深入理解 Tensor 数据结构

MegEngine 中提供了一种名为 “张量” （**`Tensor`** ）的数据结构， 区别于数学中的定义，其概念与 NumPy 中的 **`ndarray`** 更加相似， 即张量是一类同构多维数组，其中每个元素占用相同大小的内存块，并且所有块都以完全相同的方式解释。 如何解释 Tensor 中的元素由其 数据类型 决定，而每种数据类型都代表一类 Tensor.

- 我们可以基于 Tensor 数据结构，进行各式各样的科学计算；
- Tensor 也是神经网络编程时所用的主要数据结构，网络的输入、输出和转换都使用 Tensor 表示。

> **注解**
>
> 与 NumPy 的区别之处在于，MegEngine 还支持利用 GPU 设备进行更加高效的计算。 当 GPU 和 CPU 设备都可用时，MegEngine 将优先使用 GPU 作为默认计算设备，无需用户进行手动设定。
>
> - 如果有查看/改变默认计算设备的需求，请参考 Tensor 所在设备 中的说明。
> - 通过 **`Tensor.to`** 和 **`functional.copy`** 可将 Tensor 拷贝到指定设备。

> **参见**
>
> 如果你还不清楚如何获得一个 Tensor, 请参考 如何创建一个 Tensor 。

概念（术语）使用上的区分

我们所提到的 Tensor 的概念往往是其它更具体概念的概括（或者说推广），下面有一些例子：

数学	计算机科学	抽象概念	具象化例子
标量（scalar）	数字（number）	点	得分、概率
向量（vector）	数组（array）	线	列表
矩阵（matrix）	2 维数组（2d-array）	面	Excel 表格

不同的研究领域对同一个概念使用不同的术语进行描述，这很常见，对这些概念不清晰的话很容易产生疑惑。

Python 中提供了 array 的官方实现， 但其使用方法和我们提到的 NumPy 数组有所不同，因此我们可以用 Python （嵌套）列表 **`list`** 来类比举例。 在后续的页面，我们会慢慢地过渡到 Tensor 的实际使用和操作中。

注意：为了方便理解，我们这里假设此处 Python 列表中的数据类型是一致的，比如都是 Number 类型。

> **注解**
>
> 在深度学习领域，我们通常将上述这些概念统称为张量（Tensor）。

访问 Tensor 中某个元素

对于数字（或者说标量） Tensor, 显然我们可以直接得到其值，因为它只有一个元素。

```
>>> a = 20200325
>>> a
20200325
```

其它情况下，想要在 Tensor 中获得某个元素，需要提供对应位置的整数索引（Index），并使用下标运算符 `[]`:

- 注意：Tensor 的索引是基于零（Zero-based）开始计数的，和 Python 列表 / NumPy 多维数组一致；
- 比如我们想要获取向量/数组 `a = [0, 1, 2, 3, 4]` 中的第 3 个元素，我们需要使用 `a[2]`;
- 又比如我们想要获取下面这个 2d-数组 `b` 中值为 6 的元素，则需要使用 `b[1][2]`;

```
>>> b = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> b[1]
[4, 5, 6]
>>> b[1][2]
6
```

我们可以理解成先访问 `b[1]`, 再将 `b[1]` 看成单独的一部分，去访问 `b[1]` 中索引为 2 的元素。

二维情况可以类比成我们在矩阵 M 中按照先行后列的顺序去获取元素——

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad M_{(1,2)} = 6$$

在更高维度的情况下，再用专门的 “标量”，“向量”，“矩阵”... 术语去定义结构是很不现实的。

- 因此在数学中提供了 n 维张量的概念，对应地，NumPy 中提供了 n 维数组；
- n 维张量和 n 维数组中的 n 则表明从中获取元素需要提供 n 个索引值。

数学	计算机科学	获取值所需标量索引数量
标量（scalar）	数字（number）	0
向量（vector）	数组（array）	1
矩阵（matrix）	2 维数组（2d-array）	2
n 维张量（nd-tensor）	n 维数组（nd-array）	n

现在我们已经可以忘掉上面这些术语，统一用 n 来确定 Tensor 维度的数量。

因此我们可以这样理解：

- 一个标量是一个 0 维 Tensor;
- 一个向量是一个 1 维 Tensor;
- 一个矩阵是一个 2 维 Tensor;
- 一个 n 维数组是一个 n 维 Tensor.

而在访问 n 维 Tensor （假定为 T ）的特定某个元素时，可以使用如下语法：

$$T[i_1][i_2]..[i_n]$$

即我们要提供 $i_1, i_2, \ldots, i_n$ 共 n 个索引值，每次索引降低一个维度，最终得到 0 维数字（标量）。

比如我们得知要找的某个人住在某小区的 23 号楼 3 单元 902 室，因此我们需要访问 `court[23][3][9][2]`;

> **参见**
>
> 实际上，对于 Tensor 和多维数组，有着更加高效的索引方法，可参考 在多个维度进行索引 的用法。

> **注解**
>
> 深度学习领域的 Tensor 其实就是一个多维数组（N 维数组）。

使用切片获取部分元素

前面我们展示了如何访问单个的元素，另一种比较常见的情况是需要对部分元素进行访问。

与 Python 一致，我们可以使用切片（Slicing）操作符来访问和修改 Tensor 对象中的部分元素：

```
>>> a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a[2:8:2]
[2, 4, 6]
```

观察上面的例子，我们通过 `:` 符号进行了切片操作，语法为 `start:stop:step`, 对应起始索引、终止索引和步长。 这种写法实际上在背后为我们生成了一个切片对象 `slice(start:stop:step)`, 二者是等价的：

```
>>> myslice = slice(2, 8, 2)
>>> a[myslice]
[2, 4, 6]
```

> **注解**
>
> - `start`, `stop`, `step` 也可以是负数，意味着索引变化顺序与默认情况相反。
> - `start` 和 `stop` 索引区间是左闭右开的 `[start, stop)` 形式，即 `a[stop]` 本身不在切片范围之内。
> - 这个设计其实与基于零的索引方式对应，该设计的好处有很多： 当只有最后一个位置信息时，我们也可以快速计算出切片和区间内有几个元素; 同理使用 `stop` 减去 `start` 可以快速计算出切片和区间的长度，不容易混淆； 与此同时，我们可以用 `a[:i]` 和 `a[i:]` 获得原始数据分割后不重叠的两部分。

> **参见**
>
> 计算机科学家，Edsger W. Dijkstra 教授在《Why numbering should start at zero 》 中的内容为基于 0 的下标以及左闭右开的区间习惯进行了很好的解释。

另外，切片语法中的部分元素可以被省略：

- 如果下标运算符中没有任何冒号运算符如 `a[i]`, 则返回与该索引位置对应的单个元素；
- 如果下标运算符中只有一个冒号运算符，则需要根据不同的写法进行判断：
  - 如果为 `a[start:]`, 则表明从 `start` 位置往后的所有项都被提取；
  - 如果为 `a[:stop]`, 则表明从 `stop` 位置往前的所有项都被提取；

    - 如果为 `a[start:stop]`, 则表明从 `start` 到 `stop` 的所有项将被提取;
- 如果没有指定 `step`, 则默认提取切片范围内的所有项目。

多维数组也支持使用切片语法:

```
>>> b = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> b[0:2]
[[1, 2, 3], [4, 5, 6]]
```

此时可以将其当作是一个一维数组去理解，里面的每个元素又是一维数组:

```
>>> a1 = [1, 2, 3]
>>> a2 = [4, 5, 6]
>>> a3 = [7, 8, 9]
>>> b = [a1, a2, a3]
>>> b[0:2]
[[1, 2, 3], [4, 5, 6]]
>>> [a1, a2]
[[1, 2, 3], [4, 5, 6]]
```

我们这里仅仅对最外面这一层进行了索引，在 Tensor 元素索引 中会讲解更复杂的情况。

参见

使用切片索引可以从 Tensor 中访问部分元素，但有些时候我们希望获得的部分元素是不连续的， 而是几个特定位置元素的组合，此时可以使用 数组索引 。

接下来：Tensor 基础属性

通过本小节的内容，用户能够掌握最基本的 Tensor 概念。

为了方便初学者学习和过渡，在上面的代码示例中，我们一直在使用 Python 的 `list` 来举例， 以表明 MegEngine Tensor 数据结构与 Python 嵌套列表设计的一致性，但实际上二者还是存在着一定的区别。

我们再举一些例子，请你尝试猜测一下输出:

Python nested list

```
>>> c = [[1, 2, 3],
>>>      [4, 5, 6],
>>>      [7, 8, 9]]
>>> c[1, 1]
```

MegEngine 2-d Tensor

```
>>> c = Tensor([[1, 2, 3],
>>>             [4, 5, 6],
>>>             [7, 8, 9]])
>>> c[1, 1]
```

Python 嵌套列表并不支持这种语法，你能猜测出在 `[]` 运算符中使用 `,` 的作用吗?

假设我们现在需要从下面这个 2 维 Tensor 中取出蓝色部分的元素，又需要如何做呢? （解释）

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad M_{(?,?)} = (4\ 5)$$

想要解答这些问题，你必须先理解 Tensor 的 Rank, Axes 与 Shape 属性 等有关概念， 更好地理解 Tensor 所具备的一些特点，接着从 Tensor 元素索引 的内容中找到答案。

参见

Tensor 数据类型
我们提到了 Tensor 中的每个元素的数据类型一致，如果你想要知道具体有哪些数据类型的 Tensor, 请参考 Tensor 数据类型 。

Tensor 所在设备
能够利用 GPU 设备进行高效运算是 MegEngine 相较于 NumPy 的优势，想要了解不同设备之间的区别， 请参考 Tensor 所在设备 。

Tensor 具象化举例
如果你目前对于 Tensor 的概念不够直观，可以参考 Tensor 具象化举例 。

Tensor 内存布局
一些有经验的开发者喜欢研究底层的细节，可以参考 Tensor 内存布局 。

Python 数据 API 标准联盟协会

MegEngine 中的许多 Tensor 标准 API 设计遵循了 Python 数据 API 标准联盟协会的倡导， 一些常见的实现尽可能地向 NumPy 靠近，更多细节可以参考 Consortium for Python Data API Standards .