11- Advanced BMTK

Contents

Instructions

This document assumes you have completed the necessary steps in **02-Single_Cell_Hoc_BMTK** or **03-Networked_Hoc_BMTK**

Custom Synapses

Included with the mod files from our HCO network from 03-Networked_Hoc_BMTK is a custom `inhsyn.mod` file. We can direct BMTK to use this file by taking the following steps

1. Copy the entire `my_bmtk_model` folder to a new folder called `my_bmtk_model_syn`.
2. In this new folder, create a new file named `synapses.py` and add the following code to this file:

synapses.py

```python
from bmtk.simulator.bionet.pyfunction_cache import add_synapse_model
from neuron import h

def InhSyn(syn_params, sec_x, sec_id):
    """Create a inhsyn synapse
    :param syn_params: parameters of a synapse
    :param sec_x: normalized distance along the section
    :param sec_id: target section
    :return: NEURON synapse object
    """

    lsyn = h.inhsyn(sec_x, sec=sec_id)

    if syn_params.get('esyn'):
        lsyn.esyn = float(syn_params['esyn'])
    if syn_params.get('gmax'):
        lsyn.gmax = float(syn_params['gmax'])

    return lsyn
```

```
20
21 def inhsyn(syn_params, xs, secs):
22     """Create a list of inhsyn synapses
23     :param syn_params: parameters of a synapse
24     :param xs: list of normalized distances along the section
25     :param secs: target sections
26     :return: list of NEURON synpase objects
27     """
28     syns = []
29     for x, sec in zip(xs, secs):
30         syn = InhSyn(syn_params, x, sec)
31         syns.append(syn)
32     return syns
33
34 def load():
34     add_synapse_model(InhSyn, 'inhsyn', overwrite=False)
35     add_synapse_model(InhSyn, overwrite=False)
       return
```

3. Things to note:
    a. (To use this in your model simply change everywhere InhSyn and inhsyn is defined to your synapse name, with variable name changes to line 14+)
    b. Line 1: `add_synapse_model` function is called to add custom synapses to BMTK's python function cache, allowing BMTK to "see" and use your synapse file
    c. Line 4: `syn_params` will be a dictionary containing parameters defined in the json file referenced when creating edges (shown later)
    d. Line 12: `h.inhsyn` will instantiate the `inhsyn` neuron hobject
    e. Line 14-17: Set properties of the synapse by:
        i. Checking to see if the parameter has been defined
        ii. Setting the synapse value to the supplied `syn_params` value
    f. Line 19: Create an additional function for BMTK to handle lists of synapses, we simply link it to our previous InhSyn function to prevent code duplication
    g. Line 34: Call `load()` in your `build_network.py` and `run_bionet.py` scripts early on to notify BMTK that you have custom synapses. (also shown later)

4. Create a new file called `my_inhsyn.json` in `./biophys_components/synaptic_models/` and place the following into it (Note how `esyn` and `gmax` appear in this file and the synapse function we defined previously)

my_inhsyn.json

```
{
  "esyn":"-80",
  "gmax":"40e-3"
}

```

5. In `build_network1.py`, we can now import this synapse file, call the load function, and use the synapse when defining our edges. See the complete file below with explanations.

build_network1.py

```
from bmtk.builder.networks import NetworkBuilder
import synapses

synapses.load()

net1 = NetworkBuilder('hco_net')
net1.add_nodes(N=1,
               cell_name='HCOCell1',
               model_type='biophysical',
               model_template='hoc:HCOcell',
               morphology='blank.swc'
             )

net1.add_nodes(N=1,
               cell_name='HCOCell2',
               model_type='biophysical',
               model_template='hoc:HCOcell',
               morphology='blank.swc'
             )



net1.add_edges(source={'cell_name': 'HCOCell1'},
target={'cell_name':'HCOCell2'},
               connection_rule=1,
               syn_weight=40.0e-02,
               dynamics_params='my_inhsyn.json',
               model_template='inhsyn',
               delay=0.0,
               target_sections=["soma"],
               distance_range=[0,999])

net1.add_edges(source={'cell_name': 'HCOCell2'},
target={'cell_name':'HCOCell1'},
               connection_rule=1,
               syn_weight=40.0e-02,
               dynamics_params='my_inhsyn.json',
               model_template='inhsyn',
               delay=0.0,
               target_sections=["soma"],
               distance_range=[0,999])

net1.build()
net1.save_nodes(output_dir='network')

net1.build()
net1.save_edges(output_dir='network')

```

6. Things to note:

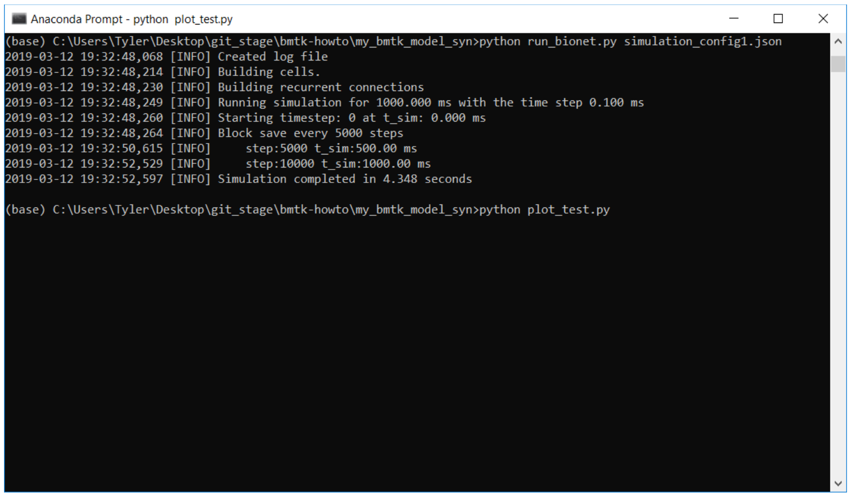
   a. Lines 2 and 4: import the synapses file we just created and load the synapses by calling the `load` function.
   b. Lines 27,28, 36,37: reference the synapse name and dynamics_params file json created earlier

7. Run `python build_network1.py` to build the network.
8. Add the synapse load function to the top of run_bionet.py like the following:

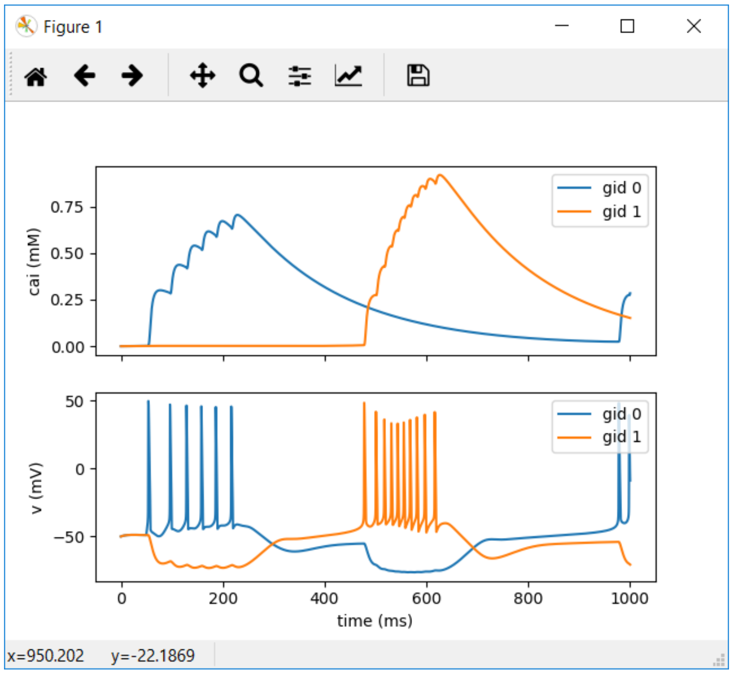
run_bionet.py (snippet)

```
import os, sys
from bmtk.simulator import bionet
from bmtk.simulator.bionet.default_setters.cell_models import loadHOC
import synapses

synapses.load()
bionet.pyfunction_cache.add_cell_model(loadHOC, directive='hoc',
model_type='biophysical')

```

9. You are now ready to run your network, run `python run_bionet.py simulation_config1.json` then `python plot_test.py`. You should notice small changes in the dynamics of the synapse output from previous tests.



```
(base) C:\Users\Tyler\Desktop\git_stage\bmtk-howto\my_bmtk_model_syn>python run_bionet.py simulation_config1.json
2019-03-12 19:32:48,068 [INFO] Created log file
2019-03-12 19:32:48,214 [INFO] Building cells.
2019-03-12 19:32:48,230 [INFO] Building recurrent connections
2019-03-12 19:32:48,249 [INFO] Running simulation for 1000.000 ms with the time step 0.100 ms
2019-03-12 19:32:48,260 [INFO] Starting timestep: 0 at t_sim: 0.000 ms
2019-03-12 19:32:48,264 [INFO] Block save every 5000 steps
2019-03-12 19:32:50,615 [INFO]     step:5000 t_sim:500.00 ms
2019-03-12 19:32:52,529 [INFO]     step:10000 t_sim:1000.00 ms
2019-03-12 19:32:52,597 [INFO] Simulation completed in 4.348 seconds

(base) C:\Users\Tyler\Desktop\git_stage\bmtk-howto\my_bmtk_model_syn>python plot_test.py
```



Custom Cell Positions

Columnar cell positioning and cell **rotation** with **randomized rotations**:

https://github.com/AllenInstitute/bmtk/blob/develop/docs/tutorial/03_single_pop.ipynb

Dynamic Synapse Properties

Edge properties like delay can be changed dynamically, per connection, rather than a blanket set value.

See the following for a great example use case:

https://github.com/AllenInstitute/bmtk/blob/develop/docs/examples/bio_450cells_exact/build_network.py

snippet.py

```
def build_edges(src, trg, sections=['basal', 'apical'],
dist_range=[50.0, 150.0]):
    """Function used to randomly assign a synaptic location based on the
section (soma, basal, apical) and an
    arc-length dist_range from the soma. This function should be passed
into the network and called during the build
    process.
    :param src: source cell (dict)
    :param trg: target cell (dict)
    :param sections: list of target cell sections to synapse onto
    :param dist_range: range (distance from soma center) to place
    :return:
    """
    # Get morphology and soma center for the target cell
    swc_reader = morphologies[trg['model_name']]
    target_coords = [trg['x'], trg['y'], trg['z']]

    sec_ids, sec_xs = swc_reader.choose_sections(sections, dist_range)
# randomly choose sec_ids
    coords = swc_reader.get_coord(sec_ids, sec_xs,
soma_center=target_coords)  # get coords of sec_ids
    dist = swc_reader.get_dist(sec_ids)
    swctype = swc_reader.get_type(sec_ids)
return sec_ids, sec_xs, coords[0][0], coords[0][1], coords[0][2],
dist[0], swctype[0]

…

cm = internal.add_edges(source={'ei': 'e'}, target={'ei': 'e',
'model_type': 'biophysical'},
                              connection_rule=n_connections,
                              connection_params={'prob': 0.2},
                              dynamics_params='AMPA_ExcToExc.json',
                              model_template='Exp2Syn',
                              delay=2.0)
    cm.add_properties('syn_weight', rule=6.0e-05, dtypes=np.float)
    cm.add_properties(['sec_id', 'sec_x', 'pos_x', 'pos_y', 'pos_z',
'dist', 'type'],
                      rule=build_edges,
                      rule_params={'sections': ['basal', 'apical'],
'dist_range': [30.0, 150.0]},
dtypes=[np.int32, np.float, np.float, np.float, np.float, np.float,
np.uint8])
```

Dynamic Node Properties

See

https://github.com/AllenInstitute/bmtk/blob/develop/bmtk/simulator/bionet/default_setters/cell_models.py#L43

When creating your node anything in the dynamics_params["params"] section will be passed to the hoc cell.

net.add_nodes(N=1, ….

dynamics_params={'params':[0.332,3.88]}

…

}

Simply use these in your hoc init as $1, $2, etc…

This is especially useful when testing multiple nodes with different leak channel conductances.

Recurrent Synapses

Example of connecting neurons back to neurons they're connected to. (Also at https://gist.github.com/tjbanks/8228e341e33f65d641bccc6f187e0895)

snippet.py

```
############################################################
# Build connections
############################################################
#Connect CA3o->CA3e Inhibitory
dynamics_file = 'CA3o2CA3e.inh.json'
conn = net.add_edges(source={'pop_name': 'CA3o'}, target={'pop_name':
'CA3e'},
            connection_rule=hipp_dist_connector,

connection_params={'con_pattern':syn[dynamics_file]['con_pattern']},
            syn_weight=5.0e-03,
            dynamics_params=dynamics_file,
            model_template=syn[dynamics_file]['level_of_detail'],
            distance_range=[0.0, 300.0],
            target_sections=['soma'],
            delay=0.0)
conn.add_properties(['sec_id','sec_x'],rule=(0, 0.5),
dtypes=[np.int32,np.float])
conn.add_properties('delay',
            rule=syn_dist_delay,
            rule_params={'base_delay':syn[dynamics_file]['delay']},
            dtypes=np.float)

############################################################
# Build recurrent connection rules
############################################################
def hipp_recurrent_connector(source,target,all_edges=[],min_syn=1,
max_syn=1):
    """
    General logic:
    1. Given a *potential* source and target
    2. Look through all edges currently made
    3. If any of the current edges contains
        a. the current source as a previous target of
        b. the current target as a previous source
    4. Return number of synapses per this connection, 0 otherwise (no
connection)
    """
    for e in all_edges:
        if source['node_id'] == e.target_gid and target['node_id'] ==
e.source_gid:
            return random.randint(min_syn,max_syn)

    return 0
```

```

############################################################
# Build recurrent connections
############################################################

#Connect CA3e->CA3o Excitatory
dynamics_file = 'CA3e2CA3o.exc.json'
conn = net.add_edges(source={'pop_name': 'CA3e'}, target={'pop_name':
'CA3o'},
            connection_rule=hipp_recurrent_connector,
            connection_params={'all_edges':synlist},
            syn_weight=5.0e-03,
            dynamics_params=dynamics_file,
            model_template=syn[dynamics_file]['level_of_detail'],
            distance_range=[0.0, 300.0],
            target_sections=['soma'],
            delay=0.0)
conn.add_properties(['sec_id','sec_x'],rule=(0, 0.5),
dtypes=[np.int32,np.float])
conn.add_properties('delay',
            rule=syn_dist_delay,

rule_params={'base_delay':syn[dynamics_file]['delay'],'dist_delay':0.1},
#Connect.hoc:274 0.1 dist delay
            dtypes=np.float)

```

When synapses are created a line similar to the following will need to be added to the connection rule:

syn_list.append({'source_gid':source['node_id'],'target_gid':target['node_id']})

syn_list = [] will need to be initialized outside of the connection method as well.

Rule Based Synapses

If we want to define connections very specifically we can make use of the “all_to_one” iterator when adding edges.

snippet.py

```
    ############################################################
    # Build strict connection rules
    ############################################################
    def hipp_MF_connector(source,targets,min_syn=1, max_syn=1):
        """
        Exactly 2 connections from source to target
        Pick a random in the target area
        /docs/tutorial/04_multi_pop.ipynb:384:
        "To tell the builder to use this schema, we must set
iterator='all_to_one'
         in the add_edges method. (By default this is set to
'one_to_one'. You can
```

```
         also use 'one_to_all' iterator which will pass in a single
source and all
         possible targets)."
        """

        total_targets = len(targets)
        syns = np.zeros(total_targets)
        x_ind = 0
        n = 0
        while n < 2:
            target_index = random.randint(0,total_targets-1)
            target = targets[target_index]
            dx = target['positions'][x_ind] - source['positions'][x_ind]

            #prob =     1/ (exp( ((abs(dx) -0)^2)/ (2 * (2^2)))) //
Standard deviation of 2 compared to 3 in pp projections More limited
longitudianal spread
            prob = 1/(math.exp(((abs(dx)-0)**2)/(2*(2**2))))
            if random.random() < prob:
                n=n+1
                syns[target_index] = random.randint(min_syn,max_syn)

        return syns

    ###########################################################
    # Build strict connections
    ###########################################################

    #Connect DGg->CA3e Excitatory (Exactly 2 connections allowed)
NOTICE: iterator is 'one_to_all'
    dynamics_file = 'DGg2CA3e.exc.json'
    conn = net.add_edges(source={'pop_name': 'DGg'}, target={'pop_name':
'CA3e'},
                iterator='one_to_all',
                connection_rule=hipp_MF_connector,
                connection_params={},
                syn_weight=5.0e-03,
                dynamics_params=dynamics_file,
                model_template=syn[dynamics_file]['level_of_detail'],
                distance_range=[0.0, 300.0],
                target_sections=['soma'],
                delay=0.0)
    conn.add_properties(['sec_id','sec_x'],rule=(0, 0.5),
dtypes=[np.int32,np.float])
    conn.add_properties('delay',
                rule=syn_dist_delay,
                rule_params={'base_delay':syn[dynamics_file]['delay']},
#Connect.hoc:274 0.1 dist delay
dtypes=np.float)
```