bindsnet Documentation

Release 0.2.5

Daniel Saunders, Hananel Hazan

Aug 06, 2019

Contents:

BindsNET is built on top of the PyTorch deep learning platform. It is used for the simulation of spiking neural networks (SNNs) and is geared towards machine learning and reinforcement learning.

BindsNET takes advantage of the `torch.Tensor` object to build spiking neurons and connections between them, and simulate them on CPUs or GPUs (for strong acceleration / parallelization) without any extra work. Recently, `torchvision.datasets` has been integrated into the library to allow the use of popular vision datasets in training SNNs for computer vision tasks. Neural network functionality contained in `torch.nn.functional` module is used to implement more complex connections between populations of spiking neurons.

Spiking neural networks are sometimes referred to as the third generation of neural networks. Rather than the simple linear layers and nonlinear activation functions of deep learning neural networks, SNNs are composed of neural units which more accurately capture properties of their biological counterparts. An important difference between spiking neurons and the artificial neurons of deep learning are the former's integration of input *in time*; they are naturally short-term memory devices by their maintenance of a (possibly decaying) membrane voltage. As a result, some have argued that SNNs are particularly well-suited to model time-varying data.

Neurons are connected together with directed edges (*synapses*) which are (in general) plastic. Synapses may have their own dynamics as well, which may or may not *depend on pre- and post-synaptic neural activity https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3395004/* or *other biological signals https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4717313/.* The modification of synaptic strengths is thought to be an important mechanism by which organisms learn. Accordingly, BindsNET provides a module (**bindsnet.learning**) which contains functions used for the updating of synapse weights.

At its core, BindsNET provides software objects and methods which support the simulation of groups of different types of neurons (**bindsnet.network.nodes**), as well as different types of connections between them (**bindsnet.network.topology**). These may be arbitrarily combined together under a single **bindsnet.network.Network** object, which is responsible for the coordination of the simulation logic of all underlying components. On creation of a network, the user can specify a simulation timestep constant, dt, which determines the granularity of the simulation. Choosing this parameter induces a trade-off between simulation speed and numerical precision: large values result in fast simulation, but poor simulation accuracy, and vice versa. Monitors (**bindsnet.network.monitors**) are available for recording state variables from arbitrary network components (e.g., the voltage v of a group of neurons).

The development of BindsNET is supported by the Defense Advanced Research Project Agency Grant DARPA/MTO HR0011-16-l-0006.

CHAPTER 1

Installation

1.1 Pip install

Issue:

```
pip install -U bindsnet
```

1.2 Installing from source

On *nix systems, issue one of the following in a shell:

```
git clone https://github.com/Hananel-Hazan/bindsnet.git  # HTTPS
git clone git@github.com:Hananel-Hazan/bindsnet.git  # SSH
```

Change directory into `bindsnet` and issue one of the following:

```
pip install . # Typical install
pip install -e .  # Editable mode (package code can be edited without reinstall)
```

This will install `bindsnet` and all its dependencies.

1.3 Running the tests

If BindsNET is installed from source, install `pytest` and issue the following from BindsNET's installation directory:

```
python -m pytest test
```

CHAPTER 2

Quickstart

Check out some example use cases for BindsNET in the `examples/` folder (link). For example, changing directory to *[bindsnet-root]/examples/mnist* and running the following will result in a near-replication of the architecture of Diehl & Cook 2015:

```
python eth_mnist.py [options]
```

The token `[options]` should be replaced with any command-line arguments you'd like to use to modify the behavior of the program.

CHAPTER 3

BindsNET User Manual

Welcome to BindsNET's user manual! To get started, click on one of the links below.

- guide_part_i
- guide_part_ii

CHAPTER 4

bindsnet package

4.1 Subpackages

4.1.1 bindsnet.analysis package

Submodules

bindsnet.analysis.plotting module

bindsnet.analysis.plotting.**plot_assignments**(*assignments: torch.Tensor*, *im: Optional[matplotlib.image.AxesImage] = None*, *figsize: Tuple[int, int] = (5, 5)*, *classes: Optional[Sized] = None*) → matplotlib.image.AxesImage

Plot the two-dimensional neuron assignments.

Parameters

- **assignments** – Vector of neuron label assignments.
- **im** – Used for re-drawing the assignments plot.
- **figsize** – Horizontal, vertical figure size in inches.
- **classes** – Iterable of labels for colorbar ticks corresponding to data labels.

Returns Used for re-drawing the assigments plot.

bindsnet.analysis.plotting.**plot_conv2d_weights**(*weights: torch.Tensor*, *wmin: float = 0.0*, *wmax: float = 1.0*, *im: Optional[matplotlib.image.AxesImage] = None*, *figsize: Tuple[int, int] = (5, 5)*, *cmap: str = 'hot_r'*) → matplotlib.image.AxesImage

Plot a connection weight matrix of a Conv2dConnection.

Parameters

- **weights** – Weight matrix of Conv2dConnection object.
- **wmin** – Minimum allowed weight value.
- **wmax** – Maximum allowed weight value.
- **im** – Used for re-drawing the weights plot.
- **figsize** – Horizontal, vertical figure size in inches.
- **cmap** – Matplotlib colormap.

Returns Used for re-drawing the weights plot.

bindsnet.analysis.plotting.**plot_input**(*image: torch.Tensor*, *inpt: torch.Tensor*, *label: Optional[int] = None*, *axes: List[matplotlib.axes._axes.Axes] = None*, *ims: List[matplotlib.image.AxesImage] = None*, *figsize: Tuple[int, int] = (8, 4)*) → Tuple[List[matplotlib.axes._axes.Axes], List[matplotlib.image.AxesImage]]

Plots a two-dimensional image and its corresponding spike-train representation.

Parameters

- **image** – A 2D array of floats depicting an input image.
- **inpt** – A 2D array of floats depicting an image's spike-train encoding.
- **label** – Class label of the input data.
- **axes** – Used for re-drawing the input plots.
- **ims** – Used for re-drawing the input plots.
- **figsize** – Horizontal, vertical figure size in inches.

Returns Tuple of `(axes, ims)` used for re-drawing the input plots.

bindsnet.analysis.plotting.**plot_locally_connected_weights**(*weights: torch.Tensor, n_filters: int, kernel_size: Union[int, Tuple[int, int]], conv_size: Union[int, Tuple[int, int]], locations: torch.Tensor, input_sqrt: Union[int, Tuple[int, int]], wmin: float = 0.0, wmax: float = 1.0, im: Optional[matplotlib.image.AxesImage] = None, lines: bool = True, figsize: Tuple[int, int] = (5, 5), cmap: str = 'hot_r'*) → matplotlib.image.AxesImage

Plot a connection weight matrix of a `Connection` with 'locally connected structure <http://yann.lecun.com/exdb/publis/pdf/gregor-nips-11.pdf>_.

Parameters

- **weights** – Weight matrix of Conv2dConnection object.

- **n_filters** – No. of convolution kernels in use.
- **kernel_size** – Side length(s) of 2D convolution kernels.
- **conv_size** – Side length(s) of 2D convolution population.
- **locations** – Indices of input receptive fields for convolution population neurons.
- **input_sqrt** – Side length(s) of 2D input data.
- **wmin** – Minimum allowed weight value.
- **wmax** – Maximum allowed weight value.
- **im** – Used for re-drawing the weights plot.
- **lines** – Whether or not to draw horizontal and vertical lines separating input regions.
- **figsize** – Horizontal, vertical figure size in inches.
- **cmap** – Matplotlib colormap.

Returns Used for re-drawing the weights plot.

bindsnet.analysis.plotting.**plot_performance**(*performances: Dict[str, List[float]], ax: Optional[matplotlib.axes._axes.Axes] = None, figsize: Tuple[int, int] = (7, 4)*) → matplotlib.axes._axes.Axes

Plot training accuracy curves.

Parameters

- **performances** – Lists of training accuracy estimates per voting scheme.
- **ax** – Used for re-drawing the performance plot.
- **figsize** – Horizontal, vertical figure size in inches.

Returns Used for re-drawing the performance plot.

bindsnet.analysis.plotting.**plot_spikes**(*spikes: Dict[str, torch.Tensor], time: Optional[Tuple[int, int]] = None, n_neurons: Optional[Dict[str, Tuple[int, int]]] = None, ims: Optional[List[matplotlib.collections.PathCollection]] = None, axes: Union[matplotlib.axes._axes.Axes, List[matplotlib.axes._axes.Axes], None] = None, figsize: Tuple[float, float] = (8.0, 4.5)*) → Tuple[List[matplotlib.image.AxesImage], List[matplotlib.axes._axes.Axes]]

Plot spikes for any group(s) of neurons.

Parameters

- **spikes** – Mapping from layer names to spiking data. Spike data has shape `[time, n_1, ..., n_k]`, where `[n_1, ..., n_k]` is the shape of the recorded layer.
- **time** – Plot spiking activity of neurons in the given time range. Default is entire simulation time.
- **n_neurons** – Plot spiking activity of neurons in the given range of neurons. Default is all neurons.
- **ims** – Used for re-drawing the plots.
- **axes** – Used for re-drawing the plots.
- **figsize** – Horizontal, vertical figure size in inches.

Returns `ims, axes`: Used for re-drawing the plots.

`bindsnet.analysis.plotting.`**`plot_voltages`**(*voltages: Dict[str, torch.Tensor], ims: Optional[List[matplotlib.image.AxesImage]] = None, axes: Optional[List[matplotlib.axes._axes.Axes]] = None, time: Tuple[int, int] = None, n_neurons: Optional[Dict[str, Tuple[int, int]]] = None, cmap: Optional[str] = 'jet', plot_type: str = 'color', thresholds: Dict[str, torch.Tensor] = None, figsize: Tuple[float, float] = (8.0, 4.5)*) → Tuple[List[matplotlib.image.AxesImage], List[matplotlib.axes._axes.Axes]]

Plot voltages for any group(s) of neurons.

Parameters

- **`voltages`** – Contains voltage data by neuron layers.
- **`ims`** – Used for re-drawing the plots.
- **`axes`** – Used for re-drawing the plots.
- **`time`** – Plot voltages of neurons in given time range. Default is entire simulation time.
- **`n_neurons`** – Plot voltages of neurons in given range of neurons. Default is all neurons.
- **`cmap`** – Matplotlib colormap to use.
- **`figsize`** – Horizontal, vertical figure size in inches.
- **`plot_type`** – The way how to draw graph. 'color' for pcolormesh, 'line' for curved lines.
- **`thresholds`** – Thresholds of the neurons in each layer.

Returns `ims, axes`: Used for re-drawing the plots.

`bindsnet.analysis.plotting.`**`plot_weights`**(*weights: torch.Tensor*, *wmin: Optional[float] = 0*, *wmax: Optional[float] = 1*, *im: Optional[matplotlib.image.AxesImage] = None*, *figsize: Tuple[int*, *int] = (5*, *5)*, *cmap: str = 'hot_r'*) → matplotlib.image.AxesImage

Plot a connection weight matrix.

Parameters

- **`weights`** – Weight matrix of `Connection` object.
- **`wmin`** – Minimum allowed weight value.
- **`wmax`** – Maximum allowed weight value.
- **`im`** – Used for re-drawing the weights plot.
- **`figsize`** – Horizontal, vertical figure size in inches.
- **`cmap`** – Matplotlib colormap.

Returns `AxesImage` for re-drawing the weights plot.

bindsnet.analysis.visualization module

bindsnet.analysis.visualization.**plot_spike_trains_for_example**(*spikes: torch.Tensor*, *n_ex: Optional[int] = None*, *top_k: Optional[int] = None*, *indices: Optional[List[int]] = None*) → None

Plot spike trains for top-k neurons or for specific indices.

Parameters

- **spikes** – Spikes for one simulation run of shape `(n_examples, n_neurons, time)`.
- **n_ex** – Allows user to pick which example to plot spikes for.
- **top_k** – Plot k neurons that spiked the most for n_ex example.
- **indices** – Plot specific neurons' spiking activity instead of top_k.

bindsnet.analysis.visualization.**plot_voltage**(*voltage: torch.Tensor*, *n_ex: int = 0*, *n_neuron: int = 0*, *time: Optional[Tuple[int, int]] = None*, *threshold: float = None*) → None

Plot voltage for a single neuron on a specific example.

Parameters

- **voltage** – Tensor or array of shape `[n_examples, n_neurons, time]`.
- **n_ex** – Allows user to pick which example to plot voltage for.
- **n_neuron** – Neuron index for which to plot voltages for.
- **time** – Plot spiking activity of neurons between the given range of time.
- **threshold** – Neuron spiking threshold.

bindsnet.analysis.visualization.**plot_weights_movie**(*ws: numpy.ndarray*, *sample_every: int = 1*) → None

Create and plot movie of weights.

Parameters

- **ws** – Array of shape `[n_examples, source, target, time]`
- **sample_every** – Sub-sample using this parameter.

Module contents

4.1.2 bindsnet.datasets package

Submodules

bindsnet.datasets.preprocess module

bindsnet.datasets.preprocess.**binary_image**(*image: numpy.ndarray*) → numpy.ndarray
Converts input image into black and white (binary)

Parameters **image** – Gray-scaled image.

Returns Black and white image.

bindsnet.datasets.preprocess.**crop**(*image: numpy.ndarray*, *x1: int*, *x2: int*, *y1: int*, *y2: int*) → numpy.ndarray
Crops an image given coordinates of cropping box.

Parameters

- **image** – 3-dimensional image.
- **x1** – Left x coordinate.
- **x2** – Right x coordinate.
- **y1** – Bottom y coordinate.
- **y2** – Top y coordinate.

Returns Image cropped using coordinates (x1, x2, y1, y2).

bindsnet.datasets.preprocess.**gray_scale**(*image: numpy.ndarray*) → numpy.ndarray
Converts RGB image into grayscale.

Parameters **image** – RGB image.

Returns Gray-scaled image.

bindsnet.datasets.preprocess.**subsample**(*image: numpy.ndarray*, *x: int*, *y: int*) → numpy.ndarray
Scale the image to (x, y).

Parameters

- **image** – Image to be rescaled.
- **x** – Output value for `image`'s x dimension.
- **y** – Output value for `image`'s y dimension.

Returns Re-scaled image.

Module contents

4.1.3 bindsnet.encoding package

Module contents

4.1.4 bindsnet.environment package

Module contents

4.1.5 bindsnet.evaluation package

Module contents

4.1.6 bindsnet.learning package

Module contents

4.1.7 bindsnet.network package

Submodules

bindsnet.network.monitors module

class bindsnet.network.monitors.**AbstractMonitor**
Bases: abc.ABC

Abstract base class for state variable monitors.

class bindsnet.network.monitors.**Monitor**(*obj: Union[bindsnet.network.nodes.Nodes, bindsnet.network.topology.AbstractConnection], state_vars: Iterable[str], time: Optional[int] = None, batch_size: int = 1*)
Bases: *bindsnet.network.monitors.AbstractMonitor*

Records state variables of interest.

Constructs a Monitor object.

Parameters

- **obj** – An object to record state variables from during network simulation.
- **state_vars** – Iterable of strings indicating names of state variables to record.
- **time** – If not None, pre-allocate memory for state variable recording.

get(*var: str*) → torch.Tensor
Return recording to user.

Parameters **var** – State variable recording to return.

Returns Tensor of shape [time, n_1, ..., n_k], where [n_1, ..., n_k] is the shape of the recorded state variable.

record() → None
Appends the current value of the recorded state variables to the recording.

reset_() → None
Resets recordings to empty ‘‘torch.Tensor‘‘s.

class bindsnet.network.monitors.**NetworkMonitor**(*network: Network, layers: Optional[Iterable[str]] = None, connections: Optional[Iterable[str]] = None, state_vars: Optional[Iterable[str]] = None, time: Optional[int] = None*)
Bases: *bindsnet.network.monitors.AbstractMonitor*

Record state variables of all layers and connections.

Constructs a NetworkMonitor object.

Parameters

- **network** – Network to record state variables from.

- **layers** – Layers to record state variables from.
- **connections** – Connections to record state variables from.
- **state_vars** – List of strings indicating names of state variables to record.
- **time** – If not `None`, pre-allocate memory for state variable recording.

get() → Dict[str, Dict[str, Union[bindsnet.network.nodes.Nodes, bindsnet.network.topology.AbstractConnection]]]
Return entire recording to user.

Returns Dictionary of dictionary of all layers' and connections' recorded state variables.

record() → None
Appends the current value of the recorded state variables to the recording.

reset_() → None
Resets recordings to empty `torch.Tensors`.

save(*path: str*, *fmt: str = 'npz'*) → None
Write the recording dictionary out to file.

Parameters

- **path** – The directory to which to write the monitor's recording.
- **fmt** – Type of file to write to disk. One of `"pickle"` or `"npz"`.

bindsnet.network.nodes module

class bindsnet.network.nodes.**AbstractInput**
Bases: `abc.ABC`

Abstract base class for groups of input neurons.

class bindsnet.network.nodes.**AdaptiveLIFNodes**(*n: Optional[int] = None*, *shape: Optional[Iterable[int]] = None*, *traces: bool = False*, *traces_additive: bool = False*, *tc_trace: Union[float, torch.Tensor] = 20.0*, *trace_scale: Union[float, torch.Tensor] = 1.0*, *sum_input: bool = False*, *rest: Union[float, torch.Tensor] = -65.0*, *reset: Union[float, torch.Tensor] = -65.0*, *thresh: Union[float, torch.Tensor] = -52.0*, *refrac: Union[int, torch.Tensor] = 5*, *tc_decay: Union[float, torch.Tensor] = 100.0*, *theta_plus: Union[float, torch.Tensor] = 0.05*, *tc_theta_decay: Union[float, torch.Tensor] = 10000000.0*, *lbound: float = None*, ***kwargs*)
Bases: *bindsnet.network.nodes.Nodes*

Layer of leaky integrate-and-fire (LIF) neurons with adaptive thresholds. A neuron's voltage threshold is increased by some constant each time it spikes; otherwise, it is decaying back to its default value.

Instantiates a layer of LIF neurons with adaptive firing thresholds.

Parameters

- **n** – The number of neurons in the layer.
- **shape** – The dimensionality of the layer.

- **traces** – Whether to record spike traces.
- **traces_additive** – Whether to record spike traces additively.
- **tc_trace** – Time constant of spike trace decay.
- **trace_scale** – Scaling factor for spike trace.
- **sum_input** – Whether to sum all inputs.
- **rest** – Resting membrane voltage.
- **reset** – Post-spike reset voltage.
- **thresh** – Spike threshold voltage.
- **refrac** – Refractory (non-firing) period of the neuron.
- **tc_decay** – Time constant of neuron voltage decay.
- **theta_plus** – Voltage increase of threshold after spiking.
- **tc_theta_decay** – Time constant of adaptive threshold decay.
- **lbound** – Lower bound of the voltage.

compute_decays(*dt*) → None
Sets the relevant decays.

forward(*x: torch.Tensor*) → None
Runs a single simulation step.

Parameters **x** – Inputs to the layer.

reset_() → None
Resets relevant state variables.

set_batch_size(*batch_size*) → None
Sets mini-batch size. Called when layer is added to a network.

Parameters **batch_size** – Mini-batch size.

class bindsnet.network.nodes.**CurrentLIFNodes**(*n: Optional[int] = None*, *shape: Optional[Iterable[int]] = None*, *traces: bool = False*, *traces_additive: bool = False*, *tc_trace: Union[float, torch.Tensor] = 20.0*, *trace_scale: Union[float, torch.Tensor] = 1.0*, *sum_input: bool = False*, *thresh: Union[float, torch.Tensor] = -52.0*, *rest: Union[float, torch.Tensor] = -65.0*, *reset: Union[float, torch.Tensor] = -65.0*, *refrac: Union[int, torch.Tensor] = 5*, *tc_decay: Union[float, torch.Tensor] = 100.0*, *tc_i_decay: Union[float, torch.Tensor] = 2.0*, *lbound: float = None*, ***kwargs*)

Bases: *bindsnet.network.nodes.Nodes*

Layer of current-based leaky integrate-and-fire (LIF) neurons. Total synaptic input current is modeled as a decaying memory of input spikes multiplied by synaptic strengths.

Instantiates a layer of synaptic input current-based LIF neurons. :param n: The number of neurons in the layer. :param shape: The dimensionality of the layer. :param traces: Whether to record spike traces. :param traces_additive: Whether to record spike traces additively. :param tc_trace: Time constant of spike trace decay. :param trace_scale: Scaling factor for spike trace. :param sum_input: Whether to sum all inputs. :param thresh: Spike threshold voltage. :param rest: Resting membrane voltage. :param reset: Post-spike reset voltage. :param

refrac: Refractory (non-firing) period of the neuron. :param tc_decay: Time constant of neuron voltage decay. :param tc_i_decay: Time constant of synaptic input current decay. :param lbound: Lower bound of the voltage.

compute_decays(*dt*) → None
Sets the relevant decays.

forward(*x: torch.Tensor*) → None
Runs a single simulation step.

Parameters **x** – Inputs to the layer.

reset_() → None
Resets relevant state variables.

set_batch_size(*batch_size*) → None
Sets mini-batch size. Called when layer is added to a network.

Parameters **batch_size** – Mini-batch size.

class bindsnet.network.nodes.**DiehlAndCookNodes**(*n: Optional[int] = None*, *shape: Optional[Iterable[int]] = None*, *traces: bool = False*, *traces_additive: bool = False*, *tc_trace: Union[float, torch.Tensor] = 20.0*, *trace_scale: Union[float, torch.Tensor] = 1.0*, *sum_input: bool = False*, *thresh: Union[float, torch.Tensor] = -52.0*, *rest: Union[float, torch.Tensor] = -65.0*, *reset: Union[float, torch.Tensor] = -65.0*, *refrac: Union[int, torch.Tensor] = 5*, *tc_decay: Union[float, torch.Tensor] = 100.0*, *theta_plus: Union[float, torch.Tensor] = 0.05*, *tc_theta_decay: Union[float, torch.Tensor] = 10000000.0*, *lbound: float = None*, *one_spike: bool = True*, ***kwargs*)

Bases: *bindsnet.network.nodes.Nodes*

Layer of leaky integrate-and-fire (LIF) neurons with adaptive thresholds (modified for Diehl & Cook 2015 replication).

Instantiates a layer of Diehl & Cook 2015 neurons.

Parameters

- **n** – The number of neurons in the layer.
- **shape** – The dimensionality of the layer.
- **traces** – Whether to record spike traces.
- **traces_additive** – Whether to record spike traces additively.
- **tc_trace** – Time constant of spike trace decay.
- **trace_scale** – Scaling factor for spike trace.
- **sum_input** – Whether to sum all inputs.
- **thresh** – Spike threshold voltage.
- **rest** – Resting membrane voltage.
- **reset** – Post-spike reset voltage.

- **refrac** – Refractory (non-firing) period of the neuron.
- **tc_decay** – Time constant of neuron voltage decay.
- **theta_plus** – Voltage increase of threshold after spiking.
- **tc_theta_decay** – Time constant of adaptive threshold decay.
- **lbound** – Lower bound of the voltage.
- **one_spike** – Whether to allow only one spike per timestep.

compute_decays(*dt*) → None
Sets the relevant decays.

forward(*x: torch.Tensor*) → None
Runs a single simulation step.

Parameters **x** – Inputs to the layer.

reset_() → None
Resets relevant state variables.

set_batch_size(*batch_size*) → None
Sets mini-batch size. Called when layer is added to a network.

Parameters **batch_size** – Mini-batch size.

class bindsnet.network.nodes.**IFNodes**(*n: Optional[int] = None*, *shape: Optional[Iterable[int]] = None*, *traces: bool = False*, *traces_additive: bool = False*, *tc_trace: Union[float, torch.Tensor] = 20.0*, *trace_scale: Union[float, torch.Tensor] = 1.0*, *sum_input: bool = False*, *thresh: Union[float, torch.Tensor] = -52.0*, *reset: Union[float, torch.Tensor] = -65.0*, *refrac: Union[int, torch.Tensor] = 5*, *lbound: float = None*, ***kwargs*)
Bases: *bindsnet.network.nodes.Nodes*

Layer of integrate-and-fire (IF) neurons.

Instantiates a layer of IF neurons.

Parameters

- **n** – The number of neurons in the layer.
- **shape** – The dimensionality of the layer.
- **traces** – Whether to record spike traces.
- **traces_additive** – Whether to record spike traces additively.
- **tc_trace** – Time constant of spike trace decay.
- **trace_scale** – Scaling factor for spike trace.
- **sum_input** – Whether to sum all inputs.
- **thresh** – Spike threshold voltage.
- **reset** – Post-spike reset voltage.
- **refrac** – Refractory (non-firing) period of the neuron.
- **lbound** – Lower bound of the voltage.

forward(*x: torch.Tensor*) → None
Runs a single simulation step.

Parameters **x** – Inputs to the layer.

reset_() → None
Resets relevant state variables.

set_batch_size(*batch_size*) → None
Sets mini-batch size. Called when layer is added to a network.

Parameters **batch_size** – Mini-batch size.

class bindsnet.network.nodes.**Input**(*n: Optional[int] = None*, *shape: Optional[Iterable[int]] = None*, *traces: bool = False*, *traces_additive: bool = False*, *tc_trace: Union[float, torch.Tensor] = 20.0*, *trace_scale: Union[float, torch.Tensor] = 1.0*, *sum_input: bool = False*, ***kwargs*)
Bases: *bindsnet.network.nodes.Nodes*, *bindsnet.network.nodes.AbstractInput*

Layer of nodes with user-specified spiking behavior.

Instantiates a layer of input neurons.

Parameters

- **n** – The number of neurons in the layer.
- **shape** – The dimensionality of the layer.
- **traces** – Whether to record decaying spike traces.
- **traces_additive** – Whether to record spike traces additively.
- **tc_trace** – Time constant of spike trace decay.
- **trace_scale** – Scaling factor for spike trace.
- **sum_input** – Whether to sum all inputs.

forward(*x: torch.Tensor*) → None
On each simulation step, set the spikes of the population equal to the inputs.

Parameters **x** – Inputs to the layer.

reset_() → None
Resets relevant state variables.

class bindsnet.network.nodes.**IzhikevichNodes**(*n: Optional[int] = None*, *shape: Optional[Iterable[int]] = None*, *traces: bool = False*, *traces_additive: bool = False*, *tc_trace: Union[float, torch.Tensor] = 20.0*, *trace_scale: Union[float, torch.Tensor] = 1.0*, *sum_input: bool = False*, *excitatory: float = 1*, *thresh: Union[float, torch.Tensor] = 45.0*, *rest: Union[float, torch.Tensor] = -65.0*, *lbound: float = None*, ***kwargs*)
Bases: *bindsnet.network.nodes.Nodes*

Layer of Izhikevich neurons.

Instantiates a layer of Izhikevich neurons.

Parameters

- **n** – The number of neurons in the layer.
- **shape** – The dimensionality of the layer.

- **traces** – Whether to record spike traces.
- **traces_additive** – Whether to record spike traces additively.
- **tc_trace** – Time constant of spike trace decay.
- **trace_scale** – Scaling factor for spike trace.
- **sum_input** – Whether to sum all inputs.
- **excitatory** – Percent of excitatory (vs. inhibitory) neurons in the layer; in range `[0, 1]`.
- **thresh** – Spike threshold voltage.
- **rest** – Resting membrane voltage.
- **lbound** – Lower bound of the voltage.

forward(*x: torch.Tensor*) → None
Runs a single simulation step.

Parameters **x** – Inputs to the layer.

reset_() → None
Resets relevant state variables.

set_batch_size(*batch_size*) → None
Sets mini-batch size. Called when layer is added to a network.

Parameters **batch_size** – Mini-batch size.

class bindsnet.network.nodes.**LIFNodes**(*n: Optional[int] = None*, *shape: Optional[Iterable[int]] = None*, *traces: bool = False*, *traces_additive: bool = False*, *tc_trace: Union[float, torch.Tensor] = 20.0*, *trace_scale: Union[float, torch.Tensor] = 1.0*, *sum_input: bool = False*, *thresh: Union[float, torch.Tensor] = -52.0*, *rest: Union[float, torch.Tensor] = -65.0*, *reset: Union[float, torch.Tensor] = -65.0*, *refrac: Union[int, torch.Tensor] = 5*, *tc_decay: Union[float, torch.Tensor] = 100.0*, *lbound: float = None*, ***kwargs*)

Bases: *bindsnet.network.nodes.Nodes*

Layer of leaky integrate-and-fire (LIF) neurons.

Instantiates a layer of LIF neurons.

Parameters

- **n** – The number of neurons in the layer.
- **shape** – The dimensionality of the layer.
- **traces** – Whether to record spike traces.
- **traces_additive** – Whether to record spike traces additively.
- **tc_trace** – Time constant of spike trace decay.
- **trace_scale** – Scaling factor for spike trace.
- **sum_input** – Whether to sum all inputs.
- **thresh** – Spike threshold voltage.

- **rest** – Resting membrane voltage.
- **reset** – Post-spike reset voltage.
- **refrac** – Refractory (non-firing) period of the neuron.
- **tc_decay** – Time constant of neuron voltage decay.
- **lbound** – Lower bound of the voltage.

compute_decays(*dt*) → None
Sets the relevant decays.

forward(*x: torch.Tensor*) → None
Runs a single simulation step.

Parameters **x** – Inputs to the layer.

reset_() → None
Resets relevant state variables.

set_batch_size(*batch_size*) → None
Sets mini-batch size. Called when layer is added to a network.

Parameters **batch_size** – Mini-batch size.

class bindsnet.network.nodes.**McCullochPitts**(*n: Optional[int] = None*, *shape: Optional[Iterable[int]] = None*, *traces: bool = False*, *traces_additive: bool = False*, *tc_trace: Union[float, torch.Tensor] = 20.0*, *trace_scale: Union[float, torch.Tensor] = 1.0*, *sum_input: bool = False*, *thresh: Union[float, torch.Tensor] = 1.0*, ***kwargs*)

Bases: *bindsnet.network.nodes.Nodes*

Layer of McCulloch-Pitts neurons.

Instantiates a McCulloch-Pitts layer of neurons.

Parameters

- **n** – The number of neurons in the layer.
- **shape** – The dimensionality of the layer.
- **traces** – Whether to record spike traces.
- **traces_additive** – Whether to record spike traces additively.
- **tc_trace** – Time constant of spike trace decay.
- **trace_scale** – Scaling factor for spike trace.
- **sum_input** – Whether to sum all inputs.
- **thresh** – Spike threshold voltage.

forward(*x: torch.Tensor*) → None
Runs a single simulation step.

Parameters **x** – Inputs to the layer.

reset_() → None
Resets relevant state variables.

set_batch_size(*batch_size*) → None
Sets mini-batch size. Called when layer is added to a network.

Parameters **batch_size** – Mini-batch size.

class bindsnet.network.nodes.**Nodes**(*n: Optional[int] = None*, *shape: Optional[Iterable[int]] = None*, *traces: bool = False*, *traces_additive: bool = False*, *tc_trace: Union[float, torch.Tensor] = 20.0*, *trace_scale: Union[float, torch.Tensor] = 1.0*, *sum_input: bool = False*, *learning: bool = True*, ***kwargs*)

Bases: torch.nn.modules.module.Module

Abstract base class for groups of neurons.

Abstract base class constructor.

Parameters

- **n** – The number of neurons in the layer.
- **shape** – The dimensionality of the layer.
- **traces** – Whether to record decaying spike traces.
- **traces_additive** – Whether to record spike traces additively.
- **tc_trace** – Time constant of spike trace decay.
- **trace_scale** – Scaling factor for spike trace.
- **sum_input** – Whether to sum all inputs.
- **learning** – Whether to be in learning or testing.

compute_decays(*dt*) → None
Abstract base class method for setting decays.

forward(*x: torch.Tensor*) → None
Abstract base class method for a single simulation step.

Parameters **x** – Inputs to the layer.

reset_() → None
Abstract base class method for resetting state variables.

set_batch_size(*batch_size*) → None
Sets mini-batch size. Called when layer is added to a network.

Parameters **batch_size** – Mini-batch size.

train(*mode: bool = True*) → bindsnet.network.nodes.Nodes
Sets the layer in training mode.

Parameters **mode** (*bool*) – Turn training on or off

Returns self as specified in *torch.nn.Module*

class bindsnet.network.nodes.**RealInput**(*n: Optional[int] = None*, *shape: Optional[Iterable[int]] = None*, *traces: bool = False*, *traces_additive: bool = False*, *tc_trace: Union[float, torch.Tensor] = 20.0*, *trace_scale: Union[float, torch.Tensor] = 1.0*, *sum_input: bool = False*, ***kwargs*)

Bases: *bindsnet.network.nodes.Nodes*, *bindsnet.network.nodes.AbstractInput*

Layer of nodes with user-specified real-valued outputs.

Instantiates a layer of input neurons.

Parameters

- **n** – The number of neurons in the layer.
- **shape** – The dimensionality of the layer.
- **traces** – Whether to record decaying spike traces.
- **traces_additive** – Whether to record spike traces additively.
- **tc_trace** – Time constant of spike trace decay.
- **trace_scale** – Scaling factor for spike trace.
- **sum_input** – Whether to sum all inputs.

forward(*x: torch.Tensor*) → None
On each simulation step, set the outputs of the population equal to the inputs.

Parameters **x** – Inputs to the layer.

reset_() → None
Resets relevant state variables.

class bindsnet.network.nodes.**SRM0Nodes**(*n: Optional[int] = None*, *shape: Optional[Iterable[int]] = None*, *traces: bool = False*, *traces_additive: bool = False*, *tc_trace: Union[float, torch.Tensor] = 20.0*, *trace_scale: Union[float, torch.Tensor] = 1.0*, *sum_input: bool = False*, *thresh: Union[float, torch.Tensor] = -50.0*, *rest: Union[float, torch.Tensor] = -70.0*, *reset: Union[float, torch.Tensor] = -70.0*, *refrac: Union[int, torch.Tensor] = 5*, *tc_decay: Union[float, torch.Tensor] = 10.0*, *lbound: float = None*, *eps_0: Union[float, torch.Tensor] = 1.0*, *rho_0: Union[float, torch.Tensor] = 1.0*, *d_thresh: Union[float, torch.Tensor] = 5.0*, ***kwargs*)

Bases: *bindsnet.network.nodes.Nodes*

Layer of simplified spike response model (SRM0) neurons with stochastic threshold (escape noise). Adapted from (Vasilaki et al., 2009).

Instantiates a layer of SRM0 neurons.

Parameters

- **n** – The number of neurons in the layer.
- **shape** – The dimensionality of the layer.
- **traces** – Whether to record spike traces.
- **traces_additive** – Whether to record spike traces additively.
- **tc_trace** – Time constant of spike trace decay.
- **trace_scale** – Scaling factor for spike trace.
- **sum_input** – Whether to sum all inputs.
- **thresh** – Spike threshold voltage.
- **rest** – Resting membrane voltage.
- **reset** – Post-spike reset voltage.
- **refrac** – Refractory (non-firing) period of the neuron.
- **tc_decay** – Time constant of neuron voltage decay.

- **lbound** – Lower bound of the voltage.
- **eps_0** – Scaling factor for pre-synaptic spike contributions.
- **rho_0** – Stochastic intensity at threshold.
- **d_thresh** – Width of the threshold region.

compute_decays(*dt*) → None
Sets the relevant decays.

forward(*x: torch.Tensor*) → None
Runs a single simulation step.

Parameters **x** – Inputs to the layer.

reset_() → None
Resets relevant state variables.

set_batch_size(*batch_size*) → None
Sets mini-batch size. Called when layer is added to a network.

Parameters **batch_size** – Mini-batch size.

bindsnet.network.topology module

class bindsnet.network.topology.**AbstractConnection**(*source: bindsnet.network.nodes.Nodes, target: bindsnet.network.nodes.Nodes, nu: Union[float, Sequence[float], None] = None, reduction: Optional[callable] = None, weight_decay: float = 0.0, **kwargs*)

Bases: abc.ABC, torch.nn.modules.module.Module

Abstract base method for connections between Nodes.

Constructor for abstract base class for connection objects.

Parameters

- **source** – A layer of nodes from which the connection originates.
- **target** – A layer of nodes to which the connection connects.
- **nu** – Learning rate for both pre- and post-synaptic events.
- **reduction** – Method for reducing parameter updates along the minibatch dimension.
- **weight_decay** – Constant multiple to decay weights by on each iteration.

Keyword arguments:

Parameters

- **update_rule** (*LearningRule*) – Modifies connection parameters according to some rule.
- **wmin** (*float*) – The minimum value on the connection weights.
- **wmax** (*float*) – The maximum value on the connection weights.
- **norm** (*float*) – Total weight per target neuron normalization.

- **norm_by_max** (*ByteTensor*) – Normalize the weight of a neuron by its max weight.
- **norm_by_max_with_shadow_weights** (*ByteTensor*) – Normalize the weight of a neuron by its max weight by original weights

compute(*s: torch.Tensor*) → None
Compute pre-activations of downstream neurons given spikes of upstream neurons.

Parameters **s** – Incoming spikes.

reset_() → None
Contains resetting logic for the connection.

update(***kwargs*) → None
Compute connection's update rule.

Keyword arguments:

Parameters

- **learning** (*bool*) – Whether to allow connection updates.
- **mask** (*ByteTensor*) – Boolean mask determining which weights to clamp to zero.

class bindsnet.network.topology.**Connection**(*source: bindsnet.network.nodes.Nodes, target: bindsnet.network.nodes.Nodes, nu: Union[float, Sequence[float], None] = None, reduction: Optional[callable] = None, weight_decay: float = 0.0, **kwargs*)
Bases: *bindsnet.network.topology.AbstractConnection*

Specifies synapses between one or two populations of neurons.

Instantiates a Connection object.

Parameters

- **source** – A layer of nodes from which the connection originates.
- **target** – A layer of nodes to which the connection connects.
- **nu** – Learning rate for both pre- and post-synaptic events.
- **reduction** – Method for reducing parameter updates along the minibatch dimension.
- **weight_decay** – Constant multiple to decay weights by on each iteration.

Keyword arguments:

Parameters

- **update_rule** (*LearningRule*) – Modifies connection parameters according to some rule.
- **w** (*torch.Tensor*) – Strengths of synapses.
- **b** (*torch.Tensor*) – Target population bias.
- **wmin** (*float*) – Minimum allowed value on the connection weights.
- **wmax** (*float*) – Maximum allowed value on the connection weights.
- **norm** (*float*) – Total weight per target neuron normalization constant.
- **norm_by_max** (*ByteTensor*) – Normalize the weight of a neuron by its max weight.
- **norm_by_max_with_shadow_weights** (*ByteTensor*) – Normalize the weight of a neuron by its max weight by original weights.

compute(*s: torch.Tensor*) → torch.Tensor
Compute pre-activations given spikes using connection weights.

Parameters **s** – Incoming spikes.

Returns Incoming spikes multiplied by synaptic weights (with or without decaying spike activation).

normalize() → None
Normalize weights so each target neuron has sum of connection weights equal to `self.norm`.

normalize_by_max() → None
Normalize weights by the max weight of the target neuron.

normalize_by_max_from_shadow_weights() → None
Normalize weights by the max weight of the target neuron.

reset_() → None
Contains resetting logic for the connection.

update(***kwargs*) → None
Compute connection's update rule.

class bindsnet.network.topology.**Conv2dConnection**(*source: bindsnet.network.nodes.Nodes, target: bindsnet.network.nodes.Nodes, kernel_size: Union[int, Tuple[int, int]], stride: Union[int, Tuple[int, int]] = 1, padding: Union[int, Tuple[int, int]] = 0, dilation: Union[int, Tuple[int, int]] = 1, nu: Union[float, Sequence[float], None] = None, reduction: Optional[callable] = None, weight_decay: float = 0.0, **kwargs*)
Bases: *bindsnet.network.topology.AbstractConnection*

Specifies convolutional synapses between one or two populations of neurons.

Instantiates a `Conv2dConnection` object.

Parameters

- **source** – A layer of nodes from which the connection originates.
- **target** – A layer of nodes to which the connection connects.
- **kernel_size** – Horizontal and vertical size of convolutional kernels.
- **stride** – Horizontal and vertical stride for convolution.
- **padding** – Horizontal and vertical padding for convolution.
- **dilation** – Horizontal and vertical dilation for convolution.
- **nu** – Learning rate for both pre- and post-synaptic events.
- **reduction** – Method for reducing parameter updates along the minibatch dimension.
- **weight_decay** – Constant multiple to decay weights by on each iteration.

Keyword arguments:

Parameters

- **update_rule** (*LearningRule*) – Modifies connection parameters according to some rule.
- **w** (*torch.Tensor*) – Strengths of synapses.
- **b** (*torch.Tensor*) – Target population bias.
- **wmin** (*float*) – Minimum allowed value on the connection weights.
- **wmax** (*float*) – Maximum allowed value on the connection weights.
- **norm** (*float*) – Total weight per target neuron normalization constant.

compute(*s: torch.Tensor*) → torch.Tensor
Compute convolutional pre-activations given spikes using layer weights.

Parameters **s** – Incoming spikes.

Returns Incoming spikes multiplied by synaptic weights (with or without decaying spike activation).

normalize() → None
Normalize weights along the first axis according to total weight per target neuron.

reset_() → None
Contains resetting logic for the connection.

update(***kwargs*) → None
Compute connection's update rule.

class bindsnet.network.topology.**LocalConnection**(*source: bindsnet.network.nodes.Nodes, target: bindsnet.network.nodes.Nodes, kernel_size: Union[int, Tuple[int, int]], stride: Union[int, Tuple[int, int]], n_filters: int, nu: Union[float, Sequence[float], None] = None, reduction: Optional[callable] = None, weight_decay: float = 0.0, **kwargs*)
Bases: *bindsnet.network.topology.AbstractConnection*

Specifies a locally connected connection between one or two populations of neurons.

Instantiates a `LocalConnection` object. Source population should be two-dimensional.

Neurons in the post-synaptic population are ordered by receptive field; that is, if there are `n_conv` neurons in each post-synaptic patch, then the first `n_conv` neurons in the post-synaptic population correspond to the first receptive field, the second `n_conv` to the second receptive field, and so on.

Parameters

- **source** – A layer of nodes from which the connection originates.
- **target** – A layer of nodes to which the connection connects.
- **kernel_size** – Horizontal and vertical size of convolutional kernels.
- **stride** – Horizontal and vertical stride for convolution.
- **n_filters** – Number of locally connected filters per pre-synaptic region.
- **nu** – Learning rate for both pre- and post-synaptic events.
- **reduction** – Method for reducing parameter updates along the minibatch dimension.
- **weight_decay** – Constant multiple to decay weights by on each iteration.

Keyword arguments:

Parameters

- **update_rule** (*LearningRule*) – Modifies connection parameters according to some rule.
- **w** (*torch.Tensor*) – Strengths of synapses.
- **b** (*torch.Tensor*) – Target population bias.
- **wmin** (*float*) – Minimum allowed value on the connection weights.
- **wmax** (*float*) – Maximum allowed value on the connection weights.
- **norm** (*float*) – Total weight per target neuron normalization constant.
- **int] input_shape** (*Tuple[int,*) – Shape of input population if it's not `[sqrt, sqrt]`.

compute(*s: torch.Tensor*) → torch.Tensor
Compute pre-activations given spikes using layer weights.

Parameters **s** – Incoming spikes.

Returns Incoming spikes multiplied by synaptic weights (with or without decaying spike activation).

normalize() → None
Normalize weights so each target neuron has sum of connection weights equal to `self.norm`.

reset_() → None
Contains resetting logic for the connection.

update(***kwargs*) → None
Compute connection's update rule.

Keyword arguments:

Parameters **mask** (*ByteTensor*) – Boolean mask determining which weights to clamp to zero.

class bindsnet.network.topology.**MaxPool2dConnection**(*source: bindsnet.network.nodes.Nodes, target: bindsnet.network.nodes.Nodes, kernel_size: Union[int, Tuple[int, int]], stride: Union[int, Tuple[int, int]] = 1, padding: Union[int, Tuple[int, int]] = 0, dilation: Union[int, Tuple[int, int]] = 1, **kwargs*)
Bases: *bindsnet.network.topology.AbstractConnection*

Specifies max-pooling synapses between one or two populations of neurons by keeping online estimates of maximally firing neurons.

Instantiates a `MaxPool2dConnection` object.

Parameters

- **source** – A layer of nodes from which the connection originates.
- **target** – A layer of nodes to which the connection connects.
- **kernel_size** – Horizontal and vertical size of convolutional kernels.

- **stride** – Horizontal and vertical stride for convolution.
- **padding** – Horizontal and vertical padding for convolution.
- **dilation** – Horizontal and vertical dilation for convolution.

Keyword arguments:

Parameters **decay** – Decay rate of online estimates of average firing activity.

compute(*s: torch.Tensor*) → torch.Tensor
Compute max-pool pre-activations given spikes using online firing rate estimates.

Parameters **s** – Incoming spikes.

Returns Incoming spikes multiplied by synaptic weights (with or without decaying spike activation).

normalize() → None
No weights -> no normalization.

reset_() → None
Contains resetting logic for the connection.

update(***kwargs*) → None
Compute connection's update rule.

class bindsnet.network.topology.**MeanFieldConnection**(*source: bindsnet.network.nodes.Nodes, target: bindsnet.network.nodes.Nodes, nu: Union[float, Sequence[float], None] = None, weight_decay: float = 0.0, **kwargs*)
Bases: *bindsnet.network.topology.AbstractConnection*

A connection between one or two populations of neurons which computes a summary of the pre-synaptic population to use as weighted input to the post-synaptic population.

Instantiates a `MeanFieldConnection` object. :param source: A layer of nodes from which the connection originates. :param target: A layer of nodes to which the connection connects. :param nu: Learning rate for both pre- and post-synaptic events. :param weight_decay: Constant multiple to decay weights by on each iteration. Keyword arguments: :param LearningRule update_rule: Modifies connection parameters according to some rule. :param torch.Tensor w: Strengths of synapses. :param float wmin: Minimum allowed value on the connection weights. :param float wmax: Maximum allowed value on the connection weights. :param float norm: Total weight per target neuron normalization constant.

compute(*s: torch.Tensor*) → torch.Tensor
Compute pre-activations given spikes using layer weights. :param s: Incoming spikes. :return: Incoming spikes multiplied by synaptic weights (with or without decaying spike activation).

normalize() → None
Normalize weights so each target neuron has sum of connection weights equal to `self.norm`.

reset_() → None
Contains resetting logic for the connection.

update(***kwargs*) → None
Compute connection's update rule.

class bindsnet.network.topology.**SparseConnection**(*source: bindsnet.network.nodes.Nodes, target: bindsnet.network.nodes.Nodes, nu: Union[float, Sequence[float], None] = None, reduction: Optional[callable] = None, weight_decay: float = None, **kwargs*)

Bases: *bindsnet.network.topology.AbstractConnection*

Specifies sparse synapses between one or two populations of neurons.

Instantiates a `Connection` object with sparse weights.

Parameters

- **source** – A layer of nodes from which the connection originates.
- **target** – A layer of nodes to which the connection connects.
- **nu** – Learning rate for both pre- and post-synaptic events.
- **reduction** – Method for reducing parameter updates along the minibatch dimension.
- **weight_decay** – Constant multiple to decay weights by on each iteration.

Keyword arguments:

Parameters

- **w** (*torch.Tensor*) – Strengths of synapses.
- **sparsity** (*float*) – Fraction of sparse connections to use.
- **update_rule** (*LearningRule*) – Modifies connection parameters according to some rule.
- **wmin** (*float*) – Minimum allowed value on the connection weights.
- **wmax** (*float*) – Maximum allowed value on the connection weights.
- **norm** (*float*) – Total weight per target neuron normalization constant.

compute(*s: torch.Tensor*) → torch.Tensor

Compute convolutional pre-activations given spikes using layer weights.

Parameters **s** – Incoming spikes.

Returns Incoming spikes multiplied by synaptic weights (with or without decaying spike activation).

normalize() → None

Normalize weights along the first axis according to total weight per target neuron.

reset_() → None

Contains resetting logic for the connection.

update(***kwargs*) → None

Compute connection's update rule.

Module contents

4.1.8 bindsnet.pipeline package

4.2 Module contents

CHAPTER 5

Indices and tables

- genindex
- search

Python Module Index

b

Index

G

I

L

M

N

P

R

S

T

U