

Aim

- Understand the data types supported by tensors
- Type conversion
- Importance of device and grad while creating a tensor

Tensor Data Types

Tensors support wide range of data types

Category	Data Type	Aliases/Notes
Floating Point	32-bit floating point	<code>torch.float32</code> , <code>torch.float</code>
	64-bit floating point	<code>torch.float64</code> , <code>torch.double</code>
	16-bit floating point	<code>torch.float16</code> , <code>torch.half</code>
	16-bit floating point (bfloat16)	<code>torch.bfloat16</code>
	8-bit floating point (e4m3)	<code>torch.float8_e4m3fn</code> (limited support)
	8-bit floating point (e5m2)	<code>torch.float8_e5m2</code> (limited support)
Complex	32-bit complex	<code>torch.complex32</code> , <code>torch.chalf</code>
	64-bit complex	<code>torch.complex64</code> , <code>torch.cfloat</code>
	128-bit complex	<code>torch.complex128</code> , <code>torch.cdouble</code>
Unsigned Integer	8-bit integer	<code>torch.uint8</code>
	16-bit integer	<code>torch.uint16</code> (limited support)
	32-bit integer	<code>torch.uint32</code> (limited support)
	64-bit integer	<code>torch.uint64</code> (limited support)
Signed Integer	8-bit integer	<code>torch.int8</code>
	16-bit integer	<code>torch.int16</code> , <code>torch.short</code>
	32-bit integer	<code>torch.int32</code> , <code>torch.int</code>
	64-bit integer	<code>torch.int64</code> , <code>torch.long</code>
Boolean	Boolean	<code>torch.bool</code>

Quantized	Quantized 8-bit integer (unsigned)	<code>torch.quint8</code>
	Quantized 8-bit integer (signed)	<code>torch.qint8</code>
	Quantized 32-bit integer (signed)	<code>torch.qint32</code>
	Quantized 4-bit integer (unsigned)	<code>torch.quint4x2</code>

torch.device

A tensor can be created on a particular device (CPU, GPU). The `device` argument to the tensor constructor specifies where the tensor needs to be created.

Device options: CPU, CUDA, MPS

Notes:

- CUDA is a parallel computing platform and application programming interface model for Nvidia GPUs.
- MPS (Metal Performance Shaders) is Apple's framework for Apple Silicon GPUs.
- Two tensors need to be on the same device to be operated on.

```
In [1]: import torch as th
print(f"Is CUDA/GPU available? {th.cuda.is_available()}")
print(f"Is MPS enabled? {th.backends.mps.is_available()}")
print("MPS built:", th.backends.mps.is_built())
```

```
Is CUDA/GPU available? False
Is MPS enabled? True
MPS built: True
```

Creating tensors of specific type

Note: `requires_grad` flag specifies if gradient needs to be tracked for a tensor. Generally used for backpropagation

```
In [2]: float_32_vector_on_mps = th.tensor([1.0, 2.0, 3.0],
                                             dtype=th.float32, # Specify the dat
                                             device='mps', # Specify the device
                                             requires_grad=False) # Specify if g

float_16_vector_on_mps = th.tensor([1.0, 2.0, 3.0],
                                    dtype=th.float16, # Specify the dat
                                    device='mps', # Specify the device
                                    requires_grad=False) # Specify if g

second_float_32_vector_on_mps = th.tensor([4.0, 5.0, 6.0],
                                            dtype=th.float32,
                                            device='mps',
                                            requires_grad=False)

float_32_vector_on_cpu = th.tensor([1.0, 2.0, 3.0],
                                   dtype=th.float32,
                                   device='cpu',
                                   requires_grad=False)

print(f'Data Type of float_32_vector_on_mps: {float_32_vector_on_mps.d
print(f'Data Type of float_32_vector_on_cpu: {float_32_vector_on_cpu.d
print(f'Device of float_32_vector_on_mps: {float_32_vector_on_mps.devi
print(f'Device of float_32_vector_on_cpu: {float_32_vector_on_cpu.devi

Data Type of float_32_vector_on_mps: torch.float32
Data Type of float_32_vector_on_cpu: torch.float32
Device of float_32_vector_on_mps: mps:0
Device of float_32_vector_on_cpu: cpu
```

```
In [3]: float_32_vector_on_mps + second_float_32_vector_on_mps
```

```
Out[3]: tensor([5., 7., 9.], device='mps:0')
```

Note: Tensors on different devices cannot be operated on together.

```
In [6]: # Tensor on MPS cannot be added to a tensor on CPU
try:
    float_32_vector_on_mps + float_32_vector_on_cpu
except RuntimeError as e:
    print(e)
```

Expected all tensors to be on the same device, but found at least two devices, mps:0 and cpu!

Tensor Attributes

```
In [10]: def print_device_properties(tensor):
    print("-----")
    print(f'Device of tensor: {tensor.device}')
    print(f'Data Type of tensor: {tensor.dtype}')
    print(f'Gradient of tensor: {tensor.requires_grad}')
    print(f'Shape of tensor: {tensor.shape}')
    print(f'Is tensor on MPS: {tensor.is_mps}')
    print(f'Is tensor on CPU: {tensor.is_cpu}')
    print(f'Is tensor on CUDA: {tensor.is_cuda}')
    print("-----")

    print_device_properties(th.randn(3, 3, device='mps'))
    print_device_properties(th.randn(3, 3, device='cpu', requires_grad=True))
```

```
-----
Device of tensor: mps:0
Data Type of tensor: torch.float32
Gradient of tensor: False
Shape of tensor: torch.Size([3, 3])
Is tensor on MPS: True
Is tensor on CPU: False
Is tensor on CUDA: False
-----
```

```
-----
Device of tensor: cpu
Data Type of tensor: torch.float32
Gradient of tensor: True
Shape of tensor: torch.Size([3, 3])
Is tensor on MPS: False
Is tensor on CPU: True
Is tensor on CUDA: False
-----
```

Tensor type conversion

The `.to` method can be used to convert the type of a tensor. It can also be used to move the tensor from one device to another.

Tensors also have `.float()`, `.bool()` methods for type conversion.

```
In [5]: f16 = float_32_vector_on_mps.to(dtype=torch.float16) # Convert to float16
print(f'Data Type of float_32_vector_on_mps and f16: {f16.dtype} and {float_32_vector_on_mps.dtype}')

f32_on_mps = float_32_vector_on_cpu.to(device='mps') # Move to MPS
print(f'Device of float_32_vector_on_mps and f32_on_mps: {f32_on_mps.device} and {float_32_vector_on_cpu.device}')

# After moving float_32_vector_on_cpu to MPS, it can be added to float_32_vector_on_mps
print(float_32_vector_on_mps + f32_on_mps)

# Type converting using .bool()
print(f'Boolean value of float_32_vector_on_mps: {float_32_vector_on_mps.bool()}')

Data Type of float_32_vector_on_mps and f16: torch.float16 and torch.float32
Device of float_32_vector_on_mps and f32_on_mps: mps:0 and mps:0
tensor([2., 4., 6.], device='mps:0')
Boolean value of float_32_vector_on_mps: tensor([True, True, True], device='mps:0')
```