

Understanding the Object Oriented design - PyTorch



Sravanth Yajamanam

Divya James Athoopallil

What is PyTorch?

- **PyTorch** is an open source Deep Learning Framework and a scientific computing package.
- PyTorch is mainly used for building neural networks and accelerated computations using GPUs(Graphics Processing Unit).

What is a Tensor?

- The fundamental unit of PyTorch is the Tensor. Tensors can be considered as multi-dimensional arrays.
- Tensors in PyTorch are similar to NumPy arrays.
- Tensors can be used on a GPU that supports **CUDA** (Compute Unified Device Architecture).

Brief History of PyTorch:

- **Soumith Chintala** is the creator of PyTorch framework. It was created at FAIR (Facebook AI Research).
- Most of the internals that are performance bottlenecks are written in C++
- Source code of PyTorch is written mostly in Python.

PyTorch: Tensor

- ▶ Tensor is the fundamental data structure used in PyTorch
- ▶ We transform the data into tensors in order to pass it through the neural network.
- ▶ Instances of the **torch.Tensor/ torch.tensor** class
 - ▶ Tensor attributes
 - ▶ `torch.dtype`: Tensors contain uniform (of the same type) numerical data eg. `torch.float32`
 - ▶ `torch.device`: specifies the device (CPU or GPU) where the tensor's data is allocated. This determines where tensor computations for the given tensor will be performed
 - ▶ `torch.layout`: specifies how the tensor is stored in memory
- ▶ **torch.Tensor** is the constructor of the `torch.Tensor` class
 - ▶ `torch.Tensor()` constructor lacks configuration options, hence we are unable to pass dtypes
- ▶ **torch.tensor** a factory function gets called constructs `torch.Tensor` objects
 - ▶ `dtype` is selected based on the incoming data i.e. the `dtype` is inferred based on the incoming data

PyTorch Internals:

- How do we implement Tensors in PyTorch, so that user can interact with Tensor from the Python shell?

For that we need to answer the following questions:

- 1) How to define a Tensor type and how PyTorch extends the interpreter to allow the new Tensor type?

It makes use of CPython's framework to extend the Python Interpreter and defines new types by generating the required code for all of them.

- 2) How does PyTorch wrap the C libraries that actually define the Tensor's properties and methods?

It is done by defining a new type THPTensor that is supported by a TH Tensor.



- 3) How does PyTorch cwrap work to generate code for Tensor methods?

The custom Input YAML-formatted code is taken and source code is generated for each method through a series of steps employing number of plugins.

- 4) How does PyTorch's build system take all of these components to compile and generate a workable application?


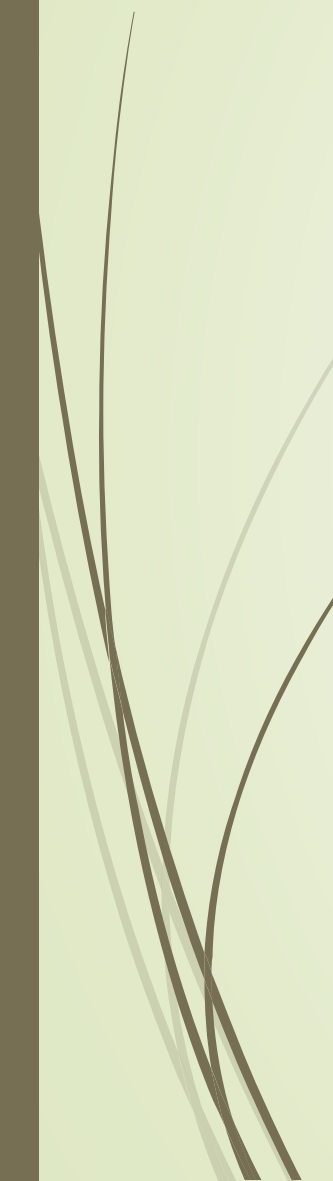
Components such as the source/header files, libraries, and compilation directives are taken in order to build an extension by using Setuptools

Components of PyTorch:



Package	Description
torch	Top-level PyTorch package & tensor library with GPU support
torch.nn	A sub package that contains modules & extensible classes for building neural networks.
torch.autograd	A sub package that supports all the differentiable Tensor operations in PyTorch
torch.nn.functional	A functional interface that contains typical operations used for building neural networks like loss functions, activation functions, and convolution operations.
torch.optim	A sub package that contains standard optimization operations like SGD and Adam.
torch.utils	A sub package that contains utility classes like data sets and data loaders that make data pre-processing easier.
torchvision	A package that provides access to popular datasets, model architectures, and image transformations for computer vision.

Neural Networks & OOPs:

- 
- By using an object-oriented approach to the design of software systems, we can achieve advantages in terms of flexibility, extensibility, and portability.
 - We can design object-oriented neural network simulation systems to achieve advantages mentioned above.
 - **Why Object oriented Programming?**
 - For example, consider a backpropagation network which is a specialized learning algorithm for specific application domains. A consequence of this is lack of flexibility.
 - If a recurrent network architecture is required for an application domain, then the system has to be redesigned to accommodate the changes.
 - Using object-oriented methodology in the development of the Neural Network system, a unified view of the problem domain can be achieved in all development phases from analysis to maintenance. The requirements for the system are :
 - a) Abstraction
 - b) Flexibility
 - c) Extensibility
- 

Building the model:

- ▶ PyTorch's neural network library contains all of the typical components needed to build neural networks. The primary component to build a neural network is a layer.
- ▶ Each layer in a neural network has two primary components:
 - ▶ A transformation (code)
 - ▶ A collection of weights (data)
- ▶ A class `Module` within the neural network package is the base class for all neural network modules with layers.
 - ▶ This is an instance of inheritance in action.
- ▶ Neural networks and layers in PyTorch extend the `nn.Module` class. This means that we must extend the `nn.Module` class when building a new layer or neural network in PyTorch.

How to build a Neural Network in PyTorch?

- In order to build the network:
 - We first design a neural network class that extends the `nn.Module` base class.
 - As we instantiate a neural network we pass the network's layers as class attributes using pre-built layers from `torch.nn` through the class constructor
- We use operations from the `nn.functional` API the network's layer attributes as well as to define the network's forward pass.

Code Sample:

```
class Network:
    def __init__(self):
        self.layer = None

    def forward(self, inp):
        inp = self.layer(inp)
        return inp
```


Define network's layers as class attributes

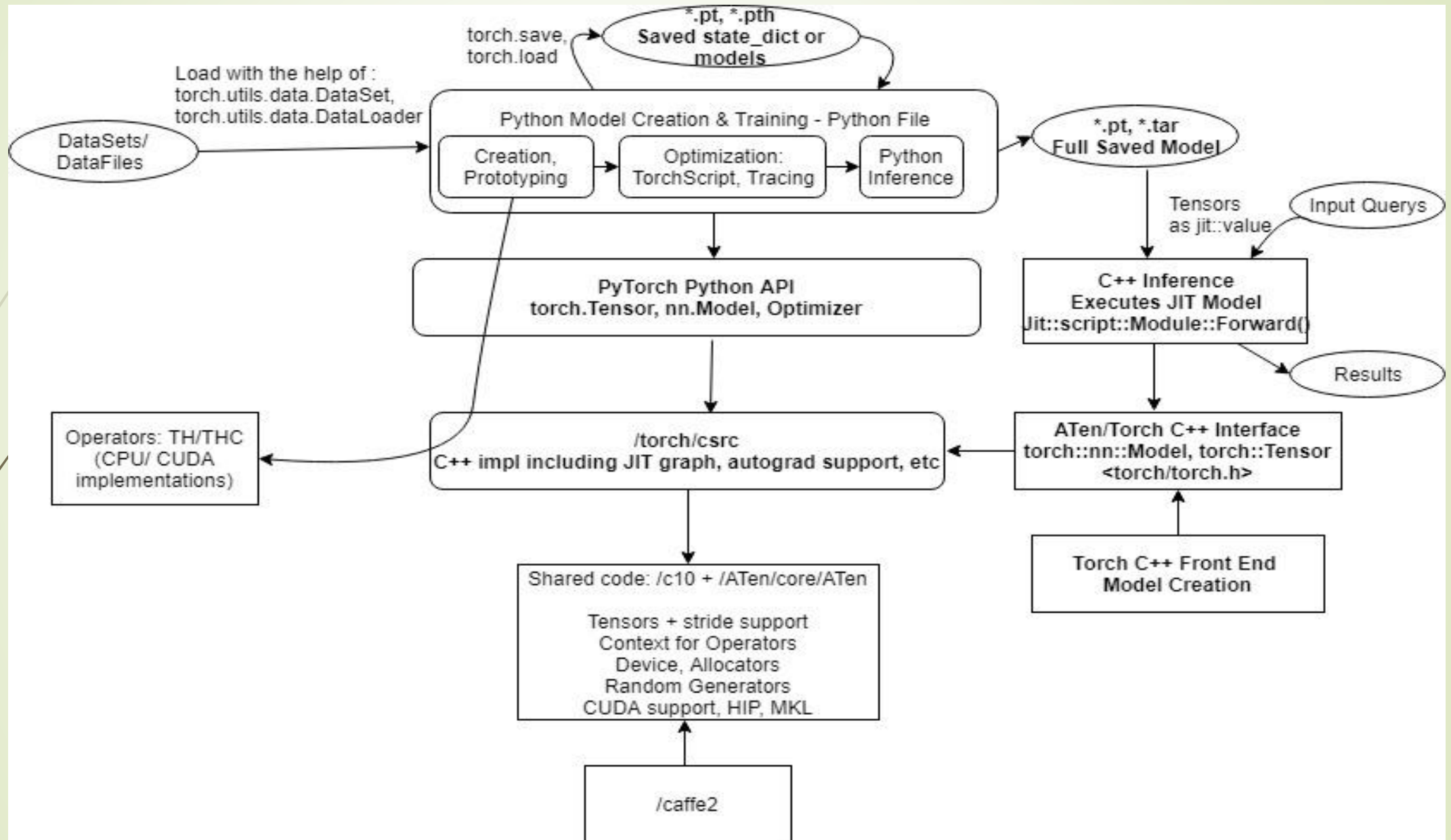
- To make our Network class extend nn.Module, we must do two additional things:
 - Specify the nn.Module class in parentheses.
 - Insert a call to the super class constructor inside the Network constructor.

Code Sample:

```
class Network(nn.Module):
    def __init__(self):
        super(Network, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5)
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=12, kernel_size=5)
        self.fc1 = nn.Linear(in_features=12 * 4 * 4, out_features=120)
        self.fc2 = nn.Linear(in_features=120, out_features=60)
        self.out = nn.Linear(in_features=60, out_features=10)

    def forward(self, t):
        # implement the forward pass
        return t
```

Data Flow & Interface Diagram of PyTorch:



Data Flow & Interface Diagram of PyTorch:

DataSet Class:

- **torch.utils.data.Dataset** is an **abstract class** representing a dataset. Our custom dataset can inherit the main Dataset class and override the below methods :
 - `__len__` so that `len(dataset)` returns the size of the dataset.
 - `__getitem__` to do the indexing such that `dataset[i]` is used to retrieve a sample `i`
- Data is loaded directly into Python and accessed with the help of DataSet abstraction. **torch.utils.data.DataLoader** can be used to support batching, shuffling and loading the data in parallel using multiprocessors.
- **PyTorch Model Creation & Training File** - Main Python file can be developed by user with help of PyTorch APIs. This file defines the model, loads data and runs the training.
- **Model Creation / Prototyping** - Network is created through **torch.nn.Model** a derived class, defining layers and a `forward()` function that connects them in the class. The model can then be trained or exported.
 - **Loading / Saving Model State** : Internal state of PyTorch model is represented by `state_dict` and it can be saved to a file with help of **torch.save** and loaded with **model.load_state_dict**.
- **Inference**: Inference refers to using trained model to make predictions on the dataset. This can be done directly in PyTorch, or in a C++ file in further stages.

Citations

- Ellinger, Barry Kristian; An Object-Oriented Approach to Neural Network;
<https://pdfs.semanticscholar.org/e582/181b2b722e00f6db92f2f64a1f5b7713dc37.pdf>
- Vasilev Ivan et.al. ; Python Deep Learning
- Tianqi Chen et.al.; MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems; <https://www.cs.cmu.edu/~muli/file/mxnet-learning-sys.pdf>
- Abadi, Martín,; TensorFlow: A System for Large-Scale Machine Learning;
<https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf>
- Lerer Adam et.al., Pytorch-Biggraph: A large scale graph embedding system
<https://www.sysml.cc/doc/2019/71.pdf>
- Liao, Qianli ; Object Oriented Deep Learning;
<https://dspace.mit.edu/bitstream/handle/1721.1/112103/CBMM-Memo-070.pdf?sequence=1>
- Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch

Citations

- Bradbury, James et.al. ;Automatic Batching as a Compiler Pass in PyTorch;
[http://learningsys.org/nips18/assets/papers/107CameraReadySubmissionMatchbox_LearningSys_Abstract_%20\(2\).pdf](http://learningsys.org/nips18/assets/papers/107CameraReadySubmissionMatchbox_LearningSys_Abstract_%20(2).pdf)
- <https://pytorch.org/features>
- <https://pytorch.org/docs/stable/index.html>
- <https://pytorch.org/blog/a-tour-of-pytorch-internals-1/>
- <https://pytorch.org/blog/a-tour-of-pytorch-internals-2/>
- <https://github.com/PyTorch/PyTorch/wiki>
- <https://github.com/PyTorch/PyTorch/wiki/PyTorch-Data-Flow-and-Interface-Diagram>
- <https://github.com/PyTorch/PyTorch/wiki/Code-review-values>
- <https://docs.python.org/3.7/extending/index.html>



THANK YOU