Automatic differentiation package - torch.autograd

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 7)

Unknown directive type "automodule".

```
.. automodule:: torch.autograd
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 8)

Unknown directive type "currentmodule".

```
.. currentmodule:: torch.autograd
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 10)

Unknown directive type "autosummary".

```
.. autosummary::
    :toctree: generated
    :nosignatures:

    backward
    grad
```

Forward-mode Automatic Differentiation

Warning

This API is in beta. Even though the function signatures are very unlikely to change, improved operator coverage is planned before we consider this stable.

Please see the forward-mode AD tutorial for detailed steps on how to use this API.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 29)

Unknown directive type "autosummary".

```
.. autosummary::
    :toctree: generated
    :nosignatures:

    forward_ad.dual_level
    forward_ad.make_dual
    forward_ad.unpack_dual
```

Functional higher level API

Warning

This API is in beta. Even though the function signatures are very unlikely to change, major improvements to performances are planned before we consider this stable.

This section contains the higher level API for the autograd that builds on the basic API above and allows you to compute jacobians, hessians, etc.

This API works with user-provided functions that take only Tensors as input and return only Tensors. If your function takes other arguments that are not Tensors or Tensors that don't have requires_grad set, you can use a lambda to capture them. For example, for a function `f` that takes three inputs, a Tensor for which we want the jacobian, another tensor that should be considered constant and a boolean flag as `f(input, constant, flag=flag)` you can use it as `functional.jacobian(lambda x: f(x, constant,`

flag=flag), input).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 57)

Unknown directive type "autosummary".

```
.. autosummary::
    :toctree: generated
    :nosignatures:

    functional.jacobian
    functional.hessian
    functional.vjp
    functional.jvp
    functional.vhp
    functional.hvp
```

Locally disabling gradient computation

See :ref:`locally-disable-grad-doc` for more information on the differences between no-grad and inference mode as well as other related mechanisms that may be confused with the two.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 73); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 77)

Unknown directive type "autosummary".

```
.. autosummary::
    :toctree: generated
    :nosignatures:

    no_grad
    enable_grad
    set_grad_enabled
    inference_mode
```

Default gradient layouts

When a non-sparse `param` receives a non-sparse gradient during :func:`torch.autograd.backward` or :func:`torch.Tensor.backward` `param.grad` is accumulated as follows.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 91); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 91); *backlink*

Unknown interpreted text role "func".

If `param.grad` is initially `None`:

1. If `param`'s memory is non-overlapping and dense, `.grad` is created with strides matching `param` (thus matching `param`'s layout).
2. Otherwise, `.grad` is created with rowmajor-contiguous strides.

If `param` already has a non-sparse `.grad` attribute:

3. If `create_graph=False`, `backward()` accumulates into `.grad` in-place, which preserves its strides.
4. If `create_graph=True`, `backward()` replaces `.grad` with a new tensor `.grad + new grad`, which attempts (but does not guarantee) matching the preexisting `.grad`'s strides.

The default behavior (letting `.grad`s be `None` before the first `backward()`, such that their layout is created according to 1 or 2, and

retained over time according to 3 or 4) is recommended for best performance. Calls to `model.zero_grad()` or `optimizer.zero_grad()` will not affect `.grad` layouts.

In fact, resetting all `.grad`s to `None` before each accumulation phase, e.g.:

```
for iterations...
    ...
    for param in model.parameters():
        param.grad = None
    loss.backward()
```

such that they're recreated according to 1 or 2 every time, is a valid alternative to `model.zero_grad()` or `optimizer.zero_grad()` that may improve performance for some networks.

Manual gradient layouts

If you need manual control over `.grad`'s strides, assign `param.grad =` a zeroed tensor with desired strides before the first `backward()`, and never reset it to `None`. 3 guarantees your layout is preserved as long as `create_graph=False`. 4 indicates your layout is *likely* preserved even if `create_graph=True`.

In-place operations on Tensors

Supporting in-place operations in autograd is a hard matter, and we discourage their use in most cases. Autograd's aggressive buffer freeing and reuse makes it very efficient and there are very few occasions when in-place operations actually lower memory usage by any significant amount. Unless you're operating under heavy memory pressure, you might never need to use them.

In-place correctness checks

All :class:`Tensor` s keep track of in-place operations applied to them, and if the implementation detects that a tensor was saved for backward in one of the functions, but it was modified in-place afterwards, an error will be raised once backward pass is started. This ensures that if you're using in-place functions and not seeing any errors, you can be sure that the computed gradients are correct.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 150); *backlink***
>
> Unknown interpreted text role "class".

Variable (deprecated)

> **Warning**
>
> The Variable API has been deprecated: Variables are no longer necessary to use autograd with tensors. Autograd automatically supports Tensors with `requires_grad` set to `True`. Below please find a quick guide on what has changed:
>
> - `Variable(tensor)` and `Variable(tensor, requires_grad)` still work as expected, but they return Tensors instead of Variables.
> - `var.data` is the same thing as `tensor.data`.
> - Methods such as `var.backward(), var.detach(), var.register_hook()` now work on tensors with the same method names.
>
> In addition, one can now create tensors with `requires_grad=True` using factory methods such as :func:`torch.randn`, :func:`torch.zeros`, :func:`torch.ones`, and others like the following:
>
> > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 172); *backlink***
> >
> > Unknown interpreted text role "func".
>
> > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 172); *backlink***
> >
> > Unknown interpreted text role "func".
>
> > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 172); *backlink***

Unknown interpreted text role "func".

```
autograd_tensor = torch.randn((2, 3, 4), requires_grad=True)
```

Tensor autograd functions

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 180)

Unknown directive type "autosummary".

```
.. autosummary::
    :nosignatures:

   torch.Tensor.grad
   torch.Tensor.requires_grad
   torch.Tensor.is_leaf
   torch.Tensor.backward
   torch.Tensor.detach
   torch.Tensor.detach_
   torch.Tensor.register_hook
   torch.Tensor.retain_grad
```

Function

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 195)

Unknown directive type "autoclass".

```
.. autoclass:: Function
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 197)

Unknown directive type "autosummary".

```
.. autosummary::
    :toctree: generated
    :nosignatures:

    Function.forward
    Function.backward
    Function.jvp
```

Context method mixins

When creating a new :class:`Function`, the following methods are available to *ctx*.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 207); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 209)

Unknown directive type "autosummary".

```
.. autosummary::
    :toctree: generated
    :nosignatures:

    function.FunctionCtx.mark_dirty
    function.FunctionCtx.mark_non_differentiable
    function.FunctionCtx.save_for_backward
    function.FunctionCtx.set_materialize_grads
```

Numerical gradient checking

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 223)

Unknown directive type "autosummary".

```
.. autosummary::
    :toctree: generated
    :nosignatures:

    gradcheck
    gradgradcheck
```

Profiler

Autograd includes a profiler that lets you inspect the cost of different operators inside your model - both on the CPU and GPU. There are two modes implemented at the moment - CPU-only using :class:`~torch.autograd.profiler.profile`. and nvprof based (registers both CPU and GPU activity) using :class:`~torch.autograd.profiler.emit_nvtx`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 233); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 233); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 239)

Unknown directive type "autoclass".

```
.. autoclass:: torch.autograd.profiler.profile
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 241)

Unknown directive type "autosummary".

```
.. autosummary::
    :toctree: generated
    :nosignatures:

    profiler.profile.export_chrome_trace
    profiler.profile.key_averages
    profiler.profile.self_cpu_time_total
    profiler.profile.total_average
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 250)

Unknown directive type "autoclass".

```
.. autoclass:: torch.autograd.profiler.emit_nvtx
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 253)

Unknown directive type "autosummary".

```
.. autosummary::
    :toctree: generated
    :nosignatures:
```

```
profiler.load_nvprof
```

Anomaly detection

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 262)

Unknown directive type "autoclass".

```
.. autoclass:: detect_anomaly
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 264)

Unknown directive type "autoclass".

```
.. autoclass:: set_detect_anomaly
```

Saved tensors default hooks

Some operations need intermediary results to be saved during the forward pass in order to execute the backward pass. You can define how these saved tensors should be packed / unpacked using hooks. A common application is to trade compute for memory by saving those intermediary results to disk or to CPU instead of leaving them on the GPU. This is especially useful if you notice your model fits on GPU during evaluation, but not training. Also see :ref:`saved-tensors-hooks-doc`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 270); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 278)

Unknown directive type "autoclass".

```
.. autoclass:: torch.autograd.graph.saved_tensors_hooks
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master][docs][source]autograd.rst, line 280)

Unknown directive type "autoclass".

```
.. autoclass:: torch.autograd.graph.save_on_cpu
```