

Pytorch Note: 一份（不太）简短的笔记

Song Chao

版本：1.0

更新：2023 年 1 月 12 日



1 Pytorch 快速入门

本章主要介绍张量、自动微分、`torch.nn` 模块的卷积池化等操作，以及数据预处理等相关的模块。

1.1 张量

张量一共有八种，但是默认的数据类型是 32 位浮点型（`torch.FloatTensor`），可以通过 `torch.set_default_tensor_type()` 函数设置默认的数据类型，但是该函数只支持设置浮点型的数据类型。

1.1.1 张量的生成

在程序中使用 `torch.tensor()` 函数生成一个张量，然后使用 `.dtype` 方法获取其数据类型，可以使用 `torch.get_default_dtype()` 函数获得默认的数据类型。

通过 `torch.tensor()` 函数可以将 Python 的列表转化为张量，张量的维度使用 `.shape` 查看，也可以使用 `.size()` 方法计算张量的形状大小，使用 `.numel()` 方法计算张量中包含元素的数量。

在使用 `torch.tensor()` 函数时，使用参数 `dtype` 来指定张量的数据类型，使用 `requires_grad` 来指定张量是否需要计算梯度¹。

针对已经生成的张量可以使用 `torch.**_like()` 系列函数生成与指定张量维度相同、性质相似的张

¹只有浮点类型的张量才允许计算梯度

量，如使用 `torch.ones_like(D)` 生成与 D 维度相同的全 1 张量，使用 `torch.zeros_like()` 生成全 0 张量，使用 `torch.rand_like()` 生成随机张量。针对一个创建好的张量 D，使用 `D.new_**()` 系列函数创建新的张量，如使用 `D.new_tensor()` 将列表转化为张量。还有一些函数可以得到新的张量²

- `D.new_full((3,3),fill_value=1)`: 3×3 使用 1 填充的张量；
- `D.new_zeros((3,3))`: 3×3 的全 0 张量；
- `D.new_empty((3,3))`: 3×3 的空张量；
- `D.new_ones((3,3))`: 3×3 的全 1 张量。

张量和 NumPy 数组可以相互转换。将 NumPy 数组转化为 Pytorch 张量，可以使用 `torch.as_tensor()` 和 `torch.from_numpy()`，但是需要注意转换成的 NumPy 数组默认是 64 位浮点型数据。对于张量，使用 `torch.numpy()` 即可转化为 NumPy 数组。

可以通过相关随机数来生成张量，并且可以指定生成随机数的分布函数等，在生成随机数之前，可以使用 `torch.manual_seed()`，指定生成随机数的种子，保证生成随机数是可重复出现的。如使用 `torch.normal()` 生成服从高斯的随机数，在该函数中，通过 `mean` 指定随机数的均值，`std` 参数指定标准差，如果这两个参数只有一个元素则只生成一个随机数，如果有多个值，可以生成多个随机数。也可以使用 `torch.rand()` 函数，在区间 $[0,1]$ 上生成服从均匀分布的张量。使用 `torch.randn()` 函数则可生成服从标准正态分布的随机数张量。使用 `torch.randperm(n)` 函数，可以将 $0 \sim n$ （包含 0，不包含 n）之间的整数进行随机排序后输出。

在 Pytorch 中可以使用 `torch.arange()` 和 `torch.linspace()` 来生成张量，前者的参数 `start` 指定开始，

²D 起的作用就是创建的张量和 D 的数据类型一致

`end` 指定结束，`step` 指定步长；后者是在范围内生成固定数量的等间隔张量；`torch.logspace()` 则可生成以对数为间隔的张量。

1.1.2 张量操作

生成张量后，有时需要改变张量的形状、获取或改变张量中的元素、将张量进行拼接和拆分等。

`A.reshape()` 可以将张量 **A** 设置为想要的形状大小，或者直接通过 `torch.reshape()` 函数改变输入张量的形状，参数 `input` 为需要改变的 `tensor`，`shape` 为想要的形状。改变张量的形状可以使用 `tensor.resize_()`，针对输入的形状大小对张量形状进行修改。还提供了 `A.resize_as_(B)`，可以将张量 **A** 的形状尺寸设置为和 **B** 一样的形状。

`torch.unsqueeze()` 可以在张量的指定维度插入新的维度得到维度提升的张量，而 `torch.squeeze()` 可以移除指定或者所有维度为 1 的维度，从而得到减小的新张量。

可以使用 `.expand()` 对张量的维度进行扩展，而 `A.expand_as(C)` 方法会将张量 **A** 根据张量 **C** 的形状大小进行拓展，得到新的张量。使用张量的 `.repeat()` 方法可以将张量堪称一个整体，然后根据指定的形状进行重复填充，得到新的张量。

从张量中利用切片和索引提取元素的方法，和在 `numpy` 中的使用方法是一致的。也可以按需将索引设置为相应的 `bool` 值，然后提取真条件下的内容。

`torch.tril()` 可以获取张量下三角部分的内容，而将上三角的元素设置为 0；`torch.triu()` 则相反；`torch.diag()` 可以获取矩阵张量的对角线元素，或者提供一个向量生成一个矩阵张量。上述三个函数可

以通过 `diagonal` 参数来控制所要考虑的对角线。`torch.diag()` 提供对角线元素，来生成对角矩阵。

Pytorch 中提供了将多个张量拼接为 1 个张量，或者将一个张量拆分为几个张量的函数，其中 `torch.cat()` 将多个张量在指定的维度进行拼接，得到新的张量。`torch.stack()` 可以将多个张量按照指定的维度进行拼接。`torch.chunk()` 可以将张量分割为特定数量的块；`torch.split()` 在将张量分割为特定数量的块时，可以指定每个块的大小。

1.1.3 张量计算

主要包括张量之间的大小比较，基本运算，与统计相关的运算，如排序、最大值及其位置。

针对张量之间的元素比较大小，主要有以下几个

- `torch.allclose()`: 比较两个元素是否接近，公式为 $|A - B| \leq atol + rtol \times |B|$;
- `torch.eq()`: 逐元素比较是否相等;
- `torch.equal()`: 判断两个张量是否具有相同的形状和元素;
- `torch.ge()`: 逐元素比较大于等于;
- `torch.gt()`: 逐元素比较大于;
- `torch.le()`: 逐元素比较小于等于;
- `torch.lt()`: 逐元素比较小于;
- `torch.ne()`: 逐元素比较不等;
- `torch.isnan()`: 判断是否为缺失值;

张量的基本运算，一种为逐元素之间的运算，如加减乘除、幂运算、平方根、对数、数据裁剪等，一种为矩阵之间的运算，如矩阵相乘、矩阵的转置、矩阵的迹等。

计算张量的幂可以用 `torch.pow()`, 或者 `**` 符号。计算指数可以使用 `torch.exp()`; 对数为 `torch.log()`; 开方为 `torch.sqrt()`; 平方根的倒数为 `torch.rsqrt()`。针对数据的裁剪, 有根据最大值裁剪 `torch.clamp_max()`; 有根据最小值裁剪 `torch.clamp_min()`; 还有根据范围裁剪 `torch.clamp()`。

矩阵运算中, 有 `torch.t()` 为转置; `torch.matmul()` 输出两个矩阵的乘积; `torch.inverse()` 为矩阵的逆; `torch.trace()` 为矩阵的迹

还有一些基础的统计计算功能, `torch.max()` 计算最大值; `torch.argmax()` 输出最大值所在的位置; `torch.min()` 和 `torch.argmin()` 也类似。`torch.sort()` 可以对一维张量进行排序, 或者对高维张量在指定的维度进行排序, 在输出排序结果的同时, 还会输出对应的值在原始位置的索引。`torch.topk()` 根据指定的 `k` 值, 计算出张量中取值大小为第 `k` 大的数值与数值所在的位置; `torch.kthvalue()` 根据指定的 `k` 值, 计算出张量中取值大小为第 `k` 小的数值与数值所在的位置。

还有一些基础函数如下所示:

- `torch.mean()`: 根据指定的维度计算均值
- `torch.sum()`: 根据指定的维度求和
- `torch.cumsum()`: 根据指定的维度计算累加和
- `torch.median()`: 根据指定的维度计算中位数
- `torch.cumprod()`: 根据指定的维度计算累乘积

- torch.std(): 计算标准差

1.1.4 Pytorch 中的自动微分

在 torch 中的 torch.autograd 模块，提供了实现任意标量值函数自动求导的类和函数，针对一个张量只需要设置参数 *requires_grad = True*，通过相关计算即可输出其在传播过程中的梯度信息。在 Pytorch 中生成一个矩阵张量 x，并且 $y = \text{sum}(x^2 + 2x + 1)$ ，计算出 y 在 x 上的导数，程序如下。

也可以通过 y.backward() 来计算 y 在 x 的每个元素上的导数。

```
1  import random
2  import collections
3  Card = collections.namedtuple('Card', ['rank', 'suit'])
4  # 一个叫做 FrenchDesk 的类。a class named FrenchDesk.
5  class FrenchDesk:
6      ranks = [str(n) for n in range(2, 11)] + list('JQKA')
7      suits = 'spades diamonds clubs hearts'.split()
8
9      def __init__(self):
10         self._cards = [Card(rank, suit) for rank in self.ranks for suit in self.suits
11                        ]
```

```
12     def __len__(self):
13         return len(self._cards)
14
15     def __getitem__(self, position):
16         return self._cards[position]
17 desk = FrenchDesk()
```