build the desired neural network. An older and more classic neural network simulator using the same method is NEURON from Yale university [2]. By inputting some particular command, user can construct large-scale neural network with realistic property. Nowadays, a new method emerges. In these simulation environments, constructing network is not only through code, but also can be achieved by selecting options in dialog [3]. This method provides a better user-software interaction experience. However, user still could not directly drag each neuron and combine them together to construct a neural network.

1.3. Neural Simulator

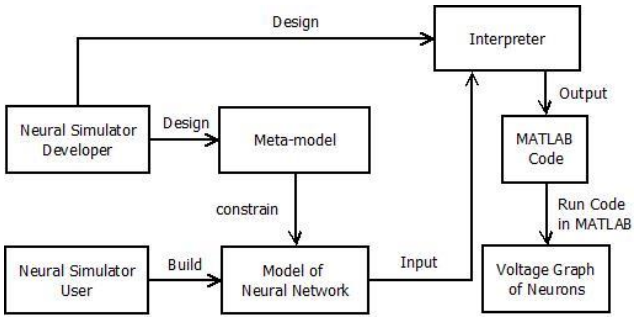

Figure 1: Organization of Neural Simulator

In this paper, a visualized neural network simulator is proposed. This simulation environment implements a domain-specific modeling language, which is particularly used for describing the structure and behavior of small-scale neural network. User will be able to drag the icons of neurons directly to workspace and build their own neural networks. The simulated neural network is biologically-reasonable, thus separating this simulator from other tools which aim at building artificial neural network. Those kind of modeling tools are usually used for constructing artificial neural network to execute data analysis tasks. However, the model in this simulator aims at describing the biological behavior of a real neural network.

The design of this visualized neural simulator consists of 2 major parts, one is model design, the other is interpreter. Each parts can be further divided into 2 sub-parts. Model design includes abstract syntax design and concrete syntax design. Interpreter design includes code generator design and graph checker design. An organization of this neural simulator is shown in figure 1.

The novelty of this neural simulator is mainly manifested by the user-simulator interaction. User will be able to

control each individual neuron and put them together to build a small-scale neural network.

2. Neuroscience Basics

Since this simulator is particularly used in neuroscience domain, it is necessary to introduce some neuroscience background. This section will provide a quick overview of some important properties that we want to model in this neural simulator.

2.1. Spiking Neurons

Spiking is the most fundamental behavior of the neuron in human's nervous system. When a neuron is stimulated by other neurons or cells, it can produce a series of electric impulses, which can travel along axons in one direction. Each electric impulse is called a 'spike'. After the impulses reach the end of axon, they are transmitted to next neuron. In this way, signals are transmitted in human's nervous system.

Although there are many types of neurons in human's nervous system, and different types of neurons have different types of spike amplitude, the amplitude of a specific type of neuron is fixed. In other words, one type of neuron can only spike at a fixed amplitude.

Given this spike property, stimulus with different strength is not encoded by the amplitude of spikes. Rather, it is encoded by the frequency of spikes. When stimulus is enhanced, the frequency that neuron spikes at will also increase. In contrast, if the stimulus is undermined, then the frequency of spikes should be reduced.

2.2. Touch Sensation System

Human's touch sensation system is a good illustration of how signals are transmitted in human's nervous system [4]. In figure 2, the left most part is called receptive field. It is composed of multiple receptor sensory neurons that can convert "touch stimulus" into "electric stimulus". The signal of this part is different from other types of signals in human's nervous system. Because unlike the signal of neuron, whose signal is "all-or-none", the signal in human's receptive field is graded.

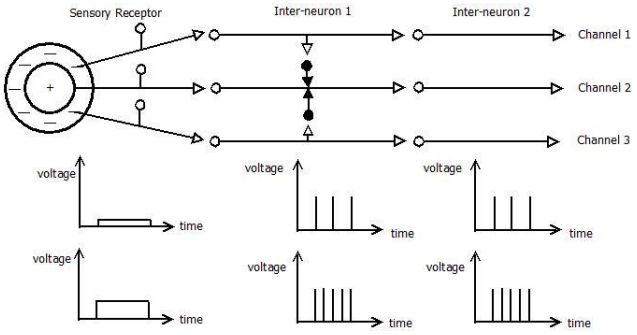

Figure 2: Touch Sensation System

Stimulus from receptive field will travel along axons of neurons, eventually reaching cerebral cortex. The ultimate destination of channel 1, channel 2 and channel 3 is cerebral cortex, where the touch sensation is finally produced. It is a little tricky that although the stimulus of touch is generated in receptive field in our skin, "the touch sensation" is instead generated in cerebral cortex.

From the receptive of skin to cerebral cortex, there are usually 2 to 3 inter neurons. The signal in inter neurons are "all-or-none". In other words, information is encoded by frequency rather than amplitude of spikes. The lower panel of figure 1 shows the pattern of signal in different neurons. As you can see, when the amplitude of signal in receptor sensory increases, the frequency of signal in inter neuron will increase as well.

2.3. Short-Term Memory

Unlike long-term memory in our brain, which is considered to be caused by the eternal modification of the structure of neural network, the formation of short-term memory has nothing to do with the modification of network structure. Instead, it is due to a special but fixed network structure, which is called "self-activation" [5].

Figure 2 shows the neural structure of short-term memory. There are 2 input sources in this neural network, with each connected to 1 particular channel. In each channel, there is 1 inter neuron, which is N1 and N2. When either of the 2 neurons is activated by stimulus from input, it will constantly activate itself though an excitatory axon attached to itself, meanwhile automatically inhibiting the activity of the other neuron. In this way, the particular input signal can be maintained in such a neural network, which ultimately forms short-term memory.

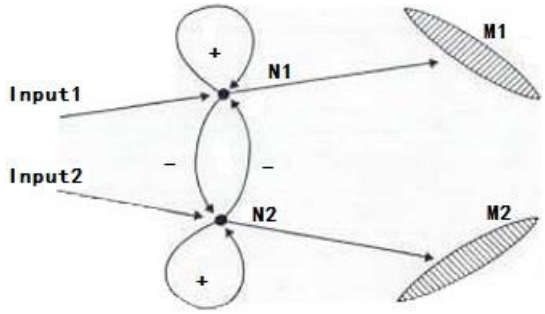

Figure 3: Neural Structure of Short-Term Memory

2.4. Synapse Plasticity and Learning Behavior

The learning behavior of brain is always the most fascinating topic in the study of brain. From an early time, scientists have been postulating that the learning behavior is highly related to synapse plasticity of neurons. However, it is not until recently that scientists have devised a formal mathematical formulation of synapse plasticity [6]. This behavior is called Spike-Timing Dependent Plasticity. The

idea is that depending on the timing that 2 neurons spike, the connection between them will be enhanced or undermined. Many perceptual phenomena can be explained by this theory in some way, including the early stage of language learning.

2.5. Functions Covered in This Neural Simulator

Given the properties of human's nervous system, some important features should be included in this simulator.

The first feature is the strength-frequency property of a single neuron. Since the strength of stimulus could influence the rate at which neuron spikes, the neuron model in this neural simulator should manifest it. The larger the stimulus is, the more intensely should each neuron spikes.

The second property is the modeling of different axons. In human's neural system, there are both inhibitory axons and excitatory axons. Moreover, the strength of each axon is different. If the strength of an axon is rather large, then a larger portion of stimulus will be conveyed to the next neuron. If the strength of axon is small, then a less portion of stimulus should be conveyed. Thu, there should be multiple types of axons in this neural simulator.

Thirdly, the network should have both start point and terminal. Start point is where stimulus is produced. They serve as the source of a network. If there are no start points, then the voltage of the whole network will not change. Terminal is the point where the signal is eventually transmitted to. In human's nervous system, it is usually glands or muscle. So in our simulator, we should also include the component like this.

Given all of the features above, we expect the simulator to be able to simulate some basic neural networks, like the touch sensation network and self-activation network.

The modeling of learning behavior will not be included in this simulator. The main reason is that learning process is a long-term process. The changing of the strength of axons is rather slow, which usually takes several hours or several days. If we model that process, the data will be too large to handle.

3. Mathematical Models of Neural Activity

The neural simulation will eventually be carried out by calculation of computer, and before calculation the behavior of neural system should be formulated in mathematical language in advance. In this section, I will review the relevant mathematical theories of neural activity. The later design of model and interpreter will be based on the mathematical theories mentioned in this section. These mathematical theories in essence define how the neural network behaves in our neural simulator.

3.1. Mathematical Formulations of a Single Neuron

Since the discovery of the spiking behavior of neurons, multiple mathematical models have been proposed to depict this behavior. These models have different perspectives. Some models aim at describing the individual behavior of the ion channels on neuron's membrane, thus describing the voltage variation of neurons. A typical method inspired by this idea is Hudgkin-Huxley model. This model describes the behavior of Potassium ion channel, Sodium ion channel and Chlorine ion channels on neuron's membrane. Each type of ion is described by a differential equation. And the whole Hudgkin-Huxley model is described by a set of differential equations. However, since the resistance of membrane is no a constant, these differential equations are not linear, thus making is hard to calculate [7].

Hudgkin-Huxley model can simulate the voltage property of different types of neurons, whose reaction are different from each other when stimulated. However, since the computational resources need for solving non-linear differential equations, Hudgkin-Huxley model is not suitable for network simulation. The iteration step for simulating a single neuron should be as small as 0.01 ms if you choose Hudgkin-Huxley model. So if you want to model the behavior of single neuron using Hudgkin-Huxley model for 1000 ms, at least 100000 times of calculation is needed. That consumes too much computational resources.

There are also other types of models to describe the behavior of neurons. These types of models are inspired by the idea that we do not have to model every component on neuron. Instead, depicting the overall behavior of the neuron is enough. A typical model enlightened by this idea is Leak Integrate-and-Fire (LIF) model. This model does not consider the ion channels on neuron's membrane. It only considers 3 property of neuron:

1, When stimulus current reaches neuron, the neuron is charged, which will raise the membrane potential of a neuron.
2, When the membrane potential of a neuron reaches a fixed value, it spikes immediately.
3, After spike, the membrane potential of neuron is reset to resting value, waiting for next spiking.

Figure 4: Sketch of LIF Algorithm

Following these important principles, Leaky Integrate-and-Fire neuron model was proposed. The term "integrate" refers the charging process of a neuron. The term "fire" means that after charging process is completed, the neuron can fire a spike. The term "leaky" means that when there is no more stimulus, the membrane voltage of a neuron will decay exponentially.

The advantage of LIF model is that is computationally reasonable. This model is described by only 1 differential equation, thus the iteration step in simulation can be rather

large. But there is disadvantage. Since there are only a few parameters in this model, the variation of this model is pretty limited, which means that it can only simulate several types of neurons.

Since neither type of the mathematical models above about neuron is satisfactory, we should further look for a better one. In 2003, E.M. Izhikevich proposed a simple mathematical model that describes the behavior of a single neuron [8]. This model is the desired one because it is both biologically realistic and computational reasonable.

Izhikevich model uses 2 variables and 4 parameters to describe the behavior of a single neuron. The set of differential equation is like this:

$$v' = 0.04v^2 + 5v = 140 - u + I$$

$$u' = a(bv - u)$$

and:

$$if\ v \geq 30, then \begin{cases} v \leftarrow c \\ u = u + d \end{cases}$$

In these equations, v is the membrane voltage of the neuron, and u is the recovery variable. There are 4 parameters in the differential equations. By changing the values of these equations, Izhikevich model can simulate many types of neurons. In addition, it is pretty ideal that this computation does not require much computational resources. So Izhikevich model actually inherits the computational advantage from LIF model and biological advantage from Hudgkin-Huxley model. So in the neural simulator, Izhikevich model is implemented to fulfill the simulation task.

3.2. Mathematical Model of Synaptic Transmission

The mechanism of synapse of between neurons is quite complex. Here we use a simplified mathematical model. The idea of the mathematical model is from Gerstner [6].

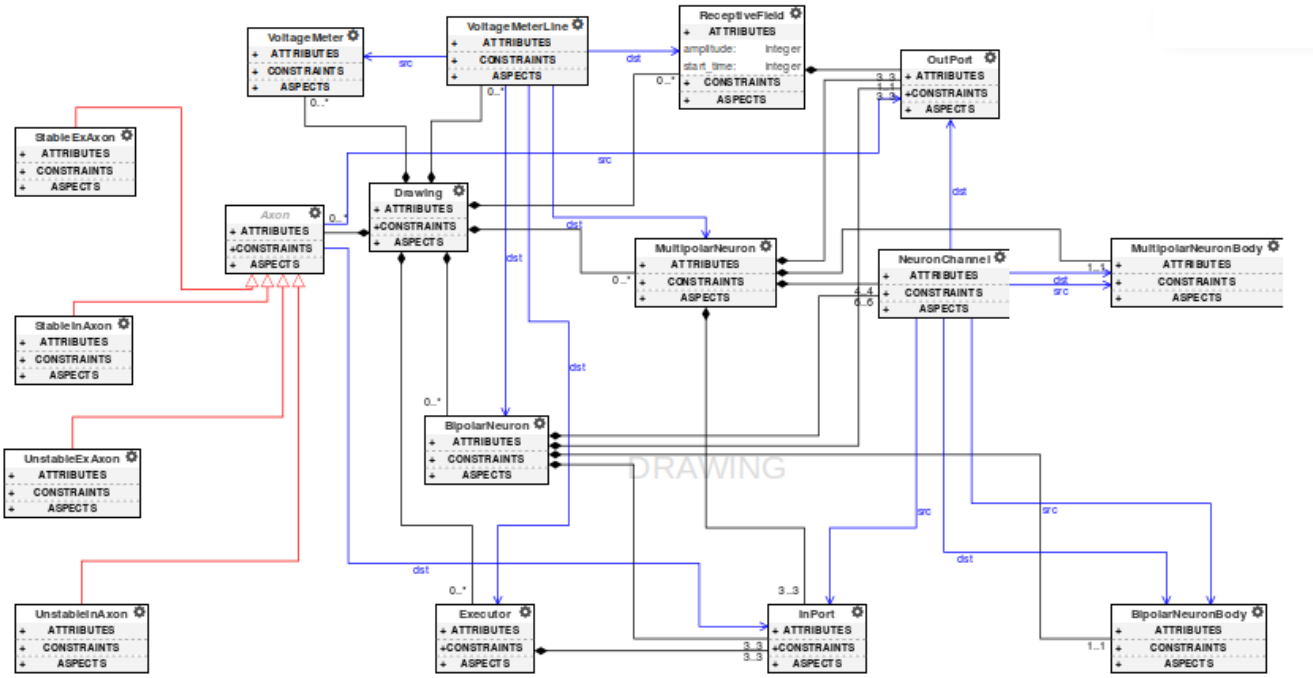

Figure 5: Meta-Model Design

According to Gerstner, two important features of inter-synaptic signal transmission should be included. The first is that the spikes in pre-synaptic neuron in converted to current when conveyed to another neuron. Second is that the current at present state is influenced by the current at previous state. Based on these 2 facts, we describe the post-synaptic current as:

$$I_j^{now} = 0.98 * I_j^{last} + \sum_k^3 (0.03 * w_{kj} * v_k)$$

In this equation, j is the post-synaptic neuron, which is stimulated. K is the number of axons connected to neuron j. Since there are three axons, k should be 1, 2 and 3. I_i^{now} is the input current at this state. It is the summation of the stimulus posed by the present voltage and previous current. v_k is the voltage of pre-synaptic neuron. In this equation, it is pre-processed so that the value of it is above 0.

4. Simulator Model Design

4.1. Function Analysis

As is analyzed in section 2, a neural network must have a start point and a terminal. Besides, there should be inter

neurons which can process information between start point and terminal. Thus, there are four types of cells in this neural simulator: receptive field, bipolar neuron, multipolar neuron and executor. Besides, there are 4 types of axons in this neural simulator. Some of these axons are excitatory and the others are inhibitory. These 4 types of axons have different value, which means they have different efficiency conveying signals between neurons. At last, there are voltage meters and voltages meter line. User can place one or several voltage meter in this simulation environment and connect them to different neurons to monitor their membrane potential.

4.2. Abstract Syntax

In this neural simulator, all cells and other components are contained by the class "drawing". It means that all cells and components has its own appearance. Each of them are defined visually in this neural simulator. In the object of drawing, users can create cell objects and make connections among them to build their own neural network. An object of "Drawing" is an individual neural network model.

All components that can be created in a drawing are contained by drawing. In figure 5, receptive field, multipolar neuron, bipolar neuron, executor, axon, voltage meter and voltage meter line are all contained by drawing. These components are all components that user can create in a model.

There are also some components that user could not build directly in their drawing, but also shown in this figure. For example, users are not allowed to create "InPort" or "OutPort" directly in drawing. These components are automatically and can only be automatically created when an object of cell is instantiated.

Figure 6 shows the containment relationship among BipolarNeuron, BipolarNeuronBody, Inports and OutPorts. A Bipolar neuron will spontaneously have 3 objects of

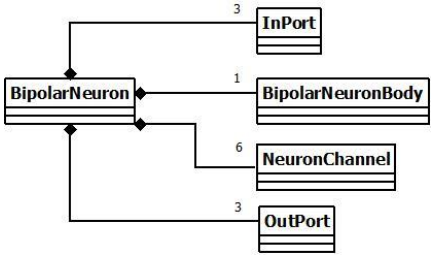

Figure 6: Meta-Model of Bipolar Neuron

InPort, 3 objects of OutPort and 1 objects of Bipolar-NeuronBody. The InPort the interface that BipolarNeuron-Body communicate the other neurons. InPort can only be the destination of other axons. Similarly, OutPort is also the interface of BipolarNeuronBody, but it can only serve as the source of axons.

This containment structure reflects the biological structure of a real bipolar neuron. A real bipolar neuron has 3 major parts: soma, which is the neuron body, dendrites and axons. Dendrite is the connection port to receive signal from other cells, and axon is the connection port from which the neuron body pass signal to another cell. In the model of neural simulator, this InPort stands for dendrites and OutPort stands for axons.

The cardinality of containment restricts the number of parts in a bipolar neuron, thus preventing user from building inappropriate model.

Similarly, the multipolar neuron has a similar structure to the bipolar neuron. The only difference is that multipolar neuron has only one OutPort (figure 8).

When it comes to receptive field and executor, the structure of them are much simpler. Since receptive is the start point of a neural network, and it is the component which can produce stimulus, it does not require input. As a result, it has only 3 objects of OutPort but no objects of InPort. In contrast, executor is the terminal of a neural

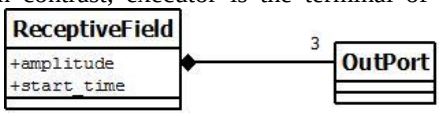

Figure 7: Meta-Model of Receptive Field

network, so it has only objects of OutPort but no objects of InPort (figure 7).

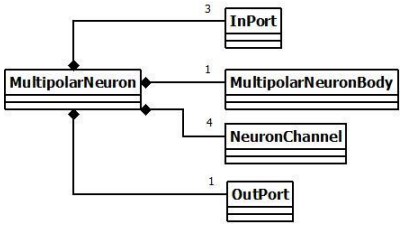

Figure 8: Meta-Model of Multipolar Neuron

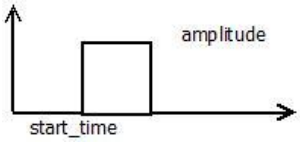

Figure 9: Stimulus

Receptive field is a little special because it has 2 attributes. These 2 attributes are relevant to the type of stimulus that receptive can produce. In this neural simulator, the simulation length is fixed, which is 1000 ms. The type of stimulus that receptive field can produce is a function shown above. It is a square wave signal. This signal is in accordance with the realistic signal in human's nervous system.

The square wave signal's duration is also fixed, which is 400 ms. This duration value is carefully chosen so that there

can be 2 un-overlapped input signal in one simulation. Although the duration of stimulus is fixed, the start time and amplitude of duration can be changed. This is why receptive field has these 2 attributes. User can modify the value of these 2 attributes in the model to make the receptive field produce different stimulus (figure 9).

The range of amplitude is from -10 μA to 30 μA, and the default value is 20 μA. The range of start time is 0 ms to 500 ms, and the default value is 300 ms. The default value is set when an object of receptive field is instantiated.

This figure shows the inheritance relationship of axons and its association with InPort and OutPort. In this neural simulator, axon is defined as a type of "connection", so it has destination and source. Intuitively, the source of destination of axon should be some types of neurons. However, that is not the case in this simulation environment. In this simulator, the source of axons is OutPort and destination of axon is InPort.

This rule seems to be counter-intuitive, but it actually does not belie the intuition. Since neurons automatically contain some objects of InPort of OutPort, connected to InPort or OutPort has the same result as connected to different types of neurons. Then what is the reason for defining InPort and OutPort? There are 2 main reasons. The first reason is that the existence of InPort and OutPort guarantees the resemblance between the model in this simulator and the real biological structure of neurons. As we have discussed, the InPort refers to dendrites of neurons and OutPort refers to axons of neurons. So defining such port can make user to connect the model with the structure of real neurons.

The second reason is that this these ports will be displayed explicitly in the composition view. For example, a bipolar neuron has 3 object of InPorts and 3 objects of OutPorts. These ports will be shown on the icon of a bipolar neuron. And since the axon is connected to port not the neuron, the alignment of axons is pretty regular. Users can easily figure out how many axons have been connected to a neuron.

The class "Axon" is defined as an abstract class, because although there are 4 sub types of axons, the behavior of them is total the same. If axon with type A can be connected to a port, then axon with type B should also be allowed to do that. Thus, we use an abstract class to regulate the behavior of all types of axons.

The differences between the sub types of axons are the values of them. Depending on whether an axon will excite or inhibit the post-synaptic neuron, the axons can be divided into "excitatory axons" and "inhibitory axons". The excitatory axons have positive value, which is the current version is 0.7 or 0.3 and the inhibitory axons have a negative value, which currently is -0.7 or -0.3.

Whether an axon is stable or not could also influence the value of axon. Stable axons are well-established axons. They form due to usually a long period of stimulus. And as long as they become stable, the speed at which they decay is very slow. It is believed by some scientists that as soon as the strength of axon exceeds a threshold, it will automatically strength itself, until the upper bound of the strength. In contrast, the unstable axons are those which are not so well established. In other words, the strength of them has no reach the threshold. In this case, the strength of them will constantly decay, eventually reaching 0.

Enlightened by these ideas, there are 4 types of axons. The stable excitatory axon has the largest positive strength,

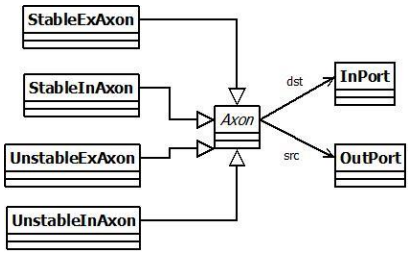

Figure 10: Meta-Model of Axon

which can strongly excite the post-synaptic neurons. The unstable excitatory neuron, which can also excite the post-synaptic neuron, has a weaker strength, which is 0.3. It means that unstable excitatory axons could slightly excite the post-synaptic neuron. The unstable inhibitory axon, with a strength of -0.3, could somehow inhibit the activity of post-synaptic neuron, and the stable inhibitory axon, with the value of -0.7, could place strong inhibition upon the post-synaptic neuron.

It was conceived that the spontaneous decaying and strengthening could be included in this simulator. However, this effect was not achieved in this version.

The last 2 components in this simulation environment are voltage meter and voltage meter line. Voltage meter can

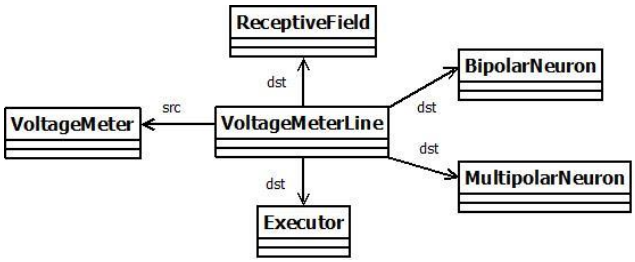

Figure 11: Meta-Model of Voltage Meter Line

monitor any types of cells in this simulator, but must achieve this through a voltage meter line. By connecting a

voltage meter to a cell with a voltage meter line, user can specify which cells should be monitored.

4.3. Concrete Syntax

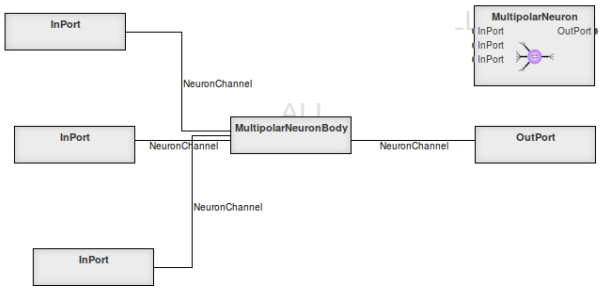

Figure 12: Concrete Syntax of Multipolar Neuron

Abstract syntax defines the data structure of the neural simulator. By defining abstract syntax using meta-model, we defined what types of data that user can create.

However, user will not have chance to edit meta-model

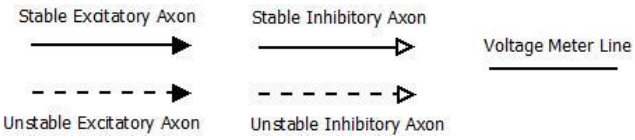

Figure 13: Concrete Syntax of Axons

directly. Instead, neural network models are built through the utility of the exact objects. In this section, I will show the exact appearance of each type of component in this neural simulator.

The appearance of a bipolar neuron is shown in figure 14. It has 3 InPorts and 3 OutPorts. The position of each port is explicitly displayed in the model. So that user will not get confused about how many axons have been connected and

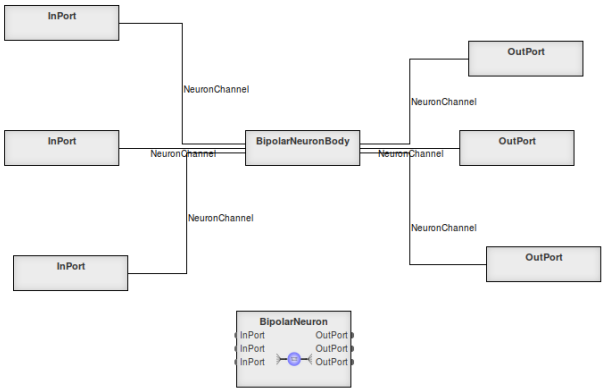

Figure 14: Concrete Syntax of Bipolar Neuron

how many ports are still available. A SVG icon is imported to this model too. The SVG icon depicts the neuron body as a blue circle. And there are 2 bunch of axons on that circle, indicating this is a bipolar neuron.

A model of multipolar neuron is shown in figure 12. A multipolar neuron has 3 InPorts and 1 OutPort. The SVG icon of a multipolar neuron is a purple circle with 4 bunches of connections, indicating that this neuron can connect with other 4 neurons at one time at most. A view of inner structure of multipolar neuron model is given.

A receptive field has only 3 OutPorts but no InPorts. The SVG icon of this receptive field is a green circle divided into 3 sub areas. A model of an executor is shown in figure 15. It has 3 InPorts but no OutPorts. The SVG icon of an executor is a red circle with one bunch of connection.

The appearance of voltage meter is shown in figure 15. They also have SVG icons. And the appearance of connections is shown in figure 13. A solid arrow indicates that it is an excitatory axon, where as an empty arrow indicates that it is an inhibitory axon. The line pattern indicates whether the axon is stable. If it is stable, then the line should be solid; if not, the line should be dashed. And there is voltage meter line in model, which is a solid line without any arrow. Since different connections have different appearance, it is easy to distinguish them.

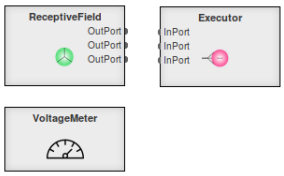

Figure 15: Concrete Syntax of RF, Ex and VM

4.4. Constraint

Some constraints are defined by meta-model. For example, axons can only be connected to the InPort object

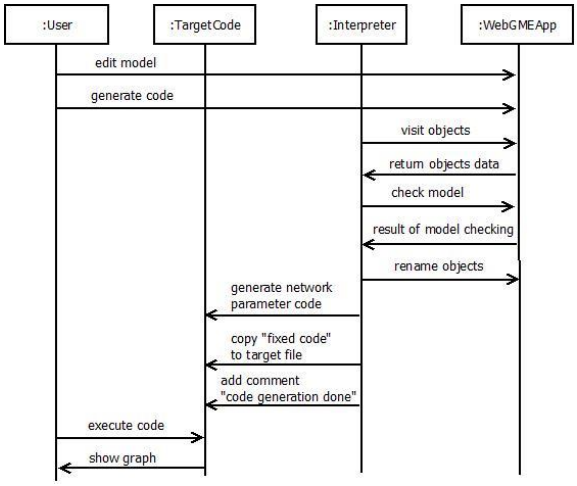

Figure 16: Sequence Model of Interpreter

or OutPort object, but not each neuron. In this way, user can easily count the number of connections attached to a neuron. And it is clear to see how many ports are still available in a neuron. Another constraint achieved by meta-

model is the number of contained objects. For example, a bipolar neuron must have exactly 3 objects of InPort, 3 objects of OutPort, 1 object of BipolarNeuronBody and 6 objects of NeuronChannel. This constraint can largely prevent accident modification to the structure of each neuron.

5. Interpreter Design

5.1. Interpreter Analysis

The function of interpreter is to convert the model that the user builds in simulator to executable code. In this simulator, the model will be converted to MATLAB code. After the MATLAB code is generated, it could be executed in MATLAB and the simulation of neural network will start.

A sequence model of the process is shown in figure 16. Interpreter will interact with the WebGMEApp, user and target code. When an interpreter starts to work, it firstly visits the model that the user has built in WebGMEApp, then return the data of all objects. Secondly, before the code generation, the interpreter will check whether the established model is valid. For example, if some the model violates some rules, interpreter will give message to user and ask user to modify the model. Next, if model checking is finished and no problems detected, the interpreter will give each neuron a unique name. The form of name is "neuron type + number". For example, if there is a bipolar neuron, and its name could be "Bipolar Neuron 2". The number of object always starts from 1. The activity diagram of interpreter is also shown.

5.2. Target Code

The target code is the code that we want interpreter to generate. Target code can be divided into 2 parts, the variating part and the fixed part. The fixed part in the piece of code that describes the simulation process of neural network. The process includes updating each neurons status and output voltage graph of the neurons that are being monitored. Since this part of procedure is the same to different neural networks, this part of target code is fixed.

The variating part of code is responsible for storing the parameters of the neural network that a user builds. These parameters described the structure of neural network. The parameters include: 1) how many neurons are in this network in total, and what the type of each one. 2) the connection matrix of the whole neural network. 3) what neurons are being monitored by voltage meter. 4) what is the output signal of each receptive field.

The relationship among variating part, fixed part and the whole target code is shown in figure 18. The target code consists of variating part and fixed part. The variating part describes the structure of model. And the parameters in variating part is the input of the fixed part (figure 18).

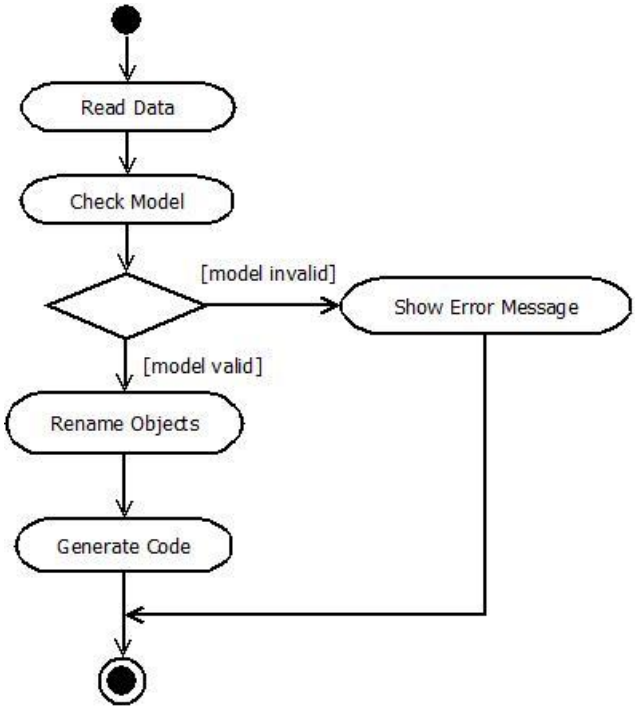

Figure 17: Activity Diagram of Interpreter

5.3. Data Representation

The parameters of variating part are represented by arrays and matrix. Array "typeAndID" stores the type of each neuron. For example, there are 2 receptive fields, 1 bipolar neuron, 1 multipolar neuron and 1 executor and 1 voltage meter in the network, then the "typeAndID" array should be:

typeAndID=['r','r','b','m','e','v'];

The element of this array is the type of neuron. 'r' stands for receptive field, 'b' stands for bipolar neuron, 'm' stands for multipolar neuron and 'v' stands for voltage meter. The ID of each neuron is its index in this array. In this way we can distinguish different objects.

The connection matrix describes the network structure. A sketch of a connection matrix is shown in figure 19. Each row in this matrix represents an InPort obeject. Since each bipolar neuron has 3 objects of InPort, each bipolar neuron takes up 3 rows in this matrix. The same situation happens

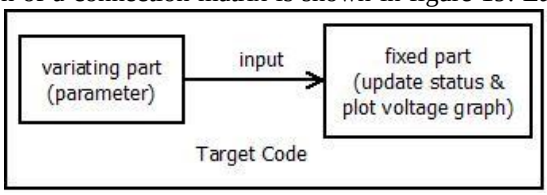

Figure 18: Parts of Target Code

to multipolar neuron and executor. Each multipolar neuron or executor takes up 3 rows in this matrix. Since receptive field has no InPort objects, it will take up no rows.

Each column in this matrix stands for an object of OutPort. Each receptive field or bipolar neuron has 3 objects of OutPort, so either of them takes up 3 columns in this matrix. A multipolar neuron has only 1 OutPort object, so each multipolar neuron takes up 1 column in this matrix.

Each element in this matrix indicates whether there is an axon between an InPort object and an OutPort object. For example, if W stands for the whole connection matrix, and receptive field 1 has a stable excitatory axon connected to bipolar neuron 1, then W(1,1) should be 0.7. If receptive field 1 has another axon connected to bipolar neuron 1, and the axon is a stable inhibitory axon, then W(2,2) should be -0.7.

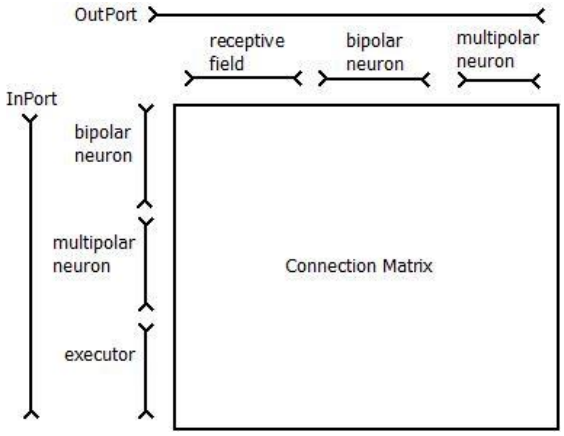

Figure 19: Connection Matrix

The neurons that are being monitored are stored in a 1-d array. Each element in that array is the ID of the monitored neuron. The array name is "subjecN" in target code.

The signal of receptive field is stored in a 2-d array named stimulus. The structure is like this:

stimulus=[stimulus1's start time, stimulus1's amplitude;

stimulus2's start time, stimulus2's amplitude;

……]

5.4. Simulation Algorithm

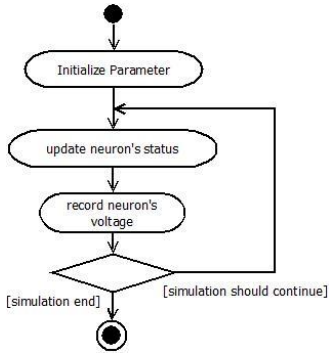

Figure 20: Activity Diagram of Simulation Program

The simulation procedure is shown in figure 20. The simulation program uses the parameters that are sent by interpreter to initialize the neural network. Then simulation program will execute simulation loop, until the simulation should end. In each loop, the program will record the voltage of each neuron.

The update of the status of neuron implements Izhikevich algorithm, which is shown in section 3. The four parameters: a, b, c, d in Izhikevich model are those which determine the property of a neuron. Although each neuron has an individual set of a, b, c, d in this simulation algorithm, at present different neurons have the same value of a, b, c, d. Thus different neurons have the same property.

5.5. Plot Function

There are two types of plot function in target code. If there is only 1 voltage meter in the model that the user builds, then the "animated" plot function will be recalled. The mechanism is to plot an individual graph in each simulation loop, so that the voltage graph is update in each loop. So it will show an animated effect.

The other plot function is an ordinary one. If there are more than one voltage meters in model, then the ordinary plot function will be recalled. The plot function will plot the voltage graph of all neurons that are being monitored, and the title of each graph will be modified accordingly.

5.6. Connected Graph Checker

Another individual plugin that this simulator uses is a "connected graph checker". The name in the simulator is "NSGraphChecker". This is an independent plugin, which can detect the number of connected graphs in current container. And a node in the smallest container will be shown in the dialog of this plugin.

Since the number of connected graphs will not influence the code generation, this plugin only serves as a reference

to user. No matter how many connected graphs are in the model, the code will be generated.

5.7. Model Checker

Model checker is part of interpreter. Before the code can be generated, the model checker will firstly examine whether the model is a valid model. 3 types model are invalid. And in those cases, the code will not be generated and an error message denoting the error will be displayed in the plugin dialog.

Those invalid features include: 1) there are more than 1 “drawing” object in model. 2) more than one axon was added to an InPort or OutPort and 3) there are no cells in the model.

5.8 Code Generation and MATLAB Output

If the model is valid, the target MATLAB code will be generated in the application’s directory. For this simulator, it should be at /neuralsimulator/MatlabSimulation.txt. User can copy the code to a MATLAB script and run it. Then the voltage graphs of neurons will be shown.

6. A Simulation Example

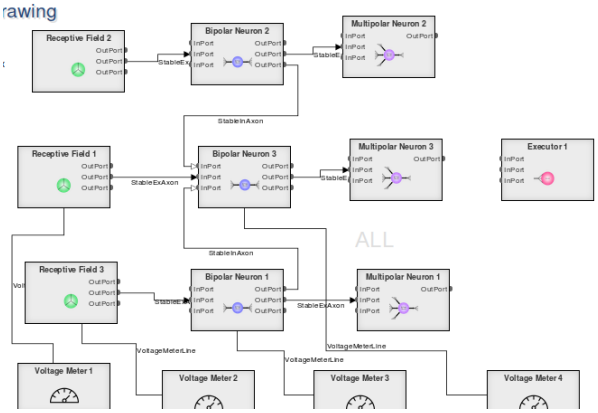

Figure 21: Sample Model

Figure 21 shows a sample model built by this neural simulator. There are 3 receptive fields, 3 bipolar neurons and 3 multipolar neurons. An executor was added but not connected to any neuron. 4 Voltage meter were added to monitor the voltage of some neurons.

7. Conclusion and Prospects

In this paper, a visualized neural simulator is proposed. Then the neuroscience and mathematical background are given. The paper then discussed the technical merits of the design of this neural simulator. The technical merits include meta-model design and interpreter design. Finally, a simulation example is given.

The advantage of such a simulation environment is that it is useful for students who are interested in neuroscience but do not have much knowledge about this field. Then this simulator can help them to establish a direct understanding of how neural network behaves. Students can use this simulation environment to simulate multiple nervous phenomena.

The disadvantage of this simulation environment is that it is still not covering the learning behavior. Maybe this part of function can be added to a later version of this simulation. Moreover, at present the characteristics of bipolar neurons and multipolar neurons are the same. This situation can be changed by changing the parameters of some specific types of neurons. The design of the simulation algorithm internally allows this changing and this task can be achieved easily. However, since the testing of changing these parameters takes a lot of time, it is not done in the current version.

References

[1] Stimberg M, Goodman D F M, Benichoux V, et al. Equation-oriented specification of neural models for simulations[J]. Frontiers in Neuroinformatics, 2014, 8: 6.

[2] Hines M L, Davison A P, Muller E. NEURON and Python[J]. 1984.

[3] Aisa B, Mingus B, O’Reilly R. The emergent neural modeling system[J]. Neural networks, 2008, 21(8): 1146-1152.

[4] Siegelbaum S A, Hudspeth A J. Principles of neural science[M]. New York: McGraw-hill, 2000.

[5] Grossberg S. Contour enhancement, short term memory, and constancies in reverberating neural networks[M]//Studies of mind and brain. Springer Netherlands, 1982: 332-378.

[6] Gerstner W, Kistler W M. Mathematical formulations of Hebbian learning[J]. Biological cybernetics, 2002, 87(5-6): 404-415.

[7] Izhikevich E M. Which model to use for cortical spiking neurons?[J]. IEEE transactions on neural networks, 2004, 15(5): 1063-1070.

[8] Izhikevich E M. Simple model of spiking neurons[J]. IEEE Transactions on neural networks, 2003, 14(6): 1569-1572.