ODSC WEST

Warmup Guide to PyTorch

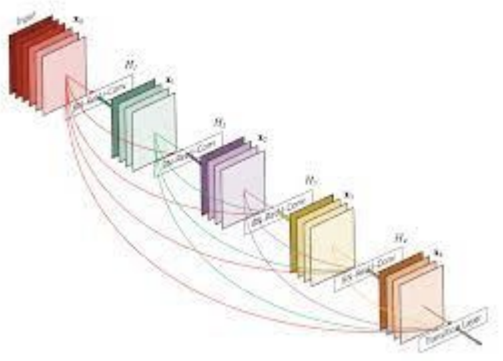
A comprehensive guide to PyTorch, a popular framework for machine learning and deep learning that you can learn more about at ODSC West 2021

Table of Contents

Table of Contents

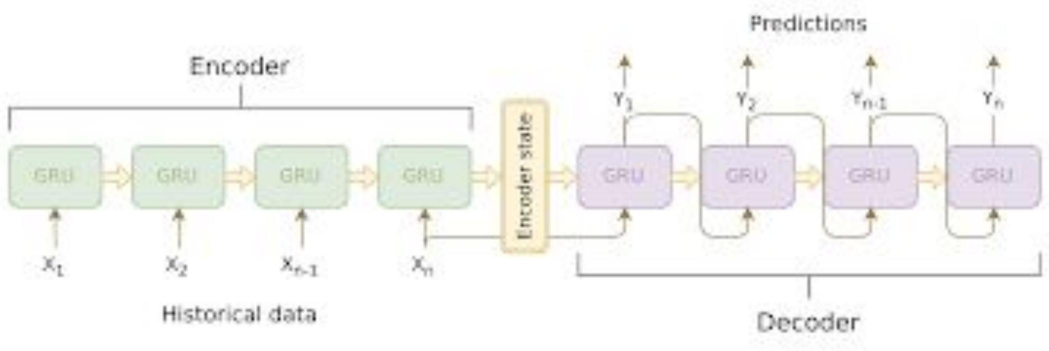
Overview

PyTorch is an open-source framework built for developing machine learning and deep learning models. In particular, this framework provides the stability and support required for building computational models in the development phase and deploying them in the production phase.



PyTorch functionalities are extensible with other Python libraries like **NumPy** and **SciPy**. Additionally, the strong GPU acceleration of PyTorch enables it to perform high-level tensor computations with ease.

It's also used with **TorchScript,** which is a built-in tool that makes PyTorch flexible while seamlessly transitioning between eager execution mode and graph mode to achieve higher speed and optimization. The latest PyTorch versions also support graph-based execution, distributed training, mobile deployment, and quantization.



Seq2seq with TorchScript. Source python.org

What Is PyTorch

Developed by **Facebook AI** in 2016, PyTorch, a Python library, enables engineers and developers to perform fast computation with a user-friendly front-end.

Currently, PyTorch is highly preferred by data scientists and artificial intelligence engineers. It is predominantly used for neural network architectures, such as Recurrent Neural Network (RNN), Convolutional Neural Network (CNN), and similar neural networks.

PyTorch also offers a wide range of features, including Object-Oriented Programming (OOP) support and dynamic computation graphs. With PyTorch's reverse-mode auto-differentiation technique, developers can modify network behavior with no lag or overhead. PyTorch is not limited only to deep learning implementations, however.

While TorchScript provides flexibility and functionalities in C++ runtime environments, there is also a provision for collective operation and support for an end-to-end workflow for machine learning integration in mobile applications. Additionally, PyTorch is well-suited for any complex mathematical computations due to its faster GPU acceleration.

Job Roles That Use PyTorch

- **Deep Learning Engineer**
- **Machine Learning Engineer**
- **AI Engineer**
- **Data Scientist**
- **Data Engineer**
- **Software Engineer**
- **Computer Vision Engineer**
- **Cloud Engineer**
- **MLOps Specialist**
- **NLP Engineer**

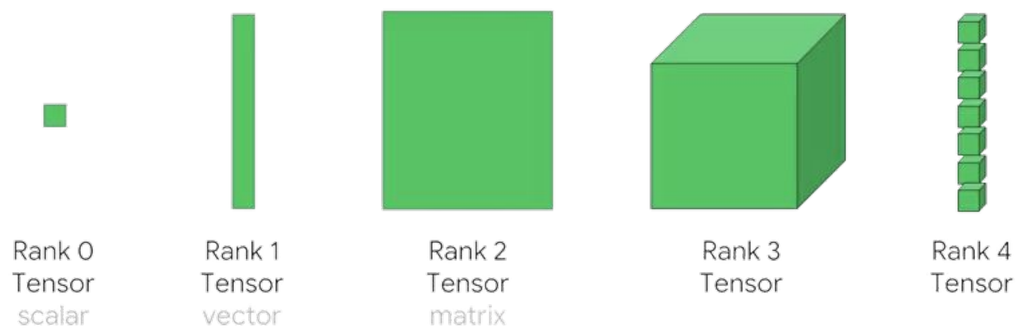
Key Terminology

To gain mastery in PyTorch, it is essential to understand some of its key terminologies.

Tensors

Tensors are the fundamental data structures of PyTorch and other deep learning frameworks like TensorFlow. They are multidimensional arrays that are similar to NumPy arrays. Tensors are one of the core components of PyTorch. The functionality of tensors is similar to that of metrics, a well-known representation of an array of numbers. A tensor is an N-dimensional array that acts as a data container. A 1D tensor is known as a vector, whereas a 2D tensor is a metric.

Similarly, a 3D tensor is a cube, and a 4D tensor is a cube vector. The PyTorch tensors utilize GPU to speed up the mathematical computations and are ideal for fitting into a neural network.

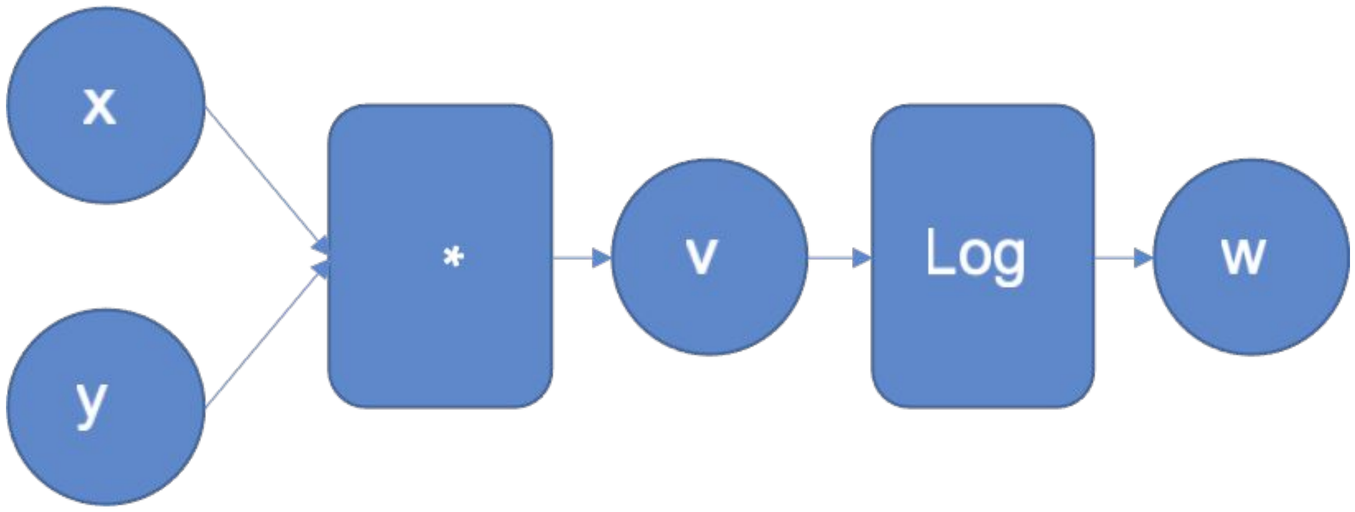


A tensor is a multidimensional (n rank) array of data

Autograd

In PyTorch, the *torch.autograd* is its automatic differentiation engine that is responsible for training the neural networks. The training of a neural network occurs mainly in two ways: forward propagation and backward propagation. This technique ensures a record is maintained on the operations performed and the records are replayed backward for gradient computation. Such a technique is essential for neural networks, as it reduces the computational time on an epoch by calculating the differentiation between the parameters during the forward pass within the network.

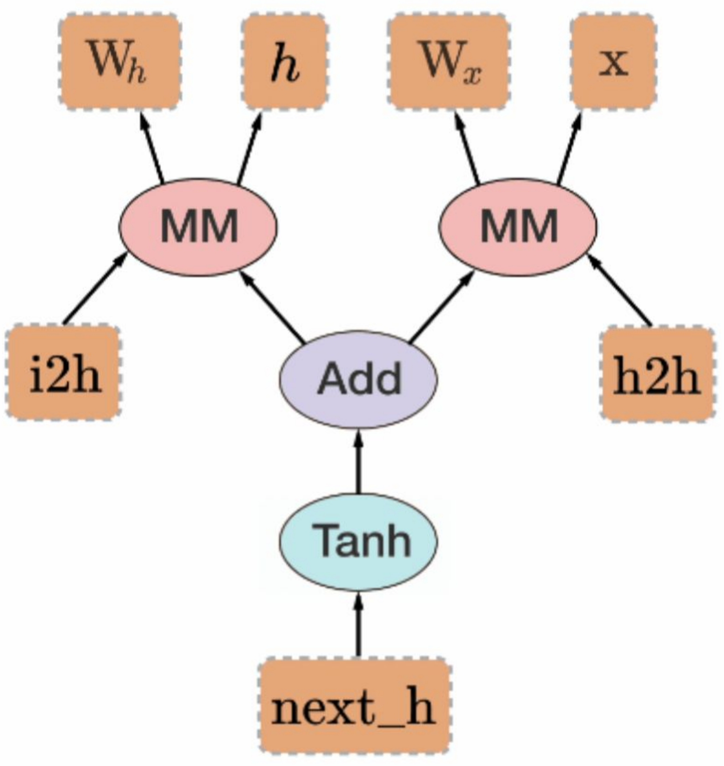
You may remember the chain rule from calculus class for finding the derivative of composite functions and allows us to calculate very complex derivatives by splitting them and recombining them later. Automatic differentiation relies on the the chain-rule to combine the derivatives of the simpler functions that compose a larger one, such as a neural network. Thus we can compute the exact value of the gradient at a given point rather than relying on approximations. To learn more, visit pytorch.org and see how the simple function below is computed



*Computational graph of f(x, y) = log(x*y)*

Dynamic Graph

Some users prefer statistical graphs because of the ability to optimize the graph. However, PyTorch provides developers with dynamic graphs that are easier to use and provide more control over changing the network behavior. Using dynamic graphs minimizes the cost incurred on optimization when the same graph needs to be reused repeatedly.



Source pytorch.org

Optim Package

The *optim* package in PyTorch is used for optimizing algorithms. The *torch.optim* module allows the implementation of a wide range of optimization algorithms for building neural networks. It can be used by importing it in the source code. Some of the supported algorithms include:

Adam Algorithm*: also known as adaptive Moment Estimation and one of the most popular*

Adamax Algorithm*: implements the Adamax algorithm which is a variant of Adam*

SGD Algorithm*: implements stochastic gradient descent*

AdamW Algorithm*: an improved version of Adam class in which weight decay is performed only after controlling the parameter-wise step size*

You can learn more about the optim package here

nn Module

PyTorch *autograd* is beneficial for computational graphs and gradient computation. However, raw *autograd* may not be suitable for complex neural networks. To address this problem, the *nn* module is useful. The *nn* package can define the set of modules capable of producing output from input with more trainable weights.

Multiprocessing

Multiprocessing is identical to the Python *multiprocessing* package, which works similarly to a process-based thread interface. It can be considered an extension of the Python distributed package, using PyTorch tensors in shared memory while providing the handle to other processes. Generally, *multiprocessing* works asynchronously, which means a specific process is enqueued once the process reaches the top of the queue. The same procedure is followed when there is a need to copy between different devices, which PyTorch performs periodically.

Getting Started with PyTorch

You can download PyTorch here.

There are several versions available for installation. Therefore, it is essential to understand how to pick the version best suited for your needs.

The PyTorch build provides a Stable and a Preview (Nightly) option. Stable represents the current versions that are supported and tested by PyTorch. On the other hand, the preview version is the latest offering, but is not thoroughly tested or supported—the nightly highlights when it is generated. Users must meet all the prerequisites for installation as any missing requirement may lead to faulty performance. As a result, Anaconda is the most preferred package manager since it installs all the required dependencies for PyTorch.

PyTorch is also available for cloud partners and mobile platforms. The available cloud platforms for PyTorch are:

- Alibaba Cloud
- Amazon Web Services
- Google Cloud Platform
- Microsoft Azure

As for the PyTorch mobile platforms, the current version is PyTorch 1.3. It supports the entire workflow from Python to the deployment on iOS and Android. Currently, there are also experimental versions available, and further upgrades are likely to be introduced. Additionally, for users working on Linux or Mac there are separate versions for installation, including older versions for all of these operating systems.

Key Competencies: Junior Level

Install and Setup PyTorch on Windows

PyTorch can be installed for different Windows distributions. However, processing time varies based on your system and computer requirements, and could take longer than expected. If you have an NVIDIA GPU, although it's not mandatory, you can harness the potential of PyTorch's CUDA support. Prerequisites include:

- Windows 7, Windows 10, or greater
- Windows Server 2008 r2 or greater is suggested
- Python 3x

To install PyTorch, you are required to install Python first and then follow the next steps. There are several ways to install Python, but most users highly prefer to download it from the Python website or use Anaconda. Check the Pytorch installation guideline here for more information.

Anaconda

To install PyTorch with Anaconda, open the Anaconda prompt from the Start Menu, then the Anaconda3 and Anaconda Prompt.

If you are installing PyTorch without CUDA, you have to select OS: Windows, Package: Conda, and set the CUDA option to *none*. Run the command that is provided after making the selection.

If you have CUDA requirements, then the CUDA option should be set to the suitable version for your system. It is important to note that the latest CUDA version is always suggested. On making the selection, you will be presented with a command that you need to run in the next step.

pip

The process of selecting the operating system remains the same. While in the package, select *pip* instead of *Conda*. The CUDA version can be set as per your requirements.

Verification

Once the installation is complete, you can verify if the installation was done correctly. To do so, you can run a sample PyTorch code.

For the command line, open it and type the command *python*, and then you can run the code:

```
import torch

x = torch.rand(5, 3)

print(x)
```

Also, to check the GPU driver and CUDA settings, run the following command:

```
import torch

torch.cuda.is_available()
```

To install from the source, follow the instructions provided here.

Working With Data

PyTorch provides two ways to work with the data, namely *torch.utils.data*. *DataLoader* and *torch.utils.data* Dataset. The primary use of the dataset is to store the sample alongside the labels. The *DataLoader*, on the other hand, mainly provides iteration capabilities to the dataset. You can also create custom datasets and implement *DataLoader* on them. A *DataLoader* ensures flexibility in programming using PyTorch. Additionally, there is a wide range of domain-specific libraries with datasets available to users when using PyTorch.

The codes required for data processing can be complex at times. Therefore, PyTorch offers data primitives for better readability. To access a pre-loaded dataset, the following parameters are crucial:

- *root* stands for the path where the training and test set are stored
- *train* implies the training or the test dataset
- *download=True* specifies the data from the internet source if it's not available at the root
- *transform* and *target_transform* implies the feature and label transformations.

Try out the sample code for loading data here.

Working With Transforms

The data in a dataset does not necessarily come in a processed format. However, when you train a machine learning algorithm, PyTorch provides a functionality for manipulating data to make it suitable for training. The *Torchvision* datasets contain two parameters, *transform* and *target_transform*, to modify the features and labels. To understand more about transforms, more information is available in this link.

Building a Neural Network

The *torch.nn* module helps create neural networks. All the modules in PyTorch are from the subclasses of the *nn* module. There is a provision for all the layers of typical neural network architecture, such as fully connected layers, convolutional layer, activation function, and the loss function. When the neural network architecture is set up, you need to feed the data into the network. To perform this operation, you need to update weights and any deviations for the network to begin the learning process. All the utilities for this technique are provided in the *torch.optim* module. Additionally, automatic differentiation is used for the backward propagation process, which can be achieved with the *torch.autograd* module.

Use of Autograd

Backpropagation is frequently used for training neural networks. Weights are adjusted with a gradient of the *loss* function. These gradients are computed with the automatic differentiation engine with a *torch.autograd* function. This enables automatic computation of the gradient for computational graphs.

Optimization of Parameters

In addition to insight into the required functions for datasets and transforms necessary for creating a neural network, it is crucial to understand how optimization of parameters works for the neural network.

Training a neural network is an iterative process that enables the model to achieve the output while calculating the *loss* (error) and the derivatives of these errors concerning the parameters. These parameters are optimized using gradient descent algorithms. With better optimization, the model is likely to perform more accurately as you train it further.

Model Compatibility

PyTorch provides *TorchScript*, which can be used for running your computational model regardless of the runtime. It acts as a virtual machine that comprises tensor-based instructions for the model. You have the option to convert the trained model using PyTorch into other formats such as *ONNX*, which allows the models to be used in other deep learning frameworks.

Benefits of PyTorch

Some of the significant benefits of PyTorch are:

Dynamic Computational Graphs

Generally, computational graphs in a deep learning framework are analyzed before runtime. In contrast, PyTorch offers the ability to build the graphs at runtime with a feature called reverse-mode-auto differentiation. This feature ensures that there is no lag or overhead when rebuilding the model. Additionally, PyTorch is considered to be among the fastest in the implementation of the reverse-mode-auto differentiation.

A dynamic computational graph helps debug and perform specific tasks like natural language processing for text and speech.

Python Based

PyTorch originated from a desire to improve the deep learning experience of Python users. Initially, it was created as a *torch* library. Over time PyTorch has evolved to become an integral part of Python today. The PyTorch functions are built as Python classes which allows seamless integration with Python functions and packages.

Faster Computation

PyTorch has individual back ends for CPU and GPU computations. A separate back end allows it to focus mainly on a specific task running on a particular processor, resulting in high memory efficiency. The deployment of PyTorch becomes easier with individual back ends, especially for heavy computational applications such as in embedded systems.

Extendable

PyTorch users can program in C/C++ with the help of an API based on Python. It can be compiled further for CPU or GPU-related operations. This provides the required flexibility and extensibility for PyTorch users.

Debugging

PyTorch is an essential part of Python, and is able to access and effectively use various Python-based debugging tools for PyTorch codes, such as Python's PyCharm. This is possible because of the use of computational graphs in PyTorch, which is defined at runtime. Similarly, Python's *pdb* and *ipdb* tools work well for debugging PyTorch codes.

PyTorch Use Cases

Because of its strong GPU acceleration, PyTorch is increasingly being used across organizations to improve productivity. For example, **Microsoft** implements PyTorch for their language modeling services and their cognitive toolkit. The company required a deep learning framework that provided consistency and a stable API that is user-friendly and flexible. Eventually, Microsoft partnered with **Facebook** to create an internal language modeling toolkit on top of PyTorch. Since PyTorch is extensible, it allowed Microsoft to build a custom task and architecture. It also allowed for the formation of a stable and intuitive API and resulted in tremendous improvements in model sizing.

Twitter is gradually shifting towards the use of PyTorch for fine-tuning their ranking algorithm on timelines.

At the same time, **Facebook** constantly improves their NLP research developments for text and speech, and PyTorch is used for most of the research for building chatbots, text-to-speech products, image, and video classification, or machine translation tasks for Facebook and other companies.

Recently, **Airbnb** has offered a dialog assistant on their website for helping users to navigate quickly, replying to user queries, and improving customer experience. The core of their machine translation model for building their dialog assistant is powered by PyTorch. Future research may include investigating how to leverage PyTorch and its machine translation library to create an agent response mechanism by building a sequence to sequence model.

Finally, **Toyota** has developed a state-of-the-art cloud platform for distributed deep learning for driver support. PyTorch enabled Toyota to scale up their deep learning system with a simple API and global Python ecosystem.

Ultimately, PyTorch has allowed us to gain faster training on large-scale data, including sensory inputs and videos, and the capability to iterate quickly with better product features.

ODSC WEST

Since 2014, ODSC has raised the bar for data science conferences, convening thousands of practicing data scientists, engineers, analysts, business managers, and academics to each event.

World-renowned leaders in AI like Kirk Borne, Cassie Kozyrkov, Mike Stonebraker, Pieter Abbeel, Regina Barzilay, and others attend each ODSC conference to draw upon their expertise and to show how you can turn this newfound knowledge into immediate action.

Backed with the trusted ODSC brand we offer our partners a wide variety of opportunities to achieve their marketing and sales goals digitally. Targeted content to qualified personas via Virtual Conference & Expo, webinars, podcasts, thought leadership, AI & Data Science trends reports and KPIs and much more.

- **KICKSTART-BOOTCAMP**
- **DEEP LEARNING & MACHINE LEARNING**
- **DATA ENGINEERING & MLOPS**
- **NLP**
- **BIG DATA ANALYTICS**
- **AI FOR GOOD & RESPONSIBLE AI**
- **RESEARCH FRONTIERS**
- **CYBERSECURITY & MACHINE LEARNING**
- **DATA VISUALIZATION**
- **R FOR MACHINE LEARNING**
- **DATA SCIENCE KICKSTART**
- **DATA SCIENCE MANAGEMENT**
- **AI BUSINESS SUMMIT**

Visit our website
odsc.com/california

Learn more about partnership opportunities
alvaro@odsc.com

Discuss your hiring goals:
hiring@odsc.com

Read more data science articles
opendatascience.com