Basics of Tensors

Curiculum

1. Core data structure
2. Tensor Creation
3. Getting information from tensors

1. Core data structure (Scalars, Vector, Matrices)

1. Core data structure (Scalars, Vector, Matrices)

Scalars can be considered as a rank-0-tensor. Let's denote scalar value as $x \in \mathbb{R}$, where $\mathbb{R}$ is a set of real numbers.

Vectors can be introduced as a rank-1-tensor. Vectors belong to linear space (vector space), which is, in simple terms, a set of possible vectors of a specific length. A vector consisting of real-valued scalars ($x \in \mathbb{R}$) can be defined as ($y \in \mathbb{R}^n$), where y - vector value and $\mathbb{R}^n$ - n-dimensional real-number vector space. y_i - i_{th} vector element (scalar):

$$y = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

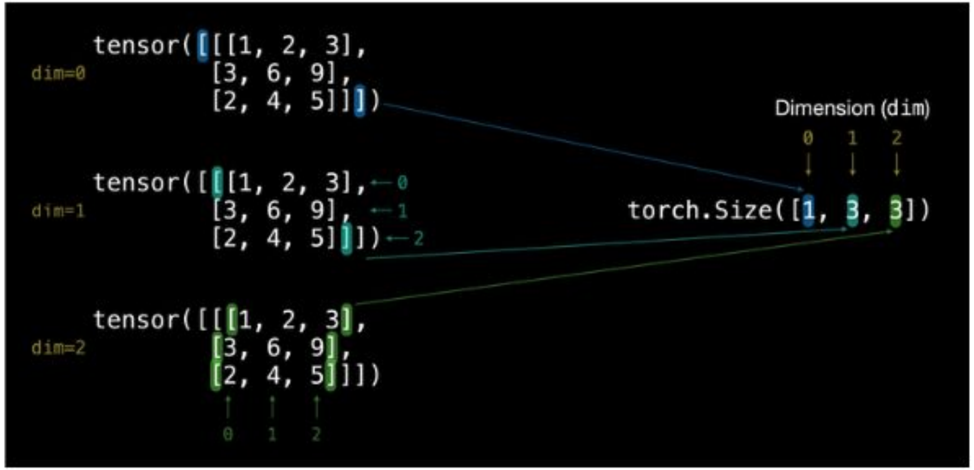
Matrices can be considered as a rank-2-tensor. A matrix of size $m \times n$, where $m, n \in \mathbb{N}$ (rows and columns number accordingly) consisting of real-valued scalars can be denoted as $A \in \mathbb{R}^{m \times n}$, where $\mathbb{R}^{m \times n}$ is a real-valued $m \times n$-dimensional vector space:

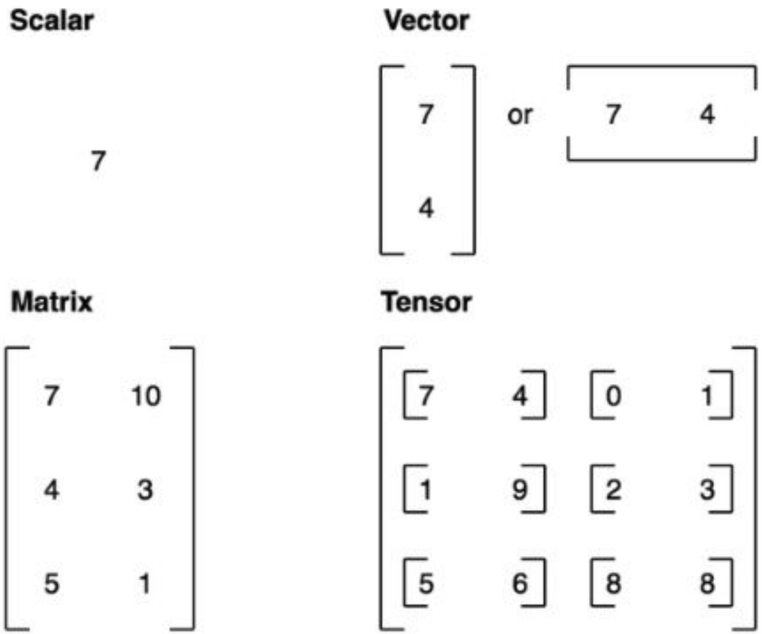
$$A = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & x_{m3} & \dots & x_{mn} \end{bmatrix}$$

Tensor is an entity with a defined number of dimensions called an order **(rank)**.

Scalar = a single number and in tensor-speak it's a zero dimension tensor.

1. Core data structure (Scalars, Vector, Matrices)





Note: You might've noticed me using lowercase letters for `scalar` and `vector` and uppercase letters for `MATRIX` and `TENSOR`. This was on purpose. In practice, you'll often see scalars and vectors denoted as lowercase letters such as `y` or `a`. And matrices and tensors denoted as uppercase letters such as `X` or `W`.

You also might notice the names martrix and tensor used interchangably. This is common. Since in PyTorch you're often dealing with `torch.Tensor`'s (hence the tensor name), however, the shape and dimensions of what's inside will dictate what it actually is.

1. Core data structure (Scalars, Vector, Matrices)

```python
matrix = torch.tensor([[7, 8],
                       [9, 10]])

print(matrix.ndim)
print(matrix.shape)
```

✓ 0.0s

```
2
torch.Size([2, 2])
```

```python
tensor = torch.tensor([[[1,2,3],
                        [4, 5, 6],
                        [2, 4 ,5]]])
print(tensor.ndim)
print(tensor.shape)
```

✓ 0.0s

```
3
torch.Size([1, 3, 3])
```

- You can tell the **number of dimensions** a tensor in PyTorch has by the number of square brackets on the outside and you only need to count one side
- Another important concept for tensors is their **shape attribute**. **The shape tells you how the elements inside them are arranged.**

2. Tensor Creation

2. Tensor Creation and Numpy interoperability

```python
empty = torch.empty(size=(3, 3))  # Tensor of shape 3x3 with uninitialized data
zeros = torch.zeros(size=(3, 4))
ones = torch.ones(size=(3, 4))
rand = torch.rand((3, 3))  # Tensor of shape 3x3 with values from uniform distribution in interval [0,1)
eye = torch.eye(5, 5)  # Returns Identity Matrix I, (I <-> Eye), matrix of shape 2x3
arange = torch.arange(start=0, end=5, step=1)  # Tensor [0, 1, 2, 3, 4], note, can also do: torch.arange(11)
linspace = torch.linspace(start=0.1, end=1, steps=10)  # x = [0.1, 0.2, ..., 1]
```

```python
print(empty, empty.dtype, empty.shape)
print(zeros, zeros.dtype, zeros.shape)
print(ones, ones.dtype, ones.shape)
print(rand, rand.dtype, rand.shape)
print(eye, eye.dtype, eye.shape)
print(arange, arange.dtype, arange.shape)
print(linspace, linspace.dtype, linspace.shape)
```

0.0s

```
tensor([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]]) torch.float32 torch.Size([3, 3])
tensor([[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]]) torch.float32 torch.Size([3, 4])
tensor([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]) torch.float32 torch.Size([3, 4])
tensor([[0.7787, 0.1542, 0.4201],
        [0.9481, 0.1338, 0.7235],
        [0.2711, 0.6595, 0.4061]]) torch.float32 torch.Size([3, 3])
tensor([[1., 0., 0., 0., 0.],
        [0., 1., 0., 0., 0.],
        [0., 0., 1., 0., 0.],
        [0., 0., 0., 1., 0.],
        [0., 0., 0., 0., 1.]]) torch.float32 torch.Size([5, 5])
tensor([0, 1, 2, 3, 4]) torch.int64 torch.Size([5])
tensor([0.1000, 0.2000, 0.3000, 0.4000, 0.5000, 0.6000, 0.7000, 0.8000, 0.9000,
        1.0000]) torch.float32 torch.Size([10])
```

2. Tensor Creation and Numpy interoperability

```python
# How to make initialized tensors to other types (int, float, double)
# These will work even if you're on CPU or CUDA!
tensor = torch.arange(4)  # [0, 1, 2, 3] Initialized as int64 by default
print(f"Converted Boolean: {tensor.bool()}")  # Converted to Boolean: 1 if nonzero
print(f"Converted int16 {tensor.short()}")  # Converted to int16
print(
    f"Converted int64 {tensor.long()}"
)  # Converted to int64 (This one is very important, used super often)
print(f"Converted float16 {tensor.half()}")  # Converted to float16
print(
    f"Converted float32 {tensor.float()}"
)  # Converted to float32 (This one is very important, used super often)
print(f"Converted float64 {tensor.double()}")  # Converted to float64$
```

✓ 0.0s

```
Converted Boolean: tensor([False,  True,  True,  True])
Converted int16 tensor([0, 1, 2, 3], dtype=torch.int16)
Converted int64 tensor([0, 1, 2, 3])
Converted float16 tensor([0., 1., 2., 3.], dtype=torch.float16)
Converted float32 tensor([0., 1., 2., 3.])
Converted float64 tensor([0., 1., 2., 3.], dtype=torch.float64)
```

3. Getting Information from tensors

3. Getting information from tensors

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

# Find out details about it
print(some_tensor)
print(f"Shape of tensor: {some_tensor.shape}")
print(f"Datatype of tensor: {some_tensor.dtype}")
print(f"Device tensor is stored on: {some_tensor.device}") # will default to CPU
```

[28] ✓ 0.0s

```
tensor([[0.2786, 0.0116, 0.5478, 0.3113],
        [0.9064, 0.6342, 0.8010, 0.2864],
        [0.6679, 0.6846, 0.7372, 0.5038]])
Shape of tensor: torch.Size([3, 4])
Datatype of tensor: torch.float32
Device tensor is stored on: cpu
```

- **shape** - what shape is the tensor? (some operations require specific shape rules)
- **dtype** - what datatype are the elements within the tensor stored in?
- **device** - what device is the tensor stored on? (usually GPU or CPU)

Note: When you run into issues in PyTorch, it's very often one to do with one of the three attributes above. So when the error messages show up, sing yourself a little song called "what, what, where":
"what shape are my tensors? what datatype are they and where are they stored? what shape, what datatype, where where where"