Chapter 1

从张量开始

1.1 实际数据转为浮点数

1.2 张量：多维数组

1.2.1 张量的本质

Python 列表或数字元组是在内存中单独分配的 Python 对象的集合，如 Figure 1.1 左侧所示。另外，PyTorch 张量或 NumPy 数组通常是连续内存块的视图，这些内存块包含未装箱的 C 数字类型，而不是 Python 对象。在本例中，每个元素都是 32 位（4 字节）的浮点数，如 Figure 1.1 右侧所示。这意味着存储 1,000,000 个浮点数的一维张量将恰好需要 4,000,000 个连续字节，再加上元数据的小开销，如维度和数字类型。

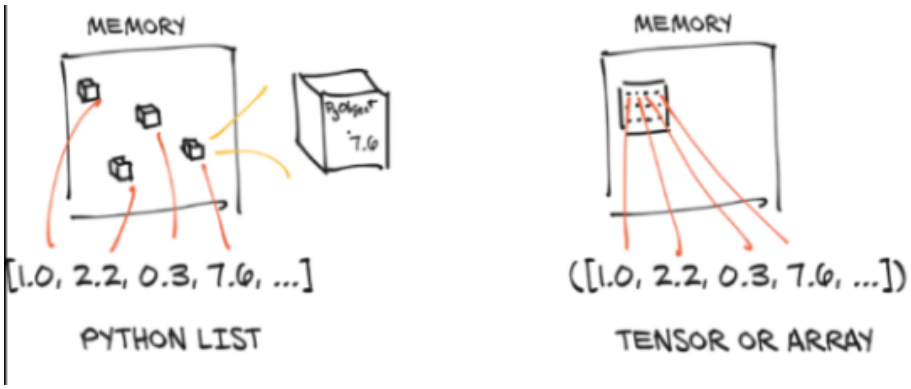

Figure 1.1: Python 列表（装箱）数值与张量或数组（非装箱）数值的对比

1.3 命名张量

张量的维度或坐标轴通常用来表示诸如像素位置或颜色通道的信息，这意味着当我们要把一个张量作为索引时，我们需要记住维度的顺序并按此顺序编写索引。在通过多个张量转换数据时，跟踪每个维度包含哪些数据可能容易出错。

PyTorch 1.3 将命名张量作为试验性的特性。张量工厂函数（诸如 tensor()和 rand()函数）有一个 names 参数，该参数是一个字符串序列。当我们已经有一个张量并且想要为其添加名称但不改变现有的名称时，我们可以对其调用 `refine_names()` 方法。与索引类似，省略号（...）允许你省略任意数量的维度。使用 `rename()` 兄弟方法，还可以覆盖或删除（通过传入 None）现有名称。

对于有 2 个输入的操作，除了常规维度检查，即检查张量维度是否相同，以及是否一个张量维度为 1 并且可以广播给另一个张量，PyTorch 还将检查张量名称。到目前为止，它还没有提供自动维度对齐功能，因此我们需要显式地进行此操作。`align_as()` 方法返回一个张量，其中添加了缺失的维度。

如果我们想在对命名的张量进行操作的函数之外使用张量，需要通过将这些张量重命名为 None 来删除它们的名称。

1.4 张量的元素类型

使用标准 Python 数字类型可能不是最优的，原因如下：

- Python 中的数字是对象。例如，一个浮点数在计算机上可能只需要 32 位来表示，而 Python会通过引用计数将它转换成一个完整的 Python 对象，等等。如果我们需要存储少量数值，采用装箱操作并不是问题，但是如果我们需要存储数百万的数据，采用装箱操作会非常低效。
- Python 中的列表属于对象的顺序集合，没有为有效地获取两个向量的点积或将向量求和而定义的操作。另外，Python 列表无法优化其内容在内存中的排列，因为它们是指向Python 对象的指针的可索引集合，这些对象可能是任何数据类型而不仅仅是数字。最后，Python 列表是一维的，尽管我们可创建元素为列表的列表，但这同样是非常低效的。
- Python 解释器与优化后的已编译的代码相比速度很慢。在大型数字类型的数据集合上执行数学运算，使用用编译过的更低级语言（如 C 语言）编写的优化代码可以快得多。

1.4.1 适合任何场合的 dtype

在神经网络中发生的计算通常是用 32 位浮点精度执行的。采用更高的精度，如 64 位，并不会提高模型精度，反而需要更多的内存和计算时间。16 位半精度浮点数的数据类型在标准 CPU 中并不存在，而是由现代 GPU 提供的。如果需要的话，可以切换到半精度来减少神经网络占用的空间，这样做对精度的影响也很小。

张量可以作为其他张量的索引，在这种情况下，PyTorch 期望索引张量为 64 位的整数。创建一个将整数作为参数的张量，例如使用 torch.tensor([2,2])，默认会创建一个 64 位的整数张量。因此，我们将把大部分时间用于处理 32 位浮点数和 64 位有符号整数。