Summary of "TensorFlow: A System for Large-Scale Machine Learning"

By Matthew Lougheed (lougheem), Jiho Yoo (jihoyoo), Zineb Benameur El Youbi (zinebbe)

1. Problem and Motivation

Advances in machine learning have recently been driven by the creation of new machine learning models, big data, and software platforms that can train such large datasets on the new models. Before TensorFlow, Google used DistBelief, a system with a parameter server architecture. However. there were some key features that DistBelief lacked, which lead to the creation of Tensorflow in 2015. For one, it was difficult for Researchers to experiment with new optimization ideas with Distbelief. Since it was written in C++ for optimization purposes, its layer class was implemented in C++. This prevented researchers from experimenting with a new layer architecture with an unfamiliar language. Similarly, changing the optimization method for training algorithms required changing the implementation of the parameter server. Additionally, the updates applied to data were only applied using the paradigm of a forward pass followed by a backward pass. More advanced models (e.g. recurrent neural networks, adversarial networks, reinforcement learning) require different execution patterns. Another important limitation in DistBelief was its inability to run on environments smaller or more heterogeneous than on a cluster of computers in a datacenter.

2. Hypothesis

To achieve more flexibility for users, the architecture of TensorFlow is based on graph representations of the data and the operations performed. This is the abstraction made apparent to users: in the Dataflow graph, the mutable states of data and the operators on those states are represented as nodes, with edges showing changes in state.

The architecture of TensorFlow is based on graph representations and how they are distributed. TensorFlow can become a generic interface for not only expressing machine learning algorithms but also particular computer-intensive tasks.
Advantages in computations based on Tensor + Graph approach in a large-scale distributed environments.

3. Solution Overview

The main idea behind Tensorflow design principles is its execution model. Tensorflow differs from its predecessor by representing all computations and state in a single Dataflow graph and the communication between subcomputations is explicitly expressed. This last feature is crucial to simplifying the distributed execution and therefore allows Tensorflow to be deployed to a heterogeneous array of computing devices.

There are several elements in Dataflow graph: tensors, operations, stateful operations including variables and queues.
The Dataflow graph can be either executed partially or concurrently, adding more flexibility since it allows users to represent a large variety of neural network models without having to make modifications to the Tensorflow code.

Tensorflow supports dynamic control flow for supporting advanced machine learning algorithms that contains conditional iterative control flow. Also Tensorflow considers differentiation and optimization which is an important part of the machine learning algorithm. As a result, it allows efficient data parallelism in large dataset and large training models.

I addition to the design principles below, Tensorflow handles fault tolerance.
Since a model training process can take a long time, Tensorflow implemented a user checkpoint mechanism in order to “save to” and “recover from” a certain checkpoint. The user will have the ability to recover a training process from a previous checkpoint or simply save the training process to a new checkpoint.

4. Limitations and Possible Improvements

Although Tensorflow addressed many of the limitations that arose with Distbelief, there are still possible improvements that could be made. Tensorflow does not have a default flexible data flow policy that works for all users (May be fixed today). Additional improvements in systems algorithms could be made for automatic placement, kernel fusion, memory management, and scheduling. The paper states that Tensorflow will also require strong consistency requirements in the future.
Computation speed is also a field where Tensorflow is delaying behind compared to other Frameworks or libraries. Also, the only full language supported by Tensorflow is Python which makes it a downside as there is a rise of other languages in deep learning (currently).
Compared to other libraries as well, Tensorflow is difficult to use and has a more steep learning curve than PyTorch for example.

Summary of "PyTorch: An Imperative Style, High-Performance Deep Learning Library"

1. Problem and Motivation

Over the past few years, many deep learning frameworks emphasized performance and scalability. Frameworks such as Caffe, CNTK and Tensorflow utilized a static dataflow graph that represented computation. This allowed for visibility of computation ahead of time, which allowed for further improvements in performance and scalability. However, using a static dataflow graph came with the cost of ease of use, debugging and flexibility. Researchers had to wait for their entire code to execute to determine if their program was successful or not. Previous frameworks such as Chainer, Torch and Dynet have attempted to solve these issues, but it came with the expense of performance.

2. Hypothesis

By preserving Python's programming model and foregoing some of the benefits that could be made through static dataflow graphs, Pytorch, a Python Library, is able to provide an efficient and easy to use deep learning library. It utilizes eager execution which is a programming environment that evaluates operations immediately. Operations can return concrete values rather than constructing a computation graph. Eager execution allows for an intuitive interface using Python's data structures, easier to debug and provides a natural control flow compared to a complicated graph control flow.

3. Solution Overview

PyTorch is a dynamic library, it is very flexible and can be used as per the user's requirements and changes.
Some of the highlights in Pytorch design principles are:

- Simple Interface: It offers an easy to use API, thus it is very simple to operate and run like Python.
- Pythonic in nature: Pytorch smoothly integrates with the Python data science stack. Thus it can leverage all the services and functionalities offered by the Python environment.
- Dynamic Computation Graphing: in PyTorch the computational graph structure (of a neural network architecture) is generated during run time. The main advantage of this property is that it provides a flexible and programmatic runtime interface that facilitates the construction and modification of systems by connecting operations. This means a new computational graph is defined at each forward pass.
- Imperative Programming: PyTorch performs computations as it goes through each line of the written code. This is quite similar to how a Python program is executed. The biggest advantage of this design principle is that the code and programming logic can be debugged easily.

4. Limitations and Possible Improvements

In this paper, performance is one of the main limitations in PyTorch, this is due to the fact that PyTorch prioritized the ease of use over performance.
Currently the lack of performance can be significantly improved by using some tools and libraries to speed up the algorithm and running time.

Summary of Class Discussion

PyTorch vs TensorFlow
PyTorch is better for small scale projects and for fast prototyping in research. It is easier to use thanks to its simple interface and similarities to the Python language.
TensorFlow is better for large-scale deployments, especially when cross-platform and embedded deployment is a consideration. TensorFlow emphasizes heterogeneous device support. Recently, they released TensorFlow Federated which enables training to occur on decentralized data at the locations of the data.

Does TensorFlow have lazy evaluation?
Tensorflow has a lazy evaluation, this means it will first create a computational graph with the operations as the nodes of the graph and tensors to its edges and the execution occurs when the graph executed in a session.

How does TensorFlow archives Parallelism?
By using explicit edges to represent dependencies between operations, it is easy for the system to identify operations that can execute in parallel. TensorFlow is lazily executed, so the system is able to identify these edges and identify the parallel operations before executing.

It also supports synchronous and asynchronous parameter updates. While asynchronous updates are faster, not all models are designed to robustly handle asynchronous updates.

PyTorch's guiding design principle of prioritizing simplicity and ease of use over performance, to enable data scientists to design and test new models, is in line with most users' beliefs. It has become equally widespread as TensorFlow. As a result, with its early 2019 release, "TensorFlow 2.0 has been redesigned with a focus on developer productivity, simplicity, and ease of use" (TensorFlowBlog). The largest functional change it that it now executes eagerly by default. This misses optimizations in distributing work from different parts of the graph. However, this has allowed them to create a cleaner interface in Python that makes it easier to create models, similarly to PyTorch.