Working with PyTorch Tensors

Janani Ravi
CO-FOUNDER, LOONYCORN
www.loonycorn.com

Overview

Tensor operations using PyTorch tensors

Interoperability with NumPy

Understanding PyTorch support for GPUs and CUDA

Working with tensors on GPU-enabled devices

Tensors in PyTorch

Tensor

The central unit of data in PyTorch. A tensor consists of a set of primitive values shaped into an array of any number of dimensions.

Data Is Represented as Tensors

Scalars are 0-D tensors

3, 6.7, "a"

Data Is Represented as Tensors

Vectors are 1-D tensors

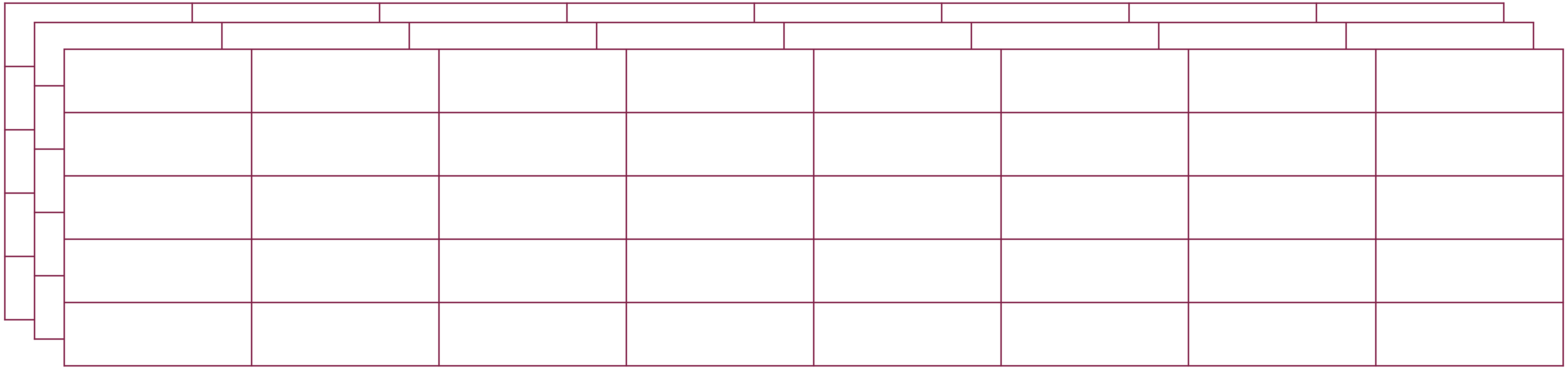
[1, 3, 5, 7, 9]

Data Is Represented as Tensors

Matrices are 2-D tensors

[[1, 3, 5],
 [7, 9, 11]]

Data Is Represented as Tensors



N-Dimensional matrices are N-D tensors

[[[1, 2], [3, 4], [5, 6]],
[[7, 8], [9, 10], [11, 12]]]

PyTorch Tensors have been architected to make optimal use of GPUs for massively parallel computations

Demo

Operations using PyTorch tensors

Demo

Conversions between PyTorch and NumPy

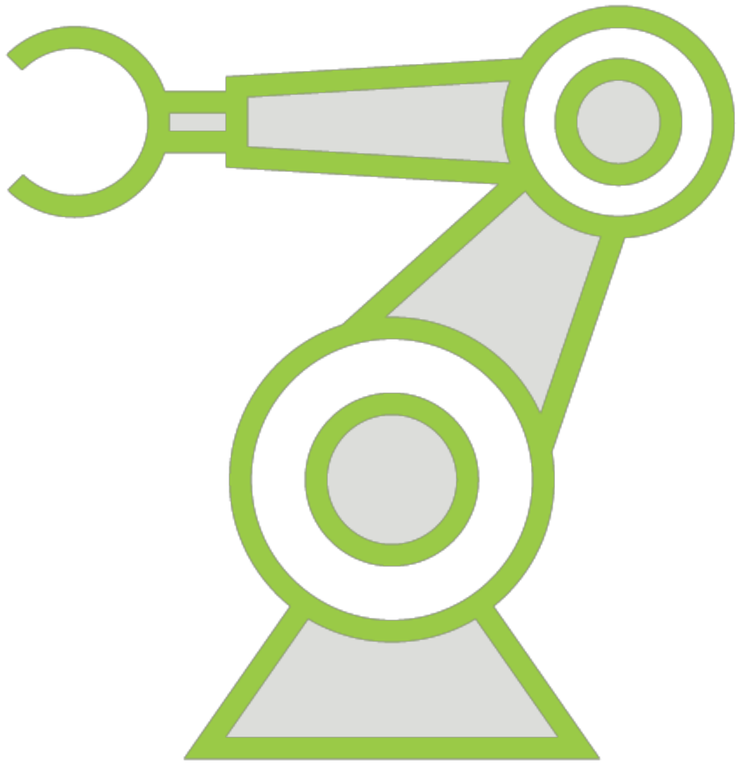
PyTorch, CUDA, and GPUs

GPU (Graphics Processing Unit)

Specialized chips with highly parallel architecture that makes them an order of magnitude faster than CPUs for some deep learning applications

PyTorch Tensors have been architected to make optimal use of GPUs for massively parallel computations

GPUs for ML

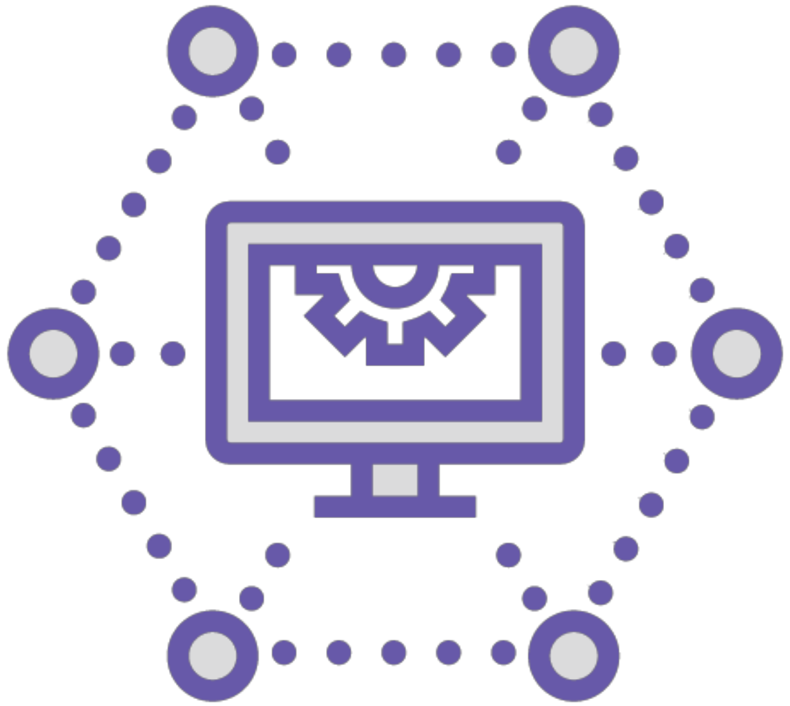


Usage of GPUs has gone far beyond video/graphics processing

Widely used in Big Data and Machine Learning applications

Speedup of 10-50X where parallelization yields big wins

CUDA



Nvidia is a major maker of GPUs

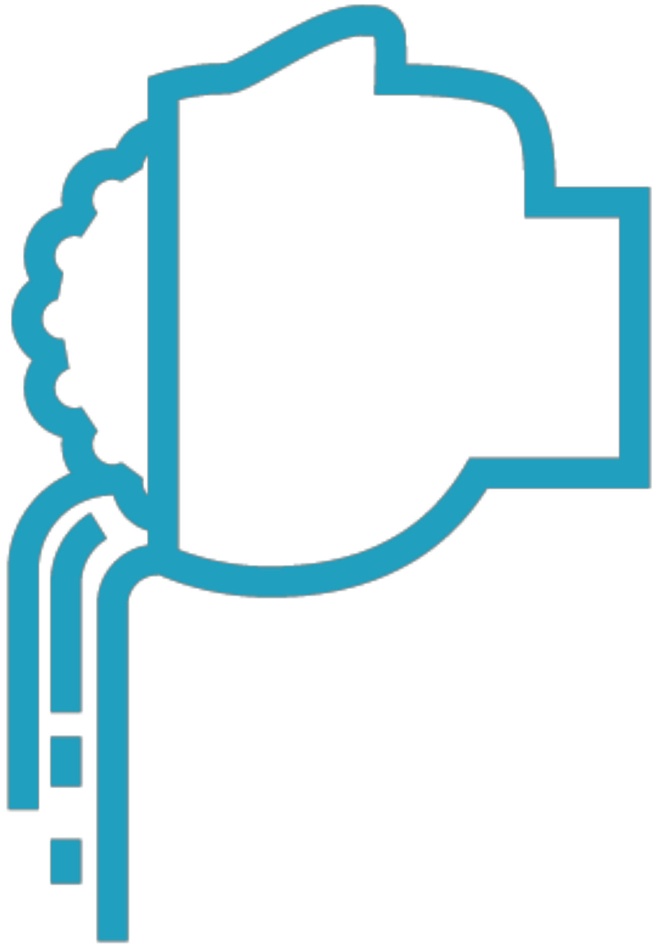
Devised CUDA, a parallel computing platform and API

A standard for general purpose (non-graphics) users of GPUs

Initially acronym for “Compute Unified Device Architecture”

Now a standalone term, not an acronym

CUDA and PyTorch

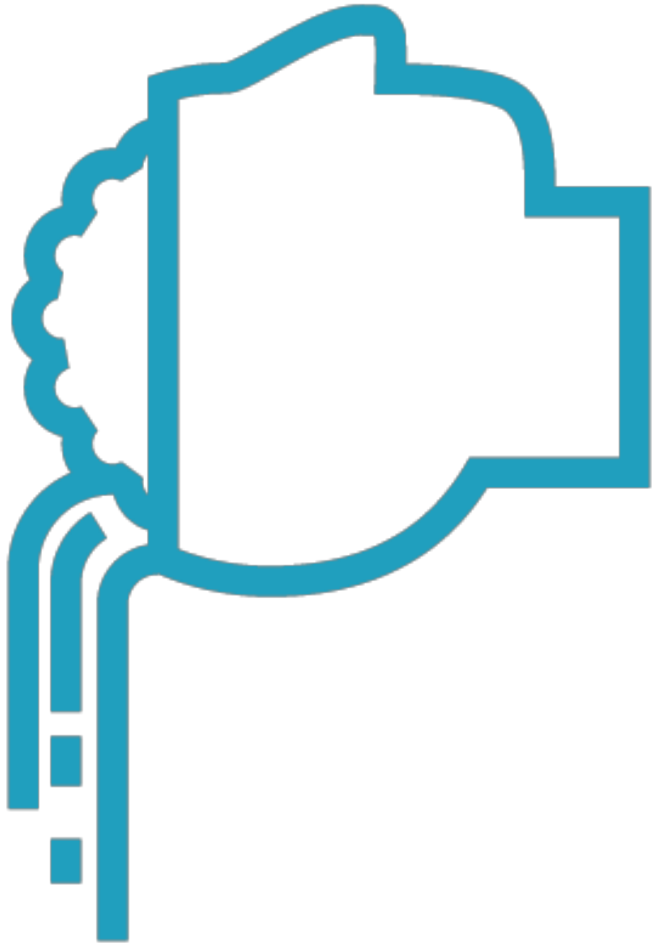


Developers can write CUDA-compliant code

Code must be understood by CUDA-aware framework (e.g. PyTorch)

If CUDA-enabled GPUs are available, speedup will automatically occur

CUDA and PyTorch



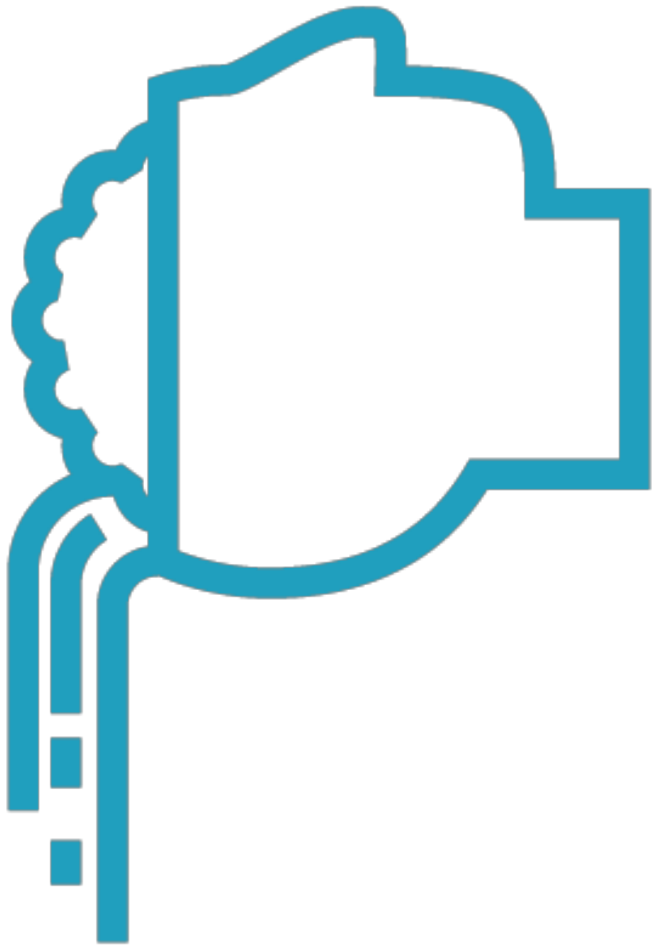
`torch.cuda` **for CUDA operations**

Special tensor types for CUDA e.g. `torch.cuda.FloatTensor`

`torch.cuda.device` **to select GPU**

Tracks currently selected GPU and creates tensors on it

CUDA and PyTorch



Cross-GPU operations not allowed by default

- Exceptions: copy_(), to(), cuda(), and other methods with copy semantics

Ops on tensors across devices will cause errors

- Can mitigate this by enabling peer-to-peer memory access

Asynchronous Execution

GPU operations are asynchronous by default

Enqueued to particular device, executed later

Allows execution of many more computations in parallel

Asynchronous Execution

Asynchronous execution typically invisible to user

- FIFO order of queuing
- Automatic synchronization by PyTorch between devices

Asynchronous Execution

Can force synchronous execution using environment variable

CUDA_LAUNCH_BLOCKING = 1

Useful for error handling and examining stack traces

Functions such as to(), copy() allow non_blocking argument

CUDA Streams

Linear sequence of operations for execution on a single device

By default, each device has a default stream

Operations within a stream are serialized by PyTorch (order is deterministic)

Order of execution across streams is not deterministic

Device-agnostic Code

Device-agnostic code explicitly handles GPU and CPU cases

Common pattern is to use argparse to read user arguments

Code can then be invoked with runtime flags to enable or disable CUDA

Demo

PyTorch and CUDA semantics

Summary

Tensor operations using PyTorch tensors

Interoperability with NumPy

Understanding PyTorch support for GPUs and CUDA

Working with tensors on GPU-enabled devices