deNEST

Mar 11, 2022

Getting Started

CHAPTER 1

Local

1. Install NEST >= v2.14.0, <3.0 by following the instructions at http://www.nest-simulator.org/installation/.
2. Set up a Python 3 environment and install deNEST with:

```
pip install denest
```

CHAPTER 2

Docker

A Docker image is provided with NEST 2.20 installed, based on nest-docker.

1. From within the repo, build the image:

```
docker build --tag denest .
```

2. Run an interactive container:

```
docker run \
  -it \
  --name denest_simulation \
  --volume $(pwd):/opt/data \
  --publish 8080:8080 \
  denest \
  /bin/bash
```

3. Install deNEST within the container:

```
pip install -e .
```

4. Use deNEST from within the container.

For more information on how to use the NEST Docker image, see nest-docker.

CHAPTER 3

Overview

3.1 Definitions

3.1.1 Network

- We use the term **network** to mean a full network in NEST, consisting of layers of units with specific models, projections of specific types with specific synapse models amongst these layers, population recorders (multimeters, spike detectors) and projection recorders (weight recorder).
  - The full network is represented in deNEST by the `Network` class.
- New NEST **models** (**neuron and generator model**, **synapse model** or **recorder model**) can be specified with arbitrary parameters. During network creation, models with specific parameters are created in NEST using a `nest.CopyModel()` or a `nest.SetDefaults()` call.
  - Synapse models are represented by the `SynapseModel` class in deNEST. All other models are represented by the `Model` class.
  - Neuron and generator models are specified as leaves of the `network/neuron_models` parameter subtree (see section below)
  - Synapse models are specified as leaves of the `network/synapse_models` parameter subtree (see "Network parameters" section below)
  - Recorder models are specified as leaves of the `network/recorder_models` parameter subtree (see "Network parameters" section below)
- A **layer** is a NEST topological layer, created with a `tp.CreateLayer()` call in NEST. A **population** is all the nodes of the same model within a layer.
  - Layers are represented by the `Layer` or `InputLayer` class in deNEST.
  - Layers can be of the type `InputLayer` when they are composed of generators. An extra population of parrot neurons can be automatically created and connected one-to-one to the generators, such that recording of generators' activity is possible. Additionally, `InputLayer` supports shifting the `origin` flag of stimulators at the start of a `Session`.

  - Layers are specified as leaves of the `network/layers` parameter subtree (see "Network parameters" section below).

- A **projection model** is a template specifying parameters passed to `tp.ConnectLayers`, and that individual projections amongst populations can inherit from.
  - Projection models are represented by the `ProjectionModel` class in deNEST.
  - Projection models are specified as leaves of the `network/projection_models` parameter subtree (see "Network parameters" section below).
- A **projection** is an individual projection between layers or populations, created with a `tp.ConnectLayers()` call. The parameters passed to `tp.ConnectLayers()` are those of the "projection model" of the specific projection.
  - The list of all individual projections within the network is specified in the `'projections'` parameter of the `network/topology` parameter subtree (see "Network parameters" section below).
- A **population recorder** is a recorder connected to all the nodes of a given population. A **projection recorder** is a recorder connected to all the synapses of a given projection. Recorders with arbitrary parameters can be defined by creating "recorder models". However, currently only recorders based on the 'multimeter', 'spike_detector' and 'weight_recorder' NEST models are supported.
  - population and projection recorders are represented by the `PopulationRecorder` and `ProjectionRecorder` classes in deNEST.
  - The list of all population recorders and projection recorders are specified in the `'population_recorders'` and `'projection_recorders'` parameters of the `network/recorders` parameter subtree (See "Network parameters" section below).

3.1.2 Simulation

- A **session model** is a template specifying parameters inherited by individual sessions.
  - session models are specified as leaves of the `session_models` parameter subtree (see "Simulation parameters" section below)
- A **session** is a period of simulation of a network with specific inputs and parameters, and corresponds to a single `nest.Simulate()` call. The parameters used by a given session are inherited from its session model.
  - A session's parameters define the operations that may be performed before running it:
    1. Modifying the state of some units (using the `Network.set_state()` method)
    2. (Possibly) shift the `origin` flag for the `InputLayer` stimulators
    3. (Possibly) deactivate the recorders for that session by setting their `start` flag to the end of the session
  - Individual sessions are represented by the `Session` object in deNEST. (see "Simulation parameters" section below)
- A **simulation** is a full experiment. It is represented by the `Simulation` object in deNEST, which contains a `Network` object and a list of `Session` objects.
  - The list of sessions run during a simulation is specified by the `sessions` parameter of the `simulation` parameter subtree (eg: `sessions: ['warmup', 'noise', 'grating', 'noise', 'grating']`) (see "Simulation parameters" section below).

3.2 Overview of a full simulation

A full deNEST simulation consists of the following steps:

1. **Initialize simulation** (`Simulation.__init__(<params_tree>)()`)
    1. **Initialize kernel**: (`Simulation.init_kernel(<kernel_subtree>)()`)
        1. Set NEST kernel parameters
        2. Set seed for NEST's random generator.
    2. **Create network** (`Simulation.create_network(<network_subtree>)()`):
        1. Initialize the network objects (`Network.__init__(<network_subtree)()`)
        2. Create the objects in NEST (`Network.create()`)
    3. **Initialize the sessions** (`Session.__init__()`)
    4. **Save the simulation's metadata**
        - Create and clean the output directory
        - Save the full simulation parameter tree
        - Save deNEST and NEST version information
        - Save session times
        - Save network metadata
        - Save session metadata
2. **Run the simulation** (`Simulation.run()`). This runs each session in turn:
    1. Initialize session (`Session.initialize()`)
        - (Possibly) reset the network
        - (Possibly) inactivate recorders for the duration of the session
        - (Possibly) shift the *origin* of stimulator devices to the start of the session
        - (Possibly) Change some of the network's parameters using the `Network.set_state()` method
            1. Change neuron parameters
            2. Change synapse parameters
    2. Call `nest.Simulate()`.

3.3 Specifying the simulation parameters

All parameters used by deNEST are specified in tree-like YAML files which are converted to `ParamsTree` objects.

In this section, we describe the `ParamsTree` objects, the expected structure of the full parameter tree interpreted by deNEST, and the expected formats and parameters of each of the subtrees that define the various aspects of the network and simulation.

3.3.1 Main parameter file

To facilitate defining parameters in separate files, *denest.run()* and *denest.load_trees()* take as input a path to a YAML file containing the relative paths of the tree-like YAML files to merge so as to define the full parameter tree (for examples, see the *Example declarative specification* or the `params/tree_paths.yml` file in the repository.).

3.3.2 The `ParamsTree` class

The `ParamsTree` class is instantiated from tree-like nested dictionaries. At each node, two reserved keys contain the node's data (called `'params'` and `'nest_params'`). All the other keys are interpreted as named children nodes.

The `'params'` key contains data interpreted by deNEST, while the `'nest_params'` key contains data passed to NEST without modification.

The `ParamsTree` class offers a tree structure with two useful characteristics:

- **Hierarchical inheritance of ancestor's data**: This provides a concise way of defining data for nested scopes. Data common to all leaves may be specified once in the root node, while more specific data may be specified further down the tree. Data lower within the tree overrides data higher in the tree. Ancestor nodes' `params` and `nest_params` are inherited independently.
- **(Horizontal) merging of trees**: `ParamsTree` objects can be merged horizontally with `ParamsTree.merge()`. During the merging of multiple params trees, the contents of the `params` and `nest_params` data keys of nodes at the same relative position are combined. This allows **splitting the deNEST parameter trees in separate files for convenience**, and **overriding the data of a node anywhere in the tree while preserving hierarchical inheritance**.

An example parameter tree

Below is an example of a YAML file with a tree-like structure that can be loaded and represented by the `ParamsTree` class:

```yaml
network:
  neuron_models:
    ht_neuron:
      params:                       # params common to all leaves
        nest_model: ht_neuron
      nest_params:                  # nest_params common to all leaves
        g_KL: 1.0
      cortical_excitatory:
        nest_params:
          tau_spike: 1.75
          tau_m: 16.0
        l1_exc:                     # leaf
        l2_exc:                     # leaf
          nest_params:
            g_KL: 2.0      # Overrides ancestor's value
      cortical_inhibitory:
        nest_params:
          tau_m: 8.0
        l1_inh:                     # leaf
```

This file can be loaded into a `ParamsTree` object. The leaves of the resulting `ParamsTree` and their respective data (`params` and `nest_params`) are as follows. Note the inheritance and override of ancestor data. The nested format above is more compact and less error prone when there are a lot of shared parameters between leaves.

```
l1_exc:
  params:
    nest_model: ht_neuron
  nest_params:
    g_KL: 1.0
    tau_spike: 1.75
    tau_m: 16.0
l2_exc:
  params:
    nest_model: ht_neuron
  nest_params:
    g_KL: 2.0
    tau_spike: 1.75
    tau_m: 16.0
l1_inh:
  params:
    nest_model: ht_neuron
  nest_params:
    g_KL: 1.0
    tau_m: 8.0
```

3.3.3 Full parameter tree: expected structure

All the aspects of the overall simulation are specified in specific named subtrees.

The overall `ParamsTree` passed to `denest.Simulation()` is expected to have no data and the following children:

- `simulation` (`ParamsTree`). Defines input and output paths, and the simulation steps performed. The following parameters (`params` field) are recognized:
  - `output_dir` (str): Path to the output directory. (Default: `'output'`)
  - `input_dir` (str): Path to the directory in which input files are searched for for each session. (Default: `'input'`)
  - `sessions` (list(str)): Order in which sessions are run. Elements of the list should be the name of session models defined in the `session_models` parameter subtree (Default: `[]`)
- `kernel` (`ParamsTree`): Used for NEST kernel initialization. Refer to `Simulation.init_kernel()` for a description of kernel parameters.
- `session_models` (`ParamsTree`): Parameter tree, the leaves of which define session models. Refer to `Sessions()` for a description of session parameters.
- `network` (`ParamsTree`): Parameter tree defining the network in NEST. Refer to `Network` for a full description of network parameters.

3.3.4 `"network"` parameter tree: expected structure

All network parameters are specified in the `network` subtree, used to initialize the `Network` object.

The `network` subtree should have no data, and the following children are expected:

- `neuron_models` (`ParamsTree`). Parameter tree, the leaves of which define neuron models. Each leaf is used to initialize a `Model` object

- `synapse_models` (`ParamsTree`). Parameter tree, the leaves of which define synapse models. Each leaf is used to initialize a `SynapseModel` object
- `layers` (`ParamsTree`). Parameter tree, the leaves of which define layers. Each leaf is used to initialize a `Layer` or `InputLayer` object depending on the value of their `type` `params` parameter.
- `projection_models` (`ParamsTree`). Parameter tree, the leaves of which define projection models. Each leaf is used to initialize a `ProjectionModel` object.
- `recorder_models` (`ParamsTree`). Parameter tree, the leaves of which define recorder models. Each leaf is used to initialize a `Model` object.
- `topology` (`ParamsTree`). `ParamsTree` object without children, the `params` of which may contain a `projections` key specifying all the individual population-to-population projections within the network as a list. `Projection` objects are created from the `topology` `ParamsTree` object by the `Network.build_projections` method. Refer to this method for a description of the `topology` parameter.
- `recorders` (`ParamsTree`). `ParamsTree` object without children, the `params` of which may contain a `population_recorders` and a `projection_recorders` key specifying all the network recorders. `PopulationRecorder` and `ProjectionRecorder` objects are created from the `recorders` `ParamsTree` object by the `Network.build_recorders` method. Refer to this method for a description of the `recorders` parameter.

3.4 Running a deNEST Simulation

- From Python (*e.g.* in a Jupyter notebook):
  - Using the `Simulation` object to run the simulation step by step:

```
import denest

# Path to the parameter files to use
params_path = 'params/tree_paths.yml'

# Override some parameters loaded from the file
overrides = [

  # Maybe change the nest kernel's settings ?
  {'kernel': {'nest_params': {'local_num_threads': 20}}},

  # Maybe change a parameter for all the projections at once ?
  {'network': {'projection_models': {'nest_params': {
      'allow_autapses': true
  }}}},
]

# Load the parameters
params = denest.load_trees(params_path, *overrides)

# Initialize the simulation
sim = denest.Simulation(params, output_dir='output')

# Run the simulation (runs all the sessions)
sim.run()
```

  - Using the *`denest.run()`* function to run the full simulation at once:

```
import denest

# Path to the parameter files to use
params_path = 'params/tree_paths.yml'

# Override parameters
overrides = []

denest.run(params_path, *overrides, output_dir=None)
```

- From the command line:

```
python -m denest <tree_paths.yml> [-o <output_dir>]
```

CHAPTER 4

Example declarative specification

Here is an example parameter file (in YAML) that specifies a full simulation:

```yaml
params: {}
nest_params: {}
session_models:
  params:
    reset_network: false
    record: true
    shift_origin: false
  nest_params: {}
  even_rate:
    params:
      simulation_time: 50.0
      unit_changes:
      - layers:
        - input_layer
        population_name: input_exc
        change_type: constant
        from_array: false
        nest_params:
          rate: 100.0
    nest_params: {}
  warmup:
    params:
      reset_network: true
      record: false
      simulation_time: 50.0
      unit_changes:
      - layers:
        - l1
        population_name: null
        change_type: constant
        from_array: false
        nest_params:
```

(continues on next page)

(continued from previous page)

```
          V_m: -70.0
      - layers:
        - input_layer
        population_name: input_exc
        change_type: constant
        from_array: false
        nest_params:
          rate: 100.0
    nest_params: {}
  arbitrary_rate:
    params:
      simulation_time: 50.0
      unit_changes:
      - layers:
        - input_layer
        population_name: input_exc
        change_type: constant
        from_array: true
        nest_params:
          rate: ./input_layer_rates_5x5x1.npy
    nest_params: {}
simulation:
  params:
    sessions:
    - warmup
    - even_rate
    - arbitrary_rate
    output_dir: ./output
    input_dir: ./params/input
  nest_params: {}
kernel:
  params:
    extension_modules: []
    nest_seed: 94
  nest_params:
    local_num_threads: 20
    resolution: 1.0
    print_time: true
    overwrite_files: true
network:
  params: {}
  nest_params: {}
  neuron_models:
    params: {}
    nest_params: {}
    ht_neuron:
      params:
        nest_model: ht_neuron
      nest_params:
        g_peak_NaP: 0.5
        g_peak_h: 0.0
        g_peak_T: 0.0
        g_peak_KNa: 0.5
        g_KL: 1.0
        E_rev_NaP: 55.0
        g_peak_AMPA: 0.1
        g_peak_NMDA: 0.15
```

(continues on next page)

(continued from previous page)

```
        g_peak_GABA_A: 0.33
        g_peak_GABA_B: 0.0132
        instant_unblock_NMDA: true
        S_act_NMDA: 0.4
        V_act_NMDA: -58.0
      cortical_inhibitory:
        params: {}
        nest_params:
          theta_eq: -53.0
          tau_theta: 1.0
          tau_spike: 0.5
          tau_m: 8.0
        l1_inh:
          params: {}
          nest_params: {}
        l2_inh:
          params: {}
          nest_params: {}
      cortical_excitatory:
        params: {}
        nest_params:
          theta_eq: -51.0
          tau_theta: 2.0
          tau_spike: 1.75
          tau_m: 16.0
        l1_exc:
          params: {}
          nest_params: {}
        l2_exc:
          params: {}
          nest_params: {}
    input_exc:
      params:
        nest_model: poisson_generator
      nest_params: {}
  layers:
    params:
      type: null
    nest_params:
      rows: 5
      columns: 5
      extent:
      - 8.0
      - 8.0
      edge_wrap: true
    input_area:
      params:
        type: InputLayer
        add_parrots: true
      nest_params: {}
      input_layer:
        params:
          populations:
            input_exc: 1
        nest_params: {}
    l1_area:
      params: {}
```

(continues on next page)

(continued from previous page)

```
    nest_params: {}
    l1:
      params:
        populations:
          l1_exc: 2
          l1_inh: 1
      nest_params: {}
  l2_area:
    params: {}
    nest_params: {}
    l2:
      params:
        populations:
          l2_exc: 2
          l2_inh: 1
      nest_params: {}
synapse_models:
  params: {}
  nest_params: {}
  static_synapse:
    params:
      nest_model: static_synapse_lbl
    nest_params: {}
    input_synapse_NMDA:
      params:
        target_neuron: ht_neuron
        receptor_type: NMDA
      nest_params: {}
    input_synapse_AMPA:
      params:
        target_neuron: ht_neuron
        receptor_type: AMPA
      nest_params: {}
  ht_synapse:
    params:
      nest_model: ht_synapse
      target_neuron: ht_neuron
    nest_params: {}
    GABA_B_syn:
      params:
        receptor_type: GABA_B
      nest_params: {}
    AMPA_syn:
      params:
        receptor_type: AMPA
      nest_params: {}
    GABA_A_syn:
      params:
        receptor_type: GABA_A
      nest_params: {}
    NMDA_syn:
      params:
        receptor_type: NMDA
      nest_params: {}
topology:
  params:
    projections:
```

(continues on next page)

(continued from previous page)

```
    - source_layers:
      - input_layer
      source_population: parrot_neuron
      target_layers:
      - l1
      target_population: l1_exc
      projection_model: input_projection_AMPA
    - source_layers:
      - input_layer
      source_population: parrot_neuron
      target_layers:
      - l1
      target_population: l1_inh
      projection_model: input_projection_AMPA
    - source_layers:
      - input_layer
      source_population: parrot_neuron
      target_layers:
      - l1
      target_population: l1_inh
      projection_model: input_projection_NMDA
    - source_layers:
      - l1
      source_population: l1_exc
      target_layers:
      - l1
      target_population: l1_exc
      projection_model: horizontal_exc
    - source_layers:
      - l1
      source_population: l1_exc
      target_layers:
      - l1
      target_population: l1_inh
      projection_model: horizontal_exc
    - source_layers:
      - l1
      source_population: l1_exc
      target_layers:
      - l2
      target_population: l2_exc
      projection_model: FF_exc
    - source_layers:
      - l1
      source_population: l1_exc
      target_layers:
      - l2
      target_population: l2_inh
      projection_model: FF_exc
  nest_params: {}
recorder_models:
  params: {}
  nest_params:
    record_to:
    - file
    - memory
    withgid: true
```

(continues on next page)

(continued from previous page)

```
      withtime: true
    spike_detector:
      params:
        nest_model: spike_detector
      nest_params: {}
    weight_recorder:
      params:
        nest_model: weight_recorder
      nest_params:
        record_to:
        - file
        - memory
        withport: false
        withrport: true
    multimeter:
      params:
        nest_model: multimeter
      nest_params:
        interval: 1.0
        record_from:
        - V_m
  recorders:
    params:
      population_recorders:
      - layers: []
        populations: []
        model: multimeter
      - layers:
        - l2
        populations:
        - l2_inh
        model: multimeter
      - layers: null
        populations:
        - l2_exc
        model: multimeter
      - layers:
        - l1
        populations: null
        model: multimeter
      - layers: null
        populations: null
        model: spike_detector
      projection_recorders:
      - source_layers:
        - input_layer
        source_population: parrot_neuron
        target_layers:
        - l1
        target_population: l1_exc
        projection_model: input_projection_AMPA
        model: weight_recorder
      - source_layers:
        - l1
        source_population: l1_exc
        target_layers:
        - l1
```

(continues on next page)

(continued from previous page)

```
      target_population: l1_exc
      projection_model: horizontal_exc
      model: weight_recorder
  nest_params: {}
projection_models:
  params:
    type: topological
  nest_params:
    allow_autapses: false
    allow_multapses: false
    allow_oversized_mask: true
  horizontal_inh:
    params: {}
    nest_params:
      connection_type: divergent
      synapse_model: GABA_A_syn
      mask:
        circular:
          radius: 7.0
      kernel:
        gaussian:
          p_center: 0.25
          sigma: 7.5
      weights: 1.0
      delays:
        uniform:
          min: 1.75
          max: 2.25
  input_projection:
    params: {}
    nest_params:
      connection_type: convergent
      mask:
        circular:
          radius: 12.0
      kernel: 0.8
      weights: 1.0
      delays:
        uniform:
          min: 1.75
          max: 2.25
    input_projection_AMPA:
      params: {}
      nest_params:
        synapse_model: input_synapse_AMPA
    input_projection_NMDA:
      params: {}
      nest_params:
        synapse_model: input_synapse_NMDA
  horizontal_exc:
    params: {}
    nest_params:
      connection_type: divergent
      synapse_model: AMPA_syn
      mask:
        circular:
          radius: 12.0
```

(continues on next page)

(continued from previous page)

```
      kernel:
        gaussian:
          p_center: 0.05
          sigma: 7.5
      weights: 1.0
      delays:
        uniform:
          min: 1.75
          max: 2.25
  FF_exc:
    params: {}
    nest_params:
      connection_type: convergent
      synapse_model: AMPA_syn
      mask:
        circular:
          radius: 12.0
      kernel: 0.8
      weights: 1.0
      delays:
        uniform:
          min: 1.75
          max: 2.25
```

CHAPTER 5

deNEST

deNEST is a Python library for specifying networks and running simulations using the NEST simulator (https://nest-simulator.org).

deNEST allows the user to concisely specify large-scale networks and simulations in hierarchically-organized **declarative parameter files**.

From these parameter files, a network is instantiated in NEST (neurons & their projections), and a simulation is run in sequential steps ("sessions"), during which the network parameters can be modified and the network can be stimulated, recorded, etc.

Some advantages of the declarative approach:

- Parameters and code are separated
- Simulations are easier to reason about, reuse, and modify
- Parameters are more readable and succinct
- Parameter files can be easily version controlled and diffs are smaller and more interpretable
- Clean separation between the specification of the "network" (the simulated neuronal system) and the "simulation" (structured stimulation and recording of the network), which facilitates running different experiments using the same network
- Parameter exploration is more easily automated

To learn how to use deNEST, please see the *Overview* section and the Jupyter notebook tutorials:

5.1 Quickstart

In this tutorial we do the following :

1. **Examine the `'network'` parameter tree**
    1. Models (`neuron_models`, `recorder_models`, `synapse_models`)
    2. Layers (`layers`)

3. Projections (`projection_models` and `topology` subtrees
4. Population and projection recorders (`recorders`)

2. **Examine the full simulation parameter tree**
    1. `network` params subtree
    2. `kernel` params subtree
    3. `session_models` params subtree
    4. `simulation` params subtree
3. **Run a full simulation**
    1. Load parameter tree from one or multiple files
    2. Using the `Simulation` object
    3. Using the `denest.run()` method
    4. From the command line

[1]:
```
import nest
import yaml
from pathlib import Path
from pprint import pprint

from denest import *
import denest
```

[2]:
```
PARAMS_DIR = Path('./data/params')
OUTPUT_DIR = Path('./data/outputs/output')
```

5.1.1 `"network"` tree

A full NEST network is represented in deNEST by the `Network` object.

The `Network` object is initialized from the full `"network"` parameter tree. The `'network'` tree should have specific children subtree, each used to initialize different elements of the network.

[3]:
```
net_tree = ParamsTree.read(PARAMS_DIR/'network_tree.yml').children['network']
```

[4]:
```
# Full 'network' tree
net_tree
```

[4]:
```
ParamsTree(name='network', parent='None')
  params: {}
  nest_params: {}
  neuron_models:
    params: {}
    nest_params: {}
    my_neuron:
      params:
        nest_model: ht_neuron
      nest_params:
        g_KL: 1.0
        g_NaL: 1.0
      l1_exc:
```

(continues on next page)

(continued from previous page)

```
      params: {}
      nest_params:
        V_m: -44.0

... [116 lines] ...

    - layers:
      - input_layer
      populations: null
      model: my_spike_detector
    projection_recorders:
    - source_layers:
      - l1
      source_population: l1_exc
      target_layers:
      - l1
      target_population: l1_inh
      projection_model: proj_1_AMPA
      model: weight_recorder
  nest_params: {}
```

[5]:
```
print("'network' tree's expected subtrees:", net_tree.children.keys())
```

```
'network' tree's expected subtrees: dict_keys(['neuron_models', 'synapse_models',
↪'layers', 'projection_models', 'topology', 'recorder_models', 'recorders'])
```

Models (`'neuron_model'`, `'synapse_models'`, `'recorder_models'` subtrees)

We can define neuron, recorder, stimulator and synapse models with arbitrary parameters from parameter trees.

Each leaf corresponds to a new (named) model. Its `nest_params` and `params` are hierarchically inherited.

The `nest_model` used is specified in the leaf's `params`

`'neuron_models'` tree

[6]:
```
net_tree.children['neuron_models']
```

[6]:
```
ParamsTree(name='neuron_models', parent='network')
  params: {}
  nest_params: {}
  my_neuron:
    params:
      nest_model: ht_neuron
    nest_params:
      g_KL: 1.0
      g_NaL: 1.0
    l1_exc:
      params: {}
      nest_params:
        V_m: -44.0
    l1_inh:
      params: {}
```

(continues on next page)

(continued from previous page)

```
      nest_params:
        V_m: -55.0
```

Below are the leaves of the tree. Each initializes a model. The `nest_params` and `params` are inherited from ancestor leaves.

[7]:
```
print("Tree's leaves and their inherited parameters:")
[
    (f'leaf name: {l.name})',
      f'leaf `params`: {dict(l.params)}',
      f'leaf `nest_params`: {dict(l.nest_params)}')
     for l in net_tree.children['neuron_models'].leaves()
]
```

```
Tree's leaves and their inherited parameters:
```

[7]:
```
[('leaf name: l1_exc)',
  "leaf `params`: {'nest_model': 'ht_neuron'}",
  "leaf `nest_params`: {'g_KL': 1.0, 'g_NaL': 1.0, 'V_m': -44.0}"),
 ('leaf name: l1_inh)',
  "leaf `params`: {'nest_model': 'ht_neuron'}",
  "leaf `nest_params`: {'g_KL': 1.0, 'g_NaL': 1.0, 'V_m': -55.0}")]
```

`'recorder_models'` subtree

Same thing as for neuron models:

[8]:
```
net_tree.children['recorder_models']
```

[8]:
```
ParamsTree(name='recorder_models', parent='network')
  params: {}
  nest_params:
    record_to:
    - memory
    - file
  weight_recorder:
    params:
      nest_model: weight_recorder
    nest_params: {}
  my_multimeter:
    params:
      nest_model: multimeter
    nest_params:
      record_from:
      - V_m
  my_spike_detector:
    params:
      nest_model: spike_detector
    nest_params: {}
```

[9]:
```
print("Tree's leaves and their inherited parameters:")
[
    (f'leaf name: {l.name})',
      f'leaf `params`: {dict(l.params)}',
```

(continues on next page)

(continued from previous page)

```
      f'leaf `nest_params`: {dict(l.nest_params)}')
     for l in net_tree.children['recorder_models'].leaves()
]
```

```
Tree's leaves and their inherited parameters:
```

[9]:
```
[('leaf name: weight_recorder)',
  "leaf `params`: {'nest_model': 'weight_recorder'}",
  "leaf `nest_params`: {'record_to': ['memory', 'file']}"),
 ('leaf name: my_multimeter)',
  "leaf `params`: {'nest_model': 'multimeter'}",
  "leaf `nest_params`: {'record_to': ['memory', 'file'], 'record_from': ['V_m']}"),
 ('leaf name: my_spike_detector)',
  "leaf `params`: {'nest_model': 'spike_detector'}",
  "leaf `nest_params`: {'record_to': ['memory', 'file']}")]
```

`'synapse_model'` subtree

- Same thing as for neuron models, with as a bonus a convenient way of specifying the receptor type of the synapse
- If specifying the `receptor_type` and `target_model` in the `SynapseModel` params, the corresponding port is determined automatically
- Connection **'weight'** nest_param should be specified in the projections parameters rather than the synapse parameters

[10]:
```
pprint(net_tree.children['synapse_models'])
```

```
ParamsTree(name='synapse_models', parent='network')
  params: {}
  nest_params: {}
  my_AMPA_synapse:
    params:
      nest_model: ht_synapse
      receptor_type: AMPA
      target_neuron: ht_neuron
    nest_params: {}
  my_GABAA_synapse:
    params:
      nest_model: ht_synapse
      receptor_type: GABA_A
      target_neuron: ht_neuron
    nest_params: {}
```

[11]:
```
print("Tree's leaves and their inherited parameters:")
[
    (f'leaf name: {l.name})',
      f'leaf `params`: {dict(l.params)}',
      f'leaf `nest_params`: {dict(l.nest_params)}')
     for l in net_tree.children['synapse_models'].leaves()
]
```

```
Tree's leaves and their inherited parameters:
```

[11]:
```
[('leaf name: my_AMPA_synapse)',
  "leaf `params`: {'nest_model': 'ht_synapse', 'receptor_type': 'AMPA', 'target_neuron
↪': 'ht_neuron'}",
  'leaf `nest_params`: {}'),
 ('leaf name: my_GABAA_synapse)',
  "leaf `params`: {'nest_model': 'ht_synapse', 'receptor_type': 'GABA_A', 'target_
↪neuron': 'ht_neuron'}",
  'leaf `nest_params`: {}')]
```

Layers (`'layers'` subtree)

- As for models, we can create `nest.Topology` layers from the leaves of a tree.
- The elements can be nest models with their default parameters, or the ones we just created with custom params
- For layers of stimulator devices, we can use the `InputLayer` object, which can automatically create paired parrot neurons for each stimulator units, by adding `type: 'InputLayer'` to the params
- The leaves should have a `population` entry in the `params` field, that specifies the number of units of each model at each layer location (replaces the `elements` nest_params)

[12]:
```
layer_tree = net_tree.children['layers']
layer_tree
```

[12]:
```
ParamsTree(name='layers', parent='network')
  params: {}
  nest_params: {}
  layers:
    params:
      type: null
    nest_params:
      rows: 5
      columns: 5
      extent:
      - 5.0
      - 5.0
      edge_wrap: true
    input_layer:
      params:
        type: InputLayer
        add_parrots: true
        populations:
          spike_generator: 1
      nest_params: {}
    l1:
      params:
        populations:
          l1_exc: 4
          l1_inh: 2
      nest_params: {}
```

[13]:
```
print("Tree's leaves and their inherited parameters:")
[
    (f'leaf name: {l.name})',
      f'leaf `params`: {dict(l.params)}',
      f'leaf `nest_params`: {dict(l.nest_params)}')
```

(continues on next page)

(continued from previous page)

```
    for l in layer_tree.leaves()
]
```

```
Tree's leaves and their inherited parameters:
```

[13]:
```
[('leaf name: input_layer)',
  "leaf `params`: {'type': 'InputLayer', 'add_parrots': True, 'populations': {'spike_
↪generator': 1}}",
  "leaf `nest_params`: {'rows': 5, 'columns': 5, 'extent': [5.0, 5.0], 'edge_wrap':␣
↪True}"),
 ('leaf name: l1)',
  "leaf `params`: {'type': None, 'populations': {'l1_exc': 4, 'l1_inh': 2}}",
  "leaf `nest_params`: {'rows': 5, 'columns': 5, 'extent': [5.0, 5.0], 'edge_wrap':␣
↪True}")]
```

Projections (`'projection_models'` and `'topology'` subtrees)

We create projections using a two step process:

1. Create `ProjectionModel` objects from a tree. Each named leaf will define a template from which individual projections can inherit their parameters (`'projection_models` subtree)
2. Create `Projection` objects from a list, specifying for each item the source layer x population, target layer x population and the projection model to inherit parameters from (`'topology'` subtree)

Define templates from the `projection_models` tree

[14]:
```
proj_model_tree = net_tree.children['projection_models']
proj_model_tree
```

[14]:
```
ParamsTree(name='projection_models', parent='network')
  params: {}
  nest_params:
    connection_type: divergent
    mask:
      circular:
        radius: 2.0
    kernel: 1.0
  proj_1_AMPA:
    params: {}
    nest_params:
      synapse_model: my_AMPA_synapse
      weights: 1.0
  proj_2_GABAA:
    params: {}
    nest_params:
      synapse_model: my_GABAA_synapse
      weights: 2.0
```

[15]:
```
print("Tree's leaves and their inherited parameters:")
[
    (f'leaf name: {l.name})',
      f'leaf `params`: {dict(l.params)}',
      f'leaf `nest_params`: {dict(l.nest_params)}')
```

(continues on next page)

(continued from previous page)

```
    for l in net_tree.children['projection_models'].leaves()
]
```

```
Tree's leaves and their inherited parameters:
```

[15]:
```
[('leaf name: proj_1_AMPA)',
  'leaf `params`: {}',
  "leaf `nest_params`: {'connection_type': 'divergent', 'mask': {'circular': {'radius
↪': 2.0}}, 'kernel': 1.0, 'synapse_model': 'my_AMPA_synapse', 'weights': 1.0}"),
 ('leaf name: proj_2_GABAA)',
  'leaf `params`: {}',
  "leaf `nest_params`: {'connection_type': 'divergent', 'mask': {'circular': {'radius
↪': 2.0}}, 'kernel': 1.0, 'synapse_model': 'my_GABAA_synapse', 'weights': 2.0}")]
```

Define individual projections from the `topology` tree

The list of projections is defined in the `projections` params of the `topology` tree

Check out the doc of `Network.build_projections` for expected formatting

For each item in the list, a projection is created for each of the `<source_layer> x <target_layer>` combinations. The `params` and `nest_params` are inherited from the template `projection_model`

[16]:
```
topo_tree = net_tree.children['topology']
topo_tree
```

[16]:
```
ParamsTree(name='topology', parent='network')
  params:
    projections:
    - source_layers:
      - input_layer
      source_population: parrot_neuron
      target_layers:
      - l1
      target_population: l1_exc
      projection_model: proj_1_AMPA
    - source_layers:
      - l1
      source_population: l1_exc
      target_layers:
      - l1
      target_population: l1_inh
      projection_model: proj_1_AMPA
    - source_layers:
      - l1
      source_population: l1_inh
      target_layers:
      - l1
      target_population: l1_exc
      projection_model: proj_2_GABAA
  nest_params: {}
```

Recorders (`recorders` subtree)

Similarly to the `topology` tree, recorders are defined from lists.

We separate recorders connected to synapses (eg weight recorder) and those connected to units (eg spike detectors), which are defined in the `projection_recorders` and `population_recorders` params (resp.) of the `recorders` tree.

Check out the doc of the `Network.build_recorders`, `Network.build_population_recorders` and `Network.build_projection_recorders` methods for expected formatting

The parameters of the recorders can be changed by using custom recorder models (in the `recorder_models` tree, see above)

[17]:
```
recorders_tree = net_tree.children['recorders']
recorders_tree
```

[17]:
```
ParamsTree(name='recorders', parent='network')
  params:
    population_recorders:
    - layers:
      - l1
      populations:
      - l1_exc
      model: my_multimeter
    - layers:
      - input_layer
      populations: null
      model: my_spike_detector
    projection_recorders:
    - source_layers:
      - l1
      source_population: l1_exc
      target_layers:
      - l1
      target_population: l1_inh
      projection_model: proj_1_AMPA
      model: weight_recorder
  nest_params: {}
```

5.1.2 Full parameter tree

A full simulation consists in

- kernel initialization
- network creation
- running the network in multiple 'sessions' at the start of which some modifications to the network can be made

The full simulation is represented by the `Simulation` object, which is initialized from a parameter tree with the following subtrees:

1. `'network'` subtree, passed to `Network()`
2. `'kernel'` subtree, passed to `Simulation.init_kernel()`
3. `'session_models'` subtree, defining the parameters for each individual `Session`
4. `'simulation'` subtree, containing the list of sessions that we run and the input/output directory paths

[18]:
```
full_tree = ParamsTree.read(PARAMS_DIR/'parameter_tree.yml')
```

```
[19]: full_tree.children.keys()
```

```
[19]: dict_keys(['kernel', 'simulation', 'session_models', 'network'])
```

'network' subtree

The 'network' subtree is expected to have the following children, as we saw above:

```
[20]: net_tree = full_tree.children['network']
      net_tree.children.keys()
```

```
[20]: dict_keys(['neuron_models', 'synapse_models', 'layers', 'projection_models', 'topology
      ↪', 'recorder_models', 'recorders'])
```

It is used to initialize a Network object, representing the full NEST network.

```
[21]: net = Network(net_tree)
```

```
2020-06-30 13:51:20,459 [denest.network] INFO: Build N=2 ``Model`` objects
2020-06-30 13:51:20,460 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
2020-06-30 13:51:20,461 [denest.network] INFO: Build N=3 ``Model`` objects
2020-06-30 13:51:20,467 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:51:20,469 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:51:20,470 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:51:20,471 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
2020-06-30 13:51:20,476 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
2020-06-30 13:51:20,484 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:51:20,485 [denest.network] INFO: Build N=1 projection recorders.
```

'kernel' subtree

The kernel subtree is passed to the Simulation.init_kernel method.

- All nest_params are passed to nest.SetKernelStatus
- Two optional params:
  - nest_seed (used to initialize at once the thread-wide and general nest rsg. The msd, grng_seed and rng_seed nest params are reserved)
  - extension_modules (used to call nest.Install())
- Note the data_path parameter is set automatically from the simulation's 'output_dir'

```
[22]: full_tree.children['kernel']
```

```
[22]: ParamsTree(name='kernel', parent='None')
  params:
    nest_seed: 10
    extension_modules: []
  nest_params:
    resolution: 0.5
```

(continues on next page)

(continued from previous page)

```
    overwrite_files: true
```

'session_models' subtree

The leaves of the 'session_models' subtree define templates from which the parameters of each Session object can be inherited.

The following params are recognized:

- simulation_time (float): Duration of the session in ms. (mandatory)
- reset_network (bool): If true, nest.ResetNetwork() is called during session initialization (default False)
- record (bool): If false, the start_time field of recorder nodes in NEST is set to the end time of the session, so that no data is recorded during the session (default True)
- shift_origin (bool): If True, the origin flag of the stimulation devices of all the network's InputLayer layers is set to the start of the session during initialization. Useful to repeat sessions when the stimulators are eg spike generators.
- unit_changes (list): List describing the changes applied to certain units before the start of the session. Passed to Network.set_state.
- synapse_changes (list): List describing the changes applied to certain synapses before the start of the session. Passed to Network.set_state. Refer to that method for a description of how synapse_changes is formatted and interpreted. No changes happen if empty. (default [])

[23]:
```
full_tree.children['session_models']
```

[23]:
```
ParamsTree(name='session_models', parent='None')
  params:
    record: true
    shift_origin: true
    simulation_time: 100.0
  nest_params: {}
  warmup:
    params:
      record: false
    nest_params: {}
  2_spikes:
    params:
      unit_changes:
      - layers:
        - input_layer
        population_name: spike_generator

  ... [3 lines] ...

          - 10.0
    nest_params: {}
  3_spikes:
    params:
      unit_changes:
      - layers:
        - input_layer
```

(continues on next page)

(continued from previous page)

```
        population_name: spike_generator
        nest_params:
          spike_times:
          - 1.0
          - 10.0
          - 20.0
    nest_params: {}
```

[24]:
```
print("Tree's leaves and their inherited parameters:")
[
    (f'leaf name: {l.name})',
      f'leaf `params`: {dict(l.params)}',
      f'leaf `nest_params`: {dict(l.nest_params)}')
     for l in full_tree.children['session_models'].leaves()
]
```

```
Tree's leaves and their inherited parameters:
```

[24]:
```
[('leaf name: warmup)',
  "leaf `params`: {'record': False, 'shift_origin': True, 'simulation_time': 100.0}",
  'leaf `nest_params`: {}'),
 ('leaf name: 2_spikes)',
  "leaf `params`: {'record': True, 'shift_origin': True, 'simulation_time': 100.0,
↪'unit_changes': [{'layers': ['input_layer'], 'population_name': 'spike_generator',
↪'nest_params': {'spike_times': [1.0, 10.0]}}]}",
  'leaf `nest_params`: {}'),
 ('leaf name: 3_spikes)',
  "leaf `params`: {'record': True, 'shift_origin': True, 'simulation_time': 100.0,
↪'unit_changes': [{'layers': ['input_layer'], 'population_name': 'spike_generator',
↪'nest_params': {'spike_times': [1.0, 10.0, 20.0]}}]}",
  'leaf `nest_params`: {}')]
```

Check out the `set_network_state` tutorial or the documentation of the `Network.set_state()` method for details on how to modify the state of the network at the start of a simulation

`'simulation'` subtree

The 'simulation' subtree contains the following parameters:

- `output_dir` (str): Path to the output directory (default 'output').
- `input_dir` (str): Path to the directory in which input files are searched for for each session. (default 'input')
- `sessions` (list(str)): Order in which sessions are run. Elements of the list should be the name of session models defined in the `session_models` parameter subtree (default [])

[25]:
```
full_tree.children['simulation']
```

[25]:
```
ParamsTree(name='simulation', parent='None')
  params:
    output_dir: data/outputs/output
    sessions:
    - warmup
    - 3_spikes
    - 2_spikes
    - 3_spikes
```

(continues on next page)

(continued from previous page)

```
  nest_params: {}
```

5.1.3 Run a full simulation

Load the full parameter tree

The full parameter tree can be loaded from a single file, or from multiple files which are merged.

From a single file by using the `yaml.load` or the `ParamsTree.read` methods:

[26]:
```
ParamsTree.read(PARAMS_DIR/'parameter_tree.yml')
```

[26]:
```
ParamsTree(name='None', parent=None)
  params: {}
  nest_params: {}
  kernel:
    params:
      nest_seed: 10
      extension_modules: []
    nest_params:
      resolution: 0.5
      overwrite_files: true
  simulation:
    params:
      output_dir: data/outputs/output
      sessions:
      - warmup
      - 3_spikes

  ... [169 lines] ...

        - layers:
          - input_layer
          populations: null
          model: my_spike_detector
        projection_recorders:
        - source_layers:
          - l1
          source_population: l1_exc
          target_layers:
          - l1
          target_population: l1_inh
          projection_model: proj_1_AMPA
          model: weight_recorder
      nest_params: {}
```

By merging multiple files, using the `denest.load_trees` method:

[27]:
```
!cat {PARAMS_DIR/'tree_paths.yml'}
```

```
- './network_tree.yml'
- './simulation.yml'
- './session_models.yml'
- './kernel.yml'
```

[28]:
```
full_tree = load_trees(PARAMS_DIR/'tree_paths.yml')
```

```
2020-06-30 13:51:21,241 [denest] INFO: Loading parameter file paths from data/params/
↪tree_paths.yml
2020-06-30 13:51:21,245 [denest] INFO: Finished loading parameter file paths
2020-06-30 13:51:21,260 [denest] INFO: Loading parameters files:
['./network_tree.yml',
 './simulation.yml',
 './session_models.yml',
 './kernel.yml']
```

[29]:
```
full_tree
```

[29]:
```
ParamsTree(name='data/params/tree_paths.yml', parent=None)
  params: {}
  nest_params: {}
  session_models:
    params:
      record: true
      shift_origin: true
      simulation_time: 100.0
    nest_params: {}
    3_spikes:
      params:
        unit_changes:
        - layers:
          - input_layer
          population_name: spike_generator
          nest_params:

  ... [168 lines] ...

          params:
            populations:
              l1_exc: 4
              l1_inh: 2
          nest_params: {}
  simulation:
    params:
      output_dir: ./data/outputs/output
      sessions:
      - warmup
      - 3_spikes
      - 2_spikes
      - 3_spikes
    nest_params: {}
```

Run with the `Simulation` object

We load the full parameter tree:

[30]:
```
tree = ParamsTree.read(PARAMS_DIR/'parameter_tree.yml')
```

And then initialize the `Simulation`:

[31]:
```
sim = Simulation(tree, output_dir=None, input_dir=None)
```

```
2020-06-30 13:51:21,596 [denest.utils.validation] INFO: Object `simulation`: params:␣
↪using default value for optional parameters:
{'input_dir': 'input'}
2020-06-30 13:51:21,598 [denest.simulation] INFO: Initializing NEST kernel and␣
↪seeds...
2020-06-30 13:51:21,600 [denest.simulation] INFO:   Resetting NEST kernel...
2020-06-30 13:51:21,609 [denest.simulation] INFO:   Setting NEST kernel status...
2020-06-30 13:51:21,609 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'resolution': 0.5, 'overwrite_files': True})`
2020-06-30 13:51:21,614 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'data_path': 'data/outputs/output/data', 'grng_seed': 11, 'rng_seeds': range(12,␣
↪13)})
2020-06-30 13:51:21,618 [denest.simulation] INFO:   Finished setting NEST kernel␣
↪status
2020-06-30 13:51:21,636 [denest.simulation] INFO:   Installing external modules...
2020-06-30 13:51:21,637 [denest.simulation] INFO:   Finished installing external␣
↪modules
2020-06-30 13:51:21,645 [denest.simulation] INFO: Finished initializing kernel
2020-06-30 13:51:21,646 [denest.simulation] INFO: Build N=3 session models
2020-06-30 13:51:21,650 [denest.simulation] INFO: Build N=4 sessions
2020-06-30 13:51:21,663 [denest.session] INFO: Creating session "00_warmup"
2020-06-30 13:51:21,671 [denest.utils.validation] INFO: Object `00_warmup`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': [], 'unit_changes': []}
2020-06-30 13:51:21,674 [denest.session] INFO: Creating session "01_3_spikes"
2020-06-30 13:51:21,678 [denest.utils.validation] INFO: Object `01_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:51:21,688 [denest.session] INFO: Creating session "02_2_spikes"
2020-06-30 13:51:21,715 [denest.utils.validation] INFO: Object `02_2_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:51:21,718 [denest.session] INFO: Creating session "03_3_spikes"
2020-06-30 13:51:21,720 [denest.utils.validation] INFO: Object `03_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:51:21,740 [denest.simulation] INFO: Sessions: ['00_warmup', '01_3_spikes
↪', '02_2_spikes', '03_3_spikes']
2020-06-30 13:51:21,744 [denest.simulation] INFO: Building network.
2020-06-30 13:51:21,772 [denest.network] INFO: Build N=2 ``Model`` objects
2020-06-30 13:51:21,778 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
2020-06-30 13:51:21,783 [denest.network] INFO: Build N=3 ``Model`` objects
2020-06-30 13:51:21,787 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:51:21,793 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:51:21,799 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:51:21,803 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
2020-06-30 13:51:21,819 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
2020-06-30 13:51:21,825 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:51:21,831 [denest.network] INFO: Build N=1 projection recorders.
2020-06-30 13:51:21,834 [denest.simulation] INFO: Creating network.
2020-06-30 13:51:21,838 [denest.network] INFO: Creating neuron models...
```

(continues on next page)

(continued from previous page)

```
100%|| 2/2 [00:00<00:00, 2590.68it/s]
2020-06-30 13:51:21,868 [denest.network] INFO: Creating synapse models...
100%|| 2/2 [00:00<00:00, 171.95it/s]
2020-06-30 13:51:21,889 [denest.network] INFO: Creating recorder models...
100%|| 3/3 [00:00<00:00, 345.82it/s]
2020-06-30 13:51:21,921 [denest.network] INFO: Creating layers...
  0%|          | 0/2 [00:00<?, ?it/s]/Users/tom/nest/nest-simulator-2.20.0/lib/
↪python3.7/site-packages/nest/lib/hl_api_helper.py:127: UserWarning:
GetNodes is deprecated and will be removed in NEST 3.0. Use             GIDCollection␣
↪instead.
100%|| 2/2 [00:00<00:00,  4.88it/s]
2020-06-30 13:51:22,365 [denest.network] INFO: Creating population recorders...
100%|| 2/2 [00:00<00:00, 92.98it/s]
2020-06-30 13:51:22,394 [denest.network] INFO: Creating projection recorders...
100%|| 1/1 [00:00<00:00, 509.39it/s]
2020-06-30 13:51:22,428 [denest.network] INFO: Connecting layers...
100%|| 3/3 [00:00<00:00, 623.69it/s]
2020-06-30 13:51:22,444 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 206
Number of projections: 6650
2020-06-30 13:51:22,446 [denest.simulation] INFO: Finished creating network
2020-06-30 13:51:22,447 [denest.simulation] INFO: Saving simulation metadata...
2020-06-30 13:51:22,448 [denest.simulation] INFO: Creating output directory: data/
↪outputs/output
2020-06-30 13:51:22,451 [denest.io.save] INFO: Clearing directory: data/outputs/output
2020-06-30 13:51:22,462 [denest.io.save] INFO: Clearing directory: data/outputs/output
2020-06-30 13:51:22,463 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output/data
2020-06-30 13:51:22,468 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output/data
2020-06-30 13:51:22,474 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output/data
2020-06-30 13:51:22,481 [denest.io.save] INFO: Clearing directory: data/outputs/output
2020-06-30 13:51:22,566 [denest.simulation] INFO: Finished saving simulation metadata
```

We can inspect the simulation elements:

[32]:
```
sim.network
```

[32]:
```
Network(params: {}
nest_params: {}
neuron_models:
  params: {}
  nest_params: {}
  my_neuron:
    params:
      nest_model: ht_neuron
    nest_params:
      g_KL: 1.0
      g_NaL: 1.0
    l1_exc:
      params: {}
      nest_params:
        V_m: -44.0
    l1_inh:
      params: {}
```

(continues on next page)

(continued from previous page)

```
      nest_params:
        V_m: -55.0
synapse_models:
  params: {}
  nest_params: {}
  my_AMPA_synapse:
    params:
      nest_model: ht_synapse
      receptor_type: AMPA
      target_neuron: ht_neuron
    nest_params: {}
  my_GABAA_synapse:
    params:
      nest_model: ht_synapse
      receptor_type: GABA_A
      target_neuron: ht_neuron
    nest_params: {}
layers:
  params: {}
  nest_params: {}
  layers:
    params:
      type: null
    nest_params:
      rows: 5
      columns: 5
      extent:
      - 5.0
      - 5.0
      edge_wrap: true
    input_layer:
      params:
        type: InputLayer
        add_parrots: true
        populations:
          spike_generator: 1
      nest_params: {}
    l1:
      params:
        populations:
          l1_exc: 4
          l1_inh: 2
      nest_params: {}
projection_models:
  params: {}
  nest_params:
    connection_type: divergent
    mask:
      circular:
        radius: 2.0
    kernel: 1.0
  proj_1_AMPA:
    params: {}
    nest_params:
      synapse_model: my_AMPA_synapse
      weights: 1.0
  proj_2_GABAA:
```

(continues on next page)

(continued from previous page)

```
    params: {}
    nest_params:
      synapse_model: my_GABAA_synapse
      weights: 2.0
topology:
  params:
    projections:
    - source_layers:
      - input_layer
      source_population: parrot_neuron
      target_layers:
      - l1
      target_population: l1_exc
      projection_model: proj_1_AMPA
    - source_layers:
      - l1
      source_population: l1_exc
      target_layers:
      - l1
      target_population: l1_inh
      projection_model: proj_1_AMPA
    - source_layers:
      - l1
      source_population: l1_inh
      target_layers:
      - l1
      target_population: l1_exc
      projection_model: proj_2_GABAA
  nest_params: {}
recorder_models:
  params: {}
  nest_params:
    record_to:
    - memory
    - file
  weight_recorder:
    params:
      nest_model: weight_recorder
    nest_params: {}
  my_multimeter:
    params:
      nest_model: multimeter
    nest_params:
      record_from:
      - V_m
      interval: 20.0
  my_spike_detector:
    params:
      nest_model: spike_detector
    nest_params: {}
recorders:
  params:
    population_recorders:
    - layers:
      - l1
      populations:
      - l1_exc
```

(continues on next page)

(continued from previous page)

```
      model: my_multimeter
    - layers:
      - input_layer
      populations: null
      model: my_spike_detector
    projection_recorders:
    - source_layers:
      - l1
      source_population: l1_exc
      target_layers:
      - l1
      target_population: l1_inh
      projection_model: proj_1_AMPA
      model: weight_recorder
  nest_params: {}
)
```

[33]:
```
sim.network.layers
```

[33]:
```
{'input_layer': InputLayer(input_layer,
  {'type': 'InputLayer',
   'add_parrots': True,
   'populations': {'spike_generator': 1, 'parrot_neuron': 1}}{'rows': 5,
   'columns': 5,
   'extent': [5.0, 5.0],
   'edge_wrap': True,
   'elements': ('spike_generator', 1, 'parrot_neuron', 1)}),
 'l1': Layer(l1,
  {'type': None, 'populations': {'l1_exc': 4, 'l1_inh': 2}}{'rows': 5,
   'columns': 5,
   'extent': [5.0, 5.0],
   'edge_wrap': True,
   'elements': ('l1_exc', 4, 'l1_inh', 2)})}
```

[34]:
```
sim.sessions
```

[34]:
```
[Session(00_warmup, {'record': False,
  'reset_network': False,
  'shift_origin': True,
  'simulation_time': 100.0,
  'synapse_changes': [],
  'unit_changes': []}),
 Session(01_3_spikes, {'record': True,
  'reset_network': False,
  'shift_origin': True,
  'simulation_time': 100.0,
  'synapse_changes': [],
  'unit_changes': [{'layers': ['input_layer'],
                    'nest_params': {'spike_times': [1.0, 10.0, 20.0]},
                    'population_name': 'spike_generator'}]}),
 Session(02_2_spikes, {'record': True,
  'reset_network': False,
  'shift_origin': True,
  'simulation_time': 100.0,
  'synapse_changes': [],
  'unit_changes': [{'layers': ['input_layer'],
                    'nest_params': {'spike_times': [1.0, 10.0]},
```

(continues on next page)

(continued from previous page)

```
                    'population_name': 'spike_generator'}]}),
 Session(03_3_spikes, {'record': True,
  'reset_network': False,
  'shift_origin': True,
  'simulation_time': 100.0,
  'synapse_changes': [],
  'unit_changes': [{'layers': ['input_layer'],
                    'nest_params': {'spike_times': [1.0, 10.0, 20.0]},
                    'population_name': 'spike_generator'}]})]
```

Finally, we actually run all the sessions in NEST by calling `Simulation.run()`:

[35]:
```
sim.run()
```

```
2020-06-30 13:51:22,828 [denest.simulation] INFO: Running 4 sessions...
2020-06-30 13:51:22,831 [denest.simulation] INFO: Running session: '00_warmup'...
2020-06-30 13:51:22,838 [denest.session] INFO: Initializing session...
2020-06-30 13:51:22,840 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_multimeter_l1_l1_exc: {'start': 100.0}
2020-06-30 13:51:22,843 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_spike_detector_input_layer_parrot_neuron: {'start': 100.0}
2020-06-30 13:51:22,846 [denest.network.recorders] INFO:   Setting status for␣
↪recorder weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh: {'start': 100.0}
2020-06-30 13:51:22,847 [denest.session] INFO: Setting `origin` flag to `0.0` for all␣
↪stimulation devices in ``InputLayers`` for session `00_warmup`
2020-06-30 13:51:22,863 [denest.session] INFO: Finished initializing session

2020-06-30 13:51:22,864 [denest.session] INFO: Running session '00_warmup' for 100 ms
2020-06-30 13:51:23,077 [denest.session] INFO: Finished running session
2020-06-30 13:51:23,079 [denest.session] INFO: Session '00_warmup' virtual running␣
↪time: 100 ms
2020-06-30 13:51:23,284 [denest.session] INFO: Session '00_warmup' real running time:␣
↪0h:00m:00s
2020-06-30 13:51:23,292 [denest.simulation] INFO: Done running session '00_warmup'
2020-06-30 13:51:23,298 [denest.simulation] INFO: Running session: '01_3_spikes'...
2020-06-30 13:51:23,301 [denest.session] INFO: Initializing session...
2020-06-30 13:51:23,305 [denest.session] INFO: Setting `origin` flag to `100.0` for␣
↪all stimulation devices in ``InputLayers`` for session `01_3_spikes`
2020-06-30 13:51:23,328 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:51:23,330 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:51:23,523 [denest.session] INFO: Finished initializing session

2020-06-30 13:51:23,524 [denest.session] INFO: Running session '01_3_spikes' for 100␣
↪ms
2020-06-30 13:51:23,746 [denest.session] INFO: Finished running session
2020-06-30 13:51:23,747 [denest.session] INFO: Session '01_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:51:23,761 [denest.session] INFO: Session '01_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:51:23,764 [denest.simulation] INFO: Done running session '01_3_spikes'
2020-06-30 13:51:23,772 [denest.simulation] INFO: Running session: '02_2_spikes'...
2020-06-30 13:51:23,777 [denest.session] INFO: Initializing session...
2020-06-30 13:51:23,777 [denest.session] INFO: Setting `origin` flag to `200.0` for␣
↪all stimulation devices in ``InputLayers`` for session `02_2_spikes`
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:51:23,791 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:51:23,795 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:51:23,985 [denest.session] INFO: Finished initializing session

2020-06-30 13:51:23,986 [denest.session] INFO: Running session '02_2_spikes' for 100␣
↪ms
2020-06-30 13:51:24,336 [denest.session] INFO: Finished running session
2020-06-30 13:51:24,337 [denest.session] INFO: Session '02_2_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:51:24,338 [denest.session] INFO: Session '02_2_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:51:24,339 [denest.simulation] INFO: Done running session '02_2_spikes'
2020-06-30 13:51:24,341 [denest.simulation] INFO: Running session: '03_3_spikes'...
2020-06-30 13:51:24,344 [denest.session] INFO: Initializing session...
2020-06-30 13:51:24,345 [denest.session] INFO: Setting `origin` flag to `300.0` for␣
↪all stimulation devices in ``InputLayers`` for session `03_3_spikes`
2020-06-30 13:51:24,352 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:51:24,371 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:51:24,470 [denest.session] INFO: Finished initializing session

2020-06-30 13:51:24,471 [denest.session] INFO: Running session '03_3_spikes' for 100␣
↪ms
2020-06-30 13:51:24,579 [denest.session] INFO: Finished running session
2020-06-30 13:51:24,580 [denest.session] INFO: Session '03_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:51:24,581 [denest.session] INFO: Session '03_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:51:24,597 [denest.simulation] INFO: Done running session '03_3_spikes'
2020-06-30 13:51:24,604 [denest.simulation] INFO: Finished running simulation
```

[36]:
```
!ls {sim.output_dir}
```

```
data                parameter_tree.yml session_times.yml  versions.txt
```

[37]:
```
!ls {Path(sim.output_dir)/'data'}
```

```
my_multimeter_l1_l1_exc-203-0.dat
my_multimeter_l1_l1_exc.yml
my_spike_detector_input_layer_parrot_neuron-204-0.gdf
my_spike_detector_input_layer_parrot_neuron.yml
weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh-205-0.csv
weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh.yml
```

Run using the `denest.run` function

The `run` function is a shorthand for the above steps:

[38]:
```
denest.run(PARAMS_DIR/'tree_paths.yml')
```

```
2020-06-30 13:51:25,053 [denest] INFO:

=== RUNNING SIMULATION ========================================================

2020-06-30 13:51:25,055 [denest] INFO: Loading parameter file paths from data/params/
↪tree_paths.yml
2020-06-30 13:51:25,061 [denest] INFO: Finished loading parameter file paths
2020-06-30 13:51:25,063 [denest] INFO: Loading parameters files:
['./network_tree.yml',
 './simulation.yml',
 './session_models.yml',
 './kernel.yml']
2020-06-30 13:51:25,119 [denest] INFO: Initializing simulation...
2020-06-30 13:51:25,146 [denest.utils.validation] INFO: Object `simulation`: params:␣
↪using default value for optional parameters:
{'input_dir': 'input'}
2020-06-30 13:51:25,149 [denest.simulation] INFO: Initializing NEST kernel and␣
↪seeds...
2020-06-30 13:51:25,149 [denest.simulation] INFO:   Resetting NEST kernel...
2020-06-30 13:51:25,163 [denest.simulation] INFO:   Setting NEST kernel status...
2020-06-30 13:51:25,257 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'resolution': 0.5, 'overwrite_files': True})`
2020-06-30 13:51:25,263 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'data_path': 'data/outputs/output/data', 'grng_seed': 11, 'rng_seeds': range(12,␣
↪13)})
2020-06-30 13:51:25,304 [denest.simulation] INFO:   Finished setting NEST kernel␣
↪status
2020-06-30 13:51:25,307 [denest.simulation] INFO:   Installing external modules...
2020-06-30 13:51:25,321 [denest.simulation] INFO:   Finished installing external␣
↪modules
2020-06-30 13:51:25,334 [denest.simulation] INFO: Finished initializing kernel
2020-06-30 13:51:25,337 [denest.simulation] INFO: Build N=3 session models
2020-06-30 13:51:25,340 [denest.simulation] INFO: Build N=4 sessions
2020-06-30 13:51:25,346 [denest.session] INFO: Creating session "00_warmup"
2020-06-30 13:51:25,349 [denest.utils.validation] INFO: Object `00_warmup`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': [], 'unit_changes': []}
2020-06-30 13:51:25,352 [denest.session] INFO: Creating session "01_3_spikes"
2020-06-30 13:51:25,360 [denest.utils.validation] INFO: Object `01_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:51:25,362 [denest.session] INFO: Creating session "02_2_spikes"
2020-06-30 13:51:25,368 [denest.utils.validation] INFO: Object `02_2_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:51:25,371 [denest.session] INFO: Creating session "03_3_spikes"
2020-06-30 13:51:25,373 [denest.utils.validation] INFO: Object `03_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:51:25,379 [denest.simulation] INFO: Sessions: ['00_warmup', '01_3_spikes
↪', '02_2_spikes', '03_3_spikes']
2020-06-30 13:51:25,384 [denest.simulation] INFO: Building network.
2020-06-30 13:51:25,408 [denest.network] INFO: Build N=2 ``Model`` objects
2020-06-30 13:51:25,414 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
2020-06-30 13:51:25,417 [denest.network] INFO: Build N=3 ``Model`` objects
2020-06-30 13:51:25,422 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:51:25,430 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:51:25,434 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:51:25,435 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
2020-06-30 13:51:25,440 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
2020-06-30 13:51:25,450 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:51:25,452 [denest.network] INFO: Build N=1 projection recorders.
2020-06-30 13:51:25,453 [denest.simulation] INFO: Creating network.
2020-06-30 13:51:25,456 [denest.network] INFO: Creating neuron models...
100%|| 2/2 [00:00<00:00, 1308.06it/s]
2020-06-30 13:51:25,465 [denest.network] INFO: Creating synapse models...
100%|| 2/2 [00:00<00:00, 1308.06it/s]
2020-06-30 13:51:25,473 [denest.network] INFO: Creating recorder models...
100%|| 3/3 [00:00<00:00, 1403.72it/s]
2020-06-30 13:51:25,483 [denest.network] INFO: Creating layers...
100%|| 2/2 [00:00<00:00,  9.27it/s]
2020-06-30 13:51:25,709 [denest.network] INFO: Creating population recorders...
100%|| 2/2 [00:00<00:00, 92.20it/s]
2020-06-30 13:51:25,736 [denest.network] INFO: Creating projection recorders...
100%|| 1/1 [00:00<00:00, 322.39it/s]
2020-06-30 13:51:25,746 [denest.network] INFO: Connecting layers...
100%|| 3/3 [00:00<00:00, 724.70it/s]
2020-06-30 13:51:25,771 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 206
Number of projections: 6650
2020-06-30 13:51:25,773 [denest.simulation] INFO: Finished creating network
2020-06-30 13:51:25,780 [denest.simulation] INFO: Saving simulation metadata...
2020-06-30 13:51:25,780 [denest.simulation] INFO: Creating output directory: ./data/
↪outputs/output
2020-06-30 13:51:25,786 [denest.io.save] INFO: Clearing directory: data/outputs/output
2020-06-30 13:51:25,795 [denest.io.save] INFO: Clearing directory: data/outputs/output
2020-06-30 13:51:25,804 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output/data
2020-06-30 13:51:25,815 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output/data
2020-06-30 13:51:25,816 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output/data
2020-06-30 13:51:25,832 [denest.io.save] INFO: Clearing directory: data/outputs/output
2020-06-30 13:51:25,912 [denest.simulation] INFO: Finished saving simulation metadata
2020-06-30 13:51:25,914 [denest] INFO: Finished initializing simulation
2020-06-30 13:51:25,915 [denest] INFO: Running simulation...
2020-06-30 13:51:25,922 [denest.simulation] INFO: Running 4 sessions...
2020-06-30 13:51:25,932 [denest.simulation] INFO: Running session: '00_warmup'...
2020-06-30 13:51:25,935 [denest.session] INFO: Initializing session...
2020-06-30 13:51:25,937 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_multimeter_l1_l1_exc: {'start': 100.0}
2020-06-30 13:51:25,950 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_spike_detector_input_layer_parrot_neuron: {'start': 100.0}
2020-06-30 13:51:25,956 [denest.network.recorders] INFO:   Setting status for␣
↪recorder weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh: {'start': 100.0}
2020-06-30 13:51:25,959 [denest.session] INFO: Setting `origin` flag to `0.0` for all␣
↪stimulation devices in ``InputLayers`` for session `00_warmup`
2020-06-30 13:51:25,974 [denest.session] INFO: Finished initializing session
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:51:25,976 [denest.session] INFO: Running session '00_warmup' for 100 ms
2020-06-30 13:51:26,173 [denest.session] INFO: Finished running session
2020-06-30 13:51:26,175 [denest.session] INFO: Session '00_warmup' virtual running␣
↪time: 100 ms
2020-06-30 13:51:26,180 [denest.session] INFO: Session '00_warmup' real running time:␣
↪0h:00m:00s
2020-06-30 13:51:26,181 [denest.simulation] INFO: Done running session '00_warmup'
2020-06-30 13:51:26,190 [denest.simulation] INFO: Running session: '01_3_spikes'...
2020-06-30 13:51:26,191 [denest.session] INFO: Initializing session...
2020-06-30 13:51:26,214 [denest.session] INFO: Setting `origin` flag to `100.0` for␣
↪all stimulation devices in ``InputLayers`` for session `01_3_spikes`
2020-06-30 13:51:26,258 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:51:26,260 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:51:26,445 [denest.session] INFO: Finished initializing session

2020-06-30 13:51:26,446 [denest.session] INFO: Running session '01_3_spikes' for 100␣
↪ms
2020-06-30 13:51:26,754 [denest.session] INFO: Finished running session
2020-06-30 13:51:26,754 [denest.session] INFO: Session '01_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:51:26,755 [denest.session] INFO: Session '01_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:51:26,758 [denest.simulation] INFO: Done running session '01_3_spikes'
2020-06-30 13:51:26,762 [denest.simulation] INFO: Running session: '02_2_spikes'...
2020-06-30 13:51:26,764 [denest.session] INFO: Initializing session...
2020-06-30 13:51:26,770 [denest.session] INFO: Setting `origin` flag to `200.0` for␣
↪all stimulation devices in ``InputLayers`` for session `02_2_spikes`
2020-06-30 13:51:26,815 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:51:26,816 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:51:26,885 [denest.session] INFO: Finished initializing session

2020-06-30 13:51:26,886 [denest.session] INFO: Running session '02_2_spikes' for 100␣
↪ms
2020-06-30 13:51:27,046 [denest.session] INFO: Finished running session
2020-06-30 13:51:27,047 [denest.session] INFO: Session '02_2_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:51:27,048 [denest.session] INFO: Session '02_2_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:51:27,063 [denest.simulation] INFO: Done running session '02_2_spikes'
2020-06-30 13:51:27,069 [denest.simulation] INFO: Running session: '03_3_spikes'...
2020-06-30 13:51:27,083 [denest.session] INFO: Initializing session...
2020-06-30 13:51:27,091 [denest.session] INFO: Setting `origin` flag to `300.0` for␣
↪all stimulation devices in ``InputLayers`` for session `03_3_spikes`
2020-06-30 13:51:27,097 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:51:27,101 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:51:27,201 [denest.session] INFO: Finished initializing session
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:51:27,202 [denest.session] INFO: Running session '03_3_spikes' for 100␣
↪ms
2020-06-30 13:51:27,455 [denest.session] INFO: Finished running session
2020-06-30 13:51:27,459 [denest.session] INFO: Session '03_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:51:27,475 [denest.session] INFO: Session '03_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:51:27,517 [denest.simulation] INFO: Done running session '03_3_spikes'
2020-06-30 13:51:27,519 [denest.simulation] INFO: Finished running simulation
2020-06-30 13:51:27,530 [denest] INFO: Finished running simulation
2020-06-30 13:51:27,543 [denest] INFO: Total simulation virtual time: 400.0 ms
2020-06-30 13:51:27,564 [denest] INFO: Total simulation real time: 0h:00m:02s
2020-06-30 13:51:27,567 [denest] INFO: Simulation output written to: /Users/tom/
↪docker/nets-dev/docs/source/tutorials/data/outputs/output
```

Run from the command line

A simulation can also be run directly from the command line as follows:

```
python3 -m denest <tree_paths.yml> [-o <output_dir>]
```

5.2 The `Network` object

We'll familiarize ourselves with the `Network` object by interactively building the elements of the network one by one.

Usually, we'd initialize all the elements of the network at once by providing the full `'network'` tree as an argument to the `Network` class during initialization.

In this tutorial we:

1. Initialize an empty `Network` object
2. **Initialize all the elements of the network** using the `Network.build_*` methods
    1. Models (neuron models, recorder models, synapse models
    2. Layers
    3. Projection models
    4. Individual projections
    5. Population and projection recorders
3. **Create the network** in NEST
4. **Access the network elements** (GIDs, etc)
5. Export and reuse the parameter tree allowing us to **replicate the network**

[1]:
```
import nest
import yaml
from pathlib import Path
from pprint import pprint
```

[2]:
```
from denest import *
import denest
```

[3]:
```
PARAMS_DIR = Path('./data/params/network')
```

5.2.1 Initialize an empty network

When initialized without argument or with an empty tree as an argument, all the expected subtrees are initialized as empty.

When building the elements interactively, the network's parameters are updated.

[4]:
```
net = Network()
```

```
2020-06-30 13:42:37,794 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child neuron_models
2020-06-30 13:42:37,795 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child synapse_models
2020-06-30 13:42:37,796 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child layers
2020-06-30 13:42:37,797 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child projection_models
2020-06-30 13:42:37,799 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child topology
2020-06-30 13:42:37,800 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child recorder_models
2020-06-30 13:42:37,803 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child recorders
2020-06-30 13:42:37,806 [denest.network] INFO: Build N=0 ``Model`` objects
2020-06-30 13:42:37,809 [denest.network] INFO: Build N=0 ``SynapseModel`` objects
2020-06-30 13:42:37,809 [denest.network] INFO: Build N=0 ``Model`` objects
2020-06-30 13:42:37,812 [denest.network] INFO: Build N=0 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:42:37,814 [denest.network] INFO: Build N=0 ``ProjectionModel`` objects
2020-06-30 13:42:37,815 [denest.utils.validation] INFO: Object `topology`: params:␣
↪using default value for optional parameters:
{'projections': []}
2020-06-30 13:42:37,816 [denest.network] INFO: Build N=0 ``TopoProjection`` objects
2020-06-30 13:42:37,819 [denest.utils.validation] INFO: Object `recorders`: params:␣
↪using default value for optional parameters:
{'population_recorders': [], 'projection_recorders': []}
2020-06-30 13:42:37,823 [denest.network] INFO: Build N=0 population recorders.
2020-06-30 13:42:37,824 [denest.network] INFO: Build N=0 projection recorders.
```

The `Network`'s parameter tree is empty:

[5]:
```
net.tree
```

[5]:
```
ParamsTree(name='None', parent=None)
  params: {}
  nest_params: {}
  neuron_models:
    params: {}
    nest_params: {}
  synapse_models:
    params: {}
    nest_params: {}
  layers:
    params: {}
    nest_params: {}
```

(continues on next page)

(continued from previous page)

```
  projection_models:
    params: {}
    nest_params: {}
  topology:
    params: {}
    nest_params: {}
  recorder_models:
    params: {}
    nest_params: {}
  recorders:
    params: {}
    nest_params: {}
```

5.2.2 Build the network components (models, layers, projections, recorders)

Define new models

We can define neuron, recorder, stimulator and synapse models with arbitrary parameters from parameter trees.

Each leaf corresponds to a new (named) model. Its `nest_params` and `params` are hierarchically inherited.

The `nest_model` used is specified in the leaf's `params`

`'neuron_models'` tree:

Initalize `Network.neuron_models` with the `Network.build_neuron_models` method

[6]:
```
neuron_models_tree = ParamsTree.read(PARAMS_DIR/'models.yml').children['neuron_models
↪']
pprint(neuron_models_tree)
```

```
ParamsTree(name='neuron_models', parent='None')
  params: {}
  nest_params: {}
  my_neuron:
    params:
      nest_model: ht_neuron
    nest_params:
      g_KL: 1.0
      g_NaL: 1.0
    l1_exc:
      params: {}
      nest_params:
        V_m: -44.0
    l1_inh:
      params: {}
      nest_params:
        V_m: -55.0
```

[7]:
```
net.build_neuron_models(neuron_models_tree)
```

```
2020-06-30 13:42:38,050 [denest.network] INFO: Build N=2 ``Model`` objects
```

[8]:
```
# The neuron models are saved as an attribute for the Network object
print("\n``Network.neuron_models`` :")
pprint(net.neuron_models)
```

```
``Network.neuron_models`` :
{'l1_exc': Model(l1_exc, {'nest_model': 'ht_neuron'}, {'V_m': -44.0, 'g_KL': 1.0, 'g_
↪NaL': 1.0}),
 'l1_inh': Model(l1_inh, {'nest_model': 'ht_neuron'}, {'V_m': -55.0, 'g_KL': 1.0, 'g_
↪NaL': 1.0})}
```

[9]:
```
pprint(net.neuron_models['l1_exc'])
pprint(net.neuron_models['l1_exc'].params)
pprint(net.neuron_models['l1_exc'].nest_params)
```

```
Model(l1_exc, {'nest_model': 'ht_neuron'}, {'V_m': -44.0, 'g_KL': 1.0, 'g_NaL': 1.0})
{'nest_model': 'ht_neuron'}
{'V_m': -44.0, 'g_KL': 1.0, 'g_NaL': 1.0}
```

`'recorder_models'` tree

Initalize `Network.recorder_models` with the `Network.build_recorder_models` method, as with neuron models:

[10]:
```
# ``Network.build_*`` methods accept as argument ``ParamsTree`` objects, but also␣
↪tree-like dictionaries
recorder_models_tree = ParamsTree.read(PARAMS_DIR/'models.yml').children['recorder_
↪models']
recorder_models_tree
```

[10]:
```
ParamsTree(name='recorder_models', parent='None')
  params: {}
  nest_params:
    record_to:
    - memory
    - file
  weight_recorder:
    params:
      nest_model: weight_recorder
    nest_params: {}
  my_multimeter:
    params:
      nest_model: multimeter
    nest_params:
      record_from:
      - V_m
      interval: 20.0
  my_spike_detector:
    params:
      nest_model: spike_detector
    nest_params: {}
```

[11]:
```
# ``Network.build_*`` methods accept as argument ``ParamsTree`` objects, but also␣
↪tree-like dictionaries
recorder_models_tree = recorder_models_tree.asdict()
recorder_models_tree
```

[11]:
```
{'params': {},
 'nest_params': {'record_to': ['memory', 'file']},
 'weight_recorder': {'params': {'nest_model': 'weight_recorder'},
  'nest_params': {}},
 'my_multimeter': {'params': {'nest_model': 'multimeter'},
  'nest_params': {'record_from': ['V_m'], 'interval': 20.0}},
 'my_spike_detector': {'params': {'nest_model': 'spike_detector'},
  'nest_params': {}}}
```

[12]:
```
net.build_recorder_models(recorder_models_tree)
```

```
2020-06-30 13:42:38,389 [denest.network] INFO: Build N=3 ``Model`` objects
```

[13]:
```
print("\n``Network.recorder_models`` :")
pprint(net.recorder_models)
```

```
``Network.recorder_models`` :
{'my_multimeter': Model(my_multimeter, {'nest_model': 'multimeter'}, {'interval': 20.
↪0, 'record_from': ['V_m'], 'record_to': ['memory', 'file']}),
 'my_spike_detector': Model(my_spike_detector, {'nest_model': 'spike_detector'}, {
↪'record_to': ['memory', 'file']}),
 'weight_recorder': Model(weight_recorder, {'nest_model': 'weight_recorder'}, {
↪'record_to': ['memory', 'file']})}
```

`'synapse_model'` tree

Initalize `Network.synapse_models` with the `Network.build_synapse_model` method.

- Same as for neuron models, with as a bonus a convenient way of specifying the receptor type of the synapse.
- If specifying the `receptor_type` and `target_model` in the `SynapseModel` params, the corresponding port is determined automatically.

[14]:
```
synapse_models_tree = ParamsTree.read(PARAMS_DIR/'models.yml').children['synapse_
↪models']
synapse_models_tree
```

[14]:
```
ParamsTree(name='synapse_models', parent='None')
  params: {}
  nest_params: {}
  my_AMPA_synapse:
    params:
      nest_model: ht_synapse
      receptor_type: AMPA
      target_neuron: ht_neuron
    nest_params: {}
  my_GABAA_synapse:
    params:
      nest_model: ht_synapse
      receptor_type: GABA_A
      target_neuron: ht_neuron
    nest_params: {}
```

[15]:
```
net.build_synapse_models(synapse_models_tree)
```

```
2020-06-30 13:42:38,570 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
```

[16]:
```
print("\n``Network.synapse_models`` :")
pprint(net.synapse_models)
```

```
``Network.synapse_models`` :
{'my_AMPA_synapse': SynapseModel(my_AMPA_synapse, {'nest_model': 'ht_synapse'}, {
↪'receptor_type': 1}),
 'my_GABAA_synapse': SynapseModel(my_GABAA_synapse, {'nest_model': 'ht_synapse'}, {
↪'receptor_type': 3})}
```

Note that the `receptor_type` nest_parameter was inferred

Define layers

As for models, we can create `nest.Topology` layers from the leaves of a tree.

- The elements can be nest models with their default parameters, or the ones we just created with custom params.
- For layers of stimulator devices, we can use the `InputLayer` object, which can automatically create paired parrot neurons for each stimulator units, by adding `type: 'InputLayer'` to the params

`'layers'` tree

[17]:
```
layer_tree = ParamsTree.read(PARAMS_DIR/'layers.yml')
layer_tree
```

[17]:
```
ParamsTree(name='None', parent=None)
  params: {}
  nest_params: {}
  layers:
    params:
      type: null
    nest_params:
      rows: 5
      columns: 5
      extent:
      - 5.0
      - 5.0
      edge_wrap: true
    input_layer:
      params:
        type: InputLayer
        add_parrots: true
        populations:
          spike_generator: 1
      nest_params: {}
    l1:
      params:
        populations:
          l1_exc: 4
          l1_inh: 2
      nest_params: {}
```

[18]:
```
net.build_layers(layer_tree)
```

```
2020-06-30 13:42:38,801 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
```

[19]:
```
pprint(net.layers)
```

```
{'input_layer': InputLayer(input_layer, {'add_parrots': True,
 'populations': {'parrot_neuron': 1, 'spike_generator': 1},
 'type': 'InputLayer'}, {'columns': 5,
 'edge_wrap': True,
 'elements': ('spike_generator', 1, 'parrot_neuron', 1),
 'extent': [5.0, 5.0],
 'rows': 5}),
 'l1': Layer(l1, {'populations': {'l1_exc': 4, 'l1_inh': 2}, 'type': None}, {'columns
↪': 5,
 'edge_wrap': True,
 'elements': ('l1_exc', 4, 'l1_inh', 2),
 'extent': [5.0, 5.0],
 'rows': 5})}
```

[20]:
```
print("'l1' layer")
pprint(net.layers['l1'].params)
pprint(net.layers['l1'].nest_params)
```

```
'l1' layer
{'populations': {'l1_exc': 4, 'l1_inh': 2}, 'type': None}
{'columns': 5,
 'edge_wrap': True,
 'elements': ('l1_exc', 4, 'l1_inh', 2),
 'extent': [5.0, 5.0],
 'rows': 5}
```

Define projections

We create projections using a two step process:

1. Create `ProjectionModel` objects from a tree. Each named leaf will define a template from which individual projections can inherit their parameters
2. Create `Projection` objects from a list, specifying for each item the source layer x population, target layer x population and the projection model to inherit parameters from

Define templates from the `projection_models` tree

[21]:
```
proj_model_tree = ParamsTree.read(PARAMS_DIR/'projections.yml').children['projection_
↪models']
proj_model_tree
```

[21]:
```
ParamsTree(name='projection_models', parent='None')
  params: {}
  nest_params:
    connection_type: divergent
    mask:
      circular:
```

(continues on next page)

(continued from previous page)

```
        radius: 2.0
    kernel: 1.0
  proj_1_AMPA:
    params: {}
    nest_params:
      synapse_model: my_AMPA_synapse
      weights: 1.0
  proj_2_GABAA:
    params: {}
    nest_params:
      synapse_model: my_GABAA_synapse
      weights: 2.0
```

[22]:
```
net.build_projection_models(proj_model_tree)
```

```
2020-06-30 13:42:39,014 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:42:39,021 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:42:39,022 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
```

[23]:
```
net.projection_models
```

[23]:
```
{'proj_1_AMPA': ProjectionModel(proj_1_AMPA,
  {'type': 'topological'}{'connection_type': 'divergent',
   'mask': {'circular': {'radius': 2.0}},
   'kernel': 1.0,
   'synapse_model': 'my_AMPA_synapse',
   'weights': 1.0}),
 'proj_2_GABAA': ProjectionModel(proj_2_GABAA,
  {'type': 'topological'}{'connection_type': 'divergent',
   'mask': {'circular': {'radius': 2.0}},
   'kernel': 1.0,
   'synapse_model': 'my_GABAA_synapse',
   'weights': 2.0})}
```

Define individual projections from the `topology` tree

The list of projections is defined in the `projections` params of the `topology` tree

Check out the doc of `Network.build_projections` for expected formatting

[24]:
```
conns_tree = ParamsTree.read(PARAMS_DIR/'projections.yml').children['topology']
conns_tree
```

[24]:
```
ParamsTree(name='topology', parent='None')
  params:
    projections:
    - source_layers:
      - input_layer
      source_population: parrot_neuron
      target_layers:
      - l1
```

(continues on next page)

(continued from previous page)

```
      target_population: l1_exc
      projection_model: proj_1_AMPA
    - source_layers:
      - l1
      source_population: l1_exc
      target_layers:
      - l1
      target_population: l1_inh
      projection_model: proj_1_AMPA
    - source_layers:
      - l1
      source_population: l1_inh
      target_layers:
      - l1
      target_population: l1_exc
      projection_model: proj_2_GABAA
  nest_params: {}
```

[25]:
```
net.projection_models
```

[25]:
```
{'proj_1_AMPA': ProjectionModel(proj_1_AMPA,
  {'type': 'topological'}{'connection_type': 'divergent',
   'mask': {'circular': {'radius': 2.0}},
   'kernel': 1.0,
   'synapse_model': 'my_AMPA_synapse',
   'weights': 1.0}),
 'proj_2_GABAA': ProjectionModel(proj_2_GABAA,
  {'type': 'topological'}{'connection_type': 'divergent',
   'mask': {'circular': {'radius': 2.0}},
   'kernel': 1.0,
   'synapse_model': 'my_GABAA_synapse',
   'weights': 2.0})}
```

[26]:
```
net.build_projections(conns_tree)
```

```
2020-06-30 13:42:39,277 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
```

[27]:
```
net.projections
```

[27]:
```
[TopoProjection(proj_1_AMPA-input_layer-parrot_neuron-l1-l1_exc,
  {'type': 'topological'}{'connection_type': 'divergent',
   'mask': {'circular': {'radius': 2.0}},
   'kernel': 1.0,
   'synapse_model': 'my_AMPA_synapse',
   'weights': 1.0,
   'sources': {'model': 'parrot_neuron'},
   'targets': {'model': 'l1_exc'}}),
 TopoProjection(proj_1_AMPA-l1-l1_exc-l1-l1_inh,
  {'type': 'topological'}{'connection_type': 'divergent',
   'mask': {'circular': {'radius': 2.0}},
   'kernel': 1.0,
   'synapse_model': 'my_AMPA_synapse',
   'weights': 1.0,
   'sources': {'model': 'l1_exc'},
   'targets': {'model': 'l1_inh'}}),
 TopoProjection(proj_2_GABAA-l1-l1_inh-l1-l1_exc,
```

(continues on next page)

(continued from previous page)

```
 {'type': 'topological'}{'connection_type': 'divergent',
  'mask': {'circular': {'radius': 2.0}},
  'kernel': 1.0,
  'synapse_model': 'my_GABAA_synapse',
  'weights': 2.0,
  'sources': {'model': 'l1_inh'},
  'targets': {'model': 'l1_exc'}})]
```

Define recorders from the `recorders` tree

Similarly to the `topology` tree, recorders are defined from lists.

We separate recorders connected to synapses (*e.g.* weight recorder) and those connected to units (*e.g.* spike detectors), which are defined in the `projection_recorders` and `population_recorders` params, respectively, of the `recorders` tree.

See the documentation for the `Network.build_recorders()`, `Network.build_population_recorders()` and `Network.build_projection_recorders()` methods for expected formatting.

The parameters of the recorders can be changed by using custom recorder models (in the `recorder_models` tree; see above).

[28]:
```
recorders_tree = ParamsTree.read(PARAMS_DIR/'recorders.yml').children['recorders']
recorders_tree
```

[28]:
```
ParamsTree(name='recorders', parent='None')
  params:
    population_recorders:
    - layers:
      - l1
      populations:
      - l1_exc
      model: my_multimeter
    - layers:
      - input_layer
      populations: null
      model: my_spike_detector
    projection_recorders:
    - source_layers:
      - l1
      source_population: l1_exc
      target_layers:
      - l1
      target_population: l1_inh
      projection_model: proj_1_AMPA
      model: weight_recorder
  nest_params: {}
```

[29]:
```
net.build_recorders(recorders_tree)
```

```
2020-06-30 13:42:39,443 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:42:39,444 [denest.network] INFO: Build N=1 projection recorders.
```

[30]:
```
net.population_recorders
```

[30]:
```
[PopulationRecorder(my_multimeter_l1_l1_exc,  {}{}),
 PopulationRecorder(my_spike_detector_input_layer_parrot_neuron,  {}{})]
```

[31]:
```
net.projection_recorders
```

[31]:
```
[ProjectionRecorder(weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh,  {}{})]
```

5.2.3 Create the network

[32]:
```
nest.ResetKernel()
nest.SetKernelStatus({'overwrite_files': True})
```

[33]:
```
net.create()
```

```
2020-06-30 13:42:39,792 [denest.network] INFO: Creating neuron models...
100%|| 2/2 [00:00<00:00, 1881.70it/s]
2020-06-30 13:42:39,804 [denest.network] INFO: Creating synapse models...
100%|| 2/2 [00:00<00:00, 1215.21it/s]
2020-06-30 13:42:39,811 [denest.network] INFO: Creating recorder models...
100%|| 3/3 [00:00<00:00, 1383.80it/s]
2020-06-30 13:42:39,839 [denest.network] INFO: Creating layers...
  0%|          | 0/2 [00:00<?, ?it/s]/Users/tom/nest/nest-simulator-2.20.0/lib/
↪python3.7/site-packages/nest/lib/hl_api_helper.py:127: UserWarning:
GetNodes is deprecated and will be removed in NEST 3.0. Use             GIDCollection␣
↪instead.
100%|| 2/2 [00:00<00:00,  9.04it/s]
2020-06-30 13:42:40,070 [denest.network] INFO: Creating population recorders...
100%|| 2/2 [00:00<00:00, 54.77it/s]
2020-06-30 13:42:40,123 [denest.network] INFO: Creating projection recorders...
100%|| 1/1 [00:00<00:00, 139.27it/s]
2020-06-30 13:42:40,145 [denest.network] INFO: Connecting layers...
100%|| 3/3 [00:00<00:00, 137.19it/s]
2020-06-30 13:42:40,182 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 206
Number of projections: 6650
```

5.2.4 Examine the network

deNEST provides convenient ways of accessing the objects in NEST.

Check the defaults of the created models

[34]:
```
print("`l1_exc` neuron models `nest_params`: ", net.neuron_models['l1_exc'].nest_
↪params)
```

```
`l1_exc` neuron models `nest_params`:  {'g_KL': 1.0, 'g_NaL': 1.0, 'V_m': -44.0}
```

[35]:
```
print('Corresponding params of the `l1_exc` model in nest:', nest.GetDefaults('l1_exc
↪', list(net.neuron_models['l1_exc'].nest_params.keys())))
```

```
Corresponding params of the `l1_exc` model in nest: (1.0, 1.0, -44.0)
```

Access the layers' units

[36]:
```
print('Layer `l1` shape: ', net.layers['l1'].layer_shape)
print('Population shapes: ', net.layers['l1'].population_shape)
```

```
Layer `l1` shape:  (5, 5)
Population shapes:  {'l1_exc': (5, 5, 4), 'l1_inh': (5, 5, 2)}
```

[37]:
```
net.layers['l1'].gids(location=(0, 0), population='l1_exc')
```

[37]:
```
[53, 78, 103, 128]
```

Access the projections created in NEST

[38]:
```
conn = net.projections[0]
conn
```

[38]:
```
TopoProjection(proj_1_AMPA-input_layer-parrot_neuron-l1-l1_exc,
{'type': 'topological'}{'connection_type': 'divergent',
 'mask': {'circular': {'radius': 2.0}},
 'kernel': 1.0,
 'synapse_model': 'my_AMPA_synapse',
 'weights': 1.0,
 'sources': {'model': 'parrot_neuron'},
 'targets': {'model': 'l1_exc'}})
```

[39]:
```
nest_conns = nest.GetConnections(
    source=conn.source.gids(conn.source_population),
    target=conn.target.gids(conn.target_population),
    synapse_model=conn.nest_params['synapse_model']
)
nest_conns[0:5]
```

[39]:
```
(array('l', [27, 53, 0, 68, 0]),
 array('l', [27, 88, 0, 68, 1]),
 array('l', [27, 83, 0, 68, 2]),
 array('l', [27, 152, 0, 68, 3]),
 array('l', [27, 84, 0, 68, 4]))
```

Access the recorders

[40]:
```
rec = net.population_recorders[0]
print(rec, rec.gid, rec.model, rec.layer, rec.population_name)
```

```
my_multimeter_l1_l1_exc (203,) my_multimeter l1 l1_exc
```

[41]:
```
connrec = net.projection_recorders[0]
print(connrec, connrec.gid, connrec.model)
```

```
weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh (205,) weight_recorder
```

5.2.5 Save and replicate the network

When building each of the network's elements using the `Network.build_*` methods, the `Network.tree` parameters were updated.

[42]:
```
net.tree
```

[42]:
```
ParamsTree(name='None', parent=None)
  params: {}
  nest_params: {}
  neuron_models:
    params: {}
    nest_params: {}
    my_neuron:
      params:
        nest_model: ht_neuron
      nest_params:
        g_KL: 1.0
        g_NaL: 1.0
      l1_exc:
        params: {}
        nest_params:
          V_m: -44.0

  ... [117 lines] ...

      - layers:
        - input_layer
        populations: null
        model: my_spike_detector
      projection_recorders:
      - source_layers:
        - l1
        source_population: l1_exc
        target_layers:
        - l1
        target_population: l1_inh
        projection_model: proj_1_AMPA
        model: weight_recorder
    nest_params: {}
```

We can save the parameter tree defining the whole network...

[43]:
```
net.tree.write(PARAMS_DIR/'network_tree.yml')
```

[43]:
```
PosixPath('data/params/network/network_tree.yml')
```

And use it to recreate another identical network.

[44]:
```
net2 = Network(ParamsTree.read(PARAMS_DIR/'network_tree.yml'))
```

```
2020-06-30 13:42:41,016 [denest.network] INFO: Build N=2 ``Model`` objects
2020-06-30 13:42:41,017 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
2020-06-30 13:42:41,018 [denest.network] INFO: Build N=3 ``Model`` objects
2020-06-30 13:42:41,019 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:42:41,021 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
```

(continues on next page)

(continued from previous page)

```
{'type': 'topological'}
2020-06-30 13:42:41,024 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:42:41,025 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
2020-06-30 13:42:41,030 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
2020-06-30 13:42:41,033 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:42:41,034 [denest.network] INFO: Build N=1 projection recorders.
```

[45]:
```
nest.ResetKernel()
net2.create()
```

```
2020-06-30 13:42:41,185 [denest.network] INFO: Creating neuron models...
100%|| 2/2 [00:00<00:00, 134.07it/s]
2020-06-30 13:42:41,215 [denest.network] INFO: Creating synapse models...
100%|| 2/2 [00:00<00:00, 1224.97it/s]
2020-06-30 13:42:41,234 [denest.network] INFO: Creating recorder models...
100%|| 3/3 [00:00<00:00, 1727.24it/s]
2020-06-30 13:42:41,249 [denest.network] INFO: Creating layers...
100%|| 2/2 [00:00<00:00, 10.27it/s]
2020-06-30 13:42:41,474 [denest.network] INFO: Creating population recorders...
100%|| 2/2 [00:00<00:00, 24.22it/s]
2020-06-30 13:42:41,596 [denest.network] INFO: Creating projection recorders...
100%|| 1/1 [00:00<00:00, 55.00it/s]
2020-06-30 13:42:41,616 [denest.network] INFO: Connecting layers...
100%|| 3/3 [00:00<00:00, 184.37it/s]
2020-06-30 13:42:41,645 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 206
Number of projections: 6650
```

[46]:
```
print(net.layers['l1'].gids(location=(0, 0), population='l1_exc'))
print(net2.layers['l1'].gids(location=(0, 0), population='l1_exc'))
```

```
[53, 78, 103, 128]
[53, 78, 103, 128]
```

5.3 Modify the network

deNEST provides a convenient way of modifying the state of some units within a network with the `Layer.set_state()` and `Network.set_state()` methods

- `Network.set_state()` and `Layer.set_state()` support **constant**, **multiplicative** or **additive** changes (`change_type` parameter)
- We can apply the same change for all units of the layer/population, or provide an array the same shape as the population to perform specific changes for each unit (`from_array` parameter). The array can be directly provided or loaded from file

In this tutorial we:

1. Change the state of units within a single population with the `Layer.set_state()` method
    1. Option 1 (`'from_array' == False`): provide a single value , used to change the state of all units of a layer or population
        1. 'constant' changes

2. 'multiplicative' changes
3. 'additive' changes

2. Option 2 (`'from_array' == True`): provide an array of values, mapped to units in the population. You can provide
   1. an NumPy array directly, or
   2. the path to a NumPy array stored on disk.

2. Change the state of multiple populations at once with the `Network.set_state()` method

[1]:
```
import os
from pathlib import Path
from pprint import pprint

import yaml
import numpy as np
import nest

from denest import *
import denest
```

[2]:
```
PARAMS_DIR = Path('./data/params/network')
```

5.3.1 Change the state of units within a population

Using `Layer.set_state()`

[3]:
```
nest.ResetKernel()
net = Network(ParamsTree.read(PARAMS_DIR/'network_tree.yml'))
net.create()
```

```
2020-06-30 13:42:56,703 [denest.network] INFO: Build N=2 ``Model`` objects
2020-06-30 13:42:56,704 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
2020-06-30 13:42:56,706 [denest.network] INFO: Build N=3 ``Model`` objects
2020-06-30 13:42:56,709 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:42:56,716 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:42:56,717 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:42:56,719 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
2020-06-30 13:42:56,725 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
2020-06-30 13:42:56,728 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:42:56,739 [denest.network] INFO: Build N=1 projection recorders.
2020-06-30 13:42:56,748 [denest.network] INFO: Creating neuron models...
100%|| 2/2 [00:00<00:00, 777.37it/s]
2020-06-30 13:42:56,783 [denest.network] INFO: Creating synapse models...
100%|| 2/2 [00:00<00:00, 1322.50it/s]
2020-06-30 13:42:56,789 [denest.network] INFO: Creating recorder models...
100%|| 3/3 [00:00<00:00, 716.00it/s]
2020-06-30 13:42:56,818 [denest.network] INFO: Creating layers...
  0%|          | 0/2 [00:00<?, ?it/s]/Users/tom/nest/nest-simulator-2.20.0/lib/
↪python3.7/site-packages/nest/lib/hl_api_helper.py:127: UserWarning:
```

(continues on next page)

(continued from previous page)

```
GetNodes is deprecated and will be removed in NEST 3.0. Use             GIDCollection␣
↪instead.
100%|| 2/2 [00:00<00:00,  6.83it/s]
2020-06-30 13:42:57,121 [denest.network] INFO: Creating population recorders...
100%|| 2/2 [00:00<00:00, 40.20it/s]
2020-06-30 13:42:57,187 [denest.network] INFO: Creating projection recorders...
100%|| 1/1 [00:00<00:00, 414.42it/s]
2020-06-30 13:42:57,217 [denest.network] INFO: Connecting layers...
100%|| 3/3 [00:00<00:00, 316.69it/s]
2020-06-30 13:42:57,242 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 206
Number of projections: 6650
```

[4]:
```
layer_name = 'l1'
population_name = 'l1_exc'

layer = net.layers['l1']

print('layer shape: ', layer.shape)
print('population shapes: ', layer.population_shape)
```

```
layer shape:  (5, 5)
population shapes:  {'l1_exc': (5, 5, 4), 'l1_inh': (5, 5, 2)}
```

Option 1: provide a single value, used to change the state of all units of a layer or population

Use `'from_array'==False` in in `Layer.set_state()`.

“constant” change type

[5]:
```
nest_params = {
    'V_m': -69.0,
    'g_peak_AMPA': 0.2,
}
```

[6]:
```
print('Unique values for l1_exc: ', { param: set(nest.GetStatus(layer.gids(population=
↪'l1_exc'), param)) for param in nest_params.keys()} )
print('Unique values for l1_inh: ', { param: set(nest.GetStatus(layer.gids(population=
↪'l1_inh'), param)) for param in nest_params.keys()} )
```

```
Unique values for l1_exc:  {'V_m': {-44.0}, 'g_peak_AMPA': {0.1}}
Unique values for l1_inh:  {'V_m': {-55.0}, 'g_peak_AMPA': {0.1}}
```

[7]:
```
# Change param for a single population

layer.set_state(
    nest_params=nest_params,
    population_name='l1_exc',
    change_type='constant',
    from_array=False,
)
```

```
2020-06-30 13:42:57,508 [denest.network.layers] INFO: Layer='l1', pop='l1_exc':␣
↪Applying 'constant' change, param='V_m', from single value')
2020-06-30 13:42:57,509 [denest.network.layers] INFO: Layer='l1', pop='l1_exc':␣
↪Applying 'constant' change, param='g_peak_AMPA', from single value')
```

[8]:
```
print('Unique values for l1_exc: ', { param: set(nest.GetStatus(layer.gids(population=
↪'l1_exc'), param)) for param in nest_params.keys()} )
print('Unique values for l1_inh: ', { param: set(nest.GetStatus(layer.gids(population=
↪'l1_inh'), param)) for param in nest_params.keys()} )
```

```
Unique values for l1_exc:  {'V_m': {-69.0}, 'g_peak_AMPA': {0.2}}
Unique values for l1_inh:  {'V_m': {-55.0}, 'g_peak_AMPA': {0.1}}
```

[9]:
```
# Change param for  all populations

layer.set_state(
    nest_params=nest_params,
    population_name=None,
#     population_name='l1_inh',
    change_type='constant',
    from_array=False,
)
```

```
2020-06-30 13:42:58,496 [denest.network.layers] INFO: Layer='l1', pop='l1_exc':␣
↪Applying 'constant' change, param='V_m', from single value')
2020-06-30 13:42:58,497 [denest.network.layers] INFO: Layer='l1', pop='l1_exc':␣
↪Applying 'constant' change, param='g_peak_AMPA', from single value')
2020-06-30 13:42:59,372 [denest.network.layers] INFO: Layer='l1', pop='l1_inh':␣
↪Applying 'constant' change, param='V_m', from single value')
2020-06-30 13:42:59,373 [denest.network.layers] INFO: Layer='l1', pop='l1_inh':␣
↪Applying 'constant' change, param='g_peak_AMPA', from single value')
```

[10]:
```
print('Unique values for l1_exc: ', { param: set(nest.GetStatus(layer.gids(population=
↪'l1_exc'), param)) for param in nest_params.keys()} )
print('Unique values for l1_inh: ', { param: set(nest.GetStatus(layer.gids(population=
↪'l1_inh'), param)) for param in nest_params.keys()} )
```

```
Unique values for l1_exc:  {'V_m': {-69.0}, 'g_peak_AMPA': {0.2}}
Unique values for l1_inh:  {'V_m': {-69.0}, 'g_peak_AMPA': {0.2}}
```

“multiplicative” change

[11]:
```
# Double the value
nest_params = {
    'g_peak_AMPA': 2.0,
}
```

[12]:
```
print('Unique values for l1_exc: ', { param: set(nest.GetStatus(layer.gids(population=
↪'l1_exc'), param)) for param in nest_params.keys()} )
print('Unique values for l1_inh: ', { param: set(nest.GetStatus(layer.gids(population=
↪'l1_inh'), param)) for param in nest_params.keys()} )
```

```
Unique values for l1_exc:  {'g_peak_AMPA': {0.2}}
Unique values for l1_inh:  {'g_peak_AMPA': {0.2}}
```

[13]:
```
# Change param for a single population

layer.set_state(
    nest_params=nest_params,
    population_name=None,
    change_type='multiplicative',
    from_array=False,
)
```

```
2020-06-30 13:43:00,387 [denest.network.layers] INFO: Layer='l1', pop='l1_exc':␣
↪Applying 'multiplicative' change, param='g_peak_AMPA', from single value')
2020-06-30 13:43:01,328 [denest.network.layers] INFO: Layer='l1', pop='l1_inh':␣
↪Applying 'multiplicative' change, param='g_peak_AMPA', from single value')
```

[14]:
```
print('Unique values for l1_exc: ', { param: set(nest.GetStatus(layer.gids(population=
↪'l1_exc'), param)) for param in nest_params.keys()} )
print('Unique values for l1_inh: ', { param: set(nest.GetStatus(layer.gids(population=
↪'l1_inh'), param)) for param in nest_params.keys()} )
```

```
Unique values for l1_exc:  {'g_peak_AMPA': {0.4}}
Unique values for l1_inh:  {'g_peak_AMPA': {0.4}}
```

“additive” change

[15]:
```
# Double the value
nest_params = {
    'V_m': 5.0,
}
```

[16]:
```
print('Unique values for l1_exc: ', { param: set(nest.GetStatus(layer.gids(population=
↪'l1_exc'), param)) for param in nest_params.keys()} )
print('Unique values for l1_inh: ', { param: set(nest.GetStatus(layer.gids(population=
↪'l1_inh'), param)) for param in nest_params.keys()} )
```

```
Unique values for l1_exc:  {'V_m': {-69.0}}
Unique values for l1_inh:  {'V_m': {-69.0}}
```

[17]:
```
# Change param for a single population

layer.set_state(
    nest_params=nest_params,
    population_name=None,
    change_type='additive',
    from_array=False,
)
```

```
2020-06-30 13:43:01,955 [denest.network.layers] INFO: Layer='l1', pop='l1_exc':␣
↪Applying 'additive' change, param='V_m', from single value')
2020-06-30 13:43:03,154 [denest.network.layers] INFO: Layer='l1', pop='l1_inh':␣
↪Applying 'additive' change, param='V_m', from single value')
```

[18]:
```
print('Unique values for l1_exc: ', { param: set(nest.GetStatus(layer.gids(population=
↪'l1_exc'), param)) for param in nest_params.keys()} )
print('Unique values for l1_inh: ', { param: set(nest.GetStatus(layer.gids(population=
↪'l1_inh'), param)) for param in nest_params.keys()} )
```

```
Unique values for l1_exc:  {'V_m': {-64.0}}
Unique values for l1_inh:  {'V_m': {-64.0}}
```

Option 2: provide an array the same shape as the population

For more flexible setting of the state of each individual unit, use `'from_array'==True` in in `Layer.set_state()`.

This can be used to set stimulator state arbitrarily (*e.g.* "spike_times" of a spike generator).

We can provide the array directly

[19]:
```
# Set V_m=-70 for all units except those at location [0, 0]

pop_shape = layer.population_shapes['l1_exc']
V_m_array = -70.0 * np.ones(pop_shape)
V_m_array[0, 0, :] = -60
```

or load the array from file

The 'input_dir' kwarg sets the directory from which arrays are loaded

[20]:
```
# Set g_peak_AMPA=0.33 for all units except those at location [0, 0]
g_peak_AMPA_array = 0.33 * np.ones(pop_shape)
g_peak_AMPA_array[0, 0, :] = 1.0

#  save the array to file
INPUT_DIR = Path('./data/input')
os.makedirs(INPUT_DIR, exist_ok=True)
array_path = INPUT_DIR/'g_peak_AMPA_array'
np.save(INPUT_DIR/'g_peak_AMPA_array', g_peak_AMPA_array)

np.load(INPUT_DIR/'g_peak_AMPA_array.npy').shape
```

[20]:
```
(5, 5, 4)
```

[21]:
```
# We provide either the array or the path to an array, relative to the 'input_dir'␣
↪directory

nest_params = {
    'V_m': V_m_array,
    'g_peak_AMPA': Path('./g_peak_AMPA_array.npy'),
}
```

[22]:
```
print('Unique values for l1_exc: ', { param: set(nest.GetStatus(layer.gids(population=
↪'l1_exc'), param)) for param in nest_params.keys()} )
```

```
Unique values for l1_exc:  {'V_m': {-64.0}, 'g_peak_AMPA': {0.4}}
```

[23]:
```
# Change param for a single population

layer.set_state(
```

(continues on next page)

(continued from previous page)

```
    nest_params=nest_params,
    input_dir=INPUT_DIR,
    population_name='l1_exc',
    change_type='constant',
    from_array=True,
)
```

```
2020-06-30 13:43:04,119 [denest.network.layers] INFO: Layer='l1', pop='l1_exc':␣
↪Applying 'constant' change, param='V_m', from array')
2020-06-30 13:43:04,129 [denest.network.layers] INFO: Layer='l1', pop='l1_exc':␣
↪Applying 'constant' change, param='g_peak_AMPA', from array')
```

[24]:
```
print('Unique values for l1_exc at location [0, 0]: ', { param: set(nest.
↪GetStatus(layer.gids(population='l1_exc', location=(0, 0)), param)) for param in␣
↪nest_params.keys()} )
print('Unique values for l1_exc at all locations: ', { param: set(nest.
↪GetStatus(layer.gids(population='l1_exc'), param)) for param in nest_params.keys()}␣
↪)
```

```
Unique values for l1_exc at location [0, 0]:  {'V_m': {-60.0}, 'g_peak_AMPA': {1.0}}
Unique values for l1_exc at all locations:  {'V_m': {-70.0, -60.0}, 'g_peak_AMPA': {0.
↪33, 1.0}}
```

5.3.2 Change the state of units throughout the network

Using `Network.set_state()`, we can specify modifications for multiple populations at once.

[25]:
```
# Set the input layer spike times

input_layer = net.layers['input_layer']
input_pop_shape = input_layer.population_shapes[
    input_layer.stimulator_model
]
print(f'`{input_layer.stimulator_model}` population shape: {input_pop_shape}')

# Build the spike times for each unit

spike_times = np.empty(input_pop_shape, dtype=np.object)

# Set the same spike times for all units...
for idx, _ in np.ndenumerate(spike_times):
    spike_times[idx] = [1.0, 10.0]

# Except one unit
idx = (0, 0, 0)
spike_times[idx] = [5.0]

print(f'input array shape: {spike_times.shape}')
print(f'spike_times array: {spike_times[:,:,0]}')
```

```
`spike_generator` population shape: (5, 5, 1)
input array shape: (5, 5, 1)
spike_times array: [[list([5.0]) list([1.0, 10.0]) list([1.0, 10.0]) list([1.0, 10.0])
  list([1.0, 10.0])]
 [list([1.0, 10.0]) list([1.0, 10.0]) list([1.0, 10.0]) list([1.0, 10.0])
  list([1.0, 10.0])]
```

(continues on next page)

(continued from previous page)

```
 [list([1.0, 10.0]) list([1.0, 10.0]) list([1.0, 10.0]) list([1.0, 10.0])
  list([1.0, 10.0])]
 [list([1.0, 10.0]) list([1.0, 10.0]) list([1.0, 10.0]) list([1.0, 10.0])
  list([1.0, 10.0])]
 [list([1.0, 10.0]) list([1.0, 10.0]) list([1.0, 10.0]) list([1.0, 10.0])
  list([1.0, 10.0])]]
```

[26]:
```
net.set_state(
    [
        {
            'layers': ['l1'],
            'population_name': None,
            'change_type': 'constant',
            'from_array': False,
            'nest_params': {
                'V_m': -69.9
            }

        },
        {
            'layers': ['input_layer'],
            'population_name': 'spike_generator',
            'change_type': 'constant',
            'from_array': True,
            'nest_params': {
                'spike_times': spike_times,
            }

        },

    ]
)
```

```
2020-06-30 13:43:05,218 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from array')
2020-06-30 13:43:05,277 [denest.network.layers] INFO: Layer='l1', pop='l1_exc':␣
↪Applying 'constant' change, param='V_m', from single value')
2020-06-30 13:43:06,332 [denest.network.layers] INFO: Layer='l1', pop='l1_inh':␣
↪Applying 'constant' change, param='V_m', from single value')
```

[27]:
```
# Get status of spike generators
print(
    nest.GetStatus(net.layers['input_layer'].gids(population='spike_generator'),
↪'spike_times')
)
```

```
(array([5.]), array([ 1., 10.]), array([ 1., 10.]), array([ 1., 10.]), array([ 1., 10.
↪]), array([ 1., 10.]), array([ 1., 10.]), array([ 1., 10.]), array([ 1., 10.]),␣
↪array([ 1., 10.]), array([ 1., 10.]), array([ 1., 10.]), array([ 1., 10.]), array([␣
↪1., 10.]), array([ 1., 10.]), array([ 1., 10.]), array([ 1., 10.]), array([ 1., 10.
↪]), array([ 1., 10.]), array([ 1., 10.]), array([ 1., 10.]), array([ 1., 10.]),␣
↪array([ 1., 10.]), array([ 1., 10.]), array([ 1., 10.]))
```

[28]:
```
# Get status of l1 units
print(
    nest.GetStatus(net.layers['l1'].gids(), 'V_m')
)
```

```
(-69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -
↪69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -
↪69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -
↪69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -
↪69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -
↪69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -
↪69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -
↪69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -
↪69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -
↪69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -
↪69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -
↪69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -69.9, -
↪69.9, -69.9, -69.9, -69.9, -69.9, -69.9)
```

5.4 The `Simulation` object

Let's familiarize ourselves with the `Simulation` object.

In this tutorial we:

1. **Initialize an empty ‘‘Simulation‘‘ object**
2. **Initialize the NEST kernel** (`Simulation.init__kernel`)
3. **Create a network** (`Simulation.create__network`)
4. Create sessions:
    1. **Build session models** (`Simulation.build__session__models`)
    2. **Build the list of sessions** from session models (`Simulation.build__sessions`)
5. **Run** the simulation
6. **Replicate** the simulation

NB: Usually we'd perform all the steps at once by providing the full simulation tree to the `Simulation` object during initialization

[1]:
```
import nest
import yaml
from pathlib import Path
from pprint import pprint

from denest import *
import denest
```

[2]:
```
PARAMS_DIR = Path('./data/params')
DATA_DIR = Path('./data/outputs')
OUTPUT_DIR = DATA_DIR/'output'
```

5.4.1 Initialize an empty Simulation object

Empty network, no kernel initialization, etc…

[3]:
```
sim = Simulation(output_dir=OUTPUT_DIR)
```

```
2020-06-30 13:52:43,106 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child kernel
2020-06-30 13:52:43,107 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child simulation
2020-06-30 13:52:43,108 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child session_models
2020-06-30 13:52:43,109 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child network
2020-06-30 13:52:43,111 [denest.utils.validation] INFO: Object `simulation`: params:␣
↪using default value for optional parameters:
{'input_dir': 'input', 'output_dir': 'output', 'sessions': []}
2020-06-30 13:52:43,116 [denest.utils.validation] INFO: Object `kernel`: params:␣
↪using default value for optional parameters:
{'extension_modules': [], 'nest_seed': 1}
2020-06-30 13:52:43,118 [denest.simulation] INFO: Initializing NEST kernel and␣
↪seeds...
2020-06-30 13:52:43,119 [denest.simulation] INFO:   Resetting NEST kernel...
2020-06-30 13:52:43,142 [denest.simulation] INFO:   Setting NEST kernel status...
2020-06-30 13:52:43,146 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({}
↪)`
2020-06-30 13:52:43,153 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'data_path': 'data/outputs/output/data', 'grng_seed': 2, 'rng_seeds': range(3, 4)})
2020-06-30 13:52:43,168 [denest.simulation] INFO:   Finished setting NEST kernel␣
↪status
2020-06-30 13:52:43,173 [denest.simulation] INFO:   Installing external modules...
2020-06-30 13:52:43,179 [denest.simulation] INFO:   Finished installing external␣
↪modules
2020-06-30 13:52:43,189 [denest.simulation] INFO: Finished initializing kernel
2020-06-30 13:52:43,193 [denest.simulation] INFO: Build N=0 session models
2020-06-30 13:52:43,197 [denest.simulation] INFO: Build N=0 sessions
2020-06-30 13:52:43,210 [denest.simulation] INFO: Sessions: []
2020-06-30 13:52:43,212 [denest.simulation] INFO: Building network.
2020-06-30 13:52:43,219 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child neuron_models
2020-06-30 13:52:43,233 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child synapse_models
2020-06-30 13:52:43,254 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child layers
2020-06-30 13:52:43,272 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child projection_models
2020-06-30 13:52:43,285 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child topology
2020-06-30 13:52:43,297 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child recorder_models
2020-06-30 13:52:43,302 [denest.utils.validation] INFO: 'None' tree: adding empty␣
↪child recorders
2020-06-30 13:52:43,305 [denest.network] INFO: Build N=0 ``Model`` objects
2020-06-30 13:52:43,308 [denest.network] INFO: Build N=0 ``SynapseModel`` objects
2020-06-30 13:52:43,310 [denest.network] INFO: Build N=0 ``Model`` objects
2020-06-30 13:52:43,315 [denest.network] INFO: Build N=0 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:52:43,318 [denest.network] INFO: Build N=0 ``ProjectionModel`` objects
2020-06-30 13:52:43,322 [denest.utils.validation] INFO: Object `topology`: params:␣
↪using default value for optional parameters:
{'projections': []}
2020-06-30 13:52:43,326 [denest.network] INFO: Build N=0 ``TopoProjection`` objects
2020-06-30 13:52:43,329 [denest.utils.validation] INFO: Object `recorders`: params:␣
↪using default value for optional parameters:
```

(continues on next page)

(continued from previous page)

```
{'population_recorders': [], 'projection_recorders': []}
2020-06-30 13:52:43,335 [denest.network] INFO: Build N=0 population recorders.
2020-06-30 13:52:43,337 [denest.network] INFO: Build N=0 projection recorders.
2020-06-30 13:52:43,338 [denest.simulation] INFO: Creating network.
2020-06-30 13:52:43,341 [denest.network] INFO: Creating neuron models...
0it [00:00, ?it/s]
2020-06-30 13:52:43,365 [denest.network] INFO: Creating synapse models...
0it [00:00, ?it/s]
2020-06-30 13:52:43,378 [denest.network] INFO: Creating recorder models...
0it [00:00, ?it/s]
2020-06-30 13:52:43,396 [denest.network] INFO: Creating layers...
0it [00:00, ?it/s]
2020-06-30 13:52:43,404 [denest.network] INFO: Creating population recorders...
0it [00:00, ?it/s]
2020-06-30 13:52:43,419 [denest.network] INFO: Creating projection recorders...
0it [00:00, ?it/s]
2020-06-30 13:52:43,426 [denest.network] INFO: Connecting layers...
0it [00:00, ?it/s]
2020-06-30 13:52:43,442 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 1
Number of projections: 0
2020-06-30 13:52:43,446 [denest.simulation] INFO: Finished creating network
2020-06-30 13:52:43,449 [denest.simulation] INFO: Saving simulation metadata...
2020-06-30 13:52:43,455 [denest.simulation] INFO: Creating output directory: data/
↪outputs/output
2020-06-30 13:52:43,459 [denest.io.save] INFO: Clearing directory: data/outputs/output
2020-06-30 13:52:43,475 [denest.io.save] INFO: Clearing directory: data/outputs/output
2020-06-30 13:52:43,486 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output/data
2020-06-30 13:52:43,494 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output/data
2020-06-30 13:52:43,500 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output/data
2020-06-30 13:52:43,503 [denest.io.save] INFO: Clearing directory: data/outputs/output
2020-06-30 13:52:43,532 [denest.simulation] INFO: Finished saving simulation metadata
```

5.4.2 Initialize the NEST kernel

`Simulation.init_kernel??` for doc

[4]:
```
kernel_tree = {
    'params':
        {
            'nest_seed': 10,
            'extension_modules': [],
        },
    'nest_params':
        {
            'resolution': 0.5,
            'overwrite_files': True
        },
}
```

[5]:
```
sim.init_kernel(kernel_tree)
```

```
2020-06-30 13:52:43,715 [denest.simulation] INFO: Initializing NEST kernel and␣
↪seeds...
2020-06-30 13:52:43,720 [denest.simulation] INFO:   Resetting NEST kernel...
2020-06-30 13:52:43,740 [denest.simulation] INFO:   Setting NEST kernel status...
2020-06-30 13:52:43,743 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'resolution': 0.5, 'overwrite_files': True})`
2020-06-30 13:52:43,746 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'data_path': 'data/outputs/output/data', 'grng_seed': 11, 'rng_seeds': range(12,␣
↪13)})
2020-06-30 13:52:43,750 [denest.simulation] INFO:   Finished setting NEST kernel␣
↪status
2020-06-30 13:52:43,756 [denest.simulation] INFO:   Installing external modules...
2020-06-30 13:52:43,784 [denest.simulation] INFO:   Finished installing external␣
↪modules
2020-06-30 13:52:43,785 [denest.simulation] INFO: Finished initializing kernel
```

[6]:
```
# nest_params have been passed to nest.SetKernelStatus
print(nest.GetKernelStatus('resolution'))
print(nest.GetKernelStatus('time'))
```

```
0.5
0.0
```

[7]:
```
# The raw data will be saved in the output directory
nest.GetKernelStatus('data_path')
```

[7]:
```
'data/outputs/output/data'
```

[8]:
```
# The kernel params are saved in the simulation's tree
sim.tree
```

[8]:
```
ParamsTree(name='None', parent=None)
  params: {}
  nest_params: {}
  kernel:
    params:
      nest_seed: 10
      extension_modules: []
    nest_params:
      resolution: 0.5
      overwrite_files: true
  simulation:
    params:
      output_dir: data/outputs/output
      sessions: []
    nest_params: {}
  session_models:
    params: {}
    nest_params: {}
  network:
    params: {}
    nest_params: {}
```

5.4.3 Create a network

We build and create the same network by passing the network tree, using the `Simulation.create_network` method

[9]:
```
net_tree = ParamsTree.read(PARAMS_DIR/'network/network_tree.yml')
```

[10]:
```
sim.create_network(net_tree)
```

```
2020-06-30 13:52:44,151 [denest.simulation] INFO: Building network.
2020-06-30 13:52:44,167 [denest.network] INFO: Build N=2 ``Model`` objects
2020-06-30 13:52:44,169 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
2020-06-30 13:52:44,174 [denest.network] INFO: Build N=3 ``Model`` objects
2020-06-30 13:52:44,180 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:52:44,183 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:52:44,192 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:52:44,198 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
2020-06-30 13:52:44,214 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
2020-06-30 13:52:44,232 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:52:44,239 [denest.network] INFO: Build N=1 projection recorders.
2020-06-30 13:52:44,254 [denest.simulation] INFO: Creating network.
2020-06-30 13:52:44,272 [denest.network] INFO: Creating neuron models...
100%|| 2/2 [00:00<00:00, 299.85it/s]
2020-06-30 13:52:44,291 [denest.network] INFO: Creating synapse models...
100%|| 2/2 [00:00<00:00, 702.21it/s]
2020-06-30 13:52:44,310 [denest.network] INFO: Creating recorder models...
100%|| 3/3 [00:00<00:00, 811.28it/s]
2020-06-30 13:52:44,322 [denest.network] INFO: Creating layers...
  0%|          | 0/2 [00:00<?, ?it/s]/Users/tom/nest/nest-simulator-2.20.0/lib/
↪python3.7/site-packages/nest/lib/hl_api_helper.py:127: UserWarning:
GetNodes is deprecated and will be removed in NEST 3.0. Use             GIDCollection␣
↪instead.
100%|| 2/2 [00:00<00:00,  8.14it/s]
2020-06-30 13:52:44,577 [denest.network] INFO: Creating population recorders...
100%|| 2/2 [00:00<00:00, 42.15it/s]
2020-06-30 13:52:44,631 [denest.network] INFO: Creating projection recorders...
100%|| 1/1 [00:00<00:00, 74.23it/s]
2020-06-30 13:52:44,679 [denest.network] INFO: Connecting layers...
100%|| 3/3 [00:00<00:00, 145.55it/s]
2020-06-30 13:52:44,736 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 206
Number of projections: 6650
2020-06-30 13:52:44,739 [denest.simulation] INFO: Finished creating network
```

[11]:
```
# Network object was created and can be accessed as we learnt
sim.network.layers['l1'].gid
```

[11]:
```
(52,)
```

[12]:
```
# network tree was saved and can be re-used
sim.tree
```

[12]:
```
ParamsTree(name='None', parent=None)
  params: {}
  nest_params: {}
  kernel:
    params:
      nest_seed: 10
      extension_modules: []
    nest_params:
      resolution: 0.5
      overwrite_files: true
  simulation:
    params:
      output_dir: data/outputs/output
      sessions: []
    nest_params: {}
  session_models:

  ... [135 lines] ...

        - layers:
          - input_layer
          populations: null
          model: my_spike_detector
        projection_recorders:
        - source_layers:
          - l1
          source_population: l1_exc
          target_layers:
          - l1
          target_population: l1_inh
          projection_model: proj_1_AMPA
          model: weight_recorder
      nest_params: {}
```

5.4.4 Create some sessions

Sessions allow us to run the network in specific conditions.

Session parameters:

- `simulation_time` (float): Duration of the session in ms. (mandatory)
- `reset_network` (bool): If true, `nest.ResetNetwork()` is called during session initialization (default `False`)
- `record` (bool): If false, the `start_time` field of recorder nodes in NEST is set to the end time of the session, so that no data is recorded during the session (default `True`)
- `shift_origin` (bool): If True, the `origin` flag of the stimulation devices of all the network's `InputLayer` layers is set to the start of the session during initialization. Useful to repeat sessions when the stimulators are eg spike generators.
- `unit_changes` (list): List describing the changes applied to certain units before the start of the session. Passed to `Network.set_state`.
- `synapse_changes` (list): List describing the changes applied to certain synapses before the start of the session. Passed to `Network.set_state`. Refer to that method for a description of how `synapse_changes`

is formatted and interpreted. No changes happen if empty. (default [])

Build session models from a tree

[13]:
```
# Notice the hierarchical inheritance as before
session_models_tree = ParamsTree.read(PARAMS_DIR/'session_models.yml').children[
↪'session_models']
print(session_models_tree)
```

```
params:
  record: true
  shift_origin: true
  simulation_time: 100.0
nest_params: {}
warmup:
  params:
    record: false
  nest_params: {}
2_spikes:
  params:
    unit_changes:
    - layers:
      - input_layer
      population_name: spike_generator
      nest_params:
        spike_times:
        - 1.0
        - 10.0
  nest_params: {}
3_spikes:
  params:
    unit_changes:
    - layers:
      - input_layer
      population_name: spike_generator
      nest_params:
        spike_times:
        - 1.0
        - 10.0
        - 20.0
  nest_params: {}
```

[14]:
```
sim.build_session_models(session_models_tree)
```

```
2020-06-30 13:52:44,980 [denest.simulation] INFO: Build N=3 session models
```

[15]:
```
sim.tree.children['session_models']
```

[15]:
```
ParamsTree(name='session_models', parent='None')
  params:
    record: true
    shift_origin: true
    simulation_time: 100.0
  nest_params: {}
  warmup:
    params:
```

(continues on next page)

(continued from previous page)

```
      record: false
    nest_params: {}
  2_spikes:
    params:
      unit_changes:
      - layers:
        - input_layer
        population_name: spike_generator

  ... [3 lines] ...

          - 10.0
    nest_params: {}
  3_spikes:
    params:
      unit_changes:
      - layers:
        - input_layer
        population_name: spike_generator
        nest_params:
          spike_times:
          - 1.0
          - 10.0
          - 20.0
    nest_params: {}
```

Build a list of sessions from the `session_models` templates

[16]:
```
sessions_order =  ['warmup', '3_spikes', '2_spikes', '3_spikes']
```

[17]:
```
sim.build_sessions(sessions_order)
```

```
2020-06-30 13:52:45,205 [denest.simulation] INFO: Build N=4 sessions
2020-06-30 13:52:45,206 [denest.session] INFO: Creating session "00_warmup"
2020-06-30 13:52:45,208 [denest.utils.validation] INFO: Object `00_warmup`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': [], 'unit_changes': []}
2020-06-30 13:52:45,212 [denest.session] INFO: Creating session "01_3_spikes"
2020-06-30 13:52:45,219 [denest.utils.validation] INFO: Object `01_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:52:45,221 [denest.session] INFO: Creating session "02_2_spikes"
2020-06-30 13:52:45,223 [denest.utils.validation] INFO: Object `02_2_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:52:45,225 [denest.session] INFO: Creating session "03_3_spikes"
2020-06-30 13:52:45,234 [denest.utils.validation] INFO: Object `03_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:52:45,239 [denest.simulation] INFO: Sessions: ['00_warmup', '01_3_spikes
↪', '02_2_spikes', '03_3_spikes']
```

[18]:
```
# Notice the session names
sim.sessions
```

[18]:
```
[Session(00_warmup, {'record': False,
  'reset_network': False,
  'shift_origin': True,
  'simulation_time': 100.0,
  'synapse_changes': [],
  'unit_changes': []}),
 Session(01_3_spikes, {'record': True,
  'reset_network': False,
  'shift_origin': True,
  'simulation_time': 100.0,
  'synapse_changes': [],
  'unit_changes': [{'layers': ['input_layer'],
                    'nest_params': {'spike_times': [1.0, 10.0, 20.0]},
                    'population_name': 'spike_generator'}]}),
 Session(02_2_spikes, {'record': True,
  'reset_network': False,
  'shift_origin': True,
  'simulation_time': 100.0,
  'synapse_changes': [],
  'unit_changes': [{'layers': ['input_layer'],
                    'nest_params': {'spike_times': [1.0, 10.0]},
                    'population_name': 'spike_generator'}]}),
 Session(03_3_spikes, {'record': True,
  'reset_network': False,
  'shift_origin': True,
  'simulation_time': 100.0,
  'synapse_changes': [],
  'unit_changes': [{'layers': ['input_layer'],
                    'nest_params': {'spike_times': [1.0, 10.0, 20.0]},
                    'population_name': 'spike_generator'}]})]
```

[19]:
```
# The session order is saved in the simulation parameters

sim.tree.children['simulation']
```

[19]:
```
ParamsTree(name='simulation', parent='None')
  params:
    output_dir: data/outputs/output
    sessions:
    - warmup
    - 3_spikes
    - 2_spikes
    - 3_spikes
  nest_params: {}
```

5.4.5 Run the simulation

[20]:
```
print('kernel time: ', nest.GetKernelStatus('time'))
```

```
kernel time:  0.0
```

[21]:
```
sim.run()
```

```
2020-06-30 13:52:45,497 [denest.simulation] INFO: Running 4 sessions...
2020-06-30 13:52:45,497 [denest.simulation] INFO: Running session: '00_warmup'...
2020-06-30 13:52:45,499 [denest.session] INFO: Initializing session...
2020-06-30 13:52:45,501 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_multimeter_l1_l1_exc: {'start': 100.0}
2020-06-30 13:52:45,504 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_spike_detector_input_layer_parrot_neuron: {'start': 100.0}
2020-06-30 13:52:45,505 [denest.network.recorders] INFO:   Setting status for␣
↪recorder weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh: {'start': 100.0}
2020-06-30 13:52:45,507 [denest.session] INFO: Setting `origin` flag to `0.0` for all␣
↪stimulation devices in ``InputLayers`` for session `00_warmup`
2020-06-30 13:52:45,523 [denest.session] INFO: Finished initializing session

2020-06-30 13:52:45,524 [denest.session] INFO: Running session '00_warmup' for 100 ms
2020-06-30 13:52:45,749 [denest.session] INFO: Finished running session
2020-06-30 13:52:45,750 [denest.session] INFO: Session '00_warmup' virtual running␣
↪time: 100 ms
2020-06-30 13:52:45,751 [denest.session] INFO: Session '00_warmup' real running time:␣
↪0h:00m:00s
2020-06-30 13:52:45,806 [denest.simulation] INFO: Done running session '00_warmup'
2020-06-30 13:52:45,807 [denest.simulation] INFO: Running session: '01_3_spikes'...
2020-06-30 13:52:45,823 [denest.session] INFO: Initializing session...
2020-06-30 13:52:45,824 [denest.session] INFO: Setting `origin` flag to `100.0` for␣
↪all stimulation devices in ``InputLayers`` for session `01_3_spikes`
2020-06-30 13:52:45,879 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:52:45,887 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:52:46,004 [denest.session] INFO: Finished initializing session

2020-06-30 13:52:46,005 [denest.session] INFO: Running session '01_3_spikes' for 100␣
↪ms
2020-06-30 13:52:46,213 [denest.session] INFO: Finished running session
2020-06-30 13:52:46,215 [denest.session] INFO: Session '01_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:52:46,216 [denest.session] INFO: Session '01_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:52:46,218 [denest.simulation] INFO: Done running session '01_3_spikes'
2020-06-30 13:52:46,219 [denest.simulation] INFO: Running session: '02_2_spikes'...
2020-06-30 13:52:46,221 [denest.session] INFO: Initializing session...
2020-06-30 13:52:46,222 [denest.session] INFO: Setting `origin` flag to `200.0` for␣
↪all stimulation devices in ``InputLayers`` for session `02_2_spikes`
2020-06-30 13:52:46,226 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:52:46,246 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:52:46,371 [denest.session] INFO: Finished initializing session

2020-06-30 13:52:46,371 [denest.session] INFO: Running session '02_2_spikes' for 100␣
↪ms
2020-06-30 13:52:46,653 [denest.session] INFO: Finished running session
2020-06-30 13:52:46,654 [denest.session] INFO: Session '02_2_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:52:46,658 [denest.session] INFO: Session '02_2_spikes' real running␣
↪time: 0h:00m:00s
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:52:46,660 [denest.simulation] INFO: Done running session '02_2_spikes'
2020-06-30 13:52:46,671 [denest.simulation] INFO: Running session: '03_3_spikes'...
2020-06-30 13:52:46,673 [denest.session] INFO: Initializing session...
2020-06-30 13:52:46,688 [denest.session] INFO: Setting `origin` flag to `300.0` for␣
↪all stimulation devices in ``InputLayers`` for session `03_3_spikes`
2020-06-30 13:52:46,741 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:52:46,745 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:52:46,876 [denest.session] INFO: Finished initializing session

2020-06-30 13:52:46,877 [denest.session] INFO: Running session '03_3_spikes' for 100␣
↪ms
2020-06-30 13:52:47,122 [denest.session] INFO: Finished running session
2020-06-30 13:52:47,193 [denest.session] INFO: Session '03_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:52:47,227 [denest.session] INFO: Session '03_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:52:47,244 [denest.simulation] INFO: Done running session '03_3_spikes'
2020-06-30 13:52:47,253 [denest.simulation] INFO: Finished running simulation
```

[22]:
```
nest.GetKernelStatus('time')
```

[22]:
```
400.0
```

Since the initialized Simulation object was empty, we need to save the metadata

[23]:
```
sim.save_metadata()
```

```
2020-06-30 13:52:47,378 [denest.simulation] INFO: Saving simulation metadata...
2020-06-30 13:52:47,379 [denest.simulation] INFO: Creating output directory: data/
↪outputs/output
2020-06-30 13:52:47,545 [denest.simulation] INFO: Finished saving simulation metadata
```

[24]:
```
!ls {OUTPUT_DIR}
```

```
data                parameter_tree.yml session_times.yml  versions.txt
```

[25]:
```
!cat {OUTPUT_DIR/'session_times.yml'}
```

```
00_warmup: !!python/tuple
- 0.0
- 100.0
01_3_spikes: !!python/tuple
- 100.0
- 200.0
02_2_spikes: !!python/tuple
- 200.0
- 300.0
03_3_spikes: !!python/tuple
- 300.0
- 400.0
```

[26]:
```
!ls {OUTPUT_DIR/'data'}
```

```
my_multimeter_l1_l1_exc-203-0.dat
my_multimeter_l1_l1_exc.yml
my_spike_detector_input_layer_parrot_neuron-204-0.gdf
my_spike_detector_input_layer_parrot_neuron.yml
weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh-205-0.csv
weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh.yml
```

5.4.6 Replicate the simulation

[27]:
```
params = ParamsTree.read(OUTPUT_DIR/'parameter_tree.yml')
params
```

[27]:
```
ParamsTree(name='None', parent=None)
  params: {}
  nest_params: {}
  kernel:
    params:
      nest_seed: 10
      extension_modules: []
    nest_params:
      resolution: 0.5
      overwrite_files: true
  simulation:
    params:
      output_dir: data/outputs/output
      sessions:
      - warmup
      - 3_spikes

  ... [169 lines] ...

        - layers:
          - input_layer
          populations: null
          model: my_spike_detector
        projection_recorders:
        - source_layers:
          - l1
          source_population: l1_exc
          target_layers:
          - l1
          target_population: l1_inh
          projection_model: proj_1_AMPA
          model: weight_recorder
      nest_params: {}
```

[28]:
```
sim = denest.Simulation(params, output_dir=DATA_DIR/'output_replicate')
sim.run()
```

```
2020-06-30 13:52:48,433 [denest.utils.validation] INFO: Object `simulation`: params:␣
↪using default value for optional parameters:
{'input_dir': 'input'}
2020-06-30 13:52:48,435 [denest.simulation] INFO: Initializing NEST kernel and␣
↪seeds...
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:52:48,437 [denest.simulation] INFO:   Resetting NEST kernel...
2020-06-30 13:52:48,544 [denest.simulation] INFO:   Setting NEST kernel status...
2020-06-30 13:52:48,546 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'resolution': 0.5, 'overwrite_files': True})`
2020-06-30 13:52:48,565 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'data_path': 'data/outputs/output_replicate/data', 'grng_seed': 11, 'rng_seeds':␣
↪range(12, 13)})
2020-06-30 13:52:48,619 [denest.simulation] INFO:   Finished setting NEST kernel␣
↪status
2020-06-30 13:52:48,630 [denest.simulation] INFO:   Installing external modules...
2020-06-30 13:52:48,647 [denest.simulation] INFO:   Finished installing external␣
↪modules
2020-06-30 13:52:48,648 [denest.simulation] INFO: Finished initializing kernel
2020-06-30 13:52:48,651 [denest.simulation] INFO: Build N=3 session models
2020-06-30 13:52:48,653 [denest.simulation] INFO: Build N=4 sessions
2020-06-30 13:52:48,655 [denest.session] INFO: Creating session "00_warmup"
2020-06-30 13:52:48,656 [denest.utils.validation] INFO: Object `00_warmup`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': [], 'unit_changes': []}
2020-06-30 13:52:48,658 [denest.session] INFO: Creating session "01_3_spikes"
2020-06-30 13:52:48,660 [denest.utils.validation] INFO: Object `01_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:52:48,662 [denest.session] INFO: Creating session "02_2_spikes"
2020-06-30 13:52:48,665 [denest.utils.validation] INFO: Object `02_2_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:52:48,668 [denest.session] INFO: Creating session "03_3_spikes"
2020-06-30 13:52:48,671 [denest.utils.validation] INFO: Object `03_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:52:48,675 [denest.simulation] INFO: Sessions: ['00_warmup', '01_3_spikes
↪', '02_2_spikes', '03_3_spikes']
2020-06-30 13:52:48,680 [denest.simulation] INFO: Building network.
2020-06-30 13:52:48,711 [denest.network] INFO: Build N=2 ``Model`` objects
2020-06-30 13:52:48,715 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
2020-06-30 13:52:48,720 [denest.network] INFO: Build N=3 ``Model`` objects
2020-06-30 13:52:48,724 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:52:48,728 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:52:48,730 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:52:48,735 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
2020-06-30 13:52:48,742 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
2020-06-30 13:52:48,751 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:52:48,757 [denest.network] INFO: Build N=1 projection recorders.
2020-06-30 13:52:48,759 [denest.simulation] INFO: Creating network.
2020-06-30 13:52:48,763 [denest.network] INFO: Creating neuron models...
100%|| 2/2 [00:00<00:00, 197.59it/s]
2020-06-30 13:52:48,784 [denest.network] INFO: Creating synapse models...
100%|| 2/2 [00:00<00:00, 409.50it/s]
2020-06-30 13:52:48,802 [denest.network] INFO: Creating recorder models...
100%|| 3/3 [00:00<00:00, 1972.24it/s]
2020-06-30 13:52:48,823 [denest.network] INFO: Creating layers...
```

(continues on next page)

(continued from previous page)

```
100%|| 2/2 [00:00<00:00,  9.01it/s]
2020-06-30 13:52:49,080 [denest.network] INFO: Creating population recorders...
100%|| 2/2 [00:00<00:00, 63.84it/s]
2020-06-30 13:52:49,120 [denest.network] INFO: Creating projection recorders...
100%|| 1/1 [00:00<00:00, 349.44it/s]
2020-06-30 13:52:49,130 [denest.network] INFO: Connecting layers...
100%|| 3/3 [00:00<00:00, 130.74it/s]
2020-06-30 13:52:49,160 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 206
Number of projections: 6650
2020-06-30 13:52:49,164 [denest.simulation] INFO: Finished creating network
2020-06-30 13:52:49,166 [denest.simulation] INFO: Saving simulation metadata...
2020-06-30 13:52:49,167 [denest.simulation] INFO: Creating output directory: data/
↪outputs/output_replicate
2020-06-30 13:52:49,169 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_replicate
2020-06-30 13:52:49,179 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_replicate
2020-06-30 13:52:49,186 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_replicate/data
2020-06-30 13:52:49,189 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_replicate/data
2020-06-30 13:52:49,191 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_replicate/data
2020-06-30 13:52:49,209 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_replicate
2020-06-30 13:52:49,346 [denest.simulation] INFO: Finished saving simulation metadata
2020-06-30 13:52:49,347 [denest.simulation] INFO: Running 4 sessions...
2020-06-30 13:52:49,350 [denest.simulation] INFO: Running session: '00_warmup'...
2020-06-30 13:52:49,352 [denest.session] INFO: Initializing session...
2020-06-30 13:52:49,359 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_multimeter_l1_l1_exc: {'start': 100.0}
2020-06-30 13:52:49,375 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_spike_detector_input_layer_parrot_neuron: {'start': 100.0}
2020-06-30 13:52:49,379 [denest.network.recorders] INFO:   Setting status for␣
↪recorder weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh: {'start': 100.0}
2020-06-30 13:52:49,381 [denest.session] INFO: Setting `origin` flag to `0.0` for all␣
↪stimulation devices in ``InputLayers`` for session `00_warmup`
2020-06-30 13:52:49,391 [denest.session] INFO: Finished initializing session

2020-06-30 13:52:49,394 [denest.session] INFO: Running session '00_warmup' for 100 ms
2020-06-30 13:52:49,737 [denest.session] INFO: Finished running session
2020-06-30 13:52:49,738 [denest.session] INFO: Session '00_warmup' virtual running␣
↪time: 100 ms
2020-06-30 13:52:49,739 [denest.session] INFO: Session '00_warmup' real running time:␣
↪0h:00m:00s
2020-06-30 13:52:49,757 [denest.simulation] INFO: Done running session '00_warmup'
2020-06-30 13:52:49,763 [denest.simulation] INFO: Running session: '01_3_spikes'...
2020-06-30 13:52:49,784 [denest.session] INFO: Initializing session...
2020-06-30 13:52:49,795 [denest.session] INFO: Setting `origin` flag to `100.0` for␣
↪all stimulation devices in ``InputLayers`` for session `01_3_spikes`
2020-06-30 13:52:49,824 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:52:49,831 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:52:49,938 [denest.session] INFO: Finished initializing session

2020-06-30 13:52:49,939 [denest.session] INFO: Running session '01_3_spikes' for 100␣
↪ms
2020-06-30 13:52:50,128 [denest.session] INFO: Finished running session
2020-06-30 13:52:50,129 [denest.session] INFO: Session '01_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:52:50,130 [denest.session] INFO: Session '01_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:52:50,131 [denest.simulation] INFO: Done running session '01_3_spikes'
2020-06-30 13:52:50,159 [denest.simulation] INFO: Running session: '02_2_spikes'...
2020-06-30 13:52:50,179 [denest.session] INFO: Initializing session...
2020-06-30 13:52:50,202 [denest.session] INFO: Setting `origin` flag to `200.0` for␣
↪all stimulation devices in ``InputLayers`` for session `02_2_spikes`
2020-06-30 13:52:50,230 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:52:50,231 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:52:50,321 [denest.session] INFO: Finished initializing session

2020-06-30 13:52:50,321 [denest.session] INFO: Running session '02_2_spikes' for 100␣
↪ms
2020-06-30 13:52:50,620 [denest.session] INFO: Finished running session
2020-06-30 13:52:50,622 [denest.session] INFO: Session '02_2_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:52:50,622 [denest.session] INFO: Session '02_2_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:52:50,623 [denest.simulation] INFO: Done running session '02_2_spikes'
2020-06-30 13:52:50,627 [denest.simulation] INFO: Running session: '03_3_spikes'...
2020-06-30 13:52:50,629 [denest.session] INFO: Initializing session...
2020-06-30 13:52:50,644 [denest.session] INFO: Setting `origin` flag to `300.0` for␣
↪all stimulation devices in ``InputLayers`` for session `03_3_spikes`
2020-06-30 13:52:50,673 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:52:50,674 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:52:50,770 [denest.session] INFO: Finished initializing session

2020-06-30 13:52:50,771 [denest.session] INFO: Running session '03_3_spikes' for 100␣
↪ms
2020-06-30 13:52:50,940 [denest.session] INFO: Finished running session
2020-06-30 13:52:50,941 [denest.session] INFO: Session '03_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:52:50,943 [denest.session] INFO: Session '03_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:52:50,946 [denest.simulation] INFO: Done running session '03_3_spikes'
2020-06-30 13:52:50,989 [denest.simulation] INFO: Finished running simulation
```

[29]:
```
!ls {str(DATA_DIR/'output_replicate')}
```

```
data                parameter_tree.yml session_times.yml  versions.txt
```

5.5 Load output

[1]:
```
from pathlib import Path

import denest.io.load
```

[2]:
```
OUTPUT_DIR = Path('./data/outputs/output')
```

5.5.1 Version information

[3]:
```
!cat {OUTPUT_DIR/'versions.txt'}
```

```
denest=1.0.1
NEST nest-2.20.0
```

5.5.2 Start and end time for each session

[4]:
```
# Load the start and end time for each session
session_times = denest.io.load.load_session_times(OUTPUT_DIR)
print(session_times)  # {<session_name>: (<session_start>, <session_end>)}
```

```
{'00_warmup': (0.0, 100.0), '01_3_spikes': (100.0, 200.0), '02_2_spikes': (200.0, 300.
↪0), '03_3_spikes': (300.0, 400.0)}
```

5.5.3 Load data from a specific recorder

Relevant information about a recorder and the population it's connected to are contained in its metadata file

[5]:
```
recorder_metadata_path = OUTPUT_DIR/'data/my_multimeter_l1_l1_exc.yml'

recorder_metadata = denest.io.load.load_yaml(recorder_metadata_path)
print(f'Metadata : {recorder_metadata}')
```

```
Metadata : {'colnames': ['gid', 'time', 'V_m'], 'filenames': ['my_multimeter_l1_l1_
↪exc-203-0.dat'], 'gids': [53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,␣
↪67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,␣
↪88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106,␣
↪107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123,
↪ 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139,␣
↪140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152], 'interval': 20.0,
↪'label': 'my_multimeter_l1_l1_exc', 'layer_name': 'l1', 'layer_shape': (5, 5),
↪'locations': None, 'population_name': 'l1_exc', 'population_shape': (5, 5, 4),
↪'record_from': ['V_m'], 'type': 'multimeter', 'units_number': 4}
```

Load a recorder's data as pandas dataframe by providing the path to its metadata file

[6]:
```
df = denest.io.load.load(recorder_metadata_path)
print(df[0:5])
```

```
2020-06-30 13:43:45,101 [denest.io.load] INFO: Loading metadata from data/outputs/
↪output/data/my_multimeter_l1_l1_exc.yml
```

```
   gid   time     V_m
0   53  120.0 -54.457
1   54  120.0 -54.457
2   55  120.0 -54.457
3   56  120.0 -54.457
4   57  120.0 -54.457
```

Get all the recorders' metadata with `denest.io.load.metadata_paths()`

[7]:
```
all_recorder_metadata_paths = denest.io.load.metadata_paths(OUTPUT_DIR)
print(all_recorder_metadata_paths)
```

```
[PosixPath('data/outputs/output/data/my_multimeter_l1_l1_exc.yml'), PosixPath('data/
↪outputs/output/data/my_spike_detector_input_layer_parrot_neuron.yml'), PosixPath(
↪'data/outputs/output/data/weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh.yml')]
```

[8]:
```
for metadata_path in all_recorder_metadata_paths:
    print(f'Recorder: {metadata_path.name}')
    print(f'{denest.io.load.load(metadata_path)[0:5]}\n')
```

```
2020-06-30 13:43:45,178 [denest.io.load] INFO: Loading metadata from data/outputs/
↪output/data/my_multimeter_l1_l1_exc.yml
2020-06-30 13:43:45,204 [denest.io.load] INFO: Loading metadata from data/outputs/
↪output/data/my_spike_detector_input_layer_parrot_neuron.yml
2020-06-30 13:43:45,220 [denest.io.load] INFO: Loading metadata from data/outputs/
↪output/data/weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh.yml
```

```
Recorder: my_multimeter_l1_l1_exc.yml
   gid   time     V_m
0   53  120.0 -54.457
1   54  120.0 -54.457
2   55  120.0 -54.457
3   56  120.0 -54.457
4   57  120.0 -54.457

Recorder: my_spike_detector_input_layer_parrot_neuron.yml
   gid   time
0   27  102.0
1   28  102.0
2   29  102.0
3   30  102.0
4   31  102.0

Recorder: weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh.yml
    0    1      2      3   4
0  53  187  104.5  0.898 NaN
1  53  159  104.5  0.898 NaN
2  53  184  104.5  0.898 NaN
3  53  199  104.5  0.898 NaN
4  53  153  104.5  0.898 NaN
```

5.6 Parameter exploration

[1]:
```
import denest
import nest
import yaml
from pathlib import Path
from pprint import pprint
```

[2]:
```
from denest import ParamsTree
```

[3]:
```
# Path to the parameter files to use
PARAMS_PATH = './data/params/tree_paths.yml'
```

5.6.1 Override parameters

Load "base" parameter files

[4]:
```
base_tree = denest.load_trees(PARAMS_PATH)
```

```
2020-06-30 13:44:15,301 [denest] INFO: Loading parameter file paths from data/params/
↪tree_paths.yml
2020-06-30 13:44:15,306 [denest] INFO: Finished loading parameter file paths
2020-06-30 13:44:15,311 [denest] INFO: Loading parameters files:
['./network_tree.yml',
 './simulation.yml',
 './session_models.yml',
 './kernel.yml']
```

Define some overrides from dictionary

[5]:
```
# We can override some parameters loaded from the file
overrides = [

  # Maybe change the nest kernel's settings ?
  {'kernel': {'nest_params': {'local_num_threads': 20}}},

  # Maybe change a parameter for all the projections at once ?
  {'network': {'projection_models': {'nest_params': {
      'allow_autapses': True
  }}}},

]
```

Define some overrides using the `AutoDict` class

The `AutoDict` class provides a convenient way of creating deeply nested dictionaries

[6]:
```
from denest.utils.autodict import AutoDict
```

[7]:
```
override = AutoDict({
    ('kernel', 'nest_params', 'local_num_threads'): 20,
    ('network', 'projection_models', 'nest_params', 'allow_autapses'): 20,
}).todict()

overrides = [override]
```

Load overriden parameter tree

The `denest.load_trees` method supports overrides:

[8]:
```
overriden_tree = denest.load_trees(PARAMS_PATH, *overrides)
```

```
2020-06-30 13:44:20,236 [denest] INFO: Loading parameter file paths from data/params/
↪tree_paths.yml
2020-06-30 13:44:20,237 [denest] INFO: Using 1 override tree(s)
2020-06-30 13:44:20,241 [denest] INFO: Finished loading parameter file paths
2020-06-30 13:44:20,245 [denest] INFO: Loading parameters files:
['./network_tree.yml',
 './simulation.yml',
 './session_models.yml',
 './kernel.yml']
```

The data defined in the overrides were inserted at the corresponding nodes:

[9]:
```
print('Base params:')
print(f"kernel nest_params: ``{base_tree.children['kernel'].nest_params}``")
print(f"root 'projection_models' nest_params: ``{base_tree.children['network'].
↪children['projection_models'].nest_params}``")
print('...')
print('Overriden params:')
print(f"kernel nest_params: ``{overriden_tree.children['kernel'].nest_params}``")
print(f"root 'projection_models' nest_params: ``{overriden_tree.children['network'].
↪children['projection_models'].nest_params}``")
```

```
Base params:
kernel nest_params: ``DeepChainMap({'resolution': 0.5, 'overwrite_files': True}, {})``
root 'projection_models' nest_params: ``DeepChainMap({'connection_type': 'divergent',
↪'mask': {'circular': {'radius': 2.0}}, 'kernel': 1.0}, {}, {})``
...
Overriden params:
kernel nest_params: ``DeepChainMap({'resolution': 0.5, 'overwrite_files': True,
↪'local_num_threads': 20}, {})``
root 'projection_models' nest_params: ``DeepChainMap({'connection_type': 'divergent',
↪'mask': {'circular': {'radius': 2.0}}, 'kernel': 1.0, 'allow_autapses': 20}, {}, {}
↪)``
```

Or merge parameter trees manually using the `ParamsTree.merge` method

[10]:
```
overriden_tree_2 = ParamsTree.merge(
    base_tree,
    *[
        ParamsTree(override)
        for override in overrides
```

(continues on next page)

(continued from previous page)

```
    ]
)
```

[11]:
```
print(overriden_tree_2 == overriden_tree)
```

```
True
```

5.6.2 Perform parameter exploration

We can run multiple simulations using overrides:

[12]:
```
param_names = [
    'g_peak_AMPA_all_neurons',
    'weights_proj_1_AMPA',
    'weights_proj_2_GABAA',
]

# Values of the parameters
explored_values = (
    [0.1, 0.2], # 'g_peak_AMPA_all_neurons',
    [1.0, 2.0], # 'weights_proj_1_AMPA',
    [1.0, 2.0], # 'weights_proj_2_GABA_A',
)

# Position in the tree of the modified params
# Be careful not to make any mistake here!!
params_paths = {
    'g_peak_AMPA_all_neurons': ('network', 'neuron_models', 'my_neuron', 'nest_params
↪', 'g_peak_AMPA'),
    'weights_proj_1_AMPA': ('network', 'projection_models', 'proj_1_AMPA', 'nest_
↪params', 'weights'),
    'weights_proj_2_GABAA': ('network', 'projection_models', 'proj_2_GABAA', 'nest_
↪params', 'weights'),
}
```

[13]:
```
import itertools

# We save all the simulations there
MAIN_OUTPUT_DIR = Path('./data/outputs/output_param_explore')

# Run one simulation for each  combination of parameters
for param_values_combination in itertools.product(*explored_values):

    param_combination = dict(zip(param_names, param_values_combination))
    print(f'param combination: {param_combination}', flush=True)

    # Create override tree to incorporate the parameters
    overrides = [

        # Incorporate parameter exploration params
        AutoDict({
            params_paths[key]: value
            for key, value in param_combination.items()
        }).todict(),
```

(continues on next page)

(continued from previous page)

```
        # Reduce the multimeter interval for disk space
        AutoDict({
            ('network', 'recorder_models', 'my_multimeter', 'nest_params', 'interval
↪'): 50.0
        }).todict(),

    ]
    print(f'Override trees:\n {overrides}', flush=True)

    # Load overriden parameter file
    overriden_tree = denest.load_trees(PARAMS_PATH, *overrides)

    # Define output directory for current simulation
    output_dir = MAIN_OUTPUT_DIR/str(param_combination)
    print(output_dir)

    # Run simulation
    print(f'Running simulation for param combination: {param_combination}')
    sim = denest.Simulation(overriden_tree, output_dir=output_dir)
    sim.run()
    print(f'done\n\n')
```

```
param combination: {'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 1.0}
Override trees:
 [{'network': {'neuron_models': {'my_neuron': {'nest_params': {'g_peak_AMPA': 0.1}}},
↪'projection_models': {'proj_1_AMPA': {'nest_params': {'weights': 1.0}}, 'proj_2_
↪GABAA': {'nest_params': {'weights': 1.0}}}}}, {'network': {'recorder_models': {'my_
↪multimeter': {'nest_params': {'interval': 50.0}}}}}]
```

```
2020-06-30 13:44:23,300 [denest] INFO: Loading parameter file paths from data/params/
↪tree_paths.yml
2020-06-30 13:44:23,301 [denest] INFO: Using 2 override tree(s)
2020-06-30 13:44:23,307 [denest] INFO: Finished loading parameter file paths
2020-06-30 13:44:23,328 [denest] INFO: Loading parameters files:
['./network_tree.yml',
 './simulation.yml',
 './session_models.yml',
 './kernel.yml']
2020-06-30 13:44:23,452 [denest.utils.validation] INFO: Object `simulation`: params:␣
↪using default value for optional parameters:
{'input_dir': 'input'}
2020-06-30 13:44:23,453 [denest.simulation] INFO: Initializing NEST kernel and␣
↪seeds...
2020-06-30 13:44:23,454 [denest.simulation] INFO:   Resetting NEST kernel...
```

```
data/outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_
↪AMPA': 1.0, 'weights_proj_2_GABAA': 1.0}
Running simulation for param combination: {'g_peak_AMPA_all_neurons': 0.1, 'weights_
↪proj_1_AMPA': 1.0, 'weights_proj_2_GABAA': 1.0}
```

```
2020-06-30 13:44:23,487 [denest.simulation] INFO:   Setting NEST kernel status...
2020-06-30 13:44:23,538 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'resolution': 0.5, 'overwrite_files': True})`
2020-06-30 13:44:23,575 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'data_path': "data/outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.1,
↪'weights_proj_1_AMPA': 1.0, 'weights_proj_2_GABAA': 1.0}/data", 'grng_seed': 11,
↪'rng_seeds': range(12, 13)})
2020-06-30 13:44:23,645 [denest.simulation] INFO:   Finished setting NEST kernel␣
↪status
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:23,651 [denest.simulation] INFO:   Installing external modules...
2020-06-30 13:44:23,651 [denest.simulation] INFO:   Finished installing external␣
↪modules
2020-06-30 13:44:23,654 [denest.simulation] INFO: Finished initializing kernel
2020-06-30 13:44:23,678 [denest.simulation] INFO: Build N=3 session models
2020-06-30 13:44:23,683 [denest.simulation] INFO: Build N=4 sessions
2020-06-30 13:44:23,685 [denest.session] INFO: Creating session "00_warmup"
2020-06-30 13:44:23,686 [denest.utils.validation] INFO: Object `00_warmup`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': [], 'unit_changes': []}
2020-06-30 13:44:23,687 [denest.session] INFO: Creating session "01_3_spikes"
2020-06-30 13:44:23,688 [denest.utils.validation] INFO: Object `01_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:23,693 [denest.session] INFO: Creating session "02_2_spikes"
2020-06-30 13:44:23,694 [denest.utils.validation] INFO: Object `02_2_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:23,695 [denest.session] INFO: Creating session "03_3_spikes"
2020-06-30 13:44:23,697 [denest.utils.validation] INFO: Object `03_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:23,699 [denest.simulation] INFO: Sessions: ['00_warmup', '01_3_spikes
↪', '02_2_spikes', '03_3_spikes']
2020-06-30 13:44:23,706 [denest.simulation] INFO: Building network.
2020-06-30 13:44:23,739 [denest.network] INFO: Build N=2 ``Model`` objects
2020-06-30 13:44:23,743 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
2020-06-30 13:44:23,749 [denest.network] INFO: Build N=3 ``Model`` objects
2020-06-30 13:44:23,754 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:44:23,757 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:44:23,762 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:44:23,768 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
2020-06-30 13:44:23,775 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
2020-06-30 13:44:23,782 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:44:23,788 [denest.network] INFO: Build N=1 projection recorders.
2020-06-30 13:44:23,791 [denest.simulation] INFO: Creating network.
2020-06-30 13:44:23,794 [denest.network] INFO: Creating neuron models...
100%|| 2/2 [00:00<00:00, 5785.25it/s]
2020-06-30 13:44:23,821 [denest.network] INFO: Creating synapse models...
100%|| 2/2 [00:00<00:00, 990.16it/s]
2020-06-30 13:44:23,831 [denest.network] INFO: Creating recorder models...
100%|| 3/3 [00:00<00:00, 315.73it/s]
2020-06-30 13:44:23,862 [denest.network] INFO: Creating layers...
  0%|          | 0/2 [00:00<?, ?it/s]/Users/tom/nest/nest-simulator-2.20.0/lib/
↪python3.7/site-packages/nest/lib/hl_api_helper.py:127: UserWarning:
GetNodes is deprecated and will be removed in NEST 3.0. Use            GIDCollection␣
↪instead.
100%|| 2/2 [00:00<00:00,  5.23it/s]
2020-06-30 13:44:24,260 [denest.network] INFO: Creating population recorders...
100%|| 2/2 [00:00<00:00, 34.38it/s]
2020-06-30 13:44:24,338 [denest.network] INFO: Creating projection recorders...
100%|| 1/1 [00:00<00:00, 123.55it/s]
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:24,361 [denest.network] INFO: Connecting layers...
100%|| 3/3 [00:00<00:00, 94.15it/s]
2020-06-30 13:44:24,413 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 206
Number of projections: 6650
2020-06-30 13:44:24,419 [denest.simulation] INFO: Finished creating network
2020-06-30 13:44:24,420 [denest.simulation] INFO: Saving simulation metadata...
2020-06-30 13:44:24,425 [denest.simulation] INFO: Creating output directory: data/
↪outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA':
↪ 1.0, 'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:24,432 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:24,443 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:24,448 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 1.0}/data
2020-06-30 13:44:24,457 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 1.0}/data
2020-06-30 13:44:24,466 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 1.0}/data
2020-06-30 13:44:24,472 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:24,536 [denest.simulation] INFO: Finished saving simulation metadata
2020-06-30 13:44:24,538 [denest.simulation] INFO: Running 4 sessions...
2020-06-30 13:44:24,565 [denest.simulation] INFO: Running session: '00_warmup'...
2020-06-30 13:44:24,571 [denest.session] INFO: Initializing session...
2020-06-30 13:44:24,580 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_multimeter_l1_l1_exc: {'start': 100.0}
2020-06-30 13:44:24,582 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_spike_detector_input_layer_parrot_neuron: {'start': 100.0}
2020-06-30 13:44:24,586 [denest.network.recorders] INFO:   Setting status for␣
↪recorder weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh: {'start': 100.0}
2020-06-30 13:44:24,589 [denest.session] INFO: Setting `origin` flag to `0.0` for all␣
↪stimulation devices in ``InputLayers`` for session `00_warmup`
2020-06-30 13:44:24,598 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:24,599 [denest.session] INFO: Running session '00_warmup' for 100 ms
2020-06-30 13:44:24,847 [denest.session] INFO: Finished running session
2020-06-30 13:44:24,848 [denest.session] INFO: Session '00_warmup' virtual running␣
↪time: 100 ms
2020-06-30 13:44:24,848 [denest.session] INFO: Session '00_warmup' real running time:␣
↪0h:00m:00s
2020-06-30 13:44:24,850 [denest.simulation] INFO: Done running session '00_warmup'
2020-06-30 13:44:24,851 [denest.simulation] INFO: Running session: '01_3_spikes'...
2020-06-30 13:44:24,882 [denest.session] INFO: Initializing session...
2020-06-30 13:44:24,899 [denest.session] INFO: Setting `origin` flag to `100.0` for␣
↪all stimulation devices in ``InputLayers`` for session `01_3_spikes`
2020-06-30 13:44:24,908 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:24,915 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:25,043 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:25,044 [denest.session] INFO: Running session '01_3_spikes' for 100␣
↪ms
2020-06-30 13:44:25,207 [denest.session] INFO: Finished running session
2020-06-30 13:44:25,215 [denest.session] INFO: Session '01_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:25,224 [denest.session] INFO: Session '01_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:25,246 [denest.simulation] INFO: Done running session '01_3_spikes'
2020-06-30 13:44:25,250 [denest.simulation] INFO: Running session: '02_2_spikes'...
2020-06-30 13:44:25,277 [denest.session] INFO: Initializing session...
2020-06-30 13:44:25,282 [denest.session] INFO: Setting `origin` flag to `200.0` for␣
↪all stimulation devices in ``InputLayers`` for session `02_2_spikes`
2020-06-30 13:44:25,296 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:25,298 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:25,408 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:25,408 [denest.session] INFO: Running session '02_2_spikes' for 100␣
↪ms
2020-06-30 13:44:25,612 [denest.session] INFO: Finished running session
2020-06-30 13:44:25,613 [denest.session] INFO: Session '02_2_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:25,614 [denest.session] INFO: Session '02_2_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:25,649 [denest.simulation] INFO: Done running session '02_2_spikes'
2020-06-30 13:44:25,655 [denest.simulation] INFO: Running session: '03_3_spikes'...
2020-06-30 13:44:25,657 [denest.session] INFO: Initializing session...
2020-06-30 13:44:25,739 [denest.session] INFO: Setting `origin` flag to `300.0` for␣
↪all stimulation devices in ``InputLayers`` for session `03_3_spikes`
2020-06-30 13:44:25,750 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:25,763 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:25,910 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:25,911 [denest.session] INFO: Running session '03_3_spikes' for 100␣
↪ms
2020-06-30 13:44:26,062 [denest.session] INFO: Finished running session
2020-06-30 13:44:26,063 [denest.session] INFO: Session '03_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:26,064 [denest.session] INFO: Session '03_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:26,065 [denest.simulation] INFO: Done running session '03_3_spikes'
2020-06-30 13:44:26,066 [denest.simulation] INFO: Finished running simulation
done


param combination: {'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 2.0}
```

(continues on next page)

(continued from previous page)

```
Override trees:
 [{'network': {'neuron_models': {'my_neuron': {'nest_params': {'g_peak_AMPA': 0.1}}},
↪'projection_models': {'proj_1_AMPA': {'nest_params': {'weights': 1.0}}, 'proj_2_
↪GABAA': {'nest_params': {'weights': 2.0}}}}}, {'network': {'recorder_models': {'my_
↪multimeter': {'nest_params': {'interval': 50.0}}}}}]
```

```
2020-06-30 13:44:26,072 [denest] INFO: Loading parameter file paths from data/params/
↪tree_paths.yml
2020-06-30 13:44:26,074 [denest] INFO: Using 2 override tree(s)
2020-06-30 13:44:26,091 [denest] INFO: Finished loading parameter file paths
2020-06-30 13:44:26,097 [denest] INFO: Loading parameters files:
['./network_tree.yml',
 './simulation.yml',
 './session_models.yml',
 './kernel.yml']
2020-06-30 13:44:26,165 [denest.utils.validation] INFO: Object `simulation`: params:␣
↪using default value for optional parameters:
{'input_dir': 'input'}
2020-06-30 13:44:26,166 [denest.simulation] INFO: Initializing NEST kernel and␣
↪seeds...
2020-06-30 13:44:26,167 [denest.simulation] INFO:   Resetting NEST kernel...
2020-06-30 13:44:26,201 [denest.simulation] INFO:   Setting NEST kernel status...
2020-06-30 13:44:26,203 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'resolution': 0.5, 'overwrite_files': True})`
```

```
data/outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_
↪AMPA': 1.0, 'weights_proj_2_GABAA': 2.0}
Running simulation for param combination: {'g_peak_AMPA_all_neurons': 0.1, 'weights_
↪proj_1_AMPA': 1.0, 'weights_proj_2_GABAA': 2.0}
```

```
2020-06-30 13:44:26,266 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'data_path': "data/outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.1,
↪'weights_proj_1_AMPA': 1.0, 'weights_proj_2_GABAA': 2.0}/data", 'grng_seed': 11,
↪'rng_seeds': range(12, 13)})
2020-06-30 13:44:26,298 [denest.simulation] INFO:   Finished setting NEST kernel␣
↪status
2020-06-30 13:44:26,301 [denest.simulation] INFO:   Installing external modules...
2020-06-30 13:44:26,303 [denest.simulation] INFO:   Finished installing external␣
↪modules
2020-06-30 13:44:26,304 [denest.simulation] INFO: Finished initializing kernel
2020-06-30 13:44:26,333 [denest.simulation] INFO: Build N=3 session models
2020-06-30 13:44:26,338 [denest.simulation] INFO: Build N=4 sessions
2020-06-30 13:44:26,355 [denest.session] INFO: Creating session "00_warmup"
2020-06-30 13:44:26,360 [denest.utils.validation] INFO: Object `00_warmup`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': [], 'unit_changes': []}
2020-06-30 13:44:26,365 [denest.session] INFO: Creating session "01_3_spikes"
2020-06-30 13:44:26,369 [denest.utils.validation] INFO: Object `01_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:26,371 [denest.session] INFO: Creating session "02_2_spikes"
2020-06-30 13:44:26,375 [denest.utils.validation] INFO: Object `02_2_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:26,382 [denest.session] INFO: Creating session "03_3_spikes"
2020-06-30 13:44:26,384 [denest.utils.validation] INFO: Object `03_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:26,386 [denest.simulation] INFO: Sessions: ['00_warmup', '01_3_spikes
↪', '02_2_spikes', '03_3_spikes']
2020-06-30 13:44:26,390 [denest.simulation] INFO: Building network.
2020-06-30 13:44:26,409 [denest.network] INFO: Build N=2 ``Model`` objects
2020-06-30 13:44:26,412 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
2020-06-30 13:44:26,414 [denest.network] INFO: Build N=3 ``Model`` objects
2020-06-30 13:44:26,421 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:44:26,423 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:44:26,424 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:44:26,425 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
2020-06-30 13:44:26,444 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
2020-06-30 13:44:26,447 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:44:26,451 [denest.network] INFO: Build N=1 projection recorders.
2020-06-30 13:44:26,453 [denest.simulation] INFO: Creating network.
2020-06-30 13:44:26,456 [denest.network] INFO: Creating neuron models...
100%|| 2/2 [00:00<00:00, 644.53it/s]
2020-06-30 13:44:26,470 [denest.network] INFO: Creating synapse models...
100%|| 2/2 [00:00<00:00, 1138.83it/s]
2020-06-30 13:44:26,487 [denest.network] INFO: Creating recorder models...
100%|| 3/3 [00:00<00:00, 977.85it/s]
2020-06-30 13:44:26,521 [denest.network] INFO: Creating layers...
100%|| 2/2 [00:00<00:00, 10.35it/s]
2020-06-30 13:44:26,732 [denest.network] INFO: Creating population recorders...
100%|| 2/2 [00:00<00:00, 111.28it/s]
2020-06-30 13:44:26,765 [denest.network] INFO: Creating projection recorders...
100%|| 1/1 [00:00<00:00, 350.49it/s]
2020-06-30 13:44:26,807 [denest.network] INFO: Connecting layers...
100%|| 3/3 [00:00<00:00, 566.03it/s]
2020-06-30 13:44:26,826 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 206
Number of projections: 6650
2020-06-30 13:44:26,831 [denest.simulation] INFO: Finished creating network
2020-06-30 13:44:26,834 [denest.simulation] INFO: Saving simulation metadata...
2020-06-30 13:44:26,837 [denest.simulation] INFO: Creating output directory: data/
↪outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA':
↪ 1.0, 'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:26,841 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:26,843 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:26,853 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 2.0}/data
2020-06-30 13:44:26,873 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 2.0}/data
2020-06-30 13:44:26,883 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 2.0}/data
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:26,891 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:26,995 [denest.simulation] INFO: Finished saving simulation metadata
2020-06-30 13:44:26,996 [denest.simulation] INFO: Running 4 sessions...
2020-06-30 13:44:26,996 [denest.simulation] INFO: Running session: '00_warmup'...
2020-06-30 13:44:26,999 [denest.session] INFO: Initializing session...
2020-06-30 13:44:27,003 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_multimeter_l1_l1_exc: {'start': 100.0}
2020-06-30 13:44:27,007 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_spike_detector_input_layer_parrot_neuron: {'start': 100.0}
2020-06-30 13:44:27,010 [denest.network.recorders] INFO:   Setting status for␣
↪recorder weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh: {'start': 100.0}
2020-06-30 13:44:27,012 [denest.session] INFO: Setting `origin` flag to `0.0` for all␣
↪stimulation devices in ``InputLayers`` for session `00_warmup`
2020-06-30 13:44:27,022 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:27,024 [denest.session] INFO: Running session '00_warmup' for 100 ms
2020-06-30 13:44:27,179 [denest.session] INFO: Finished running session
2020-06-30 13:44:27,181 [denest.session] INFO: Session '00_warmup' virtual running␣
↪time: 100 ms
2020-06-30 13:44:27,184 [denest.session] INFO: Session '00_warmup' real running time:␣
↪0h:00m:00s
2020-06-30 13:44:27,186 [denest.simulation] INFO: Done running session '00_warmup'
2020-06-30 13:44:27,188 [denest.simulation] INFO: Running session: '01_3_spikes'...
2020-06-30 13:44:27,194 [denest.session] INFO: Initializing session...
2020-06-30 13:44:27,196 [denest.session] INFO: Setting `origin` flag to `100.0` for␣
↪all stimulation devices in ``InputLayers`` for session `01_3_spikes`
2020-06-30 13:44:27,212 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:27,214 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:27,348 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:27,401 [denest.session] INFO: Running session '01_3_spikes' for 100␣
↪ms
2020-06-30 13:44:27,840 [denest.session] INFO: Finished running session
2020-06-30 13:44:27,842 [denest.session] INFO: Session '01_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:27,844 [denest.session] INFO: Session '01_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:27,898 [denest.simulation] INFO: Done running session '01_3_spikes'
2020-06-30 13:44:27,900 [denest.simulation] INFO: Running session: '02_2_spikes'...
2020-06-30 13:44:27,901 [denest.session] INFO: Initializing session...
2020-06-30 13:44:27,912 [denest.session] INFO: Setting `origin` flag to `200.0` for␣
↪all stimulation devices in ``InputLayers`` for session `02_2_spikes`
2020-06-30 13:44:27,936 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:27,944 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:28,063 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:28,064 [denest.session] INFO: Running session '02_2_spikes' for 100␣
↪ms
2020-06-30 13:44:28,213 [denest.session] INFO: Finished running session
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:28,214 [denest.session] INFO: Session '02_2_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:28,215 [denest.session] INFO: Session '02_2_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:28,218 [denest.simulation] INFO: Done running session '02_2_spikes'
2020-06-30 13:44:28,219 [denest.simulation] INFO: Running session: '03_3_spikes'...
2020-06-30 13:44:28,220 [denest.session] INFO: Initializing session...
2020-06-30 13:44:28,221 [denest.session] INFO: Setting `origin` flag to `300.0` for␣
↪all stimulation devices in ``InputLayers`` for session `03_3_spikes`
2020-06-30 13:44:28,247 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:28,248 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:28,321 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:28,322 [denest.session] INFO: Running session '03_3_spikes' for 100␣
↪ms
2020-06-30 13:44:28,475 [denest.session] INFO: Finished running session
2020-06-30 13:44:28,476 [denest.session] INFO: Session '03_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:28,477 [denest.session] INFO: Session '03_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:28,509 [denest.simulation] INFO: Done running session '03_3_spikes'
2020-06-30 13:44:28,510 [denest.simulation] INFO: Finished running simulation
```

```
done


param combination: {'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 1.0}
Override trees:
 [{'network': {'neuron_models': {'my_neuron': {'nest_params': {'g_peak_AMPA': 0.1}}},
↪'projection_models': {'proj_1_AMPA': {'nest_params': {'weights': 2.0}}, 'proj_2_
↪GABAA': {'nest_params': {'weights': 1.0}}}}}, {'network': {'recorder_models': {'my_
↪multimeter': {'nest_params': {'interval': 50.0}}}}}]
```

```
2020-06-30 13:44:28,522 [denest] INFO: Loading parameter file paths from data/params/
↪tree_paths.yml
2020-06-30 13:44:28,527 [denest] INFO: Using 2 override tree(s)
2020-06-30 13:44:28,538 [denest] INFO: Finished loading parameter file paths
2020-06-30 13:44:28,540 [denest] INFO: Loading parameters files:
['./network_tree.yml',
 './simulation.yml',
 './session_models.yml',
 './kernel.yml']
2020-06-30 13:44:28,657 [denest.utils.validation] INFO: Object `simulation`: params:␣
↪using default value for optional parameters:
{'input_dir': 'input'}
2020-06-30 13:44:28,662 [denest.simulation] INFO: Initializing NEST kernel and␣
↪seeds...
2020-06-30 13:44:28,663 [denest.simulation] INFO:   Resetting NEST kernel...
2020-06-30 13:44:28,674 [denest.simulation] INFO:   Setting NEST kernel status...
2020-06-30 13:44:28,696 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'resolution': 0.5, 'overwrite_files': True})`
```

```
data/outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_
↪AMPA': 2.0, 'weights_proj_2_GABAA': 1.0}
Running simulation for param combination: {'g_peak_AMPA_all_neurons': 0.1, 'weights_
↪proj_1_AMPA': 2.0, 'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:28,790 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'data_path': "data/outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.1,
↪'weights_proj_1_AMPA': 2.0, 'weights_proj_2_GABAA': 1.0}/data", 'grng_seed': 11,
↪'rng_seeds': range(12, 13)})
2020-06-30 13:44:28,795 [denest.simulation] INFO:   Finished setting NEST kernel␣
↪status
2020-06-30 13:44:28,796 [denest.simulation] INFO:   Installing external modules...
2020-06-30 13:44:28,839 [denest.simulation] INFO:   Finished installing external␣
↪modules
2020-06-30 13:44:28,841 [denest.simulation] INFO: Finished initializing kernel
2020-06-30 13:44:28,845 [denest.simulation] INFO: Build N=3 session models
2020-06-30 13:44:28,848 [denest.simulation] INFO: Build N=4 sessions
2020-06-30 13:44:28,851 [denest.session] INFO: Creating session "00_warmup"
2020-06-30 13:44:28,856 [denest.utils.validation] INFO: Object `00_warmup`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': [], 'unit_changes': []}
2020-06-30 13:44:28,865 [denest.session] INFO: Creating session "01_3_spikes"
2020-06-30 13:44:28,867 [denest.utils.validation] INFO: Object `01_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:28,868 [denest.session] INFO: Creating session "02_2_spikes"
2020-06-30 13:44:28,877 [denest.utils.validation] INFO: Object `02_2_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:28,881 [denest.session] INFO: Creating session "03_3_spikes"
2020-06-30 13:44:28,884 [denest.utils.validation] INFO: Object `03_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:28,886 [denest.simulation] INFO: Sessions: ['00_warmup', '01_3_spikes
↪', '02_2_spikes', '03_3_spikes']
2020-06-30 13:44:28,889 [denest.simulation] INFO: Building network.
2020-06-30 13:44:28,951 [denest.network] INFO: Build N=2 ``Model`` objects
2020-06-30 13:44:28,953 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
2020-06-30 13:44:28,956 [denest.network] INFO: Build N=3 ``Model`` objects
2020-06-30 13:44:28,960 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:44:28,961 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:44:28,979 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:44:28,985 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
2020-06-30 13:44:28,992 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
2020-06-30 13:44:28,998 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:44:29,001 [denest.network] INFO: Build N=1 projection recorders.
2020-06-30 13:44:29,008 [denest.simulation] INFO: Creating network.
2020-06-30 13:44:29,012 [denest.network] INFO: Creating neuron models...
100%|| 2/2 [00:00<00:00, 1425.66it/s]
2020-06-30 13:44:29,028 [denest.network] INFO: Creating synapse models...
100%|| 2/2 [00:00<00:00, 1391.61it/s]
2020-06-30 13:44:29,034 [denest.network] INFO: Creating recorder models...
100%|| 3/3 [00:00<00:00, 1615.47it/s]
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:29,051 [denest.network] INFO: Creating layers...
100%|| 2/2 [00:00<00:00,  7.51it/s]
2020-06-30 13:44:29,321 [denest.network] INFO: Creating population recorders...
100%|| 2/2 [00:00<00:00, 80.98it/s]
2020-06-30 13:44:29,355 [denest.network] INFO: Creating projection recorders...
100%|| 1/1 [00:00<00:00, 172.00it/s]
2020-06-30 13:44:29,364 [denest.network] INFO: Connecting layers...
100%|| 3/3 [00:00<00:00, 234.88it/s]
2020-06-30 13:44:29,388 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 206
Number of projections: 6650
2020-06-30 13:44:29,394 [denest.simulation] INFO: Finished creating network
2020-06-30 13:44:29,400 [denest.simulation] INFO: Saving simulation metadata...
2020-06-30 13:44:29,402 [denest.simulation] INFO: Creating output directory: data/
↪outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA':
↪ 2.0, 'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:29,403 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:29,406 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:29,419 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 1.0}/data
2020-06-30 13:44:29,424 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 1.0}/data
2020-06-30 13:44:29,429 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 1.0}/data
2020-06-30 13:44:29,438 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:29,548 [denest.simulation] INFO: Finished saving simulation metadata
2020-06-30 13:44:29,549 [denest.simulation] INFO: Running 4 sessions...
2020-06-30 13:44:29,555 [denest.simulation] INFO: Running session: '00_warmup'...
2020-06-30 13:44:29,563 [denest.session] INFO: Initializing session...
2020-06-30 13:44:29,565 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_multimeter_l1_l1_exc: {'start': 100.0}
2020-06-30 13:44:29,599 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_spike_detector_input_layer_parrot_neuron: {'start': 100.0}
2020-06-30 13:44:29,601 [denest.network.recorders] INFO:   Setting status for␣
↪recorder weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh: {'start': 100.0}
2020-06-30 13:44:29,602 [denest.session] INFO: Setting `origin` flag to `0.0` for all␣
↪stimulation devices in ``InputLayers`` for session `00_warmup`
2020-06-30 13:44:29,613 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:29,615 [denest.session] INFO: Running session '00_warmup' for 100 ms
2020-06-30 13:44:29,866 [denest.session] INFO: Finished running session
2020-06-30 13:44:29,867 [denest.session] INFO: Session '00_warmup' virtual running␣
↪time: 100 ms
2020-06-30 13:44:29,868 [denest.session] INFO: Session '00_warmup' real running time:␣
↪0h:00m:00s
2020-06-30 13:44:29,870 [denest.simulation] INFO: Done running session '00_warmup'
2020-06-30 13:44:29,872 [denest.simulation] INFO: Running session: '01_3_spikes'...
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:29,873 [denest.session] INFO: Initializing session...
2020-06-30 13:44:29,879 [denest.session] INFO: Setting `origin` flag to `100.0` for␣
↪all stimulation devices in ``InputLayers`` for session `01_3_spikes`
2020-06-30 13:44:29,914 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:29,922 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:30,017 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:30,018 [denest.session] INFO: Running session '01_3_spikes' for 100␣
↪ms
2020-06-30 13:44:30,299 [denest.session] INFO: Finished running session
2020-06-30 13:44:30,300 [denest.session] INFO: Session '01_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:30,300 [denest.session] INFO: Session '01_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:30,301 [denest.simulation] INFO: Done running session '01_3_spikes'
2020-06-30 13:44:30,302 [denest.simulation] INFO: Running session: '02_2_spikes'...
2020-06-30 13:44:30,318 [denest.session] INFO: Initializing session...
2020-06-30 13:44:30,326 [denest.session] INFO: Setting `origin` flag to `200.0` for␣
↪all stimulation devices in ``InputLayers`` for session `02_2_spikes`
2020-06-30 13:44:30,340 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:30,351 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:30,432 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:30,432 [denest.session] INFO: Running session '02_2_spikes' for 100␣
↪ms
2020-06-30 13:44:30,623 [denest.session] INFO: Finished running session
2020-06-30 13:44:30,669 [denest.session] INFO: Session '02_2_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:30,750 [denest.session] INFO: Session '02_2_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:30,751 [denest.simulation] INFO: Done running session '02_2_spikes'
2020-06-30 13:44:30,752 [denest.simulation] INFO: Running session: '03_3_spikes'...
2020-06-30 13:44:30,756 [denest.session] INFO: Initializing session...
2020-06-30 13:44:30,770 [denest.session] INFO: Setting `origin` flag to `300.0` for␣
↪all stimulation devices in ``InputLayers`` for session `03_3_spikes`
2020-06-30 13:44:30,787 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:30,789 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:30,951 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:30,952 [denest.session] INFO: Running session '03_3_spikes' for 100␣
↪ms
2020-06-30 13:44:31,203 [denest.session] INFO: Finished running session
2020-06-30 13:44:31,204 [denest.session] INFO: Session '03_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:31,205 [denest.session] INFO: Session '03_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:31,207 [denest.simulation] INFO: Done running session '03_3_spikes'
2020-06-30 13:44:31,236 [denest.simulation] INFO: Finished running simulation
```

```
done


param combination: {'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 2.0}
Override trees:
 [{'network': {'neuron_models': {'my_neuron': {'nest_params': {'g_peak_AMPA': 0.1}}},
↪'projection_models': {'proj_1_AMPA': {'nest_params': {'weights': 2.0}}, 'proj_2_
↪GABAA': {'nest_params': {'weights': 2.0}}}}}, {'network': {'recorder_models': {'my_
↪multimeter': {'nest_params': {'interval': 50.0}}}}}]
2020-06-30 13:44:31,249 [denest] INFO: Loading parameter file paths from data/params/
↪tree_paths.yml
2020-06-30 13:44:31,251 [denest] INFO: Using 2 override tree(s)
2020-06-30 13:44:31,266 [denest] INFO: Finished loading parameter file paths
2020-06-30 13:44:31,267 [denest] INFO: Loading parameters files:
['./network_tree.yml',
 './simulation.yml',
 './session_models.yml',
 './kernel.yml']
2020-06-30 13:44:31,387 [denest.utils.validation] INFO: Object `simulation`: params:␣
↪using default value for optional parameters:
{'input_dir': 'input'}
2020-06-30 13:44:31,390 [denest.simulation] INFO: Initializing NEST kernel and␣
↪seeds...
2020-06-30 13:44:31,392 [denest.simulation] INFO:   Resetting NEST kernel...
data/outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_
↪AMPA': 2.0, 'weights_proj_2_GABAA': 2.0}
Running simulation for param combination: {'g_peak_AMPA_all_neurons': 0.1, 'weights_
↪proj_1_AMPA': 2.0, 'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:31,440 [denest.simulation] INFO:   Setting NEST kernel status...
2020-06-30 13:44:31,478 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'resolution': 0.5, 'overwrite_files': True})`
2020-06-30 13:44:31,483 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'data_path': "data/outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.1,
↪'weights_proj_1_AMPA': 2.0, 'weights_proj_2_GABAA': 2.0}/data", 'grng_seed': 11,
↪'rng_seeds': range(12, 13)})
2020-06-30 13:44:31,533 [denest.simulation] INFO:   Finished setting NEST kernel␣
↪status
2020-06-30 13:44:31,536 [denest.simulation] INFO:   Installing external modules...
2020-06-30 13:44:31,539 [denest.simulation] INFO:   Finished installing external␣
↪modules
2020-06-30 13:44:31,547 [denest.simulation] INFO: Finished initializing kernel
2020-06-30 13:44:31,549 [denest.simulation] INFO: Build N=3 session models
2020-06-30 13:44:31,551 [denest.simulation] INFO: Build N=4 sessions
2020-06-30 13:44:31,581 [denest.session] INFO: Creating session "00_warmup"
2020-06-30 13:44:31,598 [denest.utils.validation] INFO: Object `00_warmup`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': [], 'unit_changes': []}
2020-06-30 13:44:31,602 [denest.session] INFO: Creating session "01_3_spikes"
2020-06-30 13:44:31,604 [denest.utils.validation] INFO: Object `01_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:31,606 [denest.session] INFO: Creating session "02_2_spikes"
2020-06-30 13:44:31,611 [denest.utils.validation] INFO: Object `02_2_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:31,623 [denest.session] INFO: Creating session "03_3_spikes"
2020-06-30 13:44:31,625 [denest.utils.validation] INFO: Object `03_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:31,660 [denest.simulation] INFO: Sessions: ['00_warmup', '01_3_spikes
↪', '02_2_spikes', '03_3_spikes']
2020-06-30 13:44:31,688 [denest.simulation] INFO: Building network.
2020-06-30 13:44:31,723 [denest.network] INFO: Build N=2 ``Model`` objects
2020-06-30 13:44:31,726 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
2020-06-30 13:44:31,731 [denest.network] INFO: Build N=3 ``Model`` objects
2020-06-30 13:44:31,734 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:44:31,737 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:44:31,740 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:44:31,743 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
2020-06-30 13:44:31,755 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
2020-06-30 13:44:31,763 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:44:31,764 [denest.network] INFO: Build N=1 projection recorders.
2020-06-30 13:44:31,766 [denest.simulation] INFO: Creating network.
2020-06-30 13:44:31,769 [denest.network] INFO: Creating neuron models...
100%|| 2/2 [00:00<00:00, 1111.07it/s]
2020-06-30 13:44:31,785 [denest.network] INFO: Creating synapse models...
100%|| 2/2 [00:00<00:00, 2353.71it/s]
2020-06-30 13:44:31,798 [denest.network] INFO: Creating recorder models...
100%|| 3/3 [00:00<00:00, 89.70it/s]
2020-06-30 13:44:31,838 [denest.network] INFO: Creating layers...
100%|| 2/2 [00:00<00:00,  6.30it/s]
2020-06-30 13:44:32,165 [denest.network] INFO: Creating population recorders...
100%|| 2/2 [00:00<00:00, 84.42it/s]
2020-06-30 13:44:32,192 [denest.network] INFO: Creating projection recorders...
100%|| 1/1 [00:00<00:00, 491.60it/s]
2020-06-30 13:44:32,199 [denest.network] INFO: Connecting layers...
100%|| 3/3 [00:00<00:00, 640.48it/s]
2020-06-30 13:44:32,211 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 206
Number of projections: 6650
2020-06-30 13:44:32,211 [denest.simulation] INFO: Finished creating network
2020-06-30 13:44:32,212 [denest.simulation] INFO: Saving simulation metadata...
2020-06-30 13:44:32,213 [denest.simulation] INFO: Creating output directory: data/
↪outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA':
↪ 2.0, 'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:32,213 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:32,215 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:32,216 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 2.0}/data
2020-06-30 13:44:32,218 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 2.0}/data
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:32,220 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 2.0}/data
2020-06-30 13:44:32,221 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:32,267 [denest.simulation] INFO: Finished saving simulation metadata
2020-06-30 13:44:32,267 [denest.simulation] INFO: Running 4 sessions...
2020-06-30 13:44:32,268 [denest.simulation] INFO: Running session: '00_warmup'...
2020-06-30 13:44:32,269 [denest.session] INFO: Initializing session...
2020-06-30 13:44:32,270 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_multimeter_l1_l1_exc: {'start': 100.0}
2020-06-30 13:44:32,273 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_spike_detector_input_layer_parrot_neuron: {'start': 100.0}
2020-06-30 13:44:32,277 [denest.network.recorders] INFO:   Setting status for␣
↪recorder weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh: {'start': 100.0}
2020-06-30 13:44:32,281 [denest.session] INFO: Setting `origin` flag to `0.0` for all␣
↪stimulation devices in ``InputLayers`` for session `00_warmup`
2020-06-30 13:44:32,295 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:32,296 [denest.session] INFO: Running session '00_warmup' for 100 ms
2020-06-30 13:44:32,394 [denest.session] INFO: Finished running session
2020-06-30 13:44:32,395 [denest.session] INFO: Session '00_warmup' virtual running␣
↪time: 100 ms
2020-06-30 13:44:32,397 [denest.session] INFO: Session '00_warmup' real running time:␣
↪0h:00m:00s
2020-06-30 13:44:32,400 [denest.simulation] INFO: Done running session '00_warmup'
2020-06-30 13:44:32,403 [denest.simulation] INFO: Running session: '01_3_spikes'...
2020-06-30 13:44:32,407 [denest.session] INFO: Initializing session...
2020-06-30 13:44:32,410 [denest.session] INFO: Setting `origin` flag to `100.0` for␣
↪all stimulation devices in ``InputLayers`` for session `01_3_spikes`
2020-06-30 13:44:32,426 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:32,427 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:32,487 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:32,488 [denest.session] INFO: Running session '01_3_spikes' for 100␣
↪ms
2020-06-30 13:44:32,748 [denest.session] INFO: Finished running session
2020-06-30 13:44:32,753 [denest.session] INFO: Session '01_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:32,756 [denest.session] INFO: Session '01_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:32,767 [denest.simulation] INFO: Done running session '01_3_spikes'
2020-06-30 13:44:32,769 [denest.simulation] INFO: Running session: '02_2_spikes'...
2020-06-30 13:44:32,773 [denest.session] INFO: Initializing session...
2020-06-30 13:44:32,774 [denest.session] INFO: Setting `origin` flag to `200.0` for␣
↪all stimulation devices in ``InputLayers`` for session `02_2_spikes`
2020-06-30 13:44:32,790 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:32,795 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:32,914 [denest.session] INFO: Finished initializing session
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:32,915 [denest.session] INFO: Running session '02_2_spikes' for 100␣
↪ms
2020-06-30 13:44:33,058 [denest.session] INFO: Finished running session
2020-06-30 13:44:33,059 [denest.session] INFO: Session '02_2_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:33,060 [denest.session] INFO: Session '02_2_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:33,068 [denest.simulation] INFO: Done running session '02_2_spikes'
2020-06-30 13:44:33,072 [denest.simulation] INFO: Running session: '03_3_spikes'...
2020-06-30 13:44:33,074 [denest.session] INFO: Initializing session...
2020-06-30 13:44:33,078 [denest.session] INFO: Setting `origin` flag to `300.0` for␣
↪all stimulation devices in ``InputLayers`` for session `03_3_spikes`
2020-06-30 13:44:33,085 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:33,088 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:33,149 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:33,150 [denest.session] INFO: Running session '03_3_spikes' for 100␣
↪ms
2020-06-30 13:44:33,302 [denest.session] INFO: Finished running session
2020-06-30 13:44:33,303 [denest.session] INFO: Session '03_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:33,304 [denest.session] INFO: Session '03_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:33,307 [denest.simulation] INFO: Done running session '03_3_spikes'
2020-06-30 13:44:33,308 [denest.simulation] INFO: Finished running simulation
```

```
done


param combination: {'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 1.0}
Override trees:
 [{'network': {'neuron_models': {'my_neuron': {'nest_params': {'g_peak_AMPA': 0.2}}},
↪'projection_models': {'proj_1_AMPA': {'nest_params': {'weights': 1.0}}, 'proj_2_
↪GABAA': {'nest_params': {'weights': 1.0}}}}}, {'network': {'recorder_models': {'my_
↪multimeter': {'nest_params': {'interval': 50.0}}}}}]
```

```
2020-06-30 13:44:33,330 [denest] INFO: Loading parameter file paths from data/params/
↪tree_paths.yml
2020-06-30 13:44:33,332 [denest] INFO: Using 2 override tree(s)
2020-06-30 13:44:33,338 [denest] INFO: Finished loading parameter file paths
2020-06-30 13:44:33,340 [denest] INFO: Loading parameters files:
['./network_tree.yml',
 './simulation.yml',
 './session_models.yml',
 './kernel.yml']
2020-06-30 13:44:33,399 [denest.utils.validation] INFO: Object `simulation`: params:␣
↪using default value for optional parameters:
{'input_dir': 'input'}
2020-06-30 13:44:33,401 [denest.simulation] INFO: Initializing NEST kernel and␣
↪seeds...
2020-06-30 13:44:33,403 [denest.simulation] INFO:   Resetting NEST kernel...
2020-06-30 13:44:33,413 [denest.simulation] INFO:   Setting NEST kernel status...
2020-06-30 13:44:33,491 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'resolution': 0.5, 'overwrite_files': True})`
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:33,499 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'data_path': "data/outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.2,
↪'weights_proj_1_AMPA': 1.0, 'weights_proj_2_GABAA': 1.0}/data", 'grng_seed': 11,
↪'rng_seeds': range(12, 13)})
2020-06-30 13:44:33,507 [denest.simulation] INFO:   Finished setting NEST kernel␣
↪status
```

```
data/outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_
↪AMPA': 1.0, 'weights_proj_2_GABAA': 1.0}
Running simulation for param combination: {'g_peak_AMPA_all_neurons': 0.2, 'weights_
↪proj_1_AMPA': 1.0, 'weights_proj_2_GABAA': 1.0}
```

```
2020-06-30 13:44:33,537 [denest.simulation] INFO:   Installing external modules...
2020-06-30 13:44:33,545 [denest.simulation] INFO:   Finished installing external␣
↪modules
2020-06-30 13:44:33,549 [denest.simulation] INFO: Finished initializing kernel
2020-06-30 13:44:33,551 [denest.simulation] INFO: Build N=3 session models
2020-06-30 13:44:33,553 [denest.simulation] INFO: Build N=4 sessions
2020-06-30 13:44:33,555 [denest.session] INFO: Creating session "00_warmup"
2020-06-30 13:44:33,560 [denest.utils.validation] INFO: Object `00_warmup`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': [], 'unit_changes': []}
2020-06-30 13:44:33,562 [denest.session] INFO: Creating session "01_3_spikes"
2020-06-30 13:44:33,564 [denest.utils.validation] INFO: Object `01_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:33,567 [denest.session] INFO: Creating session "02_2_spikes"
2020-06-30 13:44:33,570 [denest.utils.validation] INFO: Object `02_2_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:33,572 [denest.session] INFO: Creating session "03_3_spikes"
2020-06-30 13:44:33,575 [denest.utils.validation] INFO: Object `03_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:33,581 [denest.simulation] INFO: Sessions: ['00_warmup', '01_3_spikes
↪', '02_2_spikes', '03_3_spikes']
2020-06-30 13:44:33,584 [denest.simulation] INFO: Building network.
2020-06-30 13:44:33,620 [denest.network] INFO: Build N=2 ``Model`` objects
2020-06-30 13:44:33,622 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
2020-06-30 13:44:33,623 [denest.network] INFO: Build N=3 ``Model`` objects
2020-06-30 13:44:33,624 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:44:33,625 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:44:33,630 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:44:33,631 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
2020-06-30 13:44:33,637 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
2020-06-30 13:44:33,644 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:44:33,645 [denest.network] INFO: Build N=1 projection recorders.
2020-06-30 13:44:33,646 [denest.simulation] INFO: Creating network.
2020-06-30 13:44:33,647 [denest.network] INFO: Creating neuron models...
100%|| 2/2 [00:00<00:00, 3966.24it/s]
2020-06-30 13:44:33,651 [denest.network] INFO: Creating synapse models...
100%|| 2/2 [00:00<00:00, 1872.88it/s]
2020-06-30 13:44:33,655 [denest.network] INFO: Creating recorder models...
```

(continues on next page)

(continued from previous page)

```
100%|| 3/3 [00:00<00:00, 1431.99it/s]
2020-06-30 13:44:33,660 [denest.network] INFO: Creating layers...
100%|| 2/2 [00:00<00:00, 11.02it/s]
2020-06-30 13:44:33,852 [denest.network] INFO: Creating population recorders...
100%|| 2/2 [00:00<00:00, 90.67it/s]
2020-06-30 13:44:33,883 [denest.network] INFO: Creating projection recorders...
100%|| 1/1 [00:00<00:00, 499.26it/s]
2020-06-30 13:44:33,895 [denest.network] INFO: Connecting layers...
100%|| 3/3 [00:00<00:00, 709.74it/s]
2020-06-30 13:44:33,902 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 206
Number of projections: 6650
2020-06-30 13:44:33,904 [denest.simulation] INFO: Finished creating network
2020-06-30 13:44:33,905 [denest.simulation] INFO: Saving simulation metadata...
2020-06-30 13:44:33,906 [denest.simulation] INFO: Creating output directory: data/
↪outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA':
↪ 1.0, 'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:33,906 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:33,910 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:33,913 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 1.0}/data
2020-06-30 13:44:33,914 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 1.0}/data
2020-06-30 13:44:33,920 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 1.0}/data
2020-06-30 13:44:33,922 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:34,023 [denest.simulation] INFO: Finished saving simulation metadata
2020-06-30 13:44:34,039 [denest.simulation] INFO: Running 4 sessions...
2020-06-30 13:44:34,040 [denest.simulation] INFO: Running session: '00_warmup'...
2020-06-30 13:44:34,043 [denest.session] INFO: Initializing session...
2020-06-30 13:44:34,049 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_multimeter_l1_l1_exc: {'start': 100.0}
2020-06-30 13:44:34,051 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_spike_detector_input_layer_parrot_neuron: {'start': 100.0}
2020-06-30 13:44:34,066 [denest.network.recorders] INFO:   Setting status for␣
↪recorder weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh: {'start': 100.0}
2020-06-30 13:44:34,071 [denest.session] INFO: Setting `origin` flag to `0.0` for all␣
↪stimulation devices in ``InputLayers`` for session `00_warmup`
2020-06-30 13:44:34,093 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:34,097 [denest.session] INFO: Running session '00_warmup' for 100 ms
2020-06-30 13:44:34,191 [denest.session] INFO: Finished running session
2020-06-30 13:44:34,193 [denest.session] INFO: Session '00_warmup' virtual running␣
↪time: 100 ms
2020-06-30 13:44:34,204 [denest.session] INFO: Session '00_warmup' real running time:␣
↪0h:00m:00s
2020-06-30 13:44:34,207 [denest.simulation] INFO: Done running session '00_warmup'
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:34,209 [denest.simulation] INFO: Running session: '01_3_spikes'...
2020-06-30 13:44:34,210 [denest.session] INFO: Initializing session...
2020-06-30 13:44:34,212 [denest.session] INFO: Setting `origin` flag to `100.0` for␣
↪all stimulation devices in ``InputLayers`` for session `01_3_spikes`
2020-06-30 13:44:34,217 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:34,218 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:34,276 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:34,277 [denest.session] INFO: Running session '01_3_spikes' for 100␣
↪ms
2020-06-30 13:44:34,474 [denest.session] INFO: Finished running session
2020-06-30 13:44:34,475 [denest.session] INFO: Session '01_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:34,475 [denest.session] INFO: Session '01_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:34,479 [denest.simulation] INFO: Done running session '01_3_spikes'
2020-06-30 13:44:34,489 [denest.simulation] INFO: Running session: '02_2_spikes'...
2020-06-30 13:44:34,495 [denest.session] INFO: Initializing session...
2020-06-30 13:44:34,496 [denest.session] INFO: Setting `origin` flag to `200.0` for␣
↪all stimulation devices in ``InputLayers`` for session `02_2_spikes`
2020-06-30 13:44:34,501 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:34,502 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:34,567 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:34,568 [denest.session] INFO: Running session '02_2_spikes' for 100␣
↪ms
2020-06-30 13:44:34,717 [denest.session] INFO: Finished running session
2020-06-30 13:44:34,718 [denest.session] INFO: Session '02_2_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:34,720 [denest.session] INFO: Session '02_2_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:34,720 [denest.simulation] INFO: Done running session '02_2_spikes'
2020-06-30 13:44:34,723 [denest.simulation] INFO: Running session: '03_3_spikes'...
2020-06-30 13:44:34,724 [denest.session] INFO: Initializing session...
2020-06-30 13:44:34,726 [denest.session] INFO: Setting `origin` flag to `300.0` for␣
↪all stimulation devices in ``InputLayers`` for session `03_3_spikes`
2020-06-30 13:44:34,740 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:34,744 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:34,811 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:34,811 [denest.session] INFO: Running session '03_3_spikes' for 100␣
↪ms
2020-06-30 13:44:34,966 [denest.session] INFO: Finished running session
2020-06-30 13:44:34,967 [denest.session] INFO: Session '03_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:34,968 [denest.session] INFO: Session '03_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:34,970 [denest.simulation] INFO: Done running session '03_3_spikes'
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:34,972 [denest.simulation] INFO: Finished running simulation
done


param combination: {'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 2.0}
Override trees:
 [{'network': {'neuron_models': {'my_neuron': {'nest_params': {'g_peak_AMPA': 0.2}}},
↪'projection_models': {'proj_1_AMPA': {'nest_params': {'weights': 1.0}}, 'proj_2_
↪GABAA': {'nest_params': {'weights': 2.0}}}}}, {'network': {'recorder_models': {'my_
↪multimeter': {'nest_params': {'interval': 50.0}}}}}]
2020-06-30 13:44:34,980 [denest] INFO: Loading parameter file paths from data/params/
↪tree_paths.yml
2020-06-30 13:44:34,991 [denest] INFO: Using 2 override tree(s)
2020-06-30 13:44:34,996 [denest] INFO: Finished loading parameter file paths
2020-06-30 13:44:35,002 [denest] INFO: Loading parameters files:
['./network_tree.yml',
 './simulation.yml',
 './session_models.yml',
 './kernel.yml']
2020-06-30 13:44:35,064 [denest.utils.validation] INFO: Object `simulation`: params:␣
↪using default value for optional parameters:
{'input_dir': 'input'}
2020-06-30 13:44:35,067 [denest.simulation] INFO: Initializing NEST kernel and␣
↪seeds...
2020-06-30 13:44:35,069 [denest.simulation] INFO:   Resetting NEST kernel...
2020-06-30 13:44:35,091 [denest.simulation] INFO:   Setting NEST kernel status...
data/outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_
↪AMPA': 1.0, 'weights_proj_2_GABAA': 2.0}
Running simulation for param combination: {'g_peak_AMPA_all_neurons': 0.2, 'weights_
↪proj_1_AMPA': 1.0, 'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:35,182 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'resolution': 0.5, 'overwrite_files': True})`
2020-06-30 13:44:35,191 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'data_path': "data/outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.2,
↪'weights_proj_1_AMPA': 1.0, 'weights_proj_2_GABAA': 2.0}/data", 'grng_seed': 11,
↪'rng_seeds': range(12, 13)})
2020-06-30 13:44:35,231 [denest.simulation] INFO:   Finished setting NEST kernel␣
↪status
2020-06-30 13:44:35,234 [denest.simulation] INFO:   Installing external modules...
2020-06-30 13:44:35,245 [denest.simulation] INFO:   Finished installing external␣
↪modules
2020-06-30 13:44:35,246 [denest.simulation] INFO: Finished initializing kernel
2020-06-30 13:44:35,248 [denest.simulation] INFO: Build N=3 session models
2020-06-30 13:44:35,250 [denest.simulation] INFO: Build N=4 sessions
2020-06-30 13:44:35,253 [denest.session] INFO: Creating session "00_warmup"
2020-06-30 13:44:35,255 [denest.utils.validation] INFO: Object `00_warmup`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': [], 'unit_changes': []}
2020-06-30 13:44:35,256 [denest.session] INFO: Creating session "01_3_spikes"
2020-06-30 13:44:35,259 [denest.utils.validation] INFO: Object `01_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:35,262 [denest.session] INFO: Creating session "02_2_spikes"
2020-06-30 13:44:35,263 [denest.utils.validation] INFO: Object `02_2_spikes`: params:␣
↪using default value for optional parameters:
```

(continues on next page)

(continued from previous page)

```
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:35,265 [denest.session] INFO: Creating session "03_3_spikes"
2020-06-30 13:44:35,267 [denest.utils.validation] INFO: Object `03_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:35,268 [denest.simulation] INFO: Sessions: ['00_warmup', '01_3_spikes
↪', '02_2_spikes', '03_3_spikes']
2020-06-30 13:44:35,270 [denest.simulation] INFO: Building network.
2020-06-30 13:44:35,289 [denest.network] INFO: Build N=2 ``Model`` objects
2020-06-30 13:44:35,293 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
2020-06-30 13:44:35,295 [denest.network] INFO: Build N=3 ``Model`` objects
2020-06-30 13:44:35,295 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:44:35,298 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:44:35,299 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:44:35,300 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
2020-06-30 13:44:35,303 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
2020-06-30 13:44:35,306 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:44:35,307 [denest.network] INFO: Build N=1 projection recorders.
2020-06-30 13:44:35,308 [denest.simulation] INFO: Creating network.
2020-06-30 13:44:35,309 [denest.network] INFO: Creating neuron models...
100%|| 2/2 [00:00<00:00, 2272.10it/s]
2020-06-30 13:44:35,316 [denest.network] INFO: Creating synapse models...
100%|| 2/2 [00:00<00:00, 1175.70it/s]
2020-06-30 13:44:35,321 [denest.network] INFO: Creating recorder models...
100%|| 3/3 [00:00<00:00, 1482.26it/s]
2020-06-30 13:44:35,329 [denest.network] INFO: Creating layers...
100%|| 2/2 [00:00<00:00,  7.87it/s]
2020-06-30 13:44:35,599 [denest.network] INFO: Creating population recorders...
100%|| 2/2 [00:00<00:00, 130.80it/s]
2020-06-30 13:44:35,622 [denest.network] INFO: Creating projection recorders...
100%|| 1/1 [00:00<00:00, 615.72it/s]
2020-06-30 13:44:35,633 [denest.network] INFO: Connecting layers...
100%|| 3/3 [00:00<00:00, 333.96it/s]
2020-06-30 13:44:35,648 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 206
Number of projections: 6650
2020-06-30 13:44:35,650 [denest.simulation] INFO: Finished creating network
2020-06-30 13:44:35,651 [denest.simulation] INFO: Saving simulation metadata...
2020-06-30 13:44:35,652 [denest.simulation] INFO: Creating output directory: data/
↪outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA':
↪ 1.0, 'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:35,654 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:35,657 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:35,659 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 2.0}/data
2020-06-30 13:44:35,661 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 2.0}/data
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:35,662 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 2.0}/data
2020-06-30 13:44:35,665 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 1.0,
↪'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:35,728 [denest.simulation] INFO: Finished saving simulation metadata
2020-06-30 13:44:35,730 [denest.simulation] INFO: Running 4 sessions...
2020-06-30 13:44:35,734 [denest.simulation] INFO: Running session: '00_warmup'...
2020-06-30 13:44:35,753 [denest.session] INFO: Initializing session...
2020-06-30 13:44:35,759 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_multimeter_l1_l1_exc: {'start': 100.0}
2020-06-30 13:44:35,767 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_spike_detector_input_layer_parrot_neuron: {'start': 100.0}
2020-06-30 13:44:35,778 [denest.network.recorders] INFO:   Setting status for␣
↪recorder weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh: {'start': 100.0}
2020-06-30 13:44:35,781 [denest.session] INFO: Setting `origin` flag to `0.0` for all␣
↪stimulation devices in ``InputLayers`` for session `00_warmup`
2020-06-30 13:44:35,789 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:35,791 [denest.session] INFO: Running session '00_warmup' for 100 ms
2020-06-30 13:44:35,901 [denest.session] INFO: Finished running session
2020-06-30 13:44:35,903 [denest.session] INFO: Session '00_warmup' virtual running␣
↪time: 100 ms
2020-06-30 13:44:35,903 [denest.session] INFO: Session '00_warmup' real running time:␣
↪0h:00m:00s
2020-06-30 13:44:35,905 [denest.simulation] INFO: Done running session '00_warmup'
2020-06-30 13:44:35,906 [denest.simulation] INFO: Running session: '01_3_spikes'...
2020-06-30 13:44:35,907 [denest.session] INFO: Initializing session...
2020-06-30 13:44:35,908 [denest.session] INFO: Setting `origin` flag to `100.0` for␣
↪all stimulation devices in ``InputLayers`` for session `01_3_spikes`
2020-06-30 13:44:35,928 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:35,932 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:35,999 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:36,000 [denest.session] INFO: Running session '01_3_spikes' for 100␣
↪ms
2020-06-30 13:44:36,176 [denest.session] INFO: Finished running session
2020-06-30 13:44:36,177 [denest.session] INFO: Session '01_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:36,177 [denest.session] INFO: Session '01_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:36,181 [denest.simulation] INFO: Done running session '01_3_spikes'
2020-06-30 13:44:36,183 [denest.simulation] INFO: Running session: '02_2_spikes'...
2020-06-30 13:44:36,184 [denest.session] INFO: Initializing session...
2020-06-30 13:44:36,185 [denest.session] INFO: Setting `origin` flag to `200.0` for␣
↪all stimulation devices in ``InputLayers`` for session `02_2_spikes`
2020-06-30 13:44:36,193 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:36,194 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:36,265 [denest.session] INFO: Finished initializing session
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:36,265 [denest.session] INFO: Running session '02_2_spikes' for 100␣
↪ms
2020-06-30 13:44:36,393 [denest.session] INFO: Finished running session
2020-06-30 13:44:36,394 [denest.session] INFO: Session '02_2_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:36,395 [denest.session] INFO: Session '02_2_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:36,396 [denest.simulation] INFO: Done running session '02_2_spikes'
2020-06-30 13:44:36,397 [denest.simulation] INFO: Running session: '03_3_spikes'...
2020-06-30 13:44:36,398 [denest.session] INFO: Initializing session...
2020-06-30 13:44:36,399 [denest.session] INFO: Setting `origin` flag to `300.0` for␣
↪all stimulation devices in ``InputLayers`` for session `03_3_spikes`
2020-06-30 13:44:36,405 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:36,406 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:36,472 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:36,473 [denest.session] INFO: Running session '03_3_spikes' for 100␣
↪ms
2020-06-30 13:44:36,574 [denest.session] INFO: Finished running session
2020-06-30 13:44:36,575 [denest.session] INFO: Session '03_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:36,576 [denest.session] INFO: Session '03_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:36,577 [denest.simulation] INFO: Done running session '03_3_spikes'
2020-06-30 13:44:36,579 [denest.simulation] INFO: Finished running simulation
done


param combination: {'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 1.0}
Override trees:
 [{'network': {'neuron_models': {'my_neuron': {'nest_params': {'g_peak_AMPA': 0.2}}},
↪'projection_models': {'proj_1_AMPA': {'nest_params': {'weights': 2.0}}, 'proj_2_
↪GABAA': {'nest_params': {'weights': 1.0}}}}}, {'network': {'recorder_models': {'my_
↪multimeter': {'nest_params': {'interval': 50.0}}}}}]
2020-06-30 13:44:36,590 [denest] INFO: Loading parameter file paths from data/params/
↪tree_paths.yml
2020-06-30 13:44:36,591 [denest] INFO: Using 2 override tree(s)
2020-06-30 13:44:36,594 [denest] INFO: Finished loading parameter file paths
2020-06-30 13:44:36,596 [denest] INFO: Loading parameters files:
['./network_tree.yml',
 './simulation.yml',
 './session_models.yml',
 './kernel.yml']
2020-06-30 13:44:36,664 [denest.utils.validation] INFO: Object `simulation`: params:␣
↪using default value for optional parameters:
{'input_dir': 'input'}
2020-06-30 13:44:36,665 [denest.simulation] INFO: Initializing NEST kernel and␣
↪seeds...
2020-06-30 13:44:36,666 [denest.simulation] INFO:   Resetting NEST kernel...
2020-06-30 13:44:36,674 [denest.simulation] INFO:   Setting NEST kernel status...
2020-06-30 13:44:36,675 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'resolution': 0.5, 'overwrite_files': True})`
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:36,680 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'data_path': "data/outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.2,
↪'weights_proj_1_AMPA': 2.0, 'weights_proj_2_GABAA': 1.0}/data", 'grng_seed': 11,
↪'rng_seeds': range(12, 13)})
2020-06-30 13:44:36,709 [denest.simulation] INFO:   Finished setting NEST kernel␣
↪status
2020-06-30 13:44:36,727 [denest.simulation] INFO:   Installing external modules...
2020-06-30 13:44:36,740 [denest.simulation] INFO:   Finished installing external␣
↪modules
2020-06-30 13:44:36,745 [denest.simulation] INFO: Finished initializing kernel
2020-06-30 13:44:36,749 [denest.simulation] INFO: Build N=3 session models
2020-06-30 13:44:36,751 [denest.simulation] INFO: Build N=4 sessions
2020-06-30 13:44:36,754 [denest.session] INFO: Creating session "00_warmup"
2020-06-30 13:44:36,757 [denest.utils.validation] INFO: Object `00_warmup`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': [], 'unit_changes': []}
2020-06-30 13:44:36,763 [denest.session] INFO: Creating session "01_3_spikes"
2020-06-30 13:44:36,765 [denest.utils.validation] INFO: Object `01_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:36,768 [denest.session] INFO: Creating session "02_2_spikes"
2020-06-30 13:44:36,770 [denest.utils.validation] INFO: Object `02_2_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:36,772 [denest.session] INFO: Creating session "03_3_spikes"
2020-06-30 13:44:36,773 [denest.utils.validation] INFO: Object `03_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:36,774 [denest.simulation] INFO: Sessions: ['00_warmup', '01_3_spikes
↪', '02_2_spikes', '03_3_spikes']
```

```
data/outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_
↪AMPA': 2.0, 'weights_proj_2_GABAA': 1.0}
Running simulation for param combination: {'g_peak_AMPA_all_neurons': 0.2, 'weights_
↪proj_1_AMPA': 2.0, 'weights_proj_2_GABAA': 1.0}
```

```
2020-06-30 13:44:36,788 [denest.simulation] INFO: Building network.
2020-06-30 13:44:36,809 [denest.network] INFO: Build N=2 ``Model`` objects
2020-06-30 13:44:36,812 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
2020-06-30 13:44:36,814 [denest.network] INFO: Build N=3 ``Model`` objects
2020-06-30 13:44:36,815 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:44:36,817 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:44:36,824 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:44:36,827 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
2020-06-30 13:44:36,835 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
2020-06-30 13:44:36,846 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:44:36,852 [denest.network] INFO: Build N=1 projection recorders.
2020-06-30 13:44:36,869 [denest.simulation] INFO: Creating network.
2020-06-30 13:44:36,870 [denest.network] INFO: Creating neuron models...
100%|| 2/2 [00:00<00:00, 2306.46it/s]
2020-06-30 13:44:36,875 [denest.network] INFO: Creating synapse models...
100%|| 2/2 [00:00<00:00, 389.19it/s]
2020-06-30 13:44:36,888 [denest.network] INFO: Creating recorder models...
```

(continues on next page)

(continued from previous page)

```
100%|| 3/3 [00:00<00:00, 1564.26it/s]
2020-06-30 13:44:36,896 [denest.network] INFO: Creating layers...
100%|| 2/2 [00:00<00:00,  5.16it/s]
2020-06-30 13:44:37,292 [denest.network] INFO: Creating population recorders...
100%|| 2/2 [00:00<00:00, 90.53it/s]
2020-06-30 13:44:37,319 [denest.network] INFO: Creating projection recorders...
100%|| 1/1 [00:00<00:00, 437.09it/s]
2020-06-30 13:44:37,327 [denest.network] INFO: Connecting layers...
100%|| 3/3 [00:00<00:00, 440.01it/s]
2020-06-30 13:44:37,343 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 206
Number of projections: 6650
2020-06-30 13:44:37,345 [denest.simulation] INFO: Finished creating network
2020-06-30 13:44:37,346 [denest.simulation] INFO: Saving simulation metadata...
2020-06-30 13:44:37,347 [denest.simulation] INFO: Creating output directory: data/
↪outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA':
↪ 2.0, 'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:37,350 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:37,355 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:37,359 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 1.0}/data
2020-06-30 13:44:37,362 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 1.0}/data
2020-06-30 13:44:37,365 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 1.0}/data
2020-06-30 13:44:37,368 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 1.0}
2020-06-30 13:44:37,436 [denest.simulation] INFO: Finished saving simulation metadata
2020-06-30 13:44:37,438 [denest.simulation] INFO: Running 4 sessions...
2020-06-30 13:44:37,441 [denest.simulation] INFO: Running session: '00_warmup'...
2020-06-30 13:44:37,442 [denest.session] INFO: Initializing session...
2020-06-30 13:44:37,454 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_multimeter_l1_l1_exc: {'start': 100.0}
2020-06-30 13:44:37,457 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_spike_detector_input_layer_parrot_neuron: {'start': 100.0}
2020-06-30 13:44:37,460 [denest.network.recorders] INFO:   Setting status for␣
↪recorder weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh: {'start': 100.0}
2020-06-30 13:44:37,468 [denest.session] INFO: Setting `origin` flag to `0.0` for all␣
↪stimulation devices in ``InputLayers`` for session `00_warmup`
2020-06-30 13:44:37,485 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:37,490 [denest.session] INFO: Running session '00_warmup' for 100 ms
2020-06-30 13:44:37,715 [denest.session] INFO: Finished running session
2020-06-30 13:44:37,715 [denest.session] INFO: Session '00_warmup' virtual running␣
↪time: 100 ms
2020-06-30 13:44:37,717 [denest.session] INFO: Session '00_warmup' real running time:␣
↪0h:00m:00s
2020-06-30 13:44:37,719 [denest.simulation] INFO: Done running session '00_warmup'
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:37,721 [denest.simulation] INFO: Running session: '01_3_spikes'...
2020-06-30 13:44:37,723 [denest.session] INFO: Initializing session...
2020-06-30 13:44:37,725 [denest.session] INFO: Setting `origin` flag to `100.0` for␣
↪all stimulation devices in ``InputLayers`` for session `01_3_spikes`
2020-06-30 13:44:37,746 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:37,747 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:37,817 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:37,819 [denest.session] INFO: Running session '01_3_spikes' for 100␣
↪ms
2020-06-30 13:44:38,154 [denest.session] INFO: Finished running session
2020-06-30 13:44:38,156 [denest.session] INFO: Session '01_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:38,157 [denest.session] INFO: Session '01_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:38,160 [denest.simulation] INFO: Done running session '01_3_spikes'
2020-06-30 13:44:38,163 [denest.simulation] INFO: Running session: '02_2_spikes'...
2020-06-30 13:44:38,172 [denest.session] INFO: Initializing session...
2020-06-30 13:44:38,176 [denest.session] INFO: Setting `origin` flag to `200.0` for␣
↪all stimulation devices in ``InputLayers`` for session `02_2_spikes`
2020-06-30 13:44:38,184 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:38,186 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:38,262 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:38,263 [denest.session] INFO: Running session '02_2_spikes' for 100␣
↪ms
2020-06-30 13:44:38,482 [denest.session] INFO: Finished running session
2020-06-30 13:44:38,483 [denest.session] INFO: Session '02_2_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:38,485 [denest.session] INFO: Session '02_2_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:38,488 [denest.simulation] INFO: Done running session '02_2_spikes'
2020-06-30 13:44:38,490 [denest.simulation] INFO: Running session: '03_3_spikes'...
2020-06-30 13:44:38,497 [denest.session] INFO: Initializing session...
2020-06-30 13:44:38,502 [denest.session] INFO: Setting `origin` flag to `300.0` for␣
↪all stimulation devices in ``InputLayers`` for session `03_3_spikes`
2020-06-30 13:44:38,511 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:38,512 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:38,580 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:38,581 [denest.session] INFO: Running session '03_3_spikes' for 100␣
↪ms
2020-06-30 13:44:38,820 [denest.session] INFO: Finished running session
2020-06-30 13:44:38,821 [denest.session] INFO: Session '03_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:38,822 [denest.session] INFO: Session '03_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:38,823 [denest.simulation] INFO: Done running session '03_3_spikes'
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:38,824 [denest.simulation] INFO: Finished running simulation
done


param combination: {'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 2.0}
Override trees:
 [{'network': {'neuron_models': {'my_neuron': {'nest_params': {'g_peak_AMPA': 0.2}}},
↪'projection_models': {'proj_1_AMPA': {'nest_params': {'weights': 2.0}}, 'proj_2_
↪GABAA': {'nest_params': {'weights': 2.0}}}}}, {'network': {'recorder_models': {'my_
↪multimeter': {'nest_params': {'interval': 50.0}}}}}]
2020-06-30 13:44:38,835 [denest] INFO: Loading parameter file paths from data/params/
↪tree_paths.yml
2020-06-30 13:44:38,843 [denest] INFO: Using 2 override tree(s)
2020-06-30 13:44:38,846 [denest] INFO: Finished loading parameter file paths
2020-06-30 13:44:38,847 [denest] INFO: Loading parameters files:
['./network_tree.yml',
 './simulation.yml',
 './session_models.yml',
 './kernel.yml']
2020-06-30 13:44:38,903 [denest.utils.validation] INFO: Object `simulation`: params:␣
↪using default value for optional parameters:
{'input_dir': 'input'}
2020-06-30 13:44:38,904 [denest.simulation] INFO: Initializing NEST kernel and␣
↪seeds...
2020-06-30 13:44:38,904 [denest.simulation] INFO:   Resetting NEST kernel...
2020-06-30 13:44:38,914 [denest.simulation] INFO:   Setting NEST kernel status...
2020-06-30 13:44:38,916 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'resolution': 0.5, 'overwrite_files': True})`
2020-06-30 13:44:38,919 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'data_path': "data/outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.2,
↪'weights_proj_1_AMPA': 2.0, 'weights_proj_2_GABAA': 2.0}/data", 'grng_seed': 11,
↪'rng_seeds': range(12, 13)})
2020-06-30 13:44:38,924 [denest.simulation] INFO:   Finished setting NEST kernel␣
↪status
2020-06-30 13:44:38,925 [denest.simulation] INFO:   Installing external modules...
2020-06-30 13:44:38,926 [denest.simulation] INFO:   Finished installing external␣
↪modules
2020-06-30 13:44:38,927 [denest.simulation] INFO: Finished initializing kernel
2020-06-30 13:44:38,941 [denest.simulation] INFO: Build N=3 session models
2020-06-30 13:44:38,943 [denest.simulation] INFO: Build N=4 sessions
2020-06-30 13:44:38,945 [denest.session] INFO: Creating session "00_warmup"
2020-06-30 13:44:38,947 [denest.utils.validation] INFO: Object `00_warmup`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': [], 'unit_changes': []}
2020-06-30 13:44:38,965 [denest.session] INFO: Creating session "01_3_spikes"
2020-06-30 13:44:38,967 [denest.utils.validation] INFO: Object `01_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:38,972 [denest.session] INFO: Creating session "02_2_spikes"
2020-06-30 13:44:38,976 [denest.utils.validation] INFO: Object `02_2_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:38,979 [denest.session] INFO: Creating session "03_3_spikes"
2020-06-30 13:44:38,981 [denest.utils.validation] INFO: Object `03_3_spikes`: params:␣
↪using default value for optional parameters:
```

(continues on next page)

(continued from previous page)

```
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:44:38,986 [denest.simulation] INFO: Sessions: ['00_warmup', '01_3_spikes
↪', '02_2_spikes', '03_3_spikes']
2020-06-30 13:44:38,989 [denest.simulation] INFO: Building network.
2020-06-30 13:44:39,007 [denest.network] INFO: Build N=2 ``Model`` objects
2020-06-30 13:44:39,009 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
2020-06-30 13:44:39,010 [denest.network] INFO: Build N=3 ``Model`` objects
2020-06-30 13:44:39,012 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:44:39,013 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:44:39,015 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:44:39,016 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
2020-06-30 13:44:39,020 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
2020-06-30 13:44:39,024 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:44:39,025 [denest.network] INFO: Build N=1 projection recorders.
2020-06-30 13:44:39,025 [denest.simulation] INFO: Creating network.
```

```
data/outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_
↪AMPA': 2.0, 'weights_proj_2_GABAA': 2.0}
Running simulation for param combination: {'g_peak_AMPA_all_neurons': 0.2, 'weights_
↪proj_1_AMPA': 2.0, 'weights_proj_2_GABAA': 2.0}
```

```
2020-06-30 13:44:39,026 [denest.network] INFO: Creating neuron models...
100%|| 2/2 [00:00<00:00, 2840.71it/s]
2020-06-30 13:44:39,032 [denest.network] INFO: Creating synapse models...
100%|| 2/2 [00:00<00:00, 1960.41it/s]
2020-06-30 13:44:39,043 [denest.network] INFO: Creating recorder models...
100%|| 3/3 [00:00<00:00, 2373.24it/s]
2020-06-30 13:44:39,049 [denest.network] INFO: Creating layers...
100%|| 2/2 [00:00<00:00,  7.59it/s]
2020-06-30 13:44:39,327 [denest.network] INFO: Creating population recorders...
100%|| 2/2 [00:00<00:00, 86.03it/s]
2020-06-30 13:44:39,354 [denest.network] INFO: Creating projection recorders...
100%|| 1/1 [00:00<00:00, 423.54it/s]
2020-06-30 13:44:39,360 [denest.network] INFO: Connecting layers...
100%|| 3/3 [00:00<00:00, 342.90it/s]
2020-06-30 13:44:39,372 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 206
Number of projections: 6650
2020-06-30 13:44:39,373 [denest.simulation] INFO: Finished creating network
2020-06-30 13:44:39,374 [denest.simulation] INFO: Saving simulation metadata...
2020-06-30 13:44:39,375 [denest.simulation] INFO: Creating output directory: data/
↪outputs/output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA':
↪ 2.0, 'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:39,377 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:39,380 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:39,385 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 2.0}/data
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:39,388 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 2.0}/data
2020-06-30 13:44:39,392 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 2.0}/data
2020-06-30 13:44:39,394 [denest.io.save] INFO: Clearing directory: data/outputs/
↪output_param_explore/{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 2.0,
↪'weights_proj_2_GABAA': 2.0}
2020-06-30 13:44:39,442 [denest.simulation] INFO: Finished saving simulation metadata
2020-06-30 13:44:39,443 [denest.simulation] INFO: Running 4 sessions...
2020-06-30 13:44:39,446 [denest.simulation] INFO: Running session: '00_warmup'...
2020-06-30 13:44:39,449 [denest.session] INFO: Initializing session...
2020-06-30 13:44:39,452 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_multimeter_l1_l1_exc: {'start': 100.0}
2020-06-30 13:44:39,454 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_spike_detector_input_layer_parrot_neuron: {'start': 100.0}
2020-06-30 13:44:39,455 [denest.network.recorders] INFO:   Setting status for␣
↪recorder weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh: {'start': 100.0}
2020-06-30 13:44:39,457 [denest.session] INFO: Setting `origin` flag to `0.0` for all␣
↪stimulation devices in ``InputLayers`` for session `00_warmup`
2020-06-30 13:44:39,462 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:39,463 [denest.session] INFO: Running session '00_warmup' for 100 ms
2020-06-30 13:44:39,640 [denest.session] INFO: Finished running session
2020-06-30 13:44:39,664 [denest.session] INFO: Session '00_warmup' virtual running␣
↪time: 100 ms
2020-06-30 13:44:39,672 [denest.session] INFO: Session '00_warmup' real running time:␣
↪0h:00m:00s
2020-06-30 13:44:39,674 [denest.simulation] INFO: Done running session '00_warmup'
2020-06-30 13:44:39,676 [denest.simulation] INFO: Running session: '01_3_spikes'...
2020-06-30 13:44:39,678 [denest.session] INFO: Initializing session...
2020-06-30 13:44:39,680 [denest.session] INFO: Setting `origin` flag to `100.0` for␣
↪all stimulation devices in ``InputLayers`` for session `01_3_spikes`
2020-06-30 13:44:39,688 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:39,689 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:39,757 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:39,760 [denest.session] INFO: Running session '01_3_spikes' for 100␣
↪ms
2020-06-30 13:44:39,985 [denest.session] INFO: Finished running session
2020-06-30 13:44:39,986 [denest.session] INFO: Session '01_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:39,986 [denest.session] INFO: Session '01_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:39,988 [denest.simulation] INFO: Done running session '01_3_spikes'
2020-06-30 13:44:39,990 [denest.simulation] INFO: Running session: '02_2_spikes'...
2020-06-30 13:44:39,991 [denest.session] INFO: Initializing session...
2020-06-30 13:44:39,994 [denest.session] INFO: Setting `origin` flag to `200.0` for␣
↪all stimulation devices in ``InputLayers`` for session `02_2_spikes`
2020-06-30 13:44:40,011 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:40,012 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:44:40,081 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:40,082 [denest.session] INFO: Running session '02_2_spikes' for 100␣
↪ms
2020-06-30 13:44:40,240 [denest.session] INFO: Finished running session
2020-06-30 13:44:40,241 [denest.session] INFO: Session '02_2_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:40,242 [denest.session] INFO: Session '02_2_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:40,243 [denest.simulation] INFO: Done running session '02_2_spikes'
2020-06-30 13:44:40,245 [denest.simulation] INFO: Running session: '03_3_spikes'...
2020-06-30 13:44:40,250 [denest.session] INFO: Initializing session...
2020-06-30 13:44:40,254 [denest.session] INFO: Setting `origin` flag to `300.0` for␣
↪all stimulation devices in ``InputLayers`` for session `03_3_spikes`
2020-06-30 13:44:40,262 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:44:40,263 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:44:40,324 [denest.session] INFO: Finished initializing session

2020-06-30 13:44:40,325 [denest.session] INFO: Running session '03_3_spikes' for 100␣
↪ms
2020-06-30 13:44:40,424 [denest.session] INFO: Finished running session
2020-06-30 13:44:40,425 [denest.session] INFO: Session '03_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:44:40,426 [denest.session] INFO: Session '03_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:44:40,430 [denest.simulation] INFO: Done running session '03_3_spikes'
2020-06-30 13:44:40,439 [denest.simulation] INFO: Finished running simulation
```

```
done
```

[14]:
```
!ls {MAIN_OUTPUT_DIR}
```

```
{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 1.0, 'weights_proj_2_GABAA':␣
↪1.0}
{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 1.0, 'weights_proj_2_GABAA':␣
↪2.0}
{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 2.0, 'weights_proj_2_GABAA':␣
↪1.0}
{'g_peak_AMPA_all_neurons': 0.1, 'weights_proj_1_AMPA': 2.0, 'weights_proj_2_GABAA':␣
↪2.0}
{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 1.0, 'weights_proj_2_GABAA':␣
↪1.0}
{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 1.0, 'weights_proj_2_GABAA':␣
↪2.0}
{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 2.0, 'weights_proj_2_GABAA':␣
↪1.0}
{'g_peak_AMPA_all_neurons': 0.2, 'weights_proj_1_AMPA': 2.0, 'weights_proj_2_GABAA':␣
↪2.0}
```

5.7 Version control the parameter files

Here we learn a nice little trick to version-control the parameter files while tracking the simulation output directories for all the versions of the network.

[1]:
```
import denest
import nest
import yaml
from pathlib import Path
from pprint import pprint
```

[2]:
```
PARAMS_DIR = Path('./data/params')  # Where all the parameter files are
DATA_DIR = Path('./data/outputs')  # Where all the simulation output directories are
```

5.7.1 1. Turn `PARAMS_DIR` into a git directory

5.7.2 2. Make modifications to the parameters

5.7.3 3. Run a simulation with the modified parameter files and track the output directory

A. Generate a unique output directory name

For example, using the current date and time:

[3]:
```
from datetime import datetime

output_dir_name = datetime.now().strftime("%Y-%m-%d_%H:%M:%S")

output_dir = Path(DATA_DIR)/output_dir_name
output_dir
```

[3]:
```
PosixPath('data/outputs/2020-06-30_13:47:32')
```

B. Create a symlink to the output directory in the version controlled `PARAMS_DIR`

[4]:
```
def overwrite_dir_symlink(target, link_path):
    """Overwrite or create a symlink to a directory.

    ``target`` should be relative to ``link_path``"""
    import os, errno

    try:
        os.symlink(target, link_path)
    except OSError as e:
        if e.errno == errno.EEXIST:
            os.remove(link_path)
            os.symlink(target, link_path)
        else:
            raise e
```

[5]:
```
# Needs path RELATIVE to symlink position
overwrite_dir_symlink(Path('../outputs')/output_dir_name, PARAMS_DIR/'output')
```

C. Run the simulation

[6]:
```
denest.run(PARAMS_DIR/'tree_paths.yml', output_dir=output_dir)
```

```
2020-06-30 13:47:32,953 [denest] INFO:

=== RUNNING SIMULATION ========================================================

2020-06-30 13:47:32,956 [denest] INFO: Loading parameter file paths from data/params/
↪tree_paths.yml
2020-06-30 13:47:32,984 [denest] INFO: Finished loading parameter file paths
2020-06-30 13:47:32,989 [denest] INFO: Loading parameters files:
['./network_tree.yml',
 './simulation.yml',
 './session_models.yml',
 './kernel.yml']
2020-06-30 13:47:33,080 [denest] INFO: Initializing simulation...
2020-06-30 13:47:33,113 [denest.utils.validation] INFO: Object `simulation`: params:␣
↪using default value for optional parameters:
{'input_dir': 'input'}
2020-06-30 13:47:33,115 [denest.simulation] INFO: Initializing NEST kernel and␣
↪seeds...
2020-06-30 13:47:33,116 [denest.simulation] INFO:   Resetting NEST kernel...
2020-06-30 13:47:33,279 [denest.simulation] INFO:   Setting NEST kernel status...
2020-06-30 13:47:33,299 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'resolution': 0.5, 'overwrite_files': True})`
2020-06-30 13:47:33,302 [denest.simulation] INFO:     Calling `nest.SetKernelStatus({
↪'data_path': 'data/outputs/2020-06-30_13:47:32/data', 'grng_seed': 11, 'rng_seeds':␣
↪range(12, 13)})
2020-06-30 13:47:33,305 [denest.simulation] INFO:   Finished setting NEST kernel␣
↪status
2020-06-30 13:47:33,309 [denest.simulation] INFO:   Installing external modules...
2020-06-30 13:47:33,342 [denest.simulation] INFO:   Finished installing external␣
↪modules
2020-06-30 13:47:33,343 [denest.simulation] INFO: Finished initializing kernel
2020-06-30 13:47:33,344 [denest.simulation] INFO: Build N=3 session models
2020-06-30 13:47:33,349 [denest.simulation] INFO: Build N=4 sessions
2020-06-30 13:47:33,361 [denest.session] INFO: Creating session "00_warmup"
2020-06-30 13:47:33,365 [denest.utils.validation] INFO: Object `00_warmup`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': [], 'unit_changes': []}
2020-06-30 13:47:33,370 [denest.session] INFO: Creating session "01_3_spikes"
2020-06-30 13:47:33,375 [denest.utils.validation] INFO: Object `01_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:47:33,379 [denest.session] INFO: Creating session "02_2_spikes"
2020-06-30 13:47:33,381 [denest.utils.validation] INFO: Object `02_2_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
2020-06-30 13:47:33,388 [denest.session] INFO: Creating session "03_3_spikes"
2020-06-30 13:47:33,394 [denest.utils.validation] INFO: Object `03_3_spikes`: params:␣
↪using default value for optional parameters:
{'reset_network': False, 'synapse_changes': []}
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:47:33,401 [denest.simulation] INFO: Sessions: ['00_warmup', '01_3_spikes
↪', '02_2_spikes', '03_3_spikes']
2020-06-30 13:47:33,406 [denest.simulation] INFO: Building network.
2020-06-30 13:47:33,427 [denest.network] INFO: Build N=2 ``Model`` objects
2020-06-30 13:47:33,433 [denest.network] INFO: Build N=2 ``SynapseModel`` objects
2020-06-30 13:47:33,436 [denest.network] INFO: Build N=3 ``Model`` objects
2020-06-30 13:47:33,442 [denest.network] INFO: Build N=2 ``Layer`` or ``InputLayer``␣
↪objects.
2020-06-30 13:47:33,444 [denest.utils.validation] INFO: Object `proj_1_AMPA`: params:␣
↪using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:47:33,448 [denest.utils.validation] INFO: Object `proj_2_GABAA`: params:
↪ using default value for optional parameters:
{'type': 'topological'}
2020-06-30 13:47:33,453 [denest.network] INFO: Build N=2 ``ProjectionModel`` objects
2020-06-30 13:47:33,459 [denest.network] INFO: Build N=3 ``TopoProjection`` objects
2020-06-30 13:47:33,464 [denest.network] INFO: Build N=2 population recorders.
2020-06-30 13:47:33,467 [denest.network] INFO: Build N=1 projection recorders.
2020-06-30 13:47:33,469 [denest.simulation] INFO: Creating network.
2020-06-30 13:47:33,475 [denest.network] INFO: Creating neuron models...
100%|| 2/2 [00:00<00:00, 496.90it/s]
2020-06-30 13:47:33,512 [denest.network] INFO: Creating synapse models...
100%|| 2/2 [00:00<00:00, 678.36it/s]
2020-06-30 13:47:33,522 [denest.network] INFO: Creating recorder models...
100%|| 3/3 [00:00<00:00, 1294.67it/s]
2020-06-30 13:47:33,547 [denest.network] INFO: Creating layers...
  0%|          | 0/2 [00:00<?, ?it/s]/Users/tom/nest/nest-simulator-2.20.0/lib/
↪python3.7/site-packages/nest/lib/hl_api_helper.py:127: UserWarning:
GetNodes is deprecated and will be removed in NEST 3.0. Use             GIDCollection␣
↪instead.
100%|| 2/2 [00:00<00:00,  7.59it/s]
2020-06-30 13:47:33,823 [denest.network] INFO: Creating population recorders...
100%|| 2/2 [00:00<00:00, 57.99it/s]
2020-06-30 13:47:33,870 [denest.network] INFO: Creating projection recorders...
100%|| 1/1 [00:00<00:00, 90.88it/s]
2020-06-30 13:47:33,910 [denest.network] INFO: Connecting layers...
100%|| 3/3 [00:00<00:00, 62.17it/s]
2020-06-30 13:47:34,027 [denest.network] INFO: Network size (including recorders and␣
↪parrot neurons):
Number of nodes: 206
Number of projections: 6650
2020-06-30 13:47:34,087 [denest.simulation] INFO: Finished creating network
2020-06-30 13:47:34,094 [denest.simulation] INFO: Saving simulation metadata...
2020-06-30 13:47:34,106 [denest.simulation] INFO: Creating output directory: data/
↪outputs/2020-06-30_13:47:32
2020-06-30 13:47:34,145 [denest.io.save] INFO: Clearing directory: data/outputs/2020-
↪06-30_13:47:32
2020-06-30 13:47:34,161 [denest.io.save] INFO: Clearing directory: data/outputs/2020-
↪06-30_13:47:32
2020-06-30 13:47:34,177 [denest.io.save] INFO: Clearing directory: data/outputs/2020-
↪06-30_13:47:32/data
2020-06-30 13:47:34,189 [denest.io.save] INFO: Clearing directory: data/outputs/2020-
↪06-30_13:47:32/data
2020-06-30 13:47:34,193 [denest.io.save] INFO: Clearing directory: data/outputs/2020-
↪06-30_13:47:32/data
2020-06-30 13:47:34,195 [denest.io.save] INFO: Clearing directory: data/outputs/2020-
↪06-30_13:47:32
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:47:34,311 [denest.simulation] INFO: Finished saving simulation metadata
2020-06-30 13:47:34,315 [denest] INFO: Finished initializing simulation
2020-06-30 13:47:34,375 [denest] INFO: Running simulation...
2020-06-30 13:47:34,377 [denest.simulation] INFO: Running 4 sessions...
2020-06-30 13:47:34,379 [denest.simulation] INFO: Running session: '00_warmup'...
2020-06-30 13:47:34,383 [denest.session] INFO: Initializing session...
2020-06-30 13:47:34,396 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_multimeter_l1_l1_exc: {'start': 100.0}
2020-06-30 13:47:34,401 [denest.network.recorders] INFO:   Setting status for␣
↪recorder my_spike_detector_input_layer_parrot_neuron: {'start': 100.0}
2020-06-30 13:47:34,408 [denest.network.recorders] INFO:   Setting status for␣
↪recorder weight_recorder_proj_1_AMPA-l1-l1_exc-l1-l1_inh: {'start': 100.0}
2020-06-30 13:47:34,414 [denest.session] INFO: Setting `origin` flag to `0.0` for all␣
↪stimulation devices in ``InputLayers`` for session `00_warmup`
2020-06-30 13:47:34,418 [denest.session] INFO: Finished initializing session

2020-06-30 13:47:34,421 [denest.session] INFO: Running session '00_warmup' for 100 ms
2020-06-30 13:47:34,627 [denest.session] INFO: Finished running session
2020-06-30 13:47:34,630 [denest.session] INFO: Session '00_warmup' virtual running␣
↪time: 100 ms
2020-06-30 13:47:34,727 [denest.session] INFO: Session '00_warmup' real running time:␣
↪0h:00m:00s
2020-06-30 13:47:34,732 [denest.simulation] INFO: Done running session '00_warmup'
2020-06-30 13:47:34,734 [denest.simulation] INFO: Running session: '01_3_spikes'...
2020-06-30 13:47:34,756 [denest.session] INFO: Initializing session...
2020-06-30 13:47:34,759 [denest.session] INFO: Setting `origin` flag to `100.0` for␣
↪all stimulation devices in ``InputLayers`` for session `01_3_spikes`
2020-06-30 13:47:34,767 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:47:34,769 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:47:34,902 [denest.session] INFO: Finished initializing session

2020-06-30 13:47:34,904 [denest.session] INFO: Running session '01_3_spikes' for 100␣
↪ms
2020-06-30 13:47:35,115 [denest.session] INFO: Finished running session
2020-06-30 13:47:35,116 [denest.session] INFO: Session '01_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:47:35,117 [denest.session] INFO: Session '01_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:47:35,118 [denest.simulation] INFO: Done running session '01_3_spikes'
2020-06-30 13:47:35,120 [denest.simulation] INFO: Running session: '02_2_spikes'...
2020-06-30 13:47:35,125 [denest.session] INFO: Initializing session...
2020-06-30 13:47:35,126 [denest.session] INFO: Setting `origin` flag to `200.0` for␣
↪all stimulation devices in ``InputLayers`` for session `02_2_spikes`
2020-06-30 13:47:35,159 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:47:35,161 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:47:35,251 [denest.session] INFO: Finished initializing session

2020-06-30 13:47:35,252 [denest.session] INFO: Running session '02_2_spikes' for 100␣
↪ms
2020-06-30 13:47:35,469 [denest.session] INFO: Finished running session
2020-06-30 13:47:35,470 [denest.session] INFO: Session '02_2_spikes' virtual running␣
↪time: 100 ms
```

(continues on next page)

(continued from previous page)

```
2020-06-30 13:47:35,471 [denest.session] INFO: Session '02_2_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:47:35,474 [denest.simulation] INFO: Done running session '02_2_spikes'
2020-06-30 13:47:35,477 [denest.simulation] INFO: Running session: '03_3_spikes'...
2020-06-30 13:47:35,506 [denest.session] INFO: Initializing session...
2020-06-30 13:47:35,511 [denest.session] INFO: Setting `origin` flag to `300.0` for␣
↪all stimulation devices in ``InputLayers`` for session `03_3_spikes`
2020-06-30 13:47:35,525 [denest.utils.validation] INFO: Object `Unit changes␣
↪dictionary`: params: using default value for optional parameters:
{'change_type': 'constant', 'from_array': False}
2020-06-30 13:47:35,526 [denest.network.layers] INFO: Layer='input_layer', pop='spike_
↪generator': Applying 'constant' change, param='spike_times', from single value')
2020-06-30 13:47:35,606 [denest.session] INFO: Finished initializing session

2020-06-30 13:47:35,607 [denest.session] INFO: Running session '03_3_spikes' for 100␣
↪ms
2020-06-30 13:47:35,716 [denest.session] INFO: Finished running session
2020-06-30 13:47:35,716 [denest.session] INFO: Session '03_3_spikes' virtual running␣
↪time: 100 ms
2020-06-30 13:47:35,717 [denest.session] INFO: Session '03_3_spikes' real running␣
↪time: 0h:00m:00s
2020-06-30 13:47:35,722 [denest.simulation] INFO: Done running session '03_3_spikes'
2020-06-30 13:47:35,727 [denest.simulation] INFO: Finished running simulation
2020-06-30 13:47:35,745 [denest] INFO: Finished running simulation
2020-06-30 13:47:35,757 [denest] INFO: Total simulation virtual time: 400.0 ms
2020-06-30 13:47:35,762 [denest] INFO: Total simulation real time: 0h:00m:02s
2020-06-30 13:47:35,769 [denest] INFO: Simulation output written to: /Users/tom/
↪docker/nets-dev/docs/source/tutorials/data/outputs/2020-06-30_13:47:32
```

The simulation output was saved in the unique output directory created.

[7]:
```
print(output_dir)
!ls {output_dir}
```

```
data/outputs/2020-06-30_13:47:32
data                parameter_tree.yml session_times.yml  versions.txt
```

The `PARAMS_DIR/output` symlink points to the actual output directory.

[8]:
```
!ls {PARAMS_DIR/'output/'}
```

```
data                parameter_tree.yml session_times.yml  versions.txt
```

5.7.4 4. Commit the `PARAMS_DIR` with the updated params and `output` symlimk

CHAPTER 6

Installation

6.1 Local

1. Install NEST >= v2.14.0, <3.0 by following the instructions at http://www.nest-simulator.org/installation/.
2. Set up a Python 3 environment and install deNEST with:

```
pip install denest
```

6.2 Docker

A Docker image is provided with NEST 2.20 installed, based on nest-docker.

1. From within the repo, build the image:

```
docker build --tag denest .
```

2. Run an interactive container:

```
docker run \
  -it \
  --name denest_simulation \
  --volume $(pwd):/opt/data \
  --publish 8080:8080 \
  denest \
  /bin/bash
```

3. Install deNEST within the container:

```
pip install -e .
```

4. Use deNEST from within the container.

For more information on how to use the NEST Docker image, see nest-docker.

CHAPTER 7

Overview

7.1 Definitions

7.1.1 Network

- We use the term **network** to mean a full network in NEST, consisting of layers of units with specific models, projections of specific types with specific synapse models amongst these layers, population recorders (multimeters, spike detectors) and projection recorders (weight recorder).
  - The full network is represented in deNEST by the `Network` class.
- New NEST **models** (**neuron and generator model**, **synapse model** or **recorder model**) can be specified with arbitrary parameters. During network creation, models with specific parameters are created in NEST using a `nest.CopyModel()` or a `nest.SetDefaults()` call.
  - Synapse models are represented by the `SynapseModel` class in deNEST. All other models are represented by the `Model` class.
  - Neuron and generator models are specified as leaves of the `network/neuron_models` parameter subtree (see section below)
  - Synapse models are specified as leaves of the `network/synapse_models` parameter subtree (see "Network parameters" section below)
  - Recorder models are specified as leaves of the `network/recorder_models` parameter subtree (see "Network parameters" section below)
- A **layer** is a NEST topological layer, created with a `tp.CreateLayer()` call in NEST. A **population** is all the nodes of the same model within a layer.
  - Layers are represented by the `Layer` or `InputLayer` class in deNEST.
  - Layers can be of the type `InputLayer` when they are composed of generators. An extra population of parrot neurons can be automatically created and connected one-to-one to the generators, such that recording of generators' activity is possible. Additionally, `InputLayer` supports shifting the `origin` flag of stimulators at the start of a `Session`.

  - Layers are specified as leaves of the `network/layers` parameter subtree (see "Network parameters" section below).

- A **projection model** is a template specifying parameters passed to `tp.ConnectLayers`, and that individual projections amongst populations can inherit from.
  - Projection models are represented by the `ProjectionModel` class in deNEST.
  - Projection models are specified as leaves of the `network/projection_models` parameter subtree (see "Network parameters" section below).
- A **projection** is an individual projection between layers or populations, created with a `tp.ConnectLayers()` call. The parameters passed to `tp.ConnectLayers()` are those of the "projection model" of the specific projection.
  - The list of all individual projections within the network is specified in the `'projections'` parameter of the `network/topology` parameter subtree (see "Network parameters" section below).
- A **population recorder** is a recorder connected to all the nodes of a given population. A **projection recorder** is a recorder connected to all the synapses of a given projection. Recorders with arbitrary parameters can be defined by creating "recorder models". However, currently only recorders based on the 'multimeter', 'spike_detector' and 'weight_recorder' NEST models are supported.
  - population and projection recorders are represented by the `PopulationRecorder` and `ProjectionRecorder` classes in deNEST.
  - The list of all population recorders and projection recorders are specified in the `'population_recorders'` and `'projection_recorders'` parameters of the `network/recorders` parameter subtree (See "Network parameters" section below).

7.1.2 Simulation

- A **session model** is a template specifying parameters inherited by individual sessions.
  - session models are specified as leaves of the `session_models` parameter subtree (see "Simulation parameters" section below)
- A **session** is a period of simulation of a network with specific inputs and parameters, and corresponds to a single `nest.Simulate()` call. The parameters used by a given session are inherited from its session model.
  - A session's parameters define the operations that may be performed before running it:
    1. Modifying the state of some units (using the `Network.set_state()` method)
    2. (Possibly) shift the `origin` flag for the `InputLayer` stimulators
    3. (Possibly) deactivate the recorders for that session by setting their `start` flag to the end of the session
  - Individual sessions are represented by the `Session` object in deNEST. (see "Simulation parameters" section below)
- A **simulation** is a full experiment. It is represented by the `Simulation` object in deNEST, which contains a `Network` object and a list of `Session` objects.
  - The list of sessions run during a simulation is specified by the `sessions` parameter of the `simulation` parameter subtree (eg: `sessions: ['warmup', 'noise', 'grating', 'noise', 'grating']`) (see "Simulation parameters" section below).

7.2 Overview of a full simulation

A full deNEST simulation consists of the following steps:

1. **Initialize simulation** (`Simulation.__init__(<params_tree>)()`)
    1. **Initialize kernel**: (`Simulation.init_kernel(<kernel_subtree>)()`)
        1. Set NEST kernel parameters
        2. Set seed for NEST's random generator.
    2. **Create network** (`Simulation.create_network(<network_subtree>)()`):
        1. Initialize the network objects (`Network.__init__(<network_subtree)()`)
        2. Create the objects in NEST (`Network.create()`)
    3. **Initialize the sessions** (`Session.__init__()`)
    4. **Save the simulation's metadata**
        - Create and clean the output directory
        - Save the full simulation parameter tree
        - Save deNEST and NEST version information
        - Save session times
        - Save network metadata
        - Save session metadata
2. **Run the simulation** (`Simulation.run()`). This runs each session in turn:
    1. Initialize session (`Session.initialize()`)
        - (Possibly) reset the network
        - (Possibly) inactivate recorders for the duration of the session
        - (Possibly) shift the *origin* of stimulator devices to the start of the session
        - (Possibly) Change some of the network's parameters using the `Network.set_state()` method
            1. Change neuron parameters
            2. Change synapse parameters
    2. Call `nest.Simulate()`.

7.3 Specifying the simulation parameters

All parameters used by deNEST are specified in tree-like YAML files which are converted to `ParamsTree` objects.

In this section, we describe the `ParamsTree` objects, the expected structure of the full parameter tree interpreted by deNEST, and the expected formats and parameters of each of the subtrees that define the various aspects of the network and simulation.

7.3.1 Main parameter file

To facilitate defining parameters in separate files, *denest.run()* and *denest.load_trees()* take as input a path to a YAML file containing the relative paths of the tree-like YAML files to merge so as to define the full parameter tree (for examples, see the *Example declarative specification* or the `params/tree_paths.yml` file in the repository.).

7.3.2 The `ParamsTree` class

The `ParamsTree` class is instantiated from tree-like nested dictionaries. At each node, two reserved keys contain the node's data (called `'params'` and `'nest_params'`). All the other keys are interpreted as named children nodes.

The `'params'` key contains data interpreted by deNEST, while the `'nest_params'` key contains data passed to NEST without modification.

The `ParamsTree` class offers a tree structure with two useful characteristics:

- **Hierarchical inheritance of ancestor's data**: This provides a concise way of defining data for nested scopes. Data common to all leaves may be specified once in the root node, while more specific data may be specified further down the tree. Data lower within the tree overrides data higher in the tree. Ancestor nodes' `params` and `nest_params` are inherited independently.
- **(Horizontal) merging of trees**: `ParamsTree` objects can be merged horizontally with `ParamsTree.merge()`. During the merging of multiple params trees, the contents of the `params` and `nest_params` data keys of nodes at the same relative position are combined. This allows **splitting the deNEST parameter trees in separate files for convenience**, and **overriding the data of a node anywhere in the tree while preserving hierarchical inheritance**.

An example parameter tree

Below is an example of a YAML file with a tree-like structure that can be loaded and represented by the `ParamsTree` class:

```yaml
network:
  neuron_models:
    ht_neuron:
      params:                       # params common to all leaves
        nest_model: ht_neuron
      nest_params:                  # nest_params common to all leaves
        g_KL: 1.0
      cortical_excitatory:
        nest_params:
          tau_spike: 1.75
          tau_m: 16.0
        l1_exc:                     # leaf
        l2_exc:                     # leaf
          nest_params:
            g_KL: 2.0      # Overrides ancestor's value
      cortical_inhibitory:
        nest_params:
          tau_m: 8.0
        l1_inh:                     # leaf
```

This file can be loaded into a `ParamsTree` object. The leaves of the resulting `ParamsTree` and their respective data (`params` and `nest_params`) are as follows. Note the inheritance and override of ancestor data. The nested format above is more compact and less error prone when there are a lot of shared parameters between leaves.

```
l1_exc:
  params:
    nest_model: ht_neuron
  nest_params:
    g_KL: 1.0
    tau_spike: 1.75
    tau_m: 16.0
l2_exc:
  params:
    nest_model: ht_neuron
  nest_params:
    g_KL: 2.0
    tau_spike: 1.75
    tau_m: 16.0
l1_inh:
  params:
    nest_model: ht_neuron
  nest_params:
    g_KL: 1.0
    tau_m: 8.0
```

7.3.3 Full parameter tree: expected structure

All the aspects of the overall simulation are specified in specific named subtrees.

The overall `ParamsTree` passed to `denest.Simulation()` is expected to have no data and the following children:

- `simulation` (`ParamsTree`). Defines input and output paths, and the simulation steps performed. The following parameters (`params` field) are recognized:
  - `output_dir` (str): Path to the output directory. (Default: `'output'`)
  - `input_dir` (str): Path to the directory in which input files are searched for for each session. (Default: `'input'`)
  - `sessions` (list(str)): Order in which sessions are run. Elements of the list should be the name of session models defined in the `session_models` parameter subtree (Default: `[]`)
- `kernel` (`ParamsTree`): Used for NEST kernel initialization. Refer to `Simulation.init_kernel()` for a description of kernel parameters.
- `session_models` (`ParamsTree`): Parameter tree, the leaves of which define session models. Refer to `Sessions()` for a description of session parameters.
- `network` (`ParamsTree`): Parameter tree defining the network in NEST. Refer to `Network` for a full description of network parameters.

7.3.4 `"network"` parameter tree: expected structure

All network parameters are specified in the `network` subtree, used to initialize the `Network` object.

The `network` subtree should have no data, and the following children are expected:

- `neuron_models` (`ParamsTree`). Parameter tree, the leaves of which define neuron models. Each leaf is used to initialize a `Model` object

- `synapse_models` (`ParamsTree`). Parameter tree, the leaves of which define synapse models. Each leaf is used to initialize a `SynapseModel` object
- `layers` (`ParamsTree`). Parameter tree, the leaves of which define layers. Each leaf is used to initialize a `Layer` or `InputLayer` object depending on the value of their `type` `params` parameter.
- `projection_models` (`ParamsTree`). Parameter tree, the leaves of which define projection models. Each leaf is used to initialize a `ProjectionModel` object.
- `recorder_models` (`ParamsTree`). Parameter tree, the leaves of which define recorder models. Each leaf is used to initialize a `Model` object.
- `topology` (`ParamsTree`). `ParamsTree` object without children, the `params` of which may contain a `projections` key specifying all the individual population-to-population projections within the network as a list. `Projection` objects are created from the `topology` `ParamsTree` object by the `Network.build_projections` method. Refer to this method for a description of the `topology` parameter.
- `recorders` (`ParamsTree`). `ParamsTree` object without children, the `params` of which may contain a `population_recorders` and a `projection_recorders` key specifying all the network recorders. `PopulationRecorder` and `ProjectionRecorder` objects are created from the `recorders` `ParamsTree` object by the `Network.build_recorders` method. Refer to this method for a description of the `recorders` parameter.

7.4 Running a deNEST Simulation

- From Python (*e.g.* in a Jupyter notebook):
  - Using the `Simulation` object to run the simulation step by step:

```
import denest

# Path to the parameter files to use
params_path = 'params/tree_paths.yml'

# Override some parameters loaded from the file
overrides = [

  # Maybe change the nest kernel's settings ?
  {'kernel': {'nest_params': {'local_num_threads': 20}}},

  # Maybe change a parameter for all the projections at once ?
  {'network': {'projection_models': {'nest_params': {
      'allow_autapses': true
  }}}},
]

# Load the parameters
params = denest.load_trees(params_path, *overrides)

# Initialize the simulation
sim = denest.Simulation(params, output_dir='output')

# Run the simulation (runs all the sessions)
sim.run()
```

  - Using the *`denest.run()`* function to run the full simulation at once:

```
import denest

# Path to the parameter files to use
params_path = 'params/tree_paths.yml'

# Override parameters
overrides = []

denest.run(params_path, *overrides, output_dir=None)
```

- From the command line:

```
python -m denest <tree_paths.yml> [-o <output_dir>]
```

CHAPTER 8

Example declarative specification

Here is an example parameter file (in YAML) that specifies a full simulation:

```yaml
params: {}
nest_params: {}
session_models:
  params:
    reset_network: false
    record: true
    shift_origin: false
  nest_params: {}
  even_rate:
    params:
      simulation_time: 50.0
      unit_changes:
      - layers:
        - input_layer
        population_name: input_exc
        change_type: constant
        from_array: false
        nest_params:
          rate: 100.0
    nest_params: {}
  warmup:
    params:
      reset_network: true
      record: false
      simulation_time: 50.0
      unit_changes:
      - layers:
        - l1
        population_name: null
        change_type: constant
        from_array: false
        nest_params:
```

(continues on next page)

(continued from previous page)

```
          V_m: -70.0
      - layers:
        - input_layer
        population_name: input_exc
        change_type: constant
        from_array: false
        nest_params:
          rate: 100.0
    nest_params: {}
  arbitrary_rate:
    params:
      simulation_time: 50.0
      unit_changes:
      - layers:
        - input_layer
        population_name: input_exc
        change_type: constant
        from_array: true
        nest_params:
          rate: ./input_layer_rates_5x5x1.npy
    nest_params: {}
simulation:
  params:
    sessions:
    - warmup
    - even_rate
    - arbitrary_rate
    output_dir: ./output
    input_dir: ./params/input
  nest_params: {}
kernel:
  params:
    extension_modules: []
    nest_seed: 94
  nest_params:
    local_num_threads: 20
    resolution: 1.0
    print_time: true
    overwrite_files: true
network:
  params: {}
  nest_params: {}
  neuron_models:
    params: {}
    nest_params: {}
    ht_neuron:
      params:
        nest_model: ht_neuron
      nest_params:
        g_peak_NaP: 0.5
        g_peak_h: 0.0
        g_peak_T: 0.0
        g_peak_KNa: 0.5
        g_KL: 1.0
        E_rev_NaP: 55.0
        g_peak_AMPA: 0.1
        g_peak_NMDA: 0.15
```

(continues on next page)

(continued from previous page)

```
        g_peak_GABA_A: 0.33
        g_peak_GABA_B: 0.0132
        instant_unblock_NMDA: true
        S_act_NMDA: 0.4
        V_act_NMDA: -58.0
      cortical_inhibitory:
        params: {}
        nest_params:
          theta_eq: -53.0
          tau_theta: 1.0
          tau_spike: 0.5
          tau_m: 8.0
        l1_inh:
          params: {}
          nest_params: {}
        l2_inh:
          params: {}
          nest_params: {}
      cortical_excitatory:
        params: {}
        nest_params:
          theta_eq: -51.0
          tau_theta: 2.0
          tau_spike: 1.75
          tau_m: 16.0
        l1_exc:
          params: {}
          nest_params: {}
        l2_exc:
          params: {}
          nest_params: {}
    input_exc:
      params:
        nest_model: poisson_generator
      nest_params: {}
  layers:
    params:
      type: null
    nest_params:
      rows: 5
      columns: 5
      extent:
      - 8.0
      - 8.0
      edge_wrap: true
    input_area:
      params:
        type: InputLayer
        add_parrots: true
      nest_params: {}
      input_layer:
        params:
          populations:
            input_exc: 1
        nest_params: {}
    l1_area:
      params: {}
```

(continues on next page)

(continued from previous page)

```
      nest_params: {}
      l1:
        params:
          populations:
            l1_exc: 2
            l1_inh: 1
        nest_params: {}
    l2_area:
      params: {}
      nest_params: {}
      l2:
        params:
          populations:
            l2_exc: 2
            l2_inh: 1
        nest_params: {}
  synapse_models:
    params: {}
    nest_params: {}
    static_synapse:
      params:
        nest_model: static_synapse_lbl
      nest_params: {}
      input_synapse_NMDA:
        params:
          target_neuron: ht_neuron
          receptor_type: NMDA
        nest_params: {}
      input_synapse_AMPA:
        params:
          target_neuron: ht_neuron
          receptor_type: AMPA
        nest_params: {}
    ht_synapse:
      params:
        nest_model: ht_synapse
        target_neuron: ht_neuron
      nest_params: {}
      GABA_B_syn:
        params:
          receptor_type: GABA_B
        nest_params: {}
      AMPA_syn:
        params:
          receptor_type: AMPA
        nest_params: {}
      GABA_A_syn:
        params:
          receptor_type: GABA_A
        nest_params: {}
      NMDA_syn:
        params:
          receptor_type: NMDA
        nest_params: {}
  topology:
    params:
      projections:
```

(continues on next page)

(continued from previous page)

```
    - source_layers:
      - input_layer
      source_population: parrot_neuron
      target_layers:
      - l1
      target_population: l1_exc
      projection_model: input_projection_AMPA
    - source_layers:
      - input_layer
      source_population: parrot_neuron
      target_layers:
      - l1
      target_population: l1_inh
      projection_model: input_projection_AMPA
    - source_layers:
      - input_layer
      source_population: parrot_neuron
      target_layers:
      - l1
      target_population: l1_inh
      projection_model: input_projection_NMDA
    - source_layers:
      - l1
      source_population: l1_exc
      target_layers:
      - l1
      target_population: l1_exc
      projection_model: horizontal_exc
    - source_layers:
      - l1
      source_population: l1_exc
      target_layers:
      - l1
      target_population: l1_inh
      projection_model: horizontal_exc
    - source_layers:
      - l1
      source_population: l1_exc
      target_layers:
      - l2
      target_population: l2_exc
      projection_model: FF_exc
    - source_layers:
      - l1
      source_population: l1_exc
      target_layers:
      - l2
      target_population: l2_inh
      projection_model: FF_exc
  nest_params: {}
recorder_models:
  params: {}
  nest_params:
    record_to:
    - file
    - memory
    withgid: true
```

(continues on next page)

(continued from previous page)

```
      withtime: true
    spike_detector:
      params:
        nest_model: spike_detector
      nest_params: {}
    weight_recorder:
      params:
        nest_model: weight_recorder
      nest_params:
        record_to:
        - file
        - memory
        withport: false
        withrport: true
    multimeter:
      params:
        nest_model: multimeter
      nest_params:
        interval: 1.0
        record_from:
        - V_m
  recorders:
    params:
      population_recorders:
      - layers: []
        populations: []
        model: multimeter
      - layers:
        - l2
        populations:
        - l2_inh
        model: multimeter
      - layers: null
        populations:
        - l2_exc
        model: multimeter
      - layers:
        - l1
        populations: null
        model: multimeter
      - layers: null
        populations: null
        model: spike_detector
      projection_recorders:
      - source_layers:
        - input_layer
        source_population: parrot_neuron
        target_layers:
        - l1
        target_population: l1_exc
        projection_model: input_projection_AMPA
        model: weight_recorder
      - source_layers:
        - l1
        source_population: l1_exc
        target_layers:
        - l1
```

(continues on next page)

(continued from previous page)

```
      target_population: l1_exc
      projection_model: horizontal_exc
      model: weight_recorder
  nest_params: {}
projection_models:
  params:
    type: topological
  nest_params:
    allow_autapses: false
    allow_multapses: false
    allow_oversized_mask: true
  horizontal_inh:
    params: {}
    nest_params:
      connection_type: divergent
      synapse_model: GABA_A_syn
      mask:
        circular:
          radius: 7.0
      kernel:
        gaussian:
          p_center: 0.25
          sigma: 7.5
      weights: 1.0
      delays:
        uniform:
          min: 1.75
          max: 2.25
  input_projection:
    params: {}
    nest_params:
      connection_type: convergent
      mask:
        circular:
          radius: 12.0
      kernel: 0.8
      weights: 1.0
      delays:
        uniform:
          min: 1.75
          max: 2.25
    input_projection_AMPA:
      params: {}
      nest_params:
        synapse_model: input_synapse_AMPA
    input_projection_NMDA:
      params: {}
      nest_params:
        synapse_model: input_synapse_NMDA
  horizontal_exc:
    params: {}
    nest_params:
      connection_type: divergent
      synapse_model: AMPA_syn
      mask:
        circular:
          radius: 12.0
```

(continues on next page)

(continued from previous page)

```
        kernel:
          gaussian:
            p_center: 0.05
            sigma: 7.5
        weights: 1.0
        delays:
          uniform:
            min: 1.75
            max: 2.25
    FF_exc:
      params: {}
      nest_params:
        connection_type: convergent
        synapse_model: AMPA_syn
        mask:
          circular:
            radius: 12.0
        kernel: 0.8
        weights: 1.0
        delays:
          uniform:
            min: 1.75
            max: 2.25
```

8.1 User interface

These are the main functions and classes that comprise deNEST's user interface.

load_trees(path, *overrides)	Load a list of parameter files, optionally overriding some values.
run(path, *overrides[, output_dir, input_dir])	Run the simulation specified by the parameters at `path`.
Simulation([tree, input_dir, output_dir])	Represents a simulation.
Session(name, params[, start_time, input_dir])	Represents a sequence of stimuli.
Network([tree])	Represent a full network.
network.Layer(name, params, nest_params)	Represents a NEST layer composed of populations of units
ParamsTree([mapping, parent, name, validate])	A tree of nodes that inherit and override ancestors' data.

deNEST: a declarative frontend for NEST

denest.**load_trees**(*path*, **overrides*)

Load a list of parameter files, optionally overriding some values.

Parameters

- **path** (*str*) – The filepath to load.
- ***overrides** (*tree-like*) – Variable number of tree-like parameters that should override those from the path. Last in list is applied first.

Returns The loaded parameter tree with overrides applied.

Return type *ParamsTree*

denest.**run**(*path*, **overrides*, *output_dir=None*, *input_dir=None*)

Run the simulation specified by the parameters at `path`.

Parameters

- **path** (*str*) – The filepath of a parameter file specifying the simulation.
- ***overrides** (*tree-like*) – Variable number of tree-like parameters that should override those from the path. Last in list is applied first.

Keyword Arguments

- **input_dir** (*str | None*) – `None` or the path to the input. Passed to *Simulation*. If defined, overrides the `input_dir` simulation parameter.
- **output_dir** (*str | None*) – None or the path to the output directory. Passed to *Simulation* If defined, overrides the `output_dir` simulation parameter.

class denest.**Simulation**(*tree=None*, *input_dir=None*, *output_dir=None*)

Represents a simulation.

Handles building the network, running it with a series of sessions, and saving output.

Parameters **tree** (ParamsTree) – Full simulation parameter tree. The following `ParamsTree` subtrees are expected:

simulation (***ParamsTree***) Defines input and output paths, and the simulation steps performed. The following parameters (`params` field) are recognized:

output_dir **(str)** Path to the output directory. (Default: `'output'`)

input_dir **(str)** Path to the directory in which input files are searched for for each session. (Default: `'input'`)

sessions **(list[str])** Order in which sessions are run. Elements of the list should be the name of session models defined in the `session_models` parameter subtree. (Default: `[]`)

kernel (***ParamsTree***) Used for NEST kernel initialization. Refer to *Simulation.init_kernel()* for a description of kernel parameters.

session_models (***ParamsTree***) Parameter tree, the leaves of which define session models. Refer to `Session` for a description of session parameters.

network (***ParamsTree***) Parameter tree defining the network in NEST. Refer to *Network* for a full description of network parameters.

Keyword Arguments

- **input_dir** (*str | None*) – None or the path to the input. If defined, overrides the `input_dir` simulation parameter.
- **output_dir** (*str | None*) – None or the path to the output directory. If defined, overrides the `output_dir` simulation parameter.

Initialize simulation.

- Set input and output paths
- Initialize NEST kernel
- Initialize and build Network in NEST
- Create sessions
- Save simulation metadata

create_network(*network_tree*)
Build and create the network.

Adds `network_tree` as `'network'` child of `self.tree`

save_metadata(*clear_output_dir=False*)
Save simulation metadata.

- Save parameters
- Save deNEST git hash
- Save sessions metadata (`Session.save_metadata()`)
- Save session times (start and end kernel time for each session)
- Save network metadata (*`Network.save_metadata()`*)

Keyword Arguments **clear_output_dir** (*bool*) – If true, delete the contents of the output directory.

run()
Run simulation.

Run sessions in the order specified by the `'sessions'` simulation parameter.

build_sessions(*sessions_order*)
Build a list of sessions.

Session params are inherited from session models.

build_session_models(*tree*)
Create session models from the leaves of a tree.

Adds `tree` as the `'session_models'` child of `self.tree`

init_kernel(*kernel_tree*)
Initialize the NEST kernel.

- Call `nest.SetKernelStatus` with `nest_params`
- Set NEST kernel `data_path` and seed
- Install extension modules

Adds `kernel_tree` as the `'kernel'` child of `self.tree`.

Parameters **kernel_tree** (`ParamsTree`) – Parameter tree without children. The following parameters (`params` field) are recognized:

extension_modules: (list(str)) List of modules to install. (Default: `[]`)

nest_seed: (int) Used to set NEST kernel's RNG seed. (Default: `1`)

NEST parameters (`nest_params` field) are passed to `nest.SetKernelStatus`. The following nest parameters are reserved: `[data_path, 'grng_seed', 'rng_seed']`. The NEST seeds should be set via the `'nest_seed'` kernel parameter parameter.

class denest.**Network**(*tree=None*)
Represent a full network.

Parameters **tree** (`ParamsTree`) – "network" parameter tree. The following children are expected:

neuron_models (*ParamsTree*) Parameter tree, the leaves of which define neuron models. Each leaf is used to initialize a `Model` object.

synapse_models (*ParamsTree*) Parameter tree, the leaves of which define synapse models. Each leaf is used to initialize a `SynapseModel` object.

layers (*ParamsTree*) Parameter tree, the leaves of which define layers. Each leaf is used to initialize a :class:`Layer` or :class:`InputLayer` object depending on the value of their `type` parameter.

projection_models (*ParamsTree*) Parameter tree, the leaves of which define projection models. Each leaf is used to initialize a :class:`ProjectionModel` object.

recorder_models (*ParamsTree*) Parameter tree, the leaves of which define recorder models. Each leaf is used to initialize a :class:`Model` object.

topology (*ParamsTree*) *ParamsTree* object without children, the `params` of which may contain a `projections` key specifying all the individual population-to-population projections within the network as a list. `Projection` objects are created from the `topology` parameters by the *Network.build_projections()* method. Refer to this method for a description of the `topology` parameter.

recorders (*ParamsTree*) :class:`ParamsTree` object without children, the `params` of which may contain a `population_recorders` and a `projection_recorders` key specifying all the network recorders. `PopulationRecorder` and `ProjectionRecorder` objects are created from the `recorders` parameters by the *Network.build_recorders()* method. Refer to this method for a description of the `recorders` parameter.

Initialize the network object without creating it in NEST.

build_neuron_models(*tree*)

Initialize `self.neuron_models` from the leaves of a tree.

Note: Overwrites the `'neuron_models'` in the network's parameters.

Parameters **tree** (*tree-like or ParamsTree*) – which define neuron models. Each leaf is used to initialize a `Model` object.

build_synapse_models(*tree*)

Initialize `self.synapse_models` from the leaves of a tree.

Note: Overwrites the `'synapse_models'` in the network's parameters.

Parameters **tree** (*tree-like or ParamsTree*) – which define neuron models. Each leaf is used to initialize a `SynapseModel` object.

build_recorder_models(*tree*)

Initialize `self.recorder_models` from the leaves of a tree.

Note: Overwrites the `'recorder_models'` in the network's parameters.

Parameters **tree** (tree-like or ParamsTree) – which define neuron models. Each leaf is used to initialize a Model object.

build_layers(*tree*)
Initialize self.layers from the leaves of a tree.

Note: Overwrites the 'layers' in the network's parameters.

Parameters **tree** (*tree-like or ParamsTree*) – which define layers. Each leaf is used to initialize a Layer or InputLayer objecs depending on the value of the type parameter.

build_projection_models(*tree*)
Initialize self.projection_models from the leaves of a tree.

Note: Overwrites the 'projection_models' in the network's parameters.

Parameters **tree** (*tree-like or ParamsTree*) – which define projection models. Each leaf is used to initialize a ProjectionModel object.

build_projections(*topology_tree*)
Initialize self.projections from the topology tree.

Initialize self.projections with a list of Projection objects.

Parameters **topology_tree** (*tree-like or ParamsTree*) – Tree-like or *ParamsTree* without children, the parameters of which may contain a projections parameter entry. (Default: []). The value of the projections parameter is a list of items describing the projections to be created. Each item must be a dictionary of the following form:

```
{
    'projection_model' : <projection_model>,
    'source_layers': <source_layers_list>,
    'source_population': <source_population>,
    'target_layers': <target_layers_list>,
    'target_population': <target_population>,
}
```

where:

- <projection_model> is the name of the projection model. Projection models are specified in the projection_models network parameter.
- <source_layers_list> and <target_layers_list> are lists of source and target layer names. Projections are created for all (source_layer, target layer) pairs.
- <source_population> and <target_population> are None or the name of source and target populations for the created projection. If None, all populations in the source or target layer are connected.

Together, <projection_model_name>, <source_layer_name>, <source_population_name>, <target_layer_name>, and <target_population_name> fully specify each individual projection and must be unique.

build_recorders(*recorders_tree*)
Initialize recorders from tree.

Validates the `recorders` parameter tree and calls `Network.build_population_recorders()` and `Network.build_projection_recorders()` to initialize the `Network.population_recorders()` and `Network.projection_recorders()` attributes.

Parameters **recorders_tree** (*tree-like or ParamsTree*) – Tree-like or *ParamsTree* object without children nor `nest_params`, the parameters of which may contain a `population_recorders` (default: `[]`) and a `projection_recorders` (default: `[]`) entry specifying the network's recorders. The `population_recorders` and `projection_recorders` entries are passed to `Network.build_population_recorders()` and `Network.build_projection_recorders()` respectively.

Returns (list(`PopulationRecorder`), list(`ProjectionRecorder`))

get_recorders(*recorder_class=None*, *recorder_type=None*)
Yield all `PopulationRecorder` and `ProjectionRecorder` objects.

Parameters

- **recorder_class** (*str or None*) – Class of queried recorders. `"PopulationRecorder"`, `"ProjectionRecorder"` or `None`.
- **recorder_type** (*str or None*) – Type of queried recorders. `'multimeter'`, `'spike_detector'` or `'projection_recorder'`.

static change_synapse_states(*synapse_changes*)
Change parameters for some projections of a population.

Parameters **synapse_changes** (*list*) – List of dictionaries each of the form:

```
{
    'synapse_model': <synapse_model>,
    'params': {<param1>: <value1>}
}
```

where the dictionary in `params` is passed to `nest.SetStatus()` to set the parameters for all projections with synapse model `<synapse_model>`.

set_state(*unit_changes=None*, *synapse_changes=None*, *input_dir=None*)
Set the state of some units and synapses.

Parameters **unit_changes** (*list*) – List of dictionaries specifying the changes applied to the networks units:

```
{
    'layers': <layer_name_list>,
    'population': <pop_name>,
    'change_type': <change_type>,
    'from_array': <from_array>,
    'nest_params': {
        <param_name>: <param_change>,
    },
}
```

where `<layer_name_list>` and `<population_name>` specify all the individual populations to which the changes are applied:

- <layer_name_list> (list(str) | None) is the list of names of the considered layers. If None, the changes may be applied to populations from all the layers. (Default: [])
- <population_name> (str | None) is the name of the considered population in each layer. If None, changes are applied to all the populations of each considered layer. (Default: None)

and <change_type>, <from_array> and 'nest_params' specify the changes applied to units from each of those populations:

- <change_type> ('constant', 'multiplicative' or 'additive'). If 'multiplicative' (resp. 'additive'), the set value for each unit and parameter is the product (resp. sum) between the preexisting value and the given value. If 'constant', the given value for each unit is set without regard for the preexisting value. (Default: 'constant')
- <from_array> (bool) specifies how the <param_change> value given for each parameter is interpreted:
  - If True, param_change should be a numpy array or the relative path from input_dir to a numpy array. The given or loaded array should have the same dimension as the considered population, and its values are mapped to the population's units to set the <param_name> parameter.
  - If False, the value in param_change is used to set the <param_name> parameter for all the population's units.
- 'nest_params' (Default: {}) is the dictionary specifying the parameter changes applied to the population units. Items are the name of the modified NEST parameters (<param_name>) and the values set (<param_change>). The <change_type> and <from_array> parameters specify the interpretation of the <param_change> value.

Examples

```
>>> # Load parameter files and create the network object
>>> import denest
>>> network = denest.Network(denest.ParamsTree.read('<path_to_parameter_file>
↪'))
```

```
>>> # Instantiate the network in the NEST kernel
>>> network.create()
```

```
>>> # Set the same spike times for all the units of a population of spike
... # generators
>>> network.set_state({
...     'layers': ['input_layer'],
...     'population_name': 'spike_generator',
...     'change_type': 'constant',
...     'from_array': False,
...     'nest_params': {'spike_times': [1.0, 2.0, 3.0]}
... })
```

```
>>> # Set the voltage from values for multiple 2x2 population of neurons:␣
↪specify
... # the array directly
>>> voltages = np.array([[-70.0, -65.0], [-70.0, -65.0]])
```

(continues on next page)

(continued from previous page)

```
>>> network.set_state({
...     'layers': ['l1', 'l2'],
...     'population_name': None,
...     'change_type': 'constant',
...     'from_array': True,
...     'nest_params': {'V_m': voltages}
... })
```

```
>>> # Set the voltage from values for a 2x2 population of neurons: pass
... # the path to the array
>>> np.save(voltages, './voltage_change.npy')
>>> network.set_state({
...     'layers': ['l1'],
...     'population_name': 'l1_exc',
...     'change_type': 'constant',
...     'from_array': True,
...     'nest_params': {'V_m': './voltage_change.npy'}
... })
```

```
>>> # Multiply the leak potential by 2 for all the units
>>> network.set_state({
...     'layers': ['l1'],
...     'population_name': 'l1_exc',
...     'change_type': 'multiplicative',
...     'from_array': False,
...     'nest_params': {'g_peak_AMPA': 2.0}
... })
```

save_metadata(*output_dir*)
Save network metadata.

- Save recorder metadata

populations_by_gids(*layer_type='Layer'*)
Return a dict of the form `{'gid': (layer_name, pop_name)}`.

class denest.network.**Layer**(*name*, *params*, *nest_params*)
Represents a NEST layer composed of populations of units

Parameters

- **name** (*str*) – Name of the layer
- **params** (*dict-like*) – Dictionary of parameters. The following parameters are expected:

  populations (dict): Dictionary of the form ‘‘{<model>: <number>} specifying the elements within the layer. Analogous to the `elements` nest.Topology parameter
- **nest_params** (*dict-like*) – Dictionary of parameters that will be passed to NEST during the `nest.CreateLayer` call. The following parameters are mandatory: `['rows', 'columns']`. The `elements` parameter is reserved. Please use the `populations` parameter instead to specify layer elements.

create()
Create the layer in NEST and update attributes.

gids(*population=None*, *location=None*, *population_location=None*)
Return element GIDs, optionally filtered by population/location.

Parameters

- **population** (*str*) – Matches population name that has `population` as a substring.
- **location** (*tuple[int]*) – The location within the layer to filter by.
- **population_location** (*tuple[int]*) – The location within the population to filter by.

Returns The GID(s).

Return type list

shape
Return layer shape (eg `(nrows, ncols)`)

layer_shape
Return layer shape (eg `(nrows, ncols)`)

population_shapes
(nrows, ncols, nunits)‘‘

Type Return population shapes

Type ‘‘{<pop_name>

locations
index}‘‘ dict of locations within the layer.

Type Return ‘‘{<gid>

population_locations
index}‘‘ dict of locations within the population.

There's an extra (last) dimension for the population locations compared to the [layer] locations, corresponding to the index of the unit within the population.

Type Return ‘‘{<gid>

population_names
Return a list of population names within this layer.

recordable_population_names
Return list of names of recordable population names in this layer.

class denest.**ParamsTree**(*mapping=None*, *parent=None*, *name=None*, *validate=True*)
A tree of nodes that inherit and override ancestors' data.

A tree is created from a tree-like mapping. The key-value pairs containing the node's data are defined in `self.DATA_KEYS` (`params` and `nest_params`). Data is inherited from ancestors for each of those keys. Note that the order of traversals is undefined.

Keyword Arguments

- **mapping** (*Mapping*) – A dictionary-like object that maps names to children, but with special key-value pairs containing the node's data. Defaults to an empty dictionary.
- **parent** (ParamsTree) – The parent tree. Data from each of the data keys is inherited from ancestors.

name
Name of the node.

Type str | None

children
A dictionary of named children.

Type dict

parent
The parent of this node.

Type type(self)

data
Dictionary containing the data accessible from this node (i.e., the node's data and that inherited from its parents). Keys are values of `self.DATA_KEYS` (`'params'` and `'nest_params'`).

Type dict

params, nest_params
Contain the node data. Syntactic sugar to allow access to the node's data.

Type dict-like

node_data
The data associated with this node (not inherited from parents).

parent
This node's parent. `None` if node is the root.

name
This name of this node.

children
A dictionary of this node's children

ancestors()
Return a list of ancestors of this node.

Doesn't include self. More recent ancestors appear earlier in the list.

leaves(*root=True*)
Return a list of leaf nodes of the tree.

Traversal order is not defined. If root is False and there is not children, returns an empty list

named_leaves(*root=True*)
Return list of `(<name>, <node>)` tuples for all the leaves.

Traversal order is not defined. If root is False and there is not children, returns an empty list

classmethod merge(**trees*, *parent=None*, *name=None*)
Merge trees into a new tree. Earlier trees take precedence.

If a node exists at the same location in more than one tree and these nodes have duplicate keys, the values from the nodes in the trees that appear earlier in the argument list will take precedence over those from later trees.

Equivalent nodes' data is merged horizontally before hierarchical inheritance.

copy()
Copy this `ParamsTree`.

asdict()
Convert this `ParamsTree` to a nested dictionary.

classmethod read(*path*)
Load a YAML representation of a tree from disk.

write(*path*)
Write a YAML representation of a tree to disk.

validate(*mapping*, *path=None*)
Check that a mapping is a valid `ParamsTree`.

Python Module Index

d

Index

V

W