narchi Documentation

Mauricio Villegas

Nov 04, 2021

CONTENTS

narchi is as 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.

CHAPTER

ONE

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.

CHAPTER

TWO

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 reimplementation of existing architectures.

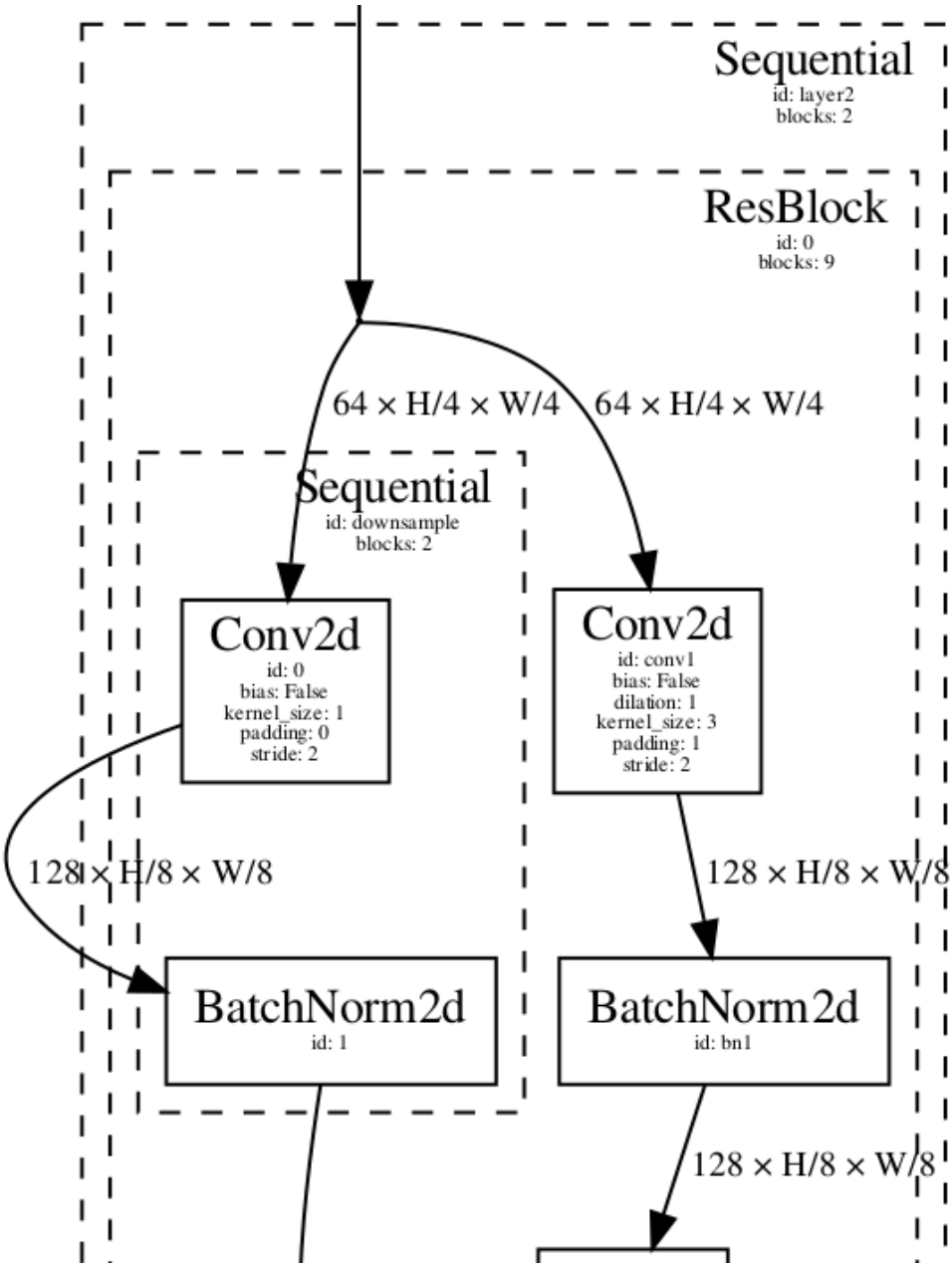
Instantiating a pytorch module from the architecture file can be easily done as follows.

```
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"
    }
  ]
}
```

CHAPTER

THREE

DOCUMENTATION CONTENTS

3.1 Json Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "https://schema.omnius.com/json/narchi/1.0/schema.json",
  "title": "Neural Network Module Architecture Schema",
  "$ref": "#/definitions/architecture",
  "definitions": {
    "id": {
      "type": "string",
      "pattern": "^[A-Za-z_][0-9A-Za-z_]*$"
    },
    "description": {
      "type": "string",
      "minLength": 8,
      "pattern": "^[^<>]+$"
    },
    "dims": {
      "type": "array",
      "minItems": 1,
      "items": {
        "oneOf": [
          {
            "type": "integer",
            "minimum": 1
          },
          {
            "type": "string",
            "pattern": "^(<<variable:([-+/*0-9A-Za-z_]+)>>|<<auto>>)$"
          }
        ]
      }
    },
    "dims_in": {
      "type": "array",
      "minItems": 1,
      "items": {
        "oneOf": [
          {
```

(continues on next page)

(continued from previous page)

```
            "type": "integer",
            "minimum": 1
          },
          {
            "type": "string",
            "pattern": "^(<<variable:([-+/*0-9A-Za-z_]+)>>|<<auto>>)$"
          },
          {
            "type": "null"
          }
        ]
      }
    },
    "shape": {
      "type": "object",
      "properties": {
        "in": {
          "$ref": "#/definitions/dims_in"
        },
        "out": {
          "$ref": "#/definitions/dims"
        }
      },
      "required": [
        "in",
        "out"
      ],
      "additionalProperties": false
    },
    "graph": {
      "type": "array",
      "minItems": 1,
      "items": {
        "type": "string",
        "pattern": "^[A-Za-z_][0-9A-Za-z_]*( +-> +[A-Za-z_][0-9A-Za-z_]*)+$"
      }
    },
    "block": {
      "type": "object",
      "properties": {
        "_class": {
          "$ref": "#/definitions/id"
        },
        "_name": {
          "$ref": "#/definitions/id"
        },
        "_id": {
          "$ref": "#/definitions/id"
        },
        "_id_share": {
          "$ref": "#/definitions/id"
        },
```

(continues on next page)

(continued from previous page)

```
        "_description": {
          "$ref": "#/definitions/description"
        },
        "_shape": {
          "$ref": "#/definitions/shape"
        },
        "_path": {
          "$ref": "#/definitions/path"
        },
        "_ext_vars": {
          "type": "object"
        },
        "blocks": {
          "$ref": "#/definitions/blocks"
        },
        "input": {
          "$ref": "#/definitions/id"
        },
        "output": {
          "$ref": "#/definitions/id"
        },
        "graph": {
          "$ref": "#/definitions/graph"
        },
        "dim": {
          "type": "integer"
        },
        "reshape_spec": {
          "$ref": "#/definitions/reshape"
        },
        "architecture": {
          "$ref": "#/definitions/architecture"
        }
      },
      "required": [
        "_class"
      ],
      "allOf": [
        {
          "if": {
            "properties": {
              "_class": {
                "enum": [
                  "Sequential",
                  "Group"
                ]
              }
            }
          },
          "then": {
            "required": [
              "blocks"
```

(continues on next page)

(continued from previous page)

```
            ]
          },
          "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",
```

(continues on next page)

(continued from previous page)

```
                "_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"
            ]
```

(continues on next page)

(continued from previous page)

```
          },
          "else": {
            "not": {
              "required": [
                "reshape_spec"
              ]
            }
          }
        }
      ]
    },
    "blocks": {
      "type": "array",
      "minItems": 1,
      "items": {
        "$ref": "#/definitions/block"
      }
    },
    "path": {
      "type": "string",
      "pattern": ".+\\.jsonnet"
    },
    "inputs_outputs": {
      "type": "array",
      "minItems": 1,
      "items": {
        "type": "object",
        "properties": {
          "_id": {
            "$ref": "#/definitions/id"
          },
          "_description": {
            "$ref": "#/definitions/description"
          },
          "_shape": {
            "$ref": "#/definitions/dims"
          }
        },
        "required": [
          "_id",
          "_shape"
        ],
        "additionalProperties": false
      }
    },
    "architecture": {
      "type": "object",
      "properties": {
        "_id": {
          "$ref": "#/definitions/id"
        },
        "_description": {
```

(continues on next page)

(continued from previous page)

```
          "$ref": "#/definitions/description"
        },
        "blocks": {
          "$ref": "#/definitions/blocks"
        },
        "graph": {
          "$ref": "#/definitions/graph"
        },
        "inputs": {
          "$ref": "#/definitions/inputs_outputs"
        },
        "outputs": {
          "$ref": "#/definitions/inputs_outputs"
        }
      },
      "required": [
        "_id",
        "blocks",
        "graph",
        "inputs",
        "outputs"
      ],
      "additionalProperties": false
    },
    "reshape": {
      "oneOf": [
        {
          "const": "flatten"
        },
        {
          "type": "array",
          "minItems": 1,
          "items": {
            "oneOf": [
              {
                "$ref": "#/definitions/reshape_index"
              },
              {
                "$ref": "#/definitions/reshape_flatten"
              },
              {
                "$ref": "#/definitions/reshape_unflatten"
              }
            ]
          }
        }
      ]
    },
    "reshape_index": {
      "type": "integer",
      "minimum": 0
    },
```

(continues on next page)

(continued from previous page)

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

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, penwidth=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:

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

Functions:

register_propagator(propagator[, replace])	Adds a propagator to the dictionary of registered propagators.
register_known_propagators()	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:

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

Concatenate(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:

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

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

continues on next page

Table 6 – continued from previous page

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

RNN(from_blocks, block[, propagators, ...])	A simple recurrent block.
LSTM(from_blocks, block[, propagators, ...])	An LSTM recurrent block.
GRU(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:

Reshape(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:

Sequential(from_blocks, block[, ...])	Sequence of blocks that are connected in the given order.
Group(from_blocks, block[, propagators, ...])	Group of blocks with connected according to a given graph.
Module(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:

get_config_parser()	Returns a ModuleArchitecture configuration parser.
__init__([architecture, cfg, parser])	Initializer for ModuleArchitecture class.
apply_config(cfg)	Applies a configuration to the ModuleArchitecture instance.
load_architecture(architecture)	Loads an 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.

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:

propagate(from_blocks, block[, propagators, ...])	Method that propagates shapes through a module.
connect_input(from_blocks, block, module)	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:

ModuleArchitectureRenderer([architecture, ...])	Class for instantiating a ModuleArchitectureRenderer 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:

get_config_parser()	Returns a ModuleArchitectureRenderer configuration parser.
apply_config(cfg)	Applies a configuration to the ModuleArchitectureRenderer instance.
create_graph()	Creates a pygraphviz graph of the architecture using the current configuration.
render([architecture, out_render, cfg])	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:

SchemasEnum(value)	Enum of the schemas defined in narchi.

Functions:

schema_as_str([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:

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

```
narchi = {'$id': 'https://schema.omnius.com/json/narchi/1.0/schema.json', '$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'}
```

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'}}, '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:

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

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

Classes:

BasePropagator(block_class)	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:

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

ConcatenatePropagator(block_class)	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:

initial_checks(from_blocks, block)	Method that does some initial checks before propagation.
propagate(from_blocks, block)	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:

ConvPropagator(block_class, conv_dims)	Propagator for convolution style blocks.
PoolPropagator(block_class, conv_dims)	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

initial_checks(from_blocks, block)	Method that does some initial checks before propagation.
propagate(from_blocks, block)	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:

AddFixedPropagator(block_class[, fixed_dims])	Propagator for blocks that adds fixed dimensions.
FixedOutputPropagator(block_class[, ...])	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:

__init__(block_class[, fixed_dims])	Initializer for AddFixedPropagator instance.
propagate(from_blocks, block)	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:

__init__(block_class[, unfixed_dims, fixed_dims])	Initializer for FixedOutputPropagator instance.
initial_checks(from_blocks, block)	Method that does some initial checks before propagation.
propagate(from_blocks, block)	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:

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

Classes:

SequentialPropagator(block_class)	Propagator for a sequence of blocks.
GroupPropagator(block_class)	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:

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.
norm_reshape_spec(reshape_spec)	Converts elements of a reshape_spec from Namespace to dict.

Classes:

ReshapePropagator(block_class)	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:

initial_checks(from_blocks, block)	Method that does some initial checks before propagation.
propagate(from_blocks, block)	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:

RnnPropagator(block_class)	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:

initial_checks(from_blocks, block)	Method that does some initial checks before propagation.
propagate(from_blocks, block)	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:

SameShapePropagator(block_class)	Propagator for blocks in which the input and output shapes are the same.
SameShapesPropagator(block_class)	Propagator for blocks that receive multiple inputs of the same shape and preserves this shape.
SameShapeConsumeDimPropagator(block_class)	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:

initial_checks(from_blocks, block)	Method that does some initial checks before propagation.
propagate(from_blocks, block)	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:

initial_checks(from_blocks, block)	Method that does some initial checks before propagation.
propagate(from_blocks, block)	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:

id_strip_parent_prefix(value)	Removes the parent prefix from an id value.
import_object(name)	Function that returns a class in a module given its dot import statement.
instantiate_block(block_cfg, ...)	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

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

INDEX

Symbols

A

B

C

D

E

F

G

I

J

L

M

N

O

P

R

S

T

U

V

W