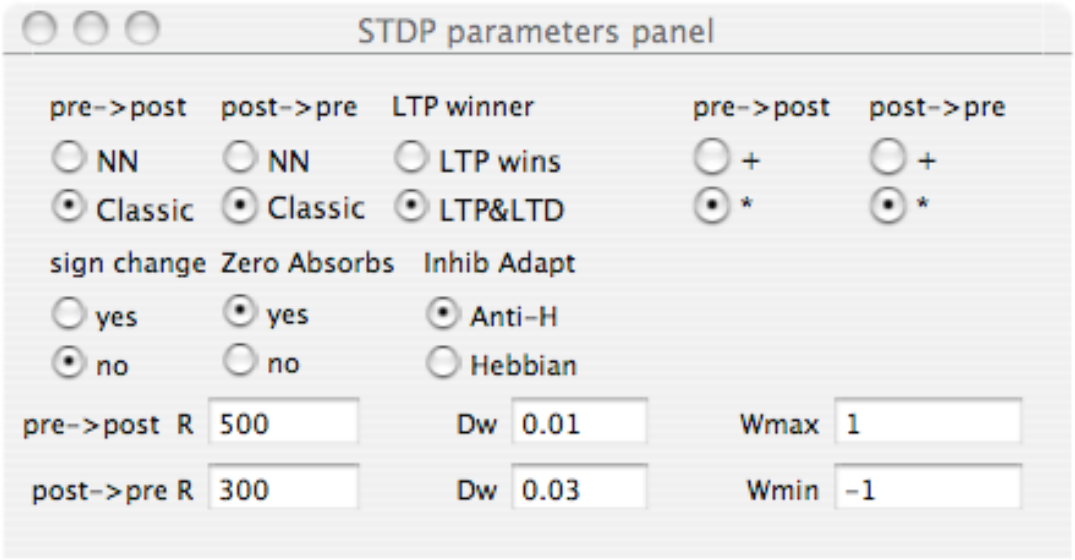
Neuron 1 Documentation. Program version 0.57 (1.0)
Last updated May 19 2008.

Overall:
Program for simulating a network of interconnected leaky integrate and fire (LIF) neurons with STDP.

Running the program:

1. Start up the application
2. Load the simulation parameters (File: Load Sim Parameters)
3. Load the Default neuron parameters (Neurons: Load Default neuron Params)
4. Load up the synapses (Synapses: Set synapse file)
5. Load up the input spikes (if any). (Spikes: Set simple spike file or Spikes: Load generated spikes)
6. (optional) Load the Noise parameters (File: Load Noise Parameters)
7. (optional) Load the non-default neuron parameters (Neurons: Set non default neurons para file)
8. Set up the STDP parameters appropriately (using the STDP parameters panel)
9. Set the simulation up by pressing the Set up network button in the main view.
10. Run the simulation by pressing the Start button in the main view.
11. (Optional) Save the spikes generated using Spikes: Save generated Spikes or Spikes: Save generated spikes (human readable). Note that saved spikes are in a format where they can be re-used in stage 5 above.
12. (Optional) Save the synapses (Synapses: Save synapses). Note that the saved synapses may be re-used as in stage 4 above.

Setting the STDP parameters.



From the top: radio button for the nature of the alteration of the pre->post synapses. The options are nearest neighbour (NN) or using all the presynaptic spikes within some relatively long time (Classic). NN uses the time between the postsynaptic spike and the most recent presynaptic spike in the calculation of the delta weight, whereas Classic uses all the presynaptic spikes (one by one) in this calculation. The second radio button has the same effect, but for the post->presynaptic weight change. The next radio button selects whether, when the STDP for pre-> post implies LTP (long term potentiation – i.e. weight increase for an excitatory synapse) LTD (from post-> pre) can be applied as well. The next two radio buttons select whether weight changes are additive or multiplicative.

In the second row, the first radio button selects whether sign change of weights is permitted. The second one selects whether a weight on 0 will "absorb": that is if the weight reaches 0, the weight becomes non-adaptive, and stays at 0. (In other words, if the weight hits 0, the synapse essentially disappears). The next radio button selects what happens to inhibitory synapses under STDP. Choosing Hebbian means that they are treated the same way as excitatory, and choosing Anti-H means that they move in the opposite direction.

The third row sets the parameters for the STDP. The value of R is the time constants used in the calculation of the effects of the delta t (time between pre/post or post/pre spikes). This is used as a multiplier: the weight change is $\Delta w = Dw * \exp(Dt * R)$ (times W or 1-W if adaptive). Dw, is the weight change size parameter, see the above equation. The maximal and minimal values of the weight may also e set.

Classes Overview:

NeuronView

Primary view class, holding all the display and drawing materials

Network

Holds an array of neurons, and a random number generator

InputNeuron

Input neuron object: takes external input (should be 1/0) and "fires" on a 1 input

Neuron: subclass of InputNeuron

Basic LIF neuron object. Contains an array of presynaptic Synapses.

synapse

Basic synapse: also contains history of what has happened at this synapse.

Dynsynapse: subclass of synapse

Dynamic synapse: has utilisation and recharge rates.

Shuntsynapse: subclass of synapse

Shunting (multiplicative inhibitory) synapse

SpikeArray

Used for storing the spikes generated during a simulation run. Holds the array of spikes

Spikes

Used to hold the SpikeArray object, and actually stored (using archiveRootObject) when spikes are saved.

Random

Random Number Generator (truly ancient: modified from code written in 1991 by Gregor Purdy)

Operational overview.

On startup:

By default the `initWithFrame` method in the primary view class is called. This creates the network (called NeuralNetwork), and the random number generator (Rng).
NSMutableDictionaries are set up for the simulation parameters (`SimParamnames`), for the Neuron parameters (`NeuronParamnames`), and for the Noise parameters (`NoiseParamnames`), and these objects are made available to the network NeuralNetwork using the `supplyParameters` method of the object NeuralNetwork.
The noise level is set to 0 both on the interface and using the method `readNoiseParams`.

The system then waits for user input. What should happen is that the various parameters should be set, and then the Set up network button should be pressed.

The parameters may be set either directly, using the interface, or (more likely) using the Load SimParameters and Load Noise Parameters menu items (under File).

Loading simulation parameters. (File: Load Sim Paramaters)

Invokes loadSimParams in NeuronView. This brings up a file open dialogue box, for files with suffix .SimParams. SimParamnmes is initialised with the content of this file (using the initWithContentsOfFile method of the NSMutableDictionary object). These are then applied to the network using the applySimParameters method of NeuralNetwork, and displayed using the showSimParameters method of NeuronView.

The number of neurons is set in the setup method of Network. This is called from setupNeuralNetwork which is invoked when the set up network button is pressed. It uses the number of neurons in the display to set the total number of neurons. However, it demands that applySimParameters has been invoked first. Pressing the Load Sim Parameters button more than once with different values for the number of neurons will give an error and the number of neurons will not be altered.

Saving Simulation Parameters (File: Save Sim Parameters}

Invokes the saveSimParams method of NeuronView. This brings up a file save dialogue, with suffix SimParams, and saves the NSMutableDictonary using the writeToFile:atomically method.

Loading Noise Parameters. (File: Load Noise Params)

Invokes the loadNoiseParams method of NeuronView. This brings up a file open dialogue box, for files with suffix .NoiseParams. The file is then used to initialise the NSMutableDictionary NoiseParamnames. The applyNoiseParamaters method in NeuralNetwork is then called. This sets the different types of noise (tonic, threshold and activation level: these are the same for all the non-input neurons). Lastly, the noise values are displayed using the showNoiseParameters method in NeuronView.

Saving Noise Parameters. (File: Save Noise Params)

Invokes the saveNoiseParams method in NeuronView. This brings up a file save dialogue, with suffix NoiseParams, and saves the NSMutableDictonary using the writeToFile:atomically method.

Setting the (default) neuron parameters.

The button Neurons: Load Default Neuron Params invokes the method loadNeuronParams in NeuronView. This brings up a file open dialogue box, for files with suffix .NeuronParams. The file is then used to initialise the NSMutableDictionary NeuronParamnames. These are then displayed using the method showNeuronParameters of NeuronView.

The button Neurons: Save Default Neuron Params invokes the method saveNeuronparams in NeuronView. This brings up a file save panel, with suffix NeuronParams, and saves the NSMutableDictonary using the writeToFile:atomically method.

The button Neurons: Set non-default neuron params file invokes the method setNonDefaultNeuronParamFile in NeuronView. This brings up a file open panel, with file name suffix neurons. The file name is stored in the object variable NonDefaultNeuronParamFile. The file format is
<neuronnumber zerolevel initphase tonicvalue threshold refperiod dissipation squaredissipation minimum_activation>*
where the neuronnumber is an integer, and all the others are floats.

The values provided are displayed in the Neural network parameters panel, where they can be updated. Note that the zerolevel is the level to which the activation of a (non-input) neuron is reset at the start and after firing: if the minimum activation level is greater, then the neuron will be reset to this level.

Setting up the synapse file (Synapses: Set Synapse File)

This invokes method setSynapseFile in NeuronView. This brings up an open file panel dialogue, with the file suffix set to .synapse. It saves the file name to SynapseFile, and displays the name of the synapse file in the synapse file box on the main view window.
The synapse file is a text file, and its format is
<type presyn postsyn weight alpha delay (newline)> where the 1st character of the type is not 'd', and
<type presyn postsyn weight alpha delay rechargerate utilrate (newline)> where the 1st character of the type is 'd'.
The type is (ugh) a character string, where the 1st character is s (plain synapse) d (depressing) or m (shunting), and characters 2,3, or 4 may be blank, or may contain an a (synapse is adaptive) and/or an r (synapse resets alpha function on its next presynaptic input.

Saving the synapses (Synapses: save Synapses)

This invokes the method saveSynapses in NeuronView. This brings up a save file dialog, with the suffix .synapse. Actually saving the synapses is performed by the saveSynapses method in NeuralNetwork.

Setting up the Network (Set up network button in "Spiking Neural Network Simulator" view)

This invokes the setUpNeuralNetwork method in NeuronView.
First this interrogates the radio button DisplayRadio (on the Neural Networks Parameters panel) to find out whether all spikes or only non-

input spikes are to be stored. The variable displayAll is set to show the result.

The STDP parameters are then loaded by invoking the method getSTDPParameters in NeuronView. This picks up all the parameters set in the STDP parameters panel, and places them in the structure adaptparams.

The neural network itself is then set up by invoking the method setup in NeuralNetwork. The parameters passed are MAXSPIKES (set in NeuronView.h) which is the maximal number of spikes to be recorded, and displayAll (see above). This clears then recreates the spikesout array, used to hold the neuron number and time of any spikes generated. Critical bvalues from the SimParamnames dictionary are stored as object variables (input_neuron, the number of input neurons, n_neuron, the total number of neurons, duration, the proposed time duration of the simulation, and time step, the time step to be used in the simulation are stored, and the number of simulation steps calculated.

The array of neurons in the simulation is set up (having been freed if necessary) in the NSMutable array NeuronArray. (Note that the maximal number of neurons is set to n_neuron, the number of neurons). The neurons are then set up: the first input_neuron of them (indices starting with 0) are input neurons, and the rest are non-input neurons.

The method applyNeuronParameters in NeuronView is invoked. This starts by setting the default values for the neurons (as determined by Neurons: Load Default Neuron Params) to all the neurons. As matters stand, the values are all 0 if no Default neuron values have been loaded: however, this will (now) give an error message on the log. Next it reads the file (if file there be) set up for non-default neuron parameters. (For file format see above. Lastly, the values are sent to the neurons themselves using the methods setRefPeriod, setTimestep, setPhase, setTonic, setThreshold, setRefPeriod, setDissipation, setSquareDissipation, setMinimumActivation of the Neuron object.

Next the existence of a Synapse File is checked: given that one has been loaded, the adaptivity parameters are set in the network using the adaptparams method of Network. Then the synapses are set up using the setSynapses method of network. This is sent the filename SynapseFile. The method setSynapses reads this file, checks that the synapses connect valid neurons (e.g. no synapses to input neurons), computes the synaptic (really axonic) delay in time steps, and sets up a new synapse of the appropriate type (synapse, dynsynapse, or shuntsynapse). These are then added to the NSMutableArray Synapses. (If Synapses already has

synapses in it, these are removed first.) Next, the synapses are linked into the network: the getPresynNeuron method of each presynaptic Neuron (which returns the presynaptic neuron) has the synapse added using the Neuron method addTarget, and the getPosysynNeuron (which returns the postsynaptic neuron) has the addSynapse method invoked to add this synapse. This essentially doubly links the Neurons and the synapses.

The source of the input to the simulation is then determined: if nothing has been specified, there is no external input. Otherwise, the external input may be specified using Spikes: Load Generated Spikes, Spikes: Set Simple Spike File or Spikes: Set Matlab Spike File.

> *Spikes: Load Generated Spikes* invokes loadSpikesGeneratedEarlier. This brings up a load file dialogue panel, with the file suffix set to outspikes. The file name is stored in ArchiveInputSpikes, and the filename displayed on the main view window. The inputsource is set to 2.
>
> *Spikes: Set Simple Spike File* invokes setSpikeInputFile. This brings up a load file dialogue panel, with the file suffix set to spikesin. The filename is stored in SpikeInputFile, the filename displayed on the main view window. and the inputsource set to 3.
>
> *Spikes: Set Matlab Spike File* invokes setInputSpikesFile. This brings up a load file dialogue panel, with the file suffix set to inspikes. The filename is stored in InputSpikeFile, the filename displayed on the main view window. and the inputsource set to 1. This is intended for use in conjunction with the audio processing system written in MatLab.

The input is then read using the appropriate method from NeuralNetwork (setInputSpikes for inputsource 1, setInputSpikesFromArchive for inputsource 2, and setSpikeInput for input source 3.

Essentially, these all produce the same input spike structure for use in Network. There is an array inspikes (whose elements are struct spike: i.e. neuron time) which is filled up in each of these methods. In addition, number_input_spikes is filled with the number of input spikes.

The spike display times (in the main view) are initialized with the start and end times of the simulation.

Running the network.

The network is run by pressing the Start button in tne main view. This invokes the start method in NeuronView. This in turn invokes the start method in the Network, then once this returns, displays the number of

spikes generated in the main view, and causes the spikes to be displayed using drawRect because it invokes setNeedsDisplay.

The start method in Network:
The primary running of the network is achieved using the method start in Network. This runs the simulation for simsteps (set in applySimParameters) time steps.

For memory management reasons, the number of time steps run at a time is divided up into sections, with a new autorelease pool allocated between them (Not sure if this is really necessary any more). Currently, 2500 time steps are run for each subpool (set in the Network.h file, value of N_SUBPOOL). Essentially, however, the simlation runs time step by time step.

Inside each time step, the current time is calculated, and external inputs which have arrived since the last time step gathered. These are placed in the spikesnow array (which has one element for each input neuron), with a maximal value of 1 for each neuron.

The neurons in the neuron array are then run through, starting with the Input neurons. The runStep method of neuron is called, and this returns a 1 if the neuron fired at this time step. Spikes are stored in the nextspike array (input neuron spikes are optionally stored, but non-input spikes are always stored). The start method eventually returns the total number of spikes.

The runStep method in InputNeuron and Neuron.

In InputNeuron, this method checks whether the neuron has fired, and if so invokes its fire method. The fire method notifies target synapses (if any) that this neuron has fired by invoking the presynFires method of any postsynaptic neurons, using the NSMutableArray PostSynaptic. It then resets the activation to the resetlevel, and returns its output value (which is 0 or 1).

In Neuron, this method starts by checking if the neuron is in its refractory period. If so, it computes the synaptic contributions (both additive and multiplicative) but does nothing with them, and returns the value 0 (not fired). Otherwise

(i) in the absence of noise: the activation is computed from the various inputs (synaptic and otherinput, which comes from the network itself and tonic), then decremented by an amount set by the dissipation and squareddissipation. Shunt synapses are applied (there is an issue about whether they should be scaled to reflect the

time step: currently not). If the activation exceeds the threshold, the fire method is invoked, and the refractory period counter started.

(ii) In the presence of noise, the tonic input and threshold are first recalculated taking the tonic and threshold noise into account. Then the activation is calculated as above. However, activation noise is applied before shunting and comparison with the (noisy) threshold. The firing action is unchanged.

The fire method behaves as the fire method in InputNeuron, and in addition invokes the postsynFires method of any presynaptic neurons.

Notes/recent changes:

11 April 2008: The maximum number of presynaptic neurons is set to 25 currently. This can be altered by changing the #define MAXPRESYNSYNAPSES in neuron.h. The maximum number of postsynaptic synapses is also 25: This can be altered by changing the #define MAXPOSTSYNSYNAPSES in inputneuron.h.

30 April 2008: The spike file (simple spike file) has the format:
<name> <value>*
spikes
<neuron_number> <time> *

where neuron numbers start at 0, and times are in seconds.
The name/value pairs currently must contain *nspikes* and the number of spikes in the file (or, if its less, the number of spikes to be processed).

Additionally, there is now another name/value pair, namely *skipover*. The value following this is subtracted from the neuron_number of all spikes. This allows the simulator to read in spike trains generated by the simulator, and to ignore all the spikes from low-numbered neurons (which are probably input neurons), and to renumber the neurons which are output neurons from 0 upwards.

19 May 2008: Two additional name/value pairs have been added: *timemultiplier* and *ignoreabove*.

ignoreabove is used to cause the simulation to ignore all spikes from neurons whose neuron number (before any skipover value is subtracted) exceeds this value. This is particularly useful when one wants to use only a limited number of the neurons whose input spikes have been supplied.

Timemultiplier is used to allow the times of input spikes to be specified in units other than seconds, For example, the times might be in milliseconds (in which case set *timemultiplier* to 0.001) or in samples at some sample rate (in which case set timemultiplier to 1/sample rate).