二. 张量

```
#r "nuget: TorchSharp"
```

Installed Packages

- TorchSharp, 0.96.7

Tensors

在 TorchSharp 中，与所有深度学习一样，基本数据类型是“张量”，它只是一个广义矩阵。 在线性代数中，一维数组称为“向量”，而二维数组称为“矩阵”。 概括地说，张量只是一个 N 维数组。

请注意这里过度使用了“维度”这个词——在物理学中，一个具有三个元素的向量（一维）用于表示空间中的一个点，每个空间维度一个元素。 当我们在这些教程中谈到“维度”时，我们感兴趣的是张量维度的数量。

所以，让我们通过创建一些张量开始。

常量填充张量

最简单的张量创建原语只是将张量的所有元素初始化为 0 或 1。 传入的参数是每个维度的大小。 将第一个维度视为表格的行，将第二个维度视为列，然后您只需在头脑中概括事物即可。 在下面的示例中，为简单起见，我们将主要创建 3x4 矩阵。

需要注意的一点是，当您在笔记本单元格的末尾说“t”时，.NET Interactive 将显示对象及其字段等。 我们希望张量显示内容，并且有一个特殊版本的 ToString() 采用布尔值，不仅显示张量的大小和类型，还显示其内容。 位于笔记本顶部的特殊 .NET 交互式格式化程序（应该位于所有使用 TorchSharp 的笔记本的顶部）使用 TorString(true)。

注意： NET Interactive 适配

```
using TorchSharp;
using static TorchSharp.TensorExtensionMethods;
using Microsoft.DotNet.Interactive.Formatting;

var style = TensorStringStyle.Julia;

Formatter.SetPreferredMimeTypesFor(typeof(torch.Tensor), "text/plain");
Formatter.Register<torch.Tensor>((torch.Tensor x) => x.ToString(style,
newLine: "\n"));
```

```
var t = torch.ones(3,4);
t
```

```
[3x4], type = Float32, device = cpu
 1 1 1 1
 1 1 1 1
 1 1 1 1
```

如果您有两个以上的维度，则 ToString(t) 的特殊版本将尝试以对人类有意义的方式对其进行格式化：

```
torch.ones(2,4,4)
```

```
[2x4x4], type = Float32, device = cpu
[0,..,..] =
 1 1 1 1
 1 1 1 1
 1 1 1 1
 1 1 1 1

[1,..,..] =
 1 1 1 1
 1 1 1 1
 1 1 1 1
 1 1 1 1
```

如果您打算用其他地方的值填充张量，换句话说，因为您预先分配了它，那么有一个“空”工厂，它比使用其他任何东西都快。 这些值就是创建张量时在内存中找到的任何值。

```
torch.empty(4,4)
```

```
[4x4], type = Float32, device = cpu
 0 0 0 0
 0 0 0 0
 0 0 0 0
 0 0 0 0
```

您还可以从您想要的任何值创建张量：

```
torch.full(4,4,3.14f)
```

```
[4x4], type = Float32, device = cpu
 3.14 3.14 3.14 3.14
```

```
 3.14 3.14 3.14 3.14
 3.14 3.14 3.14 3.14
 3.14 3.14 3.14 3.14
```

有时，您希望在一个单元格中显示多个值。 为此，老式的打印：

```
Console.WriteLine(torch.zeros(4,4).ToString(style, newLine: "\n"));
Console.Write(torch.ones(4,4).ToString(style, newLine: "\n"));
```

```
[4x4], type = Float32, device = cpu
 0 0 0 0
 0 0 0 0
 0 0 0 0
 0 0 0 0

[4x4], type = Float32, device = cpu
 1 1 1 1
 1 1 1 1
 1 1 1 1
 1 1 1 1
```

所有这些输入都会变得非常乏味，因此我们添加了一个扩展方法 print()，它可以更快地输入。

```
torch.zeros(4,4).print(style);
torch.ones(4,4).print(style);
```

```
[4x4], type = Float32, device = cpu
 0 0 0 0
 0 0 0 0
 0 0 0 0
 0 0 0 0

[4x4], type = Float32, device = cpu
 1 1 1 1
 1 1 1 1
 1 1 1 1
 1 1 1 1
```

您可能已经注意到每个张量都有一个“type = Float32”属性。 这是 TorchSharp 张量类型的一个特点——元素类型没有出现在类型中，Tensor 不是 Tensor<T>。 之所以如此，是因为底层 C++/CUDA 运行时以这种方式表示张量，并且它也使得从 Python 移植代码变得更加容易。

您可能还注意到张量是使用工厂创建的，而不是构造函数。 此外，命名约定看起来一点也不像 .NET。 我们选择远离 .NET 约定，以便更轻松地从 Python 移植代码。 我们知道这会让一些人感到不安，也让一些人感到高兴，但这是我们经过长时间考虑后做出的决定。

无论如何，'Float32' 是默认值，但您也可以创建其他类型的张量，包括复杂张量：

```
torch.zeros(4,15, dtype: torch.int32)
```

```
[4x15], type = Int32, device = cpu
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

要访问张量的内容，请将其视为多维数组（请注意，维数也不是类型本身的一部分）。 当你这样做时，你会惊讶地发现索引运算符的结果是另一个张量，一个没有形状的张量——这就是 TorchSharp 表示标量值的方式。 在本教程的后面，我们将看到原因。 现在，只知道您必须使用函数提取值，基于您希望得到的类型。

在 PyTorch 中，有一个方法 '.item()' 用于此目的。 在 TorchSharp 中，它是一种模板化方法：

```
t = torch.zeros(4,4, dtype: torch.complex64)
```

```
[4x4], type = ComplexFloat32, device = cpu
 0 0 0 0
 0 0 0 0
 0 0 0 0
 0 0 0 0
```

```
t = torch.zeros(4,4, dtype: torch.int32);
Console.Write(t[0,0]);
t[0,0].item<int>()
```

```
[], type = Int32, device = cpu
```

0

```
t[0,0] = torch.tensor(35);
t
```

```
[4x4], type = Int32, device = cpu
 35 0 0 0
  0 0 0 0
  0 0 0 0
  0 0 0 0
```

```
torch.randn(3,4)
```

```
[3x4], type = Float32, device = cpu
   0.0466   1.5172 -0.74807  0.79056
 -0.48818  0.93328 -0.38119   0.8577
 -0.61721 -0.46137 -0.38967 -0.57855
```

```
torch.rand(3,4)
```

```
[3x4], type = Float32, device = cpu
 0.84476 0.59467 0.048663 0.88139
 0.27395 0.39467  0.43786   0.917
 0.40158 0.61339  0.46068 0.98693
```

```
(torch.rand(3,4) * 10 + 100)
```

```
[3x4], type = Float32, device = cpu
 109.11 102.62 100.73 102.38
 108.51 103.05 108.99 104.07
 103.72 102.65 100.47 102.18
```

```
torch.randint(10, (3,4))
```

```
[3x4], type = Int64, device = cpu
 0 7 1 7
 6 6 5 9
 3 2 9 9
```

```
torch.arange(3,19)
```

```
[16], type = Int64, device = cpu
 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
```

```
torch.arange(3.0f, 14.0f)
```

```
[11], type = Float32, device = cpu
 3 4 5 6 7 8 9 10 11 12 13
```

```
torch.arange(3.0f, 5.0f, step: 0.1f)
```

```
[20], type = Float32, device = cpu
 3 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8
4.9
```

```
torch.arange(3.0f, 5.0f, step: 0.1f).reshape(4,5)
```

```
[4x5], type = Float32, device = cpu
   3 3.1 3.2 3.3 3.4
 3.5 3.6 3.7 3.8 3.9
   4 4.1 4.2 4.3 4.4
 4.5 4.6 4.7 4.8 4.9
```

```
torch.arange(3.0f, 5.0f, step: 0.1f).reshape(4,5).str(fltFormat: "0.00")
```

```
[4x5], type = Float32, device = cpu
 3.00 3.10 3.20 3.30 3.40
 3.50 3.60 3.70 3.80 3.90
 4.00 4.10 4.20 4.30 4.40
 4.50 4.60 4.70 4.80 4.90
```

```
t = torch.rand(3,4,4,4);
t.reshape(12, 4, 4).ToString()
```

```
[12x4x4], type = Float32, device = cpu
```

```
t.reshape(-1,4,4).ToString()
```

```
[12x4x4], type = Float32, device = cpu
```

```
t.reshape(4,-1,6).ToString()
```

```
[4x8x6], type = Float32, device = cpu
```

```
t = torch.arange(3.0f, 5.0f, step: 0.1f).reshape(2,2,5);

// The overall shape of the tensor:
t.shape
```

index	value
0	2
1	2
2	5

```
t.ndim
```

3

```
t.numel()
```

20

```
t.cpu()
```

```
[2x2x5], type = Float32, device = cpu
[0,..,..] =
   3 3.1 3.2 3.3 3.4
 3.5 3.6 3.7 3.8 3.9

[1,..,..] =
   4 4.1 4.2 4.3 4.4
 4.5 4.6 4.7 4.8 4.9
```

```
torch.arange(3.0f, 5.0f, step: 0.1f).reshape(4,5).print();
torch.arange(3.0f, 5.0f, step: 0.1f).reshape(4,5).T.print();
```

```
[4x5], type = Float32, device = cpu
   3 3.1 3.2 3.3 3.4
 3.5 3.6 3.7 3.8 3.9
   4 4.1 4.2 4.3 4.4
 4.5 4.6 4.7 4.8 4.9

[5x4], type = Float32, device = cpu
   3 3.5   4 4.5
 3.1 3.6 4.1 4.6
 3.2 3.7 4.2 4.7
 3.3 3.8 4.3 4.8
 3.4 3.9 4.4 4.9
```

```
torch.arange(3.0f, 5.0f, step: 0.1f).reshape(5,4).print(fltFormat: "0.00");
torch.arange(3.0f, 5.0f, step: 0.1f).reshape(4,5).T.print(fltFormat:
"0.00");
```

```
[5x4], type = Float32, device = cpu
 3.00 3.10 3.20 3.30
 3.40 3.50 3.60 3.70
 3.80 3.90 4.00 4.10
 4.20 4.30 4.40 4.50
 4.60 4.70 4.80 4.90

[5x4], type = Float32, device = cpu
```

```
 3.00 3.50 4.00 4.50
 3.10 3.60 4.10 4.60
 3.20 3.70 4.20 4.70
 3.30 3.80 4.30 4.80
 3.40 3.90 4.40 4.90
```

```
torch.Tensor.TotalCount
```

638

```
torch.Tensor.PeakCount
```

3812

```
// Clone a tensor:
var s = t.clone();
s[0,0,1] = torch.tensor(375);
s.print();
t
```

```
[2x2x5], type = Float32, device = cpu
[0,..,..] =
   3 375 3.2 3.3 3.4
 3.5 3.6 3.7 3.8 3.9

[1,..,..] =
   4 4.1 4.2 4.3 4.4
 4.5 4.6 4.7 4.8 4.9




[2x2x5], type = Float32, device = cpu
[0,..,..] =
   3 3.1 3.2 3.3 3.4
 3.5 3.6 3.7 3.8 3.9

[1,..,..] =
   4 4.1 4.2 4.3 4.4
 4.5 4.6 4.7 4.8 4.9
```

```
var a = t.alias();
a[0,0,1] = torch.tensor(250);
t
```

```
[2x2x5], type = Float32, device = cpu
[0,..,..] =
   3 250 3.2 3.3 3.4
 3.5 3.6 3.7 3.8 3.9

[1,..,..] =
   4 4.1 4.2 4.3 4.4
 4.5 4.6 4.7 4.8 4.9
```

```
var ct = torch.rand(3,4,dtype:torch.ScalarType.ComplexFloat32);
ct.str(fltFormat:"0.00")
```

```
[3x4], type = ComplexFloat32, device = cpu
 0.92+0.79i 0.71+0.65i 0.54+0.36i 0.30+0.41i
 0.92+0.85i 0.97+0.54i 0.47+0.49i 0.21+0.84i
 0.50+0.64i 0.01+0.94i 0.82+0.09i 0.09+0.25i
```

```
ct.real
```

```
[3x4], type = Float32, device = cpu
 0.91828  0.71431 0.54335  0.30379
 0.91944  0.96949 0.46685  0.20997
 0.49879 0.008265 0.81813 0.087883
```

```
ct.imag
```

```
[3x4], type = Float32, device = cpu
 0.79254 0.65346  0.36257 0.40828
 0.84718 0.53583  0.48775 0.83911
 0.63595 0.93814 0.085096 0.25157
```

```
torch.ones(3,4,dtype:torch.ScalarType.ComplexFloat32)
```

```
[3x4], type = ComplexFloat32, device = cpu
 1 1 1 1
 1 1 1 1
 1 1 1 1
```