Тензоры

https://colab.research.google.com/drive/1VHdqejoDpzC5ra6plw2k5qWa9aWjlGT_#scrollTo=bHLpzNn

Тензоры — это фундаментальный строительный блок машинного обучения.

Их работа — представлять данные в числовом виде.

Создание тензоров

Скаляр

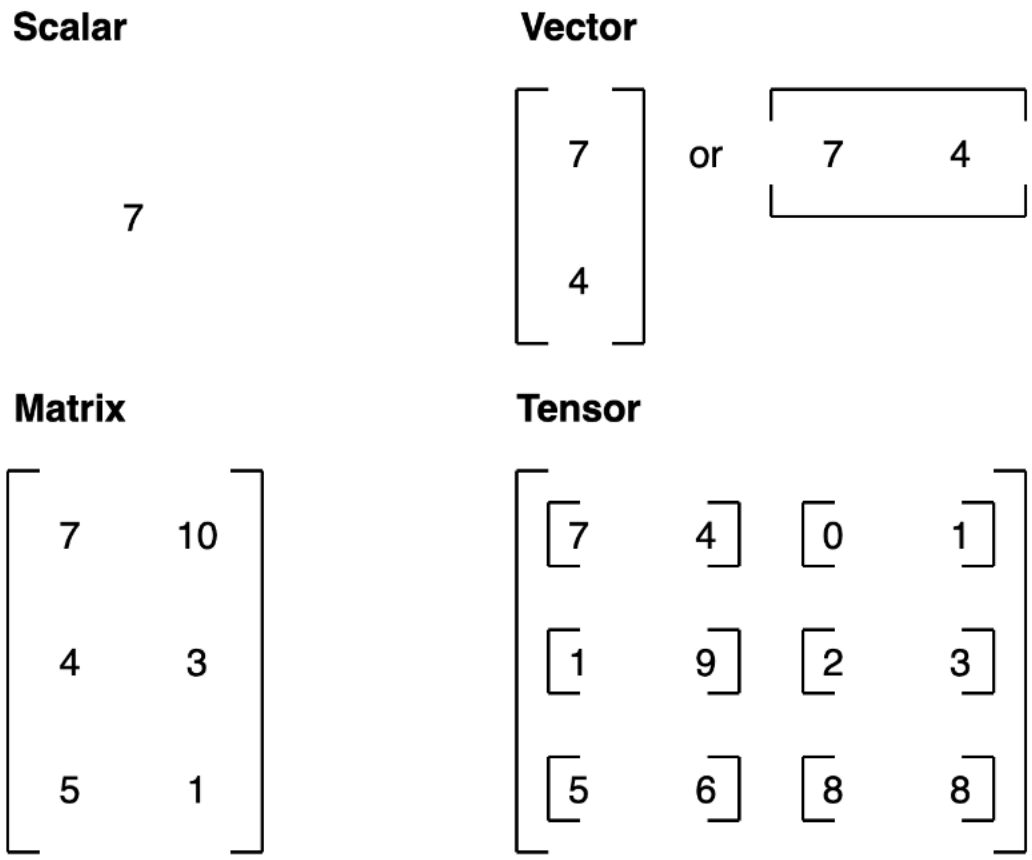
Скаляр — это одно число, и, говоря тензорным языком, это тензор нулевой размерности

```
scalar = torch.tensor(7)
#out: tensor(7)
```

Мы можем проверить размеры тензора, используя `ndim` атрибут.

```
scalar.ndim
#out: 0
```

Имя	Что это такое?	Количество измерений	Нижний или верхний (обычно/пример)
скаляр	одно число	0	Ниже (`a`)
вектор	число с указанием направления (например, скорость ветра с указанием направления), но может также иметь множество других чисел	1	Ниже (`y`)
матрица	двумерный массив чисел	2	Верхний (`Q`)
тензор	n-мерный массив чисел	может быть любым числом, 0-мерный тензор является скаляром, 1-мерный тензор является вектором	Верхний (`X`)



Тензор с рандомными числами:

```
random_tensor = torch.rand(size=(3, 4))
random_tensor
#out: (tensor([[0.6541, 0.4807, 0.2162, 0.6168],
         [0.4428, 0.6608, 0.6194, 0.8620],
         [0.2795, 0.6055, 0.4958, 0.5483]])
```

Основные мат.операции:

- Добавление
- Вычитание
- Умножение (поэлементное)
- Разделение
- Умножение матрицы

(+, -, *, /, @)

Нахождение min, max, avg, etc

```
x  =  torch . расположить в диапазоне ( 0 ,  100 ,  10 )
```

```
x.min()
x.max()
# x.mean() # это приведет к ошибке
x.type(torch.float32).mean() # не будет работать без типа данных float
x.sum()
```

Index min/max

```
torch.argmax()
torch.argmin()
```

Изменение формы

1. torch.reshape()

   Добавляет доп. измерение.

   ```
   x = torch.arange(1., 8.)
   x_reshaped= x.reshape(1, 7)
   x_reshaped, x_reshaped.shape
   #(tensor([[1., 2., 3., 4., 5., 6., 7.]]), torch.Size([1, 7]))
   ```

2. `torch.stack()`

   Разместить наш новый тензор поверх самого себя пять раз

   ```
   x_stacked = torch.stack([x, x, x, x], dim=0)
   #tensor([[5., 2., 3., 4., 5., 6., 7.],
           [5., 2., 3., 4., 5., 6., 7.],
           [5., 2., 3., 4., 5., 6., 7.],
           [5., 2., 3., 4., 5., 6., 7.]])
   #dim = 1 - транспонирование
   ```

Numpy

- `torch.from_numpy(ndarray)`
  - Массив NumPy -> Тензор PyTorch.
- `torch.Tensor.numpy()`
  - Тензор PyTorch -> Массив NumPy.

Если вы хотите преобразовать массив NumPy (float64) -> Тензор PyTorch (float64) -> Тензор PyTorch (float32), вы можете использовать

`tensor = torch.from_numpy(array).type(torch.float32)`.

Одинаковые рандомные значения

Но что, если вы хотите создать два случайных тензора с *одинаковыми* значениями.

Именно здесь появляется `torch.manual_seed(seed)`, где `seed` — целое число (например, `42`, но оно может быть чем угодно), что придает случайность.

```
random_seed = 43
torch.manual_seed(random_seed)
x1 = torch.rand(3, 4)
torch.manual_seed(random_seed)
x2 = torch.rand(3, 4)
x1, x2

#Вывод:
(tensor([[0.4540, 0.1965, 0.9210, 0.3462],
         [0.1481, 0.0858, 0.5909, 0.0659],
         [0.7476, 0.6253, 0.9392, 0.1338]]),
tensor([[0.4540, 0.1965, 0.9210, 0.3462],
         [0.1481, 0.0858, 0.5909, 0.0659],
         [0.7476, 0.6253, 0.9392, 0.1338]]))
```

Перемещение тензоров с ГПУ на ЦП и наоборот

Графические процессоры обеспечивают гораздо более быстрые численные вычисления, чем ЦП, а если графический процессор недоступен, из-за нашего **кода, не зависящего от устройства**, он будет работать на процессоре.

Давайте попробуем создать тензор и поместить его в графический процессор (если он доступен).

```
# Create tensor (default on CPU)
tensor = torch.tensor([1, 2, 3])

# Tensor not on GPU
print(tensor, tensor.device)

# Move tensor to GPU (if available)
tensor_on_gpu = tensor.to(device)
tensor_on_gpu

Вывод:
tensor([1, 2, 3]) cpu
tensor([1, 2, 3], device='cuda:0')
```