requires\_grad

如果有一个单一的输入操作需要梯度，它的输出也需要梯度。相反，只有所有输入都不需要梯度，输出才不需要。如果其中所有的变量都不需要梯度进行，后向计算不会在子图中执行。

volatile

纯粹的inference模式下推荐使用volatile，当你确定你甚至不会调用.backward()时。它比任何其他自动求导的设置更有效——它将使用绝对最小的内存来评估模型。volatile也决定了require\_grad is False。

扩展 torch.autograd

如果想要添加一个新的Operation 到autograd的话，Operation需要继承 class Function。autograd使用Function计算结果和梯度，同时编码operation的历史。每个新的 operation(function) 都需要实现三个方法：

\_\_init\_\_ (optional) 如果operation包含非Variable参数，那么就将其作为\_\_init\_\_的参数传入到operation中。例如：AddConstant Function加一个常数，Transpose Function需要指定哪两个维度需要交换。如果你的operation不需要额外的参数，你可以忽略\_\_init\_\_。

forward() - 在里面写执行此operation的代码。可以有任意数量的参数。如果你对某些参数指定了默认值，则这些参数是可传可不传的。记住：forward()的参数只能是Variable。函数的返回值既可以是 Variable也可以是Variables的tuple。同时，请参考 Function[function]的 doc，查阅有哪些 方法是只能在forward中调用的。

backward() - 梯度计算公式。 参数的个数和forward返回值的个数一样，每个参数代表传回到此operation的梯度. backward()的返回值的个数应该和此operation输入的个数一样，每个返回值对应了输入值的梯度。如果operation的输入不需要梯度，或者不可导，你可以返回None。 如果forward()存在可选参数，你可以返回比输入更多的梯度，只是返回的是None。

扩展 torch.nn

nn 包含两种接口 modules和他们的functional版本。通过这两个接口，你都可以扩展nn。但是我们建议，在扩展layer的时候，使用modules， 因为modules保存着参数和buffer。如果不需要参数的话，那么建议使用functional(激活函数，pooling，这些都不需要参数)。

增加一个模块(module)。 由于nn重度使用autograd。所以，添加一个新module需要实现一个 用来执行 计算 和 计算梯度 的Function。从现在开始，假定我们想要实现一个Linear module，记得之前已经实现了一个Linear Funciton。 只需要很少的代码就可以完成这个工作。 现在，需要实现两个方法：

\_\_init\_\_ (optional) - 输入参数，例如kernel sizes, numbers of features, 等等。同时初始化 parameters和buffers。

forward() - 实例化一个执行operation的Function，使用它执行operation。和functional wrapper(上面实现的那个简单的wrapper)十分类似。

保存模型的推荐方法

这主要有两种方法序列化和恢复模型。

第一种(推荐）只保存和加载模型参数：

torch.save(the\_model.state\_dict(), PATH)

然后：

the\_model = TheModelClass(\*args, \*\*kwargs)

the\_model.load\_state\_dict(torch.load(PATH))

第二种保存和加载整个模型：

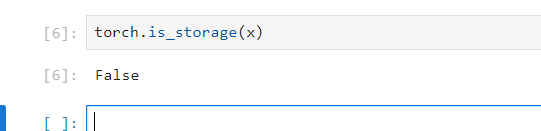
torch.save(the\_model, PATH)

然后：

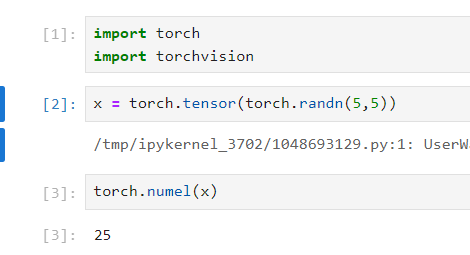
the\_model = torch.load(PATH)

torch.is\_tensor(x) 如果x是一个pytorch张量则返回True



**torch.is\_storage**(obj) 判断x是否是pytorch storage对象

torch.numel(x) 输出张量x中元素的个数



torch.set\_printoptions(precision=None, threshold=None, edgeitems=None, linewidth=None, profile=None) 设置打印选项

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 matricies will ignore this parameter.

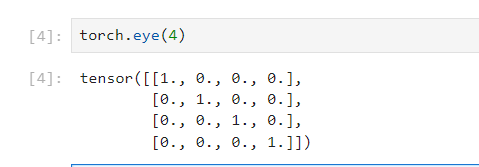
profile – pretty打印的完全默认值。 可以覆盖上述所有选项 (默认为short, full)

torch.eye(n, m=None, out=None) 返回单位矩阵

n (int) – 行数

m (int, optional) – 列数.如果为None,则默认为n

out (Tensor, optinal) - Output tenso



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。

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

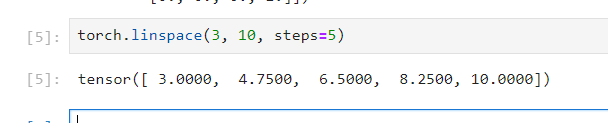
返回一个1维张量，包含在区间start 和 end 上均匀间隔的steps个点。 输出1维张量的长度为steps。

start (float) – 序列的起始点

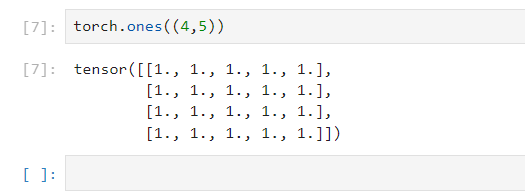
end (float) – 序列的最终值

steps (int) – 在start 和 end间生成的样本数

out (Tensor, optional) – 结果张量

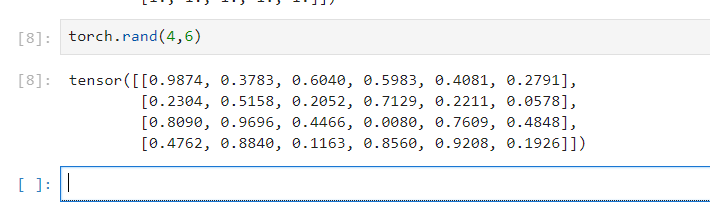


torch.ones(\*sizes, out=None) → Tensor 形成大小为sizes值为1的张量



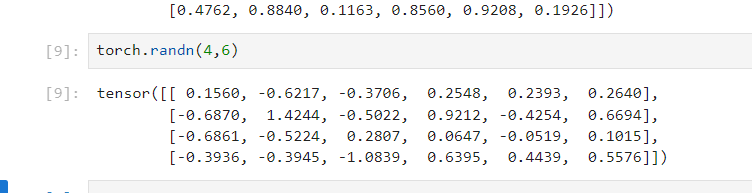
torch.rand(\*sizes, out=None) → Tensor

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。



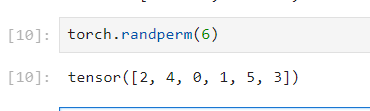
torch.randn(\*sizes, out=None) → Tensor

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



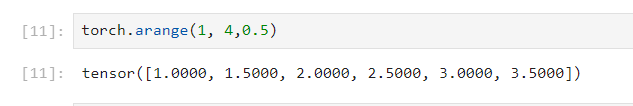
torch.randperm(n, out=None) → LongTensor

给定参数n，返回一个从0 到n -1 的随机整数排列。

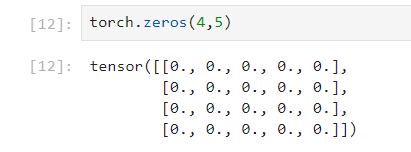


torch.arange(start, end, step=1, out=None) → Tensor

包含从start到end，以step为步长的一组序列值(默认步长为1)。



torch.ones(\*sizes, out=None) → Tensor 形成大小为sizes值为0的张量

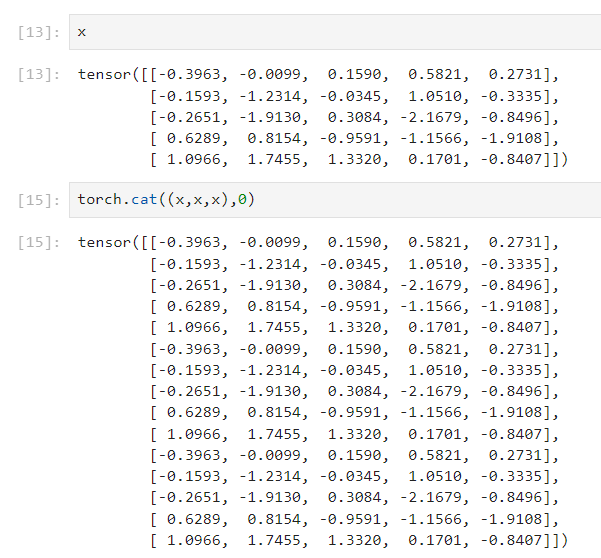


torch.cat(inputs, dimension=0) → Tensor

在给定维度上对输入的张量序列seq 进行连接操作。

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。



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

在给定维度(轴)上将输入张量进行分块儿。

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(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。

torch.index\_select(input, dim, index, out=None) → Tensor

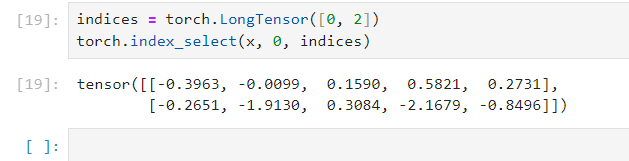
沿着指定维度对输入进行切片，取index中指定的相应项(index为一个LongTensor)，然后返回到一个新的张量， 返回的张量与原始张量\_Tensor\_有相同的维度(在指定轴上)。

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



torch.masked\_select(input, mask, out=None) → Tensor

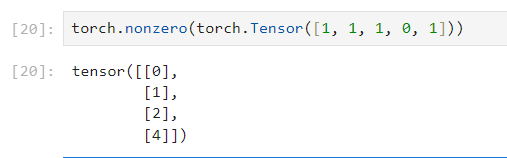
根据掩码张量mask中的二元值，取输入张量中的指定项( mask为一个 ByteTensor)，将取值返回到一个新的1D张量，

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。

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 x n, 这里 z 是输入张量input中所有非零元素的个数。



torch.split(tensor, split\_size, dim=0)

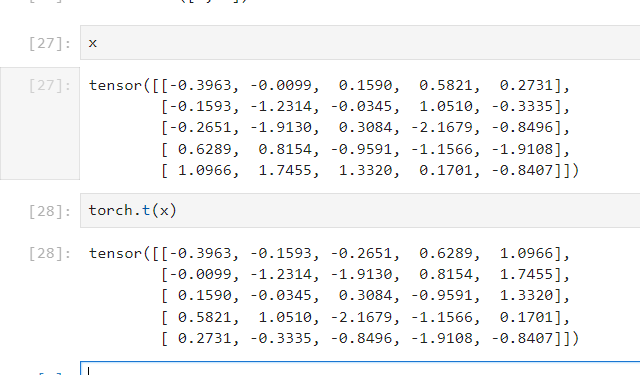
将输入张量分割成相等形状的chunks(如果可分）。 如果沿指定维的张量形状大小不能被split\_size 整分， 则最后一个分块会小于其它分块。

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

减少张量的维度

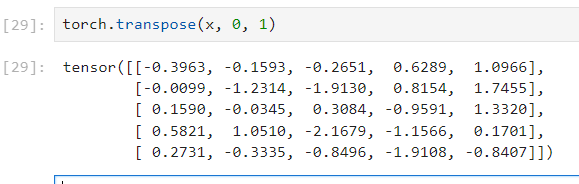
torch.t(input, out=None) → Tensor

将输入的矩阵转置



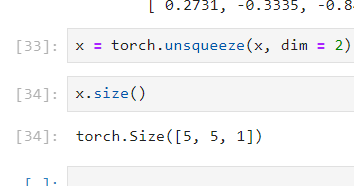
torch.transpose(input, dim0, dim1, out=None) → Tensor

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。



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



torch.multinomial(input, num\_samples,replacement=False, out=None) → LongTensor

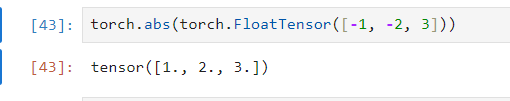
返回一个张量，每行包含从input相应行中定义的多项分布中抽取的num\_samples个样本。

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。

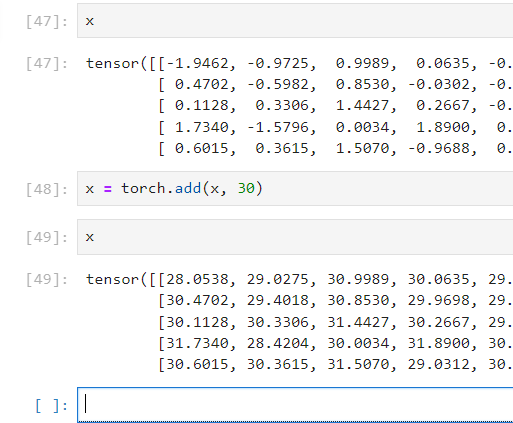
torch.abs(input, out=None) → Tensor

计算输入张量的每个元素绝对值



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

对输入张量input逐元素加上标量值value

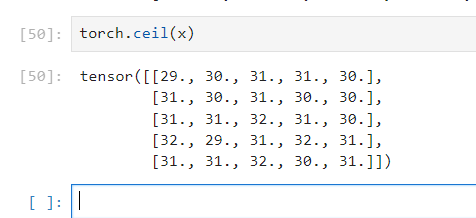


torch.addcdiv(tensor, value=1, tensor1, tensor2, out=None) → Tensor

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

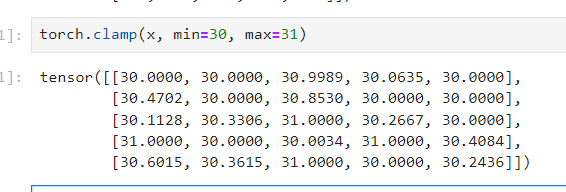
torch.ceil(input, out=None) → Tensor

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。



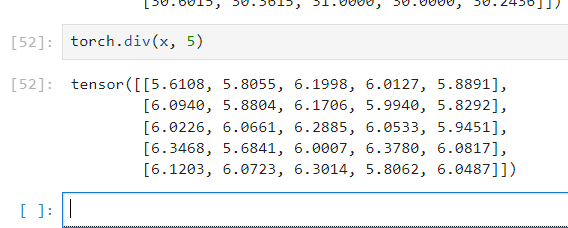
torch.clamp(input, min, max, out=None) → Tensor

将输入input张量每个元素的夹紧到区间



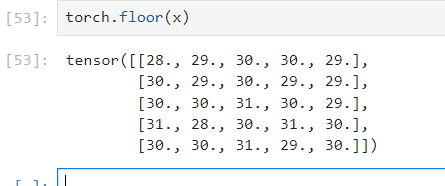
torch.div(input, value, out=None)

将input逐元素除以标量值value



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。

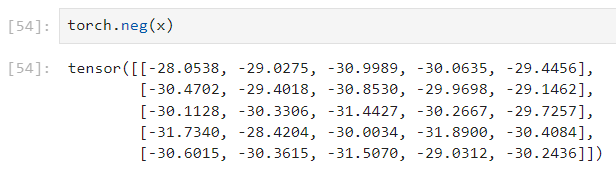


torch.mul(input, value, out=None)

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.neg(input, out=None) → Tensor

返回一个新张量，包含输入input 张量按元素取负



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

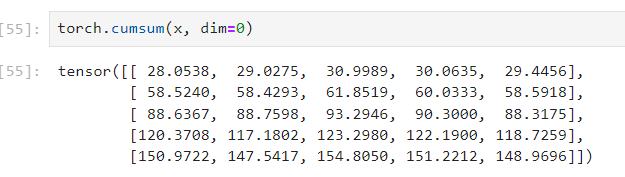
返回一个新张量，包含输入input张量每个元素的倒数，即 1.0/x。

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。

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

返回输入沿指定维度的累积和。



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

返回给定维dim上，每行的众数值。

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

比较元素相等性

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。

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。

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

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

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。

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。

byte()

将此存储转为byte类型

char()

将此存储转为char类型

clone()

返回此存储的一个副本

copy\_()

cpu()

如果当前此存储不在CPU上，则返回一个它的CPU副本

cuda(device=None, async=False)

返回此对象在CUDA内存中的一个副本。

如果此对象已在CUDA内存中且在正确的设备上，那么不会执行复制操作，直接返回原对象

double()

将此存储转为double类型

element\_size()

fill\_()

float()

将此存储转为float类型

from\_buffer()

half()

将此存储转为half类型

int()

将此存储转为int类型

is\_cuda = False

is\_pinned()

is\_shared()

is\_sparse = False

long()

将此存储转为long类型

new()

pin\_memory()

如果此存储当前未被锁定，则将它复制到锁定内存中。

resize\_()

share\_memory\_()

将此存储移动到共享内存中。

对于已经在共享内存中的存储或者CUDA存储，这是一条空指令，它们不需要移动就能在进程间共享。共享内存中的存储不能改变大小。

返回：self

short()

将此存储转为short类型

size()

tolist()

返回一个包含此存储中元素的列表

type(new\_type=None, async=False)

将此对象转为指定类型。

如果已经是正确类型，不会执行复制操作，直接返回原对象

children()

Returns an iterator over immediate children modules. 返回当前模型 子模块的迭代器

cpu(device\_id=None)

将所有的模型参数(parameters)和buffers复制到CPU

NOTE：官方文档用的move，但我觉着copy更合理。

cuda(device\_id=None)

将所有的模型参数(parameters)和buffers赋值GPU

参数说明:

device\_id (int, optional) – 如果指定的话，所有的模型参数都会复制到指定的设备上。

double()

将parameters和buffers的数据类型转换成double。

eval()

将模型设置成evaluation模式

仅仅当模型中有Dropout和BatchNorm是才会有影响。

float()

将parameters和buffers的数据类型转换成float。

forward(\* input)

定义了每次执行的 计算步骤。 在所有的子类中都需要重写这个函数。

half()

将parameters和buffers的数据类型转换成half。

load\_state\_dict(state\_dict)

将state\_dict中的parameters和buffers复制到此module和它的后代中。state\_dict中的key必须和 model.state\_dict()返回的key一致。 NOTE：用来加载模型参数。

train(mode=True)

将module设置为 training mode。

仅仅当模型中有Dropout和BatchNorm是才会有影响。

zero\_grad()

将module中的所有模型参数的梯度设置为0.

class torch.nn.Sequential(\* args)

一个时序容器。Modules 会以他们传入的顺序被添加到容器中。当然，也可以传入一个OrderedDict。

非线性激活函数

torch.nn.functional.threshold(input, threshold, value, inplace=False)

torch.nn.functional.relu(input, inplace=False)

torch.nn.functional.hardtanh(input, min\_val=-1.0, max\_val=1.0, inplace=False)

torch.nn.functional.relu6(input, inplace=False)

torch.nn.functional.elu(input, alpha=1.0, inplace=False)

torch.nn.functional.leaky\_relu(input, negative\_slope=0.01, inplace=False)

torch.nn.functional.prelu(input, weight)

torch.nn.functional.rrelu(input, lower=0.125, upper=0.333333333333333, training=False, inplace=False)

torch.nn.functional.logsigmoid(input)

torch.nn.functional.hardshrink(input, lambd=0.5)

torch.nn.functional.tanhshrink(input)

torch.nn.functional.softsign(input)

torch.nn.functional.softplus(input, beta=1, threshold=20)

torch.nn.functional.softmin(input)

torch.nn.functional.softmax(input)

torch.nn.functional.softshrink(input, lambd=0.5)

torch.nn.functional.log\_softmax(input)

torch.nn.functional.tanh(input)

torch.nn.functional.sigmoid(input)

Normalization 函数

torch.nn.functional.batch\_norm(input, running\_mean, running\_var, weight=None, bias=None, training=False, momentum=0.1, eps=1e-05)

线性函数

torch.nn.functional.linear(input, weight, bias=None)

Dropout 函数

torch.nn.functional.dropout(input, p=0.5, training=False, inplace=False)

距离函数(Distance functions）

torch.nn.functional.pairwise\_distance(x1, x2, p=2, eps=1e-06)

optimizer = optim.SGD(model.parameters(), lr = 0.01, momentum=0.9)

optimizer = optim.Adam([var1, var2], lr = 0.0001)