Tensors are the fundamental building blocks for performing mathematical operations in deep learning models.

Today, I will provide a comprehensive explanation with illustrative code examples.

Let's go! 🚀

Tensors are multi-dimensional arrays that form the backbone of numerical computing!

In PyTorch, creating tensors is a breeze!

You can initialize tensors from lists, zeros, ones, or even random values!

Initializing a Tensor

```python
import numpy as np

# From data stored in list
data = [[1, 2],[3, 4]]
x_data = torch.tensor(data)

# From a NumPy array
np_array = np.array(data)
x_np = torch.from_numpy(np_array)

# With random or constant values:
shape=(2, 2)
rand_tensor = torch.rand(shape)
ones_tensor = torch.ones(shape)
zeros_tensor = torch.zeros(shape)

print(f"Random Tensor: \n {rand_tensor} \n")
print(f"Ones Tensor: \n {ones_tensor} \n")
print(f"Zeros Tensor: \n {zeros_tensor}")
```

```
Random Tensor:
tensor([[0.5689, 0.9375],
        [0.5977, 0.9958]])

Ones Tensor:
tensor([[1., 1.],
        [1., 1.]])

Zeros Tensor:
tensor([[0., 0.],
        [0., 0.]])
```

Akshay 🚀
@akshay_pachaar

Every tensor has attributes like `dtype`, `shape`, and `device` which tells us about the nature of the tensor.

Tensors can live on CPU or GPU, and PyTorch makes it seamless to perform operations between them!

Check this out 👇

Attributes of a tensor

```python
import numpy as np
import torch

tensor = torch.rand(2,3)

print(f"Shape of tensor: {tensor.shape}")
print(f"Datatype of tensor: {tensor.dtype}")
print(f"Tensor is stored on: {tensor.device}")
```

Shape of tensor: torch.Size([2, 3])

Datatype of tensor: torch.float32

Tensor is stored on: cpu

Moving tensors to GPU

```python
# 👉 We move our tensor to the GPU if available
tensor = torch.rand(2, 3)
if torch.cuda.is_available():
    tensor = tensor.to("cuda")
```

Akshay 🚀
@akshay_pachaar

You can perform a variety of operations on tensors, like addition, element-wise multiplication, and matrix multiplication!

Operations on tensors

```python
import numpy as np
import torch

# 🔴Standard indexing and slicing just like NumPy
tensor = torch.ones(4, 4)
print(f"First row: {tensor[0]}")
print(f"First column: {tensor[:, 0]}")
print(f"Last column: {tensor[..., -1]}")
tensor[:,1] = 0
print(tensor)


# Arithmetic Ops
# Matrix multiplication
y1 = tensor.matmul(tensor.T)

# Element-wise product.
z1 = tensor * tensor

# Inplace Ops
# Notive the underscore(_) after add
tensor.add_(5)
print(tensor)
```

First row: tensor([1., 1., 1., 1.])

First column: tensor([1., 1., 1., 1.])

Last column: tensor([1., 1., 1., 1.])

```
tensor([[1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.]])
```

```
tensor([[6., 5., 6., 6.],
        [6., 5., 6., 6.],
        [6., 5., 6., 6.],
        [6., 5., 6., 6.]])
```

Akshay 🚀

@akshay_pachaar

Bridge with NumPy! 🌉

Tensors in PyTorch have a close relationship with NumPy arrays.

They share a lot of similarities, making transitioning between them a breeze! 🔄

Bridge with NumPy

Tensors on the CPU and NumPy arrays can share their underlying memory locations, and changing one will change the other.

```python
import torch
import numpy as np

# 🔴 Let's define a tensor 't' and
# and numpy array 'n' using the same tensor
t = torch.ones(5)
print(f"tensor: {t}")
n = t.numpy()
print(f"numpy array: {n}")

# 🟡 Let's add one to t and check how it
# affects n; check this out 👇

t.add_(1)
print(f"tensor: {t}")
print(f"numpy array: {n}")
```

tensor: tensor([1., 1., 1., 1., 1.])

numpy array: ([1., 1., 1., 1., 1.])

tensor: tensor([2., 2., 2., 2., 2.])

numpy array: ([2., 2., 2., 2., 2.])

Akshay 🚀
@akshay_pachaar

Bridge with NumPy

Tensors on the CPU and NumPy arrays can share their underlying memory locations, and changing one will change the other.

```
import torch
import numpy as np

# 🔴 Let's define a tensor 't' and
# and numpy array 'n' using the same tensor
t = torch.ones(5)
print(f"tensor: {t}")
n = t.numpy()
print(f"numpy array: {n}")

# 🟡 Let's add one to t and check how it
# affects n; check this out 👇

t.add_(1)
print(f"tensor: {t}")
print(f"numpy array: {n}")
```

tensor: tensor([1., 1., 1., 1., 1.])

numpy array: ([1., 1., 1., 1., 1.])

tensor: tensor([2., 2., 2., 2., 2.])

numpy array: ([2., 2., 2., 2., 2.])

Akshay 🚀

@akshay_pachaar