
onnx-chainer

Release 1.6.0

Nov 06, 2019

1	Introduction	1
1.1	Installation	1
1.2	Quick Start	1
1.3	Supported Functions	2
1.4	Tested Environments	4
1.5	Run Test	5
1.6	Contribution	5
2	Module Reference	7
2.1	Export	7
2.2	Export Utilities	9
2.3	Convert Utilities	10
2.4	Testing Utilities	13
3	Indices and tables	17
	Python Module Index	19
	Index	21

ONNX-Chainer is add-on package for ONNX, converts Chainer model to ONNX format, export it.

1.1 Installation

Use pip via PyPI:

```
$ pip install onnx-chainer
```

Or build from a cloned Git repository.:

```
$ git clone https://github.com/chainer/onnx-chainer.git
$ cd onnx-chainer
$ pip install -e .
```

1.2 Quick Start

First, install ChainerCV to get the pre-trained models.

```
import numpy as np

import chainer
import chainercv.links as C
import onnx_chainer

model = C.VGG16(pretrained_model='imagenet')

# Pseudo input
x = np.zeros((1, 3, 224, 224), dtype=np.float32)

onnx_chainer.export(model, x, filename='vgg16.onnx')
```

vgg16.onnx file will be exported.

Other export examples are put on [examples](#). Please check them.

1.3 Supported Functions

Currently 82 Chainer Functions are supported to export in ONNX format.

Activation

- ClippedReLU
- ELU
- HardSigmoid
- LeakyReLU
- LogSoftmax
- PReLUFunction
- ReLU
- Sigmoid
- Softmax
- Softplus
- Tanh

Array

- Cast
- Concat
- Copy
- Depth2Space
- Dstack
- ExpandDims
- GetItem
- Hstack
- Pad^{1,2}
- Permutate
- Repeat
- Reshape
- ResizeImages
- Separate
- Shape⁵

¹ mode should be either 'constant', 'reflect', or 'edge'

² ONNX doesn't support multiple constant values for Pad operation

⁵ Chainer doesn't support Shape function

- Space2Depth
- SplitAxis
- Squeeze
- Stack
- Swapaxes
- Tile
- Transpose
- Vstack
- Where

Connection

- Convolution2DFunction
- ConvolutionND
- Deconvolution2DFunction
- DeconvolutionND
- EmbedIDFunction³
- LinearFunction

Loss

- SoftmaxCrossEntropy

Math

- Absolute
- Add
- AddConstant
- ArgMax
- ArgMin
- BroadcastTo
- Clip
- Div
- DivFromConstant
- Exp
- Identity
- LinearInterpolate
- LogSumExp
- MatMul
- Max
- Maximum

³ Current ONNX doesn't support ignore_label for EmbedID

- Mean
- Min
- Minimum
- Mul
- MulConstant
- Neg
- PowConstVar
- PowVarConst
- PowVarVar
- Prod
- RsqrtGPU
- Sqrt
- Square
- Sub
- SubFromConstant
- Sum

Noise

- Dropout⁴

Normalization

- BatchNormalization
- FixedBatchNormalization
- LocalResponseNormalization
- NormalizeL2

Pooling

- AveragePooling2D
- AveragePoolingND
- MaxPooling2D
- MaxPoolingND
- ROIpooling2D
- Unpooling2D

1.4 Tested Environments

- OS
 - Ubuntu 16.04, 18.04

⁴ In test mode, all dropout layers aren't included in the exported file

- Windows 10
- Python 3.5.5, 3.6.7, 3.7.2
- ONNX 1.4.1, 1.5.0, 1.6.0
 - opset version 7, 8, 9, 10, 11
- Chainer 6.5.0
- ONNX-Runtime 1.0.0

1.5 Run Test

1.5.1 1. Install test modules

First, test modules for testing:

```
$ pip install onnx-chainer[test-cpu]
```

on GPU environment:

```
$ pip install cupy # or cupy-cudaXX is useful
$ pip install onnx-chainer[test-gpu]
```

1.5.2 2. Run tests

Next, run pytest:

```
$ pytest -m "not gpu"
```

on GPU environment:

```
$ pytest
```

1.6 Contribution

Any contribution to ONNX-Chainer is welcome!

- Python codes follow [Chainer Coding Guidelines](#)

2.1 Export

ONNX-Chainer exports Chainer model to ONNX graph with various options.

<code>onnx_chainer.export</code>	Export function for <code>chainer.Chain</code> in ONNX format.
<code>onnx_chainer.export_testcase</code>	Export model and I/O tensors of the model in protobuf format.

2.1.1 `onnx_chainer.export`

`onnx_chainer.export` (*model*, *args*, *filename=None*, *export_params=True*, *graph_name='Graph'*, *save_text=False*, *opset_version=None*, *input_names=None*, *output_names=None*, *train=False*, *return_named_inout=False*, *external_converters=None*, *external_opset_imports=None*, *input_shapes=None*)

Export function for `chainer.Chain` in ONNX format.

This function performs a forward computation of the given `Chain`, `model`, by passing the given arguments `args` directly. It means, the output `Variable` object `y` to make the computational graph will be created by:

```
y = model(*args)
```

`external_converters` and `external_opset_imports` are for external custom operator. When some `~chainer.FunctionNode` are expected to convert to own customized operator, set converter function with `~chainer.FunctionNode` name.

```
>>> import onnx
>>> def custom_converter(param):
...     return onnx.helper.make_node(
...         'CustomizedRelu', param.input_names, param.output_names,
...         domain='chainer'),
>>>
```

(continues on next page)

```

>>> external_converters = {'ReLU': custom_converter}
>>> external_imports = {'chainer': 0}
>>>
>>> model = chainer.Sequential(F.relu) # set the target model
>>> args = chainer.Variable(np.random.rand(1,10)) # set dummy input
>>> onnx_graph = onnx_chainer.export(
...     model, args,
...     external_converters=external_converters,
...     external_opset_imports=external_imports)

```

Returned model has CustomizedRelu node.

Parameters

- **model** (*Chain*) – The model object you want to export in ONNX format. It should have `__call__()` method because the second argument `args` is directly given to the model by the `[]` accessor.
- **args** (*list or dict*) – The arguments which are given to the model directly.
- **filename** (*str or file-like object*) – The filename used for saving the resulting ONNX model. If `None`, nothing is saved to the disk.
- **export_params** (*bool*) – If `True`, this function exports all the parameters included in the given model at the same time. If `False`, the exported ONNX model doesn't include any parameter values.
- **graph_name** (*str*) – A string to be used for the `name` field of the graph in the exported ONNX model.
- **save_text** (*bool*) – If `True`, the text format of the output ONNX model is also saved with `.txt` extension.
- **opset_version** (*int*) – The operator set version of ONNX. If not specified or `None` is given, the latest opset version of the `onnx` module is used. If an integer is given, it will be ensured that all the operator version in the exported ONNX file is less than this value.
- **input_names** (*str, list or dict*) – Customize input names of the graph. Number of `input_names` must be same as number of `args`. When set dict type, keys must be same as `args`'s keys.
- **output_names** (*str, list or dict*) – Customize output name of the graph. Number of `output_names` must be same as actual outputs from `model`. When set dict type, keys must be same as the key of `model` output.
- **train** (*bool*) – If `True`, output computational graph with train mode.
- **return_named_inout** (*bool*) – If set `True`, return ONNX model with named inputs, and named outputs.
- **external_converters** (*dict*) – Add-on converter. Convert functions keyed by `~chainer.FunctionNode` name.
- **external_opset_imports** (*dict*) – Import external opset. opset version number keyed by domain name.
- **input_shapes** (*tuple, list, dict*) – Input shape of output graph follows the customized shapes if set. When input are collection type, set list or dict. Tuple of tuple is not allowed.

Returns When `return_named_inout` is `False`, return `ModelProto` as an ONNX model. Otherwise return the tuple of `ModelProto`, named inputs and outputs, both inputs and outputs are list of `~chainer.Variable`.

Return type `ModelProto` or tuple

2.1.2 onnx_chainer.export_testcase

`onnx_chainer.export_testcase(model, args, out_dir, output_grad=False, **kwargs)`

Export model and I/O tensors of the model in protobuf format.

Similar to the `export` function, this function first performs a forward computation to a given input for obtaining an output. Then, this function saves the pair of input and output in Protobuf format, which is a defacto-standard format in ONNX.

This function also saves the model with the name “model.onnx”.

Parameters

- **model** (*Chain*) – The model object.
- **args** (*list*) – The arguments which are given to the model directly. Unlike `export` function, only *list* type is accepted.
- **out_dir** (*str*) – The directory name used for saving the input and output.
- **output_grad** (*bool*) – If `True`, this function will output model’s gradient with names ‘gradient_%d.pb’.
- ****kwargs** (*dict*) – keyword arguments for `onnx_chainer.export`.

2.2 Export Utilities

ONNX-Chainer provides some utility functions to help exporting.

<code>onnx_chainer.replace_func.</code>	The target function fakes <code>FunctionNode</code>
---	---

<code>fake_as_funcnode</code>	
-------------------------------	--

<code>onnx_chainer.replace_func.as_funcnode</code>	The target function fakes <code>FunctionNode</code>
--	---

2.2.1 onnx_chainer.replace_func.fake_as_funcnode

`onnx_chainer.replace_func.fake_as_funcnode(alt_func, name, rename_attributes=None)`

The target function fakes `FunctionNode`

The target function is replaced to the alternative function to connect variable node by acting function node. `alt_func` must satisfy the following restrictions.

1. Inputs includes one or more `chainer.Variable` to trace variables.
2. Output consists nothing but `ndarray` or `chainer.Variable`

Even if `alt_func` returns `ndarray`, the value forced to be converted to `chainer.Variable`. A caller of the target function have to care both cases, returning `ndarray` and `chainer.Variable`.

When `alt_func` returns list of variable, the wrapped function will also returns multiple variables as tuple. However dict cannot be return, the wrapped function breaks down the returned values as tuple of values, keys will be ignored.

Arguments of `alt_func` except for `chainer.Variable` are set as function attributes. Attribute names are set `argN` (N is index number) or keyword on default.

Example

```
>>> def func(x, a, b, c=1, d=2): pass
>>> # x is variable
>>> func = onnx_chainer.replace_func.fake_as_funcnode(
...     func, 'CustomNode',
...     rename_attributes=[(1, 'value'), ('c', 'y')])
```

Then `func` will be operated as a function node named “CustomNode”, and `'value', 'b', 'y', 'd'` are set as function’s attributes. See `tests/test_replace_func.py` more details.

Parameters

- **alt_func** (*func*) – actual called function. There are some constrains, see the above documentation.
- **name** (*str*) – function name. This name is used for what ONNX operator to be assigned.
- **rename_attributes** (*list or tuple*) – rename attribute name, set list of tuple(*index_of_args*, *new_name*) or tuple(*kwargs_name*, *new_name*)

Returns wrapped function, called on exporting.

Return type `func`

2.2.2 onnx_chainer.replace_func.as_funcnode

`onnx_chainer.replace_func.as_funcnode` (*name*, *rename_attributes=None*)

The target function fakes `FunctionNode`

The target function is overwrapped to connect variable node by acting function node. Expected to be used as decorator. More detail, see `fake_as_funcnode` documentation.

Example

```
>>> @onnx_chainer.replace_func.as_funcnode(
...     'CustomNode', rename_attributes=[(1, 'value'), ('c', 'y')])
... def func(x, a, b, c=1, d=2): pass
```

Parameters

- **name** (*str*) – function name. This name is used for what ONNX operator to be assigned.
- **rename_attributes** (*list or tuple*) – rename attribute name, set list of tuple(*index_of_args*, *new_name*) or tuple(*kwargs_name*, *new_name*)

2.3 Convert Utilities

These utilities helps converting from Chainer model to ONNX format, mainly used them internally.

*onnx_chainer.context.Context*Context of converter

2.3.1 onnx_chainer.context.Context

class onnx_chainer.context.Context (*model*)

Context of converter

This context shares names during exporting.

name_listlist of being exported as ONNX node name with pinned or not, keyed by instance ID. When the target variable is `chainer.Variable` or `chainer.Parameter`, instance ID of `ndarray` held by the variable is also put as key, because some functions like `F.where` internally unwrap variable.**Type** dict**Methods****add_const** (*array, name*)

Add a constant array as an ONNX Constant node.

Returns registered name.**Return type** str**add_param** (*array, name, use_original_name=False*)

Add a parameter array as an ONNX initializer.

Returns registered name.**Return type** str**get_link** (*param*)

Return link with name which has the param.

Parameters **param** (*chainer.Parameter*) – the target param.**Returns** name and link. returns `None` when not found.**Return type** tuple**get_name** (*variable*)**is_pinned** (*variable*)**set_name** (*variable, name, pinned=False*)

Set ONNX node name

Parameters

- **variable** (*var*) – target variable
- **name** (*str*) – name to be exported as ONNX node name
- **pinned** (*bool*) – if `True`, the name will not be overwritten in subsequence process.

*onnx_chainer.onnx_helper.
GraphBuilder*A helper class to build consecutive ONNX nodes.

Continued on next page

Table 4 – continued from previous page

<code>onnx_chainer.onnx_helper.set_func_name</code>	Set the name of Chainer function being converted.
<code>onnx_chainer.onnx_helper.get_func_name</code>	Return processing function name
<code>onnx_chainer.onnx_helper.make_node</code>	A thin wrapper of <code>onnx.helper.make_node</code> .
<code>onnx_chainer.onnx_helper.write_tensor_pb</code>	
<code>onnx_chainer.onnx_helper.cleanse_param_name</code>	Converts Chainer parameter names to ONNX names.

2.3.2 onnx_chainer.onnx_helper.GraphBuilder

class `onnx_chainer.onnx_helper.GraphBuilder`

A helper class to build consecutive ONNX nodes.

Methods

`node_name ()`

`nodes (output_names=None)`

Returns all nodes created so far.

Parameters `output_names (list of str)` – The names of output values to be set at the last node.

Returns A list of `onnx.NodeProto` objects, suitable as the return value of converter functions.

`op (op_name, input_names, num_outputs=1, **kwargs)`

Creates a new ONNX node and returns its outputs.

Parameters

- `op_name (str)` – The name of an ONNX op.
- `input_names (list of str)` – The names of input values.
- `num_outputs (int)` – The number of output values.
- `**kwargs (dict)` – ONNX attributes of the node.

Returns A str of the output name when `num_outputs` is 1. A tuple of str of the output names otherwise.

`op_output_named (op_name, input_names, output_names, **kwargs)`

Creates a new ONNX node with output names, and returns its outputs.

Parameters

- `op_name (str)` – The name of an ONNX op.
- `input_names (list of str)` – The names of input values.
- `output_names (int of str)` – The names of output values.
- `**kwargs (dict)` – ONNX attributes of the node.

Returns A str of the output name when number of output is 1. A tuple of str of the output names otherwise.

2.3.3 onnx_chainer.onnx_helper.set_func_name

`onnx_chainer.onnx_helper.set_func_name(func_name)`
Set the name of Chainer function being converted.

Parameters `func_name` (*str*) – The name of Chainer function.

2.3.4 onnx_chainer.onnx_helper.get_func_name

`onnx_chainer.onnx_helper.get_func_name()`
Return processing function name

2.3.5 onnx_chainer.onnx_helper.make_node

`onnx_chainer.onnx_helper.make_node(*args, **kwargs)`
A thin wrapper of `onnx.helper.make_node`.

Node name will be assigned automatically.

Parameters

- ***args** (*tuple*) – ONNX node parameters of the node
- ****kwargs** (*dict*) – ONNX attributes of the node.

Returns An `onnx.NodeProto` object.

2.3.6 onnx_chainer.onnx_helper.write_tensor_pb

`onnx_chainer.onnx_helper.write_tensor_pb(filename, name, value)`

2.3.7 onnx_chainer.onnx_helper.cleansse_param_name

`onnx_chainer.onnx_helper.cleansse_param_name(name)`
Converts Chainer parameter names to ONNX names.

Note ONNX identifiers must be a valid C identifier.

Parameters `name` (*str*) – A Chainer parameter name (e.g., //W).

Returns A valid ONNX name (e.g., param_I_W).

2.4 Testing Utilities

<code>onnx_chainer.testing.input_generator.increasing</code>	Returns a monotonically increasing ndarray for test inputs.
<code>onnx_chainer.testing.input_generator.nonzero_increasing</code>	Returns a monotonically increasing ndarray for test inputs.
<code>onnx_chainer.testing.input_generator.positive_increasing</code>	Returns a monotonically increasing ndarray for test inputs.

2.4.1 onnx_chainer.testing.input_generator.increasing

`onnx_chainer.testing.input_generator.increasing` (*shape, dtype=<class 'numpy.float32'>)

Returns a monotonically increasing ndarray for test inputs.

The output will contain both zero, negative numbers, and non integer numbers for float dtypes. A test writer is supposed to consider this function first.

Example:

```
>>> onnx_chainer.testing.input_generator.increasing(3, 4)
array([[ -3. , -2.5, -2. , -1.5],
       [ -1. , -0.5,  0. ,  0.5],
       [  1. ,  1.5,  2. ,  2.5]], dtype=float32)
```

Parameters

- **shape** (*tuple of int*) – The shape of the output array.
- **dtype** (*numpy.dtype*) – The dtype of the output array.

Returns `numpy.ndarray`

2.4.2 onnx_chainer.testing.input_generator.nonzero_increasing

`onnx_chainer.testing.input_generator.nonzero_increasing` (*shape, dtype=<class 'numpy.float32'>, bias=1e-07)

Returns a monotonically increasing ndarray for test inputs.

Similar to *increasing* but contains no zeros. Expected to be used for divisors.

Example:

```
>>> onnx_chainer.testing.input_generator.nonzero_increasing(3, 4)
array([[ -3.0000000e+00, -2.5000000e+00, -1.9999999e+00, -1.4999999e+00],
       [ -9.9999988e-01, -4.9999991e-01,  1.0000000e-07,  5.0000012e-01],
       [  1.0000001e+00,  1.5000001e+00,  2.0000000e+00,  2.5000000e+00]],
       dtype=float32)
```

Parameters

- **shape** (*tuple of int*) – The shape of the output array.
- **dtype** (*numpy.dtype*) – The dtype of the output array.
- **bias** (*float*) – The bias to avoid zero.

Returns `numpy.ndarray`

2.4.3 onnx_chainer.testing.input_generator.positive_increasing

`onnx_chainer.testing.input_generator.positive_increasing` (*shape, dtype=<class 'numpy.float32'>, bias=1e-07)

Returns a monotonically increasing ndarray for test inputs.

Similar to *increasing* but contains only positive numbers. Expected to be used for *math.log*, *math.sqrt*, etc.

Example:

```
>>> onnx_chainer.testing.input_generator.positive_increasing(3, 4)
array([[1.0000000e-07, 5.0000012e-01, 1.0000001e+00, 1.5000001e+00],
       [2.0000000e+00, 2.5000000e+00, 3.0000000e+00, 3.5000000e+00],
       [4.0000000e+00, 4.5000000e+00, 5.0000000e+00, 5.5000000e+00]],
      dtype=float32)
```

Parameters

- **shape** (*tuple of int*) – The shape of the output array.
- **dtype** (*numpy.dtype*) – The dtype of the output array.
- **bias** (*float*) – The bias to avoid zero.

Returns numpy.ndarray

CHAPTER 3

Indices and tables

- `genindex`
- `search`

O

`onnx_chainer`, 7

A

add_const() (*onnx_chainer.context.Context* method),
11

add_param() (*onnx_chainer.context.Context* method),
11

as_funcnode() (in module
onnx_chainer.replace_func), 10

C

cleanse_param_name() (in module
onnx_chainer.onnx_helper), 13

Context (class in *onnx_chainer.context*), 11

E

export() (in module *onnx_chainer*), 7

export_testcase() (in module *onnx_chainer*), 9

F

fake_as_funcnode() (in module
onnx_chainer.replace_func), 9

G

get_func_name() (in module
onnx_chainer.onnx_helper), 13

get_link() (*onnx_chainer.context.Context* method),
11

get_name() (*onnx_chainer.context.Context* method),
11

GraphBuilder (class in *onnx_chainer.onnx_helper*),
12

I

increasing() (in module
onnx_chainer.testing.input_generator), 14

is_pinned() (*onnx_chainer.context.Context* method),
11

M

make_node() (in module *onnx_chainer.onnx_helper*),
13

N

name_list (*onnx_chainer.context.Context* attribute),
11

node_name() (*onnx_chainer.onnx_helper.GraphBuilder*
method), 12

nodes() (*onnx_chainer.onnx_helper.GraphBuilder*
method), 12

nonzero_increasing() (in module
onnx_chainer.testing.input_generator), 14

O

onnx_chainer (module), 7

op() (*onnx_chainer.onnx_helper.GraphBuilder*
method), 12

op_output_named() (*onnx_chainer.onnx_helper.GraphBuilder*
method), 12

P

positive_increasing() (in module
onnx_chainer.testing.input_generator), 14

S

set_func_name() (in module
onnx_chainer.onnx_helper), 13

set_name() (*onnx_chainer.context.Context* method),
11

W

write_tensor_pb() (in module
onnx_chainer.onnx_helper), 13