Summary of "TensorFlow: A system for large-scale machine learning"

Boyu Tian (bytian), Rui Liu (ruixliu), Haojun Ma (mahaojun)

Problem and Motivation

In recent years, machine learning has driven advances in many different fields due to the invention of more sophisticated machine learning models and the availability of large datasets for tackling problems in these fields. With machine learning being more and more popular, people need a system to help them (1) experiment with new models, which requires the system to be flexible enough to support experimentation and research into the new machine learning models and system-level optimizations, and (2) train the models on large datasets, which requires the system to be able to use hundreds of powerful servers efficiently. However, existing systems either lacks the flexibility or provides the flexibility only on a single machine.

Hypothesis

Advanced users sought for flexibility including: 1. Defining new layers; 2. Refining the training algorithms; and 3. Defining new training algorithms.

Many users want to hone their model locally on a GPU-powered workstation, before scaling the same code to train on a much larger dataset. After training a model on a cluster, the next step is to push the model into production. These tasks have some common computational structure, but it is necessary to use or create separate systems that satisfy the different performance and resource requirements of each platform.

Solution Overview

Design principles

1. Dataflow graphs of primitive operators: In the dataflow graph of TensorFlow, individual mathematical operators are represented as nodes in the graph. This makes it easier for users to compose novel layers using a high-level scripting interface. It also makes it easier to differentiate the machine learning models automatically.
2. Deferred execution: By deferring the execution until the entire program is available, TensorFlow can optimize the execution phase by using global information about the computation.
3. Common abstraction for heterogeneous accelerators: Each device must implement methods for (i) issuing a kernel for execution, (ii) allocating memory for inputs and outputs, and (iii) transferring buffers to and from host memory. Each operator can have multiple specialized implementations for different devices. By doing this, the common abstraction enables the same program to easily target different kinds of devices as required for training, serving, and offline inference.

Execution models

1. Dataflow graph: Operations are modeled as vertices in TensorFlow, while the data are modeled as tensors flowing along the edges, representing input to and output from each operation. TensorFlow also supports stateful operations (variables and queues), which contains mutable state that is read and/or written each time it executes.
2. Partial and concurrent execution: The API for executing a graph allows users to specify declaratively the subgraph that should be executed. Each invocation of the API is called a step. TensorFlow supports multiple concurrent steps on the same graph. These steps can share data and synchronize using stateful operations.
3. Distributed execution: Dataflow makes communication between subcomputations explicit, which simplifies distributed execution. TensorFlow runtime places operations on particular devices subject to implicit or explicit constraints. The placement algorithm computes a feasible set of devices for each operation, calculates the sets of operations that must be co-located, and selects a satisfying device for each colocation group. Expert users can also manually place the operations in order to improve performance.
4. Dynamic control flow: TensorFlow supports advanced machine learning algorithms that contain conditional and iterative control flow, such as recursive neural networks (RNN). For this purpose, the Switch and Merge primitive from classic dynamic dataflow architectures are used. Switch is a demultiplexer which takes a data input and a control input, and selects which of the two outputs should produce a value. Merge is a multiplexer which forwards at most one non-dead input to its output, or produce a dead output if both inputs are dead.

Extensibility case studies

1. Differentiation and optimization: TensorFlow includes a user-level library that differentiates a symbolic expression for a loss function and produces a new symbolic expression representing the gradients. Users can also experiment with different optimization algorithms, such as Momentum, AdaGrad, and L-BFGS. These methods can be built in TensorFlow using Variable operations and primitive mathematical operations without modifying the underlying system.
2. Training very large models: TensorFlow implements embedding layers in the TensorFlow graph as a composition of primitive operations. This subgraph contains three operations including Gather, which extracts a sparse set of rows from a tensor, Part, which divides the incoming indices into variable-sized tensors, and Stitch, which reassembles the partial results into a single result tensor. TensorFlow also includes libraries that exposes the abstraction of a shared parameters and build the graph based on the desired degree of distribution, thus rescuing the users from doing this manually.
3. Fault tolerance: TensorFlow implements user-level checkpointing for fault tolerance with two operations: Save and Restore. Users have the flexibility to apply different policies to subsets of the variables in a model, or customize the checkpoint retention scheme.
4. Synchronous replica coordination: Since GPUs enable training with fewer machines, synchronous training may be faster than asynchronous training. To mitigate stragglers, TensorFlow implements backup workers, which are similar to MapReduce backup tasks except that backup workers in TensorFlow run proactively. All workers, including the backup workers, run together, and only the first m of n updates produced are used. The slowest responses are abandoned.

Limitations and Possible Improvements

1. TensorFlow cannot automatically determine placements of operations across devices that achieve close to optimal performance on a given set of devices.
2. It is desirable to separate device placement directives from the model definition code. This could provide a good way to manage and modify the device placement choices when needed.
3. TensorFlow uses checkpointing for fault tolerance. It would be more convenient for end-users if

checkpointing is done automatically by TensorFlow system itself.

Summary of Class Discussion

At the beginning of the class, Professor pointed out that there is no much new stuff in the TensorFlow system. All techniques are known before this paper. Although most existing systems use immutable data to simplify fault tolerance mechanism, TensorFlow decided to use mutable data because machine learning algorithms need to share parameters across iterations. Professor also mentioned that the idea of deep learning could be traced back to several decades ago, and it only becomes feasible and even popular now because of the availability of big data centers which can process very big data. Although deep learning and its supporting system, like TensorFlow are extremely popular, which brought lots of fame to persons working on them, Professor suggested us not to blindly chase hot topics. Instead, we should try to focus on what we believe is good. When the right time comes for our work to attract great attention, it is hard for others to catch up.

One question discussed in class is whether different losses are used for each layer. Generally speaking, only one loss function is used at the very end of the neural network to quantify the discrepancy of predicted value and the ground truth. Another thing raised in class is, a good system should provide good abstraction between layers, for example, between hardware and software. This will enhance the extensibility of the system because some layers may need to be redesigned, like an emerging new hardware. In this paper, authors mentioned that it is still an open problem to automatically partition the computational graph of TensorFlow. For this point, Professor mentioned that there is some recent paper on arXiv, which uses deep learning models to learn how to partition the computational graph. We also discussed how to generate batches for deep learning. Usually, every batch is just sampled from the entire dataset, which may or may not overlap with other batches. This simple way could achieve satisfactory results in practice. Some also asked how to use TensorFlow on mobile devices. It normally requires installation of TensorFlow on mobile devices if anyone want to use it on smart phones.