MXNet (Https://Mxnet.Incubator.Apache.Org) › Autograd

Autograd Package

Overview

The **`autograd`** package enables automatic differentiation of NDArray operations. In machine learning applications, **`autograd`** is often used to calculate the gradients of loss functions with respect to parameters.

Record vs Pause

`autograd` records computation history on the fly to calculate gradients later. This is only enabled inside a **`with autograd.record():`** block. A **`with auto_grad.pause()`** block can be used inside a **`record()`** block to temporarily disable recording.

To compute gradient with respect to an **`NDArray x`**, first call **`x.attach_grad()`** to allocate space for the gradient. Then, start a **`with autograd.record()`** block, and do some computation. Finally, call **`backward()`** on the result:

```
>>> x = mx.nd.array([1,2,3,4])
>>> x.attach_grad()
>>> with mx.autograd.record():
...     y = x * x + 1
>>> y.backward()
>>> print(x.grad)
[ 2.  4.  6.  8.]
```

Train mode and Predict Mode

Some operators (Dropout, BatchNorm, etc) behave differently in training and making predictions. This can be controlled with **`train_mode`** and **`predict_mode`** scope.

By default, MXNet is in **`predict_mode`**. A **`with autograd.record()`** block by default turns on **`train_mode`** (equivalent to **`with autograd.record(train_mode=True)`**). To compute a gradient in prediction mode (as when generating adversarial examples), call record with **`train_mode=False`** and then call **`backward(train_mode=False)`**

Although training usually coincides with recording, this isn't always the case. To control *training* vs *predict_mode* without changing *recording* vs *not recording*, use a `with autograd.train_mode():` or `with autograd.predict_mode():` block.

Detailed tutorials are available in Part 1 of the MXNet gluon book (http://gluon.mxnet.io/).

Autograd

`record`	Returns an autograd recording scope context to be used in 'with' statement and captures code that needs gradients to be calculated.
`pause`	Returns a scope context to be used in 'with' statement for codes that do not need gradients to be calculated.
`train_mode`	Returns a scope context to be used in 'with' statement in which forward pass behavior is set to training mode, without changing the recording states.
`predict_mode`	Returns a scope context to be used in 'with' statement in which forward pass behavior is set to inference mode, without changing the recording states.
`backward`	Compute the gradients of heads w.r.t previously marked variables.
`set_training`	Set status to training/predicting.
`is_training`	Get status on training/predicting.
`set_recording`	Set status to recording/not recording.
`is_recording`	Get status on recording/not recording.
`mark_variables`	Mark NDArrays as variables to compute gradient for autograd.
`Function`	Customize differentiation in autograd.

API Reference

Autograd for NDArray.

`mxnet.autograd.` **`set_recording`** **(**is_recording**)** **[source] (../../../_modules/mxnet/autograd.html#set_recording)**

Set status to recording/not recording. When recording, graph will be constructed for gradient computation.

Parameters: **is_recording** (*bool*) –

Returns:

Return type: previous state before this set.

`mxnet.autograd.` **`set_training`** (train_mode) [source] (../../../_modules/mxnet/autograd.html#set_training)

Set status to training/predicting. This affects ctx.is_train in operator running context. For example, Dropout will drop inputs randomly when train_mode=True while simply passing through if train_mode=False.

Parameters: **train_mode** (*bool*) –

Returns:

Return type: previous state before this set.

`mxnet.autograd.` **`is_recording`** () [source] (../../../_modules/mxnet/autograd.html#is_recording)

Get status on recording/not recording.

Returns:

Return type: Current state of recording.

`mxnet.autograd.` **`is_training`** () [source] (../../../_modules/mxnet/autograd.html#is_training)

Get status on training/predicting.

Returns:

Return type: Current state of training/predicting.

`mxnet.autograd.` **`record`** (train_mode=True) [source] (../../../_modules/mxnet/autograd.html#record)

Returns an autograd recording scope context to be used in 'with' statement and captures code that needs gradients to be calculated.

> **Note**
>
> When forwarding with train_mode=False, the corresponding backward should also use train_mode=False, otherwise gradient is undefined.

Example:

```
with autograd.record():
    y = model(x)
    backward([y])
metric.update(...)
optim.step(...)
```

Parameters: **train_mode** (*bool, default True*) – Whether the forward pass is in training or predicting mode. This controls the behavior of some layers such as Dropout, BatchNorm.

`mxnet.autograd.` **`pause`** (train_mode=False) [source] (../../../_modules/mxnet/autograd.html#pause)

Returns a scope context to be used in 'with' statement for codes that do not need gradients to be calculated.

Example:

```
with autograd.record():
    y = model(x)
    backward([y])
    with autograd.pause():
        # testing, IO, gradient updates...
```

Parameters: **train_mode** (*bool, default False*) – Whether to do forward for training or predicting.

`mxnet.autograd.` **`train_mode`** () [source] (../../../_modules/mxnet/autograd.html#train_mode)

Returns a scope context to be used in 'with' statement in which forward pass behavior is set to training mode, without changing the recording states.

Example:

```
y = model(x)
with autograd.train_mode():
    y = dropout(y)
```

`mxnet.autograd.` **`predict_mode`** () [source] (../../../_modules/mxnet/autograd.html#predict_mode)

Returns a scope context to be used in 'with' statement in which forward pass behavior is set to inference mode, without changing the recording states.

Example:

```
with autograd.record():
    y = model(x)
    with autograd.predict_mode():
        y = sampling(y)
    backward([y])
```

mxnet.autograd. **mark_variables** (variables, gradients, grad_reqs='write') (../../../_modules/mxnet/autograd.html#mark_variables) [source]

Mark NDArrays as variables to compute gradient for autograd.

Parameters:

- **variables** (*NDArray* (../ndarray/ndarray.html#mxnet.ndarray.NDArray) *or list of NDArray*) –
- **gradients** (*NDArray* (../ndarray/ndarray.html#mxnet.ndarray.NDArray) *or list of NDArray*) –
- **grad_reqs** (*str or list of str*) –

mxnet.autograd. **backward** (heads, head_grads=None, retain_graph=False, train_mode=True) (../../../_modules/mxnet/autograd.html#backward) [source]

Compute the gradients of heads w.r.t previously marked variables.

Parameters:

- **heads** (*NDArray* (../ndarray/ndarray.html#mxnet.ndarray.NDArray) *or list of NDArray*) – Output NDArray(s)
- **head_grads** (*NDArray* (../ndarray/ndarray.html#mxnet.ndarray.NDArray) *or list of NDArray or None*) – Gradients with respect to heads.
- **train_mode** (*bool, optional*) – Whether to do backward for training or predicting.

mxnet.autograd. **grad** (heads, variables, head_grads=None, retain_graph=None, create_graph=False, train_mode=True) (../../../_modules/mxnet/autograd.html#grad) [source]

Compute the gradients of heads w.r.t variables. Gradients will be returned as new NDArrays instead of stored into *variable.grad*. Supports recording gradient graph for computing higher order gradients.

> **Note**
>
> Currently only a very limited set of operators support higher order gradients.

Parameters:

- **heads** (*NDArray* (../ndarray/ndarray.html#mxnet.ndarray.NDArray) *or list of NDArray*) – Output NDArray(s)
- **variables** (*NDArray* (../ndarray/ndarray.html#mxnet.ndarray.NDArray) *or list of NDArray*) – Input variables to compute gradients for.
- **head_grads** (*NDArray* (../ndarray/ndarray.html#mxnet.ndarray.NDArray) *or list of NDArray or None*) – Gradients with respect to heads.
- **retain_graph** (*bool*) – Whether to keep computation graph to differentiate again, instead of clearing history and release memory. Defaults to the same value as create_graph.
- **create_graph** (*bool*) – Whether to record gradient graph for computing higher order
- **train_mode** (*bool, optional*) – Whether to do backward for training or prediction.

Returns: Gradients with respect to variables.

Return type: NDArray (../ndarray/ndarray.html#mxnet.ndarray.NDArray) or list of NDArray

Examples

```
>>> x = mx.nd.ones((1,))
>>> x.attach_grad()
>>> with mx.autograd.record():
...     z = mx.nd.elemwise_add(mx.nd.exp(x), x)
>>> dx = mx.autograd.grad(z, [x], create_graph=True)
>>> print(dx)
[
[ 3.71828175]
]
```

mxnet.autograd. **get_symbol** (x) (../../../_modules/mxnet/autograd.html#get_symbol) **[source]**

Retrieve recorded computation history as *Symbol*.

Parameters: **x** (*NDArray* (../ndarray/ndarray.html#mxnet.ndarray.NDArray)) – Array representing the head of computation graph.

Returns: The retrieved Symbol.

Return type: Symbol (../symbol/symbol.html#mxnet.symbol.Symbol) class **mxnet.autograd. Function (../../../_modules/mxnet/autograd.html#Function)** **[source]**

Customize differentiation in autograd.

If you don’t want to use the gradients computed by the default chain-rule, you can use Function to customize differentiation for computation. You define your computation in the forward method and provide the customized differentiation in the backward method. During gradient computation, autograd will use the user-defined backward function instead of the default chain-rule. You can also cast to numpy array and back for some operations in forward and backward.

For example, a stable sigmoid function can be defined as:

```
class sigmoid(mx.autograd.Function):
    def forward(self, x):
        y = 1 / (1 + mx.nd.exp(-x))
        self.save_for_backward(y)
        return y

    def backward(self, dy):
        # backward takes as many inputs as forward's return value,
        # and returns as many NDArrays as forward's arguments.
        y, = self.saved_tensors
        return dy * y * (1-y)
```

Then, the function can be used in the following way:

```
func = sigmoid()
x = mx.nd.random.uniform(shape=(10,))
x.attach_grad()

with mx.autograd.record():
    m = func(x)
    m.backward()
dx = x.grad.asnumpy()
```

forward (*inputs) (../../../_modules/mxnet/autograd.html#Function.forward) **[source]**

Forward computation.

backward (*output_grads) (../../../_modules/mxnet/autograd.html#Function.backward) **[source]**

Backward computation.

Takes as many inputs as forward’s outputs, and returns as many NDArrays as forward’s inputs.