# Torch

## Backward

torch.nn.Parameter和from torch.autograd import Variable定义的变量默认有梯度。直接torch.zero的变量没有梯度

torch.autograd.Variable是Autograd的核心类，它封装了Tensor，并整合了反向传播的相关实现

Varibale包含三个属性：

* data：存储了Tensor，是本体的数据
* grad：保存了data的梯度，本事是个Variable而非Tensor，与data形状一致
* grad\_fn：指向Function对象，用于反向传播的梯度计算之用

[回到顶部](https://www.cnblogs.com/hellcat/p/8439055.html" \l "_labelTop)

## **data**

|  |  |
| --- | --- |
| 1  2  3  4  5 | import torch as t  from torch.autograd import Variable    x = Variable(t.ones(2, 2), requires\_grad = True)  x  # 实际查询的是.data，是个Tensor |

实际上查询x和查询x.data返回结果一致，

Variable containing:

 1 1

 1 1

 [torch.FloatTensor of size 2x2]

[回到顶部](https://www.cnblogs.com/hellcat/p/8439055.html" \l "_labelTop)

## **梯度求解**

构建一个简单的方程:y = x[0,0] + x[0,1] + x[1,0] + x[1,1]，Variable的运算结果也是Variable，但是，中间结果反向传播中不会被求导()

这和TensorFlow不太一致，TensorFlow中中间运算果数据结构均是Tensor，

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | y = x.sum()    y  """  　　Variable containing:  　　 4  　　[torch.FloatTensor of size 1]  """ |

可以查看目标函数的.grad\_fn方法，它用来求梯度，

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | y.grad\_fn  """  　　　　<SumBackward0 at 0x18bcbfcdd30>  """    y.backward()  # 反向传播  x.grad  # Variable的梯度保存在Variable.grad中  """  　　Variable containing:  　　 1  1  　　 1  1  　　[torch.FloatTensor of size 2x2]  """ |

grad属性保存在Variable中，新的梯度下来会进行累加，可以看到再次求导后结果变成了2，

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | y.backward()  x.grad  # 可以看到变量梯度是累加的  """      Variable containing:       2  2       2  2      [torch.FloatTensor of size 2x2]  """ |

所以要归零，

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | x.grad.data.zero\_()  # 归零梯度，注意，在torch中所有的inplace操作都是要带下划线的，虽然就没有.data.zero()方法    """   0  0   0  0  [torch.FloatTensor of size 2x2]  """ |

## [指定的GPU](https://www.cnblogs.com/darkknightzh/p/6836568.html)

直接终端中设定：

CUDA\_VISIBLE\_DEVICES=1 python my\_script.py

python代码中设定：

import os

os.environ["CUDA\_VISIBLE\_DEVICES"] = "2"

使用函数 set\_device

import torch

torch.cuda.set\_device(id)

## PyTorch 常用方法总结

涉及的方法有下面几种

torch.cat() torch.Tensor.expand()

torch.squeeze() torch.Tensor.repeat()

torch.Tensor.narrow() torch.Tensor.view()

torch.Tensor.resize\_() torch.Tensor.permute()

1

2

3

4

5

6

7

拼接张量

torch.cat(seq, dim=0, out=None) → Tensor

在指定的维度dim上对序列seq进行连接操作。

参数：

seq (sequence of Tensors) - Python序列或相同类型的张量序列

dim (int, optional) - 沿着此维度连接张量

out (Tensor, optional) - 输出参数

x = torch.randn(2, 3)

x

-0.5866 -0.3784 -0.1705

-1.0125 0.7406 -1.2073

[torch.FloatTensor of size 2x3]

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

-0.5866 -0.3784 -0.1705

-1.0125 0.7406 -1.2073

-0.5866 -0.3784 -0.1705

-1.0125 0.7406 -1.2073

-0.5866 -0.3784 -0.1705

-1.0125 0.7406 -1.2073

[torch.FloatTensor of size 6x3]

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

-0.5866 -0.3784 -0.1705 -0.5866 -0.3784 -0.1705 -0.5866 -0.3784 -0.1705

-1.0125 0.7406 -1.2073 -1.0125 0.7406 -1.2073 -1.0125 0.7406 -1.2073

[torch.FloatTensor of size 2x9]

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

拼接张量2

torch.stack((Tensor), dim)

例子：

a = torch.IntTensor([[1,2,3],[11,22,33]])

b= torch.IntTensor([[4,5,6],[44,55,66]])

c=torch.stack([a,b],0)

d=torch.stack([a,b],1)

e=torch.stack([a,b],2)

print(c)

print(d)

print(e)

>>> print(c)

tensor([[[ 1, 2, 3],

[11, 22, 33]],

[[ 4, 5, 6],

[44, 55, 66]]], dtype=torch.int32)

>>> print(d)

tensor([[[ 1, 2, 3],

[ 4, 5, 6]],

[[11, 22, 33],

[44, 55, 66]]], dtype=torch.int32)

>>> print(e)

tensor([[[ 1, 4],

[ 2, 5],

[ 3, 6]],

[[11, 44],

[22, 55],

[33, 66]]], dtype=torch.int32)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

c, dim = 0时， c = [ a, b]

d, dim =1 时， d = [ [a[0] , b[0] ] , [a[1], b[1] ] ]

e, dim = 2 时， e=[[[a[0][0],b[0][0]],[a[0][1],b[0][1]],[a[0][2],b[0][2]]],[[a[1][0],b[1][0]],[a[1][1],b[0][1]],[a[1][2],b[1][2]]]]e = [ [ [ a[0][0], b[0][0] ] , [ a[0][1], b[0][1] ] , [ a[0][2],b[0][2] ] ] , [ [ a[1][0], b[1][0] ] , [ a[1][1], b[0][1] ] , [ a[1][2],b[1][2] ] ] ]e=[[[a[0][0],b[0][0]],[a[0][1],b[0][1]],[a[0][2],b[0][2]]],[[a[1][0],b[1][0]],[a[1][1],b[0][1]],[a[1][2],b[1][2]]]]

扩大张量

torch.Tensor.expand(\*sizes) → Tensor

返回张量的一个新视图，可以将张量的单个维度扩大为更大的尺寸。

张量也可以扩大为更高维，新增加的维度将附在前面。 扩大张量不需要分配新内存，仅仅是新建一个张量的视图。任意一个一维张量在不分配新内存情况下都可以扩展为任意的维度。

传入-1则意味着维度扩大不涉及这个维度。

参数：

sizes (torch.Size or int…) – 想要扩展的目标维度

例子：

x = torch.Tensor([[1], [2], [3]])

x.size()

torch.Size([3, 1])

x.expand(3, 4)

1 1 1 1

2 2 2 2

3 3 3 3

[torch.FloatTensor of size 3x4]

1

2

3

4

5

6

7

8

压缩张量

torch.squeeze(input, dim=None, out=None) → Tensor  
除去输入张量input中数值为1的维度，并返回新的张量。如果输入张量的形状为（A×1×B×C×1×D）（ A \times 1 \times B \times C \times 1 \times D ）（A×1×B×C×1×D），那么输出张量的形状为（A×B×C×D）（ A \times B \times C \times D ）（A×B×C×D）。

当通过dim参数指定维度时，维度压缩操作只会在指定的维度上进行。如果输入向量的形状为（A×1×B）（ A \times 1 \times B ）（A×1×B），squeeze(input, 0)会保持张量的维度不变，只有在执行squeeze(input, 1)时，输入张量的形状会被压缩至（A×B）（ A \times B ）（A×B）。

如果一个张量只有1个维度，那么它不会受到上述方法的影响。

输出的张量与原张量共享内存，如果改变其中的一个，另一个也会改变。

参数:

input (Tensor) – 输入张量

dim (int, optional) – 如果给定，则只会在给定维度压缩

out (Tensor, optional) – 输出张量

例子：

x = torch.zeros(2, 1, 2, 1, 2)

x.size()

torch.Size([2, 1, 2, 1, 2])

y = torch.squeeze(x)

y.size()

torch.Size([2, 2, 2])

y = torch.squeeze(x, 0)

y.size()

torch.Size([2, 1, 2, 1, 2])

y = torch.squeeze(x, 1)

y.size()

torch.Size([2, 2, 1, 2])

1

2

3

4

5

6

7

8

9

10

11

12

重复张量

torch.Tensor.repeat(\*sizes)

沿着指定的维度重复张量。不同于expand()方法，本函数复制的是张量中的数据。

参数：

size (torch.size ot int…) - 沿着每一维重复的次数  
例子：

x = torch.Tensor([1, 2, 3])

x.repeat(4, 2)

1 2 3 1 2 3

1 2 3 1 2 3

1 2 3 1 2 3

1 2 3 1 2 3

[torch.FloatTensor of size 4x6]

1

2

3

4

5

6

7

torch.Tensor.unfold(dim, size, step) → Tensor

返回一个新的张量，其中元素复制于有原张量在dim维度上的数据，复制重复size次，复制时的步进值为step。

参数：

dim (int) - 目标维度

size (int) - 复制重复的次数（展开维度）

step (int) - 步长  
例子：

x = torch.arange(1, 8)

x

1

2

3

4

5

6

7

[torch.FloatTensor of size 7]

x.unfold(0, 2, 1)

1 2

2 3

3 4

4 5

5 6

6 7

[torch.FloatTensor of size 6x2]

x.unfold(0, 2, 2)

1 2

3 4

5 6

[torch.FloatTensor of size 3x2]

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

缩小张量

torch.Tensor.narrow(dimension, start, length) → Tensor

返回一个经过缩小后的张量。操作的维度由dimension指定。缩小范围是从start开始到start+length。执行本方法的张量与返回的张量共享相同的底层内存。

参数：

dimension (int) – 要进行缩小的维度

start (int) – 开始维度索引

length (int) – 缩小持续的长度  
例子：

x = torch.Tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

x.narrow(0, 0, 2)

1 2 3

4 5 6

[torch.FloatTensor of size 2x3]

x.narrow(1, 1, 2)

2 3

5 6

8 9

[torch.FloatTensor of size 3x2]

1

2

3

4

5

6

7

8

9

10

张量变形

torch.Tensor.view(\*args) → Tensor

返回一个有相同数据但是不同形状的新的向量。

返回的装两必须与原张量有相同的数据和相同的元素个数，但是可以有不同的尺寸。

参数：

args (torch.Size or int…) - 理想的指定尺寸

x = torch.randn(4, 4)

x.size()

torch.Size([4, 4])

y = x.view(16)

y.size()

torch.Size([16])

1

2

3

4

5

6

重设张量尺寸

torch.Tensor.resize\_(\*sizes)

将张量的尺寸调整为指定的大小。如果元素个数比当前的内存大小大，就将底层存储大小调整为与新元素数目一致的大小。  
如果元素个数比当前内存小，则底层存储不会被改变。原来张量中被保存下来的元素将保持不变，但新内存将不会被初始化。  
参数：

sizes (torch.Size or int…) - 需要调整的大小

例子：

x = torch.Tensor([[1, 2], [3, 4], [5, 6]])

x.resize\_(2, 2)

x

1 2

3 4

[torch.FloatTensor of size 2x2]

1

2

3

4

5

6

置换张量维度

torch.Tensor.permute(\*dims)

将执行本方法的张量的维度换位。

参数：

dim (int) - 指定换位顺序

例子：

x = torch.randn(2, 3, 5)

x.size()

torch.Size([2, 3, 5])

x.permute(2, 0, 1).size()

torch.Size([5, 2, 3])

1

2

3

4

5

查看张量单个元素的字节数

torch.Tensor.element\_size() → int

查看某类型张量单个元素的字节数。

例子：

torch.FloatTensor().element\_size()

4

## 1. 张量Tensors

### torch.is\_tensor(obj):

如果obj是一个pytorch张量，则返回True

### torch.is\_storage(obj):

如果obj是一个pytorch storage对象，则返回True

### torch.numel(input):

返回input张量中的元素个数。

## 2. 创建操作

### torch.eye(n, m=None, out=None):

返回一个2维张量，对角线为1,其它位置为0

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

### torch.from\_numpy(ndarray):

将numpy.ndarray转换为Tensor，**返回的张量tensor和numpy的ndarray共享同一内存空间，修改一个会导致另一个也被修改，返回的张量不能改变大小**

### torch.linspace(start, end, steps=100, out=None):

返回一个1维张量，包含在start和end上均匀间隔的steps个点

* start (float) -序列起点
* end (float) - 序列终点
* steps (int) - 在start与end间生成的样本数
* out (Tensor, optional) - 结果张量

### torch.logspace(start, end, steps=100, out=None):

返回一个1维张量，包含在区间10exp(start)和10exp(end)上以对数刻度均匀间隔的 steps个点。

### torch.ones(\*sizes, out=None):

返回一个全为1的张量，形状由可变参数sizes定义

* sizes (int...) - 整数序列，定义了输出形状

### torch.rand(\*sizes, out=None):

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。

### torch.randn(\*sizes, out=None):

返回一个张量，包含了从标准正态分布(mean=0, std=1)中抽取一组随机数，形状由可变参数sizes定义。

### torch.randperm(n, out=None):

给定参数n，返回一个从0到n-1的随机整数排列

* n (int) - 上边界(不包含）

### torch.arange(start, end, step=1, out=None):

返回一个1维张量，长度为floor((end-start)/step)，以step`为步长的一组序列值。

* start (float) - 起点
* end (float) - 终点(不包含）
* step (float) - 相邻点的间隔大小
* out (Tensor, optional)

### torch.range(start, end, step=1, out=None):

**还是推荐使用**torch.arange()

### torch.zeros(\*sizes, out=None):

返回一个全为标量0的张量，形状由可变参数sizes定义

## 3. 索引，切片，连接，换位(Index, Slicing, Joining, Mutating)

### torch.cat(inputs, dimension=0):

在给定维度上对输入的张量序列seq进行连接操作。

* inputs (sequence of Tensors)
* dimension (int optional) - 沿着此维连接张量序列

### torch.chunk(tensor, chunks, dim=0):

在给定维度上将输入张量进行分块

* tensors(Tensors) - 待分场的输入张量
* chunks (int) - 分块的个数
* dim (int) - 沿着此维度

### torch.gather(input, dim, index, out=None):

沿给定轴dim,将输入索引张量index指定位置的值进行聚合。

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

### torch.index\_select(input, dim, index, out=None):

沿指定维度对输入进行切片，取index中指定的相应项，然后返回一个新的张量，返回的张量与原始张量有相同的维度(在指定轴上)，**返回的张量与原始张量不共享内存空间**

* 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 - 目标张量

### torch.masked\_select(input, mask, out=None):

根据掩码张量mask中的二元值，取输入张量中的指定项，将取值返回到一个新的1D张量。  
张量mask须跟input张量有相同的元素数目，但形状或维度不需要相同。**返回的张量不与原始张量共享内存空间**

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

### torch.nonzero(input, out=None):

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  
若输入input有n维，则输出的索引张量output形状为z \* n, 这里z是输入张量input中所有非零元素的个数

* input(Tensor) - 输入张量
* out - 包含索引值的结果张量

### 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。

* 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(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 \* 1 \* B \* 1 \* C \* 1 \*D)，那么输出形状就为：(A \* B \* C \* D)。  
当给定dim时，则只在给定维度上进行挤压，如输入形状为(A \* 1 \* B)，squeeze(input, 0)，将会保持张量不变，只有用squeeze(input, 1)，形状会变成(A \*B)。  
**输入张量与返回张量共享内存**

* input(Tensor) - 输入张量
* dim(int, optional) - 如果给定，则只在给定维度挤压
* out(Tensor, optional) - 输出张量

### torch.stack(sequence, dim=0):

沿着一个新维度对输入张量进行连接，序列中所有张量都应该为相同的形状。

* sequence(Sequence) - 待连接的张量序列
* dim(int) - 插入的维度

### torch.t(input, out=None):

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

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

### torch.transpose(input, dim0, dim1, out=None):

返回输入矩阵input的转置，交换维度dim0和dim1。**输入张量与输出张量共享内存。**

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

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

移除指定维度后，返回一个元组，包含了沿着指定维切片后的各个切片

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

### torch.unsequeeze(input, dim, out=None):

返回一个新的张量，对输入的指定位置插入维度1，**返回张量与输入张量共享内存，若**dim**为负，则将被转化为**dim+input.dim()+1

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

## 4. 随机抽样Random sampling

### torch.manual\_seed(seed):

设定生成随机数的种子，并返回一个torch.\_C.Generator对象

### torch.initial\_seed():

返回生成随机数的原始种子值

### torch.rng\_state()[source]

返回随机生成器状态(ByteTensor)

### 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.bernoulli(input, out=None):

从伯努利分布中抽取二元随机数(0或者1），输入中所有值必须在[0, 1]区间，输出张量的第i个元素值，将以输入张量的第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1

* input(Tensor) - 输入为伯努利分布的概率值
* out(Tensor, optional)

### torch.multinomial(input, num\_samples, replacement=False, out=None):

返回一个张量，每行包含从input相应行中定义的多项式分布中抽取的num\_samples个样本。  
input每行的值不需要总和为1,但必须非负且总和不能为0.

* 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，决定是否能重复抽取
* out(Tensor, optional)

### torch.normal(means, std, out=None):

返回一个张量，包含从给定means, std的离散正态分布中抽取随机数，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

* means(Tensor) - 均值
* std(Tensor) - 标准差
* out(Tensor, optional)

## 5. 序列化 Serialization

### torch.saves(obj, f, pickle\_module, pickle\_protocol=2):

保存一个对象到一个硬盘文件上

* obj - 保存对象
* f - 类文件对象
* pickle\_module - 用于pickling元数据和对象的模块
* pickle\_protocol - 指定pickle protocal可以覆盖默认参数

### torch.load(f, map\_location=None, pickle\_module=):

从磁盘文件中读取一个通过torch.save()保存的对象，可通过参数map\_location动态地进行内存重映射

* f - 类文件对象
* 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

## 6. 并行化 Parallelism

### torch.get\_num\_threads():

获得用于并行化CPU操作的OpenMP线程数

### torch.set\_num\_threads(int):

设定用于并行化CPU操作的OpenMP线程数

## 7. 数学操作 Math operations

## 7.1 Pointwise Ops

### torch.abs(input, out=None):

计算输入张量的每个元素绝对值

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

### torch.acos(input, out=None):

返回一个新张量，包含输入张量每个元素的反余弦

* input(Tensor) - 输入张量
* out(Tensor, optional)

### 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。

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

### torch.addcdiv(tensor, value=1, tensor1, tensor2, out=None):

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

* tensor(Tensor) - 张量
* value(Number, optional) - 标量
* tensor1(Tensor) - 张量，作为分子
* tensor2(Tensor) - 张量，作为分母
* out(Tensor, optional)

### torch.addcmul(tensor, value=1, tensor1, tensor2, out=None):

用tensor2对tensor1逐元素相乘，并对结果乘以标量值value然后加到tensor，张量形状不需要匹配，但元素数量必须一致。

* tensor(Tensor) - 张量
* value(Number, optional) - 标量，
* tensor1(Tensor) - 张量，乘子1
* tensor2(Tensor) - 张量，乘子2
* out(Tensor, optional)

### torch.asin(input, out=None):

返回一个新张量，包含输入input张量每个元素的反正弦函数

* input(Tensor) - 输入张量
* out(Tensor, optional)

### torch.atan(input, out=None):

返回一个新张量，包含输入input张量每个元素的反正切函数

* input(Tensor)
* out(Tensor, optional)

### torch.atan2(input1, input2, out=None):

返回一个新张量，包含两个输入张量input1和input2的反正切函数

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

### torch.ceil(input, out=None):

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

* input(Tensor) - 输入张量
* out(Tensor, optional)

### torch.clamp(input, min, max, out=None):

将输入input张量每个元素值约束到区间[min, max]，并返回结果到一个新张量  
**也可以只设定**min**或只设定**max

* 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)

### torch.cos(input, out=None):

返回一个新张量，包含输入input张量每个元素的余弦

* input(Tensor)
* out(Tensor, optional)

### torch.cosh(input, out=None):

* input(Tensor)
* out(Tensor, optional)

### 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

* input(Tensor) - 输入张量
* value(Number) - 除数
* out(Tensor, optional)

### torch.exp(tensor, out=None):

返回一个新张量，包含输入input张量每个元素的指数

* input(Tensor)
* out(Tensor, optional)

### torch.floor(input, out=None):

返回一个新张量，包含输入input张量每个元素的floor，即不大于元素的最大整数。

* input(Tensor)
* out(Tensor, optional)

### torch.fmod(input, divisor, out=None):

计算除法余数，余数的正负与被除数相同

* input(Tensor)
* divisor(Tensor or float) - 除数
* out(Tensor, optional)

### torch.frac(tensor, out=None):

返回每个元素的分数部分

### 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  
out = start + weight\*(end - start)

* start(Tensor) - 起始点张量
* end(Tensor) - 终止点张量
* weight(float) - 插值公式中的weight
* out(Tensor, optional)

### torch.log(input, out=None):

计算input的自然对数

### torch.log1p(input, out=None):

计算input + 1的自然对数y = log(x + 1)  
**对值比较小的输入，此函数比**torch.log()**更准确**

* input(Tensor)
* out(Tensor, optional)

### 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

### torch.mul(input, other, out=None):

两个张量input, other按**元素相乘**，并返回到输出张量，两个张量形状不须匹配，但总元素数须一致。当形状不匹配时，input的形状作为输出张量的形状

* input(Tensor) - 第一个张量
* other(Tensor) - 第二个张量
* out(Tensor, optional)

### torch.neg(input, out=None):

返回一个新张量，包含输入input张量按元素取负。

### torch.pow(input, exponent, out=None):

对输入input按元素求exponent次幂，并返回结果张量。幂可以为float数或与input相同元素数的张量

* input(Tensor) - 输入张量
* exponent(float or Tensor) - 幂值
* out(Tensor, optional)

### torch.pow(base, input, out=None):

base为标量浮点值，input为张量。

* base(float) - 标量值，指数的底
* input(Tensor) - 幂值
* out(Tensor, optional)

### torch.reciprocal(input, out=None):

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

### torch.remainder(input, divisor, out=None):

返回一个新张量，包含输入input张量每个元素的除法余数，余数与除数有相同的符号。

* input(Tensor) - 被除数
* divisor(Tensor or float) - 除数
* out(Tensor, optional)

### torch.round(input, out=None):

返回一个新张量，将输入input张量每个元素四舍五入到最近的整数。

### torch.rsqrt(input, out=None):

返回一个新张量，包含输入input张量每个元素的平方根倒数。

### torch.sigmoid(input, out=None):

返回一个新张量，包含输入input张量每个元素的sigmoid值

### torch.sign(input, out=None):

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。

### torch.sin(input, out=None):

返回一个新张量，包含输入input张量每个元素的正弦。

### torch.sinh(input, out=None):

返回一个新张量，包含输入input张量每个元素的双曲正弦。

### torch.sqrt(input, out=None):

返回一个新张量，包含输入input张量每个元素的平方根。

### torch.tan(input, out=None):

返回一个新张量，包含输入input张量每个元素的正切。

### torch.tanh(input, out=None):

返回一个新张量，包含输入input张量每个元素的双曲正切。

### torch.trunc(input, out=None):

返回一个新张量，包含输入input张量每个元素的截断值，使更接近零。即有符号数的小数部分被舍弃。

## 7.2 Reduction Ops

### torch.cumprod(input, dim, out=None) -> Tensor:

返回输入沿指定维度的累积积，如输入是一个N元向量，则结果也是一个N元向量，第i个输出元素值为yi = x1 \* x2 \* x3 \* ...\* xi

* input(Tensor) - 输入张量
* dim(int) - 累积乘积操作的维度
* out(Tensor, optional)

### torch.cumsum(input, dim, out=None) -> Tensor:

返回输入沿指定维度的累积和

### torch.dist(input, other, p=2, out=None) -> Tensor:

返回(input - other)的p范数

* input(Tensor) - 输入张量
* other(Tensor) - 右侧输入张量
* p(float, optional) - 要计算的范数
* out(Tensor, optional)

### torch.mean(input) -> float:

返回输入张量所有元素的均值

### torch.mean(input, dim, out=None):

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。

### 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。dim默认为输入张量的最后一维

* 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

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

返回给定维度dim上，每行的众数值，同时返回一个索引张量。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。

### torch.norm(input, p=2) -> float:

返回输入张量input的p范数。

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

### torch.norm(input, p, dim, out=None) -> Tensor:

返回输入张量给定维度dim上每行的p范数。

### torch.prod(input) -> float:

返回输入张量input所有元素的积

### torch.prod(input, dim, out=None) -> Tensor:

返回输入张量给定维度上每行的积。

### torch.std(input) -> float:

返回输入张量input所有元素的标准差

### torch.std(input, dim, out=None):

返回输入张量给定维度上每行的标准差。

### torch.sum(input) -> float:

返回输入张量input所有元素的各

### torch.sum(input, dim, out=None) -> Tensor:

返回输入疑是给定维度上每行的和

### torch.var(input) -> float:

返回输入张量所有元素的方差

### torch.var(input, dim, out=None) -> Tensor:

返回输入张量给定维度上每行的方差。

## 7.3 比较操作Comparison Ops

### 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

* 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类型或与input同类型

### torch.equal(tensor1, tensor2) -> bool:

若两个张量有相同的形状和元素值，则返回True， 否则False。

### torch.ge(input, other, out=None) -> Tensor:

逐元素比较input和other，即是否input >= 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。

* 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或与第一个参数相同类型。

### torch.gt(input, other, out=None) -> Tensor:

逐元素比较input和other，是否input > other。若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。

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

取输入张量input指定维度上第k个最小值，若不指定dim，则默认为input的最后一维。返回一个元组，其中indices是原始输入张量input中沿dim维的第k个最小值下标。

* input(Tensor) - 输入张量
* k(int) - 第k个最小值
* dim(int, optional)` - 沿着此维度进行排序
* out(tuple, optional) - 输出元组

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

逐元素比较input和other，即是否input <= 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。

### torch.lt(input, other, out=None) -> Tensor:

逐元素比较input和other，即是否input < other

### torch.max(input, dim, max=None, max\_indice=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。

* 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) - 包含给定维度上每个最大值的位置索引。

### 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。

### 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。两张量形状不需匹配，但元素数须相同。

### torch.ne(input, other, out=Tensor) -> Tensor:

逐元素比较input和other， 即是否input != 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。  
返回值：一个torch.ByteTensor张量，包含了每个位置的比较结果(如果tensor != other 为True，返回1)。

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

对输入张量input沿着指定维度按升序排序，如果不给定dim，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。  
返回两项：重排后的张量，和重排后元素在原张量的索引

* input(Tensor) - 输入张量
* dim(int, optional) - 沿此维排序，默认为最后一维
* descending(bool, optional) - 布尔值，默认升序

### 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，不指定dim，则默认为最后一维，如果largest为False，则返回最小的k个值。

## 7.4 其它操作 Other Operations

### 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。

### torch.diag(input, diagonal=0, out=None) -> Tensor:

如果输入是一个向量，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，则返回一个包含input为对角元素的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

### torch.histc(input, bins=100, min=0, max=0, out=None) -> Tensor:

计算输入张量的直方图。如果min和max都为0,则利用数据中的最大最小值作为边界。

### 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。如果p范数的值小于maxnorm，则当前子张量不需要修改。

### torch.trace(input) -> float:

返回输入2维矩阵对角元素的和(迹)

### torch.tril(input, diagonal=0, out=None) -> Tensor:

返回一个张量，包含输入张量(2D张量)的下三角部分，其余部分设为0，参数diagonal控制对角线。

### torch.triu（input, diagonal=0, out=None) -> Tensor:

返回一个张量，包含输入矩阵的上三角部分，其余被置为0。

## 7.5 BLAS and LAPACK Operations

### torch.dot(tensor1, tensor2) -> float:

计算两个张量的点乘，两个张量都为1-D向量

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

计算方阵a的特征值和特征向量。

* a(Tensor) - 方阵
* eigenvectors(bool) - 如果为True，同时计算特征值和特征微量，否则只计算特征值  
  返回值：
* 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。

### torch.inverse(input, out=None) -> Tensor:

对方阵input求逆

### torch.mm(mat1, mat2, out=None) -> Tensor:

对矩阵mat1和mat2进行相乘。

### torch.mv(mat, vec, out=None) -> Tensor:

对矩阵mat和向量vec进行相乘。