PyTorch

PyTorch is a python package that provides two high level features:

- Tensor computation with strong GPU acceleration
- Deep Neural Networks built on a tape-based autograd system.

At a granular level, PyTorch is a library that consists of the following components:

Package	Description
torch	a Tensor library like NumOy, with strong GPU support
torch.autograd	a tape based automatic differentiation library that supports all differentiable Tensor operations in torch
torch.nn	a neural networks library deeply integrated with autograd designed for maximum flexibility
torch.optim	an optimization package to be used with torch.nn with standard optimization methods such as SGD, RMSProp, LBFGS, Adam etc.
torch.multiprocessing	python multiprocessing, but with magical memory sharing of torch Tensors across processes. Useful for data loading and hogwild training.
torch.utils	DataLoader, Trainer and other utility functions for convenience
torch.legacy(.nn/.optim)	legacy code that has been ported over from torch for backward compatibility reasons

Usually one uses PyTorch either as:

- A replacement for **numpy** to use the power of GPUs.
- A deep learning research platform that provides maximum **flexibility** and **speed**

Elaborating Further:

A GPU-ready Tensor library:

If you use numpy, then you have used Tensors (a.k.a ndarray).

PyTorch provides Tensors that can live either on the CPU or the GPU, and accelerate compute by a huge amount.

We provide a wide variety of tensor routines to accelerate and fit your scientific computation needs such as slicing, indexing, math operations, linear algebra, reductions. And they are fast!

Dynamic Neural Networks: Tape based Autograd

PyTorch has a unique way of building neural networks: using and replaying a tape recorder.

Most frameworks such as TensorFlow, Theano, Caffe and CNTK have a static view of the world. One has to build a neural network, and reuse the same structure again and again. Changing the way the network behaves means that one has to start from scratch.

With PyTorch, we use a technique called Reverse-mode auto-differentiation, which allows you to change the way your network behaves arbitrarily with zero lag or overhead. Our inspiration comes from several research papers on this topic, as well as current and past work such as autograd, autograd, Chainer, etc.

While this technique is not unique to PyTorch, it's one of the fastest implementations of it to date. You get the best of speed and flexibility for your crazy research.

Python first:

PyTorch is not a Python binding into a monolothic C++ framework. It is built to be deeply integrated into Python. You can use it naturally like you would use numpy / scipy / scikit-learn etc. You can write your new neural network layers in Python itself, using your favorite libraries and use packages such as Cython and Numba. Our goal is to not reinvent the wheel where appropriate.

Imperative experiences:

PyTorch is designed to be intuitive, linear in thought and easy to use. When you execute a line of code, it gets executed. There isn't an asynchronous view of the world. When you drop into a debugger, or receive error messages and stack traces, understanding them is straight-forward. The stack-trace points to exactly where your code was defined. We hope you never spend hours debugging your code because of bad stack traces or asynchronous and opaque execution engines.

Fast and Lean:

PyTorch has minimal framework overhead. We integrate acceleration libraries such as Intel MKL and NVIDIA (CuDNN, NCCL) to maximize speed. At the core, it's CPU and GPU Tensor and Neural Network backends (TH, THC, THNN, THCUNN) are written as independent libraries with a C99 API.

They are mature and have been tested for years.

Hence, PyTorch is quite fast – whether you run small or large neural networks.

The memory usage in PyTorch is extremely efficient compared to Torch or some of the alternatives. We've written custom memory allocators for the GPU to make sure that your deep learning models are maximally memory efficient. This enables you to train bigger deep learning models than before.

Extensions without pain:

Writing new neural network modules, or interfacing with PyTorch's Tensor API was designed to be straight-forward and with minimal abstractions.

Source: www.Pytorch.org