##

Installation

##

1. Installing PyNN:

$ pip install pyNN

Or follow the instructions in

http://neuralensemble.org/docs/PyNN/installation.html

2. Compile and install PyCARL

2.1 Clone the PyCARL repository to the CARLsim root.

$ cd CARLsim4/

$ git clone https://github.com/adarshabalaji/PyCarlsim.git

2.2 Compile and generate the pyNN -> carlsim interface file (carlsim.py)

2.2.1 Install SWIG:

$ sudo apt update

$ sudo apt install swig

2.2.2 Compile and link the interface file (carlsim_wrap.cxx), generated by swig, with the libcarlsim.a library.

$ cd source

$ source build.sh

This creates a static library _carlsim.so and a pyNN -> CARLsim interface (carlsim.py)

OPTIONAL If you want to compile and link the interface file manually, then

3. 1 Compile the carlsim.i (interface file) using SWIG

$ cd source

$ swig -c++ -python carlsim.i

The output of the swig build is a wrapper file (carlsim_wrap.cxx) and the pyNN -> carlsim interface file (carlsim.py).

3. 2 In the source folder, follow the steps below to compile a new pyNN interface (carlsim.i and carlsim_wrap.cxx) with the static library libcarsim.a (generated during a CARLsim build).

*$gcc -fPIC -c carlsim_wrap.cxx -I/usr/include/python**<version>** -I<CARLsim4 include dir> -I/usr/local/cuda/include -I/usr/local/cuda/samples/common/inc -D__NO_CUDA__*

Update the above command with the python version of choice and the CARLsim4 installation directory.

3.1 Link the carlsim_wrap.o file to the libcarlsim.a file.

$ g++ -shared carlsim_wrap.o -o _carlsim.so

$ g++ -shared ~/CARL/lib/libcarlsim.a carlsim_wrap.o -o _carlsim.so

4. Copy the compiled sources and the carlsim/ folder in the pyCARL repo to pyNN.

$ cp –r CARLsim4/pyCARL/carlsim <root of pyNN Installation>

4.1 Copy the generated _carlsim.so and carlsim.py file to *<root of pyNN Installation>/carlsim*

PyCARL is now integrated with pyNN.

##

List of APIs implemented.

##

1. Create spike generator group.

###

Define the cell type for all the input groups

inputCellType = sim.SpikeSourceArray("input", <numNeurons>, "EXCITATORY_NEURON", "CUBA")

create a population of <numNeurons> neurons of cellType inputCellType.

spike_source = sim.Population(<numNeurons>, inputCellType, label='input')

###
#2. Create neuron groups.

pyCARL only supports Izhikevich neurons

###
#

Define the inzhikevich cell type for the group

izhikevichCellType = sim.Izhikevich("EXCITATORY_NEURON", a=0.02, b=0.2, c=-65, d=8)

create a pyNN population of <numNeurons> neurons of cellType izhikevichCellType

gExc = sim.Population(<numNeurons>, izhikevichCellType, label='Exc')

##

3. Setup Network

An equivalent for this function does not exist for pyNN.

Therefore the user will have to call this CARLsim funciton directly in pyNN.

##
#*sim.state.setupNetwork()*

##

4. Run Network

Input - rumtime in Ms

##

sim.run(<time in Ms>)

##

5. STDPMechanism for a group

Input - rumtime in Ms

##

define the properties of the STDP synapse.

stdp_model = sim.STDPMechanism(

timing_dependence=sim.SpikePairRule(tau_plus=20.0, tau_minus=20.0, A_plus=0.01, A_minus=0.012),

weight_dependence=sim.AdditiveWeightDependence(w_min=0, w_max=0.0000001),

weight=0.00000005,

delay=delay,

dendritic_delay_fraction=float(options.dendritic_delay_fraction))

connect the groups G1 and G2 using an all-to-all connector, with stdp enabled synapses.

connections = sim.Projection(G1, G2, sim.AllToAllConnector(), stdp_model)

##

Connect function

{'OneToOneConnector': 'one-to-one', 'AllToAllConnector': "full", "FixedProbabilityConnector": "random", "FromListConnector": "FromListConnector"}

PyCARL only supports 'full', 'one-to-one' and 'random' connectors

The corresponding

##
#

connectionID = sim.Projection(spike_source, neuron_group1, sim.OneToOneConnector(), synapse_type, receptor_type='excitatory')