**THEANO**

**What is Theano?**

Theano is an open source project released under the BSD license and was developed by the LISA (now [MILA](http://mila.umontreal.ca/)) group at the University of Montreal, Quebec, Canada (home of [Yoshua Bengio](http://www.iro.umontreal.ca/~bengioy/yoshua_en/index.html)). It is named after a [Greek mathematician](https://en.wikipedia.org/wiki/Theano_(philosopher)).

At it’s heart Theano is a compiler for mathematical expressions in Python. It knows how to take your structures and turn them into very efficient code that uses NumPy, efficient native libraries like [BLAS](http://www.netlib.org/blas/) and native code (C++) to run as fast as possible on CPUs or GPUs.

It uses a host of clever code optimizations to squeeze as much performance as possible from your hardware. If you are into the nitty-gritty of mathematical optimizations in code, [check out this interesting list](http://deeplearning.net/software/theano/optimizations.html#optimizations).

The actual syntax of Theano expressions is symbolic, which can be off putting to beginners used to normal software development. Specifically, expression are defined in the abstract sense, compiled and later actually used to make calculations.

It was specifically designed to handle the types of computation required for large neural network algorithms used in Deep Learning. It was one of the first libraries of its kind (development started in 2007) and is considered an industry standard for Deep Learning research and development.

**How to Install Theano**

Theano provides extensive installation instructions for the major operating systems: Windows, OS X and Linux. Read the [Installing Theano guide for your platform](http://deeplearning.net/software/theano/install.html).

Theano assumes a working Python 2 or Python 3 environment with [SciPy](https://www.scipy.org/). There are ways to make the installation easier, such as using [Anaconda](https://www.continuum.io/downloads) to quickly set up Python and SciPy on your machine as well as using [Docker images](http://deeplearning.net/software/theano/install.html#docker-images).

With a working Python and SciPy environment, it is relatively straightforward to install Theano. from PyPI using pip, for example:

|  |  |
| --- | --- |
| 1 | pip install Theano |

At the time of writing the last official release of Theano was version 0.8 which was released 21th March 2016.

New releases may be announced and you will want to update to get any bug fixes and efficiency improvements. You can upgrade Theano using pip as follows:

|  |  |
| --- | --- |
| 1 | sudo pip install --upgrade --no-deps theano |

You may want to use the bleeding edge version of Theano checked directly out of Github.

This may be required for some wrapper libraries that make use of bleeding edge API changes. You can install Theano directly from a Github checkout as follows:

|  |  |
| --- | --- |
| 1 | pip install --upgrade --no-deps git+git://github.com/Theano/Theano.git |

You are now ready to run Theano on your CPU, which is just fine for the development of small models.

Large models may run slowly on the CPU. If you have a Nvidia GPU, you may want to look into configuring Theano to use your GPU. Read the [Using the GPU guides for Linux](http://deeplearning.net/software/theano/install.html#using-the-gpu) or [Mac OS X to set up Theano to use the GPU](http://deeplearning.net/software/theano/install.html#gpu-macos) and the [Using the GPU guide](http://deeplearning.net/software/theano/tutorial/using_gpu.html) for how to test whether it is working.

Simple Theano Example

In this section we demonstrate a simple Python script that gives you a flavor of Theano.

It is taken from the [Theano at a Glance guide](http://deeplearning.net/software/theano/introduction.html). In this example we define two symbolic floating point variables a and b.

We define an expression that uses these variables (c = a + b).

We then compile this symbolic expression into a function using Theano that we can use later.Finally, we use our complied expression by plugging in some real values and performing the calculation using efficient compiled Theano code under the covers.

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12** | **import theano**  **from theano import tensor**  **# declare two symbolic floating-point scalars**  **a = tensor.dscalar()**  **b = tensor.dscalar()**  **# create a simple expression**  **c = a + b**  **# convert the expression into a callable object that takes (a,b)**  **# values as input and computes a value for c**  **f = theano.function([a,b], c)**  **# bind 1.5 to 'a', 2.5 to 'b', and evaluate 'c'**  **assert 4.0 == f(1.5, 2.5)** |

Running the example does not provide any output. The assertion that 1.5 + 2.5 = 4.0 is true.

This is a useful example as it gives you a flavor for how a symbolic expression can be defined, compiled and used. You can see how this may be scaled up to large vector and matrix operations required for deep learning.

## Extensions and Wrappers for Theano

If you are new to deep learning you do not have to use Theano directly.

In fact, you are highly encouraged to use one of many popular Python projects that make Theano a lot easier to use for deep learning.

These projects provide data structures and behaviors in Python, specifically designed to quickly and reliably create deep learning models whilst ensuring that fast and efficient models are created and executed by Theano under the covers.

The amount of Theano syntax exposed by the libraries varies.

* For example the [Lasagne library](https://lasagne.readthedocs.org/en/latest/) provides convenience classes for creating deep learning model but still expects you to know and make use of Theano syntax. This is good for beginners that know or are willing to learn a little Theano as well.
* Another example is [Keras](http://keras.io/) that hides Theano completely and provides a very simple API to work with to create Deep Learning models. It hides Theano so well, that it can in fact run as a wrapper for another popular foundation framework called [TensorFlow](https://www.tensorflow.org/).

**PYTORCH**

## PyTorch for deep learning

PyTorch is a library for Python programs that facilitates building deep learning projects. It emphasizes flexibility and allows deep learning models to be expressed in idiomatic Python. This approachability and ease of use found early adopters in the research community, and in the years since its first release, it has grown into one of the most prominent deep learning tools across a broad range of applications.

As Python does for programming, PyTorch provides an excellent introduction to deep learning. At the same time, PyTorch has been proven to be fully qualified for use in professional contexts for real-world, high-profile work. We believe that PyTorch’s clear syntax, streamlined API, and easy debugging make it an excellent choice for introducing deep learning. We highly recommend studying PyTorch for your first deep learning library. Whether it ought to be the last deep learning library you learn is a decision we leave up to you.

At its core, the deep learning machine in figure 1.1 is a rather complex mathematical function mapping inputs to an output. To facilitate expressing this function, PyTorch provides a core data structure, the tensor, which is a multidimensional array that shares many similarities with NumPy arrays. Around that foundation, PyTorch comes with features to perform accelerated mathematical operations on dedicated hardware, which makes it convenient to design neural network architectures and train them on individual machines or parallel computing resources.

This book is intended as a starting point for software engineers, data scientists, and motivated students fluent in Python to become comfortable using PyTorch to build deep learning projects. We want this book to be as accessible and useful as possible, and we expect that you will be able to take the concepts in this book and apply them to other domains. To that end, we use a hands-on approach and encourage you to keep your computer at the ready, so you can play with the examples and take them a step further. By the time we are through with the book, we expect you to be able to take a data source and build out a deep learning project with it, supported by the excellent official documentation.

Although we stress the practical aspects of building deep learning systems with PyTorch, we believe that providing an accessible introduction to a foundational deep learning tool is more than just a way to facilitate the acquisition of new technical skills. It is a step toward equipping a new generation of scientists, engineers, and practitioners from a wide range of disciplines with working knowledge that will be the backbone of many software projects during the decades to come.

In order to get the most out of this book, you will need two things:

Some experience programming in Python. We’re not going to pull any punches on that one; you’ll need to be up on Python data types, classes, floating-point numbers, and the like.

A willingness to dive in and get your hands dirty. We’ll be starting from the basics and building up our working knowledge, and it will be much easier for you to learn if you follow along with us.

Deep Learning with PyTorch is organized in three distinct parts. Part 1 covers the foundations, examining in detail the facilities PyTorch offers to put the sketch of deep learning in figure 1.1 into action with code. Part 2 walks you through an end-to-end project involving medical imaging: finding and classifying tumors in CT scans, building on the basic concepts introduced in part 1, and adding more advanced topics. The short part 3 rounds off the book with a tour of what PyTorch offers for deploying deep learning models to production.

## Why PyTorch?

As we’ve said, deep learning allows us to carry out a very wide range of complicated tasks, like machine translation, playing strategy games, or identifying objects in cluttered scenes, by exposing our model to illustrative examples. In order to do so in practice, we need tools that are flexible, so they can be adapted to such a wide range of problems, and efficient, to allow training to occur over large amounts of data in reasonable times; and we need the trained model to perform correctly in the presence of variability in the inputs. Let’s take a look at some of the reasons we decided to use PyTorch.

PyTorch is easy to recommend because of its simplicity. Many researchers and practitioners find it easy to learn, use, extend, and debug. It’s Pythonic, and while like any complicated domain it has caveats and best practices, using the library generally feels familiar to developers who have used Python previously.

More concretely, programming the deep learning machine is very natural in PyTorch. PyTorch gives us a data type, the Tensor, to hold numbers, vectors, matrices, or arrays in general. In addition, it provides functions for operating on them. We can program with them incrementally and, if we want, interactively, just like we are used to from Python. If you know NumPy, this will be very familiar.

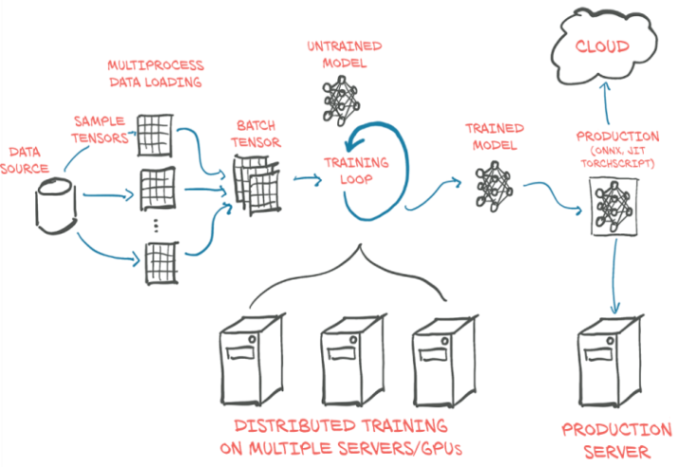
But PyTorch offers two things that make it particularly relevant for deep learning: first, it provides accelerated computation using graphical processing units (GPUs), often yielding speedups in the range of 50x over doing the same calculation on a CPU. Second, PyTorch provides facilities that support numerical optimization on generic mathematical expressions, which deep learning uses for training. Note that both features are useful for scientific computing in general, not exclusively for deep learning. In fact, we can safely characterize PyTorch as a high-performance library with optimization support for scientific computing in Python.

A design driver for PyTorch is expressivity, allowing a developer to implement complicated models without undue complexity being imposed by the library (it’s not a framework!). PyTorch arguably offers one of the most seamless translations of ideas into Python code in the deep learning landscape. For this reason, PyTorch has seen widespread adoption in research, as witnessed by the high citation counts at international conferences.[3](https://livebook.manning.com/book/deep-learning-with-pytorch/chapter-1/pgfId-1012208)

3.At the International Conference on Learning Representations (ICLR) 2019, PyTorch appeared as a citation in 252 papers, up from 87 the previous year and at the same level as TensorFlow, which appeared in 266 papers.

PyTorch also has a compelling story for the transition from research and development into production. While it was initially focused on research workflows, PyTorch has been equipped with a high-performance C++ runtime that can be used to deploy models for inference without relying on Python, and can be used for designing and training models in C++. It has also grown bindings to other languages and an interface for deploying to mobile devices. These features allow us to take advantage of PyTorch’s flexibility and at the same time take our applications where a full Python runtime would be hard to get or would impose expensive overhead.

Of course, claims of ease of use and high performance are trivial to make. We hope that by the time you are in the thick of this book, you’ll agree with us that our claims here are well founded



##### **Figure : Basic, high-level structure of a PyTorch project, with data loading, training, and deployment to production**

 we see that quite a bit of data processing is needed before the training data even reaches our model.[4](https://livebook.manning.com/book/deep-learning-with-pytorch/chapter-1/pgfId-1012633) First we need to physically get the data, most often from some sort of storage as the data source. Then we need to convert each sample from our data into a something PyTorch can actually handle: tensors. This bridge between our custom data (in whatever format it might be) and a standardized PyTorch tensor is the Dataset class PyTorch provides in torch.utils.data. As this process is wildly different from one problem to the next, we will have to implement this data sourcing ourselves. We will look in detail at how to represent various type of data we might want to work with as tensors in chapter.