**Cortical representation of touch *in silico -* NetPyNE implementation**

**Loading network data files from Matlab**

For this initial NetPyNE implementation we will instantiate the network model using the existing [Matlab implementation](https://github.com/DepartmentofNeurophysiology/Cortical-representation-of-touch-in-silico), save the cell, connectivity and input data to file, and load in NetPyNE. This includes: cells (properties and location), connections (ids of connected neurons and properties of the connections), and thalamic inputs (spike trains and connections from the barreloids to cortical cells). The Matlab model generates several output files after creating the network. These files can be generated with the function “run\_sim.m” to set up to thalamic inputs. For the NetPyNE-based simulation, we require 2 files to define the network: 1) “xxxxx\_ConData.mat”, 2) “xxxxx\_Thalamic\_Spike\_Trains.mat”.

**Instantiating the network in NetPyNE**

We can then load the information from the saved .mat files to re-create the same network in NetPyNE. We will define the network using NetPyNE's high-level Python specifications, which will then allow us to instantiate the network in NEURON.

*Single cell models:* The dynamics of each cell in the cortical network is based on a modified version of the Izhikevich model, with a history-dependent threshold. These dynamics are specified in a .mod file, based on a previous implementation of the [Izhikevich model](https://senselab.med.yale.edu/ModelDB/showmodel.cshtml?model=39948#tabs-1) by Salvador Dura-Bernal. The new file, “izhi2007b\_dyn\_thr.mod”, includes the dynamics of the threshold (for certain types of cells, “celltype<5”, see COMMENTS section), and the interface with the hoc interpreter to set all properties and communications. Two points are worthy to note:

1. The Izhikevich model has a voltage level at which a spike is generated and the reset rule is applied. This voltage level is set by the parameter “vpeak”, here set at 10 mV. Once this voltage is reached, the reset rule inside the .mod program is applied. However, the voltage update is not performed in the .mod file, but in the NEURON simulator. So, a different (lower) “monitor” threshold is set in this program (via NetPyNE, netParams.defaultThreshold = 0.0) to establish when a spike occurs, so all associated spiking events are sent to the NetCons (objects that perform the synaptic communications). This monitor threshold is set high enough so false positives are not allowed, but low enough so time discretization enables this to be reached and not skipped in the .mod calculation (otherwise, a finer time discretization may be necessary). Take into account that the difference between this “monitor” threshold for declaring the communication event and the threshold used to apply the reset rule results in an effective delay. Eventually, we can evaluate if this small delay results in significant output differences.
2. The initial condition is set in the NET\_RECEIVE block. It currently sets vr (rest potential), but can also be used to set any other parameters.

*Populations:* All populations are set according to the information in the saved Matlab structures. Individual cellular properties are loaded with a specific “cellsList”, with all needed individual properties (locations, not very important now that the connections are already established in the matlab structure, and properties relative to the intrinsic dynamics) within the popParams specifications.

Worthy to note is that regarding the locations, in NetPyNE it is customary to set the depth as the y-coordinate, so this was taken into account when reading individual positions.

*Thalamic inputs:* Thalamic inputs are set as a population of a specific object called “VecStim”, that also has a .mod file associated. Each cell in this population produces a spike train at specific times, which are loaded from the “xxxxx\_Thalamic\_Spike\_Trains.mat”. Cellular locations, which are irrelevant, are set in a very narrow spatial domain at the origin.

*Synapses:* The synaptic mechanism is programmed in a specific .mod file, “FluctExp2Syn.mod”, which includes the dynamics of the gating variable, the stochastic nature of the effective transmission (failure rate), the stochastic nature of the amplitude at the postsynaptic side, and short-term learning dynamics. Also, there is commented block NET\_RECEIVE which includes a “flag\_print” to monitor how things are calculated, useful during development stages.

*Connectivity:* All connections are read from the “ConData” structure inherited from matlab. Identity of pre- and post-synaptic cells, as well as the mean amplitude and the delay, are individually set in the “connParams” dictionary in NetPyNE. Here, the individual identifier (id) of each cell is RELATIVE to the “conds” in the rule, which in this case coincide to the identifier from matlab (minus 1, since in Python everything starts from 0). Other parameters characterizing the synapses of individual connections are set with arbitrary values, except from the type (excitatory/inhibitory).

Once these connections are specified, the network is instantiated with these arbitrary values, via the command “sim.create(…)”. Then, objects corresponding to the connections (NetCon, accessed through the corresponding “hObj” in NetPyNE objects) are modified to specify individual synaptic characteristics of each connection. Worthy to note is that, at this point, for each connection the identity of each pre/post cell is via the “global” id, so it has to be disentangled the global and the relative identifier (necessary to load the properties). The code assumes two situations: The population corresponding to the thalamic input is specified and then all populations located at the cortex, or viceversa (the population corresponding to the thalamic inputs are NOT interleaved within the cortical populations).

*NMOD file compilation:* A final reminder: Here we have three .mod files (for the dynamics of individual cells, for the synapses, and for the input spike trains). BEFORE running the NetPyNE code (and even before starting an IDE, if the program will be run from there), the .mod filed should be compiled. That is, from a terminal (or command window), setting the current directory to the location of the program, it has to be executed “nrnivmodl”.

Comment to add: Scale of the currents [mA/cm2], when the area is 100 um2.

**Preliminary simulation outputs**

*Full scale simulation WITHOUT recurrent dynamics, only thalamic projections*:

Populations:

0-599: Thalamic spike trains

600-1919: 1320 cells Pyr L4

1920-4721: 2802 cells Pyr L4

4722-5003: 282 cells Inh L4

5004-5282: 279 cells Inh L4

5283-11978: 6696 cells Pyr L2/3

11979-12296: 318 cells Inh\_FSBS L2/3

12297-12308: 12 cells Inh\_FSCH L2/3

12309-12473: 165 cells Inh\_BSPV L2/3

12474-12665: 192 cells Inh\_Mar L2/3

12665-12665: 0 cells Inh\_Bit L2/3

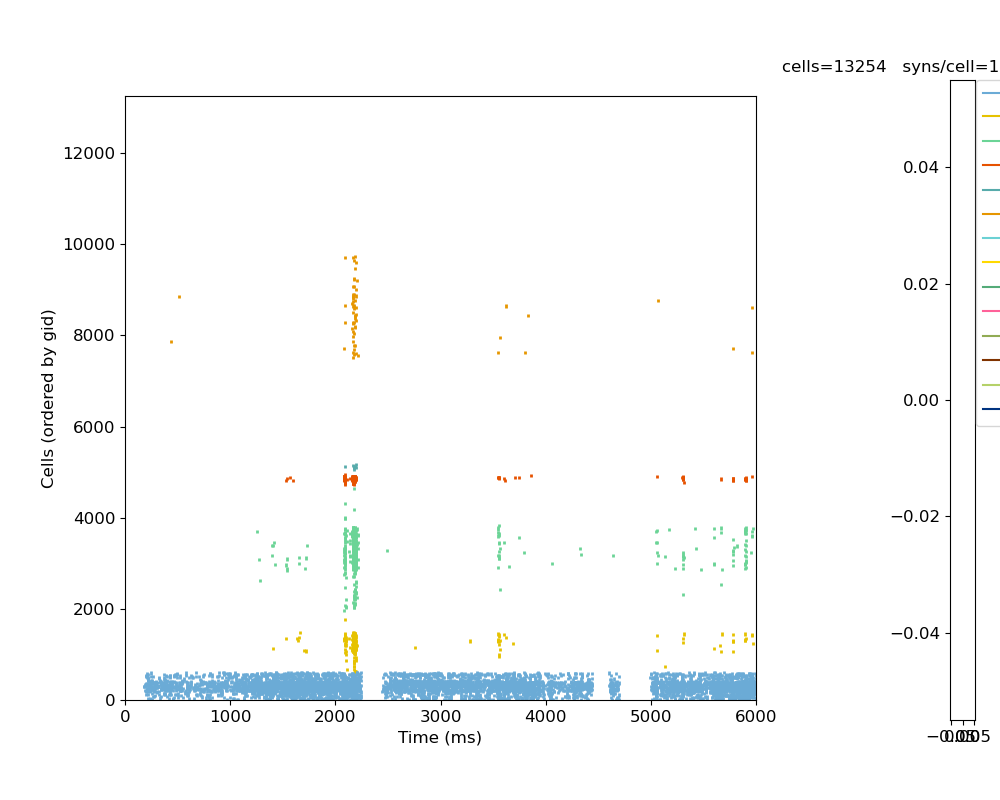
12666-12857: 192 cells Inh\_DBC L2/3

12858-12858: 0 cells Inh\_Bip L2/3

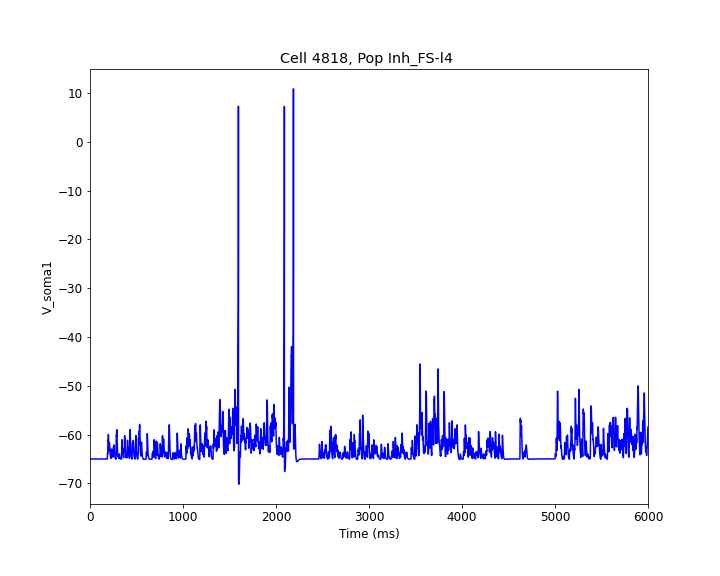
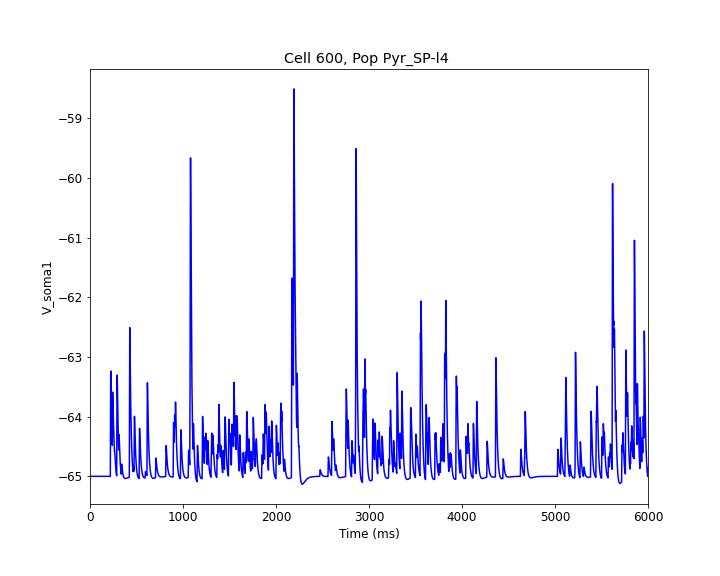
12858-12959: 102 cells Inh\_Bip L2/3

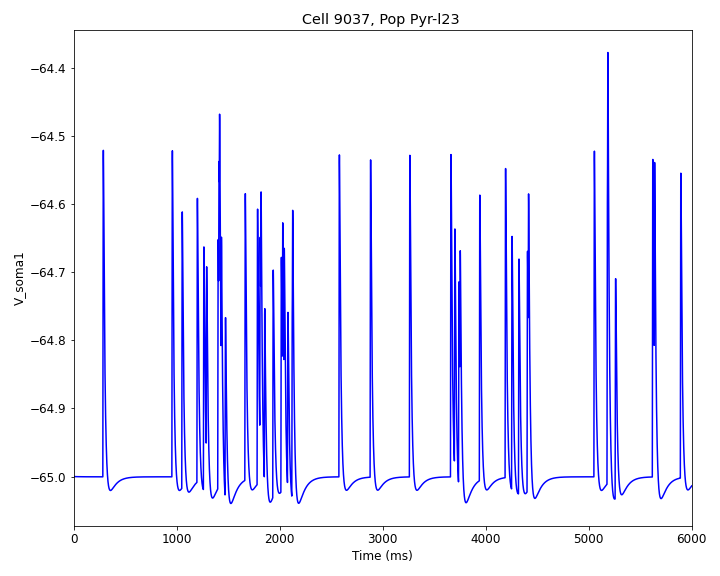
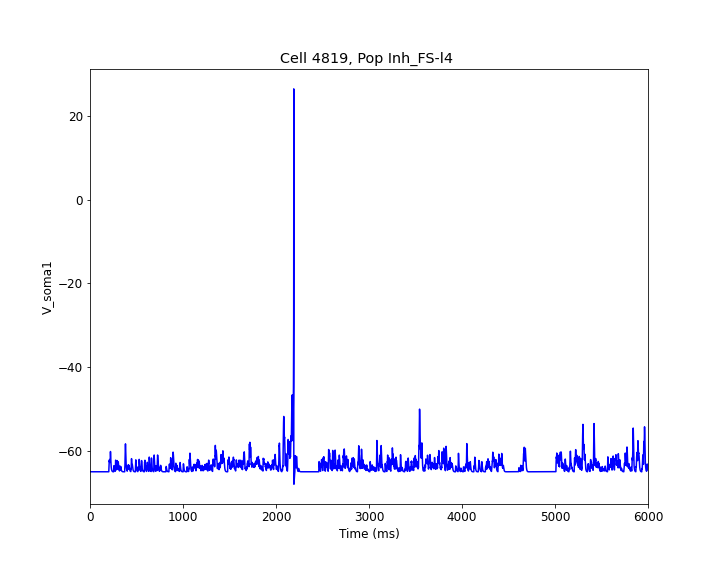
12960-13139: 180 cells Inh\_SBC L2/3

13140-12653: 114 cells Inh\_NG L2/3

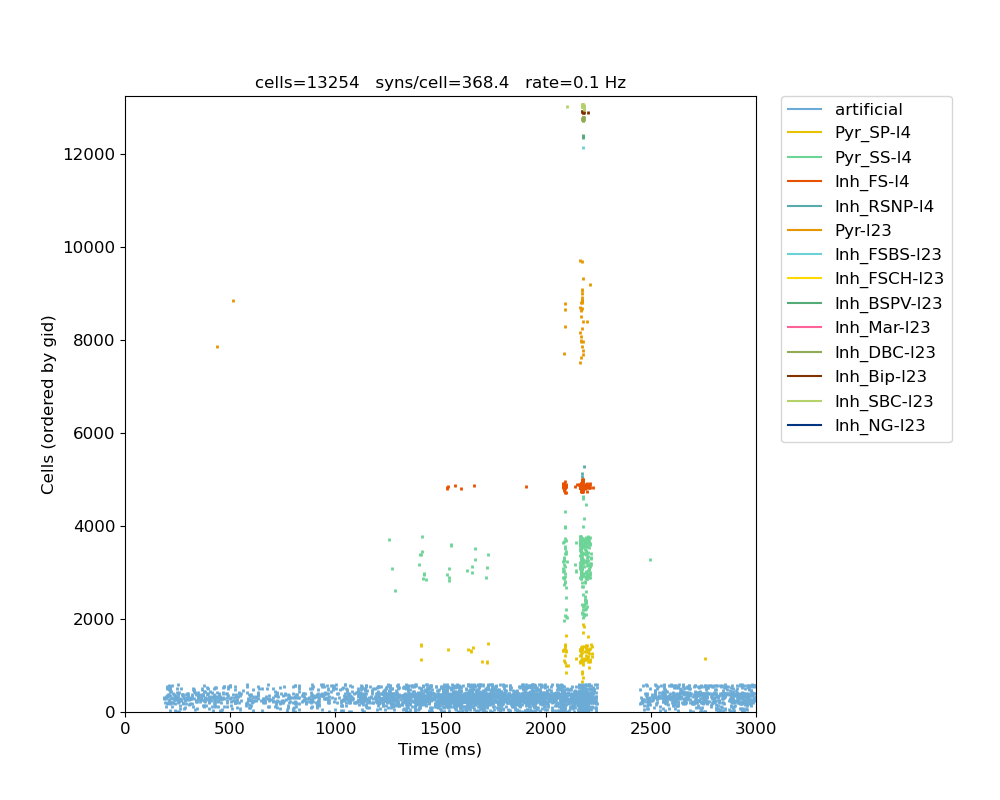


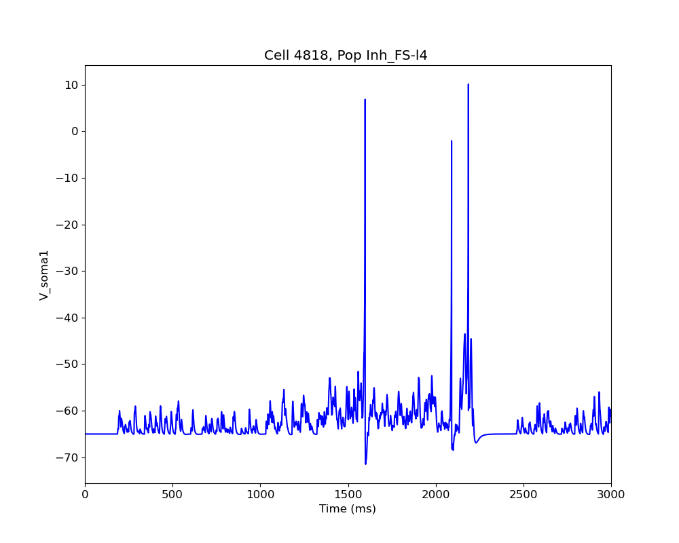
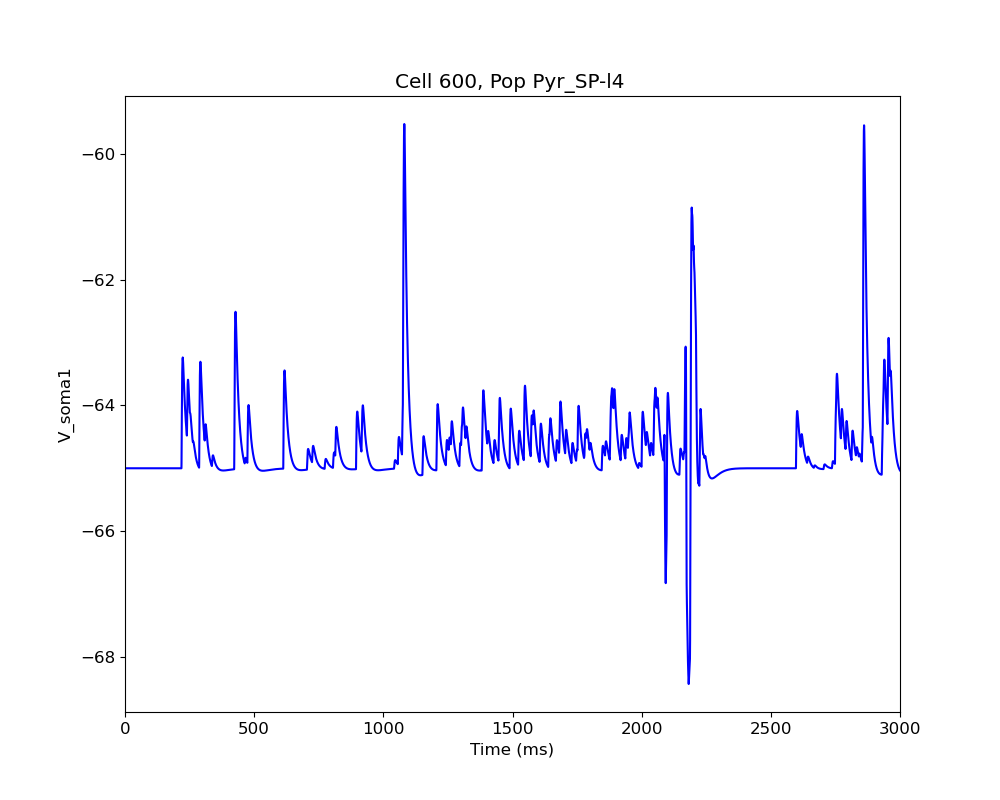
Example recorded traces:

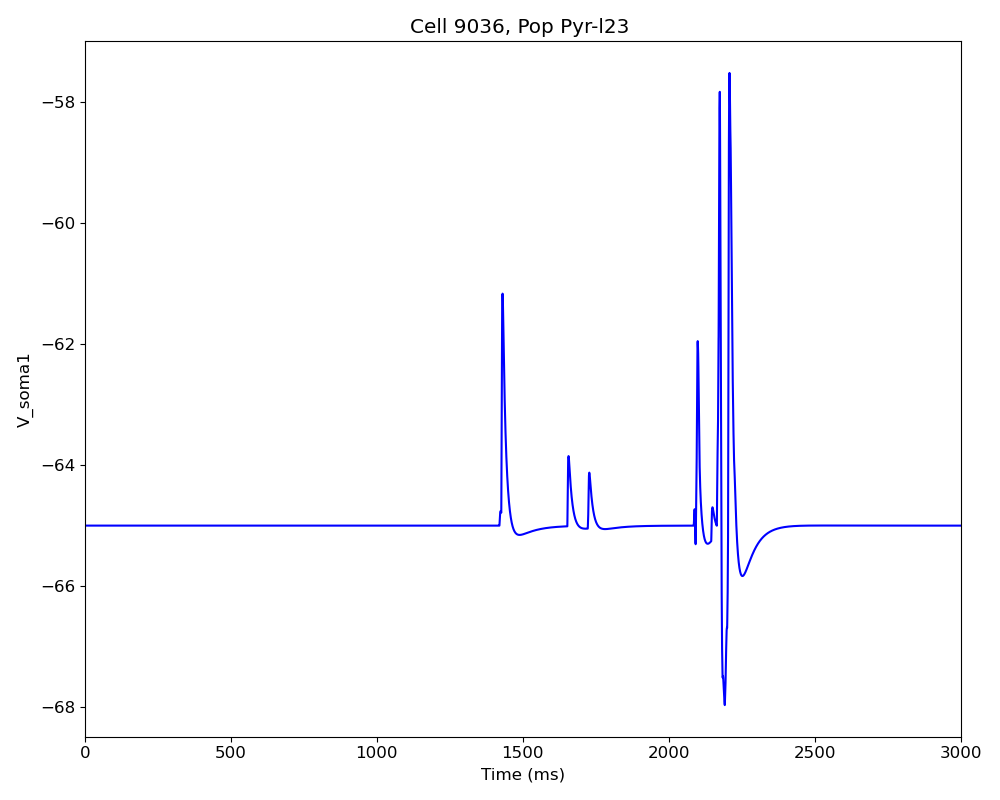
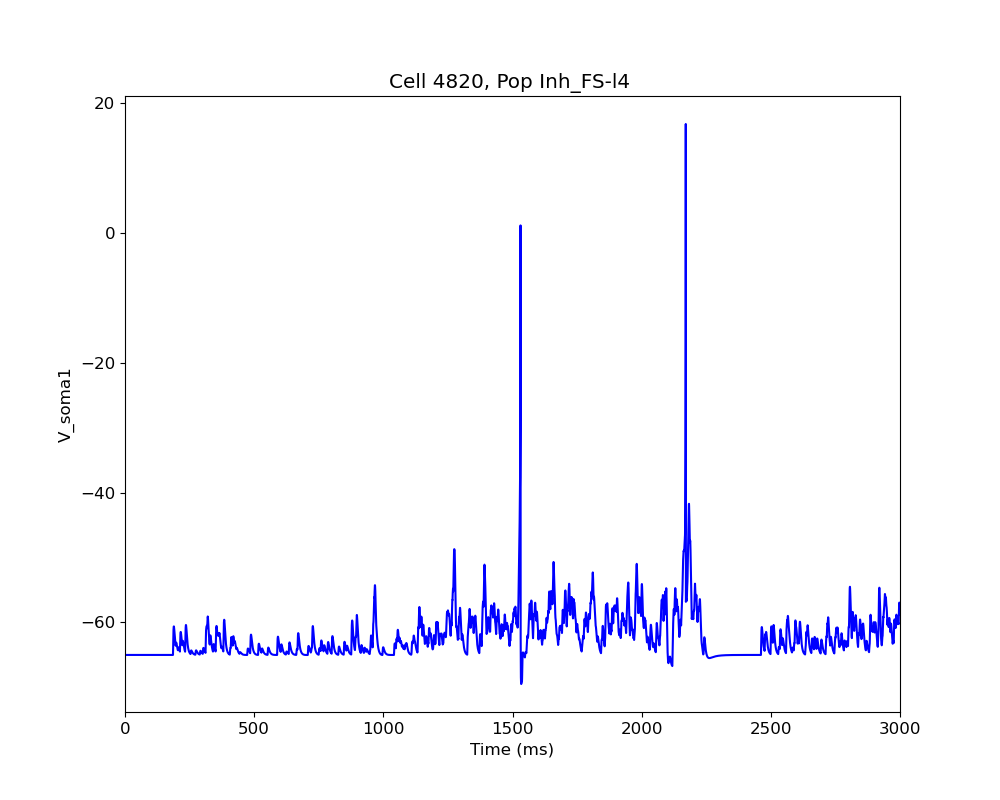




*Full scale simulation with all connections, for half of the total time (Tsim = 3 sec)*. Vr = -65 mV







Note: This is consistent with some results in the paper. For a trial (full 6 secs sim), a neuron fires 2 or 3 spikes per presentation, and the resting potential is relatively depolarized (here, vr = -65 mV). From these simulations, we can see that this is mostly due to the thalamic input, and the recurrent connections add some additional activity. This input spike train correspond to the "trial 8" in their sims.