SpykeTorch Documentation

Milad Mozafari

Feb 10, 2019

CONTENTS:

High-speed simulator of convolutional spiking neural networks with at most one spike per neuron.

CHAPTER

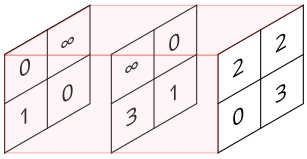
ONE

INTRODUCTION

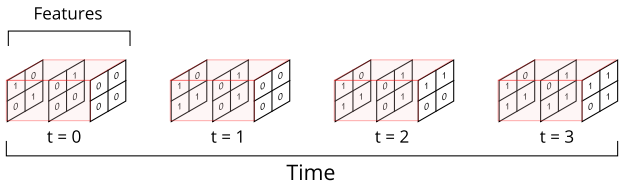
SpykeTorch is a Pytorch-based simulator of convolutional spiking neural networks (SNNs), in which the neurons emit at most one spike per stimulus. Computations with single-spike neurons can be efficiently reduced to multiple tensor operations. Thanks to the versatile Pytorch library, SpykeTorch is fast in tensor operations and easily executable on GPUs by adding a few lines of code.

SpykeTorch's modules are inherited from and compatible with Pytorch's modules. However, since the simulation of SNNs needs the concept of time, their input and output tensors include a new dimension called "time". You may not need to think about this new dimensionality while you are using SpykeTorch. But, in order to combine it with other Pytorch functionalities or extracting different kinds of information from SNNs, it is important to have a good grasp of how SpykeTorch deals with time dimension.

To illustrate the concept of time, assume that there are three 2x2 grids of neurons (three different feature maps, 12 neurons in total) that are going to represent each stimulus. Also assume that each stimulus is presented in at most 4 time steps. Let's consider a particular stimulus for which the spike times of these 12 neurons are:



where ∞ stands for no spike. The corresponding spike-wave tensor for this stimulus will be in the following structure:



As shown in the figure, if a neuron emits spike at time step t, the corresponding position in spike-wave tensor will be set to 1 from time step t to the final time step. This way of keeping the spike data helps us to compute the potentials of all the neurons in all of the time steps in a signle call of convolution function and results in a huge performance improvement. Accordingly, the information about potentials over different time steps will be kept in 4-dimensional tensors. Note that in SpykeTorch's modules and functionalities such as `Convolution` or `Pooling`, the mini-batch dimension is sacrificed for the time dimension. Therefore, we do not provide built-in batch processing at the moment.

CHAPTER

TWO

SPYKETORCH

2.1 SpykeTorch package

2.1.1 SpykeTorch.functional module

SpykeTorch.functional.**feature_inhibition**(*potentials*, *inhibited_features*)
Inhibits specified features (reset the corresponding neurons' potentials to zero).

Parameters

- **potentials** (*Tensor*) – The tensor of input potentials.
- **inhibited_features** (*List*) – The list of features to be inhibited.

Returns Inhibited potentials.

Return type Tensor

SpykeTorch.functional.**feature_inhibition_**(*potentials*, *inhibited_features*)
The inplace version of *feature_inhibition()*

SpykeTorch.functional.**fire**(*potentials*, *threshold=None*, *return_thresholded_potentials=False*)
Computes the spike-wave tensor from tensor of potentials. If *threshold* is None, all the neurons emit one spike (if the potential is greater than zero) in the last time step.

Parameters

- **potentials** (*Tensor*) – The tensor of input potentials.
- **threshold** (*float*) – Firing threshold. Default: None
- **return_thresholded_potentials** (*boolean*) – If True, the tensor of thresholded potentials will be returned
- **well as the tensor of spike-wave. Default** (*as*) – False

Returns Spike-wave tensor.

Return type Tensor

SpykeTorch.functional.**fire_**(*potentials*, *threshold=None*)
The inplace version of *fire()*

SpykeTorch.functional.**get_k_winners**(*potentials*, *kwta=1*, *inhibition_radius=0*, *spikes=None*)
Finds at most kwta winners first based on the earliest spike time, then based on the maximum potential. It returns a list of winners, each in a tuple of form (feature, row, column).

Note: Winners are selected sequentially. Each winner inhibits surrounding neruons in a specific radius in all of the other feature maps. Note that only one winner can be selected from each feature map.

Parameters

- **potentials** (*Tensor*) – The tensor of input potentials.
- **kwta** (*int, optional*) – The number of winners. Default: 1
- **inhibition_radius** (*int, optional*) – The radius of lateral inhibition. Default: 0
- **spikes** (*Tensor, optional*) – Spike-wave corresponding to the input potentials. Default: None

Returns List of winners.

Return type List

SpykeTorch.functional.**intensity_lateral_inhibition**(*intencities*, *inhibition_kernel*)

Applies lateral inhibition on intensities. For each location, this inhibition decreases the intensity of the surrounding cells that has lower intensities by a specific factor. This factor is relative to the distance of the neighbors and are put in the `inhibition_kernel`.

Parameters

- **intencities** (*Tensor*) – The tensor of input intensities.
- **inhibition_kernel** (*Tensor*) – The tensor of inhibition factors.

Returns Inhibited intensities.

Return type Tensor

SpykeTorch.functional.**local_normalization**(*input*, *normalization_radius*, *eps=1e-12*)

Applies local normalization. on each region (of size radius*2 + 1) the mean value is computed and the intensities will be divided by the mean value. The input is a 4D tensor.

Parameters

- **input** (*Tensor*) – The input tensor of shape (timesteps, features, height, width).
- **normalization_radius** (*int*) – The radius of normalization window.

Returns Locally normalized tensor.

Return type Tensor

SpykeTorch.functional.**pad**(*input*, *pad*, *value=0*)

Applies 2D padding on the input tensor.

Parameters

- **input** (*Tensor*) – The input tensor.
- **pad** (*tuple*) – A tuple of 4 integers in the form of (padLeft, padRight, padTop, padBottom)
- **value** (*int or float*) – The value of padding. Default: 0

Returns Padded tensor.

Return type Tensor

SpykeTorch.functional.**pointwise_inhibition**(*thresholded_potentials*)
Performs point-wise inhibition between feature maps. After inhibition, at most one neuron is allowed to fire at each position, which is the neuron with the earliest spike time. If the spike times are the same, the neuron with the maximum potential will be chosen. As a result, the potential of all of the inhibited neurons will be reset to zero.

Parameters **thresholded_potentials** (*Tensor*) – The tensor of thresholded input potentials.

Returns Inhibited potentials.

Return type Tensor

SpykeTorch.functional.**pooling**(*input*, *kernel_size*, *stride=None*, *padding=0*)
Performs a 2D max-pooling over an input signal (spike-wave or potentials) composed of several input planes.

Parameters

- **input** (*Tensor*) – The input tensor.
- **kernel_size** (*int or tuple*) – Size of the pooling window.
- **stride** (*int or tuple, optional*) – Stride of the pooling window. Default: None
- **padding** (*int or tuple, optional*) – Size of the padding. Default: 0

Returns The result of the max-pooling operation.

Return type Tensor

SpykeTorch.functional.**threshold**(*potentials*, *threshold=None*)
Applies a threshold on potentials by which all of the values lower or equal to the threshold becomes zero. If *threshold* is None, only the potentials corresponding to the final time step will survive.

Parameters

- **potentials** (*Tensor*) – The tensor of input potentials.
- **threshold** (*float*) – The threshold value. Default: None

Returns Thresholded potentials.

Return type Tensor

SpykeTorch.functional.**threshold_**(*potentials*, *threshold=None*)
The inplace version of *threshold()*

2.1.2 SpykeTorch.snn module

class SpykeTorch.snn.**Convolution**(*in_channels*, *out_channels*, *kernel_size*, *weight_mean=0.8*, *weight_std=0.02*)
Bases: torch.nn.modules.module.Module

Performs a 2D convolution over an input spike-wave composed of several input planes. Current version only supports stride of 1 with no padding.

The input is a 4D tensor with the size $(T, C_{in}, H_{in}, W_{in})$ and the crresponsing output is of size $(T, C_{out}, H_{out}, W_{out})$, where T is the number of time steps, C is the number of feature maps (channels), and H, and W are the hight and width of the input/output planes.

- in_channels controls the number of input planes (channels/feature maps).
- out_channels controls the number of feature maps in the current layer.

- `kernel_size` controls the size of the convolution kernel. It can be a single integer or a tuple of two integers.
- `weight_mean` controls the mean of the normal distribution used for initial random weights.
- `weight_std` controls the standard deviation of the normal distribution used for initial random weights.

Note: Since this version of convolution does not support padding, it is the user responsibility to add proper padding on the input before applying convolution.

Parameters

- **in_channels** (*int*) – Number of channels in the input.
- **out_channels** (*int*) – Number of channels produced by the convolution.
- **kernel_size** (*int or tuple*) – Size of the convolving kernel.
- **weight_mean** (*float, optional*) – Mean of the initial random weights. Default: 0.8
- **weight_std** (*float, optional*) – Standard deviation of the initial random weights. Default: 0.02

reset_weight(*weight_mean=0.8*, *weight_std=0.02*)

Resets weights to random values based on a normal distribution.

Parameters

- **weight_mean** (*float, optional*) – Mean of the random weights. Default: 0.8
- **weight_std** (*float, optional*) – Standard deviation of the random weights. Default: 0.02

class SpykeTorch.snn.**Pooling**(*kernel_size*, *stride=None*, *padding=0*)

Bases: `torch.nn.modules.module.Module`

Performs a 2D max-pooling over an input signal (spike-wave or potentials) composed of several input planes.

Note: Regarding the structure of the spike-wave tensors, application of max-pooling over spike-wave tensors results in propagation of the earliest spike within each pooling window.

The input is a 4D tensor with the size (T, C, H_{in}, W_{in}) and the crresponsing output is of size (T, C, H_{out}, W_{out}), where T is the number of time steps, C is the number of feature maps (channels), and H, and W are the hight and width of the input/output planes.

- `kernel_size` controls the size of the pooling window. It can be a single integer or a tuple of two integers.
- `stride` controls the stride of the pooling. It can be a single integer or a tuple of two integers. If the value is None, it does pooling with full stride.
- `padding` controls the amount of padding. It can be a single integer or a tuple of two integers.

Parameters

- **kernel_size** (*int or tuple*) – Size of the pooling window
- **stride** (*int or tuple, optional*) – Stride of the pooling window. Default: None

- **padding** (*int or tuple, optional*) – Size of the padding. Default: 0

class SpykeTorch.snn.**STDP**(*conv_layer*, *learning_rate*, *use_stabilizer=True*, *lower_bound=0*, *upper_bound=1*)

Bases: torch.nn.modules.module.Module

Performs STDP learning rule over synapses of a convolutional layer based on the following formulation:

$$\Delta W_{ij} = \begin{cases} a_{LTP} \times (W_{ij} - W_{LB}) \times (W_{UP} - W_{ij}) & t_j - t_i \leq 0, \\ a_{LTD} \times (W_{ij} - W_{LB}) \times (W_{UP} - W_{ij}) & t_j - t_i > 0, \end{cases}$$

where i and j refer to the post- and pre-synaptic neurons, respectively, Δw_{ij} is the amount of weight change for the synapse connecting the two neurons, and a_{LTP}, and a_{LTD} scale the magnitude of weight change. Besides, $(W_{ij} - W_{LB}) \times (W_{UP} - W_{ij})$ is a stabilizer term which slowes down the weight change when the synaptic weight is close to the weight's lower (W_{LB}) and upper (W_{UB}) bounds.

To create a STDP object, you need to provide:

- `conv_layer`: The convolutional layer on which the STDP should be applied.
- `learning_rate`: (a_{LTP}, a_{LTD}) rates. A single pair of floats or a list of pairs of floats. Each feature map has its own learning rates.
- `use_stabilizer`: Turns the stabilizer term on or off.
- `lower_bound` and `upper_bound`: Control the range of weights.

To apply STDP for a particular stimulus, you need to provide:

- `input_spikes` and `potentials` that are the input spike-wave and corresponding potentials, respectively.
- `output_spikes` that is the output spike-wave.
- `winners` or `kwta` to find winners based on the earliest spike then the maximum potential.
- `inhibition_radius` to inhibit surrounding neurons (in all feature maps) within a particular radius.

Parameters

- **conv_layer** (snn.Convolution) – Reference convolutional layer.
- **learning_rate** (*tuple of python:floats or list of tuples of python:floats*) – (LTP, LTD) rates for STDP.
- **use_stabilizer** (*boolean, optional*) – Turning stabilizer term on or off. Default: True
- **lower_bound** (*float, optional*) – Lower bound of the weight range. Default: 0
- **upper_bound** (*float, optional*) – Upper bound of the weight range. Default: 1

get_pre_post_ordering(*input_spikes*, *output_spikes*, *winners*)

Computes the ordering of the input and output spikes with respect to the position of each winner and returns them as a list of boolean tensors. True for pre-then-post (or concurrency) and False for post-then-pre. Input and output tensors must be spike-waves.

Parameters

- **input_spikes** (*Tensor*) – Input spike-wave
- **output_spikes** (*Tensor*) – Output spike-wave

- **winners** (*List of Tuples*) – List of winners. Each tuple denotes a winner in a form of a triplet (feature, row, column).

Returns pre-post ordering of spikes

Return type List

update_all_learning_rate(*ap*, *an*)
Updates learning rates of all the feature maps to a same value.

Parameters

- **ap** (*float*) – LTP rate.
- **an** (*float*) – LTD rate.

update_learning_rate(*feature*, *ap*, *an*)
Updates learning rate for a specific feature map.

Parameters **feature** (*int*) – The target feature. ap (float): LTP rate. an (float): LTD rate.

2.1.3 SpykeTorch.utils module

class SpykeTorch.utils.**CacheDataset**(*dataset*, *cache_address=None*)
Bases: torch.utils.data.dataset.Dataset

A wrapper dataset to cache pre-processed data. It can cache data on RAM or a secondary memory.

Note: Since converting image into spike-wave can be time consuming, we recommend to wrap your dataset into a *CacheDataset* object.

Parameters

- **dataset** (*torch.utils.data.Dataset*) – The reference dataset object.
- **cache_address** (*str, optional*) – The location of cache in the secondary memory. Use None to cache on RAM. Default: None

reset_cache()
Clears the cached data. It is useful when you want to change a pre-processing parameter during the training process.

class SpykeTorch.utils.**DoGKernel**(*window_size*, *sigma1*, *sigma2*)
Bases: *SpykeTorch.utils.FilterKernel*

Generates DoG filter kernel.

Parameters

- **window_size** (*int*) – The size of the window (square window).
- **sigma1** (*float*) – The sigma for the first Gaussian function.
- **sigma2** (*float*) – The sigma for the second Gaussian function.

class SpykeTorch.utils.**Filter**(*filter_kernels*, *padding=0*, *thresholds=None*, *use_abs=False*)
Bases: object

Applies a filter transform. Each filter contains a sequence of *FilterKernel* objects. The result of each filter kernel will be passed through a given threshold (if not None).

Parameters

- **filter_kernels** (*sequence of FilterKernels*) – The sequence of filter kernels.
- **padding** (*int, optional*) – The size of the padding for the convolution of filter kernels. Default: 0
- **thresholds** (*sequence of python:floats, optional*) – The threshold for each filter kernel. Default: None
- **use_abs** (*boolean, optional*) – To compute the absolute value of the outputs or not. Default: False

---

Note: The size of the compund filter kernel tensor (stack of individual filter kernels) will be equal to the greatest window size among kernels. All other smaller kernels will be zero-padded with an appropriate amount.

---

class SpykeTorch.utils.**FilterKernel**(*window_size*)
Bases: object

Base class for generating image filter kernels such as Gabor, DoG, etc. Each subclass should override `__call__` function.

class SpykeTorch.utils.**GaborKernel**(*window_size*, *orientation*, *div=4.0*)
Bases: *SpykeTorch.utils.FilterKernel*

Generates Gabor filter kernel.

Parameters

- **window_size** (*int*) – The size of the window (square window).
- **orientation** (*float*) – The orientation of the Gabor filter (in degrees).
- **div** (*float, optional*) – The divisor of the lambda equation. Default: 4.0

class SpykeTorch.utils.**Intensity2Latency**(*number_of_spike_bins*, *to_spike=False*)
Bases: object

Applies intensity to latency transform. Spike waves are generated in the form of spike bins with almost equal number of spikes.

Parameters

- **number_of_spike_bins** (*int*) – Number of spike bins (time steps).
- **to_spike** (*boolean, optional*) – To generate spike-wave tensor or not. Default: False

---

Note: If `to_spike` is `False`, then the result is intesities that are ordered and packed into bins.

---

intensity_to_latency(*intencities*)

class SpykeTorch.utils.**LateralIntencityInhibition**(*inhibition_percents*)
Bases: object

Applies lateral inhibition on intensities. For each location, this inhibition decreases the intensity of the surrounding cells that has lower intensities by a specific factor. This factor is relative to the distance of the neighbors and are put in the `inhibition_percents`.

Parameters **inhibition_percents** (*sequence*) – The sequence of inhibition factors (in range [0,1]).

intensity_lateral_inhibition(*intencities*)

SpykeTorch.utils.**generate_inhibition_kernel**(*inhibition_percents*)
Generates an inhibition kernel suitable to be used by `intensity_lateral_inhibition()`.

Parameters **inhibition_percents** (*sequence*) – The sequence of inhibition factors (in range [0,1]).

Returns Inhibition kernel.

Return type Tensor

SpykeTorch.utils.**to_pair**(*data*)
Converts a single or a tuple of data into a pair. If the data is a tuple with more than two elements, it selects the first two of them. In case of single data, it duplicates that data into a pair.

Parameters **data** (*object or tuple*) – The input data.

Returns A pair of data.

Return type Tuple

2.1.4 SpykeTorch.visualization module

SpykeTorch.visualization.**get_deep_feature**(*pre_feature*, *feature_stride*, *window_size*, *stride*, *weights=None*)

SpykeTorch.visualization.**get_deep_receptive_field**(**layers_details*)

SpykeTorch.visualization.**plot_tensor_in_image**(*fname*, *aTensor*, *_vmin=None*, *_vmax=None*)
Plots a 2D tensor in gray color map in an image file.

Parameters

- **fname** (*str*) – The file name.
- **aTensor** (*Tensor*) – The input tensor.
- **_vmin** (*float, optional*) – Minimum value. Default: None
- **_vmax** (*float, optional*) – Maximum value. Default: None

---

Note: `None` for `_vmin` or `_vmin` causes an auto-scale mode for each.

SpykeTorch.visualization.**show_tensor**(*aTensor*, *_vmin=None*, *_vmax=None*)
Plots a 2D tensor in gray color map and shows it in a window.

Parameters

- **aTensor** (*Tensor*) – The input tensor.
- **_vmin** (*float, optional*) – Minimum value. Default: None
- **_vmax** (*float, optional*) – Maximum value. Default: None

---

Note: `None` for `_vmin` or `_vmin` causes an auto-scale mode for each.

PYTHON MODULE INDEX

S