# Module 1

**Tensors and Datasets**

**Tensors 1D**

## 📌 Overview of Tensors

Tensors are the foundational data structures used to construct and operate neural networks in PyTorch.

A neural network is fundamentally a mathematical function that accepts one or multiple inputs, processes them, and returns one or more outputs.

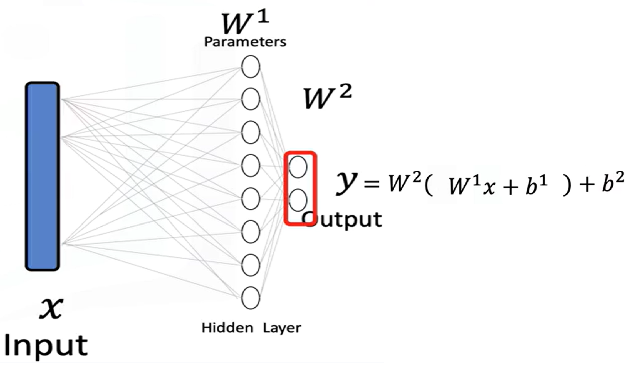
In PyTorch, this processing is performed using **tensor operations**, which are generalized versions of familiar mathematical operations like addition and multiplication.

### 🔹 Tensors as Building Blocks

PyTorch tensors serve as the unified representation for **inputs**, **outputs**, and **parameters** within a neural network. These tensors can represent vectors, matrices, or higher-dimensional data structures, depending on the application.

Tensor operations in PyTorch form the computational backbone of how data is manipulated and learned from in neural networks.

PyTorch tensors are a generalized form of **numbers and dimensional arrays** in Python.

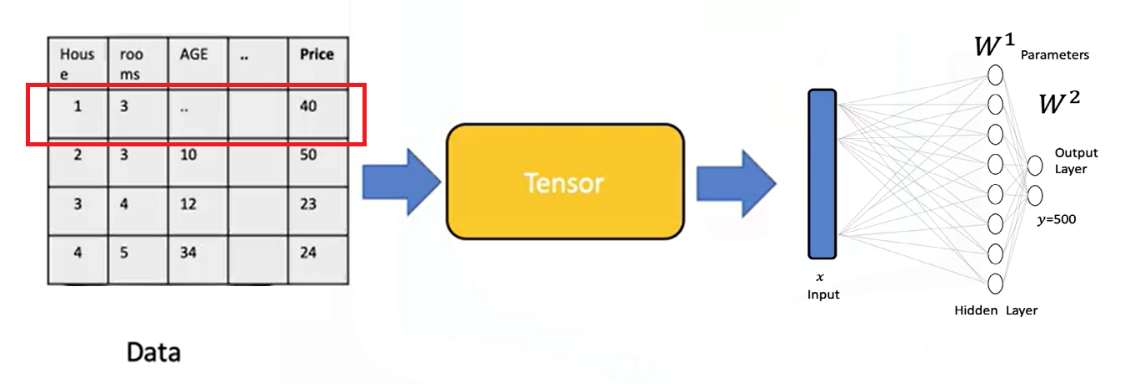
* The **input x** to a neural network is a tensor.
* The **output y** is also a tensor.
* The **parameters** of the model are tensors as well.
* Tensor operations allow the neural network to **transform inputs** into outputs during training and inference.

Neural networks use these tensor operations to apply **mathematical transformations**, often in the form of **vector and matrix operations**. These operations simulate the way real-world data is processed and are used throughout the course as the standard method for feeding data into neural models.

### 🔹 Examples of Tensors in Neural Networks

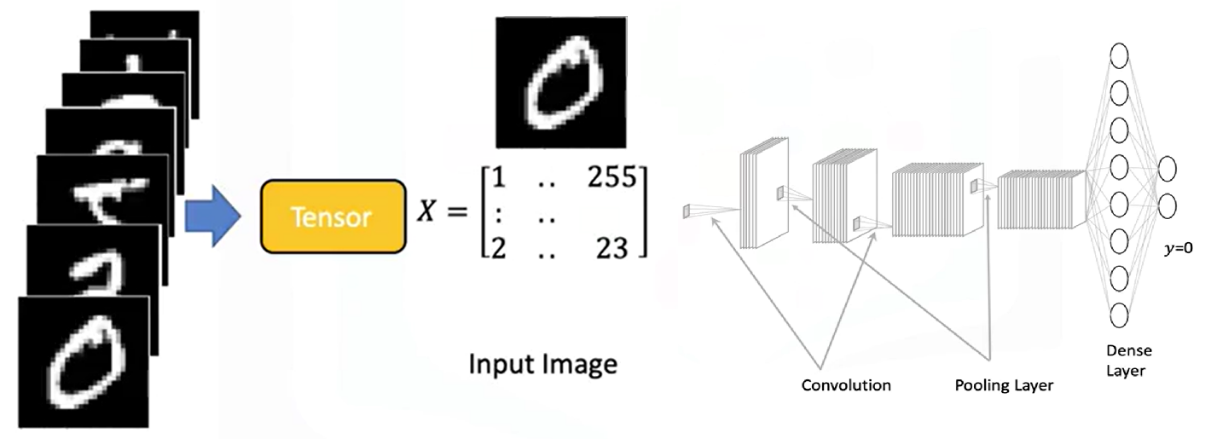
**Databases** can be treated as a series of tensors, where each row represents an input tensor (x) in a neural network.

A tensor is simply just a vector or a rectangular array consisting of numbers.



**Images** can be converted into 2D or 3D PyTorch tensors and used as input for classification tasks.

* + Each tensor of the input is simply a matrix or rectangular array.
  + Images are typically stored as arrays.
  + Neural networks can receive these as tensors and perform classification based on the processed values.
  + For instance, an image can be transformed into a tensor and classified as the digit **zero**.



### 🔹 Tensor Conversion and Compatibility

PyTorch tensors can be easily **converted to NumPy arrays**, and NumPy arrays can also be converted into PyTorch tensors.

This bidirectional conversion enables seamless operation within the **Python ecosystem** and allows integration with many existing Python libraries.

PyTorch also supports **GPU acceleration**, which is crucial for training large neural networks efficiently.

### 🔹 Parameters and Derivatives in PyTorch

**Parameters** in neural networks are specialized tensors that allow for the calculation of **gradients and derivatives**.

These gradients are essential for learning during training.

To enable gradient tracking, PyTorch tensors must be created with **requires\_grad=True**.

This setting allows PyTorch to automatically compute derivatives during backpropagation.

### 🔹 Dataset Class in PyTorch

PyTorch provides a **Dataset class** that simplifies working with large datasets.

Using this class enables efficient data handling, transformation, and loading.

It is especially useful when building neural networks that require batch processing or data augmentation.

### ✅ Takeaways

✅ PyTorch tensors are the core data structures used in building and training neural networks.

✅ Inputs, outputs, and model parameters are all represented as tensors.

✅ Tensor operations in PyTorch generalize familiar mathematical operations and are essential for transforming input data.

✅ Databases and images can be represented as tensors and processed within neural networks.

✅ PyTorch integrates seamlessly with NumPy and supports GPU acceleration for scalable training.

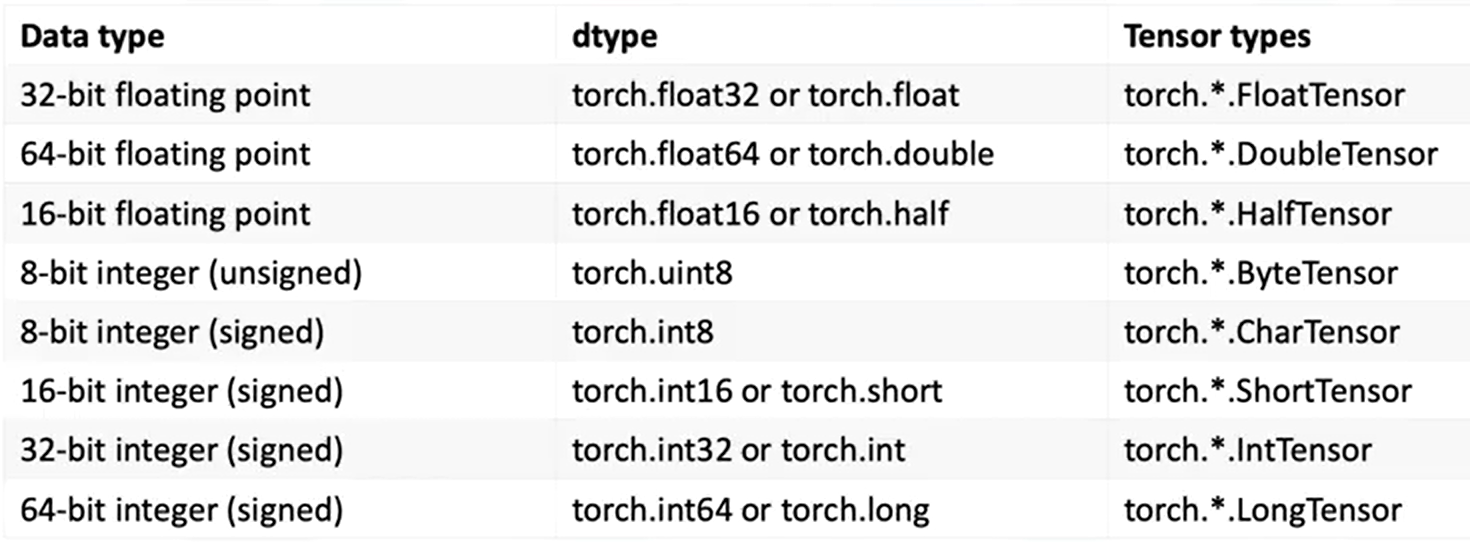
✅ Setting requires\_grad=True enables tensors to compute gradients, allowing for neural network training.

✅ The Dataset class simplifies data management and is essential for working with large-scale training data.

## 📌 Tensors 1D

### 🔹 Understanding 1D Tensors

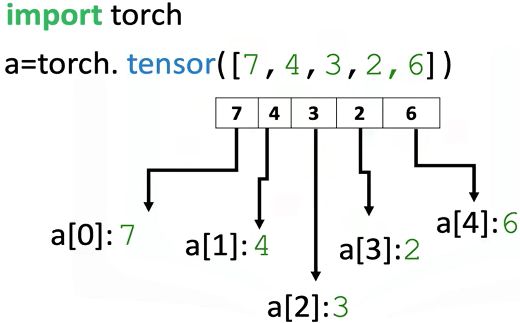
* A **0D tensor** represents a single number.
* A **1D tensor** is an array of numbers and can represent:
* A row in a dataset
* A vector
* A time series
* A tensor contains elements of a **single data type**, there is a variety of different tensor types depending the data type of the elements in the tensor, such as:
* float or double tensors (for real numbers)
* byte tensors (for 8-bit images and unsigned integers)



### 🔹 1D Tensor Operations

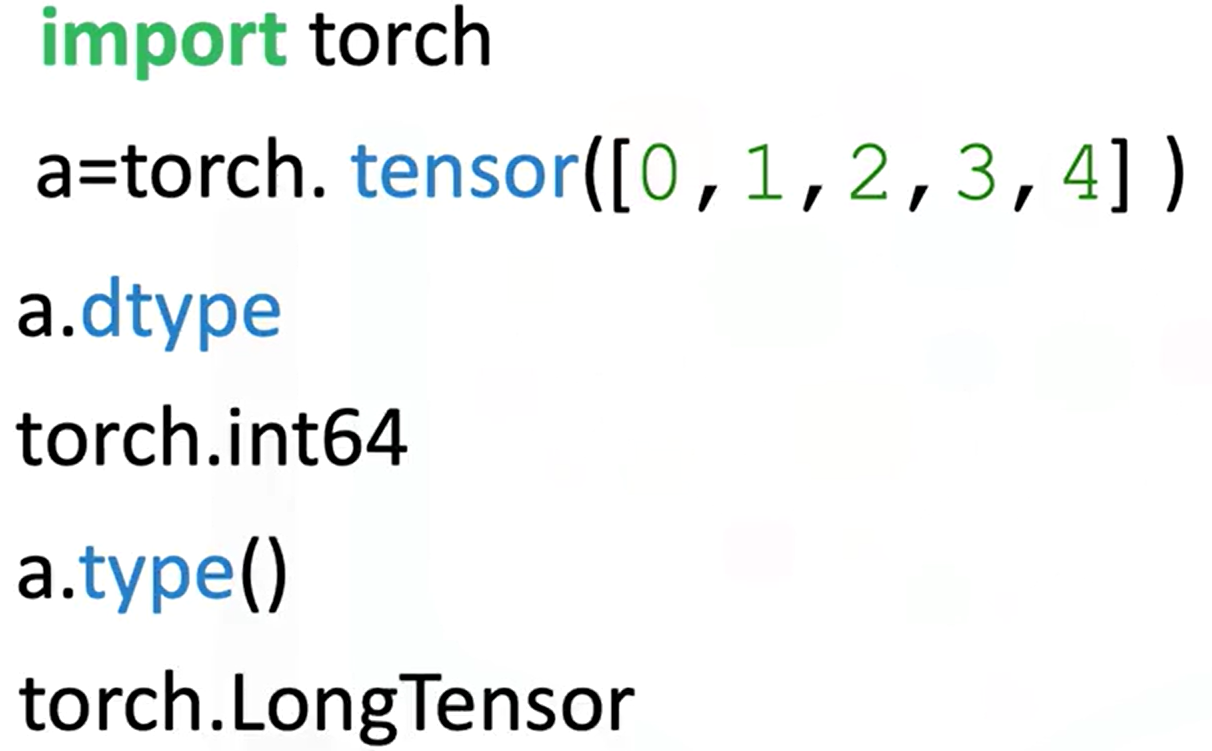
🔸 **Creating a tensor:**

* + Use **torch.tensor()** to convert the list into a tensor.
  + Data can be accessed via index.



🔸 **Tensor Type and Data Type:**

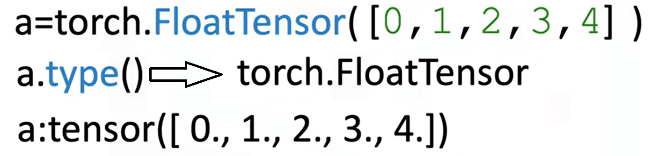
* + Use the **.dtype** attribute to identify the data type stored in a tensor.
  + Use **.type()** to identify the tensor type.



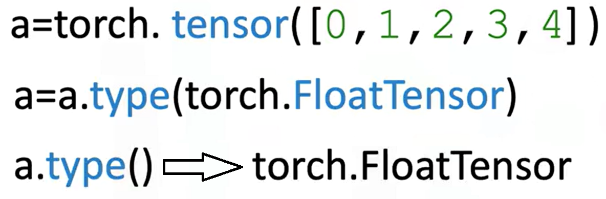
* + Explicitly set the data type using the **dtype** parameter.



* + Using classes like **torch.FloatTensor**.

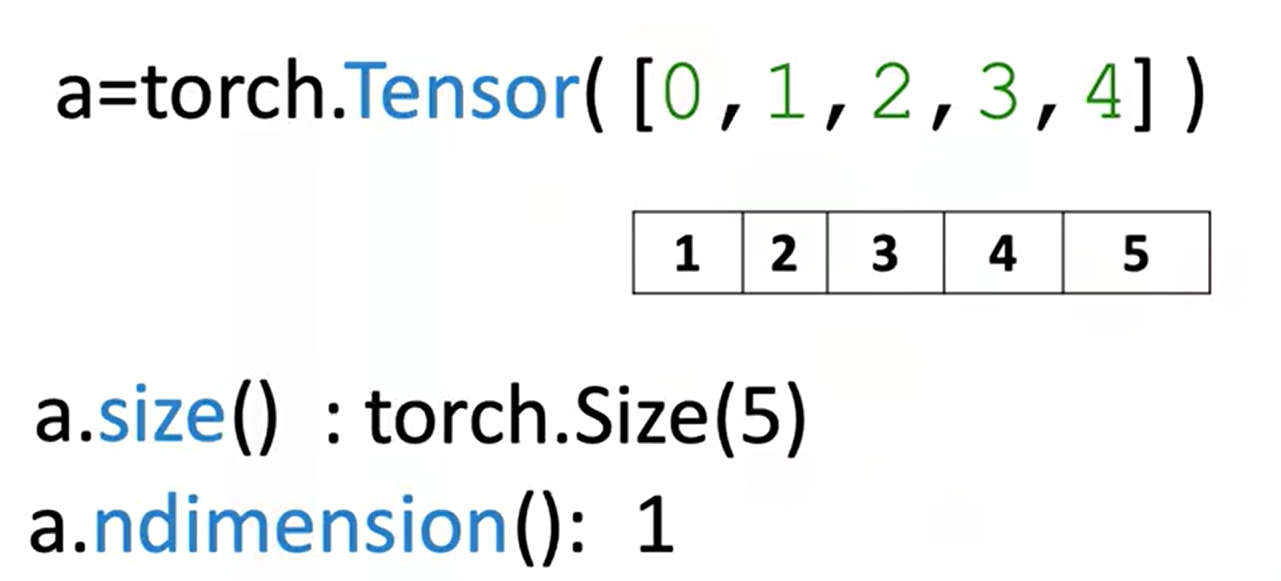


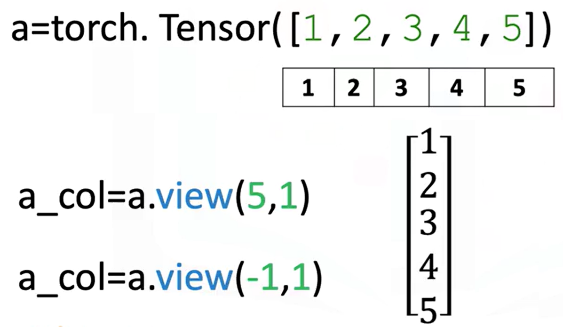
* + Use **.type(torch.FloatTensor)** to convert to a float tensor.



🔸 **Tensor Size and Shape:**

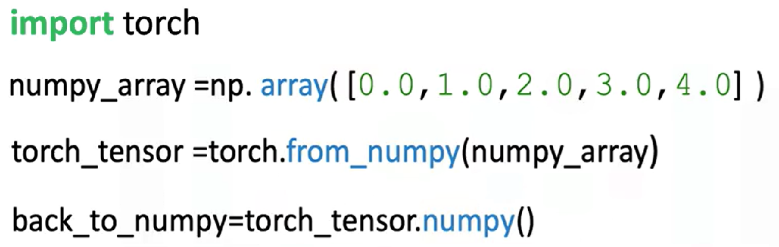
* + Use **.size()** to find the number of elements.
  + Use **.ndimension()** to find the number of dimensions (tensor rank).



* + Convert a 1D tensor to 2D using **.view(number\_rows, number\_cols)**:
* **view(5, 1)** turns a 1D tensor with 5 elements into a 2D column tensor.
* Use **view(-1,1)** to let PyTorch infer dimensions.

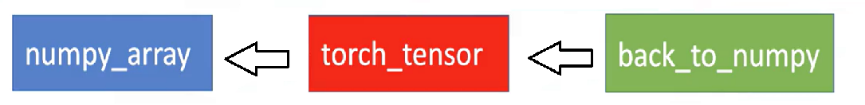
🔸 **Tensor Conversion with NumPy and Pandas**

* + Convert a NumPy array to a tensor with **torch.from\_numpy()**
  + Convert a tensor to a NumPy array using **.numpy()**

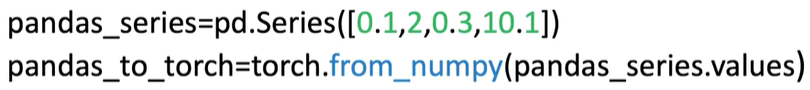


**⚠️ Memory sharing:**

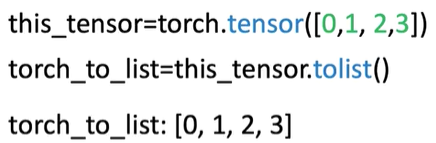
Modifying the original NumPy array affects the PyTorch tensor and vice versa.



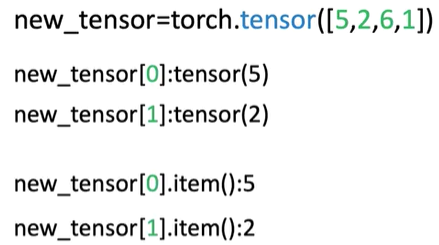
* + Convert Pandas series to tensor:
* Use **.values** to get the NumPy array
* Then apply **torch.from\_numpy()**

****

* + Convert tensor to list:
* Use **.tolist()** to get a Python list

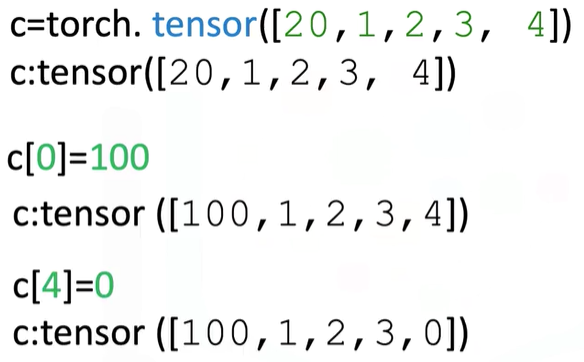


* + Convert tensor element to number:
* Use **.item()** to extract a Python number from a single-element tensor

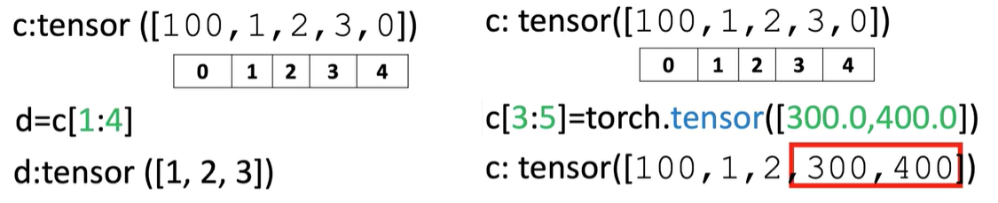


🔸 **Indexing and Slicing:**

* + Access tensor elements with index (e.g., a[0])
  + Assign new values to specific elements (e.g., a[0] = 100)



* + Slice a tensor like a list: a[1:3]
  + Assign values to slices (e.g., a[1:3] = [1, 2])

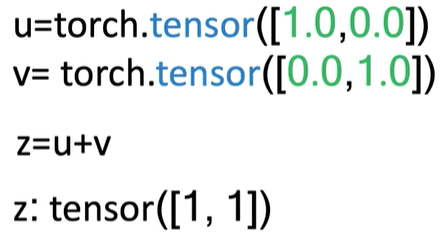


🔸 **Basic Tensor Operations:**

These operations are essential for building neural networks and understanding how tensors interact mathematically:

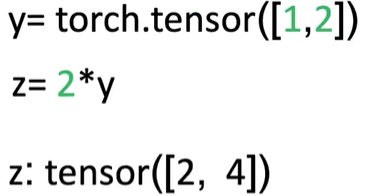
* + Vector Addition:

Combine two tensors element-wise.



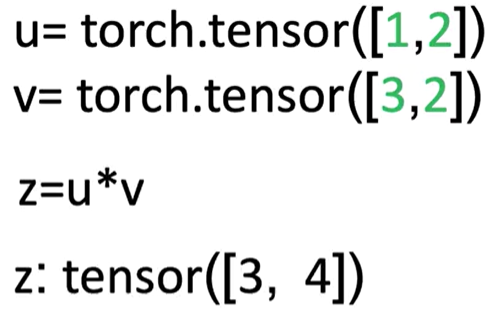
* + Scalar Multiplication:

Multiply each element of a tensor by a scalar.



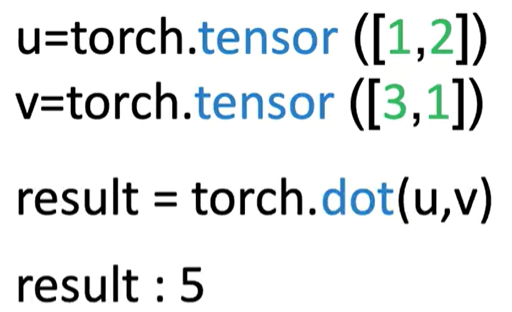
* + Hadamard Product (Element-wise Multiplication)

Multiply corresponding elements of two tensors.



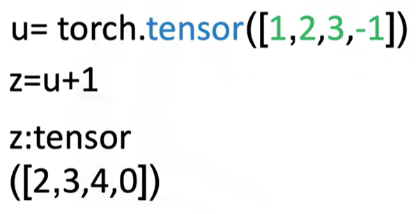
* + Dot Product

Produces a single number that measures similarity between two vectors.



* + Broadcasting

Adding a scalar to a tensor adds it to each element.

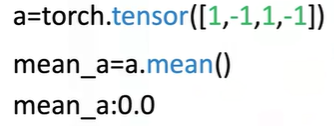


🔸 **Universal Functions:**

Apply operations across all elements:

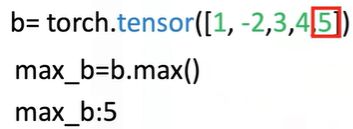
* + **a.mean()**:

Computes the average

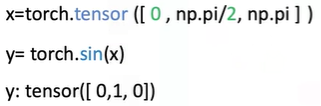


* + **b.max()**:

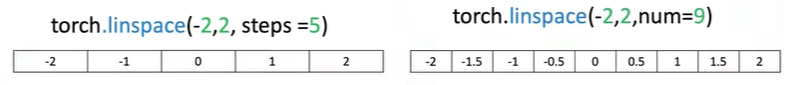
Returns the maximum value



* + Use functions like **torch.sin()** to apply to every element of a tensor

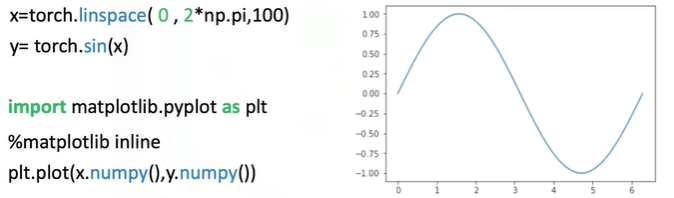


* + Use **torch.linspace(start, end, steps)** to generate evenly spaced values.



🔸 **Plotting with Tensors:**

* + Use **matplotlib.pyplot**
  + Use **%matplotlib inline** for inline notebook rendering
  + Convert tensors to NumPy before plotting: **.numpy()**



### ✅ Takeaways

✅ **1D tensors** are core structures for data representation in PyTorch.

✅ Tensors can be **easily created, indexed, sliced, and reshaped** using intuitive syntax.

✅ PyTorch supports **type casting**, **NumPy/Pandas conversion**, and **interoperability** with Python tools.

✅ Tensor operations include **vector arithmetic**, **dot product**, **broadcasting**, and **universal functions** like mean and max.

✅ PyTorch allows mathematical functions (like sine) to be applied element-wise, enabling visualization and numerical analysis.

✅ Tools like linespace and matplotlib can be combined with PyTorch tensors for **function plotting and visualization**.