Student Name: Nikhil Doifode
Student ID#: 112714121

Review of “TensorFlow: A System for Large-Scale Machine Learning”

In recent years, machine learning has become increasingly important and pervasive across more and more industries. Along with collecting data and developing suitable algorithms, having a robust, high performance and flexible architecture for supporting these machine learning calculations is important for a lot of people working in the field. The paper describes a large scale machine learning tool TensorFlow which works with distributed file systems. The tool was developed at Google and derives motivation from DistBelief which is a first-gen system for scalable distributed training and inference. Following figure shows the architecture of the TensorFlow.

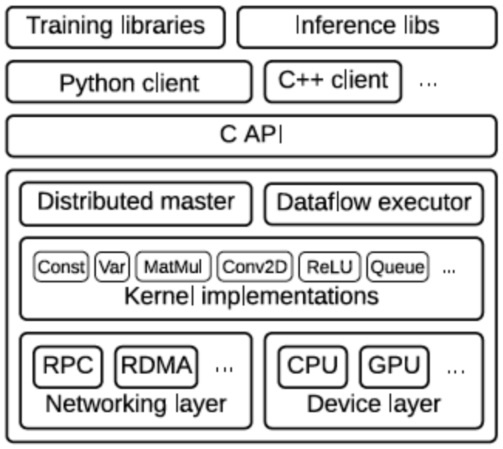


Figure 6: The layered TensorFlow architecture.

In TensorFlow, represents all computations and state in a single dataflow graph and communication between sub-computations is expressed explicitly. Each vertex represents a single computation, and an edge represents the output from a vertex or an input to another vertex. Data is modeled as dense n-dimensional arrays called tensors, and operations work on these tensors. Tensors are arbitrary dimensional arrays where underlying element type is specified at graph construction time. Special edges called as control dependencies are also present. No data flows along such edges but they indicate that the source node for the control dependence must finish before the destination node for the control starts execution.

Execution of the dataflow graph can be done partially or concurrently, adding to its flexibility, since it allows users to represent a wide variety of neural network models without having to make modifications to the internal code for TensorFlow. The decision to make communication between sub-computations explicit simplifies dataflow execution, which allows a TensorFlow to be deployed to a heterogeneous array of computing devices. Support for both conditional and iterative control flow allows TensorFlow to support more sophisticated algorithms like recurrent neural networks.

In addition to this support, TensorFlow includes a user-level library that gives users the ability to add functionality without having to hardcoded values. For example, differentiation and backpropagation can be easily implemented, allowing users to experiment with a wide range of optimization algorithms like stochastic gradient descent and momentum, to name a few.

To be able to handle very large datasets or computations as a distributed representation, TensorFlow implements sparse embedding layers as a composition of primitive operations. The Gather operator extracts a sparse set of rows from a tensor, while Part (dynamic partition) divides indices into variable-sized tensors that contain indices for each shard, and Stitch operations, which reassembles partial results from each shard into a single result tensor. Consistent with TensorFlow's design philosophy of abstracting out the low level details, these graphs do not have to be constructed manually. In order to provide fault tolerance, TensorFlow provides user-level checkpointing. It does not attempt to provide consistent checkpoints without direct user direction (specifying a synchronous approach instead), but this is usually not a problem due to the fact that neural networks and stochastic gradient descent are generally robust to small variations. Following figure shows the dataflow graph in TensorFlow.

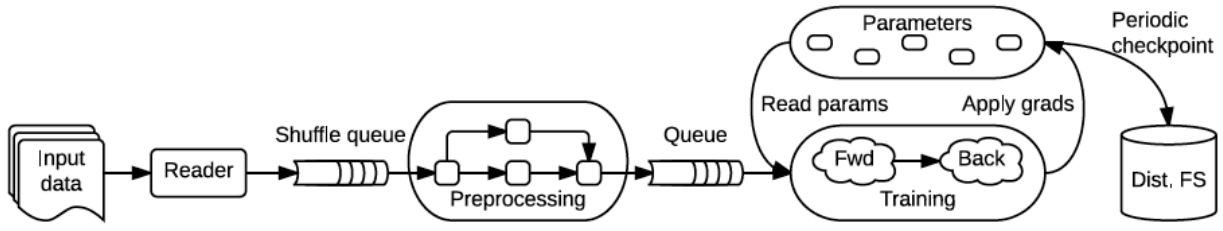


Figure 2: A schematic TensorFlow dataflow graph for a training pipeline, containing subgraphs for reading input data, preprocessing, training, and checkpointing state.

When TensorFlow was ran on both image classification and language modeling, results showed that TensorFlow has less overhead, is scalable and flexible.
Following figure shows comparison results for Image Classification.

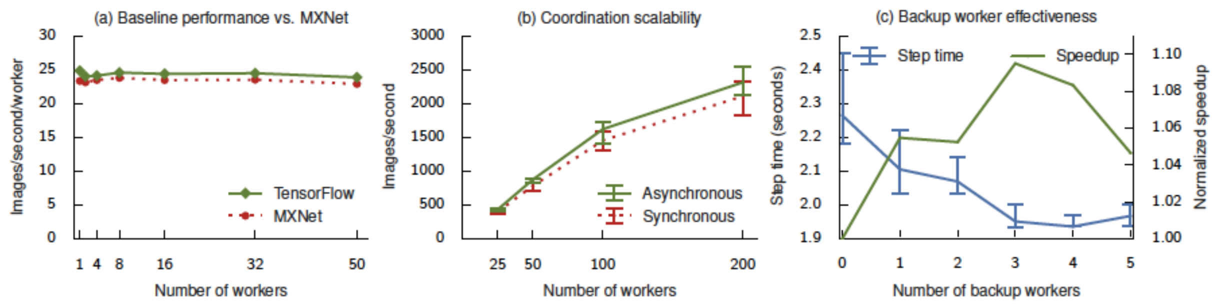
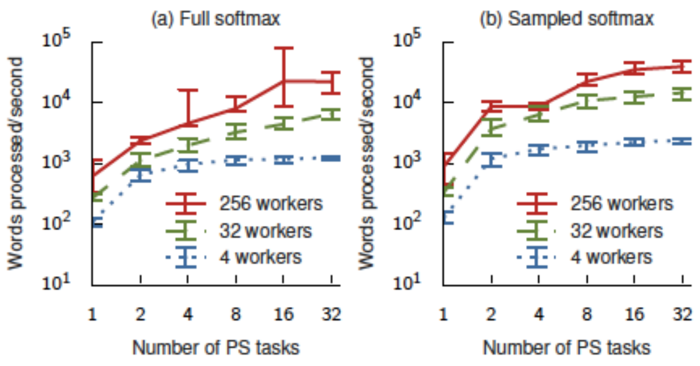


Figure 8: Results of the performance evaluation for Inception-v3 training (§6.3). (a) TensorFlow achieves slightly better throughput than MXNet for asynchronous training. (b) Asynchronous and synchronous training throughput increases with up to 200 workers. (c) Adding backup workers to a 50-worker training job can reduce the overall step time, and improve performance even when normalized for resource consumption.

Following Figure shows result for Language Modelling. Increasing the number of PS tasks leads to increased throughput for language model training, by parallelizing the softmax computation. Sampled softmax increases throughput by performing less computation.



Strengths:

1. TensorFlow provides a state of the art machine learning library.
2. It has high performance matching the best in the industry.
3. Packages are available that will let user easily program voice recognition, machine translation, video tagging, and other advanced artificial intelligence tasks.
4. It provides unique approach allows monitoring the training progress of your models and tracking several metrics.
5. It has great community support due to large number of users.

Weakness:

1. The only GPUs supported are Nvidia GPUs.
2. Some machine learning packages support more types of models out of the box.
3. The only fully supported programming language is Python.

Questions:

1. What is the difference between TensorFlow and DistBelief?

Ans:

DistBelief was based on the parameter server architecture, and satisfied most of Google's scalable machine learning requirements. However, this architecture lacked extensibility, because adding a new optimization algorithm, or experimenting with an unconventional model architecture would require users to modify the parameter server implementation. Not all the users are comfortable with making those changes due to the complexity of the high-performance parameter server implementation.

In contrast, TensorFlow provides a high-level uniform programming model that allows users to customize the code that runs in all parts of the system, and experiment with different optimization algorithms, consistency schemes, and parallelization strategies in userspace.

TensorFlow is based on the dataflow architecture. Dataflow with mutable state enables TensorFlow to mimic the functionality of a parameter server, and even provide additional flexibility. Using TensorFlow, it becomes possible to execute arbitrary dataflow subgraphs on the machines that host the shared model parameters.

2. How do differentiation and optimization work in TensorFlow?

Ans:

TensorFlow includes a user-level library that differentiates a symbolic expression for a loss function and produces a new symbolic expression representing the gradients. For example, given a neural network as a composition of layers and a loss function, the library will automatically derive the backpropagation code. The differentiation algorithm performs breadth-first search to identify all of the backwards paths from the target operation (e.g., a loss function) to a set of parameters, and sums the partial gradients that each path contributes. TensorFlow has extended this algorithm to differentiate conditional and iterative sub-computations by adding nodes to the graph that record the control flow decisions in the forward pass, and replaying those decisions in reverse during the backward pass.

Optimization algorithms compute new values for the parameters in each training step. However, there are many more advanced optimization schemes that are difficult to express as a single write operation. For example, the Momentum algorithm accumulates a “velocity” for each parameter based on its gradient over multiple iterations, then computes the parameter update from that accumulation. Several Optimization algorithms have been implemented on top of TensorFlow, including Momentum, AdaGrad, AdaDelta, RMSProp, Adam, and L-BFGS. These can be built in TensorFlow using Variable operations and primitive mathematical operations without modifying the underlying system.

3. How does synchronization work in TensorFlow?

Ans:

TensorFlow provided synchronization with help of queues. TensorFlow includes several queue implementations, which support more advanced forms of coordination. The simplest queue is FIFOQueue, which owns an internal queue of tensors, and allows concurrent access in first-in-first-out order. Other types of queues dequeue tensors in random and priority orders, which ensure that input data are sampled appropriately. Like a Variable, the FIFOQueue operation produces a reference handle that can be consumed by one of the standard queue operations, such as Enqueue and Dequeue. These operations push their input onto the tail of the queue and, respectively, pop the head element and output it. Enqueue will block if it’s given queue is full, and Dequeue will block if it’s given queue is empty. When queues are used in an input preprocessing pipeline, this blocking provides backpressure; it also supports synchronization.

While running SGD, the TensorFlow graph enables users to change how parameters are read and written when training a mode. In the asynchronous case (a), each worker reads the current values of parameters when each step begins, and applies its gradient to the (possibly different) current values at the end: this approach ensures high utilization, but the individual steps use stale parameter values, making each step less effective. The simple synchronous version (b) accumulates updates from all workers before applying them, but slow workers limit overall throughput. To mitigate stragglers, we implement backup workers (c). Backup workers run proactively, and the aggregation takes the first m of n updates produced.

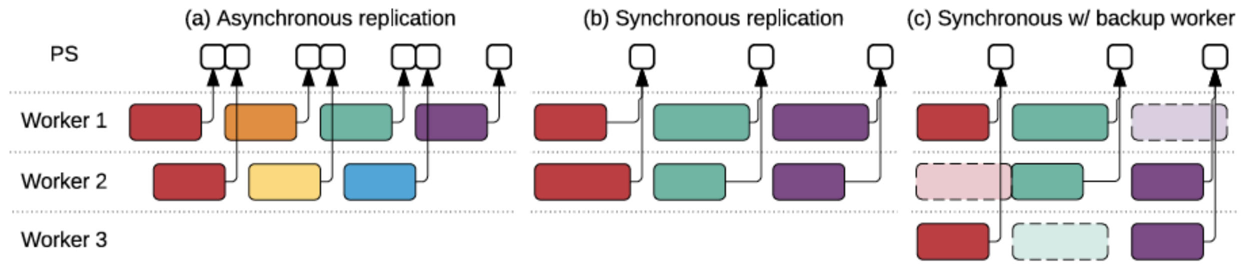

(a) Asynchronous replication
(b) Synchronous replication
(c) Synchronous w/ backup worker
PS
Worker 1
Worker 2
Worker 3

Figure 5: Three synchronization schemes for parallel SGD. Each color represents a different starting parameter value; a white square is a parameter update. In (c), a dashed rectangle represents a backup worker whose result is discarded.

References:

TensorFlow: A System for Large-Scale Machine Learning - Abadi