
narchi Documentation

Mauricio Villegas

Nov 04, 2021

CONTENTS

1 Main features	3
2 Teaser example	5
3 Documentation Contents	9
3.1 Json Schema	9
3.2 Command line tool	16
3.3 API Reference	21
4 Indices and tables	53
Python Module Index	55
Index	57

narchi is a python package that provides functionalities for defining neural network architectures in an implementation independent way. It is intended to make network architectures highly configurable while also making the task easier.

MAIN FEATURES

- Network architectures are written in `jsonnet format`, which provides useful features like input parameters and functions to define repeated blocks.
- The shapes of the tensors internal to the networks are automatically deduced by propagating the shapes of the inputs, thus requiring less effort and being less error prone.
- Propagation of shapes is done using symbolic arithmetic which makes it simple to understand relationships between inputs and the derived shapes.
- Architecture files can reference other architecture files, thus making this approach modular.
- A command line tool is included to validate jsonnet architecture files and to create detailed diagrams of the respective network architectures.
- Several `examples` intended to illustrate different features supported.
- Includes basic implementations that allows to instantiate pytorch modules:
 - Instantiation only requires a jsonnet architecture file.
 - No need to write module classes or forward function for each new architecture.
 - One basic implementation that supports instatiating several of the examples.
 - A second example that supports packed 1d and 2d sequences which illustrates the implementation independent nature of the architecture files.

TEASER EXAMPLE

Here you can see an example that illustrates what *narchi* provides. The example is for resnet18 as implemented in torchvision, though bare in mind that the potential of *narchi* is the ease of configurability of network architectures, not the reimplementing of existing architectures.

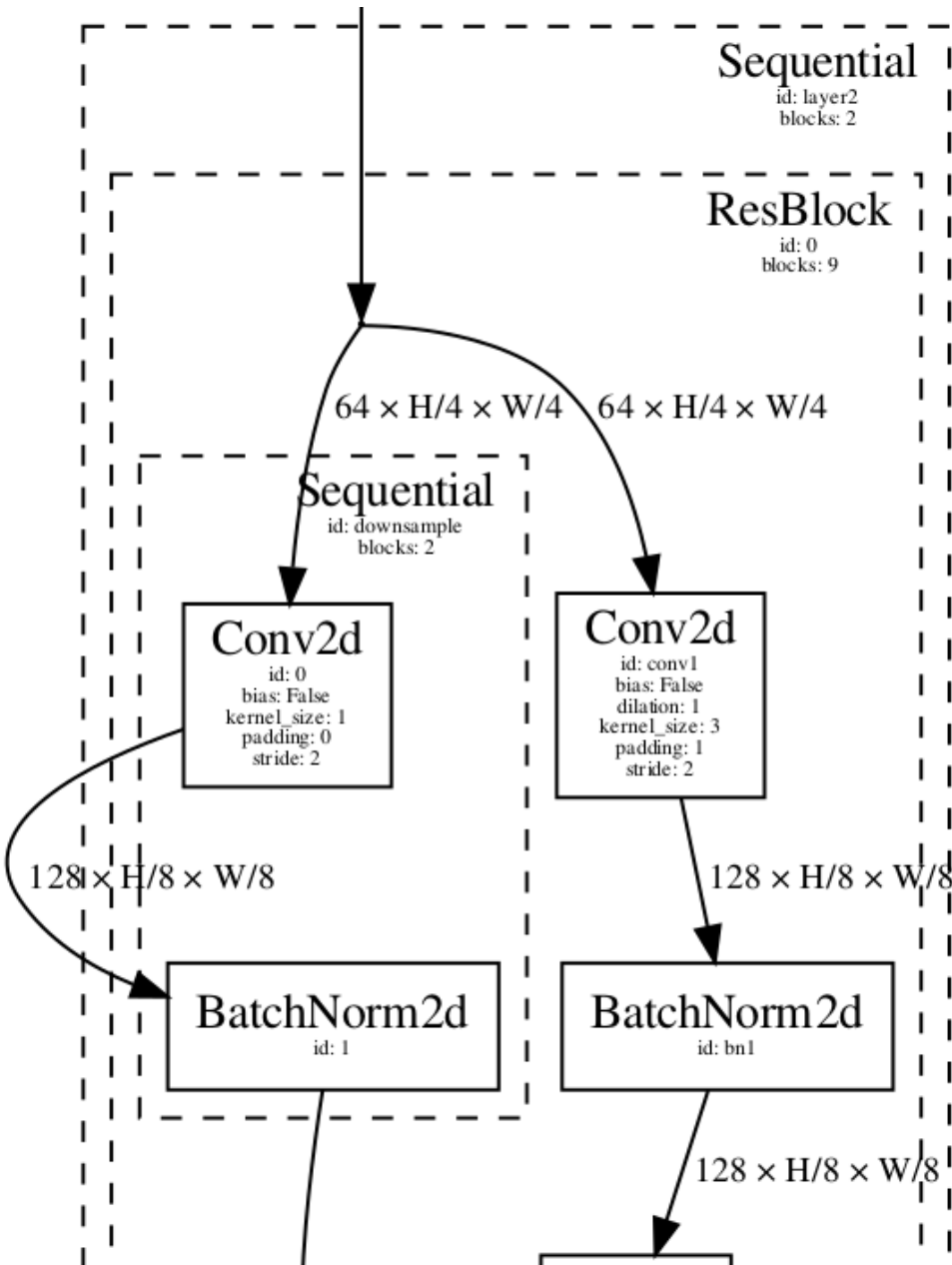
Instantiating a pytorch module from the architecture file can be easily done as follows.

```
from narchi.instantiators.pytorch import StandardModule
module = StandardModule('resnet.jsonnet',
                        state_dict='resnet18-5c106cde.pth',
                        cfg={'ext_vars': {"num_blocks": [2, 2, 2, 2]}})
```

Creating a diagram of the architecture requires a single command like the following.

```
narchi_cli.py render \  
  --ext_vars '{"num_blocks": [2, 2, 2, 2]}' \  
  --nested_depth 4 \  
  resnet.jsonnet \  
  resnet18.pdf
```

Below you can see a small part of the rendered diagram of the start of the first downsample layer of resnet18.



The part of the json that generated the previous crop of the architecture diagram can be seen below. Note that information of the shapes is not included, since these are derived automatically.

```
{
  "_class": "Sequential",
  "_id": "layer2",
  "blocks": [
    {
      "_class": "Group",
      "_name": "ResBlock",
      "blocks": [
        {
          "_class": "Identity",
```

(continues on next page)

(continued from previous page)

```
    "_id": "ident"
  },
  {
    "_class": "Conv2d",
    "_id": "conv1",
    "bias": false,
    "dilation": 1,
    "kernel_size": 3,
    "output_size": 128,
    "padding": 1,
    "stride": 2
  },
  {
    "_class": "BatchNorm2d",
    "_id": "bn1"
  },
  {
    "_class": "Sequential",
    "_id": "downsample",
    "blocks": [
      {
        "_class": "Conv2d",
        "bias": false,
        "kernel_size": 1,
        "output_size": 128,
        "padding": 0,
        "stride": 2
      },
      {
        "_class": "BatchNorm2d"
      }
    ]
  },
  {"...": "..."}
],
"graph": [
  "ident -> conv1 -> bn1 -> relu1 -> conv2 -> bn2 -> add -> relu2",
  "ident -> downsample -> add"
],
"input": "ident",
"output": "relu2"
}
]
```


DOCUMENTATION CONTENTS

3.1 Json Schema

```
1 {
2   "$schema": "http://json-schema.org/draft-07/schema#",
3   "$id": "https://schema.omnius.com/json/narchi/1.0/schema.json",
4   "title": "Neural Network Module Architecture Schema",
5   "$ref": "#/definitions/architecture",
6   "definitions": {
7     "id": {
8       "type": "string",
9       "pattern": "^[A-Za-z_][0-9A-Za-z_]*$"
10    },
11    "description": {
12      "type": "string",
13      "minLength": 8,
14      "pattern": "^[^<>]+$"
15    },
16    "dims": {
17      "type": "array",
18      "minItems": 1,
19      "items": {
20        "oneOf": [
21          {
22            "type": "integer",
23            "minimum": 1
24          },
25          {
26            "type": "string",
27            "pattern": "^(<<variable:([-+/*0-9A-Za-z_]+)>>|<<auto>>)$"
28          }
29        ]
30      }
31    },
32    "dims_in": {
33      "type": "array",
34      "minItems": 1,
35      "items": {
36        "oneOf": [
37          {
```

(continues on next page)

```

38     "type": "integer",
39     "minimum": 1
40   },
41   {
42     "type": "string",
43     "pattern": "^(<<variable:([-+/*0-9A-Za-z_]+)>>|<<auto>>)$"
44   },
45   {
46     "type": "null"
47   }
48 ]
49 }
50 },
51 "shape": {
52   "type": "object",
53   "properties": {
54     "in": {
55       "$ref": "#/definitions/dims_in"
56     },
57     "out": {
58       "$ref": "#/definitions/dims"
59     }
60   },
61   "required": [
62     "in",
63     "out"
64   ],
65   "additionalProperties": false
66 },
67 "graph": {
68   "type": "array",
69   "minItems": 1,
70   "items": {
71     "type": "string",
72     "pattern": "^[A-Za-z_][0-9A-Za-z_]*( +-> +[A-Za-z_][0-9A-Za-z_]*)+$"
73   }
74 },
75 "block": {
76   "type": "object",
77   "properties": {
78     "_class": {
79       "$ref": "#/definitions/id"
80     },
81     "_name": {
82       "$ref": "#/definitions/id"
83     },
84     "_id": {
85       "$ref": "#/definitions/id"
86     },
87     "_id_share": {
88       "$ref": "#/definitions/id"
89   },

```

(continues on next page)

(continued from previous page)

```

90     "_description": {
91         "$ref": "#/definitions/description"
92     },
93     "_shape": {
94         "$ref": "#/definitions/shape"
95     },
96     "_path": {
97         "$ref": "#/definitions/path"
98     },
99     "_ext_vars": {
100         "type": "object"
101     },
102     "blocks": {
103         "$ref": "#/definitions/blocks"
104     },
105     "input": {
106         "$ref": "#/definitions/id"
107     },
108     "output": {
109         "$ref": "#/definitions/id"
110     },
111     "graph": {
112         "$ref": "#/definitions/graph"
113     },
114     "dim": {
115         "type": "integer"
116     },
117     "reshape_spec": {
118         "$ref": "#/definitions/reshape"
119     },
120     "architecture": {
121         "$ref": "#/definitions/architecture"
122     }
123 },
124 "required": [
125     "_class"
126 ],
127 "allOf": [
128     {
129         "if": {
130             "properties": {
131                 "_class": {
132                     "enum": [
133                         "Sequential",
134                         "Group"
135                     ]
136                 }
137             }
138         },
139         "then": {
140             "required": [
141                 "blocks"

```

(continues on next page)

```
142     ]
143   },
144   "else": {
145     "not": {
146       "required": [
147         "blocks"
148       ]
149     }
150   }
151 },
152 {
153   "if": {
154     "properties": {
155       "_class": {
156         "const": "Group"
157       }
158     }
159   },
160   "then": {
161     "required": [
162       "graph",
163       "input",
164       "output"
165     ]
166   },
167   "else": {
168     "not": {
169       "required": [
170         "graph",
171         "input",
172         "output"
173       ]
174     }
175   }
176 },
177 {
178   "if": {
179     "properties": {
180       "_class": {
181         "const": "Module"
182       }
183     }
184   },
185   "then": {
186     "required": [
187       "_path"
188     ]
189   },
190   "else": {
191     "not": {
192       "required": [
193         "_path",
```

(continues on next page)

(continued from previous page)

```
194         "_ext_vars",
195         "architecture"
196     ]
197 }
198 }
199 },
200 {
201     "if": {
202         "properties": {
203             "_class": {
204                 "const": "Sequential"
205             }
206         }
207     },
208     "else": {
209         "properties": {
210             "blocks": {
211                 "items": {
212                     "required": [
213                         "_id"
214                     ]
215                 }
216             }
217         }
218     }
219 },
220 {
221     "if": {
222         "properties": {
223             "_class": {
224                 "const": "Concatenate"
225             }
226         }
227     },
228     "then": {
229         "required": [
230             "dim"
231         ]
232     }
233 },
234 {
235     "if": {
236         "properties": {
237             "_class": {
238                 "const": "Reshape"
239             }
240         }
241     },
242     "then": {
243         "required": [
244             "reshape_spec"
245         ]
246     }
247 }
```

(continues on next page)

```
246     },
247     "else": {
248       "not": {
249         "required": [
250           "reshape_spec"
251         ]
252       }
253     }
254   }
255 ]
256 },
257 "blocks": {
258   "type": "array",
259   "minItems": 1,
260   "items": {
261     "$ref": "#/definitions/block"
262   }
263 },
264 "path": {
265   "type": "string",
266   "pattern": ".+\\.\\.jsonnet"
267 },
268 "inputs_outputs": {
269   "type": "array",
270   "minItems": 1,
271   "items": {
272     "type": "object",
273     "properties": {
274       "_id": {
275         "$ref": "#/definitions/id"
276       },
277       "_description": {
278         "$ref": "#/definitions/description"
279       },
280       "_shape": {
281         "$ref": "#/definitions/dims"
282       }
283     },
284     "required": [
285       "_id",
286       "_shape"
287     ],
288     "additionalProperties": false
289   }
290 },
291 "architecture": {
292   "type": "object",
293   "properties": {
294     "_id": {
295       "$ref": "#/definitions/id"
296     },
297     "_description": {
```

(continues on next page)

(continued from previous page)

```
298     "$ref": "#/definitions/description"
299   },
300   "blocks": {
301     "$ref": "#/definitions/blocks"
302   },
303   "graph": {
304     "$ref": "#/definitions/graph"
305   },
306   "inputs": {
307     "$ref": "#/definitions/inputs_outputs"
308   },
309   "outputs": {
310     "$ref": "#/definitions/inputs_outputs"
311   }
312 },
313 "required": [
314   "_id",
315   "blocks",
316   "graph",
317   "inputs",
318   "outputs"
319 ],
320 "additionalProperties": false
321 },
322 "reshape": {
323   "oneOf": [
324     {
325       "const": "flatten"
326     },
327     {
328       "type": "array",
329       "minItems": 1,
330       "items": {
331         "oneOf": [
332           {
333             "$ref": "#/definitions/reshape_index"
334           },
335           {
336             "$ref": "#/definitions/reshape_flatten"
337           },
338           {
339             "$ref": "#/definitions/reshape_unflatten"
340           }
341         ]
342       }
343     }
344   ]
345 },
346 "reshape_index": {
347   "type": "integer",
348   "minimum": 0
349 },
```

(continues on next page)

(continued from previous page)

```

350 "reshape_dims": {
351   "type": "array",
352   "minItems": 2,
353   "items": {
354     "oneOf": [
355       {
356         "type": "integer",
357         "minimum": 1
358       },
359       {
360         "type": "string",
361         "pattern": "^(<<variable:([-/*0-9A-Za-z_]+)>>|<<auto>>)$"
362       }
363     ]
364   }
365 },
366 "reshape_flatten": {
367   "type": "array",
368   "minItems": 2,
369   "items": {
370     "$ref": "#/definitions/reshape_index"
371   }
372 },
373 "reshape_unflatten": {
374   "type": "object",
375   "minProperties": 1,
376   "maxProperties": 1,
377   "patternProperties": {
378     "^[0-9]+$": {
379       "$ref": "#/definitions/reshape_dims"
380     }
381   },
382   "additionalProperties": false
383 }
384 }
385 }

```

3.2 Command line tool

3.2.1 narchi_cli.py validate

Command for checking the validity of neural network module architecture files.

```

usage: narchi_cli.py validate [-h] [--version] [--cfg CFG]
                             [--print_config [= {comments,skip_null}+]]
                             [--validate {true,false}]
                             [--propagate {true,false}]
                             [--propagated {true,false}]

```

(continues on next page)

(continued from previous page)

```

[--propagators PROPAGATORS]
[--ext_vars EXT_VARS] [--cwd CWD]
[--parent_id PARENT_ID]
[--overwrite {true,false}] [--outdir OUTDIR]
[--save_json {true,false}]
jsonnet_paths [jsonnet_paths ...]

```

Positional Arguments

jsonnet_paths Path(s) to neural network module architecture file(s) in jsonnet narchi format.

Named Arguments

--version Print version and exit.
--cfg Path to a configuration file.
--print_config Print configuration and exit.

Loading related options

--validate Whether to validate architecture against narchi schema.
 Default: True

--propagate Whether to propagate shapes in architecture.
 Default: True

--propagated Whether architecture has already been propagated.
 Default: False

--propagators Overrides default propagators.
 Default: “default”

--ext_vars External variables required to load jsonnet.
 Default: {}

--cwd Current working directory to load inner referenced files. Default None uses directory of main architecture file.

--parent_id Identifier of parent module.
 Default: “”

Output related options

- overwrite** Whether to overwrite existing files.
Default: False
- outdir** Directory where to write output files.
Default: “.”
- save_json** Whether to write the architecture (up to the last successful step: jsonnet load, schema validation, parsing) in json format to the output directory.
Default: False

3.2.2 narchi_cli.py render

Command for rendering a neural network module architecture file.

```
usage: narchi_cli.py render [-h] [--version] [--cfg CFG]
                             [--print_config [= {comments, skip_null}+]]
                             [--validate {true, false}]
                             [--propagate {true, false}]
                             [--propagated {true, false}]
                             [--propagators PROPAGATORS] [--ext_vars EXT_VARS]
                             [--cwd CWD] [--parent_id PARENT_ID]
                             [--overwrite {true, false}] [--outdir OUTDIR]
                             [--save_json {true, false}]
                             [--save_pdf {true, false}] [--save_gv {true, false}]
                             [--block_attrs BLOCK_ATTRS]
                             [--block_labels BLOCK_LABELS]
                             [--edge_attrs EDGE_ATTRS]
                             [--nested_depth NESTED_DEPTH]
                             [--full_ids {true, false}]
                             [--layout_prog {dot, neato, twopi, circo, fdp}]
                             jsonnet_path [out_file]
```

Positional Arguments

- jsonnet_path** Path to a neural network module architecture file in jsonnet narchi format.
- out_file** Path where to write the architecture diagram (with a valid extension for pygraphviz draw). If unset a pdf is saved to the output directory.

Named Arguments

--version	Print version and exit.
--cfg	Path to a configuration file.
--print_config	Print configuration and exit.

Loading related options

--validate	Whether to validate architecture against narchi schema. Default: True
--propagate	Whether to propagate shapes in architecture. Default: True
--propagated	Whether architecture has already been propagated. Default: False
--propagators	Overrides default propagators. Default: "default"
--ext_vars	External variables required to load jsonnet. Default: {}
--cwd	Current working directory to load inner referenced files. Default None uses directory of main architecture file.
--parent_id	Identifier of parent module. Default: ""

Output related options

--overwrite	Whether to overwrite existing files. Default: False
--outdir	Directory where to write output files. Default: "."
--save_json	Whether to write the architecture (up to the last successful step: jsonnet load, schema validation, parsing) in json format to the output directory. Default: False

Rendering related options

- save_pdf** Whether to write rendered pdf file to output directory.
Default: False
- save_gv** Whether to write graphviz file to output directory.
Default: False
- block_attrs** Attributes for block nodes.
Default: { 'Default': 'shape=box', 'Input': 'shape=box, style=rounded, pen-width=1.5', 'Output': 'shape=box, style=rounded, peripheries=2', 'Nested': 'shape=box, style=dashed', 'Shared': 'style=filled', 'Reshape': 'shape=hexagon', 'Identity': 'shape=circle, width=0', 'Add': 'shape=circle, margin=0, width=0' }
- block_labels** Fixed labels for block nodes.
Default: { 'Identity': ' ', 'Add': '+' }
- edge_attrs** Attributes for edges.
Default: "fontsize=10"
- nested_depth** Maximum depth for nested subblocks to render. Set to 0 for unlimited.
Default: 3
- full_ids** Whether block IDs should include parent prefix.
Default: False
- layout_prog** Possible choices: dot, neato, twopi, circo, fdp
The graphviz layout method to use.
Default: "dot"

3.2.3 narchi_cli.py schema

Prints a schema as a pretty json.

```
usage: narchi_cli.py schema [-h] [{narchi,propagated,reshape,block,mappings}]
```

Positional Arguments

- schema** Possible choices: narchi, propagated, reshape, block, mappings
Which of the available schemas to print.
Default: "narchi"

3.3 API Reference

3.3.1 narchi.blocks

Blocks definitions and functions related to registering propagators.

Classes:

<i>SameShapeBlocksEnum</i> (value)	Enum of blocks that preserve the input shape.
<i>ConcatBlocksEnum</i> (value)	Enum of blocks that concatenate multiple inputs.
<i>FixedOutputBlocksEnum</i> (value)	Enum of blocks that have fixed outputs.
<i>ConvBlocksEnum</i> (value)	Enum of convolution-style blocks.
<i>RnnBlocksEnum</i> (value)	Enum of recurrent-style blocks.
<i>ReshapeBlocksEnum</i> (value)	Enum of blocks that transform the shape.
<i>GroupPropagatorsEnum</i> (value)	Enum of blocks that group other blocks.

Functions:

<i>register_propagator</i> (propagator[, replace])	Adds a propagator to the dictionary of registered propagators.
<i>register_known_propagators</i> ()	Function that registers all propagators defined in the modules of the package.

class `narchi.blocks.SameShapeBlocksEnum`(value)

Bases: `enum.Enum`

Enum of blocks that preserve the input shape.

Attributes:

<i>Sigmoid</i> (from_blocks, block[, propagators, ...])	Block that applies a sigmoid function.
<i>LogSigmoid</i> (from_blocks, block[, ...])	Block that applies a log-sigmoid function.
<i>Softmax</i> (from_blocks, block[, propagators, ...])	Block that applies a softmax function.
<i>LogSoftmax</i> (from_blocks, block[, ...])	Block that applies a log-softmax function.
<i>Tanh</i> (from_blocks, block[, propagators, ...])	Block that applies a hyperbolic tangent function.
<i>ReLU</i> (from_blocks, block[, propagators, ...])	Block that applies a rectified linear unit function.
<i>LeakyReLU</i> (from_blocks, block[, propagators, ...])	Block that applies a leaky rectified linear unit function.
<i>Dropout</i> (from_blocks, block[, propagators, ...])	Block that applies dropout, randomly set elements to zero.
<i>BatchNorm2d</i> (from_blocks, block[, ...])	Block that does 2D batch normalization.
<i>Identity</i> (from_blocks, block[, propagators, ...])	Block that does nothing, useful to connect one tensor to multiple blocks in a graph.
<i>Add</i> (from_blocks, block[, propagators, ...])	Block that adds the values of all input tensors.
<i>CRF</i> (from_blocks, block[, propagators, ...])	A layer that performs CRF decoding.

Sigmoid(from_blocks: List[*argparse.Namespace*], block: *argparse.Namespace*, propagators: Optional[dict] = None, ext_vars: dict = {}, cwd: Optional[str] = None) =

<narchi.propagators.same.SameShapePropagator object>

Block that applies a sigmoid function.

LogSigmoid(*from_blocks: List[argparse.Namespace]*, *block: argparse.Namespace*, *propagators: Optional[dict] = None*, *ext_vars: dict = {}*, *cwd: Optional[str] = None*) =
<narchi.propagators.same.SameShapePropagator object>

Block that applies a log-sigmoid function.

Softmax(*from_blocks: List[argparse.Namespace]*, *block: argparse.Namespace*, *propagators: Optional[dict] = None*, *ext_vars: dict = {}*, *cwd: Optional[str] = None*) =
<narchi.propagators.same.SameShapePropagator object>

Block that applies a softmax function.

LogSoftmax(*from_blocks: List[argparse.Namespace]*, *block: argparse.Namespace*, *propagators: Optional[dict] = None*, *ext_vars: dict = {}*, *cwd: Optional[str] = None*) =
<narchi.propagators.same.SameShapePropagator object>

Block that applies a log-softmax function.

Tanh(*from_blocks: List[argparse.Namespace]*, *block: argparse.Namespace*, *propagators: Optional[dict] = None*, *ext_vars: dict = {}*, *cwd: Optional[str] = None*) =
<narchi.propagators.same.SameShapePropagator object>

Block that applies a hyperbolic tangent function.

ReLU(*from_blocks: List[argparse.Namespace]*, *block: argparse.Namespace*, *propagators: Optional[dict] = None*, *ext_vars: dict = {}*, *cwd: Optional[str] = None*) =
<narchi.propagators.same.SameShapePropagator object>

Block that applies a rectified linear unit function.

LeakyReLU(*from_blocks: List[argparse.Namespace]*, *block: argparse.Namespace*, *propagators: Optional[dict] = None*, *ext_vars: dict = {}*, *cwd: Optional[str] = None*) =
<narchi.propagators.same.SameShapePropagator object>

Block that applies a leaky rectified linear unit function.

Dropout(*from_blocks: List[argparse.Namespace]*, *block: argparse.Namespace*, *propagators: Optional[dict] = None*, *ext_vars: dict = {}*, *cwd: Optional[str] = None*) =
<narchi.propagators.same.SameShapePropagator object>

Block that applies dropout, randomly set elements to zero.

BatchNorm2d(*from_blocks: List[argparse.Namespace]*, *block: argparse.Namespace*, *propagators: Optional[dict] = None*, *ext_vars: dict = {}*, *cwd: Optional[str] = None*) =
<narchi.propagators.same.SameShapePropagator object>

Block that does 2D batch normalization.

Identity(*from_blocks: List[argparse.Namespace]*, *block: argparse.Namespace*, *propagators: Optional[dict] = None*, *ext_vars: dict = {}*, *cwd: Optional[str] = None*) =
<narchi.propagators.same.SameShapePropagator object>

Block that does nothing, useful to connect one tensor to multiple blocks in a graph.

Add(*from_blocks: List[argparse.Namespace]*, *block: argparse.Namespace*, *propagators: Optional[dict] = None*, *ext_vars: dict = {}*, *cwd: Optional[str] = None*) =
<narchi.propagators.same.SameShapesPropagator object>

Block that adds the values of all input tensors. Input tensors must have the same shape.

CRF(*from_blocks: List[argparse.Namespace]*, *block: argparse.Namespace*, *propagators: Optional[dict] = None*, *ext_vars: dict = {}*, *cwd: Optional[str] = None*) =
<narchi.propagators.same.SameShapeConsumeDimPropagator object>

A layer that performs CRF decoding.

class narchi.blocks.ConcatBlocksEnum(*value*)

Bases: [enum.Enum](#)

Enum of blocks that concatenate multiple inputs.

Attributes:

<code>Concatenate</code> (from_blocks, block[, ...])	Block that concatenates multiple inputs of the same shape along a given dimension.
--	--

Concatenate(from_blocks: List[*argparse.Namespace*], block: *argparse.Namespace*, propagators: Optional[dict] = None, ext_vars: dict = {}, cwd: Optional[str] = None) =
<narchi.propagators.concat.ConcatenatePropagator object>
 Block that concatenates multiple inputs of the same shape along a given dimension.

class narchi.blocks.FixedOutputBlocksEnum(*value*)

Bases: `enum.Enum`

Enum of blocks that have fixed outputs.

Attributes:

<code>Linear</code> (from_blocks, block[, propagators, ...])	Linear transformation to the last dimension of input tensor.
<code>Embedding</code> (from_blocks, block[, propagators, ...])	A lookup table that retrieves embeddings of a fixed size.
<code>AdaptiveAvgPool1d</code> (from_blocks, block[, ...])	1D adaptive average pooling over input.
<code>AdaptiveAvgPool2d</code> (from_blocks, block[, ...])	2D adaptive average pooling over input.

Linear(from_blocks: List[*argparse.Namespace*], block: *argparse.Namespace*, propagators: Optional[dict] = None, ext_vars: dict = {}, cwd: Optional[str] = None) =
<narchi.propagators.fixed.FixedOutputPropagator object>
 Linear transformation to the last dimension of input tensor.

Embedding(from_blocks: List[*argparse.Namespace*], block: *argparse.Namespace*, propagators: Optional[dict] = None, ext_vars: dict = {}, cwd: Optional[str] = None) =
<narchi.propagators.fixed.AddFixedPropagator object>
 A lookup table that retrieves embeddings of a fixed size.

AdaptiveAvgPool1d(from_blocks: List[*argparse.Namespace*], block: *argparse.Namespace*, propagators: Optional[dict] = None, ext_vars: dict = {}, cwd: Optional[str] = None) =
<narchi.propagators.fixed.FixedOutputPropagator object>
 1D adaptive average pooling over input.

AdaptiveAvgPool2d(from_blocks: List[*argparse.Namespace*], block: *argparse.Namespace*, propagators: Optional[dict] = None, ext_vars: dict = {}, cwd: Optional[str] = None) =
<narchi.propagators.fixed.FixedOutputPropagator object>
 2D adaptive average pooling over input.

class narchi.blocks.ConvBlocksEnum(*value*)

Bases: `enum.Enum`

Enum of convolution-style blocks.

Attributes:

<code>Conv1d</code> (from_blocks, block[, propagators, ...])	1D convolution.
<code>Conv2d</code> (from_blocks, block[, propagators, ...])	2D convolution.
<code>Conv3d</code> (from_blocks, block[, propagators, ...])	3D convolution.
<code>MaxPool1d</code> (from_blocks, block[, propagators, ...])	1D maximum pooling.
<code>MaxPool2d</code> (from_blocks, block[, propagators, ...])	2D maximum pooling.

continues on next page

Table 6 – continued from previous page

<code>MaxPool3d</code> (<code>from_blocks</code> , <code>block</code> [, <code>propagators</code> , ...])	3D maximum pooling.
<code>AvgPool1d</code> (<code>from_blocks</code> , <code>block</code> [, <code>propagators</code> , ...])	1D average pooling.
<code>AvgPool2d</code> (<code>from_blocks</code> , <code>block</code> [, <code>propagators</code> , ...])	2D average pooling.
<code>AvgPool3d</code> (<code>from_blocks</code> , <code>block</code> [, <code>propagators</code> , ...])	3D average pooling.

Conv1d(`from_blocks`: `List[argparse.Namespace]`, `block`: `argparse.Namespace`, `propagators`: `Optional[dict] = None`, `ext_vars`: `dict = {}`, `cwd`: `Optional[str] = None`) =

<narchi.propagators.conv.ConvPropagator object>

1D convolution.

Conv2d(`from_blocks`: `List[argparse.Namespace]`, `block`: `argparse.Namespace`, `propagators`: `Optional[dict] = None`, `ext_vars`: `dict = {}`, `cwd`: `Optional[str] = None`) =

<narchi.propagators.conv.ConvPropagator object>

2D convolution.

Conv3d(`from_blocks`: `List[argparse.Namespace]`, `block`: `argparse.Namespace`, `propagators`: `Optional[dict] = None`, `ext_vars`: `dict = {}`, `cwd`: `Optional[str] = None`) =

<narchi.propagators.conv.ConvPropagator object>

3D convolution.

MaxPool1d(`from_blocks`: `List[argparse.Namespace]`, `block`: `argparse.Namespace`, `propagators`: `Optional[dict] = None`, `ext_vars`: `dict = {}`, `cwd`: `Optional[str] = None`) =

<narchi.propagators.conv.PoolPropagator object>

1D maximum pooling.

MaxPool2d(`from_blocks`: `List[argparse.Namespace]`, `block`: `argparse.Namespace`, `propagators`: `Optional[dict] = None`, `ext_vars`: `dict = {}`, `cwd`: `Optional[str] = None`) =

<narchi.propagators.conv.PoolPropagator object>

2D maximum pooling.

MaxPool3d(`from_blocks`: `List[argparse.Namespace]`, `block`: `argparse.Namespace`, `propagators`: `Optional[dict] = None`, `ext_vars`: `dict = {}`, `cwd`: `Optional[str] = None`) =

<narchi.propagators.conv.PoolPropagator object>

3D maximum pooling.

AvgPool1d(`from_blocks`: `List[argparse.Namespace]`, `block`: `argparse.Namespace`, `propagators`: `Optional[dict] = None`, `ext_vars`: `dict = {}`, `cwd`: `Optional[str] = None`) =

<narchi.propagators.conv.PoolPropagator object>

1D average pooling.

AvgPool2d(`from_blocks`: `List[argparse.Namespace]`, `block`: `argparse.Namespace`, `propagators`: `Optional[dict] = None`, `ext_vars`: `dict = {}`, `cwd`: `Optional[str] = None`) =

<narchi.propagators.conv.PoolPropagator object>

2D average pooling.

AvgPool3d(`from_blocks`: `List[argparse.Namespace]`, `block`: `argparse.Namespace`, `propagators`: `Optional[dict] = None`, `ext_vars`: `dict = {}`, `cwd`: `Optional[str] = None`) =

<narchi.propagators.conv.PoolPropagator object>

3D average pooling.

class `narchi.blocks.RnnBlocksEnum`(`value`)

Bases: `enum.Enum`

Enum of recurrent-style blocks.

Attributes:

<i>RNN</i> (from_blocks, block[, propagators, ...])	A simple recurrent block.
<i>LSTM</i> (from_blocks, block[, propagators, ...])	An LSTM recurrent block.
<i>GRU</i> (from_blocks, block[, propagators, ...])	A GRU recurrent block.

RNN(from_blocks: List[*argparse.Namespace*], block: *argparse.Namespace*, propagators: Optional[dict] = None, ext_vars: dict = {}, cwd: Optional[str] = None) =
<narchi.propagators.rnn.RnnPropagator object>
 A simple recurrent block.

LSTM(from_blocks: List[*argparse.Namespace*], block: *argparse.Namespace*, propagators: Optional[dict] = None, ext_vars: dict = {}, cwd: Optional[str] = None) =
<narchi.propagators.rnn.RnnPropagator object>
 An LSTM recurrent block.

GRU(from_blocks: List[*argparse.Namespace*], block: *argparse.Namespace*, propagators: Optional[dict] = None, ext_vars: dict = {}, cwd: Optional[str] = None) =
<narchi.propagators.rnn.RnnPropagator object>
 A GRU recurrent block.

class narchi.blocks.**ReshapeBlocksEnum**(value)

Bases: `enum.Enum`

Enum of blocks that transform the shape.

Attributes:

<i>Reshape</i> (from_blocks, block[, propagators, ...])	Transformation of the shape of the input.
---	---

Reshape(from_blocks: List[*argparse.Namespace*], block: *argparse.Namespace*, propagators: Optional[dict] = None, ext_vars: dict = {}, cwd: Optional[str] = None) =
<narchi.propagators.reshape.ReshapePropagator object>
 Transformation of the shape of the input.

class narchi.blocks.**GroupPropagatorsEnum**(value)

Bases: `enum.Enum`

Enum of blocks that group other blocks.

Attributes:

<i>Sequential</i> (from_blocks, block[, ...])	Sequence of blocks that are connected in the given order.
<i>Group</i> (from_blocks, block[, propagators, ...])	Group of blocks with connected according to a given graph.
<i>Module</i> (from_blocks, block[, propagators, ...])	Definition of a complete module.

Sequential(from_blocks: List[*argparse.Namespace*], block: *argparse.Namespace*, propagators: Optional[dict] = None, ext_vars: dict = {}, cwd: Optional[str] = None) =
<narchi.propagators.group.SequentialPropagator object>
 Sequence of blocks that are connected in the given order.

Group(from_blocks: List[*argparse.Namespace*], block: *argparse.Namespace*, propagators: Optional[dict] = None, ext_vars: dict = {}, cwd: Optional[str] = None) =
<narchi.propagators.group.GroupPropagator object>
 Group of blocks with connected according to a given graph.

Module(*from_blocks: List[argparse.Namespace]*, *block: argparse.Namespace*, *propagators: Optional[dict] = None*, *ext_vars: dict = {}*, *cwd: Optional[str] = None*) = **<narchi.module.ModulePropagator object>**

Definition of a complete module.

`narchi.blocks.register_propagator(propagator, replace=False)`

Adds a propagator to the dictionary of registered propagators.

`narchi.blocks.register_known_propagators()`

Function that registers all propagators defined in the modules of the package.

3.3.2 narchi.graph

Functions related to the parsing of graphs.

Functions:

`digraph_from_graph_list(graph_list)`

`parse_graph(from_blocks, block)`

Parses a graph of a block.

`narchi.graph.digraph_from_graph_list(graph_list)`

`narchi.graph.parse_graph(from_blocks, block)`

Parses a graph of a block.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to parse its graph.

Return type `Dict[str, List[str]]`

Returns Dictionary in topological order mapping node IDs to its respective input nodes IDs.

Raises

- **ValueError** – If there are problems parsing the graph.
- **ValueError** – If the graph is not directed and acyclic.
- **ValueError** – If topological sort does not include all nodes.

3.3.3 narchi.module

Classes related to neural network module architectures.

Classes:

`ModuleArchitecture([architecture, cfg, parser])`

Class for instantiating ModuleArchitecture objects.

`ModulePropagator(block_class)`

Propagator for complete modules.

class `narchi.module.ModuleArchitecture`(*architecture=None*, *cfg=None*, *parser=None*)

Bases: `object`

Class for instantiating ModuleArchitecture objects.

Attributes:

path

jsonnet

architecture

propagators

blocks

topological_predecessors

Methods:

<i>get_config_parser()</i>	Returns a ModuleArchitecture configuration parser.
<i>__init__</i> ([architecture, cfg, parser])	Initializer for ModuleArchitecture class.
<i>apply_config</i> (cfg)	Applies a configuration to the ModuleArchitecture instance.
<i>load_architecture</i> (architecture)	Loads an architecture file.
<i>validate</i> ()	Validates the architecture against the narchi or propagated schema.
<i>propagate</i> ()	Propagates the shapes of the neural network module architecture.
<i>write_json</i> (json_path)	Writes the current state of the architecture in json format to the given path.
<i>write_json_outdir</i> ()	Writes the current state of the architecture in to the configured output directory.

path = None**jsonnet = None****architecture = None****propagators = 'default'****blocks = None****topological_predecessors = None****static get_config_parser()**

Returns a ModuleArchitecture configuration parser.

__init__(architecture=None, cfg=None, parser=None)

Initializer for ModuleArchitecture class.

Parameters

- **architecture** (Union[str, Path, None]) – Path to a jsonnet architecture file.
- **cfg** (Union[str, dict, Namespace, None]) – Path to config file or config object.
- **parser** (Optional[ArgumentParser]) – Parser object in case it is an extension of `get_config_parser()`.

apply_config(cfg)

Applies a configuration to the ModuleArchitecture instance.

Parameters `cfg` (`Union[str, dict, Namespace]`) – Path to config file or config object.

`load_architecture(architecture)`

Loads an architecture file.

Parameters `architecture` (`Union[str, Path, None]`) – Path to a jsonnet architecture file.

`validate()`

Validates the architecture against the narchi or propagated schema.

`propagate()`

Propagates the shapes of the neural network module architecture.

`write_json(json_path)`

Writes the current state of the architecture in json format to the given path.

`write_json_outdir()`

Writes the current state of the architecture in to the configured output directory.

class `narchi.module.ModulePropagator(block_class)`

Bases: `narchi.propagators.base.BasePropagator`

Propagator for complete modules.

Attributes:

`num_input_blocks`

Methods:

<code>propagate(from_blocks, block[, propagators, ...])</code>	Method that propagates shapes through a module.
<code>connect_input(from_blocks, block, module)</code>	Checks fixed dimensions agree and replaces the modules's variable dimensions.

`num_input_blocks = 1`

`propagate(from_blocks, block, propagators=None, ext_vars={}, cwd=None)`

Method that propagates shapes through a module.

Parameters

- `from_blocks` (`List[Namespace]`) – The input blocks.
- `block` (`Namespace`) – The block to propagate its shapes.
- `propagators` (`Optional[dict]`) – Dictionary of propagators.
- `ext_vars` (`Namespace`) – External variables required to load jsonnet.
- `cwd` (`Optional[str]`) – Working directory to resolve relative paths.

Raises `ValueError` – If no propagator found for some block.

static `connect_input(from_blocks, block, module)`

Checks fixed dimensions agree and replaces the modules's variable dimensions.

3.3.4 narchi.render

Classes related to rendering of architectures.

Classes:

<code>ModuleArchitectureRenderer([architecture, ...])</code>	Class for instantiating a <code>ModuleArchitectureRenderer</code> objects useful for creating module architecture diagrams.
--	---

class `narchi.render.ModuleArchitectureRenderer`(*architecture=None, cfg=None, parser=None*)

Bases: `narchi.module.ModuleArchitecture`

Class for instantiating a `ModuleArchitectureRenderer` objects useful for creating module architecture diagrams.

Methods:

<code>get_config_parser()</code>	Returns a <code>ModuleArchitectureRenderer</code> configuration parser.
<code>apply_config(cfg)</code>	Applies a configuration to the <code>ModuleArchitectureRenderer</code> instance.
<code>create_graph()</code>	Creates a <code>pygraphviz</code> graph of the architecture using the current configuration.
<code>render([architecture, out_render, cfg])</code>	Renders the architecture diagram optionally writing to the given file path.

static `get_config_parser()`

Returns a `ModuleArchitectureRenderer` configuration parser.

apply_config(*cfg*)

Applies a configuration to the `ModuleArchitectureRenderer` instance.

Parameters `cfg` (`Union[str, dict, Namespace]`) – Path to config file or config object.

create_graph()

Creates a `pygraphviz` graph of the architecture using the current configuration.

render(*architecture=None, out_render=None, cfg=None*)

Renders the architecture diagram optionally writing to the given file path.

Parameters

- **architecture** (`Union[str, Path, None]`) – Path to a jsonnet architecture file.
- **out_render** (`Union[str, Path, None]`) – Path where to write the rendered diagram with a valid extension for `pygraphviz` to determine the type.
- **cfg** (`Optional[Namespace]`) – Configuration to apply before rendering.

Returns `pygraphviz` graph object.

Return type `AGraph`

3.3.5 narchi.schemas

Definition of the narchi json schemas.

Classes:

<i>SchemasEnum</i> (value)	Enum of the schemas defined in narchi.
----------------------------	--

Functions:

<i>schema_as_str</i> ([schema])	Formats a schema as a pretty printed json string.
---------------------------------	---

class narchi.schemas.SchemasEnum(value)

Bases: enum.Enum

Enum of the schemas defined in narchi.

Attributes:

<i>narchi</i>	Main schema which defines the general format for architecture files.
<i>propagated</i>	Schema for architectures in which the dimensions have been propagated.
<i>reshape</i>	Schema that defines the format to specify reshaping of tensors.
<i>block</i>	Schema for a single architecture block.
<i>mappings</i>	Schema for mappings between architectures and block implementations.

```

narchi = {'$id': 'https://schema.omnium.com/json/narchi/1.0/schema.json', '$ref':
  '#/definitions/architecture', '$schema': 'http://json-schema.org/draft-07/schema#',
  'definitions': {'architecture': {'additionalProperties': False, 'properties':
    {'_description': {'$ref': '#/definitions/description'}, '_id': {'$ref':
      '#/definitions/id'}, 'blocks': {'$ref': '#/definitions/blocks'}, 'graph':
      {'$ref': '#/definitions/graph'}, 'inputs': {'$ref':
        '#/definitions/inputs_outputs'}, 'outputs': {'$ref':
          '#/definitions/inputs_outputs'}}}, 'required': ['_id', 'blocks', 'graph', 'inputs',
        'outputs'], 'type': 'object', 'block': {'allOf': [{'if': {'properties':
          {'_class': {'enum': ['Sequential', 'Group']}}}], 'then': {'required':
            ['blocks']}, 'else': {'not': {'required': ['blocks']}}}], {'if': {'properties':
              {'_class': {'const': 'Group'}}}], 'then': {'required': ['graph', 'input',
                'output']}, 'else': {'not': {'required': ['graph', 'input', 'output']}}}], {'if':
                {'properties': {'_class': {'const': 'Module'}}}], 'then': {'required':
                  ['_path']}, 'else': {'not': {'required': ['_path', '_ext_vars',
                    'architecture']}}}], {'if': {'properties': {'_class': {'const': 'Sequential'}}}],
                'else': {'properties': {'blocks': {'items': {'required': ['_id']}}}}}], {'if':
                {'properties': {'_class': {'const': 'Concatenate'}}}], 'then': {'required':
                  ['dim']}, {'if': {'properties': {'_class': {'const': 'Reshape'}}}], 'then':
                {'required': ['reshape_spec']}, 'else': {'not': {'required':
                  ['reshape_spec']}}}], 'properties': {'_class': {'$ref': '#/definitions/id'},
                '_description': {'$ref': '#/definitions/description'}, '_ext_vars': {'type':
                  'object'}, '_id': {'$ref': '#/definitions/id'}, '_id_share': {'$ref':
                    '#/definitions/id'}, '_name': {'$ref': '#/definitions/id'}, '_path': {'$ref':
                      '#/definitions/path'}, '_shape': {'$ref': '#/definitions/shape'}, 'architecture':
                      {'$ref': '#/definitions/architecture'}, 'blocks': {'$ref':
                        '#/definitions/blocks'}, 'dim': {'type': 'integer'}, 'graph': {'$ref':
                          '#/definitions/graph'}, 'input': {'$ref': '#/definitions/id'}, 'output': {'$ref':
                            '#/definitions/id'}, 'reshape_spec': {'$ref': '#/definitions/reshape'}}},
                'required': ['_class'], 'type': 'object', 'blocks': {'items': {'$ref':
                  '#/definitions/block'}, 'minItems': 1, 'type': 'array'}, 'description':
                {'minLength': 8, 'pattern': '^[^<>]+$', 'type': 'string'}, 'dims': {'items':
                  {'oneOf': [{'type': 'integer', 'minimum': 1}, {'type': 'string', 'pattern':
                    '^(<<variable:([-+/*0-9A-Za-z_]+)>>|<<auto>>)$'}]}], 'minItems': 1, 'type':
                    'array'}, 'dims_in': {'items': {'oneOf': [{'type': 'integer', 'minimum': 1},
                      {'type': 'string', 'pattern': '^(<<variable:([-+/*0-9A-Za-z_]+)>>|<<auto>>)$'},
                      {'type': 'null'}]}], 'minItems': 1, 'type': 'array'}, 'graph': {'items':
                      {'pattern': '^[A-Za-z_][0-9A-Za-z_]*( +-> +[A-Za-z_][0-9A-Za-z_]*)+$', 'type':
                        'string'}, 'minItems': 1, 'type': 'array'}, 'id': {'pattern':
                        '^[A-Za-z_][0-9A-Za-z_]*$', 'type': 'string'}, 'inputs_outputs': {'items':
                          {'additionalProperties': False, 'properties': {'_description': {'$ref':
                            '#/definitions/description'}, '_id': {'$ref': '#/definitions/id'}, '_shape':
                              {'$ref': '#/definitions/dims'}}}, 'required': ['_id', '_shape'], 'type':
                                'object'}, 'minItems': 1, 'type': 'array'}, 'path': {'pattern': '.+\\.jsonnet',
                                  'type': 'string'}, 'reshape': {'oneOf': [{'const': 'flatten'}, {'type':
                                    'array', 'minItems': 1, 'items': {'oneOf': [{'$ref':
                                      '#/definitions/reshape_index'}, {'$ref': '#/definitions/reshape_flatten'}, {'$ref':
                                        '#/definitions/reshape_unflatten'}]}]}]}], 'reshape_dims': {'items': {'oneOf':
                                  [{'type': 'integer', 'minimum': 1}, {'type': 'string', 'pattern':
                                    '^(<<variable:([-+/*0-9A-Za-z_]+)>>|<<auto>>)$'}]}], 'minItems': 2, 'type':
                                    'array'}, 'reshape_flatten': {'items': {'$ref': '#/definitions/reshape_index'},
                                      'minItems': 2, 'type': 'array'}, 'reshape_index': {'minimum': 0, 'type':
                                        'integer'}, 'reshape_unflatten': {'additionalProperties': False, 'maxProperties':
                                          1, 'minProperties': 1, 'patternProperties': {'^[0-9]+$': {'$ref':
                                            '#/definitions/reshape_dims'}}}, 'type': 'object'}, 'shape':
                                  {'additionalProperties': False, 'properties': {'in': {'$ref':
                                    '#/definitions/dims_in'}, 'out': {'$ref': '#/definitions/dims'}}}, 'required':
                                    ['in', 'out'], 'type': 'object'}}], 'title': 'Neural Network Module Architecture
                Schema'}

```

3.3. API Reference 31

Main schema which defines the general format for architecture files.

```

propagated = {'$ref': '#/definitions/architecture', '$schema':
'http://json-schema.org/draft-07/schema#', 'definitions': {'architecture':
{'additionalProperties': False, 'properties': {'_description': {'$ref':
'/definitions/description'}, '_id': {'$ref': '#/definitions/id'}, '_shape':
{'$ref': '#/definitions/shape'}, 'blocks': {'$ref': '#/definitions/blocks'},
'graph': {'$ref': '#/definitions/graph'}, 'inputs': {'$ref':
'/definitions/inputs_outputs'}, 'outputs': {'$ref':
'/definitions/inputs_outputs'}}}, 'required': ['_id', 'blocks', 'graph', 'inputs',
'outputs'], 'type': 'object', 'block': {'allOf': [{'if': {'properties':
{'_class': {'enum': ['Sequential', 'Group']}}}], 'then': {'required':
['blocks']}, 'else': {'not': {'required': ['blocks']}}}], {'if': {'properties':
{'_class': {'const': 'Group'}}}], 'then': {'required': ['graph', 'input',
'output']}, 'else': {'not': {'required': ['graph', 'input', 'output']}}}], {'if':
{'properties': {'_class': {'const': 'Module'}}}], 'then': {'required':
['_path']}, 'else': {'not': {'required': ['_path', '_ext_vars',
'architecture']}}}], {'if': {'properties': {'_class': {'const': 'Sequential'}}}],
'else': {'properties': {'blocks': {'items': {'required': ['_id']}}}}}], {'if':
{'properties': {'_class': {'const': 'Concatenate'}}}], 'then': {'required':
['dim']}, {'if': {'properties': {'_class': {'const': 'Reshape'}}}], 'then':
{'required': ['reshape_spec']}, 'else': {'not': {'required':
['reshape_spec']}}}], 'properties': {'_class': {'$ref': '#/definitions/id'},
'_description': {'$ref': '#/definitions/description'}, '_ext_vars': {'type':
'object'}, '_id': {'$ref': '#/definitions/id'}, '_id_share': {'$ref':
'/definitions/id'}, '_name': {'$ref': '#/definitions/id'}, '_path': {'$ref':
'/definitions/path'}, '_shape': {'$ref': '#/definitions/shape'}, 'architecture':
{'$ref': '#/definitions/architecture'}, 'blocks': {'$ref':
'/definitions/blocks'}, 'dim': {'type': 'integer'}, 'graph': {'$ref':
'/definitions/graph'}, 'input': {'$ref': '#/definitions/id'}, 'output': {'$ref':
'/definitions/id'}, 'reshape_spec': {'$ref': '#/definitions/reshape'}}},
'required': ['_class', '_shape'], 'type': 'object', 'blocks': {'items':
{'$ref': '#/definitions/block'}, 'minItems': 1, 'type': 'array'}, 'description':
{'minLength': 8, 'pattern': '^[^<>]+$', 'type': 'string'}, 'dims': {'items':
{'oneOf': [{'type': 'integer', 'minimum': 1}, {'type': 'string', 'pattern':
'^<<variable:([-+/*0-9A-Za-z_]+)>>$'}]}, 'minItems': 1, 'type': 'array'},
'dims_in': {'items': {'oneOf': [{'type': 'integer', 'minimum': 1}, {'type':
'string', 'pattern': '^(<<variable:([-+/*0-9A-Za-z_]+)>>|<<auto>>)$'}, {'type':
'null'}]}, 'minItems': 1, 'type': 'array'}, 'graph': {'items': {'pattern':
'^[A-Za-z_][0-9A-Za-z_-]*( +> +[A-Za-z_][0-9A-Za-z_-]*)+$', 'type': 'string'},
'minItems': 1, 'type': 'array'}, 'id': {'pattern': '^[A-Za-z_][0-9A-Za-z_-]*$',
'type': 'string'}, 'inputs_outputs': {'items': {'additionalProperties': False,
'properties': {'_description': {'$ref': '#/definitions/description'}, '_id':
{'$ref': '#/definitions/id'}, '_shape': {'$ref': '#/definitions/dims'}}},
'required': ['_id', '_shape'], 'type': 'object'}, 'minItems': 1, 'type':
'array'}, 'path': {'pattern': '.+\\.jsonnet', 'type': 'string'}, 'reshape':
{'oneOf': [{'const': 'flatten'}, {'type': 'array', 'minItems': 1, 'items':
{'oneOf': [{'$ref': '#/definitions/reshape_index'}, {'$ref':
'/definitions/reshape_flatten'}, {'$ref': '#/definitions/reshape_unflatten'}]}]}],
'reshape_dims': {'items': {'oneOf': [{'type': 'integer', 'minimum': 1},
{'type': 'string', 'pattern': '^(<<variable:([-+/*0-9A-Za-z_]+)>>|<<auto>>)$'}]},
'minItems': 2, 'type': 'array'}, 'reshape_flatten': {'items': {'$ref':
'/definitions/reshape_index'}, 'minItems': 2, 'type': 'array'}, 'reshape_index':
{'minimum': 0, 'type': 'integer'}, 'reshape_unflatten': {'additionalProperties':
False, 'maxProperties': 1, 'minProperties': 1, 'patternProperties': {'^[0-9]+$':
{'$ref': '#/definitions/reshape_dims'}}, 'type': 'object'}, 'shape':
{'additionalProperties': False, 'properties': {'in': {'$ref':
'/definitions/dims_in'}, 'out': {'$ref': '#/definitions/dims'}}, 'required':
['_in', '_out'], 'type': 'object'}}], 'title': 'Neural Network Module Propagated
Architecture Schema'}

```

Schema for architectures in which the dimensions have been propagated.

```
reshape = {'$ref': '#/definitions/reshape', 'definitions': {'reshape': {'oneOf':
[{'const': 'flatten'}, {'type': 'array', 'minItems': 1, 'items': {'oneOf':
[{'$ref': '#/definitions/reshape_index'}, {'$ref':
'#/definitions/reshape_flatten'}, {'$ref': '#/definitions/reshape_unflatten'}]}]}]},
'reshape_dims': {'items': {'oneOf': [{'type': 'integer', 'minimum': 1},
{'type': 'string', 'pattern': '^(<<variable:([-+/*0-9A-Za-z_]+)>>|<<auto>>)$'}}],
'minItems': 2, 'type': 'array'}, 'reshape_flatten': {'items': {'$ref':
'#/definitions/reshape_index'}, 'minItems': 2, 'type': 'array'}, 'reshape_index':
{'minimum': 0, 'type': 'integer'}, 'reshape_unflatten': {'additionalProperties':
False, 'maxProperties': 1, 'minProperties': 1, 'patternProperties': {'^[0-9]+$':
{'$ref': '#/definitions/reshape_dims'}, 'type': 'object'}}}
```

Schema that defines the format to specify reshaping of tensors.

```

block = {'$ref': '#/definitions/block', 'definitions': {'architecture':
{'additionalProperties': False, 'properties': {'_description': {'$ref':
'#/definitions/description'}, '_id': {'$ref': '#/definitions/id'}, 'blocks':
{'$ref': '#/definitions/blocks'}, 'graph': {'$ref': '#/definitions/graph'},
'inputs': {'$ref': '#/definitions/inputs_outputs'}, 'outputs': {'$ref':
'#/definitions/inputs_outputs'}}}, 'required': ['_id', 'blocks', 'graph', 'inputs',
'outputs'], 'type': 'object', 'block': {'allOf': [{'if': {'properties':
{'_class': {'enum': ['Sequential', 'Group']}}}], 'then': {'required':
['blocks']}, 'else': {'not': {'required': ['blocks']}}}], {'if': {'properties':
{'_class': {'const': 'Group'}}}], 'then': {'required': ['graph', 'input',
'output']}, 'else': {'not': {'required': ['graph', 'input', 'output']}}}], {'if':
{'properties': {'_class': {'const': 'Module'}}}], 'then': {'required':
['_path']}, 'else': {'not': {'required': ['_path', '_ext_vars',
'architecture']}}}], {'if': {'properties': {'_class': {'const': 'Sequential'}}}],
'else': {'properties': {'blocks': {'items': {'required': ['_id']}}}}}], {'if':
{'properties': {'_class': {'const': 'Concatenate'}}}], 'then': {'required':
['dim']}, {'if': {'properties': {'_class': {'const': 'Reshape'}}}], 'then':
{'required': ['reshape_spec']}, 'else': {'not': {'required':
['reshape_spec']}}}], 'properties': {'_class': {'$ref': '#/definitions/id'},
'_description': {'$ref': '#/definitions/description'}, '_ext_vars': {'type':
'object'}, '_id': {'$ref': '#/definitions/id'}, '_id_share': {'$ref':
'#/definitions/id'}, '_name': {'$ref': '#/definitions/id'}, '_path': {'$ref':
'#/definitions/path'}, '_shape': {'$ref': '#/definitions/shape'}, 'architecture':
{'$ref': '#/definitions/architecture'}, 'blocks': {'$ref':
'#/definitions/blocks'}, 'dim': {'type': 'integer'}, 'graph': {'$ref':
'#/definitions/graph'}, 'input': {'$ref': '#/definitions/id'}, 'output': {'$ref':
'#/definitions/id'}, 'reshape_spec': {'$ref': '#/definitions/reshape'}}},
'required': ['_class'], 'type': 'object', 'blocks': {'items': {'$ref':
'#/definitions/block'}, 'minItems': 1, 'type': 'array'}, 'description':
{'minLength': 8, 'pattern': '^[\^<>]+$$', 'type': 'string'}, 'dims': {'items':
{'oneOf': [{'type': 'integer', 'minimum': 1}, {'type': 'string', 'pattern':
'^(<<variable:([-+/*0-9A-Za-z_]+)>>|<<auto>>)$'}]}], 'minItems': 1, 'type':
'array'}, 'dims_in': {'items': {'oneOf': [{'type': 'integer', 'minimum': 1},
{'type': 'string', 'pattern': '^(<<variable:([-+/*0-9A-Za-z_]+)>>|<<auto>>)$'},
{'type': 'null'}]}], 'minItems': 1, 'type': 'array'}, 'graph': {'items':
{'pattern': '^([A-Za-z_][0-9A-Za-z_-]*( +-> +[A-Za-z_][0-9A-Za-z_-]*)+$$', 'type':
'string'}, 'minItems': 1, 'type': 'array'}, 'id': {'pattern':
'^[A-Za-z_][0-9A-Za-z_-]*$', 'type': 'string'}, 'inputs_outputs': {'items':
{'additionalProperties': False, 'properties': {'_description': {'$ref':
'#/definitions/description'}, '_id': {'$ref': '#/definitions/id'}, '_shape':
{'$ref': '#/definitions/dims'}}}, 'required': ['_id', '_shape'], 'type':
'object'}, 'minItems': 1, 'type': 'array'}, 'path': {'pattern': '.+\\.jsonnet',
'type': 'string'}, 'reshape': {'oneOf': [{'const': 'flatten'}, {'type':
'array', 'minItems': 1, 'items': {'oneOf': [{'$ref':
'#/definitions/reshape_index'}, {'$ref': '#/definitions/reshape_flatten'}, {'$ref':
'#/definitions/reshape_unflatten'}]}]}], 'reshape_dims': {'items': {'oneOf':
[{'type': 'integer', 'minimum': 1}, {'type': 'string', 'pattern':
'^(<<variable:([-+/*0-9A-Za-z_]+)>>|<<auto>>)$'}]}], 'minItems': 2, 'type':
'array'}, 'reshape_flatten': {'items': {'$ref': '#/definitions/reshape_index'},
'minItems': 2, 'type': 'array'}, 'reshape_index': {'minimum': 0, 'type':
'integer'}, 'reshape_unflatten': {'additionalProperties': False, 'maxProperties':
1, 'minProperties': 1, 'patternProperties': {'^[0-9]+$': {'$ref':
'#/definitions/reshape_dims'}}}, 'type': 'object'}, 'shape':
{'additionalProperties': False, 'properties': {'in': {'$ref':
'#/definitions/dims_in'}, 'out': {'$ref': '#/definitions/dims'}}}, 'required':
['_in', 'out'], 'type': 'object'}}]

```

Schema for a single architecture block.

```
mappings = {'additionalProperties': False, 'minProperties': 1,
'patternProperties': {'^[A-Za-z_][0-9A-Za-z_]*$': {'properties':
{'additionalProperties': False, 'class': {'pattern': '^[A-Za-z_][0-9A-Za-z_]*$',
'type': 'string'}, 'kwargs': {'additionalProperties': False, 'patternProperties':
{'^:skip$': {'pattern': '^[A-Za-z_][0-9A-Za-z_]*$', 'type': 'string'},
'^[A-Za-z_][0-9A-Za-z_]*$': {'oneOf': [{'pattern': '^[A-Za-z_][0-9A-Za-z_]*$',
{'pattern': '^shape:in:(|-)[0-9]+$'}, {'pattern': '^const:str:[^:]+$'},
{'pattern': '^const:int:[0-9]+$'}, {'pattern': '^const:bool:(True|False)$'}]},
'type': 'string'}}}, 'type': 'object'}, 'required': ['class']}, 'type':
'object'}}
```

Schema for mappings between architectures and block implementations.

`narchi.schemas.schema_as_str(schema=None)`

Formats a schema as a pretty printed json string.

Parameters `schema` (Optional[str]) – The schema name to return among {'narchi', 'propagated', 'reshape', 'block', 'mappings'}.

Returns Pretty printed schema.

Return type str

3.3.6 narchi.sympy

Functions for symbolic operations.

Functions:

<code>is_valid_dim(value)</code>	Checks whether value is an int > 0 or str that follows <code>variable_regex.pattern</code> .
<code>sympify_variable(value)</code>	Returns the sympified object for the given value.
<code>get_nonrational_variable(value)</code>	Returns either an int or a string variable.
<code>variable_operate(value, operation)</code>	Performs a symbolic operation on a given value.
<code>variables_aggregate(values, operation)</code>	Performs a symbolic aggregation operation over all input values.
<code>sum(values)</code>	Performs a symbolic sum of all input values.
<code>prod(values)</code>	Performs a symbolic product of all input values.
<code>divide(numerator, denominator)</code>	Performs a symbolic division.
<code>conv_out_length(length, kernel, stride, ...)</code>	Performs a symbolic calculation of the output length of a convolution.

`narchi.sympy.is_valid_dim(value)`

Checks whether value is an int > 0 or str that follows `variable_regex.pattern`.

`narchi.sympy.sympify_variable(value)`

Returns the sympified object for the given value.

`narchi.sympy.get_nonrational_variable(value)`

Returns either an int or a string variable.

`narchi.sympy.variable_operate(value, operation)`

Performs a symbolic operation on a given value.

Parameters

- **value** (Union[str, int]) – The value to operate on, either an int or a variable, e.g. “<<vari-

able:W/2+H/4>>”.

- **operation** (`Union[str, int]`) – Operation to apply on value, either int or expression, e.g. “`__input__/3`”.

Return type `Union[str, int]`

Returns The result of the operation.

Raises **ValueError** –

- If operation is not int nor a valid expression. * If value is not an int or a string that follows `variable_regex.pattern`. * If value is not a valid expression or contains “`__input__`” as a free symbol.

`narchi.sympy.variables_aggregate(values, operation)`

Performs a symbolic aggregation operation over all input values.

Parameters

- **values** (`List[Union[str, int]]`) – List of values to operate on.
- **operation** (`str`) – One of ‘+’=sum, ‘*’=prod.

Return type `Union[str, int]`

Returns The result of the operation.

Raises **ValueError** – If any value is not an int or a string that follows `variable_regex.pattern`.

`narchi.sympy.sum(values)`

Performs a symbolic sum of all input values.

Parameters **values** (`List[Union[str, int]]`) – List of values to operate on.

Return type `Union[str, int]`

Returns The result of the operation.

Raises **ValueError** – If any value is not an int or a string that follows `variable_regex.pattern`.

`narchi.sympy.prod(values)`

Performs a symbolic product of all input values.

Parameters **values** (`List[Union[str, int]]`) – List of values to operate on.

Return type `Union[str, int]`

Returns The result of the operation.

Raises **ValueError** – If any value is not an int or a string that follows `variable_regex.pattern`.

`narchi.sympy.divide(numerator, denominator)`

Performs a symbolic division.

Parameters

- **numerator** (`Union[str, int]`) – Value for numerator.
- **denominator** (`Union[str, int]`) – Value for denominator.

Return type `Union[str, int]`

Returns The result of the operation.

Raises **ValueError** – If any value is not an int or a string that follows `variable_regex.pattern`.

`narchi.sympy.conv_out_length(length, kernel, stride, padding, dilation)`

Performs a symbolic calculation of the output length of a convolution.

Parameters

- **length** (`Union[str, int]`) – Length of the input, either an int or a variable.
- **kernel** (`int`) – Size of the kernel in the direction of length.
- **stride** (`int`) – Stride size in the direction of the length.
- **padding** (`int`) – Padding added at both sides in the direction of the length.
- **dilation** (`int`) – Dilation size in the direction of the length.

Return type `Union[str, int]`

Returns The result of the operation.

3.3.7 narchi.propagators.base

Base propagator class and related functions.

Functions:

<code>get_shape(key, shape)</code>	Gets the shape list for a given key among {'in','out'}.
<code>create_shape(shape_in[, shape_out])</code>	Creates a shape namespace with 'in' and 'out' attributes and copied shape arrays.
<code>set_shape_dim(key, shape, dim, val)</code>	Sets a value for a given dimension, shape and key ('in' or 'out').
<code>shapes_agree(shape_from, shape_to)</code>	Checks whether the output shape from a block agrees with input shape of another block.
<code>shape_has_auto(shape)</code>	Checks whether a shape has any <<auto>> values.
<code>check_output_feats_dims(output_feats_dims, ...)</code>	Checks the output_feats attribute of a block.

Classes:

<code>BasePropagator(block_class)</code>	Base class for block shapes propagation.
--	--

`narchi.propagators.base.get_shape(key, shape)`

Gets the shape list for a given key among {'in','out'}.

`narchi.propagators.base.create_shape(shape_in, shape_out=None)`

Creates a shape namespace with 'in' and 'out' attributes and copied shape arrays.

`narchi.propagators.base.set_shape_dim(key, shape, dim, val)`

Sets a value for a given dimension, shape and key ('in' or 'out').

`narchi.propagators.base.shapes_agree(shape_from, shape_to)`

Checks whether the output shape from a block agrees with input shape of another block.

`narchi.propagators.base.shape_has_auto(shape)`

Checks whether a shape has any <<auto>> values.

`narchi.propagators.base.check_output_feats_dims(output_feats_dims, block_class, block)`

Checks the output_feats attribute of a block.

class `narchi.propagators.base.BasePropagator(block_class)`

Bases: `object`

Base class for block shapes propagation.

Attributes:

num_input_blocks

output_feats_dims

block_class

Methods:

<i>__init__</i> (<i>block_class</i>)	Initializer for BasePropagator instance.
<i>initial_checks</i> (<i>from_blocks</i> , <i>block</i>)	Method that does some initial checks before propagation.
<i>propagate</i> (<i>from_blocks</i> , <i>block</i>)	Method that propagates shapes to a block.
<i>final_checks</i> (<i>from_blocks</i> , <i>block</i>)	Method that checks for problems after shapes have been propagated.
<i>__call__</i> (<i>from_blocks</i> , <i>block</i> [, <i>propagators</i> , ...])	Propagates shapes to the given block.

num_input_blocks = None**output_feats_dims = False***__init__*(*block_class*)

Initializer for BasePropagator instance.

Parameters *block_class* (*str*) – The name of the block class being propagated.**block_class = None***initial_checks*(*from_blocks*, *block*)

Method that does some initial checks before propagation.

Extensions of this method in derived classes should always call this base method. This base method implements the following checks:

- That the block class is the same as the one expected by the propagator.
- That the input shapes don't contain any <<auto>> values.
- If num_input_blocks is set and is an int, that there are exactly this number of input blocks.

Parameters

- **from_blocks** (*List*[*Namespace*]) – The input blocks.
- **block** (*Namespace*) – The block to propagate its shapes.

Raises

- **ValueError** – If block fails to validate against schema.
- **ValueError** – If block already has a *_shape* attribute.
- **ValueError** – If block.*_class* != *block_class*.
- **ValueError** – If input shape not present, invalid or contains <<auto>>.
- **ValueError** – If output_feats required by class and not present or invalid.
- **ValueError** – If len(*from_blocks*) != num_input_blocks.

propagate(*from_blocks*, *block*)

Method that propagates shapes to a block.

This base method should be implemented by all derived classes.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

Raises `NotImplementedError` – Always.

final_checks(*from_blocks*, *block*)

Method that checks for problems after shapes have been propagated.

This base method implements checking the output shape don't contain <<auto>> values and if there is only a single from_block, that the connecting shapes agree. Extensions of this method in derived classes should always call this base one.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

__call__(*from_blocks*, *block*, *propagators=None*, *ext_vars={}*, *cwd=None*)

Propagates shapes to the given block.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.
- **propagators** (`Optional[dict]`) – Dictionary of propagators.
- **ext_vars** (`dict`) – Dictionary of external variables required to load jsonnet.
- **cwd** (`Optional[str]`) – Working directory to resolve relative paths.

3.3.8 narchi.propagators.concat

Propagator classes for concatenating.

Classes:

<code>ConcatenatePropagator</code> (<i>block_class</i>)	Propagator for concatenating along a given dimension.
---	---

class `narchi.propagators.concat.ConcatenatePropagator`(*block_class*)

Bases: `narchi.propagators.base.BasePropagator`

Propagator for concatenating along a given dimension.

Attributes:

`num_input_blocks`

Methods:

<code>initial_checks</code> (<i>from_blocks</i> , <i>block</i>)	Method that does some initial checks before propagation.
<code>propagate</code> (<i>from_blocks</i> , <i>block</i>)	Method that propagates shapes to a block.

`num_input_blocks = '>1'`

`initial_checks`(*from_blocks*, *block*)

Method that does some initial checks before propagation.

Calls the base class checks and makes sure that the dim attribute is valid and agrees with the input dimensions.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

Raises `ValueError` – When block does not have a valid dim attribute that agrees with input dimensions.

`propagate`(*from_blocks*, *block*)

Method that propagates shapes to a block.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

3.3.9 narchi.propagators.conv

Propagator classes for convolution blocks.

Classes:

<code>ConvPropagator</code> (<i>block_class</i> , <i>conv_dims</i>)	Propagator for convolution style blocks.
<code>PoolPropagator</code> (<i>block_class</i> , <i>conv_dims</i>)	Propagator for pooling style blocks.

class `narchi.propagators.conv.ConvPropagator`(*block_class*, *conv_dims*)

Bases: `narchi.propagators.base.BasePropagator`

Propagator for convolution style blocks.

Attributes:

`num_input_blocks`

`num_features_source`

`conv_dims`

Methods:

`__init__`(*block_class*, *conv_dims*)

Initializer for ConvPropagator instance.

continues on next page

Table 31 – continued from previous page

<code>initial_checks</code> (<i>from_blocks</i> , <i>block</i>)	Method that does some initial checks before propagation.
<code>propagate</code> (<i>from_blocks</i> , <i>block</i>)	Method that propagates shapes to a block.

`num_input_blocks = 1`

`num_features_source = 'output_feats'`

`__init__`(*block_class*, *conv_dims*)

Initializer for ConvPropagator instance.

Parameters

- **block_class** (`str`) – The name of the block class being propagated.
- **conv_dims** (`int`) – Number of dimensions for the convolution.

Raises `ValueError` – If `conv_dims` not `int > 0`.

`conv_dims = None`

`initial_checks`(*from_blocks*, *block*)

Method that does some initial checks before propagation.

Calls the base class checks and makes sure that the input shape agrees with the convolution dimensions.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

Raises `ValueError` – When `conv_dims` does not agree with `from_block[0]._shape`.

`propagate`(*from_blocks*, *block*)

Method that propagates shapes to a block.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

Raises

- `ValueError` – When `block.output_feats` not valid.
- `NotImplementedError` – If `num_features_source` is not one of {"from_shape", "output_feats"}.

`class narchi.propagators.conv.PoolPropagator`(*block_class*, *conv_dims*)

Bases: `narchi.propagators.conv.ConvPropagator`

Propagator for pooling style blocks.

Attributes:

`num_features_source`

`num_features_source = 'from_shape'`

3.3.10 narchi.propagators.fixed

Propagator classes for fixed output blocks.

Classes:

<code>AddFixedPropagator(block_class[, fixed_dims])</code>	Propagator for blocks that adds fixed dimensions.
<code>FixedOutputPropagator(block_class[, ...])</code>	Propagator for fixed output size blocks.

class `narchi.propagators.fixed.AddFixedPropagator`(*block_class*, *fixed_dims=1*)

Bases: `narchi.propagators.base.BasePropagator`

Propagator for blocks that adds fixed dimensions.

Methods:

<code>__init__(block_class[, fixed_dims])</code>	Initializer for AddFixedPropagator instance.
<code>propagate</code> (<i>from_blocks</i> , <i>block</i>)	Method that propagates shapes to a block.

Attributes:

`fixed_dims`

`__init__(block_class, fixed_dims=1)`

Initializer for AddFixedPropagator instance.

Parameters

- **block_class** (`str`) – The name of the block class being propagated.
- **fixed_dims** (`int`) – Number of fixed dimensions.

Raises `ValueError` – If `fixed_dims` not `int > 0`.

`fixed_dims = 1`

`propagate`(*from_blocks*, *block*)

Method that propagates shapes to a block.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

class `narchi.propagators.fixed.FixedOutputPropagator`(*block_class*, *unfixed_dims='any'*,
fixed_dims=1)

Bases: `narchi.propagators.base.BasePropagator`

Propagator for fixed output size blocks.

Attributes:

`num_input_blocks`

`unfixed_dims`

continues on next page

Table 36 – continued from previous page

*output_feats_dims***Methods:**

<code>__init__(block_class[, unfixed_dims, fixed_dims])</code>	Initializer for FixedOutputPropagator instance.
<code>initial_checks(from_blocks, block)</code>	Method that does some initial checks before propagation.
<code>propagate(from_blocks, block)</code>	Method that propagates shapes to a block.

num_input_blocks = 1

`__init__(block_class, unfixed_dims='any', fixed_dims=1)`
 Initializer for FixedOutputPropagator instance.

Parameters

- **block_class** (`str`) – The name of the block class being propagated.
- **unfixed_dims** (`Union[int, str]`) – Number of unfixed dimensions.
- **fixed_dims** (`int`) – Number of fixed dimensions.

Raises

- **ValueError** – If `fixed_dims` not `int > 0`.
- **ValueError** – If `unfixed_dims` not “any” or `int > 0`.

unfixed_dims = 'any'**output_feats_dims = 1****initial_checks**(*from_blocks*, *block*)

Method that does some initial checks before propagation.

Calls the base class checks and makes sure that the input shape has at least (`fixed_dims+1`) dimensions if `unfixed_dims=="any"` or exactly (`fixed_dims+fixed_dims`) dimensions if `unfixed_dims` is `int`.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

Raises **ValueError** – When `fixed_dims` and `unfixed_dims` do not agree with `from_block[0]._shape`.

propagate(*from_blocks*, *block*)

Method that propagates shapes to a block.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

3.3.11 narchi.propagators.group

Propagator classes for groups of blocks.

Functions:

<code>get_blocks_dict(blocks)</code>	Function that creates a dictionary of blocks using <code>_id</code> as keys.
<code>add_ids_prefix(block, io_blocks[, skip_io])</code>	Adds to block id a prefix consisting of parent id and separator as defined in propagated schema.
<code>propagate_shapes(blocks_dict, ...[, skip_ids])</code>	Function that propagates shapes in blocks based on a connections mapping.

Classes:

<code>SequentialPropagator(block_class)</code>	Propagator for a sequence of blocks.
<code>GroupPropagator(block_class)</code>	Propagator for a sequence of blocks.

`narchi.propagators.group.get_blocks_dict(blocks)`

Function that creates a dictionary of blocks using `_id` as keys.

Parameters `blocks` (`List[dict]`) – List of blocks objects.

Return type `Dict[str, dict]`

Returns Dictionary of blocks.

`narchi.propagators.group.add_ids_prefix(block, io_blocks, skip_io=True)`

Adds to block id a prefix consisting of parent id and separator as defined in propagated schema.

`narchi.propagators.group.propagate_shapes(blocks_dict, topological_predecessors, propagators, ext_vars, cwd, skip_ids=None)`

Function that propagates shapes in blocks based on a connections mapping.

Parameters

- `blocks_dict` (`Dict[str, dict]`) – Dictionary of blocks.
- `topological_predecessors` (`Dict[str, List[str]]`) – Mapping of block IDs to its input blocks IDs.
- `propagators` (`dict`) – Dictionary of propagators.
- `ext_vars` (`dict`) – Dictionary of external variables required to load jsonnet.
- `cwd` (`str`) – Working directory to resolve relative paths.
- `skip_ids` (`Optional[set]`) – Blocks that should be skipped in propagation.

Raises

- `ValueError` – If there graph references an undefined block.
- `ValueError` – If no propagator found for some block.

`class narchi.propagators.group.SequentialPropagator(block_class)`

Bases: `narchi.propagators.base.BasePropagator`

Propagator for a sequence of blocks.

Attributes:

num_input_blocks

Methods:

propagate(from_blocks, block, propagators, ...) Method that propagates shapes in the given block.

num_input_blocks = 1

propagate(*from_blocks*, *block*, *propagators*, *ext_vars*, *cwd=None*)
Method that propagates shapes in the given block.

Parameters

- **from_blocks** (*List[Namespace]*) – The input blocks.
- **block** (*Namespace*) – The block to propagate its shapes.
- **propagators** (*dict*) – Dictionary of propagators.
- **ext_vars** (*dict*) – Dictionary of external variables required to load jsonnet.
- **cwd** (*Optional[str]*) – Working directory to resolve relative paths.

Raises

- **ValueError** – If there are multiple blocks with the same id.
- **ValueError** – If no propagator found for some block.

class `narchi.propagators.group.GroupPropagator`(*block_class*)
Bases: `narchi.propagators.group.SequentialPropagator`

Propagator for a sequence of blocks.

Methods:

propagate(from_blocks, block, propagators, ...) Method that propagates shapes in the given block.

propagate(*from_blocks*, *block*, *propagators*, *ext_vars*, *cwd=None*)
Method that propagates shapes in the given block.

Parameters

- **from_blocks** (*List[Namespace]*) – The input blocks.
- **block** (*Namespace*) – The block to propagate its shapes.
- **propagators** (*dict*) – Dictionary of propagators.
- **ext_vars** (*dict*) – Dictionary of external variables required to load jsonnet.
- **cwd** (*Optional[str]*) – Working directory to resolve relative paths.

Raises

- **ValueError** – If there are multiple blocks with the same id.
- **ValueError** – If there graph references an undefined block.
- **ValueError** – If no propagator found for some block.

3.3.12 narchi.propagators.reshape

Propagator classes for reshaping.

Functions:

<code>check_reshape_spec(reshape_spec)</code>	Checks that reshape_spec is valid according to schema, indexes range is valid and there is at most one <<auto>> in each unflatten.
<code>norm_reshape_spec(reshape_spec)</code>	Converts elements of a reshape_spec from Namespace to dict.

Classes:

<code>ReshapePropagator(block_class)</code>	Propagator for reshapping which could involve any of: permute, flatten and unflatten.
---	---

`narchi.propagators.reshape.check_reshape_spec(reshape_spec)`

Checks that reshape_spec is valid according to schema, indexes range is valid and there is at most one <<auto>> in each unflatten.

`narchi.propagators.reshape.norm_reshape_spec(reshape_spec)`

Converts elements of a reshape_spec from Namespace to dict.

class `narchi.propagators.reshape.ReshapePropagator(block_class)`

Bases: `narchi.propagators.base.BasePropagator`

Propagator for reshapping which could involve any of: permute, flatten and unflatten.

Attributes:

`num_input_blocks`

Methods:

<code>initial_checks(from_blocks, block)</code>	Method that does some initial checks before propagation.
<code>propagate(from_blocks, block)</code>	Method that propagates shapes to a block.

`num_input_blocks = 1`

initial_checks(*from_blocks*, *block*)

Method that does some initial checks before propagation.

Calls the base class checks and makes sure that the reshape_spec attribute is valid and agrees with the input dimensions.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

Raises `ValueError` – When block does not have a valid reshape_spec attribute that agrees with input dimensions.

propagate(*from_blocks*, *block*)
 Method that propagates shapes to a block.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

3.3.13 narchi.propagators.rnn

Propagator classes for recurrent blocks.

Classes:

<code>RnnPropagator</code> (<i>block_class</i>)	Propagator for recurrent style blocks.
---	--

class `narchi.propagators.rnn.RnnPropagator`(*block_class*)

Bases: `narchi.propagators.base.BasePropagator`

Propagator for recurrent style blocks.

Attributes:

`num_input_blocks`

`output_feats_dims`

Methods:

<code>initial_checks</code> (<i>from_blocks</i> , <i>block</i>)	Method that does some initial checks before propagation.
---	--

<code>propagate</code> (<i>from_blocks</i> , <i>block</i>)	Method that propagates shapes to a block.
--	---

`num_input_blocks = 1`

`output_feats_dims = 1`

initial_checks(*from_blocks*, *block*)

Method that does some initial checks before propagation.

Calls the base class checks and makes sure that the input shape has two dimensions and that block includes a valid `output_feats` attribute.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

Raises

- **ValueError** – When `block.output_feats` not valid.
- **ValueError** – When `len(from_block[0]._shape) != 2`.

propagate(*from_blocks*, *block*)

Method that propagates shapes to a block.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

Raises `ValueError` – When `bidirectional==True` and `output_feats` not even.

3.3.14 narchi.propagators.same

Propagator classes that preserve the same shape.

Classes:

<code>SameShapePropagator</code> (<code>block_class</code>)	Propagator for blocks in which the input and output shapes are the same.
<code>SameShapesPropagator</code> (<code>block_class</code>)	Propagator for blocks that receive multiple inputs of the same shape and preserves this shape.
<code>SameShapeConsumeDimPropagator</code> (<code>block_class</code>)	Propagator for blocks in which the output shape is the same as input except the last which is consumed.

class `narchi.propagators.same.SameShapePropagator`(`block_class`)

Bases: `narchi.propagators.base.BasePropagator`

Propagator for blocks in which the input and output shapes are the same.

Methods:

<code>initial_checks</code> (<code>from_blocks</code> , <code>block</code>)	Method that does some initial checks before propagation.
<code>propagate</code> (<code>from_blocks</code> , <code>block</code>)	Method that propagates shapes to a block.

initial_checks(`from_blocks`, `block`)

Method that does some initial checks before propagation.

Calls the base class checks and if multi-input makes sure that all inputs have the same shape and if not multi-input makes sure that there is only a single input block.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

Raises `ValueError` – When `multi_input==False` and `len(from_blocks) != 1`.

propagate(`from_blocks`, `block`)

Method that propagates shapes to a block.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

class `narchi.propagators.same.SameShapesPropagator`(`block_class`)

Bases: `narchi.propagators.same.SameShapePropagator`

Propagator for blocks that receive multiple inputs of the same shape and preserves this shape.

Attributes:

num_input_blocks

`num_input_blocks = '>1'`

`class narchi.propagators.same.SameShapeConsumeDimPropagator(block_class)`

Bases: `narchi.propagators.same.SameShapePropagator`

Propagator for blocks in which the output shape is the same as input except the last which is consumed.

Methods:

<code>initial_checks(<i>from_blocks</i>, <i>block</i>)</code>	Method that does some initial checks before propagation.
<code>propagate(<i>from_blocks</i>, <i>block</i>)</code>	Method that propagates shapes to a block.

`initial_checks(from_blocks, block)`

Method that does some initial checks before propagation.

Calls the base class checks and makes sure that the input has more than one dimension.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

Raises `ValueError` – When `len(input_shape) < 2`.

`propagate(from_blocks, block)`

Method that propagates shapes to a block.

Parameters

- **from_blocks** (`List[Namespace]`) – The input blocks.
- **block** (`Namespace`) – The block to propagate its shapes.

3.3.15 narchi.instantiators.common

Generic code for module architecture instantiators.

Functions:

<code>id_strip_parent_prefix(<i>value</i>)</code>	Removes the parent prefix from an id value.
<code>import_object(<i>name</i>)</code>	Function that returns a class in a module given its dot import statement.
<code>instantiate_block(<i>block_cfg</i>, ...)</code>	Function that instantiates a block given its narchi config and a mappings object.

`narchi.instantiators.common.id_strip_parent_prefix(value)`

Removes the parent prefix from an id value.

`narchi.instantiators.common.import_object(name)`

Function that returns a class in a module given its dot import statement.

`narchi.instantiators.common.instantiate_block(block_cfg, blocks_mappings, module_cfg)`

Function that instantiates a block given its narchi config and a mappings object.

3.3.16 `narchi.instantiators.pytorch`

3.3.17 `narchi.instantiators.pytorch_packed`

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

- narchi.blocks, 21
- narchi.graph, 26
- narchi.instantiators.common, 50
- narchi.module, 26
- narchi.propagators.base, 38
- narchi.propagators.concat, 40
- narchi.propagators.conv, 41
- narchi.propagators.fixed, 43
- narchi.propagators.group, 45
- narchi.propagators.reshape, 47
- narchi.propagators.rnn, 48
- narchi.propagators.same, 49
- narchi.render, 29
- narchi.schemas, 30
- narchi.sympy, 36

Symbols

`__call__()` (*narchi.propagators.base.BasePropagator* method), 40
`__init__()` (*narchi.module.ModuleArchitecture* method), 27
`__init__()` (*narchi.propagators.base.BasePropagator* method), 39
`__init__()` (*narchi.propagators.conv.ConvPropagator* method), 42
`__init__()` (*narchi.propagators.fixed.AddFixedPropagator* method), 43
`__init__()` (*narchi.propagators.fixed.FixedOutputPropagator* method), 44

A

`AdaptiveAvgPool1d` (*narchi.blocks.FixedOutputBlocksEnum* attribute), 23
`AdaptiveAvgPool2d` (*narchi.blocks.FixedOutputBlocksEnum* attribute), 23
`Add` (*narchi.blocks.SameShapeBlocksEnum* attribute), 22
`add_ids_prefix()` (in module *narchi.propagators.group*), 45
`AddFixedPropagator` (class in *narchi.propagators.fixed*), 43
`apply_config()` (*narchi.module.ModuleArchitecture* method), 27
`apply_config()` (*narchi.render.ModuleArchitectureRenderer* method), 29
`architecture` (*narchi.module.ModuleArchitecture* attribute), 27
`AvgPool1d` (*narchi.blocks.ConvBlocksEnum* attribute), 24
`AvgPool2d` (*narchi.blocks.ConvBlocksEnum* attribute), 24
`AvgPool3d` (*narchi.blocks.ConvBlocksEnum* attribute), 24

B

`BasePropagator` (class in *narchi.propagators.base*), 38

`BatchNorm2d` (*narchi.blocks.SameShapeBlocksEnum* attribute), 22
`block` (*narchi.schemas.SchemasEnum* attribute), 34
`block_class` (*narchi.propagators.base.BasePropagator* attribute), 39
`blocks` (*narchi.module.ModuleArchitecture* attribute), 27

C

`check_output_feats_dims()` (in module *narchi.propagators.base*), 38
`check_reshape_spec()` (in module *narchi.propagators.reshape*), 47
`ConcatBlocksEnum` (class in *narchi.blocks*), 22
`Concatenate` (*narchi.blocks.ConcatBlocksEnum* attribute), 23
`ConcatenatePropagator` (class in *narchi.propagators.concat*), 40
`connect_input()` (*narchi.module.ModulePropagator* static method), 28
`Conv1d` (*narchi.blocks.ConvBlocksEnum* attribute), 24
`Conv2d` (*narchi.blocks.ConvBlocksEnum* attribute), 24
`Conv3d` (*narchi.blocks.ConvBlocksEnum* attribute), 24
`conv_dims` (*narchi.propagators.conv.ConvPropagator* attribute), 42
`conv_out_length()` (in module *narchi.sympy*), 37
`ConvBlocksEnum` (class in *narchi.blocks*), 23
`ConvPropagator` (class in *narchi.propagators.conv*), 41
`create_graph()` (*narchi.render.ModuleArchitectureRenderer* method), 29
`create_shape()` (in module *narchi.propagators.base*), 38
`CRF` (*narchi.blocks.SameShapeBlocksEnum* attribute), 22

D

`digraph_from_graph_list()` (in module *narchi.graph*), 26
`divide()` (in module *narchi.sympy*), 37
`Dropout` (*narchi.blocks.SameShapeBlocksEnum* attribute), 22

E

Embedding (*narchi.blocks.FixedOutputBlocksEnum* attribute), 23

F

final_checks() (*narchi.propagators.base.BasePropagator* method), 40

fixed_dims (*narchi.propagators.fixed.AddFixedPropagator* attribute), 43

FixedOutputBlocksEnum (*class in narchi.blocks*), 23

FixedOutputPropagator (*class in narchi.propagators.fixed*), 43

G

get_blocks_dict() (*in module narchi.propagators.group*), 45

get_config_parser() (*narchi.module.ModuleArchitecture* static method), 27

get_config_parser() (*narchi.render.ModuleArchitectureRenderer* static method), 29

get_nonrational_variable() (*in module narchi.sympy*), 36

get_shape() (*in module narchi.propagators.base*), 38

Group (*narchi.blocks.GroupPropagatorsEnum* attribute), 25

GroupPropagator (*class in narchi.propagators.group*), 46

GroupPropagatorsEnum (*class in narchi.blocks*), 25

GRU (*narchi.blocks.RnnBlocksEnum* attribute), 25

I

id_strip_parent_prefix() (*in module narchi.instantiators.common*), 50

Identity (*narchi.blocks.SameShapeBlocksEnum* attribute), 22

import_object() (*in module narchi.instantiators.common*), 50

initial_checks() (*narchi.propagators.base.BasePropagator* method), 39

initial_checks() (*narchi.propagators.concat.ConcatenatePropagator* method), 41

initial_checks() (*narchi.propagators.conv.ConvPropagator* method), 42

initial_checks() (*narchi.propagators.fixed.FixedOutputPropagator* method), 44

initial_checks() (*narchi.propagators.reshape.ReshapePropagator* method), 47

initial_checks() (*narchi.propagators.rnn.RnnPropagator* method), 48

initial_checks() (*narchi.propagators.same.SameShapeConsumeDimPropagator* method), 50

initial_checks() (*narchi.propagators.same.SameShapePropagator* method), 49

instantiate_block() (*in module narchi.instantiators.common*), 50

is_valid_dim() (*in module narchi.sympy*), 36

J

jsonnet (*narchi.module.ModuleArchitecture* attribute), 27

L

LeakyReLU (*narchi.blocks.SameShapeBlocksEnum* attribute), 22

Linear (*narchi.blocks.FixedOutputBlocksEnum* attribute), 23

load_architecture() (*narchi.module.ModuleArchitecture* method), 28

LogSigmoid (*narchi.blocks.SameShapeBlocksEnum* attribute), 21

LogSoftmax (*narchi.blocks.SameShapeBlocksEnum* attribute), 22

LSTM (*narchi.blocks.RnnBlocksEnum* attribute), 25

M

mappings (*narchi.schemas.SchemasEnum* attribute), 36

MaxPool1d (*narchi.blocks.ConvBlocksEnum* attribute), 24

MaxPool2d (*narchi.blocks.ConvBlocksEnum* attribute), 24

MaxPool3d (*narchi.blocks.ConvBlocksEnum* attribute), 24

module

`narchi.blocks`, 21

`narchi.graph`, 26

`narchi.instantiators.common`, 50

`narchi.module`, 26

`narchi.propagators.base`, 38

`narchi.propagators.concat`, 40

`narchi.propagators.conv`, 41

`narchi.propagators.fixed`, 43

`narchi.propagators.group`, 45

`narchi.propagators.reshape`, 47

`narchi.propagators.rnn`, 48

- narchi.propagators.same, 49
 - narchi.render, 29
 - narchi.schemas, 30
 - narchi.sympy, 36
 - Module (*narchi.blocks.GroupPropagatorsEnum* attribute), 25
 - ModuleArchitecture (*class in narchi.module*), 26
 - ModuleArchitectureRenderer (*class in narchi.render*), 29
 - ModulePropagator (*class in narchi.module*), 28
- ## N
- narchi (*narchi.schemas.SchemasEnum* attribute), 30
 - narchi.blocks
 - module, 21
 - narchi.graph
 - module, 26
 - narchi.instantiators.common
 - module, 50
 - narchi.module
 - module, 26
 - narchi.propagators.base
 - module, 38
 - narchi.propagators.concat
 - module, 40
 - narchi.propagators.conv
 - module, 41
 - narchi.propagators.fixed
 - module, 43
 - narchi.propagators.group
 - module, 45
 - narchi.propagators.reshape
 - module, 47
 - narchi.propagators.rnn
 - module, 48
 - narchi.propagators.same
 - module, 49
 - narchi.render
 - module, 29
 - narchi.schemas
 - module, 30
 - narchi.sympy
 - module, 36
 - norm_reshape_spec() (*in module narchi.propagators.reshape*), 47
 - num_features_source (*narchi.propagators.conv.ConvPropagator* attribute), 42
 - num_features_source (*narchi.propagators.conv.PoolPropagator* attribute), 42
 - num_input_blocks (*narchi.module.ModulePropagator* attribute), 28
 - num_input_blocks (*narchi.propagators.base.BasePropagator* attribute), 39
 - num_input_blocks (*narchi.propagators.concat.ConcatenatePropagator* attribute), 41
 - num_input_blocks (*narchi.propagators.conv.ConvPropagator* attribute), 42
 - num_input_blocks (*narchi.propagators.fixed.FixedOutputPropagator* attribute), 44
 - num_input_blocks (*narchi.propagators.group.SequentialPropagator* attribute), 46
 - num_input_blocks (*narchi.propagators.reshape.ReshapePropagator* attribute), 47
 - num_input_blocks (*narchi.propagators.rnn.RnnPropagator* attribute), 48
 - num_input_blocks (*narchi.propagators.same.SameShapesPropagator* attribute), 50
- ## O
- output_feats_dims (*narchi.propagators.base.BasePropagator* attribute), 39
 - output_feats_dims (*narchi.propagators.fixed.FixedOutputPropagator* attribute), 44
 - output_feats_dims (*narchi.propagators.rnn.RnnPropagator* attribute), 48
- ## P
- parse_graph() (*in module narchi.graph*), 26
 - path (*narchi.module.ModuleArchitecture* attribute), 27
 - PoolPropagator (*class in narchi.propagators.conv*), 42
 - prod() (*in module narchi.sympy*), 37
 - propagate() (*narchi.module.ModuleArchitecture* method), 28
 - propagate() (*narchi.module.ModulePropagator* method), 28
 - propagate() (*narchi.propagators.base.BasePropagator* method), 39
 - propagate() (*narchi.propagators.concat.ConcatenatePropagator* method), 41
 - propagate() (*narchi.propagators.conv.ConvPropagator* method), 42
 - propagate() (*narchi.propagators.fixed.AddFixedPropagator* method), 43

propagate() (*narchi.propagators.fixed.FixedOutputPropagator* method), 44

propagate() (*narchi.propagators.group.GroupPropagator* method), 46

propagate() (*narchi.propagators.group.SequentialPropagator* method), 46

propagate() (*narchi.propagators.reshape.ReshapePropagator* method), 47

propagate() (*narchi.propagators.rnn.RnnPropagator* method), 48

propagate() (*narchi.propagators.same.SameShapeConsumeDimPropagator* method), 50

propagate() (*narchi.propagators.same.SameShapePropagator* method), 49

propagate_shapes() (in module *narchi.propagators.group*), 45

propagated (*narchi.schemas.SchemasEnum* attribute), 32

propagators (*narchi.module.ModuleArchitecture* attribute), 27

R

register_known_propagators() (in module *narchi.blocks*), 26

register_propagator() (in module *narchi.blocks*), 26

ReLU (*narchi.blocks.SameShapeBlocksEnum* attribute), 22

render() (*narchi.render.ModuleArchitectureRenderer* method), 29

Reshape (*narchi.blocks.ReshapeBlocksEnum* attribute), 25

reshape (*narchi.schemas.SchemasEnum* attribute), 34

ReshapeBlocksEnum (class in *narchi.blocks*), 25

ReshapePropagator (class in *narchi.propagators.reshape*), 47

RNN (*narchi.blocks.RnnBlocksEnum* attribute), 25

RnnBlocksEnum (class in *narchi.blocks*), 24

RnnPropagator (class in *narchi.propagators.rnn*), 48

S

SameShapeBlocksEnum (class in *narchi.blocks*), 21

SameShapeConsumeDimPropagator (class in *narchi.propagators.same*), 50

SameShapePropagator (class in *narchi.propagators.same*), 49

SameShapesPropagator (class in *narchi.propagators.same*), 49

schema_as_str() (in module *narchi.schemas*), 36

SchemasEnum (class in *narchi.schemas*), 30

Sequential (*narchi.blocks.GroupPropagatorsEnum* attribute), 25

SequentialPropagator (class in *narchi.propagators.group*), 45

set_shape_dim() (in module *narchi.propagators.base*), 38

shape_has_auto() (in module *narchi.propagators.base*), 38

shapes_agree() (in module *narchi.propagators.base*), 38

Sigmoid (*narchi.blocks.SameShapeBlocksEnum* attribute), 21

Softmax (*narchi.blocks.SameShapeBlocksEnum* attribute), 22

sum() (*narchi.module.narchi.sympy*), 37

simplify_variable() (in module *narchi.sympy*), 36

T

Tanh (*narchi.blocks.SameShapeBlocksEnum* attribute), 22

topological_predecessors (*narchi.module.ModuleArchitecture* attribute), 27

U

unfixed_dims (*narchi.propagators.fixed.FixedOutputPropagator* attribute), 44

V

validate() (*narchi.module.ModuleArchitecture* method), 28

variable_operate() (in module *narchi.sympy*), 36

variables_aggregate() (in module *narchi.sympy*), 37

W

write_json() (*narchi.module.ModuleArchitecture* method), 28

write_json_outdir() (*narchi.module.ModuleArchitecture* method), 28