Neurosim – Technical Overview

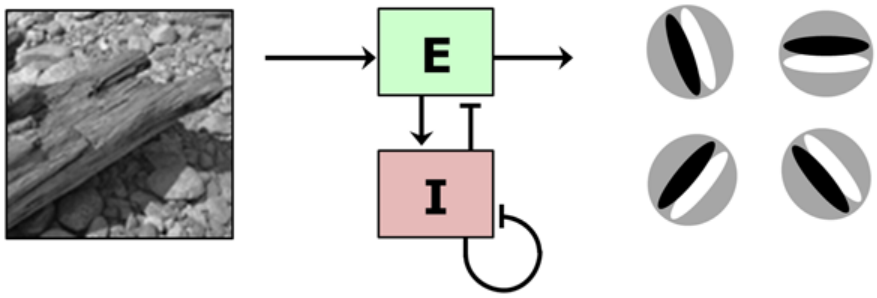
4/2013

Paul King, Redwood Center for Theoretical Neuroscience, UC Berkeley
Contact: paul@pking.org Website: www.pking.org/research/EINet

Contents:

1. Introduction

E-I Net is a spiking neural network simulation with excitatory and inhibitory neurons that can learn a sparse code from input data. When trained on whitened natural images, E-I Net learns a distributed representation of Gabor-like receptive fields as found in the primary visual cortex (area V1).



The results of the E-I Net simulation are presented in the following paper:

> King PD, Zylberberg J, DeWeese MR (2013). **Inhibitory interneurons decorrelate excitatory cells to drive sparse code formation in a spiking model of V1**. *J Neuroscience*.

This sparse code spiking model is based on previous work by Zylberberg *et al.* (2011):

> Zylberberg J, Murphy JT, DeWeese MR (2011) **A sparse coding model with synaptically local plasticity and spiking neurons can account for the diverse shapes of V1 simple cell receptive fields**. *PLoS Computational Biology*.

The MatLab source code and neural network simulator provided here includes all files needed to generate the results of King *et al.* (2013). Included and describe here is the Neurosim spiking neural model simulator which is written in MatLab and can be found in the "`neurosim`" directory of this software distribution.

The "`sample_data`" directory in this distribution contains pregenerated simulation results saved as MatLab "`.mat`" files. This data can be regenerated by running the scripts named "`train1_...`" through "`train4_...`". See the `_README.txt` file for details.

Neurosim is a spiking neural network simulator for modeling neural circuits composed of populations of cells with identical properties (“cell groups”). E-I Net uses two cell groups: an excitatory population named "`V1e`" and an inhibitory population named "`V1i`". There is also an "external" pseudo-Cell-Group named "`input`" that represents the whitened image input to the simulation.

Cell groups are connected to each other with input-side connection descriptors ("input blocks"). Each input block describes the characteristics of the connections from an output cell group to an input cell group, including properties such as the learning rule and learning rate. Each input block contains a weight matrix that is updated during learning.

All cell groups comprising the circuit to be simulated are collected in a network model structure (`NetModel`).

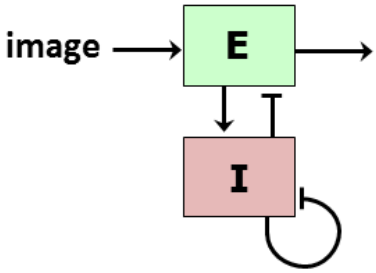
The simulator allows arbitrary circuit structures to be specified as a configuration input to the simulation using hierarchically organized MatLab "`struct`" objects. Model template files with sets of default structure properties can be found in the files with names like `NetModel_InitXxxx.m`.

The function `RunVisionNetworkSimulation()` takes as input a hierarchical set of properties describing the network to be simulated. It constructs the network and runs the simulation using whitened images as training data.

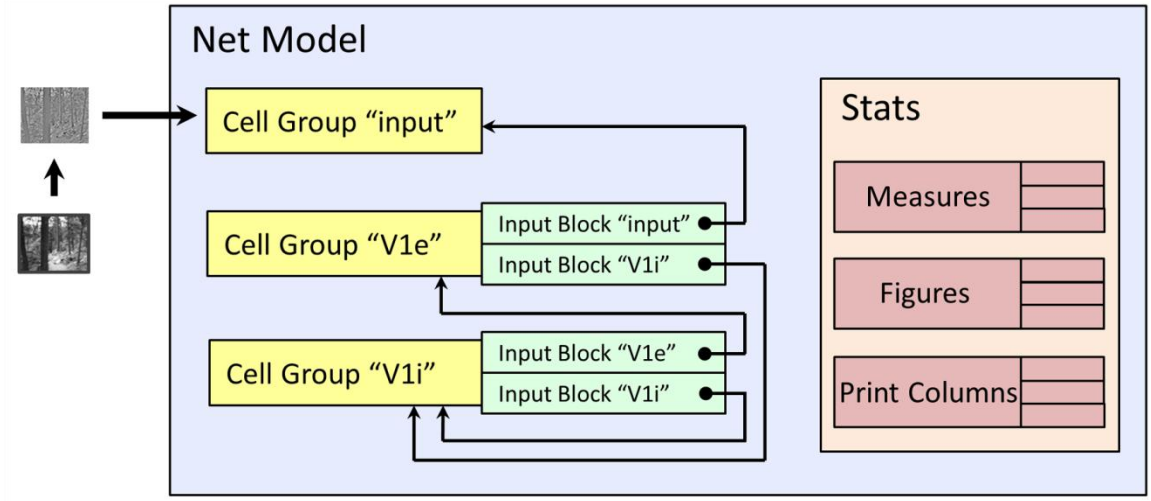
2. Model Configuration

Neurosim models a spiking network composed of “cell groups” – populations of neurons with identical behavior – which are connected to each other via input blocks. An input block is a matrix of connections in which every cell from a source cell group connects to every cell in a destination cell group. Each connection has its own connection weight. Partial connectivity can be simulated by clamping certain connection weights to zero or by an indirect mapping.

The E-I Net model can be conceptually represented as follows:



Within the spiking network simulator, E-I Net is represented this way:

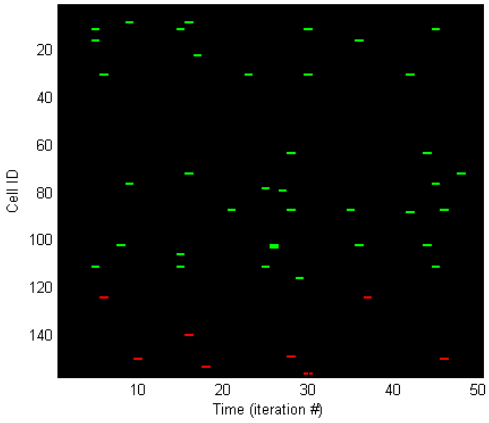


The model is hierarchically organized, with a model composed of cell groups, which is composed of input blocks.

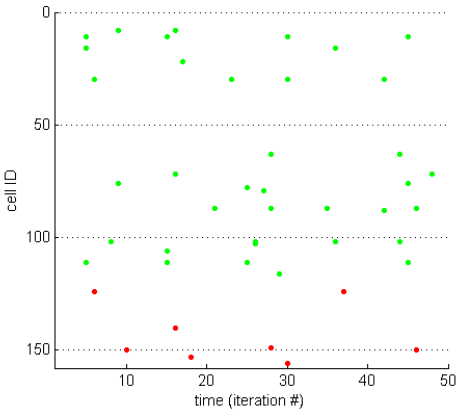
Example Plots

Several real-time network plots are available which will be updated periodically while the simulation is running. 23 different plot types are supported, 6 of which are shown below along with their `plotType` name.

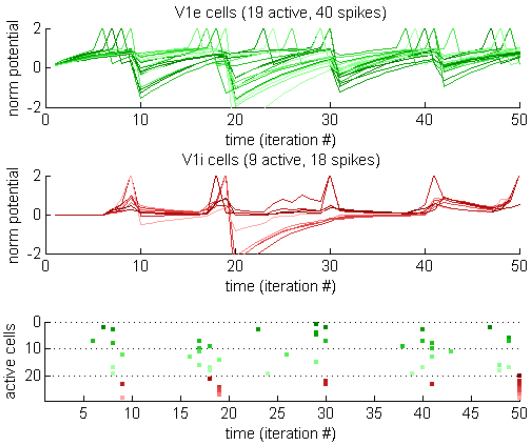
Selected plot types:



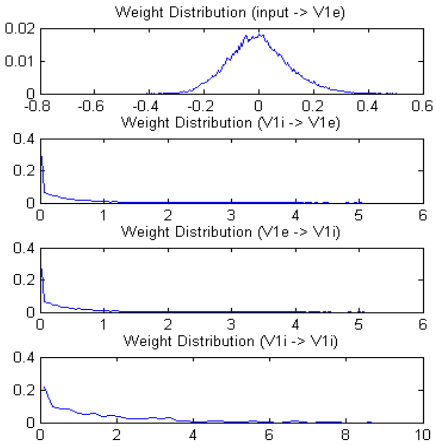
'spikeRasterAllImage'



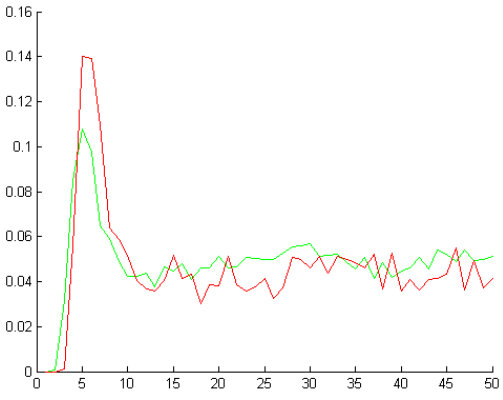
'spikeRasterAll'



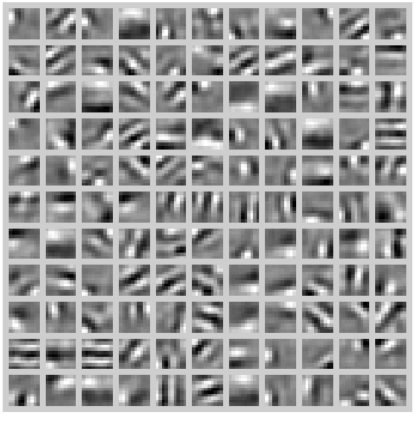
'networkDynamics'



'weightDistributionAll'



'populationActivity'



'STA', 'weightPatches'

Running a Simulation – E-I Net

The following MatLab code shows an example simulation experiment (taken from the file `demo1_EINet.m`):

```
simParams = struct();
simParams.numInputSamples              = 50000;
simParams.model.modelType              = 'V1eV1i';
simParams.model.inputDims              = [8,8];
simParams.model.numSamplesPerBatch     = 100;
simParams.model.numIterationsPerSample = 50;
simParams.model.meanRateLearning       = true;
simParams.model.learningRate           = .5;
simParams.model.cg_V1e.numCells        = 121;
simParams.model.cg_V1i.numCells        = 36;
simParams.model.cg_V1e.in_V1i.learningRate = .7;
simParams.model.autoFigures            = {'STA_E', 'STA_I', 'weightDist', ...
                                          'spikeRaster', 'networkDynamics'};
simParams.model.stats.printStatsEvery  = 2000;
randreset(); model = RunVisionNetworkSimulation(simParams);
```

In the above simulation, a hierarchical set of configuration parameters have been initialized to describe the simulation. Each line is explained in sequence.

```
simParams.numInputSamples         = 50000;
```

Run the simulation for the duration of 50,000 training samples. Each input sample will be an image patch drawn from a random region of a random image in the input image set (`IMAGES.MAT` by default).

```
simParams.model.modelType         = 'V1eV1i';
```

The model substruct describes everything about the model to be simulated. The `modelType` provides the name of a preconfigured set of model parameters to load. The `modelType='V1eV1i'` is "hard-wired" to initialize the E-I Net model default parameters by executing the MATLAB file `NetModel_InitV1eV1i.m`, which contains all the default parameters.

```
simParams.model.inputDims         = [8,8];
```

Run the simulation using 8x8 pixel input image patches.

```
simParams.model.numSamplesPerBatch = 100;
```

On each training batch, run 100 networks in parallel. This is a good number for speeding up the math considerably.

```
simParams.model.numIterationsPerSample = 50;
```

Run each image patch simulation for 20 time step iterations.

```
simParams.model.meanRateLearning  = true;
```

Use mean rate learning. If `meanRateLearning=true`, the learning rules will be applied on the average spike rate across the training sample, as is done in the SAILnet model. If `meanRateLearning=false`, then spike-timing dependent plasticity (STDP) is used in which the learning rate is applied on each time step to the instantaneous average spike rate at that moment in time. `meanRateLearning=true` runs faster.

```
simParams.model.learningRate      = .5;
```

Use a lightly lower global learning rate than the default, which is 1. The true learning rate for a particular connection is the multiplicative combination of the global learning rate, the cell-group-level learning rate, and the input-block-level learning rate.

```
simParams.model.cg_V1e.numCells   = 121;
simParams.model.cg_V1i.numCells   = 36;
```

Create a network with 121 E cells (`V1e`) and 36 I cells (`V1i`).

```
simParams.model.cg_V1e.in_V1i.learningRate = .7;
```

Parameters can also be set at the level of the input block. This example parameter sets the learning rate for the `'V1i'` input block within the `'V1e'` cell group. This is the input block representing the incoming connections from the I cells (the `'V1i'` cell group). Any parameter not specifically declared reverts to a default, and numerous defaults are defined in NetModel_InitV1eV1i.m.

```
simParams.model.autoFigures       = {'STA_E', 'STA_I', 'weightDist', ...
                                     'spikeRaster', 'networkDynamics'};
```

The `V1eV1i` model has several predefined real-time figures. These figures will be updated periodically as the learning simulation progresses. The predefined figures indicated here are the spike-triggered average image patch of the E cells (`'STA_E'`) and I cells (`'STA_I'`), the statistical distributions of all connection weights for all input blocks (`'weightDist'`), the spike raster plot using a fast-drawing line style (`'spikeRaster'`), and a plot showing the network dynamics of the active (spiking) cells (`'networkDynamics'`). These last two plots are specific to only a single simulation run, and by default, the first run of the most recent batch is displayed.

```
simParams.model.stats.printStatsEvery  = 2000;
```

A line of statistics will be printed summarizing the state of the network every so often. This line says to print the statistics after every 2,000 input samples (or 20 batches, because the batch size is 100). This interval is also used for updating the figures defined above.

```
randreset(); model = RunVisionNetworkSimulation(simParams);
```

This last line runs the actual simulation. The first statement resets the random number generator, which is optional but convenient for getting repeatable results. The second statement runs the simulation on a model defined with the previously discussed parameters. After the simulation completes, the state of the model is returned in model.

```
model = RunVisionNetworkSimulation({'numInputSamples',10000}, model);
```

This will run the simulation for an additional 10,000 samples. `simParams` has been replaced with the line `{'numInputSamples',10000},` which will set just the one parameter `numInputSamples`., and the additional second parameter, `model`, has been passed in to indicate that the simulation should pick up from the current model state. This approach to simulation allows different models to be stopped and started and compared. It is also possible to modify the parameters of the model in mid-simulation this way, for example to change the learning rates or enable or disable a type of cell in the circuit or a connection input block.

Running a Simulation – SAILnet

The following MatLab code runs a simulation of SAILnet (Zylberberg *et al*., 2011):

```
simParams = struct();
simParams.numInputSamples       = 50000;
simParams.model.modelType       = 'V1eV1i';
simParams.model.modelSubtype    = 'SAILnet';
simParams.model.cg_V1e.numCells = 121;
simParams.model.autoFigures     = {'STA_E', 'weightDist', 'networkDynamics'};
simParams.model.stats.printStatsEvery  = 2000;
randreset(); model = RunVisionNetworkSimulation(simParams);
```

This script invokes the `'SAILnet'` subtype of the `'V1eV1i'` model. This subtype (see `NetModel_InitV1eV1i.m` for details) includes the same neuron populations as E-I Net except that the inhibitory neurons have been disabled and the learning rules have been changed to match the SAILnet model.

Inspecting Model Parameters

The parameters actually assigned to a model, as well as its full structure and current simulation state can be inspected by displaying the MatLab model `struct`. Do to this, type "`model`" at the MatLab command line. For example, after running `demo1_EINet` (the first example above), you can type:

```
>> model

model =

                    fpsMax: 0
              learningRate: 0.5000
                lrateScale: 0.0100
           simTimeUnitSize: 0.0100
               simTimeStep: 0.1000
         simMovingAveInput: 0
         simInitPotentials: 'zero'
                simNoReset: 0
        numSamplesPerBatch: 100
    numIterationsPerSample: 50
                 spikeSize: 1
          meanRateLearning: 1
         stdpEnvelopeShape: 'expDecay_continuous'
```

```
             stdpTimeOffset: 0
              precisionHint: 'double'
        outputCellGroupName: 'V1e'
                refNumCells: []
                 cgDefaults: [1x1 struct]
           displayTimeUnits: 'sample'
          debugDisplayEvery: 0
                      stats: [1x1 struct]
                  modelType: 'V1eV1i'
               modelSubtype: 'cm'
                  inputDims: [8 8]
            inputPreprocess: [1x1 struct]
                 inputScale: 0.2000
                  simMethod: 'cellgroupFastBatch'
          learningRateUnits: 'per100Samples'
                  autoStats: {'weights'}
                autoFigures: {'STA_E'  'STA_I'  'weightDist'
'spikeRaster'  'networkDynamics'}
                   cg_input: [1x1 struct]
                     cg_V1e: [1x1 struct]
                     cg_V1i: [1x1 struct]
                 autoAnneal: [1x1 struct]
     numTimeUnitsPerSample: 5
                 initParams: [1x1 struct]
                  cellGroup: {[1x1 struct]  [1x1 struct]  [1x1 struct]}
           displayTimeScale: 50
           debugDisplayWait: 0
          outputCellGroupId: 2
                   snapshot: [1x1 struct]
```

The fully populated model parameters and defaults are displayed. The three cell groups are shown as substructs under cellGroup:

For example, the E cells are cell group #2, here:

```
>> model.cellGroup{2}

ans =

                  name: 'V1e'
              numCells: 121
              cellType: 'excitatory'
            isExternal: 0
          displayColor: [0 1 0]
        defLearningRule: []
         defForceDirect: 1
               location: [121x2 double]
              potential: [121x1 double]
             spikeThresh: [121x1 double]
            dspikeThresh: [121x1 double]
          spikedRecently: [121x1 double]
          causalFeedback: []
             targetCount: [121x1 double]
              inputBlock: [1x3 struct]
              membraneTC: 0.9491
          spikeDecayRate: 2
```

```
           learningRate: 1
         targetSpikeRate: 0.0500
         threshAdaptRate: 1
       updateThreshEvery: 5000
        updateThreshWait: 0
             updateEvery: 5000
              updateWait: 0
            rescaleEvery: 0
             rescaleWait: 0
               spikeRate: [1x1 struct]
```

The I→E connections are the third input block:

```
>> model.cellGroup{2}.inputBlock(3)

ans =

      connectionType: 'inhibitory'
     numSourceInputs: 36
                name: 'V1i'
               srcId: 3
           numInputs: 36
         blockWeight: 1
        learningRule: 'correlation_measuring'
        learningRate: 0.7000
    constrainWeights: 'nonneg'
           clampZero: []
        inputIndices: []
      spikedRecently: [36x1 double]
           spikeRate: [1x1 struct]
      causalFeedback: []
              weight: [121x36 double]
             dweight: [121x36 double]
```

3. Object Types and Parameters

The sections below explain the properties that can be set on each object type.

Cell Group

Cell groups are the basic unit of computation in the model and correspond to groups of similarly-typed neurons in biology. A cell group is a population of cells which function identically and serve a consistent role in a statistically regular circuit.

Inputs to the cell group are organized as a set of input blocks. Each input block is a set of inputs that are treated in a consistent way, for example excitatory, inhibitory, or modulatory. Each input block might correspond to a unique neurotransmitter, to a particular connection type (apical vs. distal), or to a particular connection source (L2/3 cells vs. L5 or L6 cells).

The input blocks are connection points in a canonical circuit which are repeated in a statistically regular fashion across all the cells of the cell group. Each cell will have the same number of inputs of each type as all other cells in the cell group.

Connection types (`connectionType`, also `cellType`) include:

`'excitatory'`	Input spike increases cell potential by weighted `spikeSize`
`'inhibitory'`	Input spike decreases cell potential by weighted `spikeSize`
`'continuous'`	Input value is integrated as a continuous input over simulation time step interval
`'disabled'`	The connection is ignored

Learning rate:

The learning rate that is applied to the connections in a particular input block is the product of the following:

`model.learningRate`	Global learning rate
`model.lrateScale`	Global scale factor for learning (to make learning rates more human readable, and to normalize across simulation time unit changes)
`cellgroup.learningRate`	Cell group specific learning rate
`inputBlock.learningRate`	Input block specific learning rate

Setting any of these to zero will disable learning at that level.

Cell Group initialization parameters (`params`):

`numCells`	Number of computational units (neurons) in this group
`name`	Name of this cell group, used to refer to the cell group when specifying connections.
`cellType`	Cell type (default = `'excitatory'`). This is used to select the appropriate signal processing and weight update rule.

`'excitatory'`	Input has a positive effect on potential
`'inhibitory'`	Input has a negative effect on potential
`'disabled'`	Cell group is ignored in all calculations
`location`	Method for allocating cell location. (default = `'tile2D'`)
`'uniform2D'`	Assign each cell two random coordinates in the range (0,1)
`'tile2D'`	Evenly tile the 2D unit square
`matrix`	If a numeric matrix(`numCells`,2) is provided, use those locations
`defLearningRule`	Default learning rule for outputs from this cell group. Only used during wiring phase. (default = [])
`defForceDirect`	Whether or not to use direct connections by default (only used during wiring phase)
`displayColor`	RGB color for graphs and plots (default = `[1 1 1]` = white)
`membraneTC`	Membrane time constant determining potential decay rate in time units (default = `10`)
`learningRate`	Learning rate to apply to all connection weights in the cell group. Suggested values are between 0 and 1, where `0` = no weight updates and `1` = recommended 'fast' update rate. Spike threshold updates are not affected (see `threshAdaptRate`). (default = `1`).
`initSpikeThresh`	Initial threshold for spiking (default = `1`)
`threshAdaptRate`	Learning rate to apply to spike thresholds; weights are not affected
`targetSpikeRate`	Target spike rate (spikes/interation) for spike rate autoranging. `0` disables autoranging. (default = `0`)
`updateThreshEvery`	How often (# iterations) up update spike thresholds (default = `100`)
`updateEvery`	How often (# iterations) to update weights (default = `100`)
`spikeRate`	Struct with various fields for spike rate tracking
`meanWindow`	Sampling window (# iterations) to use to compute long-term mean rate of spiking for network monitoring and reporting. This value has no effect on network learning behavior. (default = `10000`)
`lmeanWindow`	Sampling window (# iterations) to use to compute long-term mean rate of spiking used by learning rules. Changing this can alter network behavior. (default = `5000`)
`instantWindow`	Sampling window (# iterations) to use for measurement of instantaneous spike rate. (default = `5`)
`delayLine`	Substruct of delay line parameters
`len`	Length of delay (# iterations)

Cell Group internal state variables:

`isExternal`	If the cell group is 'external' then it is a place-holder for copies of values derived from elsewhere

location	Array [numCells,2] of (y,x) positions for each cell (optional). Positions are in unit coordinates in the range (0,1) for a simulated 2D sheet.
potential	Array [numCells,1]. The summed input that is integrated by the neuron over a exponentially receding time window. This value is analogous to the membrane voltage potential of a neuron.
spikeThresh	Threshold for triggering a spike (default = 1, updated dynamically).
spikedRecently	Array [numCells,1]. Set to 1 when a cell spikes during the processing of an iteration. From there, the signal decays exponentially over time. This decaying signal allows target neurons to estimate how recently a source neuron fired, which is needed to calculate the Spike Timing Dependent Plasticity (STDP) response curve.
targetCount	Array [numCells,1] indicating the total number of output targets for each cell. The number of output targets is determined by the model connectivity, which is calculated in NetModel_Init.
inputBlock	Struct array(numInputBlocks) describing the inputs. See ***Input Block*** section below for details.
delayLine	Substruct of delay line parameters:
len	Length of delay (# iterations)
buffer	Matrix(numCells, len) of delay holding slots
updateThreshEvery	how often (# iterations) to update spike thresholds
updateEvery	How often (# iterations) to update weights
spikeRate	Struct with various fields for spike rate tracking
meanWindow	Sampling window (# iterations) to use for long-term mean rate of spiking. See CellGroup_Init.
instantWindow	Sampling window (# iterations) to use for measurement of instantaneous spike rate. See CellGroup_Init.
mean	Array(numCells,1) the long-term measured mean spike rate for each cell (a moving average in spikes/timeUnit)
popMean	The long-term measured mean spike rate of the cell group as a population
instant	Array(numCells,1) the instantaneous spike rate for each cell (a short-term moving average in spikes/iteration)

Input Block

The input block describes a set of connections from one cell group (the “source”) to another. The latter cell group “owns” the input block. The input block primarily contains a matrix of connection weights from all inputs to all outputs.

The initialization parameters to the input block are specified during model creation as a substruct of the cg_xxx parameters struct, for example as:

```
model.cg_xxx.in_yyy.connectionPattern = 'full';
```

The connection wiring between cell groups is performed during model creation according to wiring rules specified in the input block parameters. By default, all inputs are connected to all outputs (`connectionPattern = 'full'`), however various parse connection patterns are also supported. For example with `connectionPattern = 'geolocal'`, the (x,y) coordinate assigned to the cells in the input and output cell groups are used to preferentially connect cells nearby in terms of Euclidean distance. This wiring method attempts to approximate the connection pattern between cell types believed to exist on within cortical sheet.

Learning rules (`learningRule`) include:

`'hebian_oja'`	Oja variant of Hebbian learning
`'correlation_measuring'`	Learning rule used in E-I Net
`'foldiak'`	Decorrelation rule used in SAILnet
`'none'`	No weight updating

Input configuration parameters:

`sourceCellGroup`	The cell group providing the inputs (required)
`numInputs`	Number of inputs per cell (required)
`name`	Name of the input block. Defaults to the name of the source cell group.
`connectionType`	Determines computational effect; Defaults to source `cellType` *e.g.* `'excitatory'`, `'inhibitory'`, ...
`connectionPattern`	The connection pattern to use. (default = `'full'`)
`'full'`	Full connectivity. If `numInputs` is supplied, it must equal `sourceCellGroup.numCells`. No indirection matrix (`inputIndices`) is used for greater speed.
`'fullIndirect'`	Full connectivity, but with the usual indirection matrix (really just for testing).
`'uniform'`	each target cell is connected randomly to a source
`'geolocal'`	Connect probabilistically to the closest cells according to cell location.
`'geolocalClosest'`	Connect to the closest cells according to cell location.
`'none'`	uninitialized; the caller will wire later
`connectionSigma`	When using the `'geolocal'` method, the standard deviation to use when determining connection probability. If null is passed in, a value will be calculated based on the ratio of inputs to source cells.
`noDuplicates`	If true, duplicate connections (connections between the same input/output cell pairs) will be discrarded and resampled. (default = `true`)
`noSelfConnections`	If true, this input block connects a cell group to itself, and connections where the input and output cell are the same will be disallowed. Disallowing self-connections is important for network stability to prevent self-reinforcing feedback loops and infinitely scaling connection weights. For fully-connected input blocks,

	self-connections will be suppressed by clamping the diagonal weights, `weight(i,i)`, to zero. (default = `false`)
`forceDirect`	Force all connections to be direct, with missing connections marked with connection clamping. This is the fastest method for execution as long as the input reduction is not too great (up to 10:1 reduction?).
`learningRule`	Learning rule to use (default = derive from source cell group). Possibilities include `'hebbian_oja'`, `'foldiak'`, and `'correlation_measuring'`. See section above for details.
`learningRate`	Learning rate to apply. The learning rate can be negative to change the sign of the learning rule. This is later multiplied by `cellGroup.learningRate` and `model.learningRate` to arrive at a final value. (default = `.01`)
`blockWeight`	A weighting factor applied to all inputs from this block as a whole. This can be used to simulate duplicate connections or to scale learning rule weight ranges. `0` disables connection input, which can model silent synapses that still learn without effecting cell activity. (default = `1`)
`initWeights`	Method to use for initializing weights (default = `'uniform'`). Weights are normalized to unit vector length input to each cell (unless the weights are initialized to zero). Unconstrained weights will have a zero mean. `'nonneg'` weights will be `>= 0`.
`'uniform'`	Uniform random values (default)
`'gaussian'`	Gausian distribution, or positive half-gaussian if `constrainWeights = 'nonneg'`.
`'zero'`	Initialize all weights to zero.
`<scalar value>`	Weights are initialized to that value
`matrix`	Weight are initialized to the supplied weight matrix
`constrainWeights`	Constraint to apply to weights (default = `'nonneg'`).
`'nonneg'`	Weights cannot go below zero (default)
`'none'`	No constraint is applied to the weights
`clampZero`	Array(`n,1`) or logical array(`numCells`,`numInputs`) indicating which weights, if any, to clamp to zero. (default = `[]`). Clamping weights to zero can be used to simulate partial connectivity. When a cell group is connected to itself, all self-weights (`weight(i,i)` along the diagonal) should often be clamped to zero to prevent problematic weight change behavior.

Input Block internal state variables:

`numSourceInputs`	Number of inputs to the cell group of this type
`numInputs`	Number of inputs to each cell
`clampZero`	Logical array [`numCells`, `numInputs`], or numerical array [`numClamp`,`1`] indicating which weights to clamp to zero. Clamping a weight to zero treats the connection as non-existent, allowing partial connectivity to be simulated in a fully-connected model.

`inputIndices`	Array [`numCells`, `numInputs`] indicating which input source this connection (synapse) receives from. If this is `[]`, full connectivity is assumed and `numSourceInputs` must equal `numInputs`.
`weight`	Array [`numCells`, `numInputs`] containing the connection weights.
`dweight`	Array [`numCells`, `numInputs`] containing accumulated weight changes that have not yet been applied.
`spikedRecently`	Array [`numSourceInputs`,`1`] indicating whether the input neuron spiked recently. `1` if it spiked on the most recent iteration, and something in the range [0,1) if it spiked earlier. Value decays with each iteration.

Net Model

Input configuration parameters:

`refNumCells`	A reference number of cells for each cell group. The actual number of cells for a cell group can then be specified as a fractional percentage of this number. (Used during model initialization only)
`fpsMax`	Maximum allowable iteration rate (frames per second). If set, this will slow down the simulation to a certain maximum speed.
`learningRate`	Learning rate to apply to the model as a whole. If `0`, learning is disabled for the whole model. (default = `1`).
`lrateScale`	Scaling factor to make learning rates more human-readable (default = `.001`).
`simTimeUnitSize`	Size of simulation time unit in seconds (default = `.001` = 1 ms)
`simTimeStep`	Size of discrete simulation time step in simulation time units. The default interpretation is that each simulation time unit is 1 ms and the time step is 1. (default = `1`)
`simMovingAveInput`	Simulate synaptic neurotransmitter accumulation
`simInitPotentials`	How to initialize cell membrane potentials before network simulation. Either `'zero'` or `'random'` (default = `'zero'`)
`simNoReset`	Reset potentials before each training sample? (default = `true`)
`spikeSize`	How much does a single spike contribute to a target cell's potential? (default = `1`)
`meanRateLearning`	Use mean-rate learning instead of spike-timing learning? (default = `false`)
`stdpEnvelopeShape`	Shape of weighting envelope to use for STDP moving average calculation
`'expDecay_continuous'`	Exponential decay, using `y_ave0` to start
`'expDecay'`	Exponential decay (typical moving average)
`'expDecaySymmetric`	Exponential decay but both forward and backward in time
`'gaussian'`	symmetric gaussian envelope

`stdpTimeOffset`	When doing STDP learning (non-mean-rate learning), shift input moving averages to be one time step earlier in time.
`precisionHint`	Floating point precision to favor, either `'single'` or `'double'`. `'single'` can be up to 25% faster with results that are 99.9% the same. (default = `'double'`)
`outputCellGroupName`	Name of cell group that represents the output of the model as a whole, if any. (default = `[]`)
`cgDefaults`	Parameter `struct` containing default values to apply to all cell groups in the model (default = `[]`)
`cg_<name>`	Parameters for cell group with name `<name>`. For a complete list if cell group configuration parameters, see ***Cell Group***, above. These following parameters are a selected subset:
`numCells`	Number of cells in the cell group
`cellType`	The type of cell (*e.g.* `'excitatory'` or `'inhibitory'`)
`refNumInputs`	The target number of inputs to this cell group. If provided , this number is used to calculate the input block input count when fractional ratios are provided. This parameter provides a way to experiment with different numbers of total inputs without changing the synapse type ratios. (optional)
`in_<srcName>`	Describes a set of inputs to this cell group. `<srcName>` is the name of the cell group providing the input. The value of the parameter can simply be a number, in which case it is taken to be the number of inputs to this cell from the source cell group. If the number is in the range `(0,1)` then it is assumed to be a percentage of `cg_xxx.refNumInputs`. For a complete list of input block configuration parameters, see ***Input Block***, above. These following parameters are a selected subset:
`numInputs`	If `in_<srcName>` is a property struct, then `numInputs` contains the value of the number of inputs, identical to the description above.
`connectionType`	Type of connection. Specifying this overrides the default type determined from the input cell type. (default = input cell type)
`connectionPattern`	How to wire the source cells to this group. Options include: `'geolocal'`, `'uniform'`, `'full'`, and `'none'`. See `CellGroup_AddInput` for more details.
`learningRule`	learning rule to use (the default is to derive from source cell group)
`learningRate`	Learning rate to apply. This is later multiplied by `cellGroup.learningRate` and `model.learningRate` to arrive at a final value. (default = `.01`)
`cgDefaults`	Parameter `struct` containing default values to apply to all cell groups
`displayTimeUnits`	Time units to use for interpreting display/print frequencies
`'iteration'`	Iterations
`'sample'`	Learning samples
`'simTimeUnit'`	Simulation time units

`debugDisplayEvery`	Number of `displayTimeUnits` between debug output events. `0` = never. Only used for single-sample linear simulation mode. (default = `0`)
`stats`	Struct of params to configure figures and statistics gathering. (See `NetModel_Stats` for details.)
`measure`	Measurements to take. This is a MatLab `struct`, with each named substruct describing a specific statistical measurement to take. See ***Measurements***, below, for details.
`figure`	Figures to draw. This is a cell array of substructs describing each figure to draw while the simulation is running. See ***Plots and Figures***, below, for details.
`print`	Struct of sub-structs describing console print columns
`printStatsEvery`	How often to print a line of statistics to the console. Units determined by `displayTimeUnits`. `0` means don't print statistics. (default = `1000`)
`updateFiguresEvery`	How often to update display figures. Units determined by `displayTimeUnits`. Value of `[]` will copy value from `printStatsEvery`. `0` means don't show figures. (default = `[]`)
`keepSpikeHistory`	Retain record of spikes from simulation? Useful for debugging and charting, but can take up a lot of memory. History is kept in `model.snapshot.spikeHistory`. (default = `false`)
`keepPotentialHistory`	Retain record of membrane potentials from simulation? History is kept in `model.snapshot.potentialHistory`. (default = `false`)
`numSTASamples`	Number of spike samples to include in the moving average window for STA measure calculation. Can be overridden for each STA measure.

Model internal state variables:

`cellGroup`	Cell array(`numCellGroups`) of cell group structs. See Cell Group “internal state variables”, above.
`inputBlock`	Struct array(`numInputBlocks`) of input block structs. See Input Block “internal state variables”, above.
`snapshot`	Struct containing temporarily saved information from one "batch" of simulations; can be deleted at any time to free up memory.
`inputData`	Matrix(`numInputs`,`numSamplesPerBatch`, `numIterationsPerSample`) of `inputData`.
`spikeHistory`	Cell array(`numCellGroups`) of matrix(`numCells`, `numSamplesPerBatch`, `numIterationsPerSample`) containing retained spike history for later analysis.
`potentialHistory`	Cell array(`numCellGroups`) of matrix(`numCells`, `numSamplesPerBatch`, `numIterationsPerSample`) containing retained cell potentials for later analysis.

Measurements

To help gain insight into the network dynamics, various statistical measurements can be made and collected while the network is running. The measurements collectively are indicated with model initialization properties located in:

`model.stats.measure.<measureName>,`

where `measureName` is the name given to a particular statistical measurements. Some measurements can be displayed in specialized plots, for example the measure `'STA'` is displayed using the plot `'STA'`, and the measure `'timeSTA'` is displayed with the plot `'STA_movie'`.

The type of the measure is specified with the `measureType` property. The following measurement types are supported:

`measureType`	Type of measurement:
`'resError'`	Residual error from linear reconstruction
`'STA'`	Spike-triggered average
`'timeSTA'`	Spike-triggered average for movies
`'correlation'`	RMS correlation of cells (or between two cell groups)
`'spikeRate'`	Moving average spike rate of a population
`'sparseness'`	Sparsity measures (population, lifetime, and activity sparseness)
`'deltaWeight'`	Compute moving average of RMS weight change to test for convergence
`'timelineStats'`	Track metrics over extended time
`'custom'`	User-defined callback function

Each measurement type has its own set of configuration parameters. These are the supported configuration parameters for each `measureType`:

`'resError'`:

`sourceName`	Input block for reconstruction, *e.g.* `'cg_V1e.in_input'`
`numAvgSamples`	How many input samples to include in the moving average window
`normalize`	normalize reconstruction magnitude to unit variance? (default = `true`)
`timeSeries`	Assume time-series input (default = `false`)
`sigmaWeighting`	Gaussian sigma for local (time-series) averaging. Only applies when `timeSeries == true` (default = `10`)
`rmsResErr`	(out) moving average RMS reconstruction error

`'STA'`:

`sourceName`	Cell group to analyze, *e.g.* `'cg_V1e'`
`numAvgSamples`	Number of spike samples to include in the moving average window (default = `model.stats.numSTASamples`)
`STA`	(out) measured spike-triggered average (moving average)

`'timeSTA'`:

`timeInterval`	Array(`1,2`), the time interval of the movie (-/+ # iterations). (default = `[-numInterationsPerSample/2, 0]`)
`numFrames`	Number of movie frames to collect over `timeInterval`
`STA`	(out) measured spike-triggered average movie (moving average)
`'correlation'`:	
`sourceName`	A cell group (`'cg_xxx'`) or a cell array of two cell groups (*e.g.* `{'cg_xxx', 'cg_yyy'}` indicating which cells to analyze for correlation.
`rmsCorrelation`	(out) the measured RMS correlation
`'spikeRate'`:	
`sourceName`	Cell group to analyze, *e.g.* `'cg_V1e'`
`spikeRate`	(out) measured spike rate (moving average)
`'sparseness'`:	
`sourceName`	Cell group to analyze, *e.g.* `'cg_V1e'`
`numAvgSamples`	How many test samples to include in the moving average windows
`timeSparseness`	(out) lifetime sparseness (moving average)
`popSparseness`	(out) population sparseness (moving average)
`activitySparseness`	(out) activity sparseness (moving average)
`'deltaWeight'`:	
`deltaInterval`	The interval in time units for calculating `dW` (default = `1000`)
`windowSize`	The moving average window for RMS calculation in time units (default = `deltaInterval`)
`ib(i)`	Information collected on input block # `i` (`i` is arbitrary)
`name`	A human-readable name of this input block, *e.g.* `'V1e->V1i'`
`cgId`	Cell group index of this input block
`ibId`	Input block index of this input block within its cell group
`srcId`	Cell group id of the input source
`dW`	The moving average RMS weight change (units = `dW / timeUnit`)
`cg(i)`	information collected on cell group # `i` (`i` is arbitrary)
`name`	A human-readable name of this cell group, *e.g.* `'V1e'`
`cgId`	Cell group id of this input block
`dThresh`	The moving average RMS threshold change (units = `dThresh / timeUnit`)
`dW`	The moving average RMS weight change across all weight sets (units = `dW / timeUnit`)
`ddW`	The derivative of `dW`
`dThresh`	The moving average RMS spike threshold change across all cell groups (units = `dThresh / timeUnit`)
`ddThresh`	The derivative of `dThresh`

`'timelineStats'`:	
`metricExpr`	The metrics to track (model-specific MatLab expressions)
`historySize`	Number of historical samples to collect before recycling (default = `1000`).
`'custom'`:	
`measureFn`	Function pointer of form `m = fn(m, model, spikeHistory)` of the measure to calculate.

Plots and Figures

To view what the network is going, various plots and figures are supported which will be updated periodically while the network simulation is running. These plots can also be made interactively from the MatLab command console.

To specify a figure that should be drawn periodically, a figure descriptor can be provided in the following cell array:

```
model.stats.figure{}
```

Figured can also be plotted interactively by calling:

```
NetModel_Plot(model, figureParams)
```

For an example of some of the different types of plots and their `plotType` name, see ***Example Plots*** at the top of this document.

Each figure `struct` may contain the following subfields:

`figureNum`	Figure number to use. (default, assigned automatically starting at 1001)
`title`	Title to use to label the figure
`sourceName`	A name identifying the signal source for the chart. Examples: `'cg_V1e'` or `'cg_V1e.in_V1i'`
`plotType`	Type of plot to draw in this figure (see `plotType` below)
`'STA'`	Spike-triggered average
`'STA_movie'`	Spike-triggered movie
`'inputImagePatches'`	Batch of input training samples (either still image patches or image patch movie clips)
`'weightPatches'`	Grid of 2D image patch basis functions
`'weightMatrix'`	Image showing 2D weight strengths, with scale
`'weightDistribution'`	Histogram showing distribution of weight values
`'weightDistributionAll'`	Histogram showing weight distributions for all input blocks
`'weightCorrelation'`	Scatter plot comparing recurrent A→B and B→A weights
`'spikeRateHistogram'`	Histogram of spike rates across training samples or spike rates across cells
`'populationActivity'`	Line plot of mean spike rate over time, all cell groups
`'connectionDensities'`	Shows where connections occur geospatially

`'STAWRatioHistogram'`	Bar chart showing distribution of `STA/weight` ratio
`'rStdHistogram'`	Histogram of reconstructed sample standard deviations
`'spikeRaster'`	Raster plot of spikes over time
`'spikeRasterAll'`	Raster plot of spikes over time for all cell groups
`'spikeRasterAllImage'`	Same as `'spikeRasterAll'` but with faster image drawing
`'spikeRasterAllMulti'`	A grid of multiple raster plots
`'potential'`	Timeline plot of the evolving potentials of active cells
`'networkDynamics'`	Plot cell potentials and spike raster for whole network
`'timelineStats'`	Long-term timeline ticker of tracked metrics (*e.g.* `spikeRate`, `threshold`, ...)
`'cellGroupVars'`	Plot cell group activity metrics
`'custom'`	A user-defined figure drawn via callback function
`'composite'`	A figure composed of other figures

Special fields for particular plot types (`plotType` in quotes):

`'weightPatches'`:

`maxDisplay`	Max number of patches to display.
`combinePlusMinus`	Subtract second half of weights from first to recreate receptive fields in factored plus-minus scenario

`'STA'`:

`sourceName`	Cells to use for spike trigger (*e.g.* `'cg_V1e'`)
`measureName`	Name of an STA measure to use as source for plot (use only One of `sourceName` or `measureName`)

`'STA_movie'`:

`numFrames`	How many frames to display in the movie (default = `5`)
`playDuration`	Duration of clip playback in seconds (default = `1`)
`minImageSize`	Minimum image size so it isn't too small to see (default = `300`)

`'weightDistribution'`:

`sourceName`	Which weights to display (*e.g.* `'cg_V1e.in_input'`)
`useLogScale`	If true, use a log scale on the X (weight size) axis
`minWeight`	If provided, discard all weights with magnitude smaller than `minWeight`
`numBins`	If provided, overrides the auto-calculated number of bins
`plotStyle`	either `'line'` or `'bar'`. (default = `'line'`)

`'weightDistributionAll'`:

[same properties as `'weightDistribution'`]

`'weightCorrelation'`

`sourceName`	Two bi-directed input blocks, *e.g.* `{'cg_V1i.in_V1e', 'cg_V1e.in_V1i'}`

`'spikeRateHistogram'`:

dimension	Which spike rate measurements to use, either 'perSample' or 'perCell'.
numBins	Number of bins to use when drawing histogram (optional). If numBins = 'tabulate', then perform discrete value tabulation.
'STAWRatioHistogram':	
sourceName	Which weights to display (*e.g.* 'cg_V1e.in_input')
measureName	which STA measure to use (optional)
'spikeRasterAll':	
bgColor	Background color for MatLab scatter plot (default = [0 0 0] = black)
markerSize	Area size to use for spike markers (default = 11)
'spikeRasterAllImage':	
networkId	Which network (sample) to display when for multi-sample simulations (default = 1)
'potential':	
sourceName	Which cell group to display (*e.g.* 'cg_V1e')
networkId	Which network / sample to plot (default = 1)
normalize	Rescale each cell's membrane potential to be relative to its spike threshold, so that the cell spikes at potential >= 1. (default = true)
maxLines	How many lines to draw (default = 40)
colormap	Colormap to use, from 'help colormap'. 'Lines' and 'Winter' are good. Can also be an explicit RGB table. 'cellColor' will use cell-by-cell colors in the cellGroup.cellColor field. 'colorRamp' will generate ramp based on the cell group's displayColor. absColorRamp does the same thing, but always assigns a given cell the same color. (default = 'Lines')
cellIds	Which cells to plot and in what order (overrides automatic ranking based on cell activity)
'networkDynamics':	
networkId	Which network / sample to plot (default = 1)
'timelineStats':	
metricExpr	A cell array of MatLab expressions to track
colormap	Matrix(N,3) of RGB colors to use for lines. (default = colormap('Lines'))
legendText	Cell array of labels for each line plotted
legendLocation	Where to place the legend (default = 'SouthWest').
'cellGroupVars':	
varNames	Cell array(N) of variables to plot. The first variable becomes the dependent variable on X axis, and all remaining variables are plotted on the Y axis against the X variable. (required)
showPDF	Draw probability distribution? (default = false)

`normalize`	Normalize `y` values to the mean (default = `false` unless multiple lines are drawn)
`colormap`	Colormap to use, which is a matrix(`N`,`3`) of RGB values. `[]` indicates default to `colormap('Lines')`. (default = `[]`)
`legendText`	Cell array of names for each variable (optional)
`lineWidth`	Line with to use for line plotting (default = `1.5`)
`fontSize`	Font size for text (default = `13`)
`'custom'`:	
`figureFn`	Function pointer of form `fig = fn(fig, model, spikeHistory)` to plot into a drawing context that has already been established.
`'composite'`:	
`subFigure{}`	Subfigures (only one level of nesting is allowed)
`layoutDims`	dimensions [`rows`, `cols`] for plot arrangement (optional)

Printing

To monitor network dynamics and progress during learning, a line of statistics can be printed every so often to the console. The values on the line to be printed are specified as fields of the structure:

```
model.stats.print.<printName>
```

Each `<printname>` substruct must have exactly one of the following subfields:

`measureName`	The name of a measure to use (uses the main metric)
`matlabExpr`	A string containing a MatLab expression to evaluate
`builtin`	A string naming a built-in field to evaluate
`'sampleCount'`	Total number of samples that have been used for training

4. Selected Functions

This section describes the API to some of the high-level functions of the neurosim model.

RunVisionNetworkSimulation

Build and train a spiking circuit model that works on natural images.

Usage:

```
model = RunVisionNetworkSimulation(params)
model = RunVisionNetworkSimulation(params, model)
```

Inputs:

`params`	Configuration parameters:
`numInputSamples`	Number of image patch samples to train on (default = 10000)
`printSummary`	Print column headers and time elapsed summary (default = `true`)
`model`	Parameters for constructing model. Required if generating a new model. Ignored if an already-constructed model is provided as a second parameter.
`model`	An already-constructed model to use. Providing this causes `params.model` to be ignored. (optional)

NetModel_Figure

Draw a figure based on model state. See ***Plots and Figures***, above, for details.

Inputs:

`model`	The model (read-only)
`figParams`	The figure specification parameters. See ***Plots and Figures***, above, for a detailed list of figure properties by plot type. Figure properties can include:
`figureNum`	Figure number to use (optional)
`title`	Figure title (optional)

Usage:

```
NetModel_Figure(model, figParams)
NetModel_Figure(model, plotType, param1, value1, param2, value2,
...)
```

NetModel_InitV1eV1i

Initialize an E-I Net network model with excitatory (E) and inhibitory (I) populations

Initialize a basic V1 network circuit model that uses two cell groups:

- `V1e` – Excitatory cells with input from image, V1i, V1e
  - V1e→V1e connections are disabled by default, except in SAILnet mode)
- `V1i` – Inhibitory cells with input from image, V1i, V1e
  - input→V1i connections are disabled by default

The connections from V1e→V1e are normally disabled. When enabled, these are inhibitory connections to match the lateral inhibition model of SAILnet.

A representative subset of parameters are shown here. Please see the MatLab source file `NetModel_InitV1eV1i.m` for a complete list of supported parameters and their defaults. The defaults vary by model subtype.

Usage:

```
Model = NetModel_InitV1eV1i(params)
```

Input configuration parameters:

`modelSubtype`	Which variation of the E-I network is desired:
`'jneuroscience'`	Model used in the *J. Neuroscience* paper (CM rule, STDP)
`'SAILnet'`	SAILnet: Only one neuron layer, F rule
`'cm'`	Use the `correlation_measuring` rule for all connections other than input→E (default)
`'ho_fb'`	Use the HO learning rule for E→I connections and the `foldiak_bounded_exp` rule for inhibitory connections.
`learningRate`	Learning rate to apply to the model as a whole. If 0, learning is disabled for the whole model. (default = 1).
`stdpEnvelopeShape`	How to average spikes for input to learning; ignored if `meanRateLearning` = true. (default = `'expDecay_continuous'`)
`'expDecay_continuous'`	Use exponential moving average
`'expDecay_continuous_all'`	Use exponential moving average, including for static inputs
`'gaussian'`	Use a temporal gaussian weighting
`cgDefaults`	Default parameters for all cell groups (see ***Cell Group*** for all available parameters)
`cg_V1e`	Cell group parameters for the excitatory (E) cells
`numCells`	Number of E cells to use in the circuit model
`initSpikeThresh`	Value to initialize E spike thresholds to
`threshAdaptRate`	Learning rate for E cell spike thresholds
`in_input`	Parameters for the image→E connections (see ***Input Block***)
`in_V1i`	Parameters for the I→E connections (see ***Input Block*** for full list)
`learningRate`	Learning rate for I→E connections

`in_V1e`	Parameters for the E→E connections, SAILnet model only (see ***Input Block*** for full list)
`cg_V1i`	Cell group parameters for the inhibitory (I) cells
`numCells`	Number of I cells to use in the circuit model
`initSpikeThresh`	Value to initialize I spike thresholds to
`threshAdaptRate`	Learning rate for I cell spike thresholds
`in_V1e`	Parameters for the E→I connections (see ***Input Block*** for full list)
`learningRate`	Learning rate for E→I connections
`in_V1i`	Parameters for the I→I connections (see ***Input Block*** for full list)
`learningRate`	Learning rate for I→I connections

[see ***Model***, ***CellGroup***, and ***Input Block*** sections, above, for more model parameters]

Output:

`model`	The initialized network model

NetModel_InitEINet

Initialize an E-I Net network model with excitatory (E) and inhibitory (I) populations. This is a simplified model from V1eV1i above. The cell group names are "`E`" and "`I`" instead of "`V1e`" and "`V1i`". Otherwise all the model parameters from `NetModel_InitV1eV1i` apply here also.

NetModel_Init

Initialize a spiking network simulation model.

Self-connections:

If a cell group is connected to itself, the input block parameter `'noSelfConnections'` will be set to true by default. For fully connected networks, the diagonal weights, `weight(i,i)`, will be clamped to zero.

Usage:

```
model = NetModel_Init(params)
```

NetModel_UpdateFast

Execute many time steps of a spiking network simulation. For increased learning speed, several networks can be simulated in parallel but with different inputs. This allows the

optimized matrix math routines to execute the highest floating point operations per second.

This optimized routine performs the same function as `NetModel_Update` but up to 40 times faster. It achieves additional speed in the following way:

- Performs the full network simulation in local variables
- Performs many network iterations in a single call (optional)
- Simulates multiple networks simultaneously in parallel (optional)
- Can use average firing rates rather than spike-timing plasticity (optional)
- Calculates weight updates in batch using matrix multiply

The following constants define the size of the simulation:

`numNetworks`	How many identical networks (but with different inputs and internal state) to simulate in parallel. When 100 input samples are processed simultaneously, 100 clones of the full network are created and simulated. This allows MatLab to do substantial (up to 40x) optimizations with matrix math and multi-core parallel computation.
`numIterations`	How many simulation time steps to iterate through

Each model simulated has the following basic structure:

`numCellGroups`	The number of separate neuron populations in the model. Each neuron population has its own behavior and connectivity parameters. For each `CellGroup`:
`numCells`	The number of identical neurons in that cell group
`numInputBlocks`	How many input types (from other cell groups) are receive. For each `InputBlock`:
`numInputs`	How many inputs of that type does each cell receive
`weight`	matrix of connection weights from all input cells to all output cells.
`dweight`	Accumulated (but not applied) weight matrix changes

If the `initialState` input parameter is provided, then the network state contained in `initialState` will be used instead of the state contained in the model's cell groups. Each state field (`u`, `y`, `y_ave`) can optionally contain a matrix of multiple columns, in which case each column represents a separate parallel network to be simulated in parallel and in batch for faster performance. Any variables or cell array values that are `null` or missing will be assumed to be initialized to zero.

Usage:

```
Model = NetModel_UpdateFast(model, numIterations)
Model = NetModel_UpdateFast(model, numIterations, initialState)
[model, finalState] = NetModel_UpdateFast(model, numIterations,
initialState)
```

Inputs:

`model`	The simulation model state
`numIterations`	The number of simulation iterations to execute

`initialState`	Struct containing the initial cell group state for batch processing. Any variables or values missing will be filled in with zeros. If the `initialState` struct is not provided, the state in the cell groups will be used instead. (optional)
`u`	Cell array(`numCellGroups,1`) of matrix(`numCells`, `numNetworks`) representing initial cell potentials. (optional)
`y`	cell array(`numCellGroups,1`) of Matrix(`numCells`, `numNetworks`) representing the last spike output of the cells. Alternatively, `y` can be a matrix(`numCells`, `numNetworks`, `numIterations`), in which case y represents the input arriving from an external cell group to use for each iteration. (optional)
`y_ave`	Cell array(`numCellGroups,1`) of matrix(`numCells`, `numNetworks`) representing the most recent spike rate moving average output of the cells. (optional)

Output:

`model`	The updated simulation model state
`finalState`	The final state of the network (optional)
`u`	Matrix(`numCells`,`numNetworks`): ending cell potentials
`y`	Matrix(`numCells`,`numNetworks`): last spike status
`y_ave`	Matrix(`numCells`,`numNetworks`): last iteration of running average
`y_history`	Matrix(`numCells`, `numNetworks`, `numIterations`): spike history for all iterations and all networks