كلية علوم الحاسب
College of Computer Science

Talent Bootcamp Series -1

Slides contributed by: Hasan Shahid Kamal

What is PyTorch?

An open-source machine learning and deep learning framework.

- **Why PyTorch?**

Dominant in Research: 🔬Most widely used framework for new AI papers.

Intuitive & Pythonic: 🐍Feels like standard Python code.

Ease of Debugging: 🐛Dynamic computational graphs for easy troubleshooting.

High Performance: ⚡ Optimized for acceleration via NVIDIA CUDA (GPUs).

Industry Ready: 🏭Scalable for production deployment.

Introduction to Tensors

Tensors are the fundamental data structures that power deep learning. They provide a unified way to represent and manipulate numerical data across all dimensions, from simple numbers to complex multi-dimensional arrays used in modern AI systems.

Why use them?

Representing Multi-Dimensional Data

Computational Performance and Acceleration(GPU Integration)

Optimized for Neural Network Operations

Automatic Differentiation

CORE CONCEPTS

Understanding Tensor Hierarchy

Scalar (0D)

A single number representing a zero-dimensional point in space. Example: temperature reading of 72.

Vector (1D)

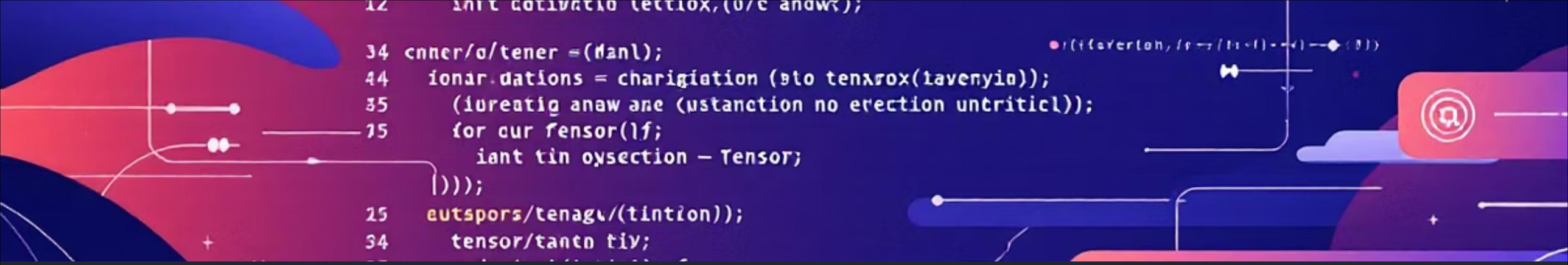
An ordered array of numbers forming a one-dimensional structure. Example: daily temperatures [65, 72, 68, 70].

Matrix (2D)

A rectangular grid organized in rows and columns. Example: spreadsheet of weekly temperature data.

Tensor (ND)

Multi-dimensional arrays that can represent complex data like images (Height × Width × RGB Channels) or video sequences.



PYTORCH BASICS

Defining Basic Tensors

Scalars

```
scalar = torch.tensor(7)
print(scalar.ndim)  # 0
print(scalar.item())  # 7
```

The `.item()` method extracts the Python number from a single-element tensor, essential for retrieving loss values during training.

Vectors

```
vector = torch.tensor([7, 7])
print(vector.ndim)  # 1
print(vector.shape)  # torch.Size([2])
```

Vector dimensions are determined by counting the levels of square brackets. Each additional bracket nesting increases the dimension by one.

Higher Dimensional Structures

As we move beyond vectors, tensors become powerful containers for complex data structures used throughout deep learning.

1

Matrices (2D)

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

Represents tabular data with rows and columns. Perfect for storing batches of feature vectors or weight matrices in neural networks.

2

3D Tensors

```
tensor_3d = torch.tensor([[[1, 2, 3],
                           [4, 5, 6],
                           [7, 8, 9]]])
```

Shape: [1, 3, 3]. Dimensions read from outer to inner. Common for representing sequences of matrices or single RGB images.

Pro Tip: Always think outer-to-inner when interpreting tensor shapes. A shape of [64, 3, 224, 224] means 64 images, each with 3 color channels, 224 pixels high, and 224 pixels wide.

INITIALIZATION

Creating Tensors from Scratch

Random Tensors

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

Critical for neural network weight initialization. Random values follow uniform distribution between 0 and 1.

Zeros & Ones

```
torch.zeros(size=(3, 4))
torch.ones(size=(3, 4))
```

Zeros initialize biases or create attention masks. Ones create identity operations or starting multiplicative values.

Ranges

```
torch.arange(start=0,
             end=10,
             step=1)
```

Generates sequences like [0,1,2...9]. Useful for creating position encodings or index tensors.

Tensor Datatypes & Precision

Choosing the right datatype balances computational efficiency with numerical accuracy—a critical decision in production deep learning systems.

1

float32 (Default)

The standard precision for most deep learning tasks. Offers excellent balance between speed, memory usage, and numerical stability.

2

float16 (Half Precision)

Runs 2-3x faster on modern GPUs with half the memory footprint. Commonly used in mixed-precision training to accelerate large model training.

3

float64 (Double Precision)

Maximum numerical precision but significantly slower. Rarely needed except for scientific computing requiring extreme accuracy.

Check Type

```
tensor.dtype
```

Check Device

```
tensor.device
```

Check Shape

```
tensor.shape
```

OPERATIONS

Tensor Arithmetic Operations

> Neural networks are fundamentally sophisticated calculators that perform millions of mathematical operations per second.

Element-wise Addition

```
result = tensor + 10
```

Adds the scalar value to every element in the tensor. Broadcasting rules apply automatically.

Element-wise Multiplication

```
result = tensor * 10
```

Multiplies each element by the scalar. Different from matrix multiplication—this is component-wise.

Element-wise Subtraction

```
result = tensor - 10
```

Subtracts the value from each element. Remember: operations aren't in-place unless you reassign!

Matrix Multiplication

The Golden Rule of Deep Learning

Matrix multiplication is the single most important operation in neural networks. Understanding its mechanics is essential for building and debugging AI models.

01

Inner Dimensions Must Match

To multiply matrices A and B, the number of columns in A must equal the number of rows in B. Example: (3,2) × (2,5) ✓ works, but (3,2) × (3,5) ✗ fails.

02

Output Shape From Outer Dimensions

The resulting matrix takes its shape from the outer dimensions. Multiplying (3,2) × (2,5) produces a (3,5) output matrix.

03

Use Transpose When Needed

If shapes don't align, use `tensor.T` to swap rows and columns. This flips dimensions: (3,2) becomes (2,3).

```
result = torch.matmul(tensor_A, tensor_B)
# Alternative: result = torch.mm(tensor_A, tensor_B)
```

STATISTICAL OPERATIONS

Aggregation Functions

Aggregation reduces many values into meaningful summary statistics—essential for calculating loss, monitoring training metrics, and making predictions.

Basic Aggregations

```
tensor.min()   # Minimum value
tensor.max()   # Maximum value
tensor.sum()   # Total sum
tensor.mean()  # Average (float only!)
```

Important: The `.mean()` function requires float datatypes. Convert with `tensor.float()` if needed.

Positional Finding

```
tensor.argmax()  # Index of max
tensor.argmin()  # Index of min
```

These functions return the *position* of extreme values rather than the values themselves. Critical for finding predicted class labels in classification tasks.

Reshaping & Common Errors

Reshape & View

Reshape changes shape while preserving total elements. **View** does the same but shares memory with the original.

Stack

Joins multiple tensors along a new dimension. Essential for batching individual samples together.

Squeeze & Unsqueeze

Squeeze removes size-1 dimensions. **Unsqueeze** adds a dimension at a specified position.

Permute

Rearranges dimension order. Critical for converting between image formats (e.g., channels-last to channels-first).

The Top 3 PyTorch Errors

Shape Mismatch

Attempting to multiply incompatible matrix dimensions. Always verify inner dimensions align.

Datatype Mismatch

Mixing integer and float tensors in operations. Cast explicitly with `.float()` or `.int()`.

Device Mismatch

Operating on tensors across CPU and GPU. Move tensors to the same device with `.to(device)`.

> **The Debug Checklist:** When errors occur, systematically check shape, dtype, and device. These three attributes solve 90% of tensor bugs.