

Tensor Attributes

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\pytorch-master [docs] [source] tensor_attributes.rst, line 1)

Unknown directive type "currentmodule".

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

Each `torch.Tensor` has a `:class:`torch.dtype``, `:class:`torch.device``, and `:class:`torch.layout``.

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

Unknown interpreted text role "class".

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

Unknown interpreted text role "class".

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

Unknown interpreted text role "class".

torch.dtype

A `:class:`torch.dtype`` is an object that represents the data type of a `:class:`torch.Tensor``. PyTorch has twelve different data types:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\pytorch-master [docs] [source] tensor_attributes.rst, line 17); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\pytorch-master [docs] [source] tensor_attributes.rst, line 17); [backlink](#)

Unknown interpreted text role "class".

| Data type | dtype | Legacy Constructors |
|---------------------------|---|-----------------------------------|
| 32-bit floating point | <code>torch.float32</code> or <code>torch.float</code> | <code>torch.FloatTensor</code> |
| 64-bit floating point | <code>torch.float64</code> or <code>torch.double</code> | <code>torch.DoubleTensor</code> |
| 64-bit complex | <code>torch.complex64</code> or <code>torch.cfloat</code> | |
| 128-bit complex | <code>torch.complex128</code> or <code>torch.cdouble</code> | |
| 16-bit floating point [1] | <code>torch.float16</code> or <code>torch.half</code> | <code>torch.HalfTensor</code> |
| 16-bit floating point [2] | <code>torch.bfloat16</code> | <code>torch.BFloat16Tensor</code> |
| 8-bit integer (unsigned) | <code>torch.uint8</code> | <code>torch.ByteTensor</code> |
| 8-bit integer (signed) | <code>torch.int8</code> | <code>torch.CharTensor</code> |
| 16-bit integer (signed) | <code>torch.int16</code> or <code>torch.short</code> | <code>torch.ShortTensor</code> |
| 32-bit integer (signed) | <code>torch.int32</code> or <code>torch.int</code> | <code>torch.IntTensor</code> |
| 64-bit integer (signed) | <code>torch.int64</code> or <code>torch.long</code> | <code>torch.LongTensor</code> |
| Boolean | <code>torch.bool</code> | <code>torch.BoolTensor</code> |

[1] Sometimes referred to as binary16: uses 1 sign, 5 exponent, and 10 significand bits. Useful when precision is important.

[2] Sometimes referred to as Brain Floating Point: use 1 sign, 8 exponent and 7 significand bits. Useful when range is important, since it has the same number of exponent bits as `float32`

To find out if a `:class:`torch.dtype`` is a floating point data type, the property `:attr:`is_floating_point`` can be used, which returns `True` if the data type is a floating point data type.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-

`master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 44); backlink`

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 44); [backlink](#)

Unknown interpreted text role "attr".

To find out if a `:class:`torch.dtype`` is a complex data type, the property `attr:`is_complex`` can be used, which returns `True` if the data type is a complex data type.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 47); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 47); [backlink](#)

Unknown interpreted text role "attr".

When the dtypes of inputs to an arithmetic operation (*add*, *sub*, *div*, *mul*) differ, we promote by finding the minimum dtype that satisfies the following rules:

- If the type of a scalar operand is of a higher category than tensor operands (where `complex > floating > integral > boolean`), we promote to a type with sufficient size to hold all scalar operands of that category.
- If a zero-dimension tensor operand has a higher category than dimensioned operands, we promote to a type with sufficient size and category to hold all zero-dim tensor operands of that category.
- If there are no higher-category zero-dim operands, we promote to a type with sufficient size and category to hold all dimensioned operands.

A floating point scalar operand has dtype `torch.get_default_dtype()` and an integral non-boolean scalar operand has dtype `torch.int64`. Unlike numpy, we do not inspect values when determining the minimum dtypes of an operand. Quantized and complex types are not yet supported.

Promotion Examples:

```
>>> float_tensor = torch.ones(1, dtype=torch.float)
>>> double_tensor = torch.ones(1, dtype=torch.double)
>>> complex_float_tensor = torch.ones(1, dtype=torch.complex64)
>>> complex_double_tensor = torch.ones(1, dtype=torch.complex128)
>>> int_tensor = torch.ones(1, dtype=torch.int)
>>> long_tensor = torch.ones(1, dtype=torch.long)
>>> uint_tensor = torch.ones(1, dtype=torch.uint8)
>>> double_tensor = torch.ones(1, dtype=torch.double)
>>> bool_tensor = torch.ones(1, dtype=torch.bool)
# zero-dim tensors
>>> long_zerodim = torch.tensor(1, dtype=torch.long)
>>> int_zerodim = torch.tensor(1, dtype=torch.int)

>>> torch.add(5, 5).dtype
torch.int64
# 5 is an int64, but does not have higher category than int_tensor so is not considered.
>>> (int_tensor + 5).dtype
torch.int32
>>> (int_tensor + long_zerodim).dtype
torch.int32
>>> (long_tensor + int_tensor).dtype
torch.int64
>>> (bool_tensor + long_tensor).dtype
torch.int64
>>> (bool_tensor + uint_tensor).dtype
torch.uint8
>>> (float_tensor + double_tensor).dtype
torch.float64
>>> (complex_float_tensor + complex_double_tensor).dtype
torch.complex128
>>> (bool_tensor + int_tensor).dtype
torch.int32
# Since long is a different kind than float, result dtype only needs to be large enough
# to hold the float.
>>> torch.add(long_tensor, float_tensor).dtype
torch.float32
```

When the output tensor of an arithmetic operation is specified, we allow casting to its *dtype* except that:

- An integral output tensor cannot accept a floating point tensor.
- A boolean output tensor cannot accept a non-boolean tensor.
- A non-complex output tensor cannot accept a complex tensor

Casting Examples:

```
# allowed:
>>> float_tensor *= float_tensor
>>> float_tensor *= int_tensor
>>> float_tensor *= uint_tensor
>>> float_tensor *= bool_tensor
>>> float_tensor *= double_tensor
>>> int_tensor *= long_tensor
>>> int_tensor *= uint_tensor
>>> uint_tensor *= int_tensor

# disallowed (RuntimeError: result type can't be cast to the desired output type):
>>> int_tensor *= float_tensor
>>> bool_tensor *= int_tensor
>>> bool_tensor *= uint_tensor
>>> float_tensor *= complex_float_tensor
```

torch.device

A `:class:`torch.device`` is an object representing the device on which a `:class:`torch.Tensor`` is or will be allocated.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 139); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 139); [backlink](#)

Unknown interpreted text role "class".

The `:class:`torch.device`` contains a device type ('cpu' or 'cuda') and optional device ordinal for the device type. If the device ordinal is not present, this object will always represent the current device for the device type, even after `:func:`torch.cuda.set_device()`` is called; e.g., a `:class:`torch.Tensor`` constructed with device 'cuda' is equivalent to 'cuda:X' where X is the result of `:func:`torch.cuda.current_device()``.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 142); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 142); [backlink](#)

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 142); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 142); [backlink](#)

Unknown interpreted text role "func".

A `:class:`torch.Tensor``'s device can be accessed via the `:attr:`Tensor.device`` property.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 148); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 148); [backlink](#)

Unknown interpreted text role "attr".

A `:class:`torch.device`` can be constructed via a string or via a string and device ordinal

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 150); [backlink](#)

Unknown interpreted text role "class".

Via a string:

```
>>> torch.device('cuda:0')
device(type='cuda', index=0)

>>> torch.device('cpu')
device(type='cpu')

>>> torch.device('cuda') # current cuda device
device(type='cuda')
```

Via a string and device ordinal:

```
>>> torch.device('cuda', 0)
device(type='cuda', index=0)

>>> torch.device('cpu', 0)
device(type='cpu', index=0)
```

Note

The `:class:`torch.device`` argument in functions can generally be substituted with a string. This allows for fast prototyping of code.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 175); [backlink](#)

Unknown interpreted text role "class".

```
>>> # Example of a function that takes in a torch.device
>>> cuda1 = torch.device('cuda:1')
>>> torch.randn((2,3), device=cuda1)

>>> # You can substitute the torch.device with a string
>>> torch.randn((2,3), device='cuda:1')
```

Note

For legacy reasons, a device can be constructed via a single device ordinal, which is treated as a cuda device. This matches `meth:`Tensor.get_device``, which returns an ordinal for cuda tensors and is not supported for cpu tensors.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 186); [backlink](#)

Unknown interpreted text role "meth".

```
>>> torch.device(1)
device(type='cuda', index=1)
```

Note

Methods which take a device will generally accept a (properly formatted) string or (legacy) integer device ordinal, i.e. the following are all equivalent:

```
>>> torch.randn((2,3), device=torch.device('cuda:1'))
>>> torch.randn((2,3), device='cuda:1')
```

```
>>> torch.randn((2,3), device=1) # legacy
```

torch.layout

Warning

The `torch.layout` class is in beta and subject to change.

A `:class:`torch.layout`` is an object that represents the memory layout of a `:class:`torch.Tensor``. Currently, we support `torch.strided` (dense Tensors) and have beta support for `torch.sparse_coo` (sparse COO Tensors).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 212); [backlink](#)

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 212); [backlink](#)

Unknown interpreted text role "class".

`torch.strided` represents dense Tensors and is the memory layout that is most commonly used. Each strided tensor has an associated `:class:`torch.Storage``, which holds its data. These tensors provide multi-dimensional, [strided](#) view of a storage. Strides are a list of integers: the k-th stride represents the jump in the memory necessary to go from one element to the next one in the k-th dimension of the Tensor. This concept makes it possible to perform many tensor operations efficiently.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 216); [backlink](#)

Unknown interpreted text role "class".

Example:

```
>>> x = torch.tensor([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
>>> x.stride()
(5, 1)

>>> x.t().stride()
(1, 5)
```

For more information on `torch.sparse_coo` tensors, see [:ref:`sparse-docs`](#).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\[pytorch-master] [docs] [source] tensor_attributes.rst, line 234); [backlink](#)

Unknown interpreted text role "ref".

torch.memory_format

A `:class:`torch.memory_format`` is an object representing the memory format on which a `:class:`torch.Tensor`` is or will be allocated.

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

Unknown interpreted text role "class".

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

Unknown interpreted text role "class".

Possible values are:

- `torch.contiguous_format`: Tensor is or will be allocated in dense non-overlapping memory. Strides represented by values in decreasing order.
- `torch.channels_last`: Tensor is or will be allocated in dense non-overlapping memory. Strides represented by values in

`strides[0] > strides[2] > strides[3] > strides[1] == 1` aka NHWC order.

- `torch.preserve_format`: Used in functions like *clone* to preserve the memory format of the input tensor. If input tensor is allocated in dense non-overlapping memory, the output tensor strides will be copied from the input. Otherwise output strides will follow `torch.contiguous_format`