layers Documentation

Release 0.0.1

Jinguo Liu

Oct 31, 2017

Contents

Layers is intended for scientific programming, with complex number support and periodic boundary condition and most importantly, extensible.

Our project repo is https://159.226.35.226/jgliu/PoorNN.git

Here I list its main features

- low abstraction
- complex number support
- periodic boundary convolution favored
- numpy based, c/fortran boost
- easy for extension, using interfaces to unify layers, which means layers are standalone modules.

But its cuda version is not realized yet.

Contents

CHAPTER 1

Tutorial

Getting started

To start using Layers, simply

clone/download this repo and run

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

Layers is built on a high-performance Fortran 90 code. Please install *lapack/mkl* and *gfortran/ifort* before running above installation.

Construct a first feed forward network

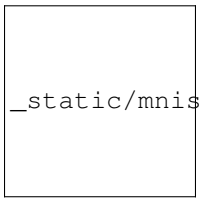
```
'''
Build a simple neural network that can be used to study mnist data set.
'''

import numpy as np
from poornn.nets import ANN
from poornn.checks import check_numdiff
from poornn import functions, Linear
from poornn.utils import typed_randn


def build_ann():
    '''
    builds a single layer network for mnist classification problem.
    '''
    F1 = 10
    I1, I2 = 28, 28
```

```
    eta = 0.1
    dtype = 'float32'

    W_fc1 = typed_randn(dtype, (F1, I1 * I2)) * eta
    b_fc1 = typed_randn(dtype, (F1,)) * eta

    # create an empty vertical network.
    ann = ANN()
    linear1 = Linear((-1, I1 * I2), dtype, W_fc1, b_fc1)
    ann.layers.append(linear1)
    ann.add_layer(functions.SoftMaxCrossEntropy, axis=1)
    ann.add_layer(functions.Mean, axis=0)
    return ann


# build and print it
ann = build_ann()
print(ann)

# random numerical differenciation check.
# prepair a one-hot target.
y_true = np.zeros(10, dtype='float32')
y_true[3] = 1

assert(all(check_numdiff(ann, var_dict={'y_true': y_true})))

# graphviz support
from poornn.visualize import viznn
viznn(ann, filename='./mnist_simple.png')
```

You will get a terminal output

```
<ANN|s>: (-1, 784)|s -> ()|s
    <Linear|s>: (-1, 784)|s -> (-1, 10)|s
      - var_mask = (1, 1)
      - is_unitary = False
    <SoftMaxCrossEntropy|s>: (-1, 10)|s -> (-1,)|s
      - axis = 1
    <Mean|s>: (-1,)|s -> ()|s
      - axis = 0
```

and an illustration of network stored in `/mnist_simple.png`, it looks like



where the shape and type of data flow is marked on lines, and operations are boxes.

Note: However, the above example raises a lib not found error if you don't have *pygraphviz* installed on your host. It is strongly recommended to try out pygraphviz.

Ideology

First, what is a `Layer`?

`Layers` is an abstract class (or interface), which defines a protocal. This protocal specifies

- **Interface information**, namely `input_shape`, `output_shape`, `itype`, `otype` and `dtype`, where `dtype` is the type of variables in this network and `itype`, `otype` are input and output array data type.
- **Data flow manipulation methods**, namely `forward()` and `backward()`. `forward()` perform action $y = f(x)$ and output y, with f defines the functionality of this network. `backward()` perform action $(x, y), \frac{\partial J}{\partial y} \rightarrow \frac{\partial J}{\partial w}, \frac{\partial J}{\partial x}$, where J and w are target cost and layer variables respectively. x and y are always required as a unified interface (benefits network design), usally they are generated during a forward run.
- **Variable getter and setter**, namely `get_variables()`, `set_variables()`, `num_variables` (as property) and `set_runtime_vars()`. `get_variables()` always return a 1D array of length `num_variables` and *set_variables* take such an array as input. Also, a layer can take runtime variables (which should be specidied in *tags*, see bellow), like a `seed` in order to take a control over a `DropOut` layer. These getter and setter are required because we need a unified interface to access variables but not to make variables unreadable in a layer realization. Notice that reshape in numpy does not change array storage, so don't worry about performance.
- **Tags (optional)**, `tags` attribute defines some additional property of a layer, which is an optional dict type variable which belongs to a class. So far, these properties includes 'runtimes' (list), 'is_inplace' (bool) and 'analytical' (int), see *poornn.core.TAG_LIST* for details. If `tags` is not defined, layer use *poornn.core.DEFAULT_TAGS* as a default tags.

Any object satisfing the above protocal can be used as a `Layer`. An immediate benefit is that it can be tested. e.g. numerical differenciation test using *poornn.checks.check_numdiff()*.

Through running the above example, we notice the following facts:

1. Layers take numpy array as inputs and generates array outputs (notice what `typed_randn()` also generates numpy array).
2. Network `ANN` is a subclass of `Layer`, it realizes all the interfaces claimed in `Layer`, it is a kind of simplest vertical `Container`. Here, `Container` is a special kind of `Layer`, it take other layers as its entity and has no independant functionality. Containers can be nested, chained, ... to realize complex networks.
3. `-1` is used as a placeholder in a shape, however, using more than 1 place holder in one shape tuple is not recommended, it raises error during reshaping.
4. Anytime, a `Layer` take `input_shape` and `itype` as first 2 parameters to initialize, even it is not needed! However, you can ommit it by using `add_layer()` method of `ANN` or `PrallelNN` network when you are trying to add a layer to existing network. `add_layer()` can infer input shape and type from previous layers. Appearantly, it fails when there is no layers in a container. Then you should use `net.layers.append()` to add a first layer, or give at least one layer when initialing a container.

Note: In linear layers, fortran ('F') ordered weights and inputs are used by default.

CHAPTER 2

Examples

The most basic example is mnist classification. It can be found in the examples-folder of Layers. The code looks as follows

```
'''
Build a simple neural network that can be used to study mnist data set.
'''

import numpy as np
from poornn.nets import ANN
from poornn.checks import check_numdiff
from poornn import functions, Linear
from poornn.utils import typed_randn


def build_ann():
    '''
    builds a single layer network for mnist classification problem.
    '''
    F1 = 10
    I1, I2 = 28, 28
    eta = 0.1
    dtype = 'float32'

    W_fc1 = typed_randn(dtype, (F1, I1 * I2)) * eta
    b_fc1 = typed_randn(dtype, (F1,)) * eta

    # create an empty vertical network.
    ann = ANN()
    linear1 = Linear((-1, I1 * I2), dtype, W_fc1, b_fc1)
    ann.layers.append(linear1)
    ann.add_layer(functions.SoftMaxCrossEntropy, axis=1)
    ann.add_layer(functions.Mean, axis=0)
    return ann
```

```
# build and print it
ann = build_ann()
print(ann)

# random numerical differenciation check.
# prepair a one-hot target.
y_true = np.zeros(10, dtype='float32')
y_true[3] = 1

assert(all(check_numdiff(ann, var_dict={'y_true': y_true})))

# graphviz support
from poornn.visualize import viznn
viznn(ann, filename='./mnist_simple.png')
```

```
$ python examples/mnist_simple.py
```

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:

core

Module contents

ABC of neural network.

class poornn.core.**Layer**(*input_shape*, *output_shape*, *itype*, *dtype=None*, *otype=None*, *tags=None*)
Bases: object

A single layer in Neural Network.

Parameters

- **input_shape** (*tuple*) – input shape of this layer.
- **output_shape** (*tuple*) – output_shape of this layer.
- **itype** (*str*) – input data type.
- **dtype** (str, default=:data:*itype*) – variable data type.
- **otype** (*str, default=?*) – output data type, if not provided, it will be set to itype, unless its 'analytical' tags is 2.
- **tags** (dict, default=:data:*poornn.core.DEFAULT_TAGS*) – tags used to describe this layer, refer *poornn.core.TAG_LIST* for detail. It change tags based on template *poornn.core.DEFAULT_TAGS*.

input_shape
tuple – input shape of this layer.

output_shape
tuple – output_shape of this layer.

itype
str – input data type.

dtype
str – variable data type.

otype
str – output data type.

tags
dict – tags used to describe this layer, refer *poornn.core.TAG_LIST* for detail.

backward(*xy*, *dy*, *mask=(1, 1)*)
back propagation to get $\frac{\partial J(w,x)}{\partial w}$ and $\frac{\partial J(w,x)}{\partial x}$, where J and w are cost function and variables respectively.

Parameters

- **xy** (*tuple<ndarray>, len=2*) – input and output array.
- **dy** (*ndarray*) – gradient of output defined as $\partial J/\partial y$.
- **mask** (*tuple*) – (do_wgrad, do_xgrad)

Returns (ndarray, ndarray), $\partial J/\partial w$ and $\partial J/\partial x$.

forward(*x*, ***kwargs*)
forward propagration to evaluate $y = f(x)$.

Parameters

- **x** (*ndarray*) – input array.
- **runtime_vars** (*dict*) – runtime variables.

Returns ndarray, output array y.

get_variables()
Get current variables.

Returns 1darray,

num_variables
number of variables.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(*variables*)
Change current variables.

Parameters **variables** (*1darray*) –

class poornn.core.**Function**(*input_shape*, *output_shape*, *itype*, *dtype=None*, *otype=None*, *tags=None*)
Bases: *poornn.core.Layer*

Function layer with no variables.

backward(*xy*, *dy*, *mask=(1, 1)*)
back propagation to get $\frac{\partial J(w,x)}{\partial w}$ and $\frac{\partial J(w,x)}{\partial x}$, where J and w are cost function and variables respectively.

Parameters

- **xy** (*tuple<ndarray>, len=2*) – input and output array.
- **dy** (*ndarray*) – gradient of output defined as $\partial J/\partial y$.
- **mask** (*tuple*) – (do_wgrad, do_xgrad)

Returns (ndarray, ndarray), $\partial J/\partial w$ and $\partial J/\partial x$.

forward(*x*, ***kwargs*)
forward propagration to evaluate $y = f(x)$.

Parameters

- **x** (*ndarray*) – input array.
- **runtime_vars** (*dict*) – runtime variables.

Returns ndarray, output array y.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.core.**ParamFunction**(*input_shape*, *output_shape*, *itype*, *params*, *var_mask*, ***kwargs*)
Bases: *poornn.core.Layer*

Function layer with params as variables and var_mask as variable mask.

Parameters

- **params** (*1darray*) – variables used in this functions.
- **var_mask** (*1darray<bool>, default=(True,True,..)*) – mask for params, a param is regarded as a constant if its mask is False.

params
1darray – variables used in this functions.

var_mask
1darray<bool> – mask for params, a param is regarded as a constant if its mask is False.

backward(*xy*, *dy*, *mask=(1, 1)*)
back propagation to get $\frac{\partial J(w,x)}{\partial w}$ and $\frac{\partial J(w,x)}{\partial x}$, where J and w are cost function and variables respectively.

Parameters

- **xy** (*tuple<ndarray>, len=2*) – input and output array.
- **dy** (*ndarray*) – gradient of output defined as $\partial J/\partial y$.
- **mask** (*tuple*) – (do_wgrad, do_xgrad)

Returns (ndarray, ndarray), $\partial J/\partial w$ and $\partial J/\partial x$.

forward(*x*, ***kwargs*)
forward propagration to evaluate $y = f(x)$.

Parameters

- **x** (*ndarray*) – input array.
- **runtime_vars** (*dict*) – runtime variables.

Returns ndarray, output array y.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

class poornn.core.**Monitor**(*input_shape*, *output_shape*, *itype*, *dtype=None*, *otype=None*, *tags=None*)
Bases: *poornn.core.Function*

A special layer used to monitor a flow, it operate on but do not change the flow.

get_variables()
Get variables, return empty (1d but with length - 0) array.

monitor_backward(*xy*, *dy*, ***kwargs*)
Monitor function used in backward,

Parameters

- **xy** (*ndarray*) – (input, output(same as input)) data.
- **dy** (*ndarray*) – gradient.

monitor_forward(*x*)
Monitor function used in forward,

Parameters **x** (*ndarray*) – forward data.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

poornn.core.**EXP_OVERFLOW = 12**
exp(x>EXP_OVERFLOW) should be taken special care of in order avoid overflow.

poornn.core.**EMPTY_VAR = array([], dtype=float32)**
Empty variable, 1d array of dtype 'float32' and length 0.

exception poornn.core.**AnalyticityError**
Bases: exceptions.Exception

Behavior conflict with the analytical type of a layer.

__init__
x.__init__(...) initializes x; see help(type(x)) for signature

poornn.core.**DEFAULT_TAGS = {'analytical': 1, 'is_inplace': False, 'runtimes': []}**
A layer without tags attributes will take this set of tags.

•no runtime variables,

•changes for flow are not inplace (otherwise it will destroy integrity of flow history).

•analytical (for complex numbers, holomophic).

poornn.core.**TAG_LIST = ['runtimes', 'is_inplace', 'analytical']**
List of tags –

- **'runtimes' (list<str>, default=[]):** runtime variables that should be supplied during each run.
- **'is_inplace' (bool, default=False):** True if the output is made by changing input inplace.
- **'analytical' (int):** the analyticaity of a layer. A table of legal values,
  - 1, yes (default)
  - 2, yes for float, no for complex, complex output for real output.
  - 3, yes for float, no for complex, complex output for complex input.
  - 4, no

checks

Module contents

poornn.checks.**dec_check_shape**(*pos*)
Check the shape of layer's method.

Parameters **pos** (*tuple*) – the positions of arguments to check shape.

Note: BUGGY.

Returns a decorator.

Return type func

poornn.checks.**check_numdiff**(*layer*, *x=None*, *num_check=10*, *eta_x=None*, *eta_w=None*, *tol=0.001*, *var_dict={}*)
Random Numerical Differentiation check.

Parameters

- **layer** (Layer) – the layer under check.
- **x** (*ndarray|None, default=None*) – input data, randomly generated if is None.
- **num_check** (*int, default=10*) – number of random derivative checks for both inputs and weights.
- **eta_x** (*number, default=0.005 if float else 0.003+0.004j*) – small change on input, in order to obtain numerical difference.
- **eta_w** (*number, default=0.005 if float else 0.003+0.004j*) – small change on weight, in order to obtain numerical difference.
- **tol** (*float, default=1e-3*) – tolerence, relative difference allowed with respect to max(letal, lgradientl).
- **var_dict** (*dict, default={}*) – feed runtime variables if needed.

Returns test results, True for passed else False.

Return type list<bool>

poornn.checks.**generate_randx**(*layer*)
Generate random input tensor.

poornn.checks.**check_shape_backward**(*f*)
Check the shape of layer's backward method.

Parameters **f** (*func*) – backward method.

Note: BUGGY.

Returns function decorator.

Return type func

poornn.checks.**check_shape_forward**(*f*)
Check the shape of layer's forward method.

Parameters **f** (*func*) – forward method.

Note: BUGGY.

Returns function decorator.

Return type func

poornn.checks.**check_shape_match**(*shape_get*, *shape_desire*)
check whether shape_get matches shape_desire.

Parameters

- **shape_get** (*tuple*) – obtained shape.
- **shape_desire** (*tuple*) – desired shape.

Returns tuple, the shape with more details.

nets

Module contents

ABC of neural network.

class poornn.nets.**ANN**(*layers=None*, *labels=None*)
Bases: poornn.core.Container

Sequential Artificial Neural network.

add_layer(*cls*, *label=None*, ***kwargs*)
Add a new layer, comparing with self.layers.append()

- input_shape of new layer is infered from output_shape of last layer.
- itype of new layer is infered from otype of last layer.

Parameters

- **cls** (*class*) – create a layer instance, take input_shape and itype as first and second parameters.
- **label** (*str|None, default=None*) – label to index this layer, leave *None* if indexing is not needed.
- ****kwargs** – keyword arguments used by `cls.__init__()`, excluding `input_shape` and `itype`.

Note: if *num_layers* is 0, this function will raise an error, because it fails to infer `input_shape` and `itype`.

Returns newly generated object.

Return type *Layer*

backward(*xy*, *dy=array(1)*, *data_cache=None*, *do_shape_check=False*)
Compute gradients.

Parameters

- **xy** (*tuple*) – input and output
- **dy** (*ndarray*) – gradient of output defined as $\partial J/\partial y$.
- **data_cache** (*dict*) – a dict with collected datas.
- **do_shape_check** (*bool*) – check shape of data flow if True.

Returns gradients for vairables in layers.

Return type list

dtype
variable data shape, which is infered from layers.

forward(*x*, *data_cache=None*, *do_shape_check=False*, ***kwargs*)
Feed input to this feed forward network.

Parameters

- **x** (*ndarray*) – input in 'F' order.
- **data_cache** (*dict|None, default=None*) – a dict used to collect datas.
- **do_shape_check** (*bool*) – check shape of data flow if True.

Note: `data_cache` should be pass to this method if you are about to call a subsequent *backward()* method, because backward need `data_cache`.

`'%d-ys'%id(self)` is used as the key to store run-time output of layers in this network. `data_cache['%d-ys'%id(self)]` is a list with contents outputs in each layers generate in this forward run.

Returns output in each layer.

Return type list

get_runtimes()
show runtime variables used in this `Container`.

get_variables()
Dump values to an array.

num_layers
number of layers.

num_variables
int – number of variables.

set_runtime_vars(*var_dict*)
Set runtime variables.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(*v*)
Load data from an array.

Parameters **v** (*1darray*) – variables.

tags
tags for this `Container`, which is infered from `self.layers`.

class poornn.nets.**ParallelNN**(*axis=0*, *layers=None*, *labels=None*)
Bases: poornn.core.Container

Parallel Artificial Neural network.

Parameters **axis** (*int, default=0*) – specify the additional axis on which outputs are packed.

axis
int – specify the additional axis on which outputs are packed.

add_layer(*cls*, ***kwargs*)
add a new layer, comparing with `self.layers.append()`

- *input_shape* of new layer is infered from *input_shape* of first layer.
- *itype* of new layer is infered from *itype* of first layer.
- *otype* of new layer is infered from *otype* of first layer.

Parameters

- **cls** (*class*) – create a layer instance, take input_shape and itype as first and second parameters.
- ****kwargs** – keyword arguments used by *cls.__init__*, excluding *input_shape* and *itype*.

Note: if *self.num_layers* is 0, this function will raise an error, because it fails to infer *input_shape*, *itype* and *otype*.

Returns newly generated object.

Return type *Layer*

backward(*xy*, *dy=array(1)*, *do_shape_check=False*, ***kwargs*)
Compute gradients.

Parameters

- **xy** (*tuple*) – input and output
- **dy** (*ndarray*) – gradient of output defined as $\partial J/\partial y$.
- **do_shape_check** (*bool*) – check shape of data flow if True.

Returns gradients for vairables in layers.

Return type list

dtype
variable data shape, which is infered from layers.

forward(*x*, *do_shape_check=False*, ***kwargs*)
Feed input, it will generate a new axis,and storge the outputs of layers parallel along this axis.

Parameters

- **x** (*ndarray*) – input in 'F' order.
- **do_shape_check** (*bool*) – check shape of data flow if True.

Returns output,

Return type ndarray

get_runtimes()
show runtime variables used in this Container.

get_variables()
Dump values to an array.

num_layers
number of layers.

num_variables
int – number of variables.

set_runtime_vars(*var_dict*)
Set runtime variables.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(*v*)
Load data from an array.

Parameters **v** (*1darray*) – variables.

tags
tags for this Container, which is infered from self.layers.

class poornn.nets.**JointComplex**(*real*, *imag*)
Bases: poornn.core.Container

Function $f(z) = h(x) + ig(y)$, where h and g are real functions. This Container can be used to generate complex layers, but its non-holomophic (*analytical* type 3).

Parameters

- **real** (Layer) – layer for real part.
- **imag** (Layer) – layer for imaginary part.

dtype
variable data shape, which is infered from layers.

get_runtimes()
show runtime variables used in this Container.

get_variables()
Dump values to an array.

imag
the imaginary part layer.

num_layers
number of layers.

num_variables
int – number of variables.

real
the real part layer.

set_runtime_vars(*var_dict*)
Set runtime variables.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(*v*)
Load data from an array.

Parameters **v** (*1darray*) – variables.

class poornn.nets.**KeepSignFunc**(*h*, *is_real=False*)
Bases: poornn.core.Container

Function $f(z) = h(|z|)(z)$, where h is a real function. This Container inherit sign from input, so it must have same input and ouput dimension. It can also be used to generate complex layers, but its non-holomophic (*analytical* type 3).

Parameters **is_real** (*bool, default=False*) – input is real if True.

is_real
bool – input is real if True.

dtype
variable data shape, which is infered from layers.

get_runtimes()
show runtime variables used in this Container.

get_variables()
Dump values to an array.

h
layer applied on amplitude.

num_layers
number of layers.

num_variables
int – number of variables.

set_runtime_vars(*var_dict*)
Set runtime variables.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(*v*)
Load data from an array.

Parameters **v** (*1darray*) – variables.

linears

Module contents

Linear Layer.

class poornn.linears.**LinearBase**(*input_shape*, *itype*, *weight*, *bias*, *var_mask=(1, 1)*)

Bases: *poornn.core.Layer*

Base of Linear Layer.

Parameters

- **weight** (*ndarray/matrix*) – weights in a matrix.
- **bias** (*1darray|None*) – bias of shape (fout,), zeros if None.
- **var_mask** (*tuple<bool>, len=2, default=(True,True)*) – variable mask for weight and bias.

weight
ndarray/matrix – weights in a matrix.

bias
1darray|None – bias of shape (fout,), zeros if None.

var_mask
tuple<bool>, len=2 – variable mask for weight and bias.

backward(*xy*, *dy*, *mask=(1, 1)*)
back propagation to get $\frac{\partial J(w,x)}{\partial w}$ and $\frac{\partial J(w,x)}{\partial x}$, where J and w are cost function and variables respectively.

Parameters

- **xy** (*tuple<ndarray>, len=2*) – input and output array.
- **dy** (*ndarray*) – gradient of output defined as $\partial J/\partial y$.
- **mask** (*tuple*) – (do_wgrad, do_xgrad)

Returns (ndarray, ndarray), $\partial J/\partial w$ and $\partial J/\partial x$.

forward(*x*, ***kwargs*)
forward propagration to evaluate $y = f(x)$.

Parameters

- **x** (*ndarray*) – input array.
- **runtime_vars** (*dict*) – runtime variables.

Returns ndarray, output array y.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

class poornn.linears.**Linear**(*input_shape*, *itype*, *weight*, *bias*, *var_mask=(1, 1)*, *is_unitary=False*, ***kwargs*)

Bases: *poornn.linears.LinearBase*

Dense Linear Layer, $f = x \cdot W^{\dagger} + b$

Parameters

- **is_unitary** (*bool, default=False*) – keep unitary if True,
- **way to keep unitary during evolution will overload set_variables method.** (*the*) –

is_unitary
bool – keep unitary if True, unitary will overload *set_variables* method.

be_unitary()
make weight unitary through qr decomposition.

check_unitary(*tol=1e-10*)
check weight is unitary or not, if not, raise an exception.

Parameters **tol** (*float, default=1e-10*) – the tolerence.

Returns error rate.

Return type float

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

class poornn.linears.**SPLinear**(*input_shape*, *itype*, *weight*, *bias*, *var_mask=(1, 1)*, ***kwargs*)
Bases: *poornn.linears.LinearBase*

Sparse Linear Layer, weight now is a sparse matrix..

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

class poornn.linears.**Apdot**(*input_shape*, *itype*, *weight*, *bias*, *var_mask=(1, 1)*)
Bases: *poornn.linears.LinearBase*

Apdot swiches roles between multiply and add in linear layer.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

spconv

Module contents

Convolution using sparse matrix.

class poornn.spconv.**SPConv**(*input_shape*, *itype*, *weight*, *bias*, *strides=None*, *boundary='P'*, *w_contiguous=True*, *var_mask=(1, 1)*, *is_unitary=False*, ***kwargs*)
Bases: *poornn.linears.LinearBase*

Convolution layer.

Parameters

- **weight** (*ndarray*) – dimensions are aranged as (feature_out, feature_in, kernel_x, ...), in 'F' order.
- **bias** (*1darray*) – length of num_feature_out.
- **strides** (*tuple, default=(1,1,..)*) – displace for convolutions.
- **boudnary** (*'P'|'O', default='P'*) – boundary type, * 'P', periodic boundary condiction. * 'O', open boundary condition.
- **is_unitary** (*bool, default=False*) – keep unitary if True, here, unitary is defined in the map *U: img_in -> feature_out*.
- **var_mask** (*tuple<bool>, len=2, default=(True,True)*) – variable mask for weight and bias.

weight
ndarray – dimensions are aranged as (feature_out, feature_in, kernel_x, ...), in 'F' order.

bias
1darray – length of num_feature_out.

strides
tuple – displace for convolutions.

boudnary
'P'|'O' – boundary type, * 'P', periodic boundary condiction. * 'O', open boundary condition.

is_unitary
bool – keep unitary if True, here, unitary is defined in the map *U: img_in -> feature_out*.

var_mask
tuple<bool>, len=2 – variable mask for weight and bias.

(Derived)

csc_indptr
1darray – column pointers for convolution matrix.

csc_indices
1darray – row indicator for input array.

weight_indices
1darray – row indicator for filter array (if not contiguous).

backward(*xy*, *dy*, ***kwargs*)

Parameters

- **xy** (*(ndarray, ndarray)*) –
  - x -> (num_batch, nfi, img_in_dims), input in 'F' order.
  - y -> (num_batch, nfo, img_out_dims), output in 'F' order.
- **dy** (*ndarray*) – (num_batch, nfo, img_out_dims), gradient of output in 'F' order.
- **mask** (*booleans*) – (do_xgrad, do_wgrad, do_bgrad).

Returns dw, dx

Return type tuple(1darray, ndarray)

forward(*x*, ***kwargs*)

Parameters **x** (*ndarray*) – (num_batch, nfi, img_in_dims), input in 'F' order.

Returns ndarray, (num_batch, nfo, img_out_dims), output in 'F' order.

img_nd
Dimension of input image.

num_feature_in
Dimension of input feature.

num_feature_out
Dimension of input feature.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

functions

Module contents

poornn.functions.**wrapfunc**(*func*, *dfunc*, *classname='GeneralFunc'*, *attrs={}*, *docstring=''*, *tags={}*, *real_out=False*)
wrap a function and its backward counterpart into a *poornn.core.Function* layer.

Parameters

- **func** (*func*) – forward function, take input (x, attrs) as parameters.
- **dfunc** (*func*) – derivative function, take input/output (x, y, **attrs) as parameters.
- **classname** (*str*) – function classname,
- **attrs** (*dict*) – attributes, and default input parameters.
- **docstring** (*str*) – the docstring of new class.
- **tags** (*dict*) – tags for this function, see *poornn.core.TAG_LIST* for detail.
- **real_out** (*bool*) – output data type is real for any input data type if True.

Returns a dynamically generated layer type.

Return type class

class poornn.functions.**Log2cosh**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = \log(2\cosh(x))$.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Logcosh**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = \log(\cosh(x))$.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Sigmoid**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = \frac{1}{1+\exp(-x)}$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Cosh**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = \cosh(x)$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Sinh**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = \sinh(x)$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Tan**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = \tan(x)$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Tanh**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = \tanh(x)$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Sum**(*input_shape*, *itype*, *axis*, ***kwargs*)
Bases: *poornn.core.Function*

np.sum along *axis*.

Parameters **axis** (*int*) – the axis along which to sum over.

axis
int – the axis along which to sum over.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Mul**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = \text{alpha} \cdot x$

Parameters **alpha** (*int*) – the multiplier.

alpha
int – the multiplier.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Mod**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = x\%n$

Parameters **n** (*number*) – the base.

n
number – the base.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Mean**(*input_shape*, *itype*, *axis*, ***kwargs*)
Bases: *poornn.core.Function*

np.mean along *axis*.

Parameters **axis** (*int*) – the axis along which to operate.

axis
int – the axis along which to operate.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**FFT**(*input_shape*, *itype*, *axis*, *kernel='fft'*, ***kwargs*)
Bases: *poornn.core.Function*

scipy.fftpack.[fft|ifft|dct|idct|...] along *axis*.

Parameters

- **axis** (*int*) – the axis along which to operate.
- **kernel** (*str, default='fft'*) – the kernel used.

axis
int – the axis along which to operate.

kernel
str, default='fft' – the kernel used. refer *scipy.fftpack* for available kernels.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**ReLU**(*input_shape*, *itype*, *leak=0.0*, *is_inplace=False*, *mode=None*, ***kwargs*)
Bases: *poornn.core.Function*

ReLU, for mode='ri',

$$f(x) = \text{relu}(x) = \begin{cases} x, & \Re[x] > 0 \wedge \Im[x] > 0 \\ \Re[x] + \text{leak} \cdot \Im[x], & \Re[x] > 0 \wedge \Im[x] < 0 \\ \Im[x] + \text{leak} \cdot \Re[x], & \Re[x] < 0 \wedge \Im[x] > 0 \\ \text{leak} \cdot x, & \Re[x] < 0 \wedge \Im[x] < 0 \end{cases} \tag{3.1}$$

for mode='r',

$$f(x) = \text{relu}(x) = \begin{cases} x, & \Re[x] > 0 \\ \text{leak} \cdot x, & \Re[x] < 0 \end{cases} \tag{3.2}$$

Parameters

- **leak** (*float, default = 0.0*) – leakage,

- **mode** (*'ri'|'r', default='r' if itype is float else 'ri'*) – non-holomophic real-imaginary (ri) relu or holomophic real (r) relu

leak
float – leakage,

mode
'ri'|'r' – non-holomophic real-imaginary (ri) relu or holomophic real (r) relu.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**ConvProd**(*input_shape*, *itype*, *powers*, *strides=None*, *boundary='O'*, ***kwargs*)
Bases: *poornn.core.Function*

Convolutional product layer, apply a kernel as powers to a subregion and make product over these elements.

Parameters

- **powers** (*ndarray*) – powers as a kernel.
- **strides** (*tuple, default=(1, 1,..)*) – stride for each dimension.
- **boundary** (*'P'|'O', default='O'*) – Periodic/Open boundary condition.

powers
ndarray – powers as a kernel.

strides
tuple – stride for each dimension.

boundary
'P'|'O' – Periodic/Open boundary condition.

Note: For input array x, axes are aranged as (num_batch, nfi, img_in_dims), stored in 'F' order. For output array y, axes are aranged as (num_batch, nfo, img_out_dims), stored in 'F' order.

---

get_variables()
Get variables, return empty (1d but with length - 0) array.

img_nd
int – dimension of image.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Pooling**(*input_shape*, *itype*, *kernel_shape*, *mode*, ***kwargs*)
Bases: *poornn.core.Function*

Pooling, see *Pooling.mode_list* for different types of kernels.

Parameters

- **kernel_shape** (*tuple*) – the shape of kernel.
- **mode** (*str*) – the strategy used for pooling,
- **Pooling.mode_list for available modes.** (*refer*) –

kernel_shape
tuple – the shape of kernel.

mode
str – the strategy used for pooling.

Note: For input array x, axes are aranged as (num_batch, nfi, img_in_dims), stored in 'F' order. For output array y, axes are aranged as (num_batch, nfo, img_out_dims), stored in 'F' order.

For complex numbers, what does max pooling looks like?

---

get_variables()
Get variables, return empty (1d but with length - 0) array.

img_nd
int – dimension of image.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**DropOut**(*input_shape*, *itype*, *keep_rate*, *axis*, *is_inplace=False*, ***kwargs*)
Bases: *poornn.core.Function*

DropOut, take runtime variable seed.

Parameters

- **axis** (*int*) – the axis along which to operate.
- **keep_rate** (*float*) – the ratio of kept data.

axis
int – the axis along which to operate.

keep_rate
float – the ratio of kept data.

Example

```
>>> layer = DropOut((3, 3), 'complex128', keep_rate = 0.5, axis=-1)
>>> layer.set_runtime_vars({'seed': 2})
>>> x = np.arange(9, dtype='complex128').reshape([3, 3], order='F')
>>> print(layer.forward(x))
[[  0.+0.j   6.+0.j   0.+0.j]
 [  2.+0.j   8.+0.j   0.+0.j]
 [  4.+0.j  10.+0.j   0.+0.j]]
```

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict*)
Set the runtime variable seed, used to generate a random mask.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Sin**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = \sin(x)$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Cos**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = \cos(x)$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**ArcTan**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = \arctan(x)$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Exp**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = \exp(x)$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Log**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = \log(x)$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**SoftPlus**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $log(1 + exp(x))$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Power**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = x^{\text{order}}$

Parameters **order** (*number*) – the order of power.

order
number – the order of power.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**SoftMax**(*input_shape*, *itype*, *axis*, *scale=1.0*, ***kwargs*)
Bases: *poornn.core.Function*

Soft max function $f(x) = \text{scale} \cdot \frac{\exp(x)}{\sum \exp(x)}$, with the sum performed over *axis*.

Parameters

- **axis** (*int*) – the axis along which to operate.
- **scale** (*number*) – the factor to rescale output.

axis
int – the axis along which to operate.

scale
number, default = 1.0 – the factor to rescale output.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**CrossEntropy**(*input_shape*, *itype*, *axis*, ***kwargs*)
Bases: *poornn.core.Function*

Cross Entropy $f(x) = \sum$ with y_true the true labels, and the sum is performed over *axis*.

Parameters **axis** (*int*) – the axis along which to operate.

axis
int – the axis along which to operate.

forward(*x*, ***kwargs*)

Parameters

- **x** (*ndarray*) – satisfying $0 < x \leq 1$.
- **y_true** (*ndarray*) – correct one-hot y.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**SoftMaxCrossEntropy**(*input_shape*, *itype*, *axis*, ***kwargs*)
Bases: *poornn.core.Function*

Soft Max & Cross Entropy $f(x) = \sum$

Parameters
axis (*int*) – the axis along which to operate.

axis
int – the axis along which to operate.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**SquareLoss**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Square Loss $f(x) = (x -$

y_true get_variables()

var_dict (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Reshape**(*input_shape*, *output_shape*, *itype*, *dtype=None*, *otype=None*, *tags=None*)
Bases: *poornn.core.Function*

Change shape of data, reshape is performed in 'F' order.

Note: output_shape is a mandatory parameter now.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Transpose**(*input_shape*, *itype*, *axes*, ***kwargs*)
Bases: *poornn.core.Function*

Transpose data flow.

Parameters **axes** (*tuple*) – the target axes order.

axes
tuple – the target axes order.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**TypeCast**(*input_shape*, *itype*, *otype*, ***kwargs*)
Bases: *poornn.core.Function*

Data type switcher.

Note: otype is a mandatory parameter now.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Filter**(*input_shape*, *itype*, *momentum*, *axes*, ***kwargs*)
Bases: *poornn.core.Function*

Momentum Filter, single component fourier transformation. $f(x) = \sum_{n=0}^{N-1} \exp(-i\pi kn/N) \cdot x[n]$, with index n iterate over *axes*.

Parameters

- **momentum** (*1darray*) – the desired momentum.
- **axes** (*tuple*) – lattice axes over which to filter out component with the desired momentum.

momentum
1darray – the desired momentum.

axes
tuple – lattice axes over which to filter out component with the desired momentum.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**BatchNorm**(*input_shape*, *itype*, *eps=1e-08*, *axis=None*, ***kwargs*)
Bases: *poornn.core.Function*

Batch normalization layer.

Parameters

- **axis** (*int|None, default = None*) – batch axis over which we take norm.
- **eps** (*float, default = 1e-8*) – small number to avoid division to 0.

axis
int|None – batch axis over which to calculate norm, if it is None, we don't use any axis as batch, instead, we need to set mean and variance manually.

eps
float – small number to avoid division to 0.

Note: shall we take mean and variance as run time variable?

---

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Normalize**(*input_shape*, *itype*, *axis*, *scale=1.0*, ***kwargs*)
Bases: *poornn.core.Function*

Normalize data, $f(x) = \text{scale} \cdot x/\|x\|$, where the norm is performed over *axis*.

Parameters

- **axis** (*int*) – axis over which to calculate norm.
- **scale** (*number, default = 1.0*) – the scaling factor.

axis
int – axis over which to calculate norm.

scale
number – the scaling factor.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Real**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = \Re[x]$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Imag**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = \Im[x]$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Conj**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = x^*$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Abs**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = |x|$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Abs2**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = |x|^2$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.functions.**Angle**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Function*

Function $f(x) = \text{Arg}(x)$

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

pfunctions

Module contents

class poornn.pfunctions.**PReLU**(*input_shape, itype, leak=0.1, var_mask=[True]*)
Bases: *poornn.core.ParamFunction*

Parametric ReLU,

$$f(x) = \text{relu}(x) = \begin{cases} x, & \Re[x] > 0 \\ \text{leak} \cdot x, & \Re[x] < 0 \end{cases} \tag{3.3}$$

where leak is a trainable parameter if var_mask[0] is True.

Parameters

- **leak** (*float, default=0.1*) – leakage,
- **var_mask** (*1darray<bool>, default=[True]*) – variable mask

leak
float – leakage,

var_mask
1darray<bool> – variable mask

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

class poornn.pfunctions.**Poly**(*input_shape*, *itype*, *params*, *kernel='polynomial'*, *var_mask=None*, *factorial_rescale=False*)

Bases: *poornn.core.ParamFunction*

Ploynomial function layer.

e.g. for polynomial kernel, we have

- $f(x) = \sum_i \text{params}[i] x^i / i!$ (factorial_rescale is True)
- $f(x) = \sum_i \text{params}[i] x^i$ (factorial_rescale is False)

Parameters

- **kernel** (*str, default='polynomial'*) – the kind of polynomial serie expansion, see *Poly.kernel_dict for detail.*
- **factorial_rescale** (*bool, default=False*) – rescale high order factors to avoid overflow.
- **var_mask** (*1darray<bool>, default=(True,True,..)*) – variable mask

kernel
str – the kind of polynomial serie expansion, see *Poly.kernel_dict for detail.*

factorial_rescale
bool – rescale high order factors to avoid overflow.

var_mask
1darray<bool> – variable mask

kernel_dict = {'chebyshev': <class 'numpy.polynomial.chebyshev.Chebyshev'>, 'legendre': <class 'numpy.polynomial
dict of available kernels, with values target functions.

max_order
int – maximum order appeared.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

class poornn.pfunctions.**Mobius**(*input_shape*, *itype*, *params*, *var_mask=None*)

Bases: *poornn.core.ParamFunction*

Mobius transformation, $f(x) = \frac{(z-a)(b-c)}{(z-c)(b-a)}$

a, b, c map to $0, 1, \infty$ respectively.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

class poornn.pfunctions.**Georgiou1992**(*input_shape*, *itype*, *params*, *var_mask=None*)

Bases: *poornn.core.ParamFunction*

Function $f(x) = \frac{x}{c+|x|/r}$

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

class poornn.pfunctions.**Gaussian**(*input_shape*, *itype*, *params*, *var_mask=None*)
Bases: *poornn.core.ParamFunction*

Function $f(x) = \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{\|x-\mu\|^2}{2\sigma^2})$, where μ, σ are mean and variance respectively.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

class poornn.pfunctions.**PMul**(*input_shape*, *itype*, *c=1.0*, *var_mask=None*)
Bases: *poornn.core.ParamFunction*

Function $f(x) = cx$, where c is trainable if var_mask[0] is True.

Parameters **c** (*number, default=1.0*) – multiplier.

c
number – multiplier.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

derivatives

Module contents

Derived functions, name prefix specifies its container, like

- KS_: nets.KeepSignFunc
- JC_: nets.JointComplex

poornn.derivatives.**KS_Tanh**(*input_shape*, *itype*, ***kwargs*)
Function $f(x) = \tanh(|x|)\exp(i\theta_x)$.

References

Hirose 1994

Returns keep sign tanh layer.

Return type *KeepSignFunc*

poornn.derivatives.**KS_Georgiou1992**(*input_shape, itype, cr, var_mask=[False, False], **kwargs*)
Function $f(x) = \frac{x}{c+|x|/r}$

Parameters

- **cr** (*tuplei, len=2*) – c and r.
- **var_mask** (*1darray, len=2, default=[False,False]*) – mask for variables (v, w), with v = -c*r and w = -cr/(1-r).

Returns keep sign Georgiou's layer.

Return type *KeepSignFunc*

poornn.derivatives.**JC_Tanh**(*input_shape*, *itype*, ***kwargs*)
Function $f(x) = \tanh(\Re[x]) + i\tanh(\Im[x])$.

References

Kechriotis 1994

Returns joint complex tanh layer.

Return type *JointComplex*

poornn.derivatives.**JC_Sigmoid**(*input_shape*, *itype*, ***kwargs*)
Function $f(x) = \sigma(\Re[x]) + i\sigma(\Im[x])$.

References

Birx 1992

Returns joint complex sigmoid layer.

Return type *JointComplex*

poornn.derivatives.**JC_Georgiou1992**(*input_shape*, *itype*, *params*, ***kwargs*)
Function $f(x) = \text{Georgiou1992}(\Re[x]) + i\text{Georgiou1992}(\Im[x])$.

Parameters **params** – params for Georgiou1992.

References

Kuroe 2005

Returns joint complex Geogiou's layer.

Return type *JointComplex*

monitors

Module contents

class poornn.monitors.**Print**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Monitor*

Print data without changing anything.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.monitors.**PlotStat**(*input_shape, itype, ax, mask=[True, False], **kwargs*)
Bases: *poornn.core.Monitor*

Print data without changing anything.

Parameters

- **ax** (*<matplotib.axes>*) –
- **mask** (*list<bool>, len=2, default=[True,False]*) – masks for forward check and backward check.

ax
<matplotib.axes>

mask
list<bool>, len=2 – masks for forward check and backward check.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

class poornn.monitors.**Cache**(*input_shape*, *itype*, ***kwargs*)
Bases: *poornn.core.Monitor*

Cache data without changing anything.

forward_list
list – cached forward data.

backward_list
list – cached backward data.

clear()
clear history.

get_variables()
Get variables, return empty (1d but with length - 0) array.

num_variables
number of variables, which is fixed to 0.

set_runtime_vars(*var_dict={}*)
Set runtime variables for layers.

Parameters **var_dict** (*dict*) – the runtime variables dict.

set_variables(**args*, ***kwargs*)
passed.

utils

Module contents

poornn.utils.**take_slice**(*arr*, *sls*, *axis*)
take slices along specific axis.

Parameters

- **arr** (*ndarray*) – target array.
- **sls** (*slice*) – the target sector.
- **axis** (*int*) – the target axis.

Returns result array.

Return type ndarray

poornn.utils.**scan2csc**(*kernel_shape*, *img_in_shape*, *strides*, *boundary*)
Scan target shape with filter, and transform it into csc_matrix.

Parameters

- **kernel_shape** (*tuple*) – shape of kernel.
- **img_in_shape** (*tuple*) – shape of image dimension.
- **strides** (*tuple*) – strides for image dimensions.
- **boundary** (*'P'|'O'*) – boundary condition.

Returns indptr for csc maitrx, indices of csc matrix, output image shape.

Return type (1darray, 1darray, tuple)

poornn.utils.**typed_random**(*dtype*, *shape*)
generate a random numbers with specific data type.

Parameters

- **dtype** (*str*) – data type.
- **shape** (*tuple*) – shape of desired array.

Returns random array in 'F' order.

Return type ndarray

poornn.utils.**typed_randn**(*dtype*, *shape*)
generate a normal distributed random numbers with specific data type.

Parameters

- **dtype** (*str*) – data type.
- **shape** (*tuple*) – shape of desired array.

Returns random array in 'F' order.

Return type ndarray

poornn.utils.**typed_uniform**(*dtype*, *shape*, *low=-1.0*, *high=1.0*)
generate a uniformly distributed random numbers with specific data type.

Parameters

- **dtype** (*str*) – data type.
- **shape** (*tuple*) – shape of desired array.

Returns random array in 'F' order.

Return type ndarray

poornn.utils.**tuple_prod**(*tp*)
product over a tuple of numbers.

Parameters **tp** (*tuple*) – the target tuple to product over.

Returns product of tuple.

Return type number

poornn.utils.**masked_concatenate**(*vl*, *mask*)
concatenate multiple arrays with mask True.

Parameters

- **vl** (*list<ndarray>*) – arrays.
- **mask** (*list<bool>*) – masks for arrays.

Returns result array.

Return type ndarray

poornn.utils.**dtype2token**(*dtype*)
Parse data type to token.

Parameters **dtype** (*'float32'|'float64'|'complex64'|'complex128'*) – data type.

Returns 's'|'d'|'c'|'z'

Return type str

poornn.utils.**dtype_c2r**(*complex_dtype*)
Get corresponding real data type from complex data type.

Parameters **dtype** (*'complex64'|'complex128'*) – data type.

Returns ('float32'|'float64')

Return type str

poornn.utils.**dtype_r2c**(*real_dtype*)
Get corresponding complex data type from real data type :param dtype: data type. :type dtype: 'float32'|'float64'

Returns ('complex64'|'complex128')

Return type str

poornn.utils.**complex_backward**(*dz*, *dzc*)
Complex propagation rule.

Parameters

- **dz** (*ndarray*) – $\partial J/\partial z$
- **dzc** (*ndarray*) – $\partial J/\partial z^*$

Returns backward function that take xy and dy as input.

Return type func

poornn.utils.**fsign**(*x*)
sign function that work properly for complex numbers $x/|x|$,

Parameters **x** (*ndarray*) – input array.

Returns sign of x.

Return type ndarray

Note: if x is 0, return 0.

visualize

Module contents

Visualization for neural networks.

poornn.visualize.**viznn**(*nn*, *filename=None*)
Visualize a neural network using pygraphviz.

Parameters

- **nn** (Layer) – target network
- **filename** (*str*) – specify filename to save result, default is "G.sv"

Returns graphviz diagram instance.

Return type AGraph

Python Module Index

p

Index

Symbols

A

B

C

D

E

L

M

N

O

P

R

S

T

V

W