

torch

包 `torch` 包含了多维张量的数据结构以及基于其上的多种数学操作。另外，它也提供了多种工具，其中一些可以更有效地对张量和任意类型进行序列化。

它有CUDA的对应实现，可以在NVIDIA GPU上进行张量运算(计算能力 ≥ 2.0)。

张量 Tensors

`torch.is_tensor`[\[source\]](#)

```
torch.is_tensor(obj)
```

如果`obj`是一个pytorch张量，则返回True

- 参数: `obj` (Object) – 判断对象

`torch.is_storage` [\[source\]](#)

```
torch.is_storage(obj)
```

如何`obj`是一个pytorch storage对象，则返回True

- 参数: `input` (Object) – 判断对象

`torch.set_default_tensor_type`[\[source\]](#)

```
torch.set_default_tensor_type(t)
```

`torch.numel`

```
torch.numel(input)->int
```

返回 `input` 张量中的元素个数

- 参数: input (*Tensor*) – 输入张量

例子:

```
>>> a = torch.randn(1,2,3,4,5)
>>> torch.numel(a)
120
>>> a = torch.zeros(4,4)
>>> torch.numel(a)
16
```

torch.set_printoptions[source]

```
torch.set_printoptions(precision=None, threshold=None, edgeitems=None, linewidth=None, profile=None)
```

设置打印选项。 完全参考自 [Numpy](#)。

参数:

- precision – 浮点数输出的精度位数 (默认为8)
- threshold – 阈值, 触发汇总显示而不是完全显示(repr)的数组元素的总数 (默认为1000)
- edgeitems – 汇总显示中, 每维 (轴) 两端显示的项数 (默认值为3)
- linewidth – 用于插入行间隔的每行字符数 (默认为80) 。 Thresholded matrices will ignore this parameter.
- profile – pretty打印的完全默认值。 可以覆盖上述所有选项 (默认为short, full)

创建操作 Creation Ops

torch.eye

```
torch.eye(n, m=None, out=None)
```

返回一个2维张量, 对角线位置全1, 其它位置全0

参数:

- n (*int*) – 行数
- m (*int, optional*) – 列数.如果为None,则默认为n
- out (*Tensor, optional*) - Output tensor

返回值: 对角线位置全1, 其它位置全0的2维张量

返回值类型: *Tensor*

例子:

```
>>> torch.eye(3)
 1  0  0
 0  1  0
 0  0  1
[torch.FloatTensor of size 3x3]
```

from_numpy

```
torch.from_numpy(ndarray) → Tensor
```

Numpy桥，将 `numpy.ndarray` 转换为pytorch的 `Tensor`。返回的张量tensor和numpy的ndarray共享同一内存空间。修改一个会导致另外一个也被修改。返回的张量不能改变大小。

例子:

```
>>> a = numpy.array([1, 2, 3])
>>> t = torch.from_numpy(a)
>>> t
torch.LongTensor([1, 2, 3])
>>> t[0] = -1
>>> a
array([-1,  2,  3])
```

torch.linspace

```
torch.linspace(start, end, steps=100, out=None) → Tensor
```

返回一个1维张量，包含在区间 `start` 和 `end` 上均匀间隔的 `steps` 个点。输出1维张量的长度为 `steps`。

参数:

- start (float) – 序列的起始点
- end (float) – 序列的最终值
- steps (int) – 在 `start` 和 `end` 间生成的样本数
- out (Tensor, optional) – 结果张量

例子:

```
>>> torch.linspace(3, 10, steps=5)

 3.0000
 4.7500
 6.5000
 8.2500
10.0000
[torch.FloatTensor of size 5]

>>> torch.linspace(-10, 10, steps=5)

-10
 -5
  0
  5
 10
[torch.FloatTensor of size 5]

>>> torch.linspace(start=-10, end=10, steps=5)

-10
 -5
  0
  5
 10
[torch.FloatTensor of size 5]
```

torch.logspace

```
torch.logspace(start, end, steps=100, out=None) → Tensor
```

返回一个1维张量，包含在区间 10^{start} 和 10^{end} 上以对数刻度均匀间隔的 `steps` 个点。输出1维张量的长度为 `steps`。

参数:

- start (float) – 序列的起始点
- end (float) – 序列的最终值
- steps (int) – 在 `start` 和 `end` 间生成的样本数
- out (Tensor, optional) – 结果张量

例子:

```
>>> torch.logspace(start=-10, end=10, steps=5)

1.0000e-10
1.0000e-05
1.0000e+00
1.0000e+05
1.0000e+10
[torch.FloatTensor of size 5]

>>> torch.logspace(start=0.1, end=1.0, steps=5)

1.2589
2.1135
3.5481
5.9566
10.0000
[torch.FloatTensor of size 5]
```

torch.ones

```
torch.ones(*sizes, out=None) → Tensor
```

返回一个全为1的张量，形状由可变参数 `sizes` 定义。

参数:

- `sizes (int...)` – 整数序列，定义了输出形状
- `out (Tensor, optional)` – 结果张量 例子:

```
>>> torch.ones(2, 3)

1  1  1
1  1  1
[torch.FloatTensor of size 2x3]

>>> torch.ones(5)

1
1
1
1
1
[torch.FloatTensor of size 5]
```

torch.rand

```
torch.rand(*sizes, out=None) → Tensor
```

返回一个张量，包含了从区间[0,1)的均匀分布中抽取的一组随机数，形状由可变参数 `sizes` 定义。

参数:

- `sizes (int...)` – 整数序列，定义了输出形状

- out (*Tensor*, *optinal*) - 结果张量 例子:

```
>>> torch.rand(4)

0.9193
0.3347
0.3232
0.7715
[torch.FloatTensor of size 4]

>>> torch.rand(2, 3)

0.5010  0.5140  0.0719
0.1435  0.5636  0.0538
[torch.FloatTensor of size 2x3]
```

torch.randn

```
torch.randn(*sizes, out=None) → Tensor
```

返回一个张量，包含了从标准正态分布(均值为0，方差为 1，即高斯白噪声)中抽取一组随机数，形状由可变参数 `sizes` 定义。参数:

- sizes (int...) - 整数序列，定义了输出形状
- out (*Tensor*, *optinal*) - 结果张量

例子: :

```
>>> torch.randn(4)

-0.1145
0.0094
-1.1717
0.9846
[torch.FloatTensor of size 4]

>>> torch.randn(2, 3)

1.4339  0.3351 -1.0999
1.5458 -0.9643 -0.3558
[torch.FloatTensor of size 2x3]
```

torch.randperm

```
torch.randperm(n, out=None) → LongTensor
```

给定参数 `n`，返回一个从 `0` 到 `n - 1` 的随机整数排列。

参数:

 v: latest ▼

- n (int) - 上边界(不包含)

例子:

```
>>> torch.randperm(4)

2
1
3
0
[torch.LongTensor of size 4]
```

torch.arange

```
torch.arange(start, end, step=1, out=None) → Tensor
```

返回一个1维张量，长度为 $\text{floor}((\text{end} - \text{start})/\text{step})$ 。包含从 `start` 到 `end`，以 `step` 为步长的一组序列值(默认步长为1)。

参数:

- start (float) – 序列的起始点
- end (float) – 序列的终止点
- step (float) – 相邻点的间隔大小
- out (Tensor, optional) – 结果张量

例子:

```
>>> torch.arange(1, 4)

1
2
3
[torch.FloatTensor of size 3]

>>> torch.arange(1, 2.5, 0.5)

1.0000
1.5000
2.0000
[torch.FloatTensor of size 3]
```

torch.range

```
torch.range(start, end, step=1, out=None) → Tensor
```

返回一个1维张量，有 $\text{floor}((\text{end} - \text{start})/\text{step}) + 1$ 个元素。包含在半开区间 `[start, end)` 从 `start` 开始，以 `step` 为步长的一组值。`step` 是两个值之间的间隔，即

$$x_{i+1} = x_i + \text{step}$$

警告： 建议使用函数 `torch.arange()`

参数:

- start (float) – 序列的起始点
- end (float) – 序列的最终值
- step (int) – 相邻点的间隔大小
- out (Tensor, optional) – 结果张量

例子:

```
>>> torch.range(1, 4)

1
2
3
4
[torch.FloatTensor of size 4]

>>> torch.range(1, 4, 0.5)

1.0000
1.5000
2.0000
2.5000
3.0000
3.5000
4.0000
[torch.FloatTensor of size 7]
```

torch.zeros

```
torch.zeros(*sizes, out=None) → Tensor
```

返回一个全为标量 0 的张量，形状由可变参数 `sizes` 定义。

参数:

- sizes (int...) – 整数序列，定义了输出形状
- out (Tensor, optional) – 结果张量

例子:


```
>>> torch.zeros(2, 3)

0  0  0
0  0  0
[torch.FloatTensor of size 2x3]

>>> torch.zeros(5)

0
0
0
0
0
[torch.FloatTensor of size 5]
```

索引,切片,连接,换位Indexing, Slicing, Joining, Mutating Ops

torch.cat

```
torch.cat(inputs, dimension=0) → Tensor
```

在给定维度上对输入的张量序列 `seq` 进行连接操作。

`torch.cat()` 可以看做 `torch.split()` 和 `torch.chunk()` 的反操作。 `cat()` 函数可以通过下面例子更好的理解。

参数:

- `inputs` (*sequence of Tensors*) – 可以是任意相同Tensor 类型的python 序列
- `dimension` (*int, optional*) – 沿着此维连接张量序列。

例子:

```
>>> x = torch.randn(2, 3)
>>> x

0.5983 -0.0341  2.4918
1.5981 -0.5265 -0.8735
[torch.FloatTensor of size 2x3]

>>> torch.cat((x, x, x), 0)

0.5983 -0.0341  2.4918
1.5981 -0.5265 -0.8735
0.5983 -0.0341  2.4918
1.5981 -0.5265 -0.8735
0.5983 -0.0341  2.4918
1.5981 -0.5265 -0.8735
[torch.FloatTensor of size 6x3]

>>> torch.cat((x, x, x), 1)

0.5983 -0.0341  2.4918  0.5983 -0.0341  2.4918  0.5983 -0.0341  2.4918
1.5981 -0.5265 -0.8735  1.5981 -0.5265 -0.8735  1.5981 -0.5265 -0.8735
[torch.FloatTensor of size 2x9]
```

torch.chunk

```
torch.chunk(tensor, chunks, dim=0)
```

在给定维度(轴)上将输入张量进行分块儿。

参数:

- tensor (Tensor) – 待分块的输入张量
- chunks (int) – 分块的个数
- dim (int) – 沿着此维度进行分块

torch.gather

```
torch.gather(input, dim, index, out=None) → Tensor
```

沿给定轴 `dim`，将输入索引张量 `index` 指定位置的值进行聚合。

对一个3维张量，输出可以定义为：

```
out[i][j][k] = tensor[index[i][j][k]][j][k] # dim=0
out[i][j][k] = tensor[i][index[i][j][k]][k] # dim=1
out[i][j][k] = tensor[i][j][index[i][j][k]] # dim=3
```

例子:


```
>>> t = torch.Tensor([[1,2],[3,4]])
>>> torch.gather(t, 1, torch.LongTensor([[0,0],[1,0]]))
 1  1
 4  3
[torch.FloatTensor of size 2x2]
```

参数:

- input (Tensor) – 源张量
- dim (int) – 索引的轴
- index (LongTensor) – 聚合元素的下标
- out (Tensor, optional) – 目标张量

torch.index_select

```
torch.index_select(input, dim, index, out=None) → Tensor
```

沿着指定维度对输入进行切片，取 `index` 中指定的相应项(`index` 为一个LongTensor)， [v: latest](#) ▾
回到一个新的张量，返回的张量与原始张量_Tensor_有相同的维度(在指定轴上)。

注意：返回的张量不与原始张量共享内存空间。

参数:

- input (Tensor) – 输入张量
- dim (int) – 索引的轴
- index (LongTensor) – 包含索引下标的一维张量
- out (Tensor, optional) – 目标张量

例子:

```
>>> x = torch.randn(3, 4)
>>> x

 1.2045  2.4084  0.4001  1.1372
 0.5596  1.5677  0.6219 -0.7954
 1.3635 -1.2313 -0.5414 -1.8478
[torch.FloatTensor of size 3x4]

>>> indices = torch.LongTensor([0, 2])
>>> torch.index_select(x, 0, indices)

 1.2045  2.4084  0.4001  1.1372
 1.3635 -1.2313 -0.5414 -1.8478
[torch.FloatTensor of size 2x4]

>>> torch.index_select(x, 1, indices)

 1.2045  0.4001
 0.5596  0.6219
 1.3635 -0.5414
[torch.FloatTensor of size 3x2]
```

torch.masked_select

```
torch.masked_select(input, mask, out=None) → Tensor
```

根据掩码张量 `mask` 中的二元值，取输入张量中的指定项(`mask` 为一个 *ByteTensor*)，将取值返回到一个新的1D张量，

张量 `mask` 须跟 `input` 张量有相同数量的元素数目，但形状或维度不需要相同。注意：返回的张量不与原始张量共享内存空间。

参数:

- input (Tensor) – 输入张量
- mask (ByteTensor) – 掩码张量，包含了二元索引值
- out (Tensor, optional) – 目标张量

例子:

```
>>> x = torch.randn(3, 4)
>>> x

1.2045  2.4084  0.4001  1.1372
0.5596  1.5677  0.6219 -0.7954
1.3635 -1.2313 -0.5414 -1.8478
[torch.FloatTensor of size 3x4]

>>> indices = torch.LongTensor([0, 2])
>>> torch.index_select(x, 0, indices)

1.2045  2.4084  0.4001  1.1372
1.3635 -1.2313 -0.5414 -1.8478
[torch.FloatTensor of size 2x4]

>>> torch.index_select(x, 1, indices)

1.2045  0.4001
0.5596  0.6219
1.3635 -0.5414
[torch.FloatTensor of size 3x2]
```

torch.nonzero

```
torch.nonzero(input, out=None) → LongTensor
```

返回一个包含输入 `input` 中非零元素索引的张量。输出张量中的每行包含输入中非零元素的索引。

如果输入 `input` 有 `n` 维，则输出的索引张量 `output` 的形状为 $z \times n$ ，这里 z 是输入张量 `input` 中所有非零元素的个数。

参数:

- input (Tensor) – 源张量
- out (LongTensor, optional) – 包含索引值的结果张量

例子:

```
>>> torch.nonzero(torch.Tensor([1, 1, 1, 0, 1]))

0
1
2
4
[torch.LongTensor of size 4x1]

>>> torch.nonzero(torch.Tensor([[0.6, 0.0, 0.0, 0.0],
...                             [0.0, 0.4, 0.0, 0.0],
...                             [0.0, 0.0, 1.2, 0.0],
...                             [0.0, 0.0, 0.0, -0.4]])))

0 0
1 1
2 2
3 3
[torch.LongTensor of size 4x2]
```

 v: latest ▼

torch.split

```
torch.split(tensor, split_size, dim=0)
```

将输入张量分割成相等形状的chunks（如果可分）。如果沿指定维的张量形状大小不能被 `split_size` 整分，则最后一个分块会小于其它分块。

参数:

- tensor (Tensor) – 待分割张量
- split_size (int) – 单个分块的形状大小
- dim (int) – 沿着此维进行分割

torch.squeeze

```
torch.squeeze(input, dim=None, out=None)
```

将输入张量形状中的 `1` 去除并返回。如果输入是形如 $(A \times 1 \times B \times 1 \times C \times 1 \times D)$ ，那么输出形状就为: $(A \times B \times C \times D)$

当给定 `dim` 时，那么挤压操作只在给定维度上。例如，输入形状为: $(A \times 1 \times B)$ ，`squeeze(input, 0)` 将会保持张量不变，只有用 `squeeze(input, 1)`，形状会变成 $(A \times B)$ 。

注意：返回张量与输入张量共享内存，所以改变其中一个的内容会改变另一个。

参数:

- input (Tensor) – 输入张量
- dim (int, optional) – 如果给定，则 `input` 只会在给定维度挤压
- out (Tensor, optional) – 输出张量

例子:

```
>>> x = torch.zeros(2,1,2,1,2)
>>> x.size()
(2L, 1L, 2L, 1L, 2L)
>>> y = torch.squeeze(x)
>>> y.size()
(2L, 2L, 2L)
>>> y = torch.squeeze(x, 0)
>>> y.size()
(2L, 1L, 2L, 1L, 2L)
>>> y = torch.squeeze(x, 1)
>>> y.size()
(2L, 2L, 1L, 2L)
```

torch.stack[source]

```
torch.stack(sequence, dim=0)
```

沿着一个新维度对输入张量序列进行连接。序列中所有的张量都应该为相同形状。

参数:

- `squence` (Sequence) – 待连接的张量序列
- `dim` (int) – 插入的维度。必须介于 0 与 待连接的张量序列数之间。

`torch.t`

```
torch.t(input, out=None) → Tensor
```

输入一个矩阵（2维张量），并转置0, 1维。可以被视为函数 `transpose(input, 0, 1)` 的简写函数。

参数:

- `input` (Tensor) – 输入张量
- `out` (Tensor, optional) – 结果张量

```
>>> x = torch.randn(2, 3)
>>> x

 0.4834  0.6907  1.3417
-0.1300  0.5295  0.2321
[torch.FloatTensor of size 2x3]

>>> torch.t(x)

 0.4834 -0.1300
 0.6907  0.5295
 1.3417  0.2321
[torch.FloatTensor of size 3x2]
```

`torch.transpose`

```
torch.transpose(input, dim0, dim1, out=None) → Tensor
```

返回输入矩阵 `input` 的转置。交换维度 `dim0` 和 `dim1`。输出张量与输入张量共享内存，所以改变其中一个会导致另外一个也被修改。

参数:

- `input` (Tensor) – 输入张量
- `dim0` (int) – 转置的第一维
- `dim1` (int) – 转置的第二维

```
>>> x = torch.randn(2, 3)
>>> x

0.5983 -0.0341  2.4918
1.5981 -0.5265 -0.8735
[torch.FloatTensor of size 2x3]

>>> torch.transpose(x, 0, 1)

0.5983  1.5981
-0.0341 -0.5265
2.4918 -0.8735
[torch.FloatTensor of size 3x2]
```

torch.unbind

```
torch.unbind(tensor, dim=0)[source]
```

移除指定维后，返回一个元组，包含了沿着指定维切片后的各个切片

参数:

- tensor (Tensor) – 输入张量
- dim (int) – 删除的维度

torch.unsqueeze

```
torch.unsqueeze(input, dim, out=None)
```

返回一个新的张量，对输入的制定位置插入维度 1

注意：返回张量与输入张量共享内存，所以改变其中一个的内容会改变另一个。

如果 `dim` 为负，则将会被转化 $dim + input.dim() + 1$

参数:

- tensor (Tensor) – 输入张量
- dim (int) – 插入维度的索引
- out (Tensor, optional) – 结果张量

```
>>> x = torch.Tensor([1, 2, 3, 4])
>>> torch.unsqueeze(x, 0)
1  2  3  4
[torch.FloatTensor of size 1x4]
>>> torch.unsqueeze(x, 1)
1
2
3
4
[torch.FloatTensor of size 4x1]
```

随机抽样 Random sampling

torch.manual_seed

```
torch.manual_seed(seed)
```

设定生成随机数的种子，并返回一个 *torch._C.Generator* 对象。

参数: seed (int or long) – 种子。

torch.initial_seed

```
torch.initial_seed()
```

返回生成随机数的原始种子值（python long）。

torch.get_rng_state

```
torch.get_rng_state()[source]
```

返回随机生成器状态(*ByteTensor*)

torch.set_rng_state

```
torch.set_rng_state(new_state)[source]
```

设定随机生成器状态 参数: new_state (torch.ByteTensor) – 期望的状态

torch.default_generator

```
torch.default_generator = <torch._C.Generator object>
```

torch.bernoulli

```
torch.bernoulli(input, out=None) → Tensor
```

从伯努利分布中抽取二元随机数(0 或者 1)。

输入张量须包含用于抽取上述二元随机值的概率。因此，输入中的所有值都必须在 [0,1] 区间，即 $0 \leq input_i \leq 1$

输出张量的第 i 个元素值，将会以输入张量的第 i 个概率值等于 1 。

返回值将会是与输入相同大小的张量，每个值为0或者1 参数:

- input (Tensor) – 输入为伯努利分布的概率值
- out (Tensor, optional) – 输出张量(可选)

例子:

```
>>> a = torch.Tensor(3, 3).uniform_(0, 1) # generate a uniform random matrix with range [0, 1]
>>> a

0.7544  0.8140  0.9842
0.5282  0.0595  0.6445
0.1925  0.9553  0.9732
[torch.FloatTensor of size 3x3]

>>> torch.bernoulli(a)

1  1  1
0  0  1
0  1  1
[torch.FloatTensor of size 3x3]

>>> a = torch.ones(3, 3) # probability of drawing "1" is 1
>>> torch.bernoulli(a)

1  1  1
1  1  1
1  1  1
[torch.FloatTensor of size 3x3]

>>> a = torch.zeros(3, 3) # probability of drawing "1" is 0
>>> torch.bernoulli(a)

0  0  0
0  0  0
0  0  0
[torch.FloatTensor of size 3x3]
```

torch.multinomial


```
torch.multinomial(input, num_samples, replacement=False, out=None) → LongTensor
```

返回一个张量，每行包含从 `input` 相应行中定义的多项分布中抽取的 `num_samples` 个样本。

[注意]:输入 `input` 每行的值不需要总和为1 (这里我们用来做权重)，但是必须非负且总和不能为0。

当抽取样本时，依次从左到右排列(第一个样本对应第一列)。

如果输入 `input` 是一个向量，输出 `out` 也是一个相同长度 `num_samples` 的向量。如果输入 `input` 是有 m 行的矩阵，输出 `out` 是形如 $m \times n$ 的矩阵。

如果参数 `replacement` 为 `True`, 则样本抽取可以重复。否则，一个样本在每行不能被重  [v: latest](#) ▼

参数 `num_samples` 必须小于 `input` 长度(即, `input` 的列数, 如果是 `input` 是一个矩阵)。

参数:

- `input` (Tensor) – 包含概率值的张量
- `num_samples` (int) – 抽取的样本数
- `replacement` (bool, optional) – 布尔值, 决定是否重复抽取
- `out` (Tensor, optional) – 结果张量

例子:

```
>>> weights = torch.Tensor([0, 10, 3, 0]) # create a Tensor of weights
>>> torch.multinomial(weights, 4)

1
2
0
0
[torch.LongTensor of size 4]

>>> torch.multinomial(weights, 4, replacement=True)

1
2
1
2
[torch.LongTensor of size 4]
```

`torch.normal()`

```
torch.normal(means, std, out=None)
```

返回一个张量, 包含从给定参数 `means`, `std` 的离散正态分布中抽取随机数。均值 `means` 是一个张量, 包含每个输出元素相关的正态分布的均值。 `std` 是一个张量, 包含每个输出元素相关的正态分布的标准差。均值和标准差的形状不须匹配, 但每个张量的元素个数须相同。

参数:

- `means` (Tensor) – 均值
- `std` (Tensor) – 标准差
- `out` (Tensor) – 可选的输出张量

```
torch.normal(means=torch.arange(1, 11), std=torch.arange(1, 0, -0.1))

1.5104
1.6955
2.4895
4.9185
4.9895
6.9155
7.3683
8.1836
8.7164
9.8916
[torch.FloatTensor of size 10]
```

```
torch.normal(mean=0.0, std, out=None)
```

与上面函数类似，所有抽取的样本共享均值。

参数:

- means (Tensor, optional) – 所有分布均值
- std (Tensor) – 每个元素的标准差
- out (Tensor) – 可选的输出张量

例子:

```
>>> torch.normal(mean=0.5, std=torch.arange(1, 6))  
  
 0.5723  
 0.0871  
-0.3783  
-2.5689  
10.7893  
[torch.FloatTensor of size 5]
```

```
torch.normal(means, std=1.0, out=None)
```

与上面函数类似，所有抽取的样本共享标准差。

参数:

- means (Tensor) – 每个元素的均值
- std (float, optional) – 所有分布的标准差
- out (Tensor) – 可选的输出张量

例子:

```
>>> torch.normal(means=torch.arange(1, 6))  
  
 1.1681  
 2.8884  
 3.7718  
 2.5616  
 4.2500  
[torch.FloatTensor of size 5]
```

序列化 Serialization

`torch.save`[\[source\]](#)

```
torch.save(obj, f, pickle_module=<module 'pickle' from '/home/jenkins/miniconda/lib/python3.5/pic
```

保存一个对象到一个硬盘文件上 参考: [Recommended approach for saving a model](#) 参数:

- obj - 保存对象
- f - 类文件对象 (返回文件描述符) 或一个保存文件名的字符串
- pickle_module - 用于pickling元数据和对象的模块
- pickle_protocol - 指定pickle protocol 可以覆盖默认参数

torch.load[source]

```
torch.load(f, map_location=None, pickle_module=<module 'pickle' from '/home/jenkins/miniconda/lib
```

从磁盘文件中读取一个通过 `torch.save()` 保存的对象。 `torch.load()` 可通过参数 `map_location` 动态地进行内存重映射, 使其能从不动设备中读取文件。一般调用时, 需两个参数: storage 和 location tag. 返回不同地址中的storage, 或着返回None (此时地址可以通过默认方法进行解析). 如果这个参数是字典的话, 意味着其是从文件的地址标记到当前系统的地址标记的映射。默认情况下, location tags中 "cpu"对应host tensors, 'cuda:device_id' (e.g. 'cuda:2') 对应 cuda tensors。用户可以通过register_package进行扩展, 使用自己定义的标记和反序列化方法。

参数:

- f - 类文件对象 (返回文件描述符) 或一个保存文件名的字符串
- map_location - 一个函数或字典规定如何remap存储位置
- pickle_module - 用于unpickling元数据和对象的模块 (必须匹配序列化文件时的 pickle_module)

例子:

```
>>> torch.load('tensors.pt')
# Load all tensors onto the CPU
>>> torch.load('tensors.pt', map_location=lambda storage, loc: storage)
# Map tensors from GPU 1 to GPU 0
>>> torch.load('tensors.pt', map_location={'cuda:1': 'cuda:0'})
```

并行化 Parallelism

torch.get_num_threads

```
torch.get_num_threads() → int
```

 v: latest ▼

获得用于并行化CPU操作的OpenMP线程数

torch.set_num_threads

```
torch.set_num_threads(int)
```

设定用于并行化CPU操作的OpenMP线程数

数学操作Math operations

Pointwise Ops

torch.abs

```
torch.abs(input, out=None) → Tensor
```

计算输入张量的每个元素绝对值

例子：

```
>>> torch.abs(torch.FloatTensor([-1, -2, 3]))  
FloatTensor([1, 2, 3])
```

torch.acos(input, out=None) → Tensor

```
torch.acos(input, out=None) → Tensor
```

返回一个新张量，包含输入张量每个元素的反余弦。 参数：

- input (Tensor) – 输入张量
- out (Tensor, optional) – 结果张量

例子：

```
>>> a = torch.randn(4)  
>>> a  
  
-0.6366  
0.2718  
0.4469  
1.3122  
[torch.FloatTensor of size 4]  
  
>>> torch.acos(a)  
2.2608  
1.2956  
1.1075  
nan  
[torch.FloatTensor of size 4]
```

 v: latest ▼

torch.add()

```
torch.add(input, value, out=None)
```

对输入张量 `input` 逐元素加上标量值 `value`，并返回结果到一个新的张量 `out`，即 $out = tensor + value$ 。

如果输入 `input` 是 `FloatTensor` or `DoubleTensor` 类型，则 `value` 必须为实数，否则须为整数。【译注：似乎并非如此，无关输入类型，`value` 取整数、实数皆可。】

- `input` (Tensor) – 输入张量
- `value` (Number) – 添加到输入每个元素的数
- `out` (Tensor, optional) – 结果张量

```
>>> a = torch.randn(4)
>>> a

 0.4050
-1.2227
 1.8688
-0.4185
[torch.FloatTensor of size 4]

>>> torch.add(a, 20)

20.4050
18.7773
21.8688
19.5815
[torch.FloatTensor of size 4]
```

```
torch.add(input, value=1, other, out=None)
```

`other` 张量的每个元素乘以一个标量值 `value`，并加到 `input` 张量上。返回结果到输出张量 `out`。即， $out = input + (other * value)$

两个张量 `input` and `other` 的尺寸不需要匹配，但元素总数必须一样。

注意：当两个张量形状不匹配时，输入张量的形状会作为输出张量的尺寸。

如果 `other` 是 `FloatTensor` or `DoubleTensor` 类型，则 `value` 必须为实数，否则须为整数。【译注：似乎并非如此，无关输入类型，`value` 取整数、实数皆可。】

参数:

- `input` (Tensor) – 第一个输入张量
- `value` (Number) – 用于第二个张量的尺寸因子
- `other` (Tensor) – 第二个输入张量
- `out` (Tensor, optional) – 结果张量

例子:

```
>>> import torch
>>> a = torch.randn(4)
>>> a

-0.9310
 2.0330
 0.0852
-0.2941
[torch.FloatTensor of size 4]

>>> b = torch.randn(2, 2)
>>> b

 1.0663  0.2544
-0.1513  0.0749
[torch.FloatTensor of size 2x2]

>>> torch.add(a, 10, b)
 9.7322
 4.5770
-1.4279
 0.4552
[torch.FloatTensor of size 4]
```

torch.addcddiv

```
torch.addcddiv(tensor, value=1, tensor1, tensor2, out=None) → Tensor
```

用 `tensor2` 对 `tensor1` 逐元素相除，然后乘以标量值 `value` 并加到 `tensor`。

张量的形状不需要匹配，但元素数量必须一致。

如果输入是FloatTensor or DoubleTensor类型，则 `value` 必须为实数，否则须为整数。

参数：

- tensor (Tensor) – 张量，对 tensor1 ./ tensor 进行相加
- value (Number, optional) – 标量，对 tensor1 ./ tensor2 进行相乘
- tensor1 (Tensor) – 张量，作为被除数(分子)
- tensor2 (Tensor) – 张量，作为除数(分母)
- out (Tensor, optional) – 输出张量

例子：

```
>>> t = torch.randn(2, 3)
>>> t1 = torch.randn(1, 6)
>>> t2 = torch.randn(6, 1)
>>> torch.addcddiv(t, 0.1, t1, t2)

 0.0122 -0.0188 -0.2354
 0.7396 -1.5721  1.2878
[torch.FloatTensor of size 2x3]
```

torch.addcmul

```
torch.addcmul(tensor, value=1, tensor1, tensor2, out=None) → Tensor
```

用 `tensor2` 对 `tensor1` 逐元素相乘，并对结果乘以标量值 `value` 然后加到 `tensor`。张量的形状不需要匹配，但元素数量必须一致。如果输入是 `FloatTensor` or `DoubleTensor` 类型，则 `value` 必须为实数，否则须为整数。

参数：

- `tensor (Tensor)` – 张量，对 `tensor1 ./ tensor` 进行相加
- `value (Number, optional)` – 标量，对 `tensor1 . tensor2` 进行相乘
- `tensor1 (Tensor)` – 张量，作为乘子1
- `tensor2 (Tensor)` – 张量，作为乘子2
- `out (Tensor, optional)` – 输出张量

例子：

```
>>> t = torch.randn(2, 3)
>>> t1 = torch.randn(1, 6)
>>> t2 = torch.randn(6, 1)
>>> torch.addcmul(t, 0.1, t1, t2)

0.0122 -0.0188 -0.2354
0.7396 -1.5721 1.2878
[torch.FloatTensor of size 2x3]
```

torch.asin

```
torch.asin(input, out=None) → Tensor
```

返回一个新张量，包含输入 `input` 张量每个元素的反正弦函数

参数：

- `tensor (Tensor)` – 输入张量
- `out (Tensor, optional)` – 输出张量

例子：


```
>>> a = torch.randn(4)
>>> a
-0.6366
 0.2718
 0.4469
 1.3122
[torch.FloatTensor of size 4]

>>> torch.asin(a)
-0.6900
 0.2752
 0.4633
   nan
[torch.FloatTensor of size 4]
```

torch.atan

```
torch.atan(input, out=None) → Tensor
```

返回一个新张量，包含输入 `input` 张量每个元素的反正切函数

参数：

- tensor (Tensor) – 输入张量
- out (Tensor, optional) – 输出张量

例子：

```
>>> a = torch.randn(4)
>>> a
-0.6366
 0.2718
 0.4469
 1.3122
[torch.FloatTensor of size 4]

>>> torch.atan(a)
-0.5669
 0.2653
 0.4203
 0.9196
[torch.FloatTensor of size 4]
```

torch.atan2

```
torch.atan2(input1, input2, out=None) → Tensor
```

返回一个新张量，包含两个输入张量 `input1` 和 `input2` 的反正切函数

参数：

- input1 (Tensor) – 第一个输入张量
- input2 (Tensor) – 第二个输入张量
- out (Tensor, optional) – 输出张量

例子：

```
>>> a = torch.randn(4)
>>> a
-0.6366
 0.2718
 0.4469
 1.3122
[torch.FloatTensor of size 4]

>>> torch.atan2(a, torch.randn(4))
-2.4167
 2.9755
 0.9363
 1.6613
[torch.FloatTensor of size 4]
```

torch.ceil

```
torch.ceil(input, out=None) → Tensor
```

天井函数，对输入 `input` 张量每个元素向上取整，即取不小于每个元素的最小整数，并返回结果到输出。

参数：

- input (Tensor) – 输入张量
- out (Tensor, optional) – 输出张量

例子：

```
>>> a = torch.randn(4)
>>> a
 1.3869
 0.3912
-0.8634
-0.5468
[torch.FloatTensor of size 4]

>>> torch.ceil(a)
 2
 1
-0
-0
[torch.FloatTensor of size 4]
```

torch.clamp

```
torch.clamp(input, min, max, out=None) → Tensor
```

将输入 `input` 张量每个元素的夹紧到区间 $[min, max]$ ，并返回结果到一个新张量。

 v: latest ▼

操作定义如下：

```
y_i = | min, if x_i < min
      | x_i, if min <= x_i <= max
      | max, if x_i > max
```

如果输入是FloatTensor or DoubleTensor类型，则参数 `min` `max` 必须为实数，否则须为整数。【译注：似乎并非如此，无关输入类型，`min`，`max`取整数、实数皆可。】

参数：

- input (Tensor) – 输入张量
- min (Number) – 限制范围下限
- max (Number) – 限制范围上限
- out (Tensor, optional) – 输出张量

例子：

```
>>> a = torch.randn(4)
>>> a

 1.3869
 0.3912
-0.8634
-0.5468
[torch.FloatTensor of size 4]

>>> torch.clamp(a, min=-0.5, max=0.5)

 0.5000
 0.3912
-0.5000
-0.5000
[torch.FloatTensor of size 4]
```

```
torch.clamp(input, *, min, out=None) → Tensor
```

将输入 `input` 张量每个元素的限制到不小于 `min`，并返回结果到一个新张量。

如果输入是FloatTensor or DoubleTensor类型，则参数 `min` 必须为实数，否则须为整数。【译注：似乎并非如此，无关输入类型，`min`取整数、实数皆可。】

参数：

- input (Tensor) – 输入张量
- value (Number) – 限制范围下限
- out (Tensor, optional) – 输出张量

例子：

```
>>> a = torch.randn(4)
>>> a

1.3869
0.3912
-0.8634
-0.5468
[torch.FloatTensor of size 4]

>>> torch.clamp(a, min=0.5)

1.3869
0.5000
0.5000
0.5000
[torch.FloatTensor of size 4]
```

```
torch.clamp(input, *, max, out=None) → Tensor
```

将输入 `input` 张量每个元素的限制到不大于 `max`，并返回结果到一个新张量。

如果输入是FloatTensor or DoubleTensor类型，则参数 `max` 必须为实数，否则须为整数。

【译注：似乎并非如此，无关输入类型，`max` 取整数、实数皆可。】

参数：

- input (Tensor) – 输入张量
- value (Number) – 限制范围上限
- out (Tensor, optional) – 输出张量

例子：

```
>>> a = torch.randn(4)
>>> a

1.3869
0.3912
-0.8634
-0.5468
[torch.FloatTensor of size 4]

>>> torch.clamp(a, max=0.5)

0.5000
0.3912
-0.8634
-0.5468
[torch.FloatTensor of size 4]
```

torch.cos

```
torch.cos(input, out=None) → Tensor
```

返回一个新张量，包含输入 `input` 张量每个元素的余弦。

 v: latest ▼

参数：

- input (Tensor) – 输入张量
- out (Tensor, optional) – 输出张量

例子:

```
>>> a = torch.randn(4)
>>> a
-0.6366
 0.2718
 0.4469
 1.3122
[torch.FloatTensor of size 4]

>>> torch.cos(a)
 0.8041
 0.9633
 0.9018
 0.2557
[torch.FloatTensor of size 4]
```

torch.cosh

```
torch.cosh(input, out=None) → Tensor
```

返回一个新张量，包含输入 `input` 张量每个元素的双曲余弦。

参数:

- input (Tensor) – 输入张量
- out (Tensor, optional) – 输出张量

例子:

```
>>> a = torch.randn(4)
>>> a
-0.6366
 0.2718
 0.4469
 1.3122
[torch.FloatTensor of size 4]

>>> torch.cosh(a)
 1.2095
 1.0372
 1.1015
 1.9917
[torch.FloatTensor of size 4]
```

torch.div()

```
torch.div(input, value, out=None)
```

将 `input` 逐元素除以标量值 `value`，并返回结果到输出张量 `out`。即 $out = tensor / \dots$  [v: latest](#) ▼

如果输入是FloatTensor or DoubleTensor类型，则参数 `value` 必须为实数，否则须为整数。

【译注：似乎并非如此，无关输入类型，`value` 取整数、实数皆可。】

参数：

- input (Tensor) – 输入张量
- value (Number) – 除数
- out (Tensor, optional) – 输出张量

例子：

```
>>> a = torch.randn(5)
>>> a

-0.6147
-1.1237
-0.1604
-0.6853
 0.1063
[torch.FloatTensor of size 5]

>>> torch.div(a, 0.5)

-1.2294
-2.2474
-0.3208
-1.3706
 0.2126
[torch.FloatTensor of size 5]
```

```
torch.div(input, other, out=None)
```

两张量 `input` 和 `other` 逐元素相除，并将结果返回到输出。即， $out_i = input_i / other_i$

两张量形状不须匹配，但元素数须一致。

注意：当形状不匹配时，`input` 的形状作为输出张量的形状。

参数：

- input (Tensor) – 张量(分子)
- other (Tensor) – 张量(分母)
- out (Tensor, optional) – 输出张量

例子：

```

>>> a = torch.randn(4,4)
>>> a

-0.1810  0.4017  0.2863 -0.1013
 0.6183  2.0696  0.9012 -1.5933
 0.5679  0.4743 -0.0117 -0.1266
-0.1213  0.9629  0.2682  1.5968
[torch.FloatTensor of size 4x4]

>>> b = torch.randn(8, 2)
>>> b

 0.8774  0.7650
 0.8866  1.4805
-0.6490  1.1172
 1.4259 -0.8146
 1.4633 -0.1228
 0.4643 -0.6029
 0.3492  1.5270
 1.6103 -0.6291
[torch.FloatTensor of size 8x2]

>>> torch.div(a, b)

-0.2062  0.5251  0.3229 -0.0684
-0.9528  1.8525  0.6320  1.9559
 0.3881 -3.8625 -0.0253  0.2099
-0.3473  0.6306  0.1666 -2.5381
[torch.FloatTensor of size 4x4]

```

torch.exp

```
torch.exp(tensor, out=None) → Tensor
```

返回一个新张量，包含输入 `input` 张量每个元素的指数。

参数：

- input (Tensor) – 输入张量
- out (Tensor, optional) – 输出张量

```

>>> torch.exp(torch.Tensor([0, math.log(2)]))
torch.FloatTensor([1, 2])

```

torch.floor

```
torch.floor(input, out=None) → Tensor
```

床函数: 返回一个新张量，包含输入 `input` 张量每个元素的floor，即不小于元素的最大整数。

参数：

- input (Tensor) – 输入张量
- out (Tensor, optional) – 输出张量

 v: latest ▼

例子：

```
>>> a = torch.randn(4)
>>> a

 1.3869
 0.3912
-0.8634
-0.5468
[torch.FloatTensor of size 4]

>>> torch.floor(a)

 1
 0
-1
-1
[torch.FloatTensor of size 4]
```

torch.fmod

```
torch.fmod(input, divisor, out=None) → Tensor
```

计算除法余数。除数与被除数可能同时含有整数和浮点数。此时，余数的正负与被除数相同。

参数：- input (Tensor) – 被除数 - divisor (Tensor or float) – 除数，一个数或与被除数相同类型的张量 - out (Tensor, optional) – 输出张量

例子：

```
>>> torch.fmod(torch.Tensor([-3, -2, -1, 1, 2, 3]), 2)
torch.FloatTensor([-1, -0, -1, 1, 0, 1])
>>> torch.fmod(torch.Tensor([1, 2, 3, 4, 5]), 1.5)
torch.FloatTensor([1.0, 0.5, 0.0, 1.0, 0.5])
```

参考: [torch.remainder\(\)](#) , 计算逐元素余数，相当于python中的%操作符。

torch.frac

```
torch.frac(tensor, out=None) → Tensor
```

返回每个元素的分数部分。

例子：

```
>>> torch.frac(torch.Tensor([1, 2.5, -3.2]))
torch.FloatTensor([0, 0.5, -0.2])
```

torch.lerp

```
torch.lerp(start, end, weight, out=None)
```


对两个张量以 `start` , `end` 做线性插值, 将结果返回到输出张量。

即, $out_i = start_i + weight * (end_i - start_i)$

参数:

- `start` (Tensor) – 起始点张量
- `end` (Tensor) – 终止点张量
- `weight` (float) – 插值公式的weight
- `out` (Tensor, optional) – 结果张量

例子:

```
>>> start = torch.arange(1, 5)
>>> end = torch.Tensor(4).fill_(10)
>>> start

 1
 2
 3
 4
[torch.FloatTensor of size 4]

>>> end

10
10
10
10
[torch.FloatTensor of size 4]

>>> torch.lerp(start, end, 0.5)

5.5000
6.0000
6.5000
7.0000
[torch.FloatTensor of size 4]
```

torch.log

```
torch.log(input, out=None) → Tensor
```

计算 `input` 的自然对数

参数:

- `input` (Tensor) – 输入张量
- `out` (Tensor, optional) – 输出张量

例子:

```
>>> a = torch.randn(5)
>>> a

-0.4183
 0.3722
-0.3091
 0.4149
 0.5857
[torch.FloatTensor of size 5]

>>> torch.log(a)

      nan
-0.9883
      nan
-0.8797
-0.5349
[torch.FloatTensor of size 5]
```

torch.log1p

```
torch.log1p(input, out=None) → Tensor
```

计算 $\text{input} + 1$ 的自然对数 $y_i = \log(x_i + 1)$

注意：对值比较小的输入，此函数比 `torch.log()` 更准确。

如果输入是FloatTensor or DoubleTensor类型，则 `value` 必须为实数，否则须为整数。

参数：

- input (Tensor) – 输入张量
- out (Tensor, optional) – 输出张量

例子：

```
>>> a = torch.randn(5)
>>> a

-0.4183
 0.3722
-0.3091
 0.4149
 0.5857
[torch.FloatTensor of size 5]

>>> torch.log1p(a)

-0.5418
 0.3164
-0.3697
 0.3471
 0.4611
[torch.FloatTensor of size 5]
```

torch.mul

```
torch.mul(input, value, out=None)
```

用标量值 `value` 乘以输入 `input` 的每个元素，并返回一个新的结果张量。

$out = tensor * value$

如果输入是FloatTensor or DoubleTensor类型，则 `value` 必须为实数，否则须为整数。【译注：似乎并非如此，无关输入类型，`value` 取整数、实数皆可。】

参数：

- input (Tensor) – 输入张量
- value (Number) – 乘到每个元素的数
- out (Tensor, optional) – 输出张量

例子：

```
>>> a = torch.randn(3)
>>> a

-0.9374
-0.5254
-0.6069
[torch.FloatTensor of size 3]

>>> torch.mul(a, 100)

-93.7411
-52.5374
-60.6908
[torch.FloatTensor of size 3]
```

```
torch.mul(input, other, out=None)
```

两个张量 `input` , `other` 按元素进行相乘，并返回到输出张量。即计算 $out_i = input_i * other_i$

两计算张量形状不须匹配，但总元素数须一致。 **注意：**当形状不匹配时，`input` 的形状作为输入张量的形状。

参数：

- input (Tensor) – 第一个相乘张量
- other (Tensor) – 第二个相乘张量
- out (Tensor, optional) – 结果张量

例子：

```

>>> a = torch.randn(4,4)
>>> a

-0.7280  0.0598 -1.4327 -0.5825
-0.1427 -0.0690  0.0821 -0.3270
-0.9241  0.5110  0.4070 -1.1188
-0.8308  0.7426 -0.6240 -1.1582
[torch.FloatTensor of size 4x4]

>>> b = torch.randn(2, 8)
>>> b

 0.0430 -1.0775  0.6015  1.1647 -0.6549  0.0308 -0.1670  1.0742
-1.2593  0.0292 -0.0849  0.4530  1.2404 -0.4659 -0.1840  0.5974
[torch.FloatTensor of size 2x8]

>>> torch.mul(a, b)

-0.0313 -0.0645 -0.8618 -0.6784
 0.0934 -0.0021 -0.0137 -0.3513
 1.1638  0.0149 -0.0346 -0.5068
-1.0304 -0.3460  0.1148 -0.6919
[torch.FloatTensor of size 4x4]

```

torch.neg

```
torch.neg(input, out=None) → Tensor
```

返回一个新张量，包含输入 `input` 张量按元素取负。即， $out = -1 * input$

参数：

- input (Tensor) – 输入张量
- out (Tensor, optional) – 输出张量

例子：

```

>>> a = torch.randn(5)
>>> a

-0.4430
 1.1690
-0.8836
-0.4565
 0.2968
[torch.FloatTensor of size 5]

>>> torch.neg(a)

 0.4430
-1.1690
 0.8836
 0.4565
-0.2968
[torch.FloatTensor of size 5]

```

torch.pow

```
torch.pow(input, exponent, out=None)
```

对输入 `input` 的按元素求 `exponent` 次幂值，并返回结果张量。幂值 `exponent` 可以为单一 `float` 数或者与 `input` 相同元素数的张量。

当幂值为标量时，执行操作：

$$out_i = x^{exponent}$$

当幂值为张量时，执行操作：

$$out_i = x^{exponent_i}$$

参数：

- `input` (Tensor) – 输入张量
- `exponent` (float or Tensor) – 幂值
- `out` (Tensor, optional) – 输出张量

例子：

```
>>> a = torch.randn(4)
>>> a

-0.5274
-0.8232
-2.1128
 1.7558
[torch.FloatTensor of size 4]

>>> torch.pow(a, 2)

 0.2781
 0.6776
 4.4640
 3.0829
[torch.FloatTensor of size 4]

>>> exp = torch.arange(1, 5)
>>> a = torch.arange(1, 5)
>>> a

 1
 2
 3
 4
[torch.FloatTensor of size 4]

>>> exp

 1
 2
 3
 4
[torch.FloatTensor of size 4]

>>> torch.pow(a, exp)

 1
 4
27
256
[torch.FloatTensor of size 4]
```

```
torch.pow(base, input, out=None)
```

`base` 为标量浮点值, `input` 为张量, 返回的输出张量 `out` 与输入张量相同形状。

执行操作为:

$$out_i = base^{input_i}$$

参数:

- `base` (float) – 标量值, 指数的底
- `input` (Tensor) – 幂值
- `out` (Tensor, optional) – 输出张量

例子:

```
>>> exp = torch.arange(1, 5)
>>> base = 2
>>> torch.pow(base, exp)

 2
 4
 8
16
[torch.FloatTensor of size 4]
```

torch.reciprocal

```
torch.reciprocal(input, out=None) → Tensor
```

返回一个新张量, 包含输入 `input` 张量每个元素的倒数, 即 $1.0/x$ 。

参数:

- `input` (Tensor) – 输入张量
- `out` (Tensor, optional) – 输出张量

例子:

```
>>> a = torch.randn(4)
>>> a

 1.3869
 0.3912
-0.8634
-0.5468
[torch.FloatTensor of size 4]

>>> torch.reciprocal(a)

 0.7210
 2.5565
-1.1583
-1.8289
[torch.FloatTensor of size 4]
```

torch.remainder

```
torch.remainder(input, divisor, out=None) → Tensor
```

返回一个新张量，包含输入 `input` 张量每个元素的除法余数。除数与被除数可能同时包含整数或浮点数。余数与除数有相同的符号。

参数：

- input (Tensor) – 被除数
- divisor (Tensor or float) – 除数，一个数或者与除数相同大小的张量
- out (Tensor, optional) – 输出张量

例子：

```
>>> torch.remainder(torch.Tensor([-3, -2, -1, 1, 2, 3]), 2)
torch.FloatTensor([1, 0, 1, 1, 0, 1])
>>> torch.remainder(torch.Tensor([1, 2, 3, 4, 5]), 1.5)
torch.FloatTensor([1.0, 0.5, 0.0, 1.0, 0.5])
```

参考：函数 `torch.fmod()` 同样可以计算除法余数，相当于 C 的库函数 `fmod()`

torch.round

```
torch.round(input, out=None) → Tensor
```

返回一个新张量，将输入 `input` 张量每个元素舍入到最近的整数。

参数：

- input (Tensor) – 输入张量
- out (Tensor, optional) – 输出张量

 v: latest ▼

例子：

```
>>> a = torch.randn(4)
>>> a

1.2290
1.3409
-0.5662
-0.0899
[torch.FloatTensor of size 4]

>>> torch.round(a)

1
1
-1
-0
[torch.FloatTensor of size 4]
```

torch.rsqrt

```
torch.rsqrt(input, out=None) → Tensor
```

返回一个新张量，包含输入 `input` 张量每个元素的平方根倒数。

参数：

- input (Tensor) – 输入张量
- out (Tensor, optional) – 输出张量

例子：

```
>>> a = torch.randn(4)
>>> a

1.2290
1.3409
-0.5662
-0.0899
[torch.FloatTensor of size 4]

>>> torch.rsqrt(a)

0.9020
0.8636
nan
nan
[torch.FloatTensor of size 4]
```

torch.sigmoid

```
torch.sigmoid(input, out=None) → Tensor
```

返回一个新张量，包含输入 `input` 张量每个元素的sigmoid值。

参数：

- input (Tensor) – 输入张量

- out (Tensor, optional) – 输出张量

例子:

```
>>> a = torch.randn(4)
>>> a

-0.4972
 1.3512
 0.1056
-0.2650
[torch.FloatTensor of size 4]

>>> torch.sigmoid(a)

 0.3782
 0.7943
 0.5264
 0.4341
[torch.FloatTensor of size 4]
```

torch.sign

```
torch.sign(input, out=None) → Tensor
```

符号函数: 返回一个新张量, 包含输入 `input` 张量每个元素的正负。

参数:

- input (Tensor) – 输入张量
- out (Tensor, optional) – 输出张量

例子:

```
>>> a = torch.randn(4)
>>> a

-0.6366
 0.2718
 0.4469
 1.3122
[torch.FloatTensor of size 4]

>>> torch.sign(a)

-1
 1
 1
 1
[torch.FloatTensor of size 4]
```

torch.sin

```
torch.sin(input, out=None) → Tensor
```

 v: latest ▼

返回一个新张量, 包含输入 `input` 张量每个元素的正弦。

参数:

- input (Tensor) – 输入张量
- out (Tensor, optional) – 输出张量

例子:

```
>>> a = torch.randn(4)
>>> a
-0.6366
 0.2718
 0.4469
 1.3122
[torch.FloatTensor of size 4]

>>> torch.sin(a)
-0.5944
 0.2684
 0.4322
 0.9667
[torch.FloatTensor of size 4]
```

torch.sinh

```
torch.sinh(input, out=None) → Tensor
```

返回一个新张量, 包含输入 `input` 张量每个元素的双曲正弦。

参数:

- input (Tensor) – 输入张量
- out (Tensor, optional) – 输出张量

例子:

```
>>> a = torch.randn(4)
>>> a
-0.6366
 0.2718
 0.4469
 1.3122
[torch.FloatTensor of size 4]

>>> torch.sinh(a)
-0.6804
 0.2751
 0.4619
 1.7225
[torch.FloatTensor of size 4]
```

torch.sqrt

```
torch.sqrt(input, out=None) → Tensor
```

 v: latest ▼

返回一个新张量，包含输入 `input` 张量每个元素的平方根。

参数：

- `input` (Tensor) – 输入张量
- `out` (Tensor, optional) – 输出张量

例子：

```
>>> a = torch.randn(4)
>>> a

 1.2290
 1.3409
-0.5662
-0.0899
[torch.FloatTensor of size 4]

>>> torch.sqrt(a)

 1.1086
 1.1580
      nan
      nan
[torch.FloatTensor of size 4]
```

torch.tan

```
torch.tan(input, out=None) → Tensor
```

返回一个新张量，包含输入 `input` 张量每个元素的正切。

参数：

- `input` (Tensor) – 输入张量
- `out` (Tensor, optional) – 输出张量

例子：

```
>>> a = torch.randn(4)
>>> a

-0.6366
 0.2718
 0.4469
 1.3122
[torch.FloatTensor of size 4]

>>> torch.tan(a)

-0.7392
 0.2786
 0.4792
 3.7801
[torch.FloatTensor of size 4]
```

torch.tanh

```
torch.tanh(input, out=None) → Tensor
```

返回一个新张量，包含输入 `input` 张量每个元素的双曲正切。

参数：

- input (Tensor) – 输入张量
- out (Tensor, optional) – 输出张量

例子：

```
>>> a = torch.randn(4)
>>> a
-0.6366
 0.2718
 0.4469
 1.3122
[torch.FloatTensor of size 4]

>>> torch.tanh(a)
-0.5625
 0.2653
 0.4193
 0.8648
[torch.FloatTensor of size 4]
```

torch.trunc

```
torch.trunc(input, out=None) → Tensor
```

返回一个新张量，包含输入 `input` 张量每个元素的截断值(标量x的截断值是最接近其的整数，其比x更接近零。简而言之，有符号数的小数部分被舍弃)。

参数：

- input (Tensor) – 输入张量
- out (Tensor, optional) – 输出张量

例子：

```
>>> a = torch.randn(4)
>>> a

-0.4972
 1.3512
 0.1056
-0.2650
[torch.FloatTensor of size 4]

>>> torch.trunc(a)

-0
 1
 0
-0
[torch.FloatTensor of size 4]
```

Reduction Ops

torch.cumprod

```
torch.cumprod(input, dim, out=None) → Tensor
```

返回输入沿指定维度的累积积。例如，如果输入是一个N元向量，则结果也是一个N元向量，第 i 个输出元素值为 $y_i = x_1 * x_2 * x_3 * \dots * x_i$

参数：

- input (Tensor) – 输入张量
- dim (int) – 累积积操作的维度
- out (Tensor, optional) – 结果张量

例子：

```

>>> a = torch.randn(10)
>>> a

 1.1148
 1.8423
 1.4143
-0.4403
 1.2859
-1.2514
-0.4748
 1.1735
-1.6332
-0.4272
[torch.FloatTensor of size 10]

>>> torch.cumprod(a, dim=0)

 1.1148
 2.0537
 2.9045
-1.2788
-1.6444
 2.0578
-0.9770
-1.1466
 1.8726
-0.8000
[torch.FloatTensor of size 10]

>>> a[5] = 0.0
>>> torch.cumprod(a, dim=0)

 1.1148
 2.0537
 2.9045
-1.2788
-1.6444
 0.0000
 0.0000
 0.0000
-0.0000
 0.0000
[torch.FloatTensor of size 10]

```

torch.cumsum

```
torch.cumsum(input, dim, out=None) → Tensor
```

返回输入沿指定维度的累积和。例如，如果输入是一个N元向量，则结果也是一个N元向量，第 i 个输出元素值为 $y_i = x_1 + x_2 + x_3 + \dots + x_i$

参数：

- input (Tensor) – 输入张量
- dim (int) – 累积和操作的维度
- out (Tensor, optional) – 结果张量

例子：

```
>>> a = torch.randn(10)
>>> a

-0.6039
-0.2214
-0.3705
-0.0169
 1.3415
-0.1230
 0.9719
 0.6081
-0.1286
 1.0947
[torch.FloatTensor of size 10]

>>> torch.cumsum(a, dim=0)

-0.6039
-0.8253
-1.1958
-1.2127
 0.1288
 0.0058
 0.9777
 1.5858
 1.4572
 2.5519
[torch.FloatTensor of size 10]
```

torch.dist

```
torch.dist(input, other, p=2, out=None) → Tensor
```

返回 $(\text{input} - \text{other})$ 的 p 范数。

参数：

- input (Tensor) – 输入张量
- other (Tensor) – 右侧输入张量
- p (float, optional) – 所计算的范数
- out (Tensor, optional) – 结果张量

例子：

```
>>> x = torch.randn(4)
>>> x

0.2505
-0.4571
-0.3733
0.7807
[torch.FloatTensor of size 4]

>>> y = torch.randn(4)
>>> y

0.7782
-0.5185
1.4106
-2.4063
[torch.FloatTensor of size 4]

>>> torch.dist(x, y, 3.5)
3.302832063224223
>>> torch.dist(x, y, 3)
3.3677282206393286
>>> torch.dist(x, y, 0)
inf
>>> torch.dist(x, y, 1)
5.560028076171875
```

torch.mean

```
torch.mean(input) → float
```

返回输入张量所有元素的均值。

参数：input (Tensor) – 输入张量

例子：

```
>>> a = torch.randn(1, 3)
>>> a

-0.2946 -0.9143 2.1809
[torch.FloatTensor of size 1x3]

>>> torch.mean(a)
0.32398951053619385
```

```
torch.mean(input, dim, out=None) → Tensor
```

返回输入张量给定维度 `dim` 上每行的均值。

输出形状与输入相同，除了给定维度上为1。

参数：

- input (Tensor) – 输入张量
- dim (int) – the dimension to reduce
- out (Tensor, optional) – 结果张量

例子:

```
>>> a = torch.randn(4, 4)
>>> a

-1.2738 -0.3058  0.1230 -1.9615
 0.8771 -0.5430 -0.9233  0.9879
 1.4107  0.0317 -0.6823  0.2255
-1.3854  0.4953 -0.2160  0.2435
[torch.FloatTensor of size 4x4]

>>> torch.mean(a, 1)

-0.8545
 0.0997
 0.2464
-0.2157
[torch.FloatTensor of size 4x1]
```

torch.median

```
torch.median(input, dim=-1, values=None, indices=None) -> (Tensor, LongTensor)
```

返回输入张量给定维度每行的中位数，同时返回一个包含中位数的索引的 `LongTensor`。

`dim` 值默认为输入张量的最后一维。输出形状与输入相同，除了给定维度上为1。

注意: 这个函数还没有在 `torch.cuda.Tensor` 中定义

参数:

- input (Tensor) – 输入张量
- dim (int) – 缩减的维度
- values (Tensor, optional) – 结果张量
- indices (Tensor, optional) – 返回的索引结果张量

```

>>> a
-0.6891 -0.6662
 0.2697  0.7412
 0.5254 -0.7402
 0.5528 -0.2399
[torch.FloatTensor of size 4x2]

>>> a = torch.randn(4, 5)
>>> a
 0.4056 -0.3372  1.0973 -2.4884  0.4334
 2.1336  0.3841  0.1404 -0.1821 -0.7646
-0.2403  1.3975 -2.0068  0.1298  0.0212
-1.5371 -0.7257 -0.4871 -0.2359 -1.1724
[torch.FloatTensor of size 4x5]

>>> torch.median(a, 1)
(
  0.4056
  0.1404
  0.0212
 -0.7257
[torch.FloatTensor of size 4x1]
,
  0
  2
  4
  1
[torch.LongTensor of size 4x1]
)

```

torch.mode

```
torch.mode(input, dim=-1, values=None, indices=None) -> (Tensor, LongTensor)
```

返回给定维 `dim` 上，每行的众数值。同时返回一个 `LongTensor`，包含众数值的索引。`dim` 值默认为输入张量的最后一维。

输出形状与输入相同，除了给定维度上为1。

注意：这个函数还没有在 `torch.cuda.Tensor` 中定义

参数：

- input (Tensor) – 输入张量
- dim (int) – 缩减的维度
- values (Tensor, optional) – 结果张量
- indices (Tensor, optional) – 返回的索引张量

例子：

```
>>> a
-0.6891 -0.6662
 0.2697  0.7412
 0.5254 -0.7402
 0.5528 -0.2399
[torch.FloatTensor of size 4x2]

>>> a = torch.randn(4, 5)
>>> a
 0.4056 -0.3372  1.0973 -2.4884  0.4334
 2.1336  0.3841  0.1404 -0.1821 -0.7646
-0.2403  1.3975 -2.0068  0.1298  0.0212
-1.5371 -0.7257 -0.4871 -0.2359 -1.1724
[torch.FloatTensor of size 4x5]

>>> torch.mode(a, 1)
(
-2.4884
-0.7646
-2.0068
-1.5371
[torch.FloatTensor of size 4x1]
,
 3
 4
 2
 0
[torch.LongTensor of size 4x1]
)
```

torch.norm

```
torch.norm(input, p=2) → float
```

返回输入张量 `input` 的p 范数。

参数：


- input (Tensor) – 输入张量
- p (float,optional) – 范数计算中的幂指数值

例子：

```
>>> a = torch.randn(1, 3)
>>> a
-0.4376 -0.5328  0.9547
[torch.FloatTensor of size 1x3]

>>> torch.norm(a, 3)
1.0338925067372466
```

```
torch.norm(input, p, dim, out=None) → Tensor
```

返回输入张量给定维 `dim` 上每行的p 范数。输出形状与输入相同，除了给定维度上为  [v: latest](#) ▼

参数：

- input (Tensor) – 输入张量
- p (float) – 范数计算中的幂指数值
- dim (int) – 缩减的维度
- out (Tensor, optional) – 结果张量

例子：

```
>>> a = torch.randn(4, 2)
>>> a

-0.6891 -0.6662
 0.2697  0.7412
 0.5254 -0.7402
 0.5528 -0.2399
[torch.FloatTensor of size 4x2]

>>> torch.norm(a, 2, 1)

 0.9585
 0.7888
 0.9077
 0.6026
[torch.FloatTensor of size 4x1]

>>> torch.norm(a, 0, 1)

 2
 2
 2
 2
[torch.FloatTensor of size 4x1]
```

torch.prod

```
torch.prod(input) → float
```

返回输入张量 `input` 所有元素的积。

参数：input (Tensor) – 输入张量

例子：

```
>>> a = torch.randn(1, 3)
>>> a

 0.6170  0.3546  0.0253
[torch.FloatTensor of size 1x3]

>>> torch.prod(a)
0.005537458061418483
```

```
torch.prod(input, dim, out=None) → Tensor
```

返回输入张量给定维度上每行的积。 输出形状与输入相同，除了给定维度上为1.

 v: latest ▼

参数：

- input (Tensor) – 输入张量
- dim (int) – 缩减的维度
- out (Tensor, optional) – 结果张量

例子：

```
>>> a = torch.randn(4, 2)
>>> a

 0.1598 -0.6884
-0.1831 -0.4412
-0.9925 -0.6244
-0.2416 -0.8080
[torch.FloatTensor of size 4x2]

>>> torch.prod(a, 1)

-0.1100
 0.0808
 0.6197
 0.1952
[torch.FloatTensor of size 4x1]
```

torch.std

```
torch.std(input) → float
```

返回输入张量 `input` 所有元素的标准差。

参数：- input (Tensor) – 输入张量

例子：

```
>>> a = torch.randn(1, 3)
>>> a

-1.3063  1.4182 -0.3061
[torch.FloatTensor of size 1x3]

>>> torch.std(a)
1.3782334731508061
```

```
torch.std(input, dim, out=None) → Tensor
```

返回输入张量给定维度上每行的标准差。输出形状与输入相同，除了给定维度上为1。

参数：

- input (Tensor) – 输入张量
- dim (int) – 缩减的维度
- out (Tensor, optional) – 结果张量

 v: latest ▼

例子：

```
>>> a = torch.randn(4, 4)
>>> a

 0.1889 -2.4856  0.0043  1.8169
-0.7701 -0.4682 -2.2410  0.4098
 0.1919 -1.1856 -1.0361  0.9085
 0.0173  1.0662  0.2143 -0.5576
[torch.FloatTensor of size 4x4]

>>> torch.std(a, dim=1)

 1.7756
 1.1025
 1.0045
 0.6725
[torch.FloatTensor of size 4x1]
```

torch.sum

```
torch.sum(input) → float
```

返回输入张量 `input` 所有元素的和。

输出形状与输入相同，除了给定维度上为1。

参数：

- input (Tensor) – 输入张量

例子：

```
>>> a = torch.randn(1, 3)
>>> a

 0.6170  0.3546  0.0253
[torch.FloatTensor of size 1x3]

>>> torch.sum(a)
0.9969287421554327
```

```
torch.sum(input, dim, out=None) → Tensor
```

返回输入张量给定维度上每行的和。输出形状与输入相同，除了给定维度上为1。

参数：

- input (Tensor) – 输入张量
- dim (int) – 缩减的维度
- out (Tensor, optional) – 结果张量

例子：

```
>>> a = torch.randn(4, 4)
>>> a

-0.4640  0.0609  0.1122  0.4784
-1.3063  1.6443  0.4714 -0.7396
-1.3561 -0.1959  1.0609 -1.9855
 2.6833  0.5746 -0.5709 -0.4430
[torch.FloatTensor of size 4x4]

>>> torch.sum(a, 1)

 0.1874
 0.0698
-2.4767
 2.2440
[torch.FloatTensor of size 4x1]
```

torch.var

```
torch.var(input) → float
```

返回输入张量所有元素的方差

输出形状与输入相同，除了给定维度上为1.

参数：

- input (Tensor) – 输入张量

例子：

```
>>> a = torch.randn(1, 3)
>>> a

-1.3063  1.4182 -0.3061
[torch.FloatTensor of size 1x3]

>>> torch.var(a)
1.899527506513334
```

```
torch.var(input, dim, out=None) → Tensor
```

返回输入张量给定维度上每行的方差。输出形状与输入相同，除了给定维度上为1.

参数：

- input (Tensor) – 输入张量
- dim (int) – the dimension to reduce
- out (Tensor, optional) – 结果张量 例子：

```
>>> a = torch.randn(4, 4)
>>> a

-1.2738 -0.3058  0.1230 -1.9615
 0.8771 -0.5430 -0.9233  0.9879
 1.4107  0.0317 -0.6823  0.2255
-1.3854  0.4953 -0.2160  0.2435
[torch.FloatTensor of size 4x4]

>>> torch.var(a, 1)

 0.8859
 0.9509
 0.7548
 0.6949
[torch.FloatTensor of size 4x1]
```

比较操作 Comparison Ops

torch.eq

```
torch.eq(input, other, out=None) → Tensor
```

比较元素相等性。第二个参数可为一个数或与第一个参数同类型形状的张量。

参数：

- input (Tensor) – 待比较张量
- other (Tensor or float) – 比较张量或数
- out (Tensor, optional) – 输出张量，须为 ByteTensor类型 or 与 `input` 同类型

返回值：一个 `torch.ByteTensor` 张量，包含了每个位置的比较结果(相等为1，不等为0)

返回类型： Tensor

例子：

```
>>> torch.eq(torch.Tensor([[1, 2], [3, 4]]), torch.Tensor([[1, 1], [4, 4]]))
 1  0
 0  1
[torch.ByteTensor of size 2x2]
```

torch.equal

```
torch.equal(tensor1, tensor2) → bool
```

如果两个张量有相同的形状和元素值，则返回 `True`，否则 `False`。

例子：


```
>>> torch.equal(torch.Tensor([1, 2]), torch.Tensor([1, 2]))
True
```

torch.ge

```
torch.ge(input, other, out=None) → Tensor
```

逐元素比较 `input` 和 `other`，即是否 $input \geq other$ 。

如果两个张量有相同的形状和元素值，则返回 `True`，否则 `False`。第二个参数可以为一个数或与第一个参数相同形状和类型的张量

参数:

- `input` (Tensor) – 待对比的张量
- `other` (Tensor or float) – 对比的张量或 `float` 值
- `out` (Tensor, optional) – 输出张量。必须为 `ByteTensor` 或者与第一个参数 `tensor` 相同类型。

返回值: 一个 `torch.ByteTensor` 张量，包含了每个位置的比较结果(是否 $input \geq other$)。返回类型: Tensor

例子:

```
>>> torch.ge(torch.Tensor([[1, 2], [3, 4]]), torch.Tensor([[1, 1], [4, 4]]))
 1  1
 0  1
[torch.ByteTensor of size 2x2]
```

torch.gt

```
torch.gt(input, other, out=None) → Tensor
```

逐元素比较 `input` 和 `other`，即是否 $input > other$ 如果两个张量有相同的形状和元素值，则返回 `True`，否则 `False`。第二个参数可以为一个数或与第一个参数相同形状和类型的张量

参数:

- `input` (Tensor) – 要对比的张量
- `other` (Tensor or float) – 要对比的张量或 `float` 值
- `out` (Tensor, optional) – 输出张量。必须为 `ByteTensor` 或者与第一个参数 `tensor` 相同类型。

返回值: 一个 `torch.ByteTensor` 张量，包含了每个位置的比较结果(是否 $input > other$) 返回类型: Tensor

例子:

```
>>> torch.gt(torch.Tensor([[1, 2], [3, 4]]), torch.Tensor([[1, 1], [4, 4]]))
0  1
0  0
[torch.ByteTensor of size 2x2]
```

torch.kthvalue

```
torch.kthvalue(input, k, dim=None, out=None) -> (Tensor, LongTensor)
```

取输入张量 `input` 指定维上第 `k` 个最小值。如果不指定 `dim`，则默认为 `input` 的最后一维。

返回一个元组 $(values, indices)$ ，其中 `indices` 是原始输入张量 `input` 中沿 `dim` 维的第 `k` 个最小值下标。

参数:

- `input` (Tensor) – 要对比的张量
- `k` (int) – 第 `k` 个最小值
- `dim` (int, optional) – 沿着此维进行排序
- `out` (tuple, optional) – 输出元组 (Tensor, LongTensor) 可选地给定作为 输出 buffers

例子:

```
>>> x = torch.arange(1, 6)
>>> x

1
2
3
4
5
[torch.FloatTensor of size 5]

>>> torch.kthvalue(x, 4)
(
  4
[torch.FloatTensor of size 1]
,
  3
[torch.LongTensor of size 1]
)
```

torch.le

```
torch.le(input, other, out=None) -> Tensor
```

逐元素比较 `input` 和 `other`，即是否 $input \leq other$ 第二个参数可以为一个数或与第一个参数相同形状和类型的张量

 v: latest ▼

参数:

- input (Tensor) – 要对比的张量
- other (Tensor or float) – 对比的张量或 `float` 值
- out (Tensor, optional) – 输出张量。必须为 `ByteTensor` 或者与第一个参数 `tensor` 相同类型。

返回值： 一个 `torch.ByteTensor` 张量，包含了每个位置的比较结果(是否 $\text{input} \geq \text{other}$)。 返回类型： Tensor

例子：

```
>>> torch.le(torch.Tensor([[1, 2], [3, 4]]), torch.Tensor([[1, 1], [4, 4]]))
 1  0
 1  1
[torch.ByteTensor of size 2x2]
```

torch.lt

```
torch.lt(input, other, out=None) → Tensor
```

逐元素比较 `input` 和 `other`， 即是否 $\text{input} < \text{other}$

第二个参数可以为一个数或与第一个参数相同形状和类型的张量

参数:

- input (Tensor) – 要对比的张量
- other (Tensor or float) – 对比的张量或 `float` 值
- out (Tensor, optional) – 输出张量。必须为 `ByteTensor` 或者与第一个参数 `tensor` 相同类型。

input: 一个 `torch.ByteTensor` 张量，包含了每个位置的比较结果(是否 $\text{tensor} \geq \text{other}$)。 返回类型： Tensor

例子：

```
>>> torch.lt(torch.Tensor([[1, 2], [3, 4]]), torch.Tensor([[1, 1], [4, 4]]))
 0  0
 1  0
[torch.ByteTensor of size 2x2]
```

torch.max

```
torch.max()
```

返回输入张量所有元素的最大值。

 v: latest ▼

参数:

- input (Tensor) – 输入张量

例子:

```
>>> a = torch.randn(1, 3)
>>> a

0.4729 -0.2266 -0.2085
[torch.FloatTensor of size 1x3]

>>> torch.max(a)
0.4729
```

```
torch.max(input, dim, max=None, max_indices=None) -> (Tensor, LongTensor)
```

返回输入张量给定维度上每行的最大值，并同时返回每个最大值的位置索引。

输出形状中，将 `dim` 维设定为1，其它与输入形状保持一致。

参数:

- input (Tensor) – 输入张量
- dim (int) – 指定的维度
- max (Tensor, optional) – 结果张量，包含给定维度上的最大值
- max_indices (LongTensor, optional) – 结果张量，包含给定维度上每个最大值的位置索引

例子:

```
>> a = torch.randn(4, 4)
>> a

0.0692  0.3142  1.2513 -0.5428
0.9288  0.8552 -0.2073  0.6409
1.0695 -0.0101 -2.4507 -1.2230
0.7426 -0.7666  0.4862 -0.6628
torch.FloatTensor of size 4x4]

>>> torch.max(a, 1)
(
  1.2513
  0.9288
  1.0695
  0.7426
[torch.FloatTensor of size 4x1]
,
  2
  0
  0
  0
[torch.LongTensor of size 4x1]
)
```

```
torch.max(input, other, out=None) -> Tensor
```

返回输入张量给定维度上每行的最大值，并同时返回每个最大值的位置索引。即， $out_i = \max(input_i, other_i)$

输出形状中，将 `dim` 维设定为1，其它与输入形状保持一致。

参数:

- input (Tensor) – 输入张量
- other (Tensor) – 输出张量
- out (Tensor, optional) – 结果张量

例子:

```
>>> a = torch.randn(4)
>>> a

 1.3869
 0.3912
-0.8634
-0.5468
[torch.FloatTensor of size 4]

>>> b = torch.randn(4)
>>> b

 1.0067
-0.8010
 0.6258
 0.3627
[torch.FloatTensor of size 4]

>>> torch.max(a, b)

 1.3869
 0.3912
 0.6258
 0.3627
[torch.FloatTensor of size 4]
```

torch.min

```
torch.min(input) → float
```

返回输入张量所有元素的最小值。

参数: input (Tensor) – 输入张量

例子:

```
>>> a = torch.randn(1, 3)
>>> a

0.4729 -0.2266 -0.2085
[torch.FloatTensor of size 1x3]

>>> torch.min(a)
-0.22663167119026184
```

```
torch.min(input, dim, min=None, min_indices=None) -> (Tensor, LongTensor)
```

返回输入张量给定维度上每行的最小值，并同时返回每个最小值的位置索引。

输出形状中，将 `dim` 维设定为1，其它与输入形状保持一致。

参数:

- input (Tensor) – 输入张量
- dim (int) – 指定的维度
- min (Tensor, optional) – 结果张量，包含给定维度上的最小值
- min_indices (LongTensor, optional) – 结果张量，包含给定维度上每个最小值的位置索引

例子:

```
>> a = torch.randn(4, 4)
>> a

0.0692  0.3142  1.2513 -0.5428
0.9288  0.8552 -0.2073  0.6409
1.0695 -0.0101 -2.4507 -1.2230
0.7426 -0.7666  0.4862 -0.6628
torch.FloatTensor of size 4x4]

>> torch.min(a, 1)

0.5428
0.2073
2.4507
0.7666
torch.FloatTensor of size 4x1]

3
2
2
1
torch.LongTensor of size 4x1]
```

```
torch.min(input, other, out=None) -> Tensor
```

`input` 中逐元素与 `other` 相应位置的元素对比，返回最小值到输出张量。即， $out_i = \min(tensor_i, other_i)$

两张量形状不需匹配，但元素数须相同。

 v: latest ▼

注意：当形状不匹配时，`input` 的形状作为返回张量的形状。

参数:

- input (Tensor) – 输入张量
- other (Tensor) – 第二个输入张量
- out (Tensor, optional) – 结果张量

例子:

```
>>> a = torch.randn(4)
>>> a

 1.3869
 0.3912
-0.8634
-0.5468
[torch.FloatTensor of size 4]

>>> b = torch.randn(4)
>>> b

 1.0067
-0.8010
 0.6258
 0.3627
[torch.FloatTensor of size 4]

>>> torch.min(a, b)

 1.0067
-0.8010
-0.8634
-0.5468
[torch.FloatTensor of size 4]
```

torch.ne

```
torch.ne(input, other, out=None) → Tensor
```

逐元素比较 `input` 和 `other`，即是否 $input \neq other$ 。第二个参数可以为一个数或与第一个参数相同形状和类型的张量

参数:

- input (Tensor) – 待对比的张量
- other (Tensor or float) – 对比的张量或 `float` 值
- out (Tensor, optional) – 输出张量。必须为 `ByteTensor` 或者与 `input` 相同类型。

返回值: 一个 `torch.ByteTensor` 张量, 包含了每个位置的比较结果 (如果 $tensor \neq other$ 为 `True`, 返回 1)。

返回类型: Tensor

例子:

```
>>> torch.ne(torch.Tensor([[1, 2], [3, 4]]), torch.Tensor([[1, 1], [4, 4]]))
0  1
1  0
[torch.ByteTensor of size 2x2]
```

torch.sort

```
torch.sort(input, dim=None, descending=False, out=None) -> (Tensor, LongTensor)
```

对输入张量 `input` 沿着指定维按升序排序。如果不给定 `dim`，则默认为输入的最后一维。如果指定参数 `descending` 为 `True`，则按降序排序

返回元组 (sorted_tensor, sorted_indices)，`sorted_indices` 为原始输入中的下标。

参数:

- input (Tensor) – 要对比的张量
- dim (int, optional) – 沿着此维排序
- descending (bool, optional) – 布尔值，控制升降排序
- out (tuple, optional) – 输出张量。必须为 `ByteTensor` 或者与第一个参数 `tensor` 相同类型。

例子:

```
>>> x = torch.randn(3, 4)
>>> sorted, indices = torch.sort(x)
>>> sorted

-1.6747  0.0610  0.1190  1.4137
-1.4782  0.7159  1.0341  1.3678
-0.3324 -0.0782  0.3518  0.4763
[torch.FloatTensor of size 3x4]

>>> indices

0  1  3  2
2  1  0  3
3  1  0  2
[torch.LongTensor of size 3x4]

>>> sorted, indices = torch.sort(x, 0)
>>> sorted

-1.6747 -0.0782 -1.4782 -0.3324
0.3518  0.0610  0.4763  0.1190
1.0341  0.7159  1.4137  1.3678
[torch.FloatTensor of size 3x4]

>>> indices

0  2  1  2
2  0  2  0
1  1  0  1
[torch.LongTensor of size 3x4]
```

torch.topk


```
torch.topk(input, k, dim=None, largest=True, sorted=True, out=None) -> (Tensor, LongTensor)
```

沿给定 `dim` 维度返回输入张量 `input` 中 `k` 个最大值。如果不指定 `dim`，则默认为 `input` 的最后一维。如果为 `largest` 为 `False`，则返回最小的 `k` 个值。

返回一个元组 $(values, indices)$ ，其中 `indices` 是原始输入张量 `input` 中元素下标。如果设定布尔值 `sorted` 为 `True`，将会确保返回的 `k` 个值被排序。

参数:

- `input` (Tensor) – 输入张量
- `k` (int) – “top-k”中的 `k`
- `dim` (int, optional) – 排序的维
- `largest` (bool, optional) – 布尔值，控制返回最大或最小值
- `sorted` (bool, optional) – 布尔值，控制返回值是否排序
- `out` (tuple, optional) – 可选输出张量 (Tensor, LongTensor) output buffers

```
>>> x = torch.arange(1, 6)
>>> x

1
2
3
4
5
[torch.FloatTensor of size 5]

>>> torch.topk(x, 3)
(
  5
  4
  3
[torch.FloatTensor of size 3]
,
  4
  3
  2
[torch.LongTensor of size 3]
)
>>> torch.topk(x, 3, 0, largest=False)
(
  1
  2
  3
[torch.FloatTensor of size 3]
,
  0
  1
  2
[torch.LongTensor of size 3]
)
```

其它操作 Other Operations

torch.cross

 v: latest ▼

```
torch.cross(input, other, dim=-1, out=None) -> Tensor
```

返回沿着维度 `dim` 上，两个张量 `input` 和 `other` 的向量积（叉积）。`input` 和 `other` 必须有相同的形状，且指定的 `dim` 维上size必须为 `3`。

如果不指定 `dim`，则默认为第一个尺度为 `3` 的维。

参数：

- `input` (Tensor) – 输入张量
- `other` (Tensor) – 第二个输入张量
- `dim` (int, optional) – 沿着此维进行叉积操作
- `out` (Tensor, optional) – 结果张量

例子：

```
>>> a = torch.randn(4, 3)
>>> a

-0.6652 -1.0116 -0.6857
 0.2286  0.4446 -0.5272
 0.0476  0.2321  1.9991
 0.6199  1.1924 -0.9397
[torch.FloatTensor of size 4x3]

>>> b = torch.randn(4, 3)
>>> b

-0.1042 -1.1156  0.1947
 0.9947  0.1149  0.4701
-1.0108  0.8319 -0.0750
 0.9045 -1.3754  1.0976
[torch.FloatTensor of size 4x3]

>>> torch.cross(a, b, dim=1)

-0.9619  0.2009  0.6367
 0.2696 -0.6318 -0.4160
-1.6805 -2.0171  0.2741
 0.0163 -1.5304 -1.9311
[torch.FloatTensor of size 4x3]

>>> torch.cross(a, b)

-0.9619  0.2009  0.6367
 0.2696 -0.6318 -0.4160
-1.6805 -2.0171  0.2741
 0.0163 -1.5304 -1.9311
[torch.FloatTensor of size 4x3]
```

torch.diag

```
torch.diag(input, diagonal=0, out=None) → Tensor
```

- 如果输入是一个向量(1D 张量)，则返回一个以 `input` 为对角线元素的2D方阵
- 如果输入是一个矩阵(2D 张量)，则返回一个包含 `input` 对角线元素的1D张量

参数 `diagonal` 指定对角线:

- `diagonal` = 0, 主对角线
- `diagonal` > 0, 主对角线之上
- `diagonal` < 0, 主对角线之下

参数:

- input (Tensor) – 输入张量
- diagonal (int, optional) – 指定对角线
- out (Tensor, optional) – 输出张量

例子:

- 取得以 `input` 为对角线的方阵:

```
>>> a = torch.randn(3)
>>> a

 1.0480
-2.3405
-1.1138
[torch.FloatTensor of size 3]

>>> torch.diag(a)

 1.0480  0.0000  0.0000
 0.0000 -2.3405  0.0000
 0.0000  0.0000 -1.1138
[torch.FloatTensor of size 3x3]

>>> torch.diag(a, 1)

 0.0000  1.0480  0.0000  0.0000
 0.0000  0.0000 -2.3405  0.0000
 0.0000  0.0000  0.0000 -1.1138
 0.0000  0.0000  0.0000  0.0000
[torch.FloatTensor of size 4x4]
```

- 取得给定矩阵第 `k` 个对角线:

```
>>> a = torch.randn(3, 3)
>>> a

-1.5328 -1.3210 -1.5204
 0.8596  0.0471 -0.2239
-0.6617  0.0146 -1.0817
[torch.FloatTensor of size 3x3]

>>> torch.diag(a, 0)

-1.5328
 0.0471
-1.0817
[torch.FloatTensor of size 3]

>>> torch.diag(a, 1)

-1.3210
-0.2239
[torch.FloatTensor of size 2]
```

torch.histc

```
torch.histc(input, bins=100, min=0, max=0, out=None) → Tensor
```

计算输入张量的直方图。以 `min` 和 `max` 为range边界，将其均分成 `bins` 个直条，然后将排序好的数据划分到各个直条(bins)中。如果 `min` 和 `max` 都为0,则利用数据中的最大最小值作为边界。

参数：

- input (Tensor) – 输入张量
- bins (int) – 直方图 bins(直条)的个数(默认100个)
- min (int) – range的下边界(包含)
- max (int) – range的上边界(包含)
- out (Tensor, optional) – 结果张量

返回：直方图 返回类型：张量

例子：

```
>>> torch.histc(torch.FloatTensor([1, 2, 1]), bins=4, min=0, max=3)
FloatTensor([0, 2, 1, 0])
```

torch.renorm

```
torch.renorm(input, p, dim, maxnorm, out=None) → Tensor
```

返回一个张量，包含规范化后的各个子张量，使得沿着 `dim` 维划分的各子张量的p范数小于 `maxnorm`。

注意 如果p范数的值小于 `maxnorm`，则当前子张量不需要修改。

注意: 更详细解释参考[torch7](#) 以及 [Hinton et al. 2012, p. 2](#)

参数：

- input (Tensor) – 输入张量
- p (float) – 范数的p
- dim (int) – 沿着此维切片，得到张量子集
- maxnorm (float) – 每个子张量的范数的最大值
- out (Tensor, optional) – 结果张量

例子：

```
>>> x = torch.ones(3, 3)
>>> x[1].fill_(2)
>>> x[2].fill_(3)
>>> x

 1  1  1
 2  2  2
 3  3  3
[torch.FloatTensor of size 3x3]

>>> torch.renorm(x, 1, 0, 5)

1.0000  1.0000  1.0000
1.6667  1.6667  1.6667
1.6667  1.6667  1.6667
[torch.FloatTensor of size 3x3]
```

torch.trace

```
torch.trace(input) → float
```

返回输入2维矩阵对角线元素的和(迹)

例子：

```
>>> x = torch.arange(1, 10).view(3, 3)
>>> x

 1  2  3
 4  5  6
 7  8  9
[torch.FloatTensor of size 3x3]

>>> torch.trace(x)
15.0
```

torch.tril

```
torch.tril(input, k=0, out=None) → Tensor
```

返回一个张量 `out`，包含输入矩阵(2D张量)的下三角部分，`out` 其余部分被设为 `0`。这里所说的下三角部分为矩阵指定对角线 `diagonal` 之上的元素。

参数 `k` 控制对角线: - `k` = 0, 主对角线 - `k` > 0, 主对角线之上 - `k` < 0, 主对角线之下

参数：

- input (Tensor) – 输入张量
- k (int, optional) – 指定对角线
- out (Tensor, optional) – 输出张量

例子：

```

>>> a = torch.randn(3,3)
>>> a

 1.3225  1.7304  1.4573
-0.3052 -0.3111 -0.1809
 1.2469  0.0064 -1.6250
[torch.FloatTensor of size 3x3]

>>> torch.tril(a)

 1.3225  0.0000  0.0000
-0.3052 -0.3111  0.0000
 1.2469  0.0064 -1.6250
[torch.FloatTensor of size 3x3]

>>> torch.tril(a, k=1)

 1.3225  1.7304  0.0000
-0.3052 -0.3111 -0.1809
 1.2469  0.0064 -1.6250
[torch.FloatTensor of size 3x3]

>>> torch.tril(a, k=-1)

 0.0000  0.0000  0.0000
-0.3052  0.0000  0.0000
 1.2469  0.0064  0.0000
[torch.FloatTensor of size 3x3]

```

torch.triu

```
torch.triu(input, k=0, out=None) → Tensor
```

返回一个张量，包含输入矩阵(2D张量)的上三角部分，其余部分被设为 `0`。这里所说的上三角部分为矩阵指定对角线 `diagonal` 之上的元素。

参数 `k` 控制对角线: - `k` = 0, 主对角线 - `k` > 0, 主对角线之上 - `k` < 0, 主对角线之下

参数:

- input (Tensor) – 输入张量
- k (int, optional) – 指定对角线
- out (Tensor, optional) – 输出张量

例子:

```

>>> a = torch.randn(3,3)
>>> a

 1.3225  1.7304  1.4573
-0.3052 -0.3111 -0.1809
 1.2469  0.0064 -1.6250
[torch.FloatTensor of size 3x3]

>>> torch.triu(a)

 1.3225  1.7304  1.4573
 0.0000 -0.3111 -0.1809
 0.0000  0.0000 -1.6250
[torch.FloatTensor of size 3x3]

>>> torch.triu(a, k=1)

 0.0000  1.7304  1.4573
 0.0000  0.0000 -0.1809
 0.0000  0.0000  0.0000
[torch.FloatTensor of size 3x3]

>>> torch.triu(a, k=-1)

 1.3225  1.7304  1.4573
-0.3052 -0.3111 -0.1809
 0.0000  0.0064 -1.6250
[torch.FloatTensor of size 3x3]

```

BLAS and LAPACK Operations

torch.addbmm

```
torch.addbmm(beta=1, mat, alpha=1, batch1, batch2, out=None) → Tensor
```

对两个批 `batch1` 和 `batch2` 内存储的矩阵进行批矩阵乘操作，附带reduced add 步骤(所有矩阵乘结果沿着第一维相加)。矩阵 `mat` 加到最终结果。`batch1` 和 `batch2` 都为包含相同数量矩阵的3维张量。如果 `batch1` 是形为 $b \times n \times m$ 的张量，`batch2` 是形为 $b \times m \times p$ 的张量，则 `out` 和 `mat` 的形状都是 $n \times p$ ，即

$$res = (beta * M) + (alpha * sum(batch1_i @ batch2_i, i = 0, b))$$

对类型为 *FloatTensor* 或 *DoubleTensor* 的输入，`alpha` and `beta` 必须为实数，否则两个参数须为整数。

参数：

- beta (Number, optional) – 用于 `mat` 的乘子
- mat (Tensor) – 相加矩阵
- alpha (Number, optional) – 用于 `batch1 @ batch2` 的乘子
- batch1 (Tensor) – 第一批相乘矩阵
- batch2 (Tensor) – 第二批相乘矩阵
- out (Tensor, optional) – 输出张量

例子:

```
>>> M = torch.randn(3, 5)
>>> batch1 = torch.randn(10, 3, 4)
>>> batch2 = torch.randn(10, 4, 5)
>>> torch.addbmm(M, batch1, batch2)

-3.1162  11.0071   7.3102   0.1824  -7.6892
 1.8265   6.0739   0.4589  -0.5641  -5.4283
-9.3387  -0.1794  -1.2318  -6.8841  -4.7239
[torch.FloatTensor of size 3x5]
```

torch.addmm

```
torch.addmm(beta=1, mat, alpha=1, mat1, mat2, out=None) → Tensor
```

对矩阵 `mat1` 和 `mat2` 进行矩阵乘操作。矩阵 `mat` 加到最终结果。如果 `mat1` 是一个 $n \times m$ 张量, `mat2` 是一个 $m \times p$ 张量, 那么 `out` 和 `mat` 的形状为 $n \times p$ 。 `alpha` 和 `beta` 分别是两个矩阵 `mat1@mat2` 和 `mat` 的比例因子, 即, $out = (beta * M) + (alpha * mat1@mat2)$

对类型为 *FloatTensor* 或 *DoubleTensor* 的输入, `beta` and `alpha` 必须为实数, 否则两个参数须为整数。

参数：

- `beta` (Number, optional) – 用于 `mat` 的乘子
- `mat` (Tensor) – 相加矩阵
- `alpha` (Number, optional) – 用于 `mat1@mat2` 的乘子
- `mat1` (Tensor) – 第一个相乘矩阵
- `mat2` (Tensor) – 第二个相乘矩阵
- `out` (Tensor, optional) – 输出张量

```
>>> M = torch.randn(2, 3)
>>> mat1 = torch.randn(2, 3)
>>> mat2 = torch.randn(3, 3)
>>> torch.addmm(M, mat1, mat2)

-0.4095 -1.9703  1.3561
 5.7674 -4.9760  2.7378
[torch.FloatTensor of size 2x3]
```

torch.addmv

```
torch.addmv(beta=1, tensor, alpha=1, mat, vec, out=None) → Tensor
```

对矩阵 `mat` 和向量 `vec` 对进行相乘操作。向量 `tensor` 加到最终结果。如果 `mat` 是一个 $n \times m$ 维矩阵, `vec` 是一个 m 维向量, 那么 `out` 和 `mat` 的为 n 元向量。可选参数 `alpha` 和 `beta` 分别是 `mat * vec` 和 `mat` 的比例因子, 即, $out = (beta * tensor) + (alpha * (mat@vec))$

对类型为_FloatTensor_或_DoubleTensor_的输入, `alpha` and `beta` 必须为实数, 否则两个参数须为整数。

参数：

- `beta` (Number, optional) - 用于 `mat` 的乘子
- `mat` (Tensor) - 相加矩阵
- `alpha` (Number, optional) - 用于 `mat1@vec` 的乘子
- `mat` (Tensor) - 相乘矩阵
- `vec` (Tensor) - 相乘向量
- `out` (Tensor, optional) - 输出张量

```
>>> M = torch.randn(2)
>>> mat = torch.randn(2, 3)
>>> vec = torch.randn(3)
>>> torch.addmv(M, mat, vec)

-2.0939
-2.2950
[torch.FloatTensor of size 2]
```

torch.addr

```
torch.addr(beta=1, mat, alpha=1, vec1, vec2, out=None) → Tensor
```

对向量 `vec1` 和 `vec2` 对进行张量积操作。矩阵 `mat` 加到最终结果。如果 `vec1` 是一个 n 维向量, `vec2` 是一个 m 维向量, 那么矩阵 `mat` 的形状须为 $n \times m$ 。可选参数 `beta` 和 `alpha` 分别是两个矩阵 `mat` 和 `vec1@vec2` 的比例因子, 即,

$$resi = (beta * Mi) + (alpha * batch1i \times batch2i)$$

对类型为_FloatTensor_或_DoubleTensor_的输入, `alpha` and `beta` 必须为实数, 否则两个参数须为整数。

参数：

- `beta` (Number, optional) - 用于 `mat` 的乘子
- `mat` (Tensor) - 相加矩阵
- `alpha` (Number, optional) - 用于两向量 `vec1`, `vec2` 外积的乘子
- `vec1` (Tensor) - 第一个相乘向量
- `vec2` (Tensor) - 第二个相乘向量
- `out` (Tensor, optional) - 输出张量

```
>>> vec1 = torch.arange(1, 4)
>>> vec2 = torch.arange(1, 3)
>>> M = torch.zeros(3, 2)
>>> torch.addr(M, vec1, vec2)
 1  2
 2  4
 3  6
[torch.FloatTensor of size 3x2]
```

torch.baddbmm

```
torch.baddbmm(beta=1, mat, alpha=1, batch1, batch2, out=None) → Tensor
```

对两个批 `batch1` 和 `batch2` 内存储的矩阵进行批矩阵乘操作，矩阵 `mat` 加到最终结果。

`batch1` 和 `batch2` 都为包含相同数量矩阵的3维张量。如果 `batch1` 是形为 $b \times n \times m$ 的张量，`batch2` 是形为 $b \times m \times p$ 的张量，则 `out` 和 `mat` 的形状都是 $n \times p$ ，即

$$resi = (\text{beta} * M_i) + (\text{alpha} * \text{batch1}_i \times \text{batch2}_i)$$

对类型为 `FloatTensor` 或 `DoubleTensor` 的输入，`alpha` and `beta` 必须为实数，否则两个参数须为整数。

参数：

- `beta` (Number, optional) – 用于 `mat` 的乘子
- `mat` (Tensor) – 相加矩阵
- `alpha` (Number, optional) – 用于 `batch1@batch2` 的乘子
- `batch1` (Tensor) – 第一批相乘矩阵
- `batch2` (Tensor) – 第二批相乘矩阵
- `out` (Tensor, optional) – 输出张量


```
>>> M = torch.randn(10, 3, 5)
>>> batch1 = torch.randn(10, 3, 4)
>>> batch2 = torch.randn(10, 4, 5)
>>> torch.baddbmm(M, batch1, batch2).size()
torch.Size([10, 3, 5])
```

torch.bmm

```
torch.bmm(batch1, batch2, out=None) → Tensor
```

对存储在两个批 `batch1` 和 `batch2` 内的矩阵进行批矩阵乘操作。`batch1` 和 `batch2` 都为包含相同数量矩阵的3维张量。如果 `batch1` 是形为 $b \times n \times m$ 的张量，`batch2` 是形为 $b \times m \times p$ 的张量，则 `out` 和 `mat` 的形状都是 $n \times p$ ，即

$$res = (\text{beta} * M) + (\text{alpha} * \text{sum}(\text{batch1}_i @ \text{batch2}_i, i = 0, b))$$

对类型为 `FloatTensor` 或 `DoubleTensor` 的输入，`alpha` and `beta` 必须为实数，否则两  `v:latest` 须为整数。

参数:

- batch1 (Tensor) – 第一批相乘矩阵
- batch2 (Tensor) – 第二批相乘矩阵
- out (Tensor, optional) – 输出张量

```
>>> batch1 = torch.randn(10, 3, 4)
>>> batch2 = torch.randn(10, 4, 5)
>>> res = torch.bmm(batch1, batch2)
>>> res.size()
torch.Size([10, 3, 5])
```

torch.btrifact

```
torch.btrifact(A, info=None) → Tensor, IntTensor
```

返回一个元组，包含LU分解和 `pivots`。可选参数 `info` 决定是否对每个minibatch样本进行分解。`info` are from dgetrf and a non-zero value indicates an error occurred. 如果用CUDA的话，这个值来自于CUBLAS，否则来自LAPACK。

参数: A (Tensor) – 待分解张量

```
>>> A = torch.randn(2, 3, 3)
>>> A_LU = A.btrifact()
```

torch.btrisolve

```
torch.btrisolve(b, LU_data, LU_pivots) → Tensor
```

返回线性方程组 $Ax = b$ 的LU解。

参数:

- b (Tensor) – RHS 张量.
- LU_data (Tensor) – Pivoted LU factorization of A from btrifact.
- LU_pivots (IntTensor) – LU 分解的Pivots.

例子:

```
>>> A = torch.randn(2, 3, 3)
>>> b = torch.randn(2, 3)
>>> A_LU = torch.btrifact(A)
>>> x = b.btrisolve(*A_LU)
>>> torch.norm(A.bmm(x.unsqueeze(2)) - b)
6.664001874625056e-08
```

 v: latest ▼

torch.dot

```
torch.dot(tensor1, tensor2) → float
```

计算两个张量的点乘(内乘),两个张量都为1-D 向量.

例子:

```
>>> torch.dot(torch.Tensor([2, 3]), torch.Tensor([2, 1]))
7.0
```

torch.eig

```
torch.eig(a, eigenvectors=False, out=None) -> (Tensor, Tensor)
```

计算实方阵 `a` 的特征值和特征向量

参数:

- `a` (Tensor) – 方阵, 待计算其特征值和特征向量
- `eigenvectors` (bool) – 布尔值, 如果 `True`, 则同时计算特征值和特征向量, 否则只计算特征值。
- `out` (tuple, optional) – 输出元组

返回值: 元组, 包括:

- `e` (Tensor): `a` 的右特征向量
- `v` (Tensor): 如果 `eigenvectors` 为 `True`, 则为包含特征向量的张量; 否则为空张量

返回值类型: (Tensor, Tensor)

torch.gels


```
torch.gels(B, A, out=None) → Tensor
```

对形如 $m \times n$ 的满秩矩阵 `a` 计算其最小二乘和最小范数问题的解。如果 $m \geq n$, `gels` 对最小二乘问题进行求解, 即:

$$\text{minimize} \quad \|AX - B\|_F$$

如果 $m < n$, `gels` 求解最小范数问题, 即:

$$\text{minimize} \quad \|X\|_F \quad \text{subject to} \quad a \quad bAX = B$$

返回矩阵 X 的前 n 行包含解。余下的行包含以下残差信息: 相应列从第 n 行开始计算的  `v: latest` 欧式距离。

注意：返回矩阵总是被转置，无论输入矩阵的原始布局如何，总会被转置；即，总是有 stride (1, m) 而不是 (m, 1).

参数：

- B (Tensor) – 矩阵B
- A (Tensor) – $m \times n$ 矩阵
- out (tuple, optional) – 输出元组

返回值： 元组，包括：

- X (Tensor): 最小二乘解
- qr (Tensor): QR 分解的细节

返回值类型： (Tensor, Tensor)

例子：

```
>>> A = torch.Tensor([[1, 1, 1],
...                   [2, 3, 4],
...                   [3, 5, 2],
...                   [4, 2, 5],
...                   [5, 4, 3]])
>>> B = torch.Tensor([[ -10, -3],
...                   [ 12, 14],
...                   [ 14, 12],
...                   [ 16, 16],
...                   [ 18, 16]])
>>> X, _ = torch.gels(B, A)
>>> X
2.0000  1.0000
1.0000  1.0000
1.0000  2.0000
[torch.FloatTensor of size 3x2]
```

torch.geqrf

```
torch.geqrf(input, out=None) -> (Tensor, Tensor)
```

这是一个直接调用LAPACK的底层函数。一般使用 `torch.qr()`

计算输入的QR 分解，但是并不会分别创建Q,R两个矩阵，而是直接调用LAPACK 函数 Rather, this directly calls the underlying LAPACK function ?geqrf which produces a sequence of ‘elementary reflectors’.

参考 [LAPACK文档](#) 获取更详细信息。

参数：

- input (Tensor) – 输入矩阵

- out (tuple, optional) – 元组，包含输出张量 (Tensor, Tensor)

torch.ger

```
torch.ger(vec1, vec2, out=None) → Tensor
```

计算两向量 `vec1`, `vec2` 的张量积。如果 `vec1` 的长度为 `n`, `vec2` 长度为 `m`, 则输出 `out` 应为形如 $n \times m$ 的矩阵。

参数:

- vec1 (Tensor) – 1D 输入向量
- vec2 (Tensor) – 1D 输入向量
- out (tuple, optional) – 输出张量

例子:

```
>>> v1 = torch.arange(1, 5)
>>> v2 = torch.arange(1, 4)
>>> torch.ger(v1, v2)

 1  2  3
 2  4  6
 3  6  9
 4  8 12
[torch.FloatTensor of size 4x3]
```

torch.gesv

```
torch.gesv(B, A, out=None) -> (Tensor, Tensor)
```

$X, LU = \text{torch.gesv}(B, A)$, 返回线性方程组 $AX = B$ 的解。

LU 包含两个矩阵 L, U。A 须为非奇异方阵，如果 A 是一个 $m \times m$ 矩阵，B 是 $m \times k$ 矩阵，则 LU 是 $m \times m$ 矩阵，X 为 $m \times k$ 矩阵

参数:

- B (Tensor) – $m \times k$ 矩阵
- A (Tensor) – $m \times m$ 矩阵
- out (Tensor, optional) – 可选地输出矩阵 X

例子:

```
>>> A = torch.Tensor([[6.80, -2.11, 5.66, 5.97, 8.23],
...                    [-6.05, -3.30, 5.36, -4.44, 1.08],
...                    [-0.45, 2.58, -2.70, 0.27, 9.04],
...                    [8.32, 2.71, 4.35, -7.17, 2.14],
...                    [-9.67, -5.14, -7.26, 6.08, -6.87]]).t()
>>> B = torch.Tensor([[4.02, 6.19, -8.22, -7.57, -3.03],
...                    [-1.56, 4.00, -8.67, 1.75, 2.86],
...                    [9.81, -4.09, -4.57, -8.61, 8.99]]).t()
>>> X, LU = torch.gesv(B, A)
>>> torch.dist(B, torch.mm(A, X))
9.250057093890353e-06
```

torch.inverse

```
torch.inverse(input, out=None) → Tensor
```

对方阵输入 `input` 取逆。

注意： Irrespective of the original strides, the returned matrix will be transposed, i.e. with strides (1, m) instead of (m, 1)

参数：

- input (Tensor) – 输入2维张量
- out (Tensor, optional) – 输出张量

例子:

```

>>> x = torch.rand(10, 10)
>>> x

0.7800  0.2267  0.7855  0.9479  0.5914  0.7119  0.4437  0.9131  0.1289  0.1982
0.0045  0.0425  0.2229  0.4626  0.6210  0.0207  0.6338  0.7067  0.6381  0.8196
0.8350  0.7810  0.8526  0.9364  0.7504  0.2737  0.0694  0.5899  0.8516  0.3883
0.6280  0.6016  0.5357  0.2936  0.7827  0.2772  0.0744  0.2627  0.6326  0.9153
0.7897  0.0226  0.3102  0.0198  0.9415  0.9896  0.3528  0.9397  0.2074  0.6980
0.5235  0.6119  0.6522  0.3399  0.3205  0.5555  0.8454  0.3792  0.4927  0.6086
0.1048  0.0328  0.5734  0.6318  0.9802  0.4458  0.0979  0.3320  0.3701  0.0909
0.2616  0.3485  0.4370  0.5620  0.5291  0.8295  0.7693  0.1807  0.0650  0.8497
0.1655  0.2192  0.6913  0.0093  0.0178  0.3064  0.6715  0.5101  0.2561  0.3396
0.4370  0.4695  0.8333  0.1180  0.4266  0.4161  0.0699  0.4263  0.8865  0.2578
[torch.FloatTensor of size 10x10]

>>> x = torch.rand(10, 10)
>>> y = torch.inverse(x)
>>> z = torch.mm(x, y)
>>> z

1.0000  0.0000  0.0000 -0.0000  0.0000  0.0000  0.0000  0.0000 -0.0000 -0.0000
0.0000  1.0000 -0.0000  0.0000  0.0000  0.0000 -0.0000 -0.0000 -0.0000 -0.0000
0.0000  0.0000  1.0000 -0.0000 -0.0000  0.0000  0.0000  0.0000 -0.0000 -0.0000
0.0000  0.0000  0.0000  1.0000  0.0000  0.0000  0.0000 -0.0000 -0.0000  0.0000
0.0000  0.0000 -0.0000 -0.0000  1.0000  0.0000  0.0000 -0.0000 -0.0000 -0.0000
0.0000  0.0000  0.0000 -0.0000  0.0000  1.0000 -0.0000 -0.0000 -0.0000 -0.0000
0.0000  0.0000  0.0000 -0.0000  0.0000  0.0000  1.0000  0.0000 -0.0000  0.0000
0.0000  0.0000 -0.0000 -0.0000  0.0000  0.0000 -0.0000  1.0000 -0.0000  0.0000
-0.0000  0.0000 -0.0000 -0.0000  0.0000  0.0000 -0.0000 -0.0000  1.0000 -0.0000
-0.0000  0.0000 -0.0000 -0.0000 -0.0000  0.0000 -0.0000 -0.0000  0.0000  1.0000
[torch.FloatTensor of size 10x10]

>>> torch.max(torch.abs(z - torch.eye(10))) # Max nonzero
5.096662789583206e-07

```

torch.mm

```
torch.mm(mat1, mat2, out=None) → Tensor
```

对矩阵 `mat1` 和 `mat2` 进行相乘。如果 `mat1` 是一个 $n \times m$ 张量, `mat2` 是一个 $m \times p$ 张量, 将会输出一个 $n \times p$ 张量 `out`。

参数：

- `mat1` (Tensor) – 第一个相乘矩阵
- `mat2` (Tensor) – 第二个相乘矩阵
- `out` (Tensor, optional) – 输出张量

例子:

```

>>> mat1 = torch.randn(2, 3)
>>> mat2 = torch.randn(3, 3)
>>> torch.mm(mat1, mat2)
0.0519 -0.3304  1.2232
4.3910 -5.1498  2.7571
[torch.FloatTensor of size 2x3]

```

torch.mv


```
torch.mv(mat, vec, out=None) → Tensor
```

对矩阵 `mat` 和向量 `vec` 进行相乘。如果 `mat` 是一个 $n \times m$ 张量, `vec` 是一个 m 元 1 维张量, 将会输出一个 n 元 1 维张量。

参数：

- `mat` (Tensor) – 相乘矩阵
- `vec` (Tensor) – 相乘向量
- `out` (Tensor, optional) – 输出张量

例子:

```
>>> mat = torch.randn(2, 3)
>>> vec = torch.randn(3)
>>> torch.mv(mat, vec)
-2.0939
-2.2950
[torch.FloatTensor of size 2]
```

torch.orgqr

```
torch.orgqr()
```

torch.ormqr

```
torch.ormqr()
```

torch.potrf

```
torch.potrf()
```

torch.potri

```
torch.potri()
```

torch.potrs

```
torch.potrs()
```

torch.pstrf

```
torch.pstrf()
```

 v: latest ▼

torch.qr

```
torch.qr(input, out=None) -> (Tensor, Tensor)
```

计算输入矩阵的QR分解：返回两个矩阵 q, r ，使得 $x = q * r$ ，这里 q 是一个半正交矩阵与 r 是一个上三角矩阵

本函数返回一个thin(reduced)QR分解。

注意 如果输入很大，可能可能会丢失精度。

注意 本函数依赖于你的LAPACK实现，虽然总能返回一个合法的分解，但不同平台可能得到不同的结果。

Irrespective of the original strides, the returned matrix q will be transposed, i.e. with strides (1, m) instead of (m, 1).

参数：

- input (Tensor) – 输入的2维张量
- out (tuple, optional) – 输出元组 `tuple`，包含Q和R

例子：

```
>>> a = torch.Tensor([[12, -51, 4], [6, 167, -68], [-4, 24, -41]])
>>> q, r = torch.qr(a)
>>> q

-0.8571  0.3943  0.3314
-0.4286 -0.9029 -0.0343
 0.2857 -0.1714  0.9429
[torch.FloatTensor of size 3x3]

>>> r

-14.0000 -21.0000  14.0000
  0.0000 -175.0000  70.0000
  0.0000  0.0000 -35.0000
[torch.FloatTensor of size 3x3]

>>> torch.mm(q, r).round()

 12  -51   4
  6  167 -68
 -4   24 -41
[torch.FloatTensor of size 3x3]

>>> torch.mm(q.t(), q).round()

 1  0  0
 0  1  0
 0  0  1
[torch.FloatTensor of size 3x3]
```

torch.svd

 v: latest ▼

```
torch.svd(input, some=True, out=None) -> (Tensor, Tensor, Tensor)
```

$U, S, V = \text{torch.svd}(A)$ 。返回对形如 $n \times m$ 的实矩阵 A 进行奇异值分解的结果，使得 $A = USV^*$ 。 U 形状为 $n \times n$ ， S 形状为 $n \times m$ ， V 形状为 $m \times m$

`some` 代表了需要计算的奇异值数目。如果 `some=True`， it computes some and `some=False` computes all.

Irrespective of the original strides, the returned matrix U will be transposed, i.e. with strides $(1, n)$ instead of $(n, 1)$.

参数：

- input (Tensor) – 输入的2维张量
- some (bool, optional) – 布尔值，控制需计算的奇异值数目
- out (tuple, optional) – 结果 `tuple`

例子：

```
>>> a = torch.Tensor([[8.79, 6.11, -9.15, 9.57, -3.49, 9.84],
...                    [9.93, 6.91, -7.93, 1.64, 4.02, 0.15],
...                    [9.83, 5.04, 4.86, 8.83, 9.80, -8.99],
...                    [5.45, -0.27, 4.85, 0.74, 10.00, -6.02],
...                    [3.16, 7.98, 3.01, 5.80, 4.27, -5.31]]).t()
>>> a
      8.7900  9.9300  9.8300  5.4500  3.1600
      6.1100  6.9100  5.0400 -0.2700  7.9800
     -9.1500 -7.9300  4.8600  4.8500  3.0100
      9.5700  1.6400  8.8300  0.7400  5.8000
     -3.4900  4.0200  9.8000 10.0000  4.2700
      9.8400  0.1500 -8.9900 -6.0200 -5.3100
[torch.FloatTensor of size 6x5]

>>> u, s, v = torch.svd(a)
>>> u
-0.5911  0.2632  0.3554  0.3143  0.2299
-0.3976  0.2438 -0.2224 -0.7535 -0.3636
-0.0335 -0.6003 -0.4508  0.2334 -0.3055
-0.4297  0.2362 -0.6859  0.3319  0.1649
-0.4697 -0.3509  0.3874  0.1587 -0.5183
 0.2934  0.5763 -0.0209  0.3791 -0.6526
[torch.FloatTensor of size 6x5]

>>> s
27.4687
22.6432
 8.5584
 5.9857
 2.0149
[torch.FloatTensor of size 5]

>>> v
-0.2514  0.8148 -0.2606  0.3967 -0.2180
-0.3968  0.3587  0.7008 -0.4507  0.1402
-0.6922 -0.2489 -0.2208  0.2513  0.5891
-0.3662 -0.3686  0.3859  0.4342 -0.6265
-0.4076 -0.0980 -0.4932 -0.6227 -0.4396
[torch.FloatTensor of size 5x5]

>>> torch.dist(a, torch.mm(torch.mm(u, torch.diag(s)), v.t()))
8.934150226306685e-06
```

torch.symeig

```
torch.symeig(input, eigenvectors=False, upper=True, out=None) -> (Tensor, Tensor)
```

$e, V = \text{torch.symeig}(\text{input})$ 返回实对称矩阵 `input` 的特征值和特征向量。

input 和 V 为 $m \times m$ 矩阵, e 是一个 m 维向量。此函数计算 `input` 的所有特征值(和特征向量), 使得 $\text{input} = V \text{diag}(e) V'$

布尔值参数 `eigenvectors` 规定是否只计算特征向量。如果为 `False`, 则只计算特征值; 若设为 `True`, 则两者都会计算。因为输入矩阵 input 是对称的, 所以默认只需要上三角矩阵。如果参数 `upper` 为 `False`, 下三角矩阵部分也被利用。

注意: 不管原来 Irrespective of the original strides, the returned matrix V will be transposed, i.e. with strides (1, m) instead of (m, 1)

参数:

- `input` (Tensor) – 输入对称矩阵
- `eigenvectors` (boolean, optional) – 布尔值 (可选), 控制是否计算特征向量
- `upper` (boolean, optional) – 布尔值 (可选), 控制是否考虑上三角或下三角区域
- `out` (tuple, optional) – 输出元组(Tensor, Tensor)

例子:

```
>>> a = torch.Tensor([[ 1.96,  0.00,  0.00,  0.00,  0.00],
...                   [-6.49,  3.80,  0.00,  0.00,  0.00],
...                   [-0.47, -6.39,  4.17,  0.00,  0.00],
...                   [-7.20,  1.50, -1.51,  5.70,  0.00],
...                   [-0.65, -6.34,  2.67,  1.80, -7.10]]).t()

>>> e, v = torch.symeig(a, eigenvectors=True)
>>> e

-11.0656
-6.2287
 0.8640
 8.8655
16.0948
[torch.FloatTensor of size 5]

>>> v

-0.2981 -0.6075  0.4026 -0.3745  0.4896
-0.5078 -0.2880 -0.4066 -0.3572 -0.6053
-0.0816 -0.3843 -0.6600  0.5008  0.3991
-0.0036 -0.4467  0.4553  0.6204 -0.4564
-0.8041  0.4480  0.1725  0.3108  0.1622
[torch.FloatTensor of size 5x5]
```

torch.trtrs

```
torch.trtrs()
```

 v: latest ▼

