Toolbox for Building Deep Spiking ConvNets - Documentation

Release 1.0

Sven Gronauer

Feb 28, 2018

CONTENTS:

CHAPTER

ONE

INTRODUCTION

That has a paragraph about a main subject and is set when the '=' is at least the same length of the title itself.

1.1 Subtitle

Subtitles are set with '-' and are required to have the same length of the subtitle itself, just like titles.

Lists can be unnumbered like:

- Item Foo
- Item Bar

Or automatically numbered:

1. Item 1
2. Item 2

1.2 Subtitle

Words can have *emphasis in italics* ors be **bold** and you can define code samples with back quotes, like when you talk about a command: `sudo` gives you super user powers!

Build the documentation:

1. change directory `cd docs/`
2. render `/programming/test_sphynx/docs$ sphinx-apidoc -f -o source/ ../SpikingConvNet/`
3. excute `make html`

CHAPTER

TWO

SAMPLE CODE

This section describes how to setup a simple Deep Spiking Convolutional Neural Network (DSCNN).

2.1 Simple 1-Layer ConvNet

Let's start with training a simple SCNN with one convolutional Layer. By creating firstly the model structure with the following python code:

```
model = SpikingModel(input_tensor=(28,28,1), run_control=rc)
model.add(ConvLayer(4,shape=(5,5), stride=2))
model.add(Classifier())
```

In order to build the network structure on SpiNNaker Hardware, you have to execute commands in the terminal:

```
$python main.py --mode loaddata
$python main.py --mode training --layer 1
$python main.py --mode training --layer svm
$python main.py --mode testing
```

2.2 Deeper ConvNet

Theoritically, as many layers as appreciated can be build. Therefore convolutional layers are added to the model are added in sequential manner.

```
model = SpikingModel(input_tensor=(28,28,1), run_control=rc)
model.add(ConvLayer(4,shape=(5,5), stride=2))
model.add(ConvLayer(4,shape=(5,5), stride=2))
...
model.add(ConvLayer(4,shape=(3,3), stride=2))
model.add(Classifier())
```

```
$python main.py --mode loaddata
$python main.py --mode training --layer 1
$python main.py --mode training --layer 2
...
$python main.py --mode training --layer n
$python main.py --mode training --layer svm
$python main.py --mode testing
```

Note: The training of the Network is done layer by layer, hence the input spikes of the currently trained layer depend on the previous layer. So a new simulation cycle is started the previously calculated layers are flattend to achieve parallel computation.

CHAPTER THREE

SPIKINGCONVNET

3.1 SpikingConvNet package

3.1.1 Submodules

3.1.2 SpikingConvNet.algorithms module

Deep Spiking Convolutional Neural Network with STDP Learning Rule on MNIST data ____ Research Internship Technical University Munich Creator: Sven Gronauer Date: February 2018

`SpikingConvNet.algorithms.`**`input_flattend_spikes`**(*X_train*, *tensor_input*, *kernel_shape*)

Create flattend SpikeSourceArray for input neurons instance: rebuilding network

For rebuilding the network structure the input neurons are not windowed over time, instead the input layer is flattend and a whole image is presented to the network in each simulation interval The corresponding spiketimes depend on pixel intensities and are stochastically rate-coded.

Parameters

- **X_train** (*np.array, shape = [n_examples, image_intensities.flatten()]*) – dataset of MNIST input images as 2d-array
- **tensor_input** (*tuple of int*) – Dimensions of input layer
- **kernel_shape** (*tuple of int*) – Kernel shape

Returns **spiketrains** – Each datapoint contains precise spike times for each input neuron

Return type np.array, shape = [n_examples, spike_times]

`SpikingConvNet.algorithms.`**`input_windowed_spikes`**(*X_train*, *tensor_input*, *kernel_shape*, *stride*)

Create windowed SpikeSourceArray for input neurons instance: training layer in network

Input patterns are windowed over time, post-neurons are presented only a subset of the input pattern. The corresponding spiketimes depend on pixel intensities and are stochastically rate-coded.

Parameters **X_train** (*np.array, shape = [n_examples, image_intensities.flatten()]*)

Returns **spiketrains** – Each datapoint contains precise spike times for each input neuron

Return type np.array, shape = [n_examples*n_windows, spike_times]

SpikingConvNet.algorithms.**rebuild_fixed_connections**(*tensor_first*, *tensor_second*, *kernel_shape*, *stride*, *weights_tensor*)

Construct connections between flattend layers use learned STDP weights and fix them now computation in parallel

Parameters

- **tensor_first** (*tuple of int*) – Dimensions of previous layer
- **tensor_second** (*tuple of int*) – Dimensions of posterior layer
- **kernel_shape** (*tuple of int*) – Kernel shape
- **stride** (*int*) – Specified stride over convolved layer
- **weights_tensor** (*np.array, shape=[n_kernel, kernel_height*kernel_width]*) – Previously trained STDP weights, now initialized as fixed weights

Returns **connection_list** – list for s.FromListConnector()

Return type list of [position_1, position_2, weight , delay]

SpikingConvNet.algorithms.**rebuild_inhibitory_connections**(*tensor_prev*, *tensor_layer*, *inhib_weight*)

construct inhibitory connections within flattend layers

-> not used in final implementation !

Parameters

- **tensor_prev** (*tuple of int*) – Dimensions of previous layer
- **tensor_layer** (*tuple of int*) – Dimensions of actual layer
- **inhib_weight** (*float32*) – fixed weight for inhibitory connection

Returns **inhib_connection_list** – list for s.FromListConnector()

Return type list of [position_1, position_2, weight , delay]

SpikingConvNet.algorithms.**spikes_for_classifier**(*rc*, *tensor*, *spiketrains*)

Transform spiketrains to plain two-dimensional dataset

to reduce the power of Support Vector Machine, the quantity of spikes within each simulation interval are counted for each neuron in the last layer

Parameters

- **tensor** (*tuple of int*) – Tensor of last Convolutional layer in network
- **spiketrains** (*SpikeTrain object*) – Retrieved spikes from last layer on SpiNNaker board SpikeTrains objects are extracted from Neo Block Segments

Returns **X** – Each datapoint contains number of spikes for each post-neuron within one sim interval

Return type np.array, shape = [datapoints, n_neurons_last_layer]

SpikingConvNet.algorithms.**windowed_spikes**(*spiketrains_input*, *tensor_first*, *tensor_second*, *kernel_shape*, *stride*)

Create windowed SpikeSourceArray instance: training deeper layer in network

Input spiketrains are windowed over time, post-neurons are presented only a subset of the input spiketrains. The corresponding output spiketimes depend on calculated spikes (spiketrains_input) of previous layer.

Parameters

- **spiketrains_input** (*SpikeTrain object*) – Spiketrains from previous layer

- **tensor_first** (*tuple of int*) – Dimensions of previous layer
- **tensor_second** (*tuple of int*) – Dimensions of posterior layer
- **kernel_shape** (*tuple of int*) – Kernel shape
- **stride** (*int*) – Specified stride over convolved layer

Returns **spiketrains** – Each datapoint contains precise spike times for each neuron

Return type np.array, shape = [n_examples*windows, spike_times]

3.1.3 SpikingConvNet.classes module

Deep Spiking Convolutional Neural Network with STDP Learning Rule on MNIST data
____ Research Internship Technical University Munich Creator: Sven Gronauer Date: February 2018

class `SpikingConvNet.classes.`**`Classifier`**
Bases: `SpikingConvNet.classes.Layer`

`classify`(*X_test*, *y_test*)
determine classification accuracy of SVC with given Testset

`train`(*X_train*, *y_train*)
train parameters with given Trainset

class `SpikingConvNet.classes.`**`ConvLayer`**(*kernels*, *shape*, *stride*)
Bases: `SpikingConvNet.classes.Layer`

class `SpikingConvNet.classes.`**`InputLayer`**(*tensor*)
Bases: `SpikingConvNet.classes.Layer`

class `SpikingConvNet.classes.`**`Layer`**(*rc*)
Bases: `object`

class `SpikingConvNet.classes.`**`SpikingModel`**(*input_tensor*, *run_control*)
Bases: `object`

`add`(*layer*)

`calculate_tensors`()

`print_structure`()

class `SpikingConvNet.classes.`**`Spinnaker_Network`**(*runcontrol*, *model*, *deepspikes=None*)
Class for implementing neural network on SpiNNaker

The following steps are processed through calling the class constructor (almost the same as PyNN basic setup structure) #. Initialize with constructor #. load datasets (Train and Testset) from files #. Load previously calculated weights for layers #. Create populations #. Build STDP-model #. Build projections between populations #. Setup recordings

These methods must be called from external fuction(s): * update_kernel_weights() - Determine current weights in STDP trained layer * retrieve_data() - Receive observed data from SpiNNaker * print_parameters() - Display Information of Neural Network

Parameters

- **runcotrol** (*RunControl object*) – Structure that contains basic information for program flow such as passed args from terminal command, backup commands, building options for SpiNNaker network
- **model** (*SpikingModel object*) – predefined model of spiking neural network

- **deepspikes** (*Spiketrain object*) – training a deeper layer requires preprocessed spikes from previous layer, hence training of Spiking Neural Network is done layer by layer

print_parameters()

retrieve_data()

Transmit observed data of spikes and voltages from SpiNNaker Board to host computer

Returns

- **spiketrains** (*SpikeTrain object*) – Spiketrains from last layer in neural network
- **list** (*[spikes_in, spikes_1, v_1]*) – *spikes_in: spiketimes input layer *spikes_1: spikes post-neurons *v_1: membrane potentials of post-neurons

update_kernel_weights()

Update the internal stored weights of trained layer with STDP Rule

returns current STDP weight values of trained Layer

3.1.4 SpikingConvNet.parameters module

Deep Spiking Convolutional Neural Network with STDP Learning Rule on MNIST data ____ Research Internship Technical University Munich Creator: Sven Gronauer Date: February 2018

3.1.5 SpikingConvNet.utils module

Utilities for controling program flow, data handling and data plotting

class SpikingConvNet.utils.**RunControl**(*args*, *trainlayer=0*, *trainsvm=False*, *rebuild=False*)

Bases: `object`

Object for controlling program flow, contains args from console and sets up the logging utility

Parameters

- **args** (*ArgumentParser object*) – Passed arguments from terminal command
- **trainlayer** (*int, optional*) – If not zero, specifies which layer of network to train
- **trainsvm** (*bool, optional*) – If given, classifier is trained
- **rebuild** (*bool, optional*) – Controls the behaviour of the follow up build of neural network (as a variable of programs state machine)
  - `rebuild==True` in order to train layer n, the spikes of layer n-1 must be determined
  - `rebuild==False` a layer or the Classifier is trained

setup_logger()

Setup logger for tracking infos and errors

SpikingConvNet.utils.**convert_rate_code**(*intensity*, *total_intensity=None*)

Rate Coding of input pixel

calculate spike times depended on pixel intensity of pre-neuron

Args: intensity: pixel intensity [0, 255] total_intensity: Sum of intensities of input pattern,

used for normalization

`SpikingConvNet.utils.`**`convert_time_code`**`(`*intensity*`)`
assign pixel intensity to time intervall

`SpikingConvNet.utils.`**`dog_filter`**`(`*image*`)`
Apply Difference of Gaussian Filter to image

Parameters **image** (*np.array, shape=[height, width]*) – Image to be transformed

Returns **norm_dog** – Transformed image

Return type np.array, shape=[height, width]

`SpikingConvNet.utils.`**`load_MNIST_digits_ordered`**`(`*args*`)`
Load MNIST dataset in chronological order

`SpikingConvNet.utils.`**`load_MNIST_digits_shuffled`**`(`*rc*, *mode*`)`
Load MNIST dataset

load defined number of examples (see parameters.py) loaded subset of digits is defined in SUBSET_DIGITS

shuffle data and return as 2d-arrays

`SpikingConvNet.utils.`**`plot_confusion_matrix`**`(`*rc*, *cm*, *normalize=True*, *title='Confusion_matrix'*, *cmap=<matplotlib.colors.LinearSegmentedColormap object>*`)`
This function prints and plots the confusion matrix. Normalization can be applied by setting *normalize=True*.

`SpikingConvNet.utils.`**`plot_heatpmap`**`(`*rc*, *list_of_elements*, *title='Default Title'*, *delta=False*`)`
plot matrix of heatmaps

Delta if True - plot differential images

`SpikingConvNet.utils.`**`plot_membran_voltages`**`(`*rc*, *v_data*, *simtime*, *title='Membrane potentials'*, *path=None*`)`

`SpikingConvNet.utils.`**`plot_spike_activity`**`(`*rc*, *spiketrains*, *tensor*, *title='plot_spike_activity'*`)`

`SpikingConvNet.utils.`**`plot_spikes`**`(`*rc*, *pre*, *post=None*, *title='Spikes Plot'*, *path=None*`)`
Plot spikes of given layers

Parameters

- **rc** (*RunControl object*) – contains information of backup behaviour
- **pre** (*SpikeTrain object*) – Spiketrains of first layer to plot

Returns **norm_dog** – Transformed image

Return type np.array, shape=[height, width]

3.1.6 Module contents

Package for building Spiking Deep Convolutional Neural Networks on SpiNNaker

- Neuroscientific System Theory
- Technical University Munich
- Creator: Sven Gronauer
- Date: February 2018

The modules inside of this package are packed with useful features for the programmer who needs to build convolutional networks on SpiNNaker:

`classes`

> This module provides classes for creating objects of the neural network model and the necessary infracture for building networks on SpiNNaker

`algorithms`

> Here, algorithms are provided for generating sparse connections between populations with the support of convolutions and kernels.

`utils`

> Supporting functions to plot data, manipulate images, load MNIST dataset and convert spike coding scheme.

PYTHON MODULE INDEX

S

INDEX