SpikeNet Documentation

Yifan Gu
Complex System Group, School of Physics, USYD
yigu8115@gmail.com

June 14, 2016

1 Overview

SpikeNet is a software developed for simulating spiking neuronal networks, of which the design provides the following four features.

Configurability SpikeNet supports any user-defined structure of synaptic connectivity topologies, coupling strengths and conduction delays. It can be easily extended by developers to support any variations of integrate-and-fire neuron and synapse models. For the models that are currently available, see Neuron and synapse models.

Performance Simulation of spiking neuronal network quickly becomes computationally intensive if the number of neurons N in the network exceeds a few thousand. To achieve superior performance, various measures have been taken at both algorithmic and implementation level. Notably, at the algorithmic level, kinetic models are exclusively chosen for their algorithmic efficiency (e.g., [2, 3, 5, 6]). Furthermore, for such kinetic models, computational cost of the simulation can be dramatically reduced with the commonly used mathematical abstraction that synapses can be simplified into a few groups, within which their dynamics (or more specifically, the time constants) are identical [1]. At the implementation level, C++, a programming language renowned for its high-performance computing, is used.

User-friendly interface In SpikeNet, although, C++ is used for heavy-duty computation, its user-interface is written in a high-level programming language (Matlab) for user-friendliness. This means SpikeNet does not require non-developer users to be familiar with C++. In practise, the typical work-flow of SpikeNet user is as following.

1. In Matlab, use SpikeNet functions to generate the input files (plain text) that define the simulation case.
2. Evoke the SpikeNet C++ simulator, which reads the input files, runs the simulation and generates the output files.

3. In Matlab, use SpikeNet functions to parse the output files (plain text) into Matlab data for post-processing.

Please see Matlab user interface for more detailed descriptions of the workflow. Developer users will be interested to know that the data format of the aforementioned input and output files follows pre-defined protocols (see Output protocols and Input protocols). This design essentially makes the SpikeNet C++ simulator a stand-alone software. Based on these protocols, the C++ simulator can easily interface with any other high-level programming languages if Matlab is not prefered.

Scalability The design of the SpikeNet C++ simulator readily supports parallel computing used Message Passing Interface (MPI). The simulator has two central C++ classes, representing populations of neurons and synaptic connectivities among these populations, respectively. For example, for a recurrent network consisting of an excitatory neuron population (E) and an inhibitory one (I), two neuron population objects will be created together with up to four synaptic connectivity objects including $E \leftarrow E$, $I \leftarrow E$, $E \leftarrow I$ and $I \leftarrow I$. These six objects can be created and simulated in parallel and the amount of message passing among them at each simulation time step is minimal.

2 Tech Stack

- C++11 standard is required (GCC 4.2.1 or later; Intel C++ 12.0 or later).
- Matlab is optional but highly recommended.
- Portable Batch System (PBS) is optional but highly recommended.
- Message Passing Interface (MPI) implementation is optional.

3 Quickstart

Following are the steps to set up the SpikeNet C++ simulator.

- Ask for read permission from one of the contributors with admin rights.
- Make a new directory: `mkdir tmp; cd tmp`
- Clone SpikeNet: `git clone git@github.com:BrainDynamicsUSYD/SpikeNet.git`
- Go into the directory: `cd SpikeNet`
- Build the C++ simulator: `make; make clean; cd ..; ls`

Now you should see the "simulator" in the current directory, with which you can run simulations by creating input files according to Input protocols. However, it will be much easier to work with the Matlab user interface. If you do not have access to Matlab, try to contact the contributors to request interfaces with other high-level programming languages (Python for example). Following are the steps to use the Matlab user interface.

- Make a new directory for storing data: `mkdir tmp_data`
- Start Matlab: `matlab -nodisplay`
- Set up the environment for Matlab: `cd SpikeNet; addpath(genpath(cd));`
- Generate the example input files: `cd ../tmp_data; main_demo;`
- Quit Matlab: `quit`
- Run the simulator with the input files: `cd tmp_data; ../simulator *ygin;`
- Start Matlab: `cd ..; matlab -nodisplay`
- Set up the environment for Matlab: `cd SpikeNet; addpath(genpath(cd));`
- Parse the output files, run some basic post-processing and visualization: `cd ../tmp_data; PostProcessYG()`
- Load the simulation result: `d = dir('*RYG.mat'); R = load(d(1).name)` (You may need to correct the single quotes in Matlab if you are directly copying the code from here.)

For those who have access to a high-performance computing cluster with PBS, SpikeNet also provides bash script that fully automates the above Matlab → C++ → Matlab workflow for PBS job array submission.

- Go to the tmp directory
- Make a copy of the script: `cp SpikeNet/*sh.bak all_in_one.sh`
- Change it to executable: `chmod +x all_in_one.sh`
- Edit the following variables in the bash script accordingly:
  - MATLAB_PRE_PROCESS_FUNC='your_function'
  - MATLAB_POST_PROCESS_FUNC='your_function'
  - MATLAB_SOURCE_PATH='SpikeNet'
- Make a directory for PBS output: `mkdir PBSout`
- Submit the job: `qsub -t 1-X -q queue_name all_in_one.sh`

For MPI jobs with SpikeNet, please contact Yifan Gu for more technical details.

Here are some basic tips on git.

- To get the most recent version of SpikeNet: `git pull origin master`
- Avoid directly modifying any existing files in SpikeNet unless you are a developer. Create your own files.

4 Matlab user interface

See main_demo.m for a complete list of available case-building matlab functions and their usages.

(More need to be added on the available post-processing Matlab functions. SpikeNet has a variety of statistical analysis and visualization tools for spiking networks.)

5 Neuron and synapse models

The default neuron model in SpikeNet is leaky integrate-and-fire neuron. The membrane potential V_i^α of the i-th neuron ($i = 1, \cdots, N_\alpha$) from population α evolves according to

$$C_m \frac{dV_i^\alpha}{dt} = -g_L(V_i^\alpha - V_L) + I_{i,syn}^\alpha(t) + I_{i,app}^\alpha(t), \text{ if } V_i^\alpha < V_{th}. \tag{5.1}$$

When neurons reach the threshold V_{th}, a spike is emitted and they are reset to V_{rt} for an absolute refractory period τ_{ref}. The spike times t_i^α are recorded.

The default synapse model in SpikeNet is conductance-based. The synaptic currents received by a neuron are given by

$$I_{i,syn}^\alpha(t) = \sum_{\beta=1}^{P} I_i^{\alpha\beta}(t) \tag{5.2}$$

$$= \sum_{\beta=1}^{P} [-g_i^{\alpha\beta}(V_i^\alpha - V_{rev}^\beta)] \tag{5.3}$$

$$= \sum_{\beta=1}^{P} \{-[\sum_{j=1}^{N_\beta} a_{ij}^{\alpha\beta} g_{ij}^{\alpha\beta} s_{ij}^{\alpha\beta}(t)](V_i^\alpha - V_{rev}^\beta)\} \tag{5.4}$$

where V_{rev}^β is the reversal potential of the corresponding current $I_i^{\alpha\beta}$ induced by pre-synaptic population β. $a_{ij}^{\alpha\beta}$ is a binary variable which determines the existence of synapse from the j-th neuron in population β to the i-th neuron

in population α, while $g_{ij}^{\alpha\beta}$ reflects the (maximal) strength of the synaptic conductance. The gating variable $s_{ij}^{\alpha\beta}(t)$ models the instananeous value of synaptic conductance in terms of the fraction of open channels, described by

$$\frac{ds_{ij}^{\alpha\beta}}{dt} = -\frac{s_{ij}^{\alpha\beta}}{\tau_d^{\beta}} + \sum_{t_j^{\beta}} h^{\beta}(t - t_j^{\beta} - d_{ij}^{\alpha\beta})(1 - s_{ij}^{\alpha\beta}) \tag{5.5}$$

where h models the concentration time-course of the channel-opening neurotransmitters, arrived with a conduction delay $d_{ij}^{\alpha\beta}$ after the pre-synaptic spike time t_j^{β}. The $(1 - s_{ij}^{\alpha\beta})$ term introduces saturation effect. Following the simplification in [1], a rectangular pulse with unitary area is used for h

$$h(t) = \begin{cases} 1/\tau_r, & \text{if } 0 \le t \le \tau_r \\ 0, & \text{otherwise} \end{cases} \tag{5.6}$$

All numerical values are in consistant units unless mentioned otherwise (ms for time, mV for voltage, nA for current, nF for capacitance and μS for conductance). Default numerical integration is performed using Euler method and the suggested time-step is 0.1 ms [4].

6 Input protocols

The input files are plain text files. The default input filename is "filename.ygin". It is advised that synapse definitions always be given in a separate file. The default synapse definition filename is "filename.ygin_syn". Non-default synapse definition file path and name can be specified by "SYNF001" protocol. Following is the complete list of the input data protocols. Note that the protocols use three special characters, including ">" for starting a new protocol, "#" for commenting and "," for delimiting.

```
> INIT001 # number of neurons in each population
        N1, N2, ...,

> INIT002 # time step length and total number of steps
        dt (ms), step_tot,

> INIT003 # random initial distributions for V (deprecated)
        p_fire_pop1 (range (−∞,1]), ..., p_fire_popN,

> INIT004 # external Gaussian currents
        pop_ind,
        mean_1 (nA), mean_2, ...,  mean_N, (for each neuron)
        std_1 (nA), std_2, ...,  std_N,

> INIT005 # external Poissonian spikes
        pop_ind, type_ext, K_ext (μS), Num_ext, ia, ib,
```

```
        rate_1 (Hz), rate_2, ..., rate_step_tot

> INIT006 # chemical connection definition
        type, pop_ind_pre, pop_ind_post,
        I (row vector),
        J (row vector),
        K (row vector),
        D (row vector),

> INIT007 # set perturbation (remove one spike)
        pop_ind, step_perturb,

> INIT008 # add short-term depression
        pop_ind_pre, pop_ind_post, STD_on_step,

> INIT009 # add inhibitory STDP
        pop_ind_pre, pop_ind_post, STDP_on_step,

> INIT010 # add spike-frequency adaptation
        pop_ind,

> INIT011 # random initial conditions for V and firing probabilities
        r_V0_pop1 (range (0,1]), ...,r_V0_popN,
        p_fire_pop1 (range (0,1]), ..., p_fire_popN,

> SYNF001 # non-default synapse definition file name
        path/to/file_name (no comma!)

> KILL001 # runaway killer setting
        pop_ind, min_ms, runaway_Hz, Hz_ms,

> PARA001 # non-default neuron population parameter
        pop_ind, number_of_parameters,
        parameter_name1, value1,
        parameter_name2, value2,
        ...

> PARA002 # non-default synapse parameter
        number_of_parameters,
        parameter_name1, value1,
        parameter_name2, value2,
        ...

> SAMP001 # neuron data sampling
        pop_ind,
        data_type (logical vector),
        ind1, ind2, ..., indX, (sample neuron indices)
        1, 1, 0, 0, ..., (1-by-step_tot logical values)
        # Note that data_type specifies sample data types
        # and it must correspond to
```

```
        # [V,I_leak,I_AMPA,I_GABA,I_NMDA,I_GJ,I_ext]

> SAMP002 # synapse data sampling
        pop_ind_pre, pop_ind_post, syn_type,
        ind1, ind2, ..., indX, (post-synaptic sample neuron indices)
        1, 1, 0, 0, ..., (1-by-step_tot logical values)

> SAMP003 # neuron population statistical data record
        pop_ind,

> SAMP004 # synaptic currents  statistical data record
        pop_ind_pre, pop_ind_post, syn_type,

> SAMP005 # local field potential data record
        pop_ind,
        1, 0, 0, 1, ..., (1-by-N logical values)
```

7 Output protocols

The output files are plain text files. The default output filename is "filename-X.ygout", where X is a time-stamp that uniquely marks the output file so that multiple simulations can be run for the same input files. For data completeness, the corresponding input file will be attached to the output file. Following is the complete list of the output data protocols.

```
> KILL002 # step at which runaway activity is killed
        step_killed,

> POPD001 # spike history of neuron population
        pop_ind,
        spike_neuron_ind (row vector),
        num_spikes_t (1-by-step_tot row vector),
        num_ref_t (1-by-step_tot row vector),

> POPD002 # neuron parameters in the population
        pop_ind, number_of_parameters,
        parameter_name1, value1,
        parameter_name2, value2,
        ...

> POPD003 # neuron population statistical data
        pop_ind,
        V_mean_t1, V_mean_t2, ... (1-by-step_tot)
        V_std_t1, V_std_t2, ... (1-by-step_tot)
        I_input_mean_t1, I_input_mean_t2, ... (1-by-step_tot)
        I_input_std_t1, I_input_std_t2, ... (1-by-step_tot)
```

```
> POPD004 # sampled neuron data (deprecated)
        pop_ind, number_of_sample_neurons,
        data_name1, ..., data_nameX,
        data_1 (sampled_neurons-by-sampled_steps matrix),
        data_2 (sampled_neurons-by-sampled_steps matrix),
        ...
        data_X (sampled_neurons-by-sampled_steps matrix),


> POPD005 # E-I ratio for each neuron
        pop_ind,
        EI_ratio_1, EI_ratio_2, ... (1-by-N),

> SAMF001 # sampled data file name
        path/to/file_name (no comma!)

> POPD006 # sampled neuron data
        pop_ind, number_of_neurons times number_of_steps
        data_name1, ..., data_nameX,
        step_1 (sampled_neurons-by-data_types matrix),
        step_2 (sampled_neurons-by-data_types matrix),
        ...
        step_X (sampled_neurons-by-data_types matrix),

> POPD007 # local field potential data
        pop_ind,
        LFP_1, LFP_2, ..., LFP_step_tot,

> SYND001 # synaptic connection parameters
        number_of_parameters,
        parameter_name1, value1,
        parameter_name2, value2,
        ...

> SYND002 # sampled synapse  data
        pop_ind_pre, pop_ind_post, syn_type, number_of_sample_neurons,
        data (sampled_neurons-by-sampled_steps matrix),

> SYND003 # synapse statistical data
        pop_ind_pre, pop_ind_post, syn_type,
        I_mean_t1, I_mean_t2, ... (1-by-step_tot)
        I_std_t1, I_std_t2, ... (1-by-step_tot)
```

References

[1] Alain Destexhe, Zachary F Mainen, and Terrence J Sejnowski. An efficient method for computing synaptic conductances based on a kinetic model of receptor binding. *Neural Computation*, 6(1):14–18, 1994.

[2] Alain Destexhe, Zachary F Mainen, and Terrence J Sejnowski. Kinetic models of synaptic transmission. In *Methods in neuronal modeling 2*, pages 1–25. 1998.

[3] Matthias H Hennig. Theoretical models of synaptic short term plasticity. *Neural Information Processing with Dynamical Synapses*, page 5, 2015.

[4] Ashok Litwin-Kumar and Brent Doiron. Slow dynamics and high variability in balanced cortical networks with clustered connections. *Nature neuroscience*, 15(11):1498–1505, 2012.

[5] Alessandro Treves. Mean-field analysis of neuronal spike dynamics. *Network: Computation in Neural Systems*, 4(3):259–284, 1993.

[6] TP Vogels, Henning Sprekeler, Friedemann Zenke, Claudia Clopath, and Wulfram Gerstner. Inhibitory plasticity balances excitation and inhibition in sensory pathways and memory networks. *Science*, 334(6062):1569–1573, 2011.