# PyTorch, MxNet & Theano

Hy Truong Son
STATT 37790 - Topics in Statistical Machine Learning:
High-Performance Machine Learning System Design

The University of Chicago

May 2019

THE UNIVERSITY OF CHICAGO

# Reference

Reference:

1. Automatic differentiation in PyTorch, Paszke et. al (NIPS 2017)
2. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems, Chen et. al (NIPS 2016)
3. Theano: A Python framework for fast computation of mathematical expressions, Al-Rfou et. al, 2016

To get this presentation slides:

`http://people.cs.uchicago.edu/~hytruongson/MXNet-PyTorch.pdf`

# Overview

# What is PyTorch?

**Definition [Paszke et. al, 2017]**

PyTorch is a library designed to enable rapid research on machine learning models and provides a high performance environment with easy access to automatic differentiation of models executed on different devices (CPU and GPU). PyTorch is built upon **Lua Torch**, **Chainer** and **HIPS Autograd**.

# What is PyTorch?

## Definition [Paszke et. al, 2017]

PyTorch is a library designed to enable rapid research on machine learning models and provides a high performance environment with easy access to automatic differentiation of models executed on different devices (CPU and GPU). PyTorch is built upon **Lua Torch**, **Chainer** and **HIPS Autograd**.

## Question 1

Why is PyTorch called a **library**?

THE UNIVERSITY OF CHICAGO

# What is PyTorch?

**Definition [Paszke et. al, 2017]**

PyTorch is a <u>library</u> designed to enable <u>rapid</u> research on machine learning models and provides a high performance environment with easy access to automatic differentiation of models executed on <u>different devices</u> (CPU and GPU). PyTorch is built upon **Lua Torch**, **Chainer** and **HIPS Autograd**.

**Question 1**

Why is PyTorch called a **library**?

**Question 2**

Why did the authors say **rapid research**? What was TensorFlow designed for (discussed last time)?

# What is PyTorch?

**Definition [Paszke et. al, 2017]**

PyTorch is a library designed to enable rapid research on machine learning models and provides a high performance environment with easy access to automatic differentiation of models executed on different devices (CPU and GPU). PyTorch is built upon **Lua Torch**, **Chainer** and **HIPS Autograd**.

**Question 1**

Why is PyTorch called a **library**?

**Question 2**

Why did the authors say **rapid research**? What was TensorFlow designed for (discussed last time)?

**Question 3**

What is the scale of PyTorch (e.g. different devices) vs. TensorFlow?

# How is PyTorch similar and different from others?

1. **Similarity:** Like most other deep learning libraries, PyTorch supports reverse-mode automatic differentiation of scalar functions (or vector - Jacobian products of functions with multiple outputs). The most important form of automatic differentiation for deep learning applications is usually differentiating a single scalar loss.

# How is PyTorch similar and different from others?

1. **Similarity:** Like most other deep learning libraries, PyTorch supports reverse-mode automatic differentiation of scalar functions (or vector - Jacobian products of functions with multiple outputs). The most important form of automatic differentiation for deep learning applications is usually differentiating a single scalar loss.
2. **Difference:** To make prototyping easier, PyTorch does not follow the symbolic approach used in many other deep learning frameworks, but focuses on differentiation of purely imperative programs, with a focus on extensibility and low overhead.

THE UNIVERSITY OF CHICAGO

# Features of autograd

1. **Dynamic, define-by-run execution:** A dynamic framework defines the function to be differentiated simply by running the desired computation. In contrast, a static graph structure is differentiated symbolically ahead of time and then run many times.

# Features of autograd

1. **Dynamic, define-by-run execution:** A dynamic framework defines the function to be differentiated simply by running the desired computation. In contrast, a static graph structure is differentiated symbolically ahead of time and then run many times.
2. **Immediate, eager execution:** An eager framework runs tensor computations as it encounters them, avoids materializing a *forward graph*, and records only what is necessary to differentiate the computation.

THE UNIVERSITY OF CHICAGO

# Dynamic eager execution (1)

Traditional reverse - mode differentiation records a tape (also known as a **Wengert list**) describing the order in which operations were originally executed; this optimization allows implementations to **avoid a topological sort**.

Every intermediate result **records only the subset** of the computation graph that was relevant to their computation. PyTorch users can **mix and match independent graphs** (without explicit synchronization). When a portion of the graph becomes dead, it is automatically freed (to free large memory chunks).

# Dynamic eager execution (1)

Traditional reverse - mode differentiation records a tape (also known as a **Wengert list**) describing the order in which operations were originally executed; this optimization allows implementations to **avoid a topological sort**.

Every intermediate result **records only the subset** of the computation graph that was relevant to their computation. PyTorch users can **mix and match independent graphs** (without explicit synchronization). When a portion of the graph becomes dead, it is automatically freed (to free large memory chunks).

**Question**

Can this design work for multiple workers (each worker has multiple devices) connecting by RPC (or gRPC) as in TensorFlow?

# Dynamic eager execution (2)

## History

PyTorch started its life as a Python library.

# Dynamic eager execution (2)

## History

PyTorch started its life as a Python library.

## Question

Python is an interpreter language. Is that a problem for automatic differentiation (AD)?

# Dynamic eager execution (2)

**History**

PyTorch started its life as a Python library.

**Question**

Python is an interpreter language. Is that a problem for automatic differentiation (AD)?

**Core logic in C++**

Interpreter overhead is too high for core AD logic. PyTorch is in the process of moving core operator definitions to C++.

The authors claimed that PyTorch can achieve much lower overhead compared to other frameworks(?).

# Interface - Example

Consider the following example:

```
from torch.autograd import Variable
x, prev_h =
      Variable(torch.randn(1, 10)),
      Variable(torch.randn(1, 20))
W_h, W_x =
      Variable(torch.randn(20, 20)),
      Variable(torch.randn(20, 10))
i2h = torch.matmul(W_x, x.t())
h2h = torch.matmul(W_h, prev_h.t())
(i2h + h2h).tanh().sum().backward()
```

# Implementation - Metadata

**Observation:** You write code as if you were executing tensor operations directly; however, instead of operating on Tensors (PyTorch's equivalent of Numpy multi-dimensional array), the user manipulates `Variable`, which store **extra metadata** necessary for automatic differentiation.

# Implementation - Metadata

**Observation:** You write code as if you were executing tensor operations directly; however, instead of operating on Tensors (PyTorch's equivalent of Numpy multi-dimensional array), the user manipulates `Variable`, which store **extra metadata** necessary for automatic differentiation.

**Question**

What is the **extra metadata**?

THE UNIVERSITY OF CHICAGO

# Implementation - Metadata

**Observation:** You write code as if you were executing tensor operations directly; however, instead of operating on Tensors (PyTorch's equivalent of Numpy multi-dimensional array), the user manipulates `Variable`, which store **extra metadata** necessary for automatic differentiation.

**Question**

What is the **extra metadata**?

**Solution**

1. Variables support a `backward()` method, which computes the gradient of all input Variables involved in computation.
2. Gradients are accumulated in the `grad` field of input variables, a design inherited from Chainer.

# Interface - Functional Programming

## Autograd style

PyTorch provides a HIPS autograd-style functional interface for computing gradients.

# Interface - Functional Programming

## Autograd style

PyTorch provides a HIPS autograd-style functional interface for computing gradients.

## Example

The function `torch.autograd.grad(f(x, y, z), (x, y))` computes the derivative of `f` w.r.t. `x` and `y` only (no gradient is computed for `z`).

# Interface - Functional Programming

## Autograd style

PyTorch provides a HIPS autograd-style functional interface for computing gradients.

## Example

The function `torch.autograd.grad(f(x, y, z), (x, y))` computes the derivative of `f` w.r.t. `x` and `y` only (no gradient is computed for `z`).

## No mutation for `.grad` attributes

Unlike the Chainer-style API, the call does not mutate `.grad` attributes; instead, it returns a tuple containing gradient w.r.t. each of the inputs requested in the call.

# Interface - Other properties

1. Excluding subgraphs from derivative computation when they are not needed for computational saving.
2. PyTorch users can create custom differentiable operations by specifying a pair of `forward()` and `backward()` functions in Python. The `forward()` computes the operation, while the `backward()` extends the vector-Jacobian product.

# Implementation - Memory management (1)

**Problem 1**

One of the biggest limitations of GPUs is low memory capacity.

# Implementation - Memory management (1)

**Problem 1**

One of the biggest limitations of GPUs is low memory capacity.

**Solution**

1. PyTorch frees all intermediate values as soon as they become unneeded.
2. Python is well-suited for this purpose, because it is reference counted by default.

THE UNIVERSITY OF CHICAGO

**Problem 2**

A naive implementation of automatic differentiation can easily introduce **reference cycles**. For example, when a differentiable function wants to save a reference to its output. Another challenge is avoiding reference cycles.

# Implementation - Memory management (2)

**Problem 2**

A naive implementation of automatic differentiation can easily introduce **reference cycles**. For example, when a differentiable function wants to save a reference to its output. Another challenge is avoiding reference cycles.

**Solution**

PyTorch does **not** record a full-fledged variable, but instead a **saved variable**, which omits a pointer to the Function in such cases.

# What is MxNet?

## Definition [Chen et. al, 2016]

MxNet is a **multi-language** machine learning (ML) library to ease the development of ML algorithms, especially for deep neural networks. Embedded in the host language, it blends **declrative symbolic expression** with **imperative tensor computation** in a **unified fashion**. It offers auto differentation to derive gradients. MxNet is computation and memory efficient and runs **on various heterogeneous systems**, ranging from mobile devices to distributed GPU clusters.

# Programming paradigms

Possible programming paradigms are:

1. **Imperative:** The user specifies exactly **how** computation needs to be performed.
2. **Declarative:** The user focuses on **what** to be done.

# Programming paradigms

Possible programming paradigms are:

1. **Imperative:** The user specifies exactly **how** computation needs to be performed.
2. **Declarative:** The user focuses on **what** to be done.

## Mixture of programming paradigms

Frameworks such as Theano and TensorFlow can also be viewed as a mixture of both **imperaive** and **declarative** programming paradigms. They declare a computational graph, yet the computation within the graph is imperatively specified.

# Executions

Execution (how the computation is carried out) can be:

1. **Concrete:** The result is returend right away on the same thread.
2. **Asynchronize (delayed):**

   The statements are gathered and transformed into a dataflow graph as an intermediate representation first, before released to available devices.

# Executions

Execution (how the computation is carried out) can be:

1. **Concrete:** The result is returend right away on the same thread.
2. **Asynchronize (delayed):**

   The statements are gathered and transformed into a dataflow graph as an intermediate representation first, before released to available devices.

## Mixture of executions

Concrete execution is restrictive (e.g. parallelized matrix multiplication), whereas asynchronized/delayed execution additionally identified all parallelism within the scope of an instance of dataflow graph automatically.

# Compare to other popular open-source ML libraries

**Mixture of different approaches**

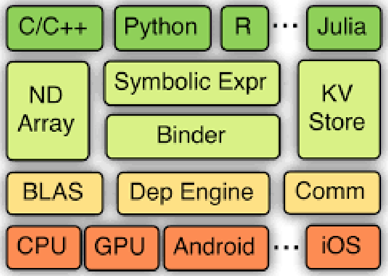
Combined effort from Amazon resulted in MXNet (or mix-net, previously called CXXNet), intending to blend advantages of different approaches.

## Mixture of different approaches

Combined effort from Amazon resulted in MXNet (or mix-net, previously called CXXNet), intending to blend advantages of different approaches.

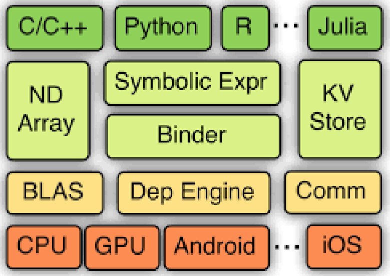
| System | Core Lang | Binding Langs | Devices (beyond CPU) | Distri-buted | Imperative Program | Declarative Program |
|---|---|---|---|---|---|---|
| Caffe [7] | C++ | Python/Matlab | GPU | × | × | √ |
| Torch7 [3] | Lua | - | GPU/FPGA | × | √ | × |
| Theano [1] | Python | - | GPU | × | × | √ |
| TensorFlow [11] | C++ | Python | GPU/Mobile | √ | × | √ |
| MXNet | C++ | Python/R/Julia/Go | GPU/Mobile | √ | √ | √ |

[Chen et. al, 2016]




Similar to other Machine Learning systems, MXNet embeds a **domain-specific language** (DSL) into a host language (e.g. Python, Lua, C++).

# MXNet - Declarative Symbolic Expressions

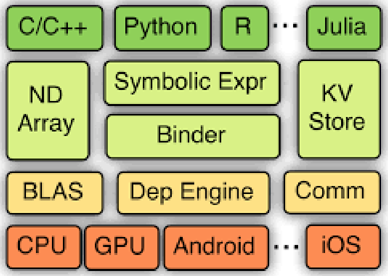



MXNet uses **multi-output symbolic expressions**, `Symbol`, declare the computation graph:

- Symbols are composited by operators, such as matrix operations or complex neural network layers.
- An operator can take several input variables, produce more than one output variables, and have internal state variables.

THE UNIVERSITY OF CHICAGO

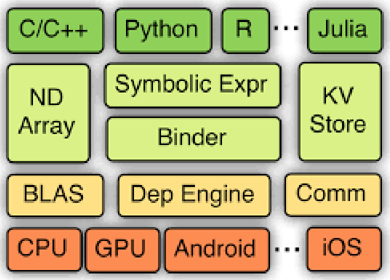
# MXNet - Declarative Symbolic Expressions




MXNet uses **multi-output symbolic expressions**, `Symbol`, declare the computation graph:

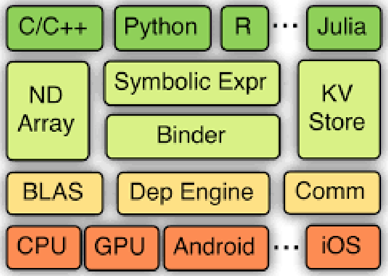
- Symbols are composited by operators, such as matrix operations or complex neural network layers.
- An operator can take several input variables, produce more than one output variables, and have internal state variables.

# MXNet - Imperative Tensor Computation



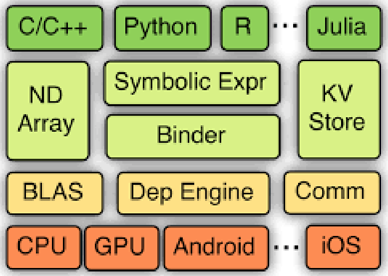

MXNet offers `NDArray` with **imperative tensor computation** to fill the gap between the declarative symbolic expression and the host language.

# MXNet - Declarative vs. Imperative (1)



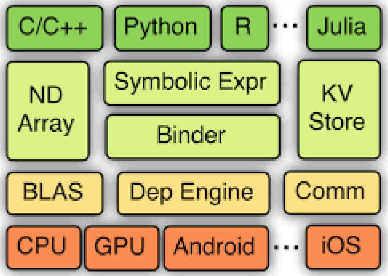

**Declarative** programs offer **clear boundary** on the global computation graph, discovering **more optimization** opportunity, whereas **imperative** programs offer **more flexibility**.

# MXNet - Declarative vs. Imperative (2)




- **Declarative** programming is useful in **specifying the computation structure** in neural network configurations.
- **Imperative** programming is more natural for **parameter updates** and **interactive debugging**.

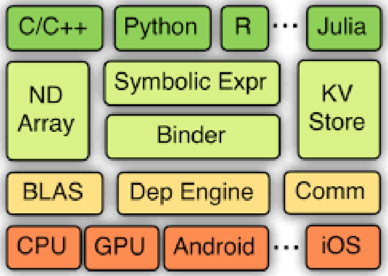
# MXNet - Data Synchronization Over Devices




The `KVStore` is a distributed key - value store for data synchronization over multiple devices. It supports two primitives:

1. **Push:** pushing a key-value pair from a device to the store.
2. **Pull:** pulling the value on a key from the store.

# MXNet - Data Synchronization Over Devices



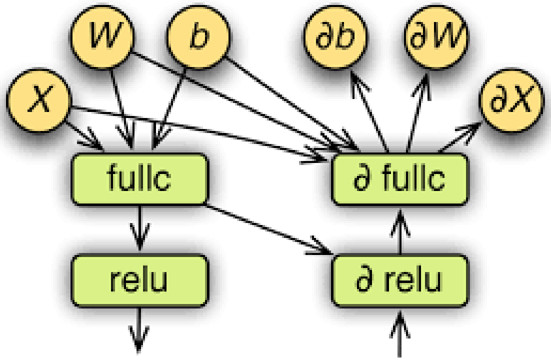

The `KVStore` is a distributed key - value store for data synchronization over multiple devices. It supports two primitives:

1. **Push:** pushing a key-value pair from a device to the store.
2. **Pull:** pulling the value on a key from the store.

**Question**

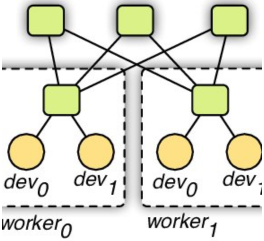
What are the equivalent Google technologies?

# Implementaiton - Computation Graph



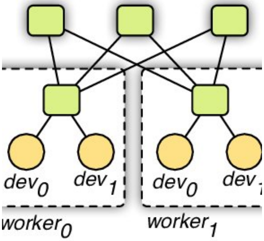

Computation graph for both forward and backward

# Implementaiton - Data Communication




Data communication of data centers

# Implementaiton - Data Communication

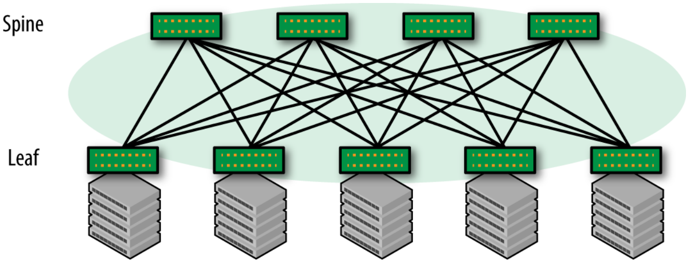



Data communication of data centers

**Question**

What is the name of this topology?

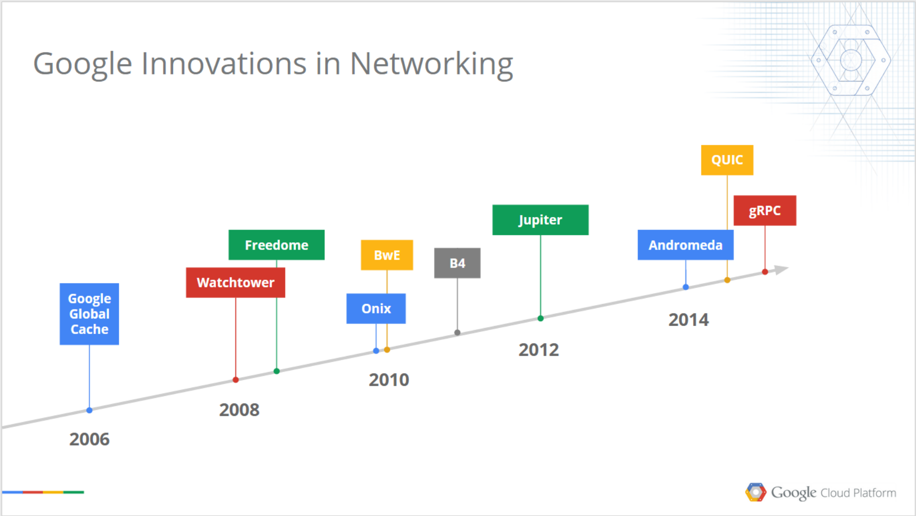
# Implementaiton - CLOS topology



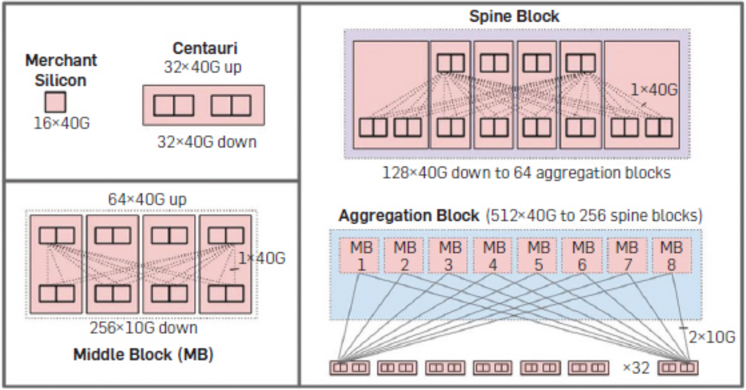

## CLOS topology

In the field of telecommunications, a Clos network is a kind of multi-stage circuit-switching network which represents a theoretical idealization of practical, multistage switching systems. It was invented by Edson Erwin in 1938 and first formalized by Charles Clos in 1952.

# Amazon vs. Google

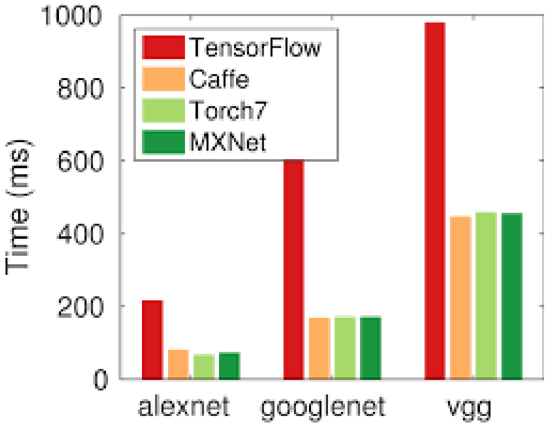


# Google's Jupiter (CLOS topology)




Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network, SIGCOMM'15, Google Inc.

# Performance




Tested on **convnet-benchmarks** with batch size 32 on a single Nvidia GTX 980 card. TensorFlow is always 2x slower (which might be due to its use of a lower CUDNN version).

# History lesson: What is Theano?

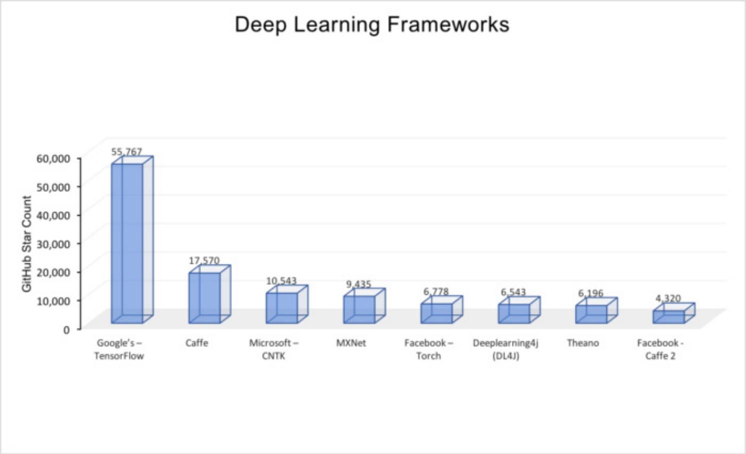
Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. Theano takes an declarative approach, enabling more global graph-aware optimization. CXXNet (and later, MXNet) adopts declarative programming (over tensor abstraction) and concrete execution, similar to Caffe.

**Legacy**

Forgotten, but influenced other Deep Learning frameworks.

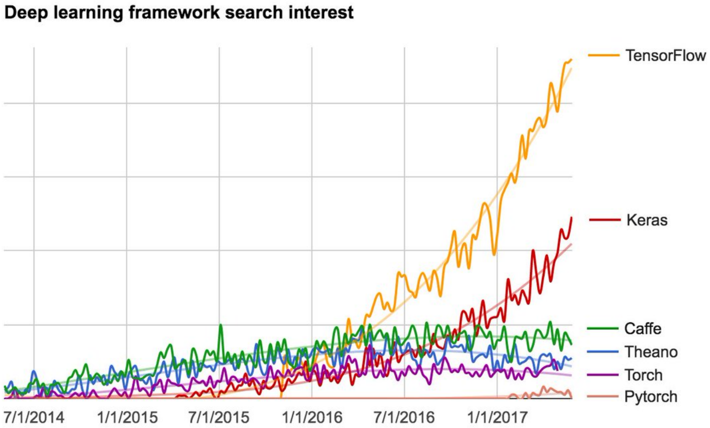
THE UNIVERSITY OF CHICAGO

# Popularity - Github repositories




May 2017

# Popularity - Search interest




February 2018