Understanding Tensors in PyTorch

Sayan CHAKI

March 27, 2025

Outline

- What are Tensors?
- Creating Tensors
- Tensor Operations
- Device Management
- Advanced Tensor Manipulation

What are Tensors?

- Fundamental data structure in PyTorch
- Multidimensional arrays similar to NumPy arrays
- Support GPU acceleration and automatic differentiation

```
import torch

# 0D tensor (scalar)
scalar = torch.tensor(5)

# 1D tensor (vector)
vector = torch.tensor([1, 2, 3])

# 2D tensor (matrix)
matrix = torch.tensor([[1, 2], [3, 4]])

# Multidimensional tensor
tensor_3d = torch.zeros(3, 4, 5)
```

Creating Tensors

Common Tensor Creation Methods:

- `torch.zeros()`
- `torch.ones()`
- `torch.rand()`
- `torch.randn()`

```
# Zeros tensor
zeros = torch.zeros(3, 3)

# Random tensor
random_uniform = torch.rand(2, 2)

# Random normal distribution
random_normal = torch.randn(3, 3)

# Tensor from list
custom = torch.tensor([1, 2, 3])

# Tensor type conversion
float_tensor = torch.FloatTensor([1, 2, 3])
```

Tensor Operations

```python
# Basic arithmetic
a = torch.tensor([1, 2, 3])
b = torch.tensor([4, 5, 6])

# Element-wise operations
addition = a + b
multiplication = a * b

# Matrix multiplication
matrix_a = torch.tensor([[1, 2], [3, 4]])
matrix_b = torch.tensor([[5, 6], [7, 8]])
matrix_mult = torch.matmul(matrix_a, matrix_b)

# Reshaping
tensor = torch.tensor([1, 2, 3, 4, 5, 6])
reshaped = tensor.view(2, 3)  # 2x3 matrix
```

- Supports element-wise and matrix operations

Device Management

```
# Check available device
print(torch.cuda.is_available())

# Create tensor on specific device
# CPU
cpu_tensor = torch.tensor([1, 2, 3])

# GPU (if available)
if torch.cuda.is_available():
    gpu_tensor = torch.tensor([1, 2, 3], device='cuda'
        )

    # Move tensor between devices
    cpu_tensor = cpu_tensor.to('cuda')
    gpu_tensor = gpu_tensor.to('cpu')
```

- Seamless CPU and GPU tensor management
- .to() method for device transfer

Advanced Tensor Manipulation

Advanced Operations:

- Indexing
- Slicing
- Reduction
- Broadcasting

```
# Indexing and Slicing
tensor = torch.tensor([[1, 2, 3],
                       [4, 5, 6],
                       [7, 8, 9]])
subset = tensor[1:, 1:]

# Reduction
mean_val = tensor.mean()
sum_val = tensor.sum()
```

Conclusion

- Tensors are core to PyTorch's deep learning functionality
- Flexible creation and manipulation
- Support for GPU acceleration
- Essential for building neural networks

Questions?