viznet Documentation

Release 0.3.0

Jinguo Liu

Nov 13, 2019

Contents

viznet is a flexible framework based on matplotlib. it is intended for plotting neural networks, tensor networks and quantum circuits.

Our project repo is https://github.com/GiggleLiu/viznet

Contents

CHAPTER 1

Tutorial

1.1 Getting started

To start using viznet, simply

```
$ pip install viznet
```

or clone/download this repository and run

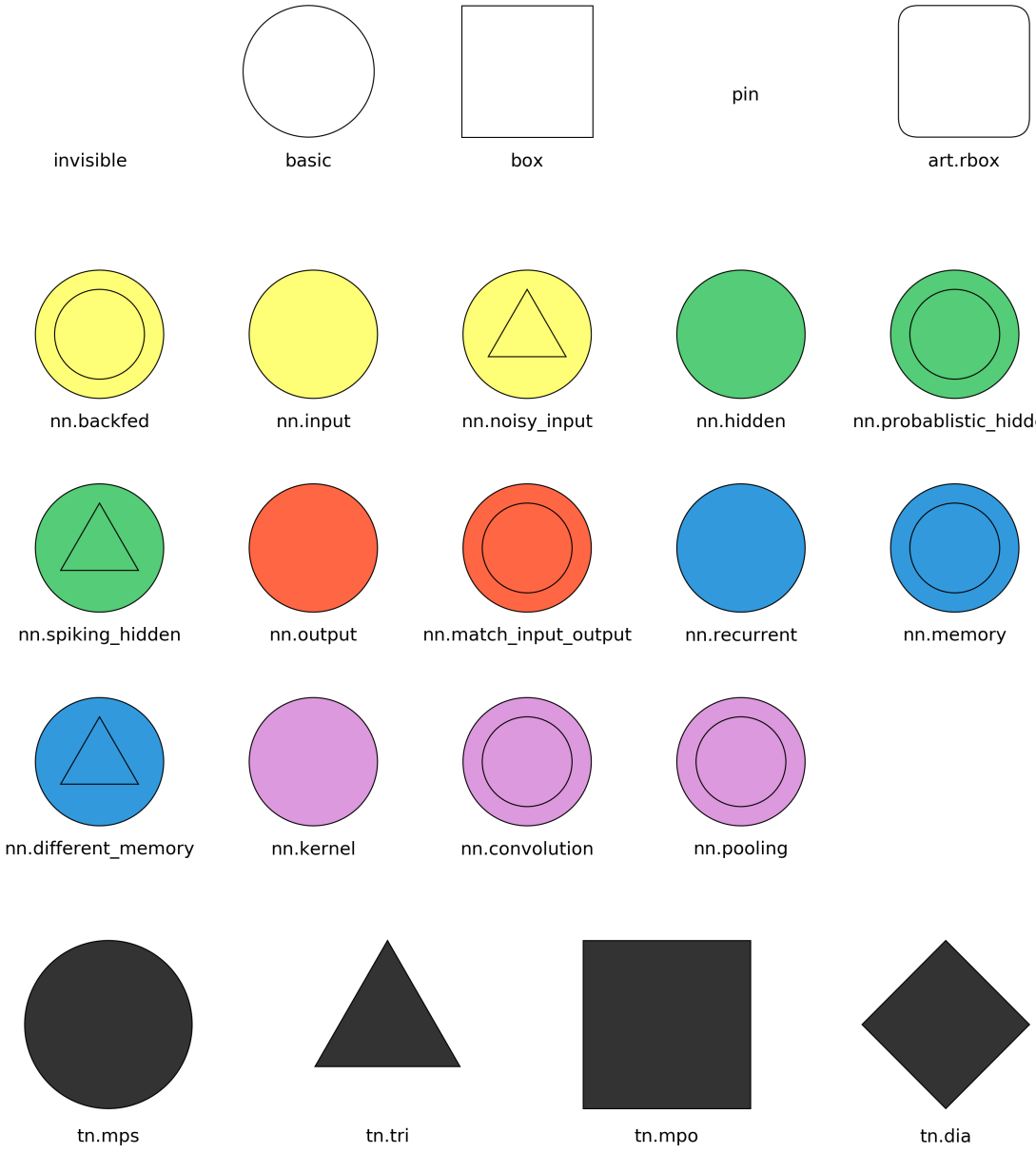
```
$ cd viznet/
$ pip install -r requirements.txt
$ python setup.py install
```

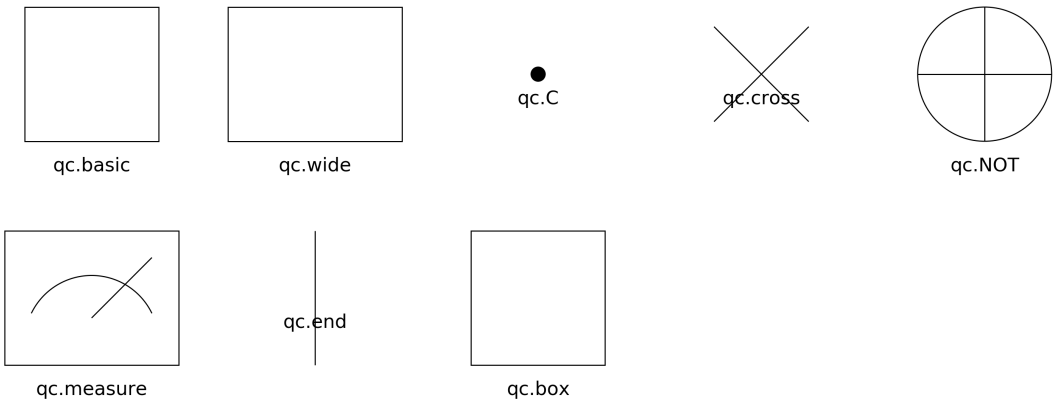
1.2 Node and Edge Brush

viznet focuses on node-edge based graphs. Instead of directly drawing nodes and edges, a brush system is used. The following code examplify how to draw two nodes and connection them using a directed edge.

```
>> from viznet import NodeBrush, EdgeBrush, DynamicShow
>> with DynamicShow() as d:
>>     brush = NodeBrush('nn.input', size='normal')
>      node1 = brush >> (1,0)  # paint a node at (x=1, y=0)
>>     node2 = brush >> (2,0)
>>     edge = EdgeBrush('->-', lw=2)
>>     edge >> (node1, node2)  # connect two nodes
>>     node1.text('First', 'center', fontsize=18)  # add text to node1
>>     node2.text('Second', 'center', fontsize=18)
```

`DynamicShow` is a utility class that automatically equalize axes and then remove axes to make graph clean. `NodeBrush` take the style string as its first argument, besides elementary styles like *basic* and *invisible* styles, styles for neural network (nn.) and tensor network (tn.) are defined as

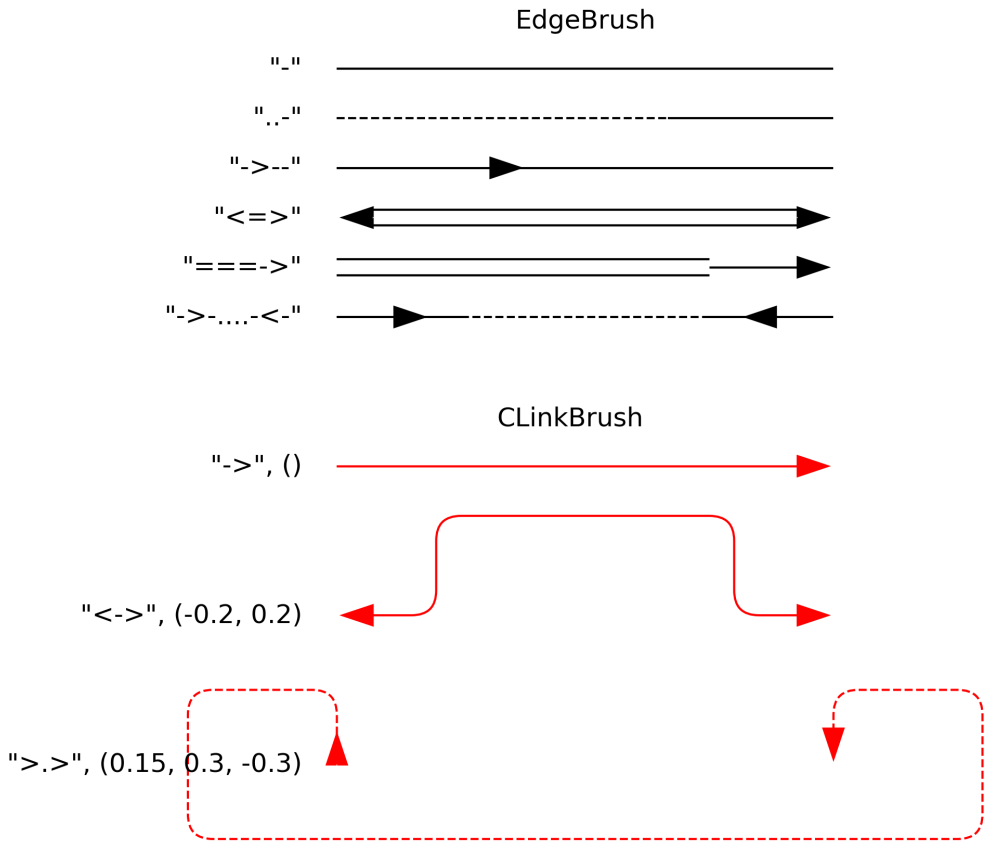

invisible
basic
box
pin
art.rbox
nn.backfed
nn.input
nn.noisy_input
nn.hidden
nn.probablistic_hidden
nn.spiking_hidden
nn.output
nn.match_input_output
nn.recurrent
nn.memory
nn.different_memory
nn.kernel
nn.convolution
nn.pooling
tn.mps
tn.tri
tn.mpo
tn.dia



`EdgeBrush` (or `CLinkBrush`, the curvy edge class) take a string as style, this must must be composed of characters in [- | . | =

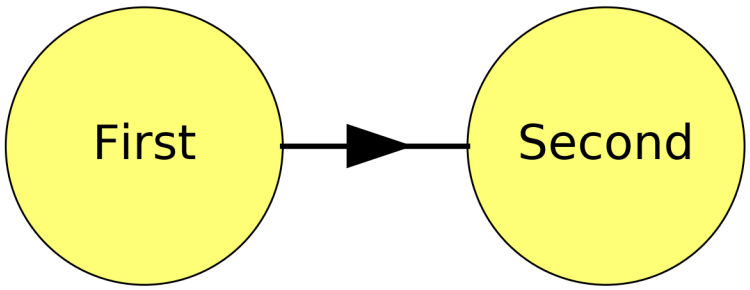
- '-': solid line,
- '=': double solid line,
- '.': dashed line,
- '>': arrow towards end of line, no length,
- '<': arrow towards start of line, no length.

For example,



In this example, the `CLinkBrush` instances use the roundness of *0.2* (default is *0*) to round the turning points, while the grow directions of lines are controled by offsets.

Also, you can set color and width of your line for this `EdgeBrush` by passing arguments into construction method.

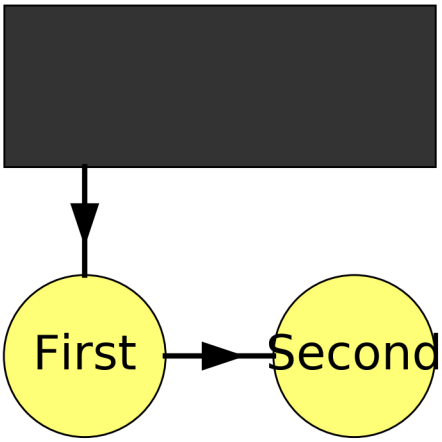


1.3 Pins

The above naive connection may not be what you want, pinning is needed. A pin is a special node, with no size, and is designed for connecting edges. Let's continue the above example,

```
>>      mpo21 = NodeBrush('tn.mpo', size='normal')
>>      mpo21.size = (0.7, 0.3)
>>      node3 = mpo21 >> (1.5, 1.0)
>>      left_bottom_pin = node3.pin('bottom', align=node1)
>>      edge >> (left_bottom_pin, node1)
```

Now, your canvas looks like



1.4 More

Cluster operations like one to one connections and all to all connections between different layers in neural network are frequently used APIs.

For a quantum circuit, we also have a facility *viznet.QuantumCircuit* to help us build it easily.

To learn more, you may go through `this notebook`.

Read some examples under *path/to/viznet/apps/nn/* and *path/to/viznet/apps/qc/* to learn about them. Also, *Examples* chapter of this documentation gives some examples.

CHAPTER 2

Examples

The first example is a feed forward network

```
import numpy as np
from viznet import connecta2a, node_sequence, NodeBrush, EdgeBrush, DynamicShow


def draw_feed_forward(ax, num_node_list):
    '''
    draw a feed forward neural network.

    Args:
        num_node_list (list<int>): number of nodes in each layer.
    '''
    num_hidden_layer = len(num_node_list) - 2
    token_list = ['\sigma^z'] + \
        ['y^{(%s)}' % (i + 1) for i in range(num_hidden_layer)] + ['\psi']
    kind_list = ['nn.input'] + ['nn.hidden'] * num_hidden_layer + ['nn.output']
    radius_list = [0.3] + [0.2] * num_hidden_layer + [0.3]
    y_list = 1.5 * np.arange(len(num_node_list))

    seq_list = []
    for n, kind, radius, y in zip(num_node_list, kind_list, radius_list, y_list):
        b = NodeBrush(kind, ax)
        seq_list.append(node_sequence(b, n, center=(0, y)))

    eb = EdgeBrush('-->', ax)
    for st, et in zip(seq_list[:-1], seq_list[1:]):
        connecta2a(st, et, eb)


def real_bp():
    with DynamicShow((6, 6), '_feed_forward.png') as d:
        draw_feed_forward(d.ax, num_node_list=[5, 4, 1])
```

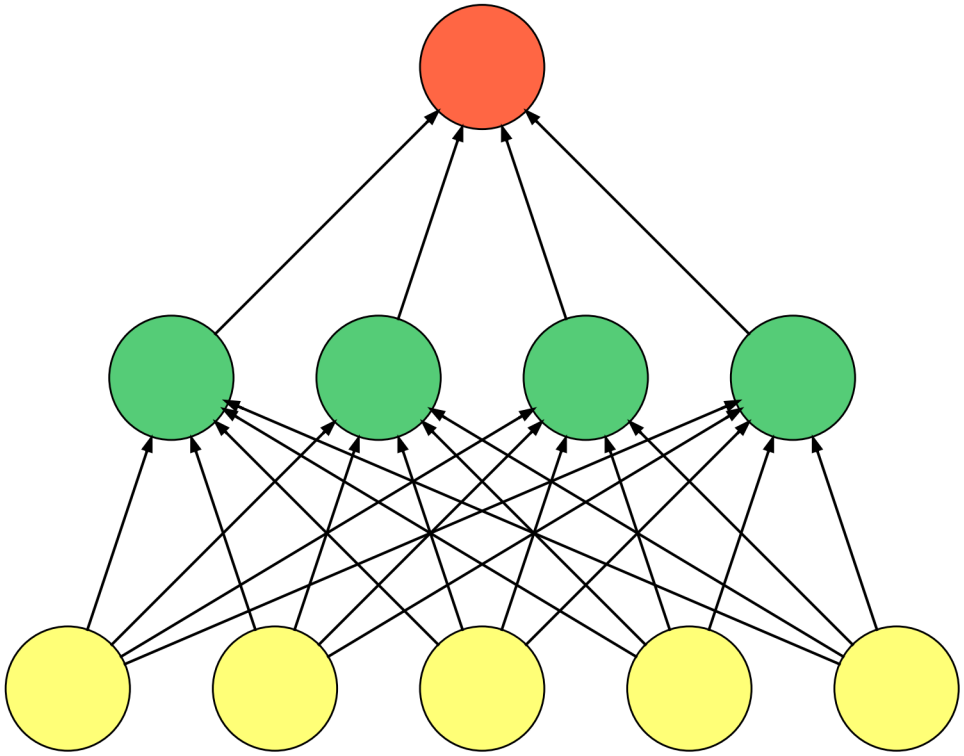
(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    real_bp()
```

```
$ python apps/nn/feed_foward.py
```

The output is



The second example is tensor network TEBD algorithm, it is also a good example to learn the grid system.

```
from viznet import theme, EdgeBrush, DynamicShow, Grid, NodeBrush


def tebd():
    # define a grid with grid space dx = 1, and dy = 1.
```

(continues on next page)

(continued from previous page)

```
    grid = Grid((1, 1))

    # define a set of brushes.
    # NodeBrush can place a node at some location, like `node_brush >> (x, y)`,
    # and it will return a Node instance.
    # EdgeBrush can connect two Nodes (or Pin as a special Node),
    # like `edge_brush >> node_a, node_b`, and will return an Edge instance.
    size = 'normal'
    mps = NodeBrush('tn.mps', size=size)
    # invisible node can be used as a placeholder
    invisible_mps = NodeBrush('invisible', size=size)
    # define a two site mpo, which will occupy 1 column, 0 rows.
    mpo21 = NodeBrush('tn.mpo', size=size)
    edge = EdgeBrush('-', lw=2.)

    with DynamicShow((6, 4), filename='_tebd.png') as ds:
        # add a sequence of mps nodes, a store them in a list for future use.
        mps_list = []
        for i in range(8):
            mps_list.append(mps >> grid[i, 0])
            mps_list[-1].text(r'$\sigma_%d$' % i, position='bottom')
        mps_list.append(invisible_mps >> grid[i + 1, 0])

        # add mpo and connect nodes
        for layer in range(4):
            # set brush color, it will overide theme color!
            # You can set brush color to None to restore theme color.
            mpo21.color = theme.RED if layer % 2 == 0 else theme.GREEN
            mpo_list = []
            start = layer % 2
            for i, (mps_l, mps_r) in enumerate(zip(mps_list[start::2],
                                                   mps_list[start + 1::2])):
                # place an two site mpo slightly above the center of two mps nodes
                y = mps_l.position[1]+layer + 1
                mpo_list.append(mpo21 >> grid[mps_l.position[0]:mps_r.position[0],␣
↪y:y])
                if layer == 0:
                    # if this is the first mpo layer, connect mps and newly added mpo.
                    pin_l = mps_l
                    pin_r = mps_r
                else:
                    # otherwise, place a pin at the top surface of previous mpo,
                    # we also require it horizontally aligned to some `mps_l` object.
                    # pin is a special node, which is zero sized,
                    # we can use it to connect nodes, add texts.
                    # if you're about to place some pin at `left` or
                    # `right` surface of a node,
                    # align is then intepreted as vertial align.
                    pin_l = mpo_list_pre[i].pin('top', align=mps_l)
                    pin_r = mpo_list_pre[i].pin('top', align=mps_r)
                if layer < 2:
                    edge >> (mps_l, mps_r)
                edge >> (pin_l, mpo_list[-1].pin('bottom', align=mps_l))
                edge >> (pin_r, mpo_list[-1].pin('bottom', align=mps_r))
            mpo_list_pre = mpo_list
```

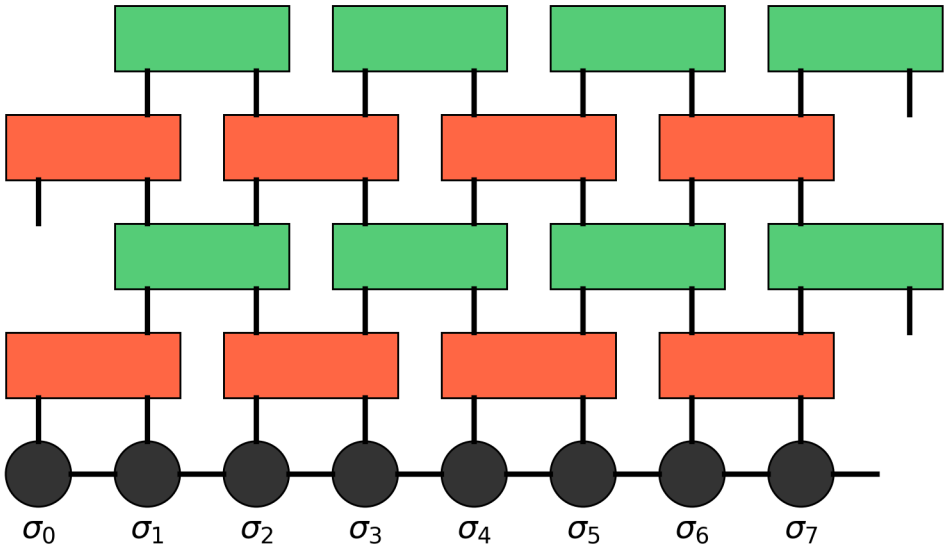
(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    tebd()
```

```
$ python apps/tn/tebd.py
```

The output is



The third example is a quantum circuit

```
import matplotlib.pyplot as plt
from viznet import DynamicShow, QuantumCircuit
from viznet import parsecircuit as _


def ghz4():
    '''4 bit GHZ circuit, applicable on ibmqx4 circuit.'''
    num_bit = 4
    with DynamicShow((5, 3), '_exact_ghz4_circuit.png') as ds:
        handler = QuantumCircuit(num_bit=4, y0=2.)
        handler.x += 0.5
        handler.gate(_.GATE, 0, 'X')
        for i in range(1, num_bit):
            handler.gate(_.GATE, i, 'H')
        handler.x += 1
        handler.gate((_.C, _.NOT), (1, 0))
        handler.gate((_.C, _.NOT), (3, 2))
        handler.x += 0.7
        handler.gate((_.C, _.NOT), (2, 0))
        handler.x += 0.7
```

(continues on next page)

(continued from previous page)

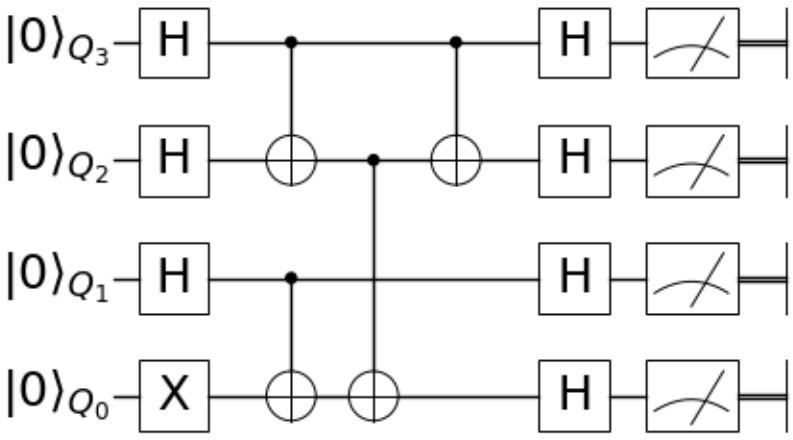
```
        handler.gate((_.C, _.NOT), (3, 2))
        handler.x += 1
        for i in range(num_bit):
            handler.gate(_.GATE, i, 'H')
        handler.x += 1
        for i in range(num_bit):
            handler.gate(_.MEASURE, i)
        handler.edge.style = '='
        handler.x += 0.8
        for i in range(num_bit):
            handler.gate(_.END, i)

        # text |0>s
        for i in range(num_bit):
            plt.text(*handler.get_position(i, x=-0.5), r'$\left\vert0\right\rangle_{Q_
↪%d}$' %
                     i, va='center', ha='center', fontsize=18)


if __name__ == '__main__':
    ghz4()
```

```
$ python apps/qc/ghz.py
```

The output is



Here, we used the `QuantumCircuit` instance `handler` to help us build the circuit. *gate* method of `handler` take brush(es) as first argument and line(lines) as second argument. *handler.x* decide the *x* axis of this gate.

CHAPTER 3

Code Documentation

Welcome to the package documentation of ProjectQ. You may now browse through the entire documentation and discover the capabilities of the ProjectQ framework.

For a detailed documentation of a subpackage or module, click on its name below:

3.1 viznet

3.1.1 Module contents

class viznet.**Brush**
Bases: object

Base Class of brushes.

class viznet.**CLinkBrush**(*style*, *ax=None*, *offsets=(0.2*, *)*, *roundness=0*, *lw=1*, *color='k'*, *zorder=0*, *solid_capstyle='butt'*)
Bases: viznet.brush.EdgeBrush

Brush for C type link.

style
e.g. '<->', right-side grow with respect to the line direction.

Type str

__init__(*style*, *ax=None*, *offsets=(0.2*, *)*, *roundness=0*, *lw=1*, *color='k'*, *zorder=0*, *solid_capstyle='butt'*)
Initialize self. See help(type(self)) for accurate signature.

class viznet.**CurveBrush**(*style*, *ax=None*, *lw=1*, *color='k'*, *zorder=0*, *solid_capstyle='butt'*, *ls='-'*)
Bases: viznet.brush.Brush

a brush for drawing edges.

style
the style of edge, same as arrowprops in https://matplotlib.org/api/_as_gen/matplotlib.pyplot.annotate.html.

Type str

ax
matplotlib Axes instance.

Type Axes

lw
line width.

Type float

color
the color of painted edge by this brush.

Type str

__init__(*style*, *ax=None*, *lw=1*, *color='k'*, *zorder=0*, *solid_capstyle='butt'*, *ls='-'*)
Initialize self. See help(type(self)) for accurate signature.

class viznet.**DynamicShow**(*figsize=(6, 4)*, *filename=None*, *dpi=300*, *fps=1*)
Bases: object

Dynamic plot context, intended for displaying geometries. like removing axes, equal axis, dynamically tune your figure and save it.

Parameters

- **figsize** (*tuple, default=(6,4)*) – figure size.
- **filename** (*filename, str*) – filename to store generated figure, if None, it will not save a figure.

figsize
figure size.

Type tuple, default=(6,4)

filename
filename to store generated figure, if None, it will not save a figure.

Type filename, str

ax
matplotlib Axes instance.

Type Axes

Examples

with DynamicShow() as ds: c = Circle([2, 2], radius=1.0) ds.ax.add_patch(c)

__init__(*figsize=(6, 4)*, *filename=None*, *dpi=300*, *fps=1*)
Initialize self. See help(type(self)) for accurate signature.

class viznet.**Edge**(*objs*, *start_xy*, *end_xy*, *start*, *end*, *brush*)
Bases: viznet.edgenode.EdgeNode

An Edge connecting two *EdgeNode* instance.

obj
matplotlib line object.

Type Patch

start_xy
start position.

Type tuple

end_xy
end position.

Type tuple

start
start node.

Type EdgeNode

end
end node.

Type EdgeNode

brush
brush.

Type *EdgeBrush*

__init__(*objs*, *start_xy*, *end_xy*, *start*, *end*, *brush*)
Initialize self. See help(type(self)) for accurate signature.

ax
get the primary object.

class viznet.**EdgeBrush**(*style*, *ax=None*, *lw=1*, *color='k'*, *zorder=0*, *solid_capstyle='butt'*)
Bases: viznet.brush.Brush

a brush for drawing edges.

style
the style of edge, must be a combination of ('>'|'<'|'-'|'.'). * '>', right arrow * '<', left arrow, * '-', line, * '.', dashed line.

Type str

ax
matplotlib Axes instance.

Type Axes

lw
line width.

Type float

color
the color of painted edge by this brush.

Type str

__init__(*style*, *ax=None*, *lw=1*, *color='k'*, *zorder=0*, *solid_capstyle='butt'*)
Initialize self. See help(type(self)) for accurate signature.

class viznet.**Grid**(*dxy=(1, 1)*, *ax=None*, *offset=(0, 0)*)
Bases: object

Grid for affine transformation.

Parameters

- **dxy** (*tuple*) – space in x, y directions.
- **ax** – matplotlib.pyplot.Axes.
- **offset** (*tuple*) – the global offset.

__init__(*dxy=(1, 1)*, *ax=None*, *offset=(0, 0)*)
Initialize self. See help(type(self)) for accurate signature.

class viznet.**Node**(*objs*, *position*, *brush*)
Bases: viznet.edgenode.EdgeNode

A patch with shape and style, defines the allowed connection points, and create pins for connection.

objs
a list matplotlib patch object, with the first the primary object.

Type list

brush
brush.

Type *NodeBrush*

__init__(*objs*, *position*, *brush*)
Initialize self. See help(type(self)) for accurate signature.

ax
get the primary object.

get_connection_point(*direction*)

Parameters **direction** (*1darray*) – unit vector pointing to target direction.

mass_center
mass center of a node

obj
get the primary object.

pin(*direction*, *align=None*)
obtain a pin on specific surface.

Parameters

- **direction** (*'top'|'bottom'|'left'|'right'|float*) – specifies the surface to place a pin, or theta to specift the direction.
- **align** (viznet.EdgeNode|tuple|None, default=None) – align y-axis for 'left' and 'right' pin, x-axis for 'top' and 'bottom' pin.

Returns the pin for wire connection.

Return type *viznet.Pin*

class viznet.**NodeBrush**(*style*, *ax=None*, *color=None*, *size='normal'*, *roundness=0*, *zorder=0*, *rotate=0.0*, *ls='-'*, *lw=None*, *edgecolor=None*, *props=None*)
Bases: viznet.brush.Brush

a brush class used to draw node.

style
refer keys for *viznet.theme.NODE_THEME_DICT*.

Type str

ax
matplotlib Axes instance.

Type Axes

color
the color of painted node by this brush, it will overide theme color if is not *None*.

Type str|None

size
size of node.

Type 'huge'|'large'|'normal'|'small'|'tiny'|'dot'|tuple|float

roundness
the roundness of edges.

Type float

zorder
same to matplotlib zorder.

Type int

rotate
angle for rotation.

Type float

ls
line style.

Type str

props
other arguments passed to handler.

Type dict

__init__(*style*, *ax=None*, *color=None*, *size='normal'*, *roundness=0*, *zorder=0*, *rotate=0.0*, *ls='-'*, *lw=None*, *edgecolor=None*, *props=None*)
Initialize self. See help(type(self)) for accurate signature.

class viznet.**Pin**
Bases: numpy.ndarray, viznet.edgenode.EdgeNode

Simple Dot used for connecting wires.

class viznet.**QuantumCircuit**(*num_bit*, *ax=None*, *x=0*, *y0=0*, *locs=None*, ***kwargs*)
Bases: object

Parameters

- **ax** – matplotlib.pyplot.Axes.
- **num_bit** (*int*) – number of bits.
- **y0** (*float*) – the y offset.

__init__(*num_bit*, *ax=None*, *x=0*, *y0=0*, *locs=None*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

block (*sls*, *pad_x=0.35*, *pad_y=0.35*, *brush=None*)
strike out a block.

Parameters

- **sls** (*int*) – the slice for starting and ending lines.
- **pad_x** (*float*) – x padding between gates and box.
- **pad_y** (*float*) – y padding between gates and box.
- **brush** (*NodeBrush|None*) – the brush used to paint this box.

Returns context that return boxes.

Return type context

focus (*lines*)
focus to target lines

Parameters **lines** (*list*) – the target lines to put up.

gate (*brush*, *position*, *text=''*, *fontsize=18*, *noline=False*)
place a gate at specific position.

get_position (*line*, *x=None*)
get the position of specific line

viznet.**connect121** (*start_nodes*, *end_nodes*, *brush*)

Parameters

- **start_token** (*str*) – the start layer generation token (pointed from).
- **end_token** (*str*) – the end layer generation token (pointed to).
- **brush** (EdgeBrush) – edge brush instance.

viznet.**connecta2a** (*start_nodes*, *end_nodes*, *brush*)

Parameters

- **start_token** (*str*) – the start layer generation token (pointed from).
- **end_token** (*str*) – the end layer generation token (pointed to).
- **brush** (EdgeBrush) – edge brush instance.

viznet.**dict2circuit** (*datamap*, *handler=None*, *blockdict=None*, *putstart=None*)
parse a dict (probably from a yaml file) to a circuit.

Parameters

- **datamap** (*dict*) – the dictionary defining a circuit.
- **handler** (*None|QuantumCircuit*) – the handler.
- **blockdict** (*dict, default=datamap*) – the dictionary for block includes.
- **putstart** (*bool, default=handler==None*) – put a start at the begining if True.

viznet.**node_ring** (*brush*, *num_node*, *center*, *radius*)
add a sequence of nodes placed on a ring.

Parameters

- **brush** (*NodeBrush*) – node brush.
- **num_node** (*int*) – number of node to be added.

- **center** (*tuple*) – center of this ring.
- **radius** (*float*) – the raidus of the ring.

Returns a list of nodes

Return type list

viznet.**node_sequence**(*brush*, *num_node*, *center*, *space=(1, 0)*)
add a sequence of nodes along direction specified by space.

Parameters

- **brush** (NodeBrush) – brush instance.
- **num_node** (*int*) – number of node to be added.
- **center** (*tuple*) – center of this sequence.
- **space** (*tuple | float*) – space between nodes.

Returns a list of node names, you can visit this node by accesing *self.node_dict[node_name]*.

Return type list

viznet.**vizcode**(*handler*, *code*, *blockdict={}*)
visualize a code

Parameters

- **handler** (QuantumCircuit) – circuit handler.
- **code** (*str*) – the string defining a primitive gate.
- **blockdict** (*dict, default={}*) – the refence dict for block includes.

3.2 viznet.theme

3.2.1 Module contents

viznet.theme.**NODE_THEME_DICT**
A table of theme for nodes. values are *COLOR* | *SHAPE* | *INNER_SHAPE*.

```
NODE_THEME_DICT = {
"basic": [
    "none",
    "circle",
    "none"
],
"box": [
    "none",
    "rectangle",
    "none"
],
"invisible": [
    null,
    "circle",
    "none"
],
"nn.backfed": [
    "#FFFF77",
```

(continues on next page)

(continued from previous page)

```
    "circle",
    "circle"
],
"nn.convolution": [
    "#DD99DD",
    "circle",
    "circle"
],
"nn.different_memory": [
    "#3399DD",
    "circle",
    "triangle"
],
"nn.hidden": [
    "#55CC77",
    "circle",
    "none"
],
"nn.input": [
    "#FFFF77",
    "circle",
    "none"
],
"nn.kernel": [
    "#DD99DD",
    "circle",
    "none"
],
"nn.match_input_output": [
    "#FF6644",
    "circle",
    "circle"
],
"nn.memory": [
    "#3399DD",
    "circle",
    "circle"
],
"nn.noisy_input": [
    "#FFFF77",
    "circle",
    "triangle"
],
"nn.output": [
    "#FF6644",
    "circle",
    "none"
],
"nn.pooling": [
    "#DD99DD",
    "circle",
    "circle"
],
"nn.probablistic_hidden": [
    "#55CC77",
    "circle",
    "circle"
```

(continues on next page)

(continued from previous page)

```
],
"nn.recurrent": [
    "#3399DD",
    "circle",
    "none"
],
"nn.spiking_hidden": [
    "#55CC77",
    "circle",
    "triangle"
],
"pin": [
    null,
    "empty",
    "none"
],
"qc.C": [
    "#333333",
    "dot",
    "none"
],
"qc.NC": [
    "none",
    "dot",
    "none"
],
"qc.NOT": [
    "none",
    "circle",
    "plus"
],
"qc.basic": [
    "none",
    "square",
    "none"
],
"qc.box": [
    "none",
    "rectangle",
    "none"
],
"qc.cross": [
    null,
    "empty",
    "cross"
],
"qc.end": [
    null,
    "empty",
    "vbar"
],
"qc.measure": [
    "none",
    "golden",
    "measure"
],
"qc.wide": [
```

(continues on next page)

(continued from previous page)

```
    "none",
    "golden",
    "none"
],
"tn.dia": [
    "#333333",
    "diamond",
    "none"
],
"tn.mpo": [
    "#333333",
    "rectangle",
    "none"
],
"tn.mps": [
    "#333333",
    "circle",
    "none"
],
"tn.tri": [
    "#333333",
    "triangle",
    "none"
]
}
```

3.3 viznet.setting

3.3.1 Module contents

contains default settings for annotate, node, arrow and grid,

- annotate_setting
- node_setting
- edge_setting

Example

disable edge for nodes

```
from viznet.setting import node_setting
node_setting['lw'] = 0
```

viznet.setting.**annotate_setting = {'fontsize': 12, 'text_offset': 0.07}**
global text setting

```
annotate_setting = {
"fontsize": 12,
"text_offset": 0.07
}
```

viznet.setting.**node_setting = {'edgecolor': 'k', 'inner_edgecolor': 'k', 'inner_facecolor**

global node style setting

```
node_setting = {
"edgecolor": "k",
"inner_edgecolor": "k",
"inner_facecolor": "none",
"inner_lw": 0.7,
"lw": 0.7
}
```

viznet.setting.**edge_setting = {'arrow_head_length': 0.06, 'arrow_head_width': 0.04, 'doub**

global edge style setting

```
edge_setting = {
"arrow_head_length": 0.06,
"arrow_head_width": 0.04,
"doubleline_space": 0.016
}
```

Python Module Index

v

Index

P

Q

R

S

V

Z