PyTorch

Table of Contents

# TORCH

The torch package contains data structures for multi-dimensional tensors and defines mathematical operations over these tensors. Additionally, it provides many utilities for efficient serializing of Tensors and arbitrary types, and other useful utilities.

It has a CUDA counterpart, that enables you to run your tensor computations on an NVIDIA GPU with compute capability >= 3.0.

## Tensors

| | |
|---|---|
| `is_tensor` | Returns True if *obj* is a PyTorch tensor. |
| `is_storage` | Returns True if *obj* is a PyTorch storage object. |
| `is_complex` | Returns True if the data type of `input` is a complex data type i.e., one of `torch.complex64`, and `torch.complex128`. |
| `is_conj` | Returns True if the `input` is a conjugated tensor, i.e. its conjugate bit is set to *True*. |
| `is_floating_point` | Returns True if the data type of `input` is a floating point data type i.e., one of `torch.float64`, `torch.float32`, `torch.float16`, and `torch.bfloat16`. |
| `is_nonzero` | Returns True if the `input` is a single element tensor which is not equal to zero after type conversions. |
| `set_default_dtype` | Sets the default floating point dtype to `d`. |
| `get_default_dtype` | Get the current default floating point `torch.dtype`. |
| `set_default_tensor_type` | Sets the default `torch.Tensor` type to floating point tensor type `t`. |
| `numel` | Returns the total number of elements in the `input` tensor. |
| `set_printoptions` | Set options for printing. |
| `set_flush_denormal` | Disables denormal floating numbers on CPU. |

## Creation Ops

• NOTE

Random sampling creation ops are listed under Random sampling and include: `torch.rand()` `torch.rand_like()` `torch.randn()` `torch.randn_like()` `torch.randint()` `torch.randint_like()` `torch.randperm()` You may also use `torch.empty()` with the In-place random sampling methods to create `torch.Tensor` s with values sampled from a broader range of distributions.

| | |
|---|---|
| `tensor` | Constructs a tensor with no autograd history (also known as a "leaf tensor", see Autograd mechanics) by copying `data`. |
| `sparse_coo_tensor` | Constructs a sparse tensor in COO(rdinate) format with specified values at the given `indices`. |

| | |
|---|---|
| `asarray` | Converts `obj` to a tensor. |
| `as_tensor` | Converts data into a tensor, sharing data and preserving autograd history if possible. |
| `as_strided` | Create a view of an existing *torch.Tensor* `input` with specified `size`, `stride` and `storage_offset`. |
| `from_numpy` | Creates a `Tensor` from a `numpy.ndarray`. |
| `from_dlpack` | Converts a tensor from an external library into a `torch.Tensor`. |
| `frombuffer` | Creates a 1-dimensional `Tensor` from an object that implements the Python buffer protocol. |
| `zeros` | Returns a tensor filled with the scalar value *0*, with the shape defined by the variable argument `size`. |
| `zeros_like` | Returns a tensor filled with the scalar value *0*, with the same size as `input`. |
| `ones` | Returns a tensor filled with the scalar value *1*, with the shape defined by the variable argument `size`. |
| `ones_like` | Returns a tensor filled with the scalar value *1*, with the same size as `input`. |
| `arange` | Returns a 1-D tensor of size $\left\lceil \frac{\text{end} - \text{start}}{\text{step}} \right\rceil$ with values from the interval `[start, end)` taken with common difference `step` beginning from *start*. |
| `range` | Returns a 1-D tensor of size $\left\lfloor \frac{\text{end} - \text{start}}{\text{step}} \right\rfloor + 1$ with values from `start` to `end` with step `step`. |
| `linspace` | Creates a one-dimensional tensor of size `steps` whose values are evenly spaced from `start` to `end`, inclusive. |
| `logspace` | Creates a one-dimensional tensor of size `steps` whose values are evenly spaced from $\text{base}^{\text{start}}$ to $\text{base}^{\text{end}}$, inclusive, on a logarithmic scale with base `base`. |
| `eye` | Returns a 2-D tensor with ones on the diagonal and zeros elsewhere. |
| `empty` | Returns a tensor filled with uninitialized data. |
| `empty_like` | Returns an uninitialized tensor with the same size as `input`. |
| `empty_strided` | Creates a tensor with the specified `size` and `stride` and filled with undefined data. |
| `full` | Creates a tensor of size `size` filled with `fill_value`. |
| `full_like` | Returns a tensor with the same size as `input` filled with `fill_value`. |
| `quantize_per_tensor` | Converts a float tensor to a quantized tensor with given scale and zero point. |

| | |
|---|---|
| `quantize_per_channel` | Converts a float tensor to a per-channel quantized tensor with given scales and zero points. |
| `dequantize` | Returns an fp32 Tensor by dequantizing a quantized Tensor |
| `complex` | Constructs a complex tensor with its real part equal to `real` and its imaginary part equal to `imag`. |
| `polar` | Constructs a complex tensor whose elements are Cartesian coordinates corresponding to the polar coordinates with absolute value `abs` and angle `angle`. |
| `heaviside` | Computes the Heaviside step function for each element in `input`. |

## Indexing, Slicing, Joining, Mutating Ops

| | |
|---|---|
| `adjoint` | Returns a view of the tensor conjugated and with the last two dimensions transposed. |
| `argwhere` | Returns a tensor containing the indices of all non-zero elements of `input`. |
| `cat` | Concatenates the given sequence of `seq` tensors in the given dimension. |
| `concat` | Alias of `torch.cat()`. |
| `concatenate` | Alias of `torch.cat()`. |
| `conj` | Returns a view of `input` with a flipped conjugate bit. |
| `chunk` | Attempts to split a tensor into the specified number of chunks. |
| `dsplit` | Splits `input`, a tensor with three or more dimensions, into multiple tensors depthwise according to `indices_or_sections`. |
| `column_stack` | Creates a new tensor by horizontally stacking the tensors in `tensors`. |
| `dstack` | Stack tensors in sequence depthwise (along third axis). |
| `gather` | Gathers values along an axis specified by *dim*. |
| `hsplit` | Splits `input`, a tensor with one or more dimensions, into multiple tensors horizontally according to `indices_or_sections`. |
| `hstack` | Stack tensors in sequence horizontally (column wise). |
| `index_add` | See `index_add_()` for function description. |
| `index_copy` | See `index_add_()` for function description. |
| `index_reduce` | See `index_reduce_()` for function description. |

| | |
|---|---|
| `index_select` | Returns a new tensor which indexes the `input` tensor along dimension `dim` using the entries in `index` which is a *LongTensor*. |
| `masked_select` | Returns a new 1-D tensor which indexes the `input` tensor according to the boolean mask `mask` which is a *BoolTensor*. |
| `movedim` | Moves the dimension(s) of `input` at the position(s) in `source` to the position(s) in `destination`. |
| `moveaxis` | Alias for `torch.movedim()`. |
| `narrow` | Returns a new tensor that is a narrowed version of `input` tensor. |
| `nonzero` | |
| `permute` | Returns a view of the original tensor `input` with its dimensions permuted. |
| `reshape` | Returns a tensor with the same data and number of elements as `input`, but with the specified shape. |
| `row_stack` | Alias of `torch.vstack()`. |
| `select` | Slices the `input` tensor along the selected dimension at the given index. |
| `scatter` | Out-of-place version of `torch.Tensor.scatter_()` |
| `diagonal_scatter` | Embeds the values of the `src` tensor into `input` along the diagonal elements of `input`, with respect to `dim1` and `dim2`. |
| `select_scatter` | Embeds the values of the `src` tensor into `input` at the given index. |
| `slice_scatter` | Embeds the values of the `src` tensor into `input` at the given dimension. |
| `scatter_add` | Out-of-place version of `torch.Tensor.scatter_add_()` |
| `scatter_reduce` | Out-of-place version of `torch.Tensor.scatter_reduce_()` |
| `split` | Splits the tensor into chunks. |
| `squeeze` | Returns a tensor with all the dimensions of `input` of size *1* removed. |
| `stack` | Concatenates a sequence of tensors along a new dimension. |
| `swapaxes` | Alias for `torch.transpose()`. |
| `swapdims` | Alias for `torch.transpose()`. |

| | |
|---|---|
| `t` | Expects `input` to be <= 2-D tensor and transposes dimensions 0 and 1. |
| `take` | Returns a new tensor with the elements of `input` at the given indices. |
| `take_along_dim` | Selects values from `input` at the 1-dimensional indices from `indices` along the given `dim`. |
| `tensor_split` | Splits a tensor into multiple sub-tensors, all of which are views of `input`, along dimension `dim` according to the indices or number of sections specified by `indices_or_sections`. |
| `tile` | Constructs a tensor by repeating the elements of `input`. |
| `transpose` | Returns a tensor that is a transposed version of `input`. |
| `unbind` | Removes a tensor dimension. |
| `unsqueeze` | Returns a new tensor with a dimension of size one inserted at the specified position. |
| `vsplit` | Splits `input`, a tensor with two or more dimensions, into multiple tensors vertically according to `indices_or_sections`. |
| `vstack` | Stack tensors in sequence vertically (row wise). |
| `where` | Return a tensor of elements selected from either `x` or `y`, depending on `condition`. |

## Generators

| | |
|---|---|
| `Generator` | Creates and returns a generator object that manages the state of the algorithm which produces pseudo random numbers. |

## Random sampling

| | |
|---|---|
| `seed` | Sets the seed for generating random numbers to a non-deterministic random number. |
| `manual_seed` | Sets the seed for generating random numbers. |
| `initial_seed` | Returns the initial seed for generating random numbers as a Python *long*. |
| `get_rng_state` | Returns the random number generator state as a *torch.ByteTensor*. |
| `set_rng_state` | Sets the random number generator state. |

torch.default_generator *Returns the default CPU torch.Generator*

| | |
|---|---|
| `bernoulli` | Draws binary random numbers (0 or 1) from a Bernoulli distribution. |

| | |
|---|---|
| `multinomial` | Returns a tensor where each row contains `num_samples` indices sampled from the multinomial probability distribution located in the corresponding row of tensor `input`. |
| `normal` | Returns a tensor of random numbers drawn from separate normal distributions whose mean and standard deviation are given. |
| `poisson` | Returns a tensor of the same size as `input` with each element sampled from a Poisson distribution with rate parameter given by the corresponding element in `input` i.e., |
| `rand` | Returns a tensor filled with random numbers from a uniform distribution on the interval $[0, 1)$ |
| `rand_like` | Returns a tensor with the same size as `input` that is filled with random numbers from a uniform distribution on the interval $[0, 1)$. |
| `randint` | Returns a tensor filled with random integers generated uniformly between `low` (inclusive) and `high` (exclusive). |
| `randint_like` | Returns a tensor with the same shape as Tensor `input` filled with random integers generated uniformly between `low` (inclusive) and `high` (exclusive). |
| `randn` | Returns a tensor filled with random numbers from a normal distribution with mean *0* and variance *1* (also called the standard normal distribution). |
| `randn_like` | Returns a tensor with the same size as `input` that is filled with random numbers from a normal distribution with mean 0 and variance 1. |
| `randperm` | Returns a random permutation of integers from `0` to `n - 1`. |

## In-place random sampling

There are a few more in-place random sampling functions defined on Tensors as well. Click through to refer to their documentation:

- `torch.Tensor.bernoulli_()` - in-place version of `torch.bernoulli()`
- `torch.Tensor.cauchy_()` - numbers drawn from the Cauchy distribution
- `torch.Tensor.exponential_()` - numbers drawn from the exponential distribution
- `torch.Tensor.geometric_()` - elements drawn from the geometric distribution
- `torch.Tensor.log_normal_()` - samples from the log-normal distribution
- `torch.Tensor.normal_()` - in-place version of `torch.normal()`
- `torch.Tensor.random_()` - numbers sampled from the discrete uniform distribution
- `torch.Tensor.uniform_()` - numbers sampled from the continuous uniform distribution

## Quasi-random sampling

| | |
|---|---|
| `quasirandom.SobolEngine` | The `torch.quasirandom.SobolEngine` is an engine for generating (scrambled) Sobol sequences. |

## Serialization

| | |
|---|---|
| `save` | Saves an object to a disk file. |
| `load` | Loads an object saved with `torch.save()` from a file. |

## Parallelism

| | |
|---|---|
| `get_num_threads` | Returns the number of threads used for parallelizing CPU operations |

| | |
|---|---|
| `set_num_threads` | Sets the number of threads used for intraop parallelism on CPU. |
| `get_num_interop_threads` | Returns the number of threads used for inter-op parallelism on CPU (e.g. |
| `set_num_interop_threads` | Sets the number of threads used for interop parallelism (e.g. |

## Locally disabling gradient computation

The context managers `torch.no_grad()`, `torch.enable_grad()`, and `torch.set_grad_enabled()` are helpful for locally disabling and enabling gradient computation. See Locally disabling gradient computation for more details on their usage. These context managers are thread local, so they won't work if you send work to another thread using the `threading` module, etc.

Examples:

```
>>> x = torch.zeros(1, requires_grad=True)
>>> with torch.no_grad():
...     y = x * 2
>>> y.requires_grad
False

>>> is_train = False
>>> with torch.set_grad_enabled(is_train):
...     y = x * 2
>>> y.requires_grad
False

>>> torch.set_grad_enabled(True)  # this can also be used as a function
>>> y = x * 2
>>> y.requires_grad
True

>>> torch.set_grad_enabled(False)
>>> y = x * 2
>>> y.requires_grad
False
```

| | |
|---|---|
| `no_grad` | Context-manager that disabled gradient calculation. |
| `enable_grad` | Context-manager that enables gradient calculation. |
| `set_grad_enabled` | Context-manager that sets gradient calculation to on or off. |
| `is_grad_enabled` | Returns True if grad mode is currently enabled. |
| `inference_mode` | Context-manager that enables or disables inference mode |
| `is_inference_mode_enabled` | Returns True if inference mode is currently enabled. |

## Math operations

### Pointwise Ops

| | |
|---|---|
| `abs` | Computes the absolute value of each element in `input`. |
| `absolute` | Alias for `torch.abs()` |
| `acos` | Computes the inverse cosine of each element in `input`. |
| `arccos` | Alias for `torch.acos()`. |

| | |
|---|---|
| `acosh` | Returns a new tensor with the inverse hyperbolic cosine of the elements of `input`. |
| `arccosh` | Alias for `torch.acosh()`. |
| `add` | Adds `other`, scaled by `alpha`, to `input`. |
| `addcdiv` | Performs the element-wise division of `tensor1` by `tensor2`, multiply the result by the scalar `value` and add it to `input`. |
| `addcmul` | Performs the element-wise multiplication of `tensor1` by `tensor2`, multiply the result by the scalar `value` and add it to `input`. |
| `angle` | Computes the element-wise angle (in radians) of the given `input` tensor. |
| `asin` | Returns a new tensor with the arcsine of the elements of `input`. |
| `arcsin` | Alias for `torch.asin()`. |
| `asinh` | Returns a new tensor with the inverse hyperbolic sine of the elements of `input`. |
| `arcsinh` | Alias for `torch.asinh()`. |
| `atan` | Returns a new tensor with the arctangent of the elements of `input`. |
| `arctan` | Alias for `torch.atan()`. |
| `atanh` | Returns a new tensor with the inverse hyperbolic tangent of the elements of `input`. |
| `arctanh` | Alias for `torch.atanh()`. |
| `atan2` | Element-wise arctangent of $\text{input}_i / \text{other}_i$ with consideration of the quadrant. |
| `arctan2` | Alias for `torch.atan2()`. |
| `bitwise_not` | Computes the bitwise NOT of the given input tensor. |
| `bitwise_and` | Computes the bitwise AND of `input` and `other`. |
| `bitwise_or` | Computes the bitwise OR of `input` and `other`. |
| `bitwise_xor` | Computes the bitwise XOR of `input` and `other`. |
| `bitwise_left_shift` | Computes the left arithmetic shift of `input` by `other` bits. |
| `bitwise_right_shift` | Computes the right arithmetic shift of `input` by `other` bits. |

| | |
|---|---|
| `ceil` | Returns a new tensor with the ceil of the elements of `input`, the smallest integer greater than or equal to each element. |
| `clamp` | Clamps all elements in `input` into the range [ `min`, `max` ]. |
| `clip` | Alias for `torch.clamp()`. |
| `conj_physical` | Computes the element-wise conjugate of the given `input` tensor. |
| `copysign` | Create a new floating-point tensor with the magnitude of `input` and the sign of `other`, elementwise. |
| `cos` | Returns a new tensor with the cosine of the elements of `input`. |
| `cosh` | Returns a new tensor with the hyperbolic cosine of the elements of `input`. |
| `deg2rad` | Returns a new tensor with each of the elements of `input` converted from angles in degrees to radians. |
| `div` | Divides each element of the input `input` by the corresponding element of `other`. |
| `divide` | Alias for `torch.div()`. |
| `digamma` | Alias for `torch.special.digamma()`. |
| `erf` | Alias for `torch.special.erf()`. |
| `erfc` | Alias for `torch.special.erfc()`. |
| `erfinv` | Alias for `torch.special.erfinv()`. |
| `exp` | Returns a new tensor with the exponential of the elements of the input tensor `input`. |
| `exp2` | Alias for `torch.special.exp2()`. |
| `expm1` | Alias for `torch.special.expm1()`. |
| `fake_quantize_per_channel_affine` | Returns a new tensor with the data in `input` fake quantized per channel using `scale`, `zero_point`, `quant_min` and `quant_max`, across the channel specified by `axis`. |
| `fake_quantize_per_tensor_affine` | Returns a new tensor with the data in `input` fake quantized using `scale`, `zero_point`, `quant_min` and `quant_max`. |
| `fix` | Alias for `torch.trunc()` |
| `float_power` | Raises `input` to the power of `exponent`, elementwise, in double precision. |

| | |
|---|---|
| `floor` | Returns a new tensor with the floor of the elements of `input`, the largest integer less than or equal to each element. |
| `floor_divide` | |
| `fmod` | Applies C++'s std::fmod entrywise. |
| `frac` | Computes the fractional portion of each element in `input`. |
| `frexp` | Decomposes `input` into mantissa and exponent tensors such that $\text{input} = \text{mantissa} \times 2^{\text{exponent}}$. |
| `gradient` | Estimates the gradient of a function $g : \mathbb{R}^n \to \mathbb{R}$ in one or more dimensions using the second-order accurate central differences method. |
| `imag` | Returns a new tensor containing imaginary values of the `self` tensor. |
| `ldexp` | Multiplies `input` by 2**:attr:*other*. |
| `lerp` | Does a linear interpolation of two tensors `start` (given by `input`) and `end` based on a scalar or tensor `weight` and returns the resulting `out` tensor. |
| `lgamma` | Computes the natural logarithm of the absolute value of the gamma function on `input`. |
| `log` | Returns a new tensor with the natural logarithm of the elements of `input`. |
| `log10` | Returns a new tensor with the logarithm to the base 10 of the elements of `input`. |
| `log1p` | Returns a new tensor with the natural logarithm of (1 + `input`). |
| `log2` | Returns a new tensor with the logarithm to the base 2 of the elements of `input`. |
| `logaddexp` | Logarithm of the sum of exponentiations of the inputs. |
| `logaddexp2` | Logarithm of the sum of exponentiations of the inputs in base-2. |
| `logical_and` | Computes the element-wise logical AND of the given input tensors. |
| `logical_not` | Computes the element-wise logical NOT of the given input tensor. |
| `logical_or` | Computes the element-wise logical OR of the given input tensors. |
| `logical_xor` | Computes the element-wise logical XOR of the given input tensors. |
| `logit` | Alias for `torch.special.logit()`. |

| | |
|---|---|
| `hypot` | Given the legs of a right triangle, return its hypotenuse. |
| `i0` | Alias for `torch.special.i0()`. |
| `igamma` | Alias for `torch.special.gammainc()`. |
| `igammac` | Alias for `torch.special.gammaincc()`. |
| `mul` | Multiplies `input` by `other`. |
| `multiply` | Alias for `torch.mul()`. |
| `mvlgamma` | Alias for `torch.special.multigammaln()`. |
| `nan_to_num` | Replaces `NaN`, positive infinity, and negative infinity values in `input` with the values specified by `nan`, `posinf`, and `neginf`, respectively. |
| `neg` | Returns a new tensor with the negative of the elements of `input`. |
| `negative` | Alias for `torch.neg()` |
| `nextafter` | Return the next floating-point value after `input` towards `other`, elementwise. |
| `polygamma` | Alias for `torch.special.polygamma()`. |
| `positive` | Returns `input`. |
| `pow` | Takes the power of each element in `input` with `exponent` and returns a tensor with the result. |
| `quantized_batch_norm` | Applies batch normalization on a 4D (NCHW) quantized tensor. |
| `quantized_max_pool1d` | Applies a 1D max pooling over an input quantized tensor composed of several input planes. |
| `quantized_max_pool2d` | Applies a 2D max pooling over an input quantized tensor composed of several input planes. |
| `rad2deg` | Returns a new tensor with each of the elements of `input` converted from angles in radians to degrees. |
| `real` | Returns a new tensor containing real values of the `self` tensor. |
| `reciprocal` | Returns a new tensor with the reciprocal of the elements of `input` |
| `remainder` | Computes Python's modulus operation entrywise. |
| `round` | Rounds elements of `input` to the nearest integer. |

| | |
|---|---|
| `rsqrt` | Returns a new tensor with the reciprocal of the square-root of each of the elements of `input`. |
| `sigmoid` | Alias for `torch.special.expit()`. |
| `sign` | Returns a new tensor with the signs of the elements of `input`. |
| `sgn` | This function is an extension of torch.sign() to complex tensors. |
| `signbit` | Tests if each element of `input` has its sign bit set or not. |
| `sin` | Returns a new tensor with the sine of the elements of `input`. |
| `sinc` | Alias for `torch.special.sinc()`. |
| `sinh` | Returns a new tensor with the hyperbolic sine of the elements of `input`. |
| `sqrt` | Returns a new tensor with the square-root of the elements of `input`. |
| `square` | Returns a new tensor with the square of the elements of `input`. |
| `sub` | Subtracts `other`, scaled by `alpha`, from `input`. |
| `subtract` | Alias for `torch.sub()`. |
| `tan` | Returns a new tensor with the tangent of the elements of `input`. |
| `tanh` | Returns a new tensor with the hyperbolic tangent of the elements of `input`. |
| `true_divide` | Alias for `torch.div()` with `rounding_mode=None`. |
| `trunc` | Returns a new tensor with the truncated integer values of the elements of `input`. |
| `xlogy` | Alias for `torch.special.xlogy()`. |

## Reduction Ops

| | |
|---|---|
| `argmax` | Returns the indices of the maximum value of all elements in the `input` tensor. |
| `argmin` | Returns the indices of the minimum value(s) of the flattened tensor or along a dimension |
| `amax` | Returns the maximum value of each slice of the `input` tensor in the given dimension(s) `dim`. |
| `amin` | Returns the minimum value of each slice of the `input` tensor in the given dimension(s) `dim`. |

| | |
|---|---|
| `aminmax` | Computes the minimum and maximum values of the `input` tensor. |
| `all` | Tests if all elements in `input` evaluate to *True*. |
| `any` | Tests if any element in `input` evaluates to *True*. |
| `max` | Returns the maximum value of all elements in the `input` tensor. |
| `min` | Returns the minimum value of all elements in the `input` tensor. |
| `dist` | Returns the p-norm of (`input` - `other`) |
| `logsumexp` | Returns the log of summed exponentials of each row of the `input` tensor in the given dimension `dim`. |
| `mean` | Returns the mean value of all elements in the `input` tensor. |
| `nanmean` | Computes the mean of all *non-NaN* elements along the specified dimensions. |
| `median` | Returns the median of the values in `input`. |
| `nanmedian` | Returns the median of the values in `input`, ignoring `NaN` values. |
| `mode` | Returns a namedtuple `(values, indices)` where `values` is the mode value of each row of the `input` tensor in the given dimension `dim`, i.e. a value which appears most often in that row, and `indices` is the index location of each mode value found. |
| `norm` | Returns the matrix norm or vector norm of a given tensor. |
| `nansum` | Returns the sum of all elements, treating Not a Numbers (NaNs) as zero. |
| `prod` | Returns the product of all elements in the `input` tensor. |
| `quantile` | Computes the q-th quantiles of each row of the `input` tensor along the dimension `dim`. |
| `nanquantile` | This is a variant of `torch.quantile()` that "ignores" `NaN` values, computing the quantiles `q` as if `NaN` values in `input` did not exist. |
| `std` | If `unbiased` is `True`, Bessel's correction will be used. |
| `std_mean` | If `unbiased` is `True`, Bessel's correction will be used to calculate the standard deviation. |
| `sum` | Returns the sum of all elements in the `input` tensor. |
| `unique` | Returns the unique elements of the input tensor. |

| | |
|---|---|
| `unique_consecutive` | Eliminates all but the first element from every consecutive group of equivalent elements. |
| `var` | If `unbiased` is `True`, Bessel's correction will be used. |
| `var_mean` | If `unbiased` is `True`, Bessel's correction will be used to calculate the variance. |
| `count_nonzero` | Counts the number of non-zero values in the tensor `input` along the given `dim`. |

## Comparison Ops

| | |
|---|---|
| `allclose` | This function checks if all `input` and `other` satisfy the condition: |
| `argsort` | Returns the indices that sort a tensor along a given dimension in ascending order by value. |
| `eq` | Computes element-wise equality |
| `equal` | `True` if two tensors have the same size and elements, `False` otherwise. |
| `ge` | Computes $\text{input} \geq \text{other}$ element-wise. |
| `greater_equal` | Alias for `torch.ge()`. |
| `gt` | Computes $\text{input} > \text{other}$ element-wise. |
| `greater` | Alias for `torch.gt()`. |
| `isclose` | Returns a new tensor with boolean elements representing if each element of `input` is "close" to the corresponding element of `other`. |
| `isfinite` | Returns a new tensor with boolean elements representing if each element is *finite* or not. |
| `isin` | Tests if each element of `elements` is in `test_elements`. |
| `isinf` | Tests if each element of `input` is infinite (positive or negative infinity) or not. |
| `isposinf` | Tests if each element of `input` is positive infinity or not. |
| `isneginf` | Tests if each element of `input` is negative infinity or not. |
| `isnan` | Returns a new tensor with boolean elements representing if each element of `input` is NaN or not. |
| `isreal` | Returns a new tensor with boolean elements representing if each element of `input` is real-valued or not. |
| `kthvalue` | Returns a namedtuple `(values, indices)` where `values` is the `k` th smallest element of each row of the `input` tensor in the given dimension `dim`. |

| | |
|---|---|
| `le` | Computes $\text{input} \leq \text{other}$ element-wise. |
| `less_equal` | Alias for `torch.le()`. |
| `lt` | Computes $\text{input} < \text{other}$ element-wise. |
| `less` | Alias for `torch.lt()`. |
| `maximum` | Computes the element-wise maximum of `input` and `other`. |
| `minimum` | Computes the element-wise minimum of `input` and `other`. |
| `fmax` | Computes the element-wise maximum of `input` and `other`. |
| `fmin` | Computes the element-wise minimum of `input` and `other`. |
| `ne` | Computes $\text{input} \neq \text{other}$ element-wise. |
| `not_equal` | Alias for `torch.ne()`. |
| `sort` | Sorts the elements of the `input` tensor along a given dimension in ascending order by value. |
| `topk` | Returns the `k` largest elements of the given `input` tensor along a given dimension. |
| `msort` | Sorts the elements of the `input` tensor along its first dimension in ascending order by value. |

## Spectral Ops

| | |
|---|---|
| `stft` | Short-time Fourier transform (STFT). |
| `istft` | Inverse short time Fourier Transform. |
| `bartlett_window` | Bartlett window function. |
| `blackman_window` | Blackman window function. |
| `hamming_window` | Hamming window function. |
| `hann_window` | Hann window function. |
| `kaiser_window` | Computes the Kaiser window with window length `window_length` and shape parameter `beta`. |

## Other Operations

| | |
|---|---|
| `atleast_1d` | Returns a 1-dimensional view of each input tensor with zero dimensions. |
| `atleast_2d` | Returns a 2-dimensional view of each input tensor with zero dimensions. |
| `atleast_3d` | Returns a 3-dimensional view of each input tensor with zero dimensions. |
| `bincount` | Count the frequency of each value in an array of non-negative ints. |
| `block_diag` | Create a block diagonal matrix from provided tensors. |
| `broadcast_tensors` | Broadcasts the given tensors according to Broadcasting semantics. |
| `broadcast_to` | Broadcasts `input` to the shape `shape`. |
| `broadcast_shapes` | Similar to `broadcast_tensors()` but for shapes. |
| `bucketize` | Returns the indices of the buckets to which each value in the `input` belongs, where the boundaries of the buckets are set by `boundaries`. |
| `cartesian_prod` | Do cartesian product of the given sequence of tensors. |
| `cdist` | Computes batched the p-norm distance between each pair of the two collections of row vectors. |
| `clone` | Returns a copy of `input`. |
| `combinations` | Compute combinations of length $r$ of the given tensor. |
| `corrcoef` | Estimates the Pearson product-moment correlation coefficient matrix of the variables given by the `input` matrix, where rows are the variables and columns are the observations. |
| `cov` | Estimates the covariance matrix of the variables given by the `input` matrix, where rows are the variables and columns are the observations. |
| `cross` | Returns the cross product of vectors in dimension `dim` of `input` and `other`. |
| `cummax` | Returns a namedtuple `(values, indices)` where `values` is the cumulative maximum of elements of `input` in the dimension `dim`. |
| `cummin` | Returns a namedtuple `(values, indices)` where `values` is the cumulative minimum of elements of `input` in the dimension `dim`. |
| `cumprod` | Returns the cumulative product of elements of `input` in the dimension `dim`. |
| `cumsum` | Returns the cumulative sum of elements of `input` in the dimension `dim`. |

| | |
|---|---|
| `diag` | • If `input` is a vector (1-D tensor), then returns a 2-D square tensor |
| `diag_embed` | Creates a tensor whose diagonals of certain 2D planes (specified by `dim1` and `dim2`) are filled by `input`. |
| `diagflat` | • If `input` is a vector (1-D tensor), then returns a 2-D square tensor |
| `diagonal` | Returns a partial view of `input` with the its diagonal elements with respect to `dim1` and `dim2` appended as a dimension at the end of the shape. |
| `diff` | Computes the n-th forward difference along the given dimension. |
| `einsum` | Sums the product of the elements of the input `operands` along dimensions specified using a notation based on the Einstein summation convention. |
| `flatten` | Flattens `input` by reshaping it into a one-dimensional tensor. |
| `flip` | Reverse the order of a n-D tensor along given axis in dims. |
| `fliplr` | Flip tensor in the left/right direction, returning a new tensor. |
| `flipud` | Flip tensor in the up/down direction, returning a new tensor. |
| `kron` | Computes the Kronecker product, denoted by $\otimes$, of `input` and `other`. |
| `rot90` | Rotate a n-D tensor by 90 degrees in the plane specified by dims axis. |
| `gcd` | Computes the element-wise greatest common divisor (GCD) of `input` and `other`. |
| `histc` | Computes the histogram of a tensor. |
| `histogram` | Computes a histogram of the values in a tensor. |
| `histogramdd` | Computes a multi-dimensional histogram of the values in a tensor. |
| `meshgrid` | Creates grids of coordinates specified by the 1D inputs in *attr*:tensors. |
| `lcm` | Computes the element-wise least common multiple (LCM) of `input` and `other`. |
| `logcumsumexp` | Returns the logarithm of the cumulative summation of the exponentiation of elements of `input` in the dimension `dim`. |
| `ravel` | Return a contiguous flattened tensor. |
| `renorm` | Returns a tensor where each sub-tensor of `input` along dimension `dim` is normalized such that the *p*-norm of the sub-tensor is lower than the value `maxnorm` |

| | |
|---|---|
| `repeat_interleave` | Repeat elements of a tensor. |
| `roll` | Roll the tensor `input` along the given dimension(s). |
| `searchsorted` | Find the indices from the *innermost* dimension of `sorted_sequence` such that, if the corresponding values in `values` were inserted before the indices, when sorted, the order of the corresponding *innermost* dimension within `sorted_sequence` would be preserved. |
| `tensordot` | Returns a contraction of a and b over multiple dimensions. |
| `trace` | Returns the sum of the elements of the diagonal of the input 2-D matrix. |
| `tril` | Returns the lower triangular part of the matrix (2-D tensor) or batch of matrices `input`, the other elements of the result tensor `out` are set to 0. |
| `tril_indices` | Returns the indices of the lower triangular part of a `row`-by- `col` matrix in a 2-by-N Tensor, where the first row contains row coordinates of all indices and the second row contains column coordinates. |
| `triu` | Returns the upper triangular part of a matrix (2-D tensor) or batch of matrices `input`, the other elements of the result tensor `out` are set to 0. |
| `triu_indices` | Returns the indices of the upper triangular part of a `row` by `col` matrix in a 2-by-N Tensor, where the first row contains row coordinates of all indices and the second row contains column coordinates. |
| `unflatten` | Expands a dimension of the input tensor over multiple dimensions. |
| `vander` | Generates a Vandermonde matrix. |
| `view_as_real` | Returns a view of `input` as a real tensor. |
| `view_as_complex` | Returns a view of `input` as a complex tensor. |
| `resolve_conj` | Returns a new tensor with materialized conjugation if `input`'s conjugate bit is set to *True*, else returns `input`. |
| `resolve_neg` | Returns a new tensor with materialized negation if `input`'s negative bit is set to *True*, else returns `input`. |

## BLAS and LAPACK Operations

| | |
|---|---|
| `addbmm` | Performs a batch matrix-matrix product of matrices stored in `batch1` and `batch2`, with a reduced add step (all matrix multiplications get accumulated along the first dimension). |
| `addmm` | Performs a matrix multiplication of the matrices `mat1` and `mat2`. |
| `addmv` | Performs a matrix-vector product of the matrix `mat` and the vector `vec`. |
| `addr` | Performs the outer-product of vectors `vec1` and `vec2` and adds it to the matrix `input`. |
| `baddbmm` | Performs a batch matrix-matrix product of matrices in `batch1` and `batch2`. |

| | |
|---|---|
| `bmm` | Performs a batch matrix-matrix product of matrices stored in `input` and `mat2`. |
| `chain_matmul` | Returns the matrix product of the $N$ 2-D tensors. |
| `cholesky` | Computes the Cholesky decomposition of a symmetric positive-definite matrix $A$ or for batches of symmetric positive-definite matrices. |
| `cholesky_inverse` | Computes the inverse of a symmetric positive-definite matrix $A$ using its Cholesky factor $u$: returns matrix `inv`. |
| `cholesky_solve` | Solves a linear system of equations with a positive semidefinite matrix to be inverted given its Cholesky factor matrix $u$. |
| `dot` | Computes the dot product of two 1D tensors. |
| `geqrf` | This is a low-level function for calling LAPACK's geqrf directly. |
| `ger` | Alias of `torch.outer()`. |
| `inner` | Computes the dot product for 1D tensors. |
| `inverse` | Alias for `torch.linalg.inv()` |
| `det` | Alias for `torch.linalg.det()` |
| `logdet` | Calculates log determinant of a square matrix or batches of square matrices. |
| `slogdet` | Alias for `torch.linalg.slogdet()` |
| `lu` | Computes the LU factorization of a matrix or batches of matrices `A`. |
| `lu_solve` | Returns the LU solve of the linear system $Ax = b$ using the partially pivoted LU factorization of A from `lu_factor()`. |
| `lu_unpack` | Unpacks the LU decomposition returned by `lu_factor()` into the *P, L, U* matrices. |
| `matmul` | Matrix product of two tensors. |
| `matrix_power` | Alias for `torch.linalg.matrix_power()` |
| `matrix_exp` | Alias for `torch.linalg.matrix_exp()`. |
| `mm` | Performs a matrix multiplication of the matrices `input` and `mat2`. |
| `mv` | Performs a matrix-vector product of the matrix `input` and the vector `vec`. |

| | |
|---|---|
| `orgqr` | Alias for `torch.linalg.householder_product()`. |
| `ormqr` | Computes the matrix-matrix multiplication of a product of Householder matrices with a general matrix. |
| `outer` | Outer product of `input` and `vec2`. |
| `pinverse` | Alias for `torch.linalg.pinv()` |
| `qr` | Computes the QR decomposition of a matrix or a batch of matrices `input`, and returns a namedtuple (Q, R) of tensors such that $\text{input} = QR$ with $Q$ being an orthogonal matrix or batch of orthogonal matrices and $R$ being an upper triangular matrix or batch of upper triangular matrices. |
| `svd` | Computes the singular value decomposition of either a matrix or batch of matrices `input`. |
| `svd_lowrank` | Return the singular value decomposition `(U, S, V)` of a matrix, batches of matrices, or a sparse matrix $A$ such that $A \approx Udiag(S)V^T$. |
| `pca_lowrank` | Performs linear Principal Component Analysis (PCA) on a low-rank matrix, batches of such matrices, or sparse matrix. |
| `symeig` | This function returns eigenvalues and eigenvectors of a real symmetric or complex Hermitian matrix `input` or a batch thereof, represented by a namedtuple (eigenvalues, eigenvectors). |
| `lobpcg` | Find the k largest (or smallest) eigenvalues and the corresponding eigenvectors of a symmetric positive definite generalized eigenvalue problem using matrix-free LOBPCG methods. |
| `trapz` | Alias for `torch.trapezoid()`. |
| `trapezoid` | Computes the trapezoidal rule along `dim`. |
| `cumulative_trapezoid` | Cumulatively computes the trapezoidal rule along `dim`. |
| `triangular_solve` | Solves a system of equations with a square upper or lower triangular invertible matrix $A$ and multiple right-hand sides $b$. |
| `vdot` | Computes the dot product of two 1D vectors along a dimension. |

## Utilities

| | |
|---|---|
| `compiled_with_cxx11_abi` | Returns whether PyTorch was built with _GLIBCXX_USE_CXX11_ABI=1 |
| `result_type` | Returns the `torch.dtype` that would result from performing an arithmetic operation on the provided input tensors. |
| `can_cast` | Determines if a type conversion is allowed under PyTorch casting rules described in the type promotion documentation. |
| `promote_types` | Returns the `torch.dtype` with the smallest size and scalar kind that is not smaller nor of lower kind than either *type1* or *type2*. |

| | |
|---|---|
| `use_deterministic_algorithms` | Sets whether PyTorch operations must use "deterministic" algorithms. |
| `are_deterministic_algorithms_enabled` | Returns True if the global deterministic flag is turned on. |
| `is_deterministic_algorithms_warn_only_enabled` | Returns True if the global deterministic flag is set to warn only. |
| `set_deterministic_debug_mode` | Sets the debug mode for deterministic operations. |
| `get_deterministic_debug_mode` | Returns the current value of the debug mode for deterministic operations. |
| `set_float32_matmul_precision` | Sets the internal precision of float32 matrix multiplications. |
| `get_float32_matmul_precision` | Returns the current value of float32 matrix multiplication precision. |
| `set_warn_always` | When this flag is False (default) then some PyTorch warnings may only appear once per process. |
| `is_warn_always_enabled` | Returns True if the global warn_always flag is turned on. |
| `_assert` | A wrapper around Python's assert which is symbolically traceable. |

## Operator Tags

CLASS torch.Tag

Members:

inplace_view

nondeterministic_seeded

dynamic_output_shape

data_dependent_output

view_copy

generated

nondeterministic_bitwise

PROPERTY name

‹ Previous Next ›