DORA

Release 0.2.3

Alex Doumas, Ivan Vegner, Miles Rose

Apr 01, 2025

CONTENTS:

Add your content using `reStructuredText` syntax. See the reStructuredText documentation for details.

CHAPTER

ONE

NODES PACKAGE

1.1 Subpackages

1.1.1 nodes.nodeTests package

Submodules

nodes.nodeTests.test_1 module

nodes.nodeTests.test_1.**test_nodes_from_file**()

Module contents

1.2 Submodules

1.3 nodes.nodeBuilder module

class nodes.nodeBuilder.**Build_children**(*set:* Set, *tokens:* Token_set, *sems:* Sem_set, *symProps: list[dict]*)

Bases: `object`

A class for building the list of children for each token.

set

The set of the tokens.

Type
Set

tokens

The token set.

Type
Token_set

sems

The semantic set.

Type
Sem_set

symProps

A list of symProps relating to the set.

Type
list

get_children()

Recursively add child nodes IDs to each token objects children list.

get_object(*name*)

Return token object if it exists. Else return None

Returns
The token object. None: If the token does not exist.

Return type
token (*Token*)

get_po_children(*name*, *sems: list*)

Step three in recursively adding child nodes IDs to each token objects children list.

get_prop_children(*prop: dict*)

Step one in recursively adding child nodes IDs to each token objects children list.

get_rb_children(*rb: dict*)

Step two in recursively adding child nodes IDs to each token objects children list.

class nodes.nodeBuilder.**Build_connections**(*token_sets: dict[*Set, Token_set*]*, *sems:* Sem_set)

Bases: object

A class for building the links and connections for each set.

token_sets

A dictionary of token sets, mapping set to token set.

Type
dict

sems

The semantic set object.

Type
Sem_set

build_connections_links()

Build the connections and links for each set.

build_set_connections(*token_set:* Token_set)

Build the connections matrix for a given set.

Returns
The NxN connections matrix for the set.

Return type
connections (np.ndarray)

build_set_links(*token_set:* Token_set)

Build the links matrix for a given set.

Returns
The NxM links matrix for the set.

Return type
links (np.ndarray)

class nodes.nodeBuilder.**Build_sems**(*symProps: list[dict]*)

Bases: object

A class for building the semantic objects.

sems

A list of semantic names.

Type
list

nodes

A list of semantic objects.

Type
list

name_dict

A dictionary of semantic objects, mapping name to semantic object in nodes.

Type
dict

id_dict

A dictionary of semantic objects, mapping ID to semantic object in nodes.

Type
dict

num_sems

The number of semantics, iterated from 0 when assigning IDs.

Type
int

symProps

A list of symProps relating to the set.

Type
list

build_sems()

Create the sem_set object.

get_sems(*symProps: list[dict]*)

Get the list of all semantic names in the symProps.

nodulate()

Turn each unique semantic into a semantic object (node) with a unique ID.

class nodes.nodeBuilder.**Build_set**(*symProps: list[dict]*, *set:* Set)

Bases: object

A class for building the nodes for a given set.

symProps

A list of symProps relating to the set.

Type
list

tokens

A dictionary of tokens, mapping type to list of tokens.

Type
dict

set

The set to be built.

Type
Set

name_dict

A dictionary of tokens, mapping name to token in tokens.

Type
dict

id_dict

A dictionary of tokens, mapping ID to token in tokens.

Type
dict

build_set()

Returns a new token_set object

create_token(*name*, *token_class*, *analog*, *is_pred=None*)

Create a token object and add it to the name/dict.

Parameters

- **name** (*str*) – The name of the token.
- **token_class** (Token) – The class of the token.
- **analog** (*int*) – The analog of the token.
- **is_pred** (*bool*) – Whether the token is a predicate.

get_nodes()

Gets lists of unique tokens by type.

id_tokens()

Assign each token an ID, unique for the set.

class nodes.nodeBuilder.**Node**(*name*)

Bases: object

An intermediate class for representing a node in the network.

name

The name of the node.

Type
str

features

A list of features for the node.

Type
list

ID

The ID of the node.

Type
int

set(*feature*, *value: float*)

Set the feature of the node.

Parameters

- **feature** (*str*) – The feature to set.
- **value** (*float*) – The value to set the feature to.

set_ID(*ID*)

Set the ID of the node.

Parameters
ID (*int*) – The ID to set the node to.

class nodes.nodeBuilder.**PO**(*name*, *set*, *analog*, *is_pred: bool*)

Bases: *Token*

An intermediate class for representing a PO node.

name

The name of the PO.

Type
str

features

A list of features for the PO, indexed by TF.

Type
list

ID

The ID of the PO.

Type
int

children

A list of children of the PO, for use in building the connections matrix.

Type
list

class nodes.nodeBuilder.**Prop**(*name*, *set*, *analog*)

Bases: *Token*

An intermediate class for representing a Prop node.

name

The name of the Prop.

Type
str

features

A list of features for the Prop, indexed by TF.

Type
list

ID

The ID of the Prop.

Type
int

children

A list of children of the Prop, for use in building the connections matrix.

Type
list

class nodes.nodeBuilder.**RB**(*name*, *set*, *analog*)

Bases: *Token*

An intermediate class for representing a RB node.

name

The name of the RB.

Type
str

features

A list of features for the RB, indexed by TF.

Type
list

ID

The ID of the RB.

Type
int

children

A list of children of the RB, for use in building the connections matrix.

Type
list

class nodes.nodeBuilder.**Sem_set**(*sems: list[*Semantic*]*, *name_dict: dict[str,* Semantic*]*, *id_dict: dict[int,* Semantic*]*)

Bases: object

An intermediate class for representing a set of semantics.

sems

A list of semantics.

Type
list

name_dict

A dictionary of semantics, mapping name to semantic in sems.

Type
dict

id_dict

A dictionary of semantics, mapping ID to semantic in sems.

Type
dict

num_sems

The number of semantics in the set.

Type
int

connections

A matrix of connections between semantics.

Type
np.ndarray

get_sem(*name*)

Get a semantic from the semantic set by name.

Parameters
name (*str*) – The name of the semantic.

get_sem_by_id(*ID*)

Get a semantic from the semantic set by ID.

Parameters
ID (*int*) – The ID of the semantic.

tensorise()

Tensorise the semantic set, creating a tensor of semantics, and a tensor of connections between semantics.

class nodes.nodeBuilder.**Semantic**(*name*)

Bases: *Node*

An intermediate class for representing a semantic node.

name

The name of the semantic.

Type
str

features

A list of features for the semantic, indexed by SF.

Type
list

ID

The ID of the semantic.

Type
int

floatate_features()

Convert semantic features to floats, required for tensorisation.

initialise_defaults()

Initialise the default features for the semantic.

class nodes.nodeBuilder.**Token**(*name*, *set:* Set, *analog: int*)

Bases: *Node*

An intermediate class for representing a token node.

name

The name of the token.

Type
str

features

A list of features for the token, indexed by TF.

Type
list

ID

The ID of the token.

Type
int

children

A list of children of the token, for use in building the connections matrix.

Type
list

floatate_features()

Convert token features to floats, required for tensorisation.

initialise_defaults()

Initialise the default features for the token.

class nodes.nodeBuilder.**Token_set**(*set:* Set, *tokens: dict[*Type*, list[*Token*]]*, *name_dict: dict[str,* Token*]*, *id_dict: dict[int,* Token*]*)

Bases: object

An intermediate class for representing a set of tokens.

set

The set of the tokens.

Type
Set

tokens

A dictionary of tokens, mapping type to list of tokens.

Type
dict

name_dict

A dictionary of tokens, mapping name to token in tokens.

Type
dict

id_dict

A dictionary of tokens, mapping ID to token in tokens.

Type
dict

num_tokens

The number of tokens in the set.

Type
int

connections

A matrix of connections between tokens.

Type
np.ndarray

links

A matrix of links between tokens and semantics.

Type
np.ndarray

get_token(*name*)

Get a token from the token set by name.

Parameters
name (*str*) – The name of the token.

get_token_by_id(*ID*)

Get a token from the token set by ID.

Parameters
ID (*int*) – The ID of the token.

get_token_tensor()

Get the token tensor for the token set.

tensorise()

Tensorise the token set, creating a tensor of tokens, and tensors of connections and links to semantics.

class nodes.nodeBuilder.**nodeBuilder**(*symProps: list[dict] = None*, *file_path: str = None*)

Bases: object

A class for building the nodes object.

symProps

A list of symProps.

Type
list

file_path

The path to the sym file.

Type
str

token_sets

A dictionary of token sets, mapping set to token set object.

Type
dict

set_map

A dictionary of set mappings, mapping set name to set. Used for reading set from symProps file.

Type
dict

build_mem_objects()

Build the mem objects. (links, mappings)

build_node_tensors()

Build the per set tensor objects. (driver, recipient, memory, new_set, semantics)

build_nodes(*DORA_mode=True*)

Build the nodes object.

Parameters
DORA_mode (*bool*) – Whether to use DORA mode.

Returns
The nodes object.

Return type
nodes (*Nodes*)

Raises
ValueError – If no symProps or file_path set.

build_set_tensors()

Build the sem_set and token_sets.

get_symProps_from_file()

Read the symProps from the file into a list of dicts.

1.4 nodes.nodeEnums module

class nodes.nodeEnums.**B**(*value*)

Bases: IntEnum

FALSE = 0

TRUE = 1

class nodes.nodeEnums.**MappingFields**(*value*)

Bases: IntEnum

CONNETIONS = 3

HYPOTHESIS = 1

MAX_HYP = 2

WEIGHT = 0

class nodes.nodeEnums.**Mode**(*value*)

Bases: IntEnum

CHILD = 0

NEUTRAL = 1

PARENT = 2

class nodes.nodeEnums.**OntStatus**(*value*)

Bases: IntEnum

SDM = 2

STATE = 0

VALUE = 1

class nodes.nodeEnums.**SF**(*value*)

Bases: IntEnum

ACT = 6

AMOUNT = 3

ID = 0

INPUT = 4

MAX_INPUT = 5

ONT_STATUS = 2

TYPE = 1

class nodes.nodeEnums.**Set**(*value*)

Bases: IntEnum

DRIVER = 0

MEMORY = 2

NEW_SET = 3

RECIPIENT = 1

class nodes.nodeEnums.**TF**(*value*)

Bases: IntEnum

ACT = 12

ANALOG = 3

BU_INPUT = 18

COPIED_DR_INDEX = 26

COPY_FOR_DR = 25

DELETED = 28

GROUP_LAYER = 8

ID = 0

INFERRED = 23

INHIBITOR_ACT = 15

INHIBITOR_INPUT = 14

INHIBITOR_THRESHOLD = 7

LATERAL_INPUT = 19

MADE_UNIT = 5

MAKER_UNIT = 6

MAP_INPUT = 20

MAX_ACT = 13

MAX_MAP = 16

MAX_MAP_UNIT = 4

MAX_SEM_WEIGHT = 22

MODE = 9

NET_INPUT = 21

PRED = 29

RETRIEVED = 24

SEM_COUNT = 11

SET = 2

SIM_MADE = 27

TD_INPUT = 17

TIMES_FIRED = 10

TYPE = 1

class nodes.nodeEnums.**Type**(*value*)

Bases: IntEnum

GROUP = 3

P = 2

PO = 0

RB = 1

SEMANTIC = 4

1.5 nodes.nodeMemObjects module

class nodes.nodeMemObjects.**Links**(*driver_links*, *recipient_links*, *memory_links*)

Bases: object

A class for representing weighted connections between token sets and semantics.

add_links(*set:* Set, *links*)

Add links to the adjacency matrix. TODO: implement

class nodes.nodeMemObjects.**Mappings**(*connections*, *weights*, *hypotheses*, *max_hyps*)

Bases: object

A class for storing mappings and hypothesis information.

add_mappings(*mappings*)

Add mappings to the adjacency matrix. TODO: implement

connections()

Return the connections matrix from the adjacency matrix.

hypotheses()

Return the hypotheses matrix from the adjacency matrix.

max_hyps()

Return the max hypotheses matrix from the adjacency matrix.

updateHypotheses(*hypotheses*)

Update the hypotheses matrix. TODO: implement

weights()

Return the weights matrix from the adjacency matrix.

1.6 nodes.nodePrinter module

class nodes.nodePrinter.**C**(*value*)

Bases: IntEnum

Enum for the characters to print.

BOTTOM_LEFT = 2

BOTTOM_RIGHT = 3

CROSS = 6

HORIZONTAL = 4

HORIZONTAL_DOWN = 7

HORIZONTAL_UP = 8

TOP_LEFT = 0

TOP_RIGHT = 1

VERTICAL = 5

VERTICAL_LEFT = 9

VERTICAL_RIGHT = 10

class nodes.nodePrinter.**lineTypes**(*value*)

Bases: IntEnum

Enum for the type of line to print.

BOTTOM = 2

MIDDLE = 1

SPLIT = 3

TOP = 0

class nodes.nodePrinter.**nodePrinter**(*nodes: Nodes*, *print_to_console: bool = True*, *log_file: str = None*)

Bases: object

This class is used to print the nodes and their tensors to the console or a file.

nodes

The nodes object to print.

Type
Nodes

print_to_console

Whether to print to the console.

Type
bool

log_file

The file to print to.

Type
str

label_values(*row: list[float]*, *types: list[TF]*, *names: dict[int, str]*)

Label the values and names of the given row. :param row: The row to label. :type row: list :param types: The types of the features to label. (IE: TF.SET, TF.ID, etc.) :type types: list :param names: The names of the tokens. If None, the names will not be added to the row. :type names: dict

print_con_tensor(*tensor: Tensor*, *mask=None*, *names=None*, *headers=None*)

Print the given connections tensor. :param tensor: The tensor to print. :type tensor: torch.Tensor :param mask: The mask to apply to the tensor. :type mask: torch.Tensor :param names: The names of the tokens. If None, the names will not be printed. :type names: dict :param headers: The headers to print, defaults to "Connections:" if left as None. :type headers: list

print_links_tensor(*tensor: Tensor*, *mask=None*, *names=None*, *headers=None*)

Print the given links tensor. :param tensor: The tensor to print. :type tensor: torch.Tensor :param mask: The mask to apply to the tensor. :type mask: torch.Tensor :param names: The names of the tokens. If None, the names will not be printed. :type names: dict :param headers: The headers to print, defaults to "Links:" if left as None. :type headers: list

print_tk_tensor(*tensor: Tensor*, *types=None*, *label_values=True*, *names=None*, *headers=None*)

Print the given tensor of tokens. :param tensor: The tensor to print. :type tensor: torch.Tensor :param types: List of features to print. (IE: TF.SET, TF.ID, etc.) :type types: list :param label_values: Whether to convert feature floats to their enum names. (IE: TF.TYPE == 0.0 -> TYPE(0.0).name) :type label_values: bool :param names: The names of the tokens. If None, the names will not be printed. :type names: dict :param headers: The headers to print, defaults to "Tokens Tensor:" if left as None. :type headers: list

print_token_tensor(*set: Set*, *feature_types=None*, *mask=None*, *label_values=True*, *label_names=True*, *headers=None*, *print_cons=True*, *cons_headers=None*, *links_headers=None*)

Print the token tensor for a given set. :param set: The set to print. :type set: Set :param feature_types: List of features to print. (IE: TF.SET, TF.ID, etc.) :type feature_types: list :param mask: Mask of subtensor to print. :type mask: torch.Tensor :param label_values: Whether to convert features floats to their enum names. (IE: TF.TYPE == 0.0 -> TYPE(0.0).name) :type label_values: bool :param label_names: Whether to include the names for each node. (IE: ID==0 -> tensor.names[0]) :type label_names: bool :param headers: The headers to print, defaults to "Set: {set.name} Tokens" if left as None. :type headers: list :param print_cons: Whether to print the connections tensor. :type print_cons: bool :param cons_headers: The connections headers to print, defaults to "Set: {set.name} Connections" if left as None. :type cons_headers: list :param links_headers: The links headers to print, defaults to "Set: {set.name} Links" if left as None. :type links_headers: list

print_tokens(*set: Set*, *token_ids: list[int]*, *types: list[TF]*)

Print the given tokens. :param set: The set to print. :type set: Set :param token_ids: The ids of the tokens to print. :type token_ids: list :param types: The types of the features to print. (IE: TF.SET, TF.ID, etc.) :type types: list

class nodes.nodePrinter.**tablePrinter**(*columns: list[str]*, *rows: list[list[str]]*, *headers: list[str]*, *log_file: str = None*, *print_to_console: bool = True*)

Bases: `object`

Print a table of data.

columns

The columns of the table.

Type
list

rows

The rows of the table.

Type
list

headers

The headers of the table.

Type
list

log_file

The file to log to. Only logs if provided.

Type
str

print_to_console
Whether to print to the console.

Type
bool

calc_col_widths()
Calculate the widths of the columns, based on longest content in each column.

calc_header_width()
Calculate the widths of the header strings.

check_row_column_lengths()
Check that the number of columns in each row matches the number of columns in the table.

format(*data: list[str]*)
Format the given data into strings. :param data: The data to format. :type data: list

format_rows(*rows: list[list[str]]*)
Format the given rows data into strings. :param rows: The rows to format. :type rows: list

get_col_string(*startc*, *fillc*, *width*, *format_data=None*)
Return a string for the given column. :param startc: The character to start the column with. :type startc: str :param fillc: The character to fill the column with. :type fillc: str :param width: The width of the column. :type width: int :param format_data: The data to format. :type format_data: str

Returns
String for the given column.

Return type
str

get_line(*line_type:* lineTypes, *char_set: str*, *widths*, *format_data=None*)
Get the line of the given type. If self.print_to_console is True, print the line to the console. If self.log_file is not None, write the line to the file.

Parameters

- **line_type** (*lineTypes*) – The type of line to get.
- **char_set** (*str*) – The character set to use.
- **widths** (*list*) – The widths of the columns.
- **format_data** (*list*) – The data to format.

get_line_chars(*line_type:* lineTypes, *char_set: str*)
Get the characters for the given line type and character set.

get_line_no_data(*line_type:* lineTypes, *char_set: str*, *widths*)
Return a string for the given line type. :param line_type: The type of line to get. :type line_type: lineTypes :param char_set: The character set to use. :type char_set: str :param widths: The widths of the columns. :type widths: list

Returns
Line of the given type.

Return type
str

get_line_with_data(*line_type:* lineTypes, *char_set: str*, *widths*, *format_data*)

Return a string for the given row in the table. :param line_type: The type of line to get. :type line_type: lineTypes :param char_set: The character set to use. :type char_set: str :param widths: The widths of the columns. :type widths: list :param format_data: The data to format. :type format_data: list

Returns
The line of the given type, with data.

Return type
str

open_file(*filename*)

Open the given file. If the file exists, open it in append mode. If the file does not exist, create it and open it in write mode.

Parameters
filename (*str*) – The name of the file to open.

Returns
The file object.

Return type
file

print_column_names(*split=True*, *char_set='table'*)

Print the column names. :param split: Whether to add a split line after column names, otherwise add a bottom line. Default is True. :type split: bool :param char_set: The character set to use. :type char_set: str

print_header(*char_set='header'*)

Print the header. :param char_set: The character set to use. :type char_set: str

print_rows(*print_top=False*, *print_bottom=True*, *char_set='table'*, *split=False*)

Print all rows of data in the table. :param print_top: Whether to print a top line. Default is False. :type print_top: bool :param print_bottom: Whether to print a bottom line. Default is True. :type print_bottom: bool :param char_set: The character set to use. :type char_set: str :param split: Whether to add a split line after each row. Default is False. :type split: bool

print_table(*header=True*, *column_names=True*, *header_char_set='header'*, *column_char_set='table'*, *row_char_set='table'*, *split=False*)

Print the header, column names, and rows of data in the table. :param header: Whether to print the header. Default is True. :type header: bool :param column_names: Whether to print the column names. Default is True. :type column_names: bool :param header_char_set: The character set to use for the header. Default is “header”. :type header_char_set: str :param column_char_set: The character set to use for the column names. Default is “table”. :type column_char_set: str :param row_char_set: The character set to use for the rows. Default is “table”. :type row_char_set: str :param split: Whether to add a split line after each row. Default is False. :type split: bool

1.7 nodes.nodeTensors module

class nodes.nodeTensors.**Driver**(*floatTensor*, *connections*, *links*, *names: dict[int, str] = None*)

Bases: *Tokens*

A class for representing the driver set of tokens.

names

A dictionary mapping token IDs to token names. Defaults to None.

Type
dict, optional

nodes

An NxTokenFeatures tensor of floats representing the tokens.

Type
torch.Tensor

analogs

An Ax1 tensor listing all analogs in the tensor.

Type
torch.Tensor

analog_counts

An Ax1 tensor listing the number of tokens per analog

Type
torch.Tensor

links

A Tensor of links from tokens in this set to the semantics.

Type
torch.Tensor

connections

An NxN tensor of connections from parent to child for tokens in this set.

Type
torch.Tensor

masks

A Tensor of masks for the tokens in this set.

Type
torch.Tensor

check_global_inhibitor()

Return true if any RB.inhibitor_act == 1.0

check_local_inhibitor()

Return true if any PO.inhibitor_act == 1.0

update_input(*as_DORA*)

Update all input in driver

update_input_p_child(*as_DORA*)

Update input in driver for P units in child mode

update_input_p_parent()

Update input in driver for P units in parent mode

update_input_po(*as_DORA*)

Update input in driver for PO units

update_input_rb(*as_DORA*)

Update input in driver for RB units

class nodes.nodeTensors.**Recipient**(*floatTensor*, *connections*, *links*, *names=None*)

Bases: *Tokens*

A class for representing the recipient set of tokens.

names

A dictionary mapping token IDs to token names. Defaults to None.

Type
dict, optional

nodes

An NxTokenFeatures tensor of floats representing the tokens.

Type
torch.Tensor

analogs

An Ax1 tensor listing all analogs in the tensor.

Type
torch.Tensor

analog_counts

An Ax1 tensor listing the number of tokens per analog

Type
torch.Tensor

links

A Tensor of links between tokens in this set and the semantics.

Type
torch.Tensor

connections

An NxN tensor of connections from parent to child for tokens in this set.

Type
torch.Tensor

map_input(*t_mask*, *mappings:* Mappings, *driver:* Driver)

Calculate mapping input for tokens in mask

Parameters

- **t_mask** (*torch.Tensor*) – A mask of tokens to calculate mapping input for
- **mappings** (Mappings) – A Mappings object
- **driver** (Driver) – A Driver object

Returns
A (sum(t_mask) x 1) matrix of mapping input for tokens in mask

Return type
torch.Tensor

update_input(*as_DORA*, *phase_set*, *lateral_input_level*, *semantics*, *mappings*, *driver*, *ignore_object_semantics=False*)

Update all input in recipient

Parameters

- **as_DORA** (*bool*) – Whether to use DORA mode
- **phase_set** (*int*) – The current phase set
- **lateral_input_level** (*float*) – The level of lateral input
- **semantics** (*Semantics*) – A Semantics object
- **mappings** (Mappings) – A Mappings object
- **driver** (Driver) – A Driver object
- **ignore_object_semantics** (*bool*) – Whether to ignore object semantics

update_input_p_child(*as_DORA*, *phase_set*, *lateral_input_level*, *mappings:* Mappings, *driver:* Driver)

Update input for P units in child mode

Parameters

- **as_DORA** (*bool*) – Whether to use DORA mode
- **phase_set** (*int*) – The current phase set
- **lateral_input_level** (*float*) – The level of lateral input
- **mappings** (Mappings) – A Mappings object
- **driver** (Driver) – A Driver object

update_input_p_parent(*phase_set*, *lateral_input_level*, *mappings:* Mappings, *driver:* Driver)

Update input for P units in parent mode

Parameters

- **phase_set** (*int*) – The current phase set
- **lateral_input_level** (*float*) – The level of lateral input
- **mappings** (Mappings) – A Mappings object
- **driver** (Driver) – A Driver object

update_input_po(*as_DORA*, *phase_set*, *lateral_input_level*, *semantics*, *mappings*, *driver*, *ignore_object_semantics=False*)

Update input for PO units

Parameters

- **as_DORA** (*bool*) – Whether to use DORA mode
- **phase_set** (*int*) – The current phase set
- **lateral_input_level** (*float*) – The level of lateral input

update_input_rb(*phase_set*, *lateral_input_level*, *mappings:* Mappings, *driver:* Driver)

Update input for RB units

Parameters

- **phase_set** (*int*) – The current phase set
- **lateral_input_level** (*float*) – The level of lateral input
- **mappings** (Mappings) – A Mappings object
- **driver** (Driver) – A Driver object

class nodes.nodeTensors.**Semantic**(*nodes*, *connections*, *links:* Links, *names=None*)

Bases: object

A class for representing semantics nodes.

names

A dictionary mapping semantic IDs to semantic names. Defaults to None.

Type
dict, optional

nodes

An NxSemanticFeatures tensor of floats representing the semantics.

Type
torch.Tensor

connections

An NxN tensor of connections from parent to child for semantics in this set.

Type
torch.Tensor

links

A Links object containing links from token sets to semantics.

Type
Links

initialise_input(*refresh*)

Initialise the input of the semantics

intitialse_sem()

Initialise the semantics

set_max_input(*max_input*)

Set the max input of the semantics

update_act()

Update the acts of the semantics

update_input(*driver*, *recipient*, *memory=None*, *ignore_obj=False*, *ignore_mem=False*)

Update the input of the semantics

update_input_from_set(*tensor:* Tokens, *set:* Set, *ignore_obj=False*)

Update the input of the semantics from a set of tokens

class nodes.nodeTensors.**Tokens**(*floatTensor*, *connections*, *links*, *names: dict[int, str] = None*)

Bases: object

A class for holding a tensor of tokens, and performing general tensor operations.

names

A dictionary mapping token IDs to token names. Defaults to None.

Type
dict, optional

nodes

An NxTokenFeatures tensor of floats representing the tokens.

Type
torch.Tensor

analogs

An Ax1 tensor listing all analogs in the tensor.

Type
torch.Tensor

analog_counts

An Ax1 tensor listing the number of tokens per analog

Type
torch.Tensor

links

A Tensor of links from tokens in this set to the semantics.

Type
torch.Tensor

connections

An NxN tensor of connections from parent to child for tokens in this set.

Type
torch.Tensor

masks

A Tensor of masks for the tokens in this set.

Type
torch.Tensor

add_nodes(*nodes*)

Add nodes to tensor

analog_node_count()

Update list of analogs in tensor, and their node counts

cache_masks(*types_to_recompute=None*)

Compute and cache masks, specify types to recompute via list of tokenTypes

compute_mask(*token_type:* Type)

Compute the mask for a token type

del_Nodes(*nodes*)

Delete nodes from tensor

get_combined_mask(*n_types: list[*Type*]*)

Return combined mask of given types

get_mask(*token_type:* Type)

Return mask for given token type

initialise_act(*n_type: list[*Type*]*)

Initialize act to 0.0, and call initialise_inputs

Parameters
n_type (*list[Type]*) – The types of nodes to initialise.

initialise_float(*n_type: list[Type]*, *features: list[TF]*)

Initialise given features

Parameters

- **n_type** (*list[Type]*) – The types of nodes to initialise.
- **features** (*list[TF]*) – The features to initialise.

initialise_input(*n_type: list[Type]*, *refresh: float*)

Initialize inputs to 0, and td_input to refresh

Parameters

- **n_type** (*list[Type]*) – The types of nodes to initialise.
- **refresh** (*float*) – The value to set the td_input to.

initialise_state(*n_type: list[Type]*)

Set self.retrieved to false, and call initialise_act

Parameters
n_type (*list[Type]*) – The types of nodes to initialise.

p_get_mode()

Set mode for all P units

p_initialise_mode()

Initialize mode to neutral for all P units.

po_get_max_semantic_weight()

Set max link weight feature for all PO nodes

po_get_weight_length()

Set sem count feature for all PO nodes

reset_inhibitor(*n_type: list[Type]*)

Reset the inhibitor input and act to 0.0 for given type

Parameters
n_type (*list[Type]*) – The types of nodes to reset inhibitor inputs and acts.

update_act(*gamma: float*, *delta: float*, *HebbBias: float*)

Update act of nodes

Parameters

- **gamma** (*float*) – Effects the increase in act for each unit.
- **delta** (*float*) – Effects the decrease in act for each unit.
- **HebbBias** (*float*) – The bias for mapping input relative to TD/BU/LATERAL inputs.

update_inhibitor_act(*n_type: list[Type]*)

Update the inhibitor act for given type

Parameters
n_type (*list[Type]*) – The types of nodes to update inhibitor acts.

update_inhibitor_input(*n_type: list[Type]*)

Update inputs to inhibitors by current activation for nodes of type n_type

Parameters

n_type (*list[Type]*) – The types of nodes to update inhibitor inputs.

zero_lateral_input(*n_type: list[Type]*)

Set lateral_input to 0; to allow synchrony at different levels by 0-ing lateral inhibition at that level (e.g., to bind via synchrony, 0 lateral inhibition in POs).

Parameters

n_type (*list[Type]*) – The types of nodes to set lateral_input to 0.

1.8 nodes.nodes module

class nodes.nodes.**Nodes**(*driver: Driver*, *recipient: Recipient*, *LTM: Tokens*, *new_set: Tokens*, *semantics: Semantic*, *mappings: Mappings*, *DORA_mode: bool*)

Bases: object

A class for holding token tensors for each set, and accessing node operations.

checkDriverPOs()

Check local inhibitor activation.

checkDriverRBs()

Check global inhibitor activation.

fire_global_inhibitor()

Fire the global inhibitor.

fire_local_inhibitor()

Fire the local inhibitor.

update_acts_am(*gamma*, *delta*, *hebb_bias*)

Update the acts in the active memory.

update_acts_driver(*gamma*, *delta*, *hebb_bias*)

Update the acts in the driver.

update_acts_recipient(*gamma*, *delta*, *hebb_bias*)

Update the acts in the recipient.

update_inputs_am(*as_DORA*, *phase_set*, *lateral_input_level*, *ignore_object_semantics=False*)

Update the inputs in the active memory.

Parameters

- **as_DORA** (*bool*) – Whether to use DORA mode.
- **phase_set** (*Int*) – The current phase set.
- **lateral_input_level** (*float*) – The lateral input level.
- **ignore_object_semantics** (*bool, optional*) – Whether to ignore object semantics input. Defaults to False.

update_inputs_driver(*as_DORA*)

Update the inputs in the driver.

Parameters
as_DORA (*bool*) – Whether to use DORA mode.

update_inputs_recpient(*as_DORA*, *phase_set*, *lateral_input_level*, *ignore_object_semantics=False*)

Update the inputs in the recipient.

Parameters

- **as_DORA** (*bool*) – Whether to use DORA mode.
- **phase_set** (*int*) – The current phase set.
- **lateral_input_level** (*float*) – The lateral input level.
- **ignore_object_semantics** (*bool, optional*) – Whether to ignore object semantics input. Defaults to False.

1.9 nodes.tensorOps module

nodes.tensorOps.**diag_zeros**(*M*)

Return MxM matrix of all ones except the diagonal from T[0, 0] to T[M, M] :param M: The size of the matrix :type M: int

Returns
A MxM matrix of all ones except the diagonal from T[0, 0] to T[M, M]

Return type
torch.Tensor

nodes.tensorOps.**refine_mask**(*tensor*, *mask*, *index*, *value*, *in_place=False*)

Returns a mask, that is the union of mask and the submask where tensor[mask, index] == value :param tensor: The input tensor :type tensor: torch.Tensor :param mask: The input mask :type mask: torch.Tensor :param index: The index of the value to check :type index: int :param value: The value to check for :type value: int :param in_place: Whether to modify the input mask in place :type in_place: bool

Returns
Mask(size of input mask) with union of input mask and submask where tensor[mask, index] == value

Return type
torch.Tensor

nodes.tensorOps.**sub_union**(*mask*, *submask*, *in_place=False*)

Returns a mask, that is the union of the input mask and its submask :param mask: The input mask :type mask: torch.Tensor :param submask: The submask :type submask: torch.Tensor :param in_place: Whether to modify the mask in place :type in_place: bool

Returns
Mask(size of input mask) with union of input mask and submask

Return type
torch.Tensor

nodes.tensorOps.**undirected**(*T*)

Returns the undirected matrix made by OR of both directions of a given matrix T :param T: The input matrix :type T: torch.Tensor

Returns
The undirected matrix made by OR of both directions of T

Return type
torch.Tensor

1.10 Module contents

PYTHON MODULE INDEX

n

INDEX