Turing Neural Networks Documentation

Release 0.1

Giovanni Sirio Carmantini

Feb 21, 2022

CONTENTS

CHAPTER

ONE

LICENSE

The MIT License (MIT)

Copyright (c) <2015> <Giovanni Sirio Carmantini>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER

TWO

INTRODUCTION

This files documents the code used in the paper [A modular architecture for transparent computation in Recurrent Neural Networks](https://arxiv.org/pdf/1609.01926.pdf), implementing the presented mapping. In the example included with the code it is possible to observe how to create a Neural Network simulating the computation of any arbitrary Turing Machine given the Machine description.

For the most up-to-date version of the code and documentation please visit the git repository at https://github.com/TuringApproved/Turing_Neural_Networks.

For installation instructions, follow the README.txt included with the code.

CHAPTER

THREE

SYMDYN SUBMODULE DOCUMENTATION

The symdyn submodule contains various classes implementing objects in representation theory, such as Gödel encoders and Generalized Shifts, needed to construct Nonlinear Dynamical Automata [Moore91].

3.1 Code

tnnpy.symdyn.**as_list**(*arg*)
Convenience function used to make sure that a sequence is always represented by a list of symbols. Sometimes a single-symbol sequence is passed in the form of a string representing the symbol. This has to be converted to a one-item list for consistency, as a list of symbols is what many of the following functions expect.

class tnnpy.symdyn.**FractalEncoder**
Abstract class implementing a Fractal Encoder (generalizing the Godel Encoders).

class tnnpy.symdyn.**GodelEncoder**(*alphabet*)
Create a Godel Encoding given an alphabet. Given the encoding, you can then encode strings as simple numbers or cylinder sets (defined by their upper and lower bounds)

Parameters **alphabet** – a list of symbols. The encoder will first associate one number to each symbol. In particular the symbol in alphabet[0] will be associated with number 0, and so on.

encode_sequence(*sequence*)
Return Godel encoding of a sequence (passed as a list of strings, each one representing a symbol, or a string in case of a single symbol).

encode_cylinder(*sequence*, *rescale=False*)
Return Godel encoding of cylinder set of a sequence (passed as a list of strings, each one representing a symbol, or a string in case of a single symbol).

If rescale=True the values are no longer bound to $[0, 1]$, and the most significant digit of the returned values in base g, where g is the number of symbols in the alphabet, is equal to the Godel encoding of the first symbol in the string.

class tnnpy.symdyn.**CompactGodelEncoder**(*ge_q*, *ge_s*)
Create Godel Encoder as described in our submitted paper.

Parameters

- **ge_q** – A Fractal Encoder for the simulated TM states.
- **ge_s** – A Fractal Encoder for the simulated TM tape symbols.

encode_sequence(*sequence*)
Return Godel encoding of a sequence (passed as a list of strings, each one representing a symbol, or a string in case of a single symbol).

encode_cylinder(*sequence*)
Return Godel encoding of cylinder set of a sequence (passed as a list of strings, each one representing a symbol, or a string in case of a single symbol).

class tnnpy.symdyn.**AbstractGeneralizedShift**(*alpha_dod*, *beta_dod*)
The general class.

class tnnpy.symdyn.**SimpleCFGeneralizedShift**(*alpha_symbols*, *beta_symbols*, *grammar_rules*)
A simple parser to reproduce example NDA computation in beim Graben, P., & Potthast, R. (2014). Universal neural field computation. In Neural Fields (pp. 299-318).

Parameters

- **alpha_symbols** – list of symbols in the stack alphabet
- **beta_symbols** – list of symbols in the input alphabet
- **grammar_rules** – a dictionary where each entry has the the form "X: Y", where X is a string, and Y is a list of strings. Each entry corresponds to a rule in a Context Free Grammar, with X being a Variable

  and Y being a list of Variables and/or Terminals.

psi(*alpha*, *beta*)
Apply Generalized Shift phi to dotted sequence.

predict(*alpha*, *beta*)
If a rule is present defining how to substitute the symbol in the alpha DoD, return the new sequence. Otherwise return False (so that psi can decide to apply attach instead).

attach(*alpha*, *beta*)
If the symbols in the alpha DoD and the beta DoD are equal, pop them and return the new subsequences. Otherwise just return the original.

lintransf_params(*ge_alpha*, *ge_beta*, *alpha*, *beta*)
Return two arrays containing respectively the x parameters and the y parameters of the linear transformation representing the GS action on the symbologram.

Parameters

- **ge_alpha** – Fractal Encoder for left side of dotted sequence.
- **ge_beta** – Fractal Encoder for right side of dotted sequence.
- **alpha** – reversed left side of dotted sequence (as list of symbols or string representing one symbol).
- **beta** – right side of dotted sequence (as list of symbols or string representing one symbol).

Returns $[\lambda_x, a_x]$, $[\lambda_y, a_y]$

Return type tuple(numpy.ndarray, numpy.ndarray)

class tnnpy.symdyn.**TMGeneralizedShift**(*states*, *tape_symbols*, *moves*)
Class implementing a Generalized Shift simulating a Turing Machine. For the definition of Generalized Shift, and the characteristics of GS simulating Turing Machines, see Moore (1991).

Parameters

- **states** – list of states.
- **tape_symbols** – list of tape symbols.
- **moves** – a dict of the form (state, symbol): (new state, new symbol, movement)`, where movement is equal to either "L", "S" or "R", that is Left, Stay or Right.

psi(*alpha*, *beta*)
Given the right and the inverted left part of a dotted sequence, apply the generalized shift on them.

Parameters

- **alpha** – the inverted left part of a dotted sequence, as list of symbols.
- **beta** – the right part of a dotted sequence, as list of symbols.

F(*alpha*, *beta*)
Return direction of shift given simulated TM rule to apply on the basis of the symbols in the DoD of the dotted sequence.

substitution(*alpha*, *beta*)
Return new dotted sequence from substitution.

Substitute symbols in the DoE of the dotted sequence based on the relevant TM symbol substitution rule, given the symbols in the dotted sequence DoD.

shift(*shift_dir*, *alpha*, *beta*)
Shift dotted sequence left or right (shift_dir = -1 or 1).

lintransf_params(*ge_alpha:* tnnpy.symdyn.CompactGodelEncoder, *ge_beta:* tnnpy.symdyn.GodelEncoder, *alpha: List[str]*, *beta: List[str]*)
Return two arrays containing respectively the x parameters and the y parameters of the linear transformation representing the GS action on the symbologram given a dotted sequence.

Parameters

- **ge_alpha** – Fractal Encoder for left side of dotted sequence.
- **ge_beta** – Fractal Encoder for right side of dotted sequence.
- **alpha** – reversed left side of dotted sequence (as list of symbols).
- **beta** – right side of dotted sequence (as list of symbols).

class tnnpy.symdyn.**NonlinearDynamicalAutomaton**(*generalized_shift:* tnnpy.symdyn.AbstractGeneralizedShift, *godel_enc_alpha: Union[*tnnpy.symdyn.GodelEncoder, tnnpy.symdyn.CompactGodelEncoder*]*, *godel_enc_beta:* tnnpy.symdyn.GodelEncoder)
A Nonlinear Dynamical Automaton from a Generalized Shift and a Godel Encoding.

Parameters

- **generalized_shift** – an AbstractGeneralizedShift object (as a base class)
- **godel_enc_alpha** – a GodelEncoder for the α' reversed one-side infinite subsequence of the dotted sequence $\alpha.\beta$ representing a configuration of the Turing Machine to be simulated.
- **godel_enc_beta** – a GodelEncoder for the β one-side infinite subsequence of dotted sequence $\alpha.\beta$ representing a configuration

  of the Turing Machine to be simulated.

check_cell(*x: float*, *y: float*, *gencoded: bool = False*)
Return the coordinates i,j of the input on the unit square partition.

If gencoded=True, the input is assumed to be already encoded.

find_flow_parameters()
Convert the generalized shift dynamics in dynamics on the plane, finding the parameters of the linear transformation for each NDA cell.

`flow`(*x: float*, *y: float*)

Given (x_t, y_t) **return** $\Psi(x_t, y_t) = (x_{t+1}, y_{t+1})$

`iterate`(*init_x: float*, *init_y: float*, *n_iterations: int*)

Apply $\Psi^n(x_0, y_0)$**,** where x_0 = init_x, y_0 = init_y, and n = n_iterations

CHAPTER

FOUR

SIMPLENNLIB SUBMODULE DOCUMENTATION

The simpleNNlib submodule contains classes to build Neural Networks with saturated-linear and Heaviside activation functions.

4.1 Code

class tnnpy.simpleNNlib.**AbstractNNLayer**(*n_units: int*, *initial_values=None*)

The base class for all neuron layers. Each layer has to implement the possibility to add connections from other layers, to compute the input to each neuron by summing all contribution from it connections, and to compute the output given the input and some activation function.

In particular, derived classes will override the activation funcion with their own. For example, a ramp layer will use a ramp activation function, a heaviside layer will use a heaviside function, etc…

Parameters

- **n_units** – the number of units in the layer
- **initial_values** – a list of activation values with which the units will be initialized

sum_input() → numpy.ndarray

Computes the input contribution from connected layers.

activate()

Computes the units activation.

class tnnpy.simpleNNlib.**HeavisideLayer**(*n_units*, *centers*, *inclusive=None*)

Class implementing a layer of neurons with Heaviside activation function.

Parameters

- **n_units** – number of units in layer.
- **centers** – specifies c_i for each unit i , given the unit activation function $H_i(x - c_i)$. Equivalent to adding a bias of $-c_i$.
- **inclusive** – if $H_i(-c_i) = 1$, the i-th item in the list is True, otherwise it's False and $H_i(-c_i) = 0$.

activate()

Computes the units activation

class tnnpy.simpleNNlib.**RampLayer**(*n_units*, *biases=0*, *initial_values=None*)

Class implementing a layer of neurons with saturated linear activation function.

Parameters

- **n_units** – number of units in layer.

- **biases** – a list specifying the bias for each neuron.
- **initial_values** – initial activation values

activate()
Computes the units activation.

class tnnpy.simpleNNlib.**LayerSlice**(*layer*, *a_slice*)
This class is used so that to access the activation for a given layer *nnlayer*, you can just use the notation *nnlayer[someslice]*, where *someslice* is the usual Python slice.

class tnnpy.simpleNNlib.**Connection**(*from_layer*, *to_layer*, *connection_matrix*)
Class implementing a connection. Basically used to keep track of who is connected to who at this moment. Also, you can modify the connection by modifying conn_mat.

Parameters

- **from_layer** – layer from which the connection is established.
- **to_layer** – layer to which the connection is established.
- **connection_matrix** – matrix specifying connections and weights.

CHAPTER

FIVE

NEURALTM SUBMODULE DOCUMENTATION

The submodule NeuralTM contains classes to build an NDA Neural Network simulating a Turing Machine in real time.

5.1 Code

class tnnpy.neuraltm.**NeuralTM**(*nda:* tnnpy.symdyn.NonlinearDynamicalAutomaton, *cylinder_sets: bool = False*)

Given an NDA, constructs the equivalent neural network described in our submitted paper.

Parameters **nda** – a symdyn.NonlinearDynamicalAutomaton object

cn_BSLbx_LTL()

Generate the connection matrix between the x cell selection layer and the linear transformation layer. The connection pattern is discussed in our submitted paper.

Returns connection matrix between x cell selection layer and

linear transformation layer

Return type numpy ndarray

cn_BSLby_LTL()

Generates the connection matrix between the y cell selection layer and the linear transformation layer. The connection pattern is discussed in our submitted paper.

Returns connection matrix between y cell selection layer and

linear transformation layer

Return type numpy ndarray

cn_MCLx_LTL()

Generates the connection matrix between the x input neurons and the linear transformation layer. These are the connections implementing the linear transformations of the NDA piecewise linear system. The multiplication constants are implemented as multiplicative weights in the connection matrix, whereas the additive constants are implemented as biases to the linear transformation layer, outside this function.

Returns connection matrix between x input neurons and linear transformation layer

Return type numpy ndarray

cn_MCLy_LTL()

Generates the connection matrix between the y input neurons and the linear transformation layer. These are the connections implementing the linear transformations of the NDA piecewise linear system. The multiplication constants are implemented as multiplicative weights in the connection matrix, whereas the additive constants are implemented as biases to the linear transformation layer, outside this function.

Returns connection matrix between y input neurons and linear

transformation layer

Return type numpy ndarray

`LTL_biases_from_params()`
Generates the biases for the neurons in the linear transformation layer, which implement the additive constants of the linear transfomations of the NDA piecewise linear system.

Returns biases for linear transformation layer neurons

Return type numpy ndarray

`cn_LTL_MCLx()`

Returns connection matrix between linear transformation layer and

x input neurons.

Return type numpy ndarray

`cn_LTL_MCLy()`

Returns connection matrix between linear transformation layer and

x input neurons.

Return type numpy ndarray

`run_net(`*init_x: Optional[numpy.ndarray] = None*, *init_y: Optional[numpy.ndarray] = None*, *n_iterations=1*`)`
Run the network, given the initial values for the x and y input neurons and the number of iterations

Parameters `init_x` – initial activation values for left and right x

input neurons

Parameters `init_y` – initial activation values for left and right y

input neurons

Parameters `n_iterations` – the desired number of network iterations

Returns list containing the input activation for each iteration. Each tuple item in the list contains two arrays, respectively containing the activations for the x and the y input neurons. `[ (np.array([x_l_act0, x_r_act0]), np.array([y_l_act0, y_r_act0])), ... ]`

Return type list

CHAPTER

SIX

PLOTTING SUBMODULE DOCUMENTATION

The plotting submodule contains utility functions for plotting.

6.1 Code

tnnpy.plotting.**plot_symbologram**(*ax: matplotlib.axes._axes.Axes*, *alpha_symbols: List[str]*, *beta_symbols: List[str]*, *ge_alpha:* tnnpy.symdyn.GodelEncoder, *ge_beta: Union[*tnnpy.symdyn.GodelEncoder, tnnpy.symdyn.CompactGodelEncoder*]*, *TM: bool = True*)

Plot symbologram

tnnpy.plotting.**plot_cylinders**(*ax: matplotlib.axes._axes.Axes*, *cylinders_2d: List[Tuple[numpy.ndarray, numpy.ndarray]]*)

Plot cylinder sets on ax.

CHAPTER

SEVEN

UTILS SUBMODULE DOCUMENTATION

The utility submodule contains generic utility functions.

7.1 Code

tnnpy.utils.**get_invariant_partition_1d**(*alphabet: List[str]*, *sequence: List[str]*) → List[numpy.ndarray]

Return Godel-encoded cylinders for sequence, applying all possible enumerations for alphabet symbols in encoding.

Parameters

- **alphabet** – A list of symbols comprising an alphabet
- **sequence** – A list of symbols from the alphabet

Returns encoded cylinders for sequence when applying Godel encodings with all possible enumerations, with each element of returned list being of the form np.ndarray([seq_left_bound, seq_right_bound])

tnnpy.utils.**get_invariant_partition_2d**(*alpha_alphabet: List[str]*, *beta_alphabet: List[str]*, *alpha_sequence: List[str]*, *beta_sequence: List[str]*) → List[Tuple[numpy.ndarray, numpy.ndarray]]

Return Godel-encoded cylinder sets for dotted sequence, applying all possible enumerations for symbols in alpha and beta alphabets.

Parameters

- **alpha_alphabet** – A list of symbols comprising the alphabet for the left part of the dotted sequence
- **beta_alphabet** – A list of symbols comprising the alphabet for the right part of the dotted sequence
- **alpha_sequence** – left part of the dotted sequence (right-to-left from the dot)
- **beta_sequence** – right part of the dotted sequence

Returns

encoded cylinder sets for dotted sequence when applying Godel encodings with all possible enumerations, with each element in the returned list being of the form: `np.ndarray([alpha_seq_left_bound, alpha_seq_right_bound]),

np.ndarray([beta_seq_left_bound, beta_seq_right_bound])`

BIBLIOGRAPHY

[Moore91] Moore, C. (1991). Generalized shifts: unpredictability and undecidability in dynamical systems. Nonlinearity, 4(2), 199.

PYTHON MODULE INDEX

t

INDEX

R

S

T