TensorFlow: A System for Learning-Scale Machine Learning

Google Brain

The Problem

– – –

- Machine learning is everywhere
- This is in large part due to:
  1. Invention of more sophisticated machine learning models
  2. Availability of large datasets to solve problems
  3. *Development of software platforms for training such models on these datasets*
- Development of scalable and flexible machine learning systems can have wide-ranging impact
- TensorFlow is (the 2016) solution from Google Brain

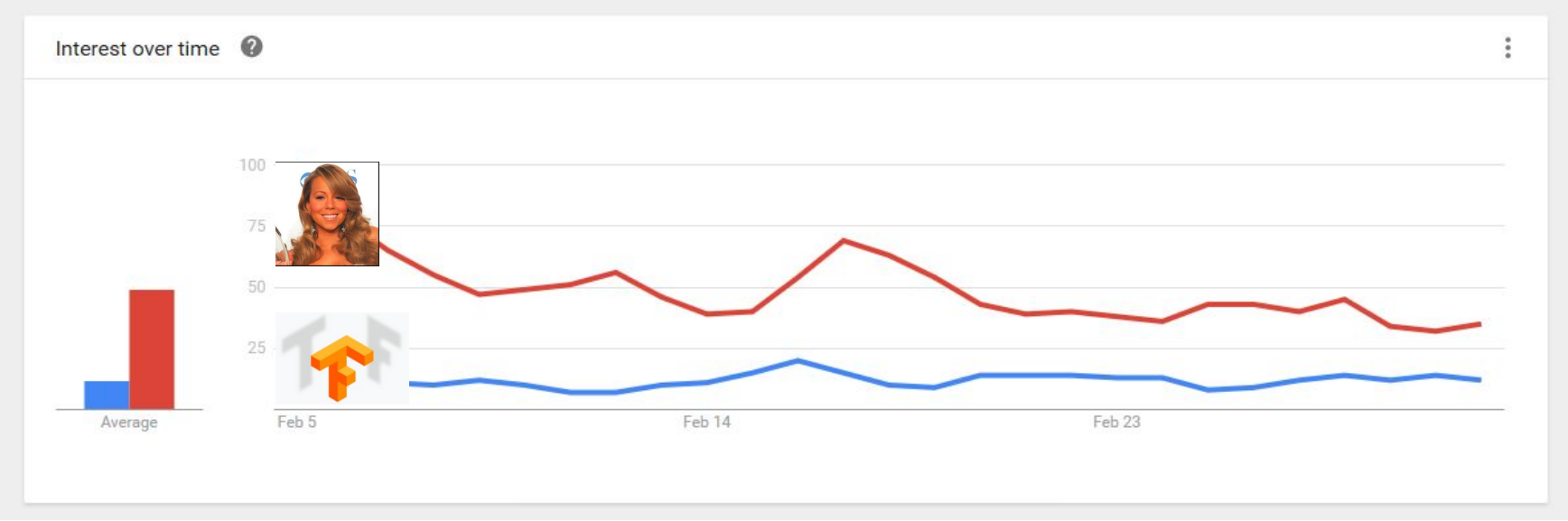
The Problem

- TensorFlow is a system that allows users to experiment with new models, train them on large datasets, and move them into production
- Successor to popular DistBelief (first generation distributed training and inference system) but varies in some meaningful ways
- Unifies computation and state in a single dataflow graph (more on this)
- Seamless design and deployment for heterogeneous clusters comprising CPU, GPU

Why is the Problem Interesting?

- Good solutions to this problem can have wide-ranging impact
  - Rise of devices better suited to machine learning computation such as GPUs, which result in heterogeneous clusters (traditional systems do not handle this well)
  - More complex learning algorithms such as RNN and reinforcement learning are not served well under existing infrastructure
- Upon release a large number of groups at Google deployed TensorFlow in production
- It was released as an open-source software
  - Over 14,000 people have forked the source code repository
  - The binary distribution has been downloaded over one million times
  - Dozens of machine learning models that use TensorFlow have been published

Just How Interesting?



Related Work: Limitations of DistBelief

– – –

- Based off the first-generation system: DistBelief (2011-2016), which had some issues
- *Layers:* Python based scripting interface for composing pre-defined layers, but layers are C++ classes. Experimenting with new layers was hard
- *Optimization Techniques:* experimenting with optimization methods outside of SGD is not easy and might not work well
- *NN structures:* Fixed execution pattern that works for simple feed-forward neural nets but fails for more advanced models (e.g, recurrent neural nets due to loops)
- *Heterogeneity*: DistBelief is not geared towards this.

Related Work: Other Frameworks

- **Single machine framework:**
  - Caffe programming model is similar to DistBelief and so shares a lot of the inflexibility issues
  - Torch allows fine-grained control over the execution order and memory utilization, but doesn't use dataflow graph
- **Batch dataflow systems:** e.g, MapReduce and improvements for machine learning algorithms
  - Require the input data to be immutable and all of the subcomputation to be deterministic
  - This makes updating a machine learning model an expensive operation

Related Work: Parameter Servers

- **Parameter servers:** this architecture meets many of the requirements
- MXNet is possibly the closest system in design to TensorFlow and even uses dataflow graphs
- Takes engineering effort to build the desired features into a parameter server
- What are the benefits of TensorFlow over parameter server? Why build a new system?

Technical Contribution: TensorFlow Building Blocks

- Uses a single dataflow graph to represent all computation and state in a machine learning algorithm

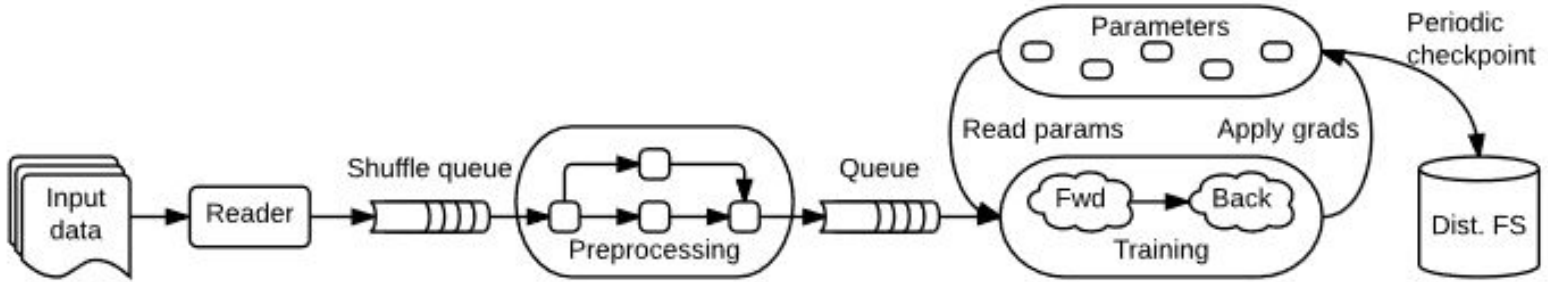


Figure 2: A schematic TensorFlow dataflow graph for a training pipeline, containing subgraphs for reading input data, preprocessing, training, and checkpointing state.

Building Blocks

- Vertex represents a unit of local computation and edge represents the output from or input to a vertex.
- *Operations* are the computations at vertices and *tensors* are values across the edges
  - Tensors represent the inputs to and results of common operations such as matrix multiplication
  - Operations take tensors as input and produce tensors as output. They can have mutable state that is read and/or written each time it is executed (advantages to this to be discussed)
- Simplifies distributed execution by making subcomputation communication explicit

Building Blocks

- Departing from traditional dataflow systems where graph vertices represent functional computation on immutable data,
    - the TensorFlow graph vertices may have mutable state that can be shared between different executions of the graph
    - Model supports multiple concurrent executions on overlapping subgraphs of the overall graph
- Unifying computation and state management allows programmers to experiment with parallelization schemes, optimizers, consistency schemes, etc
- Allows for partial and concurrent execution during training

Distributed Execution

- Each operation resides on a device in a particular task
- A device is responsible for executing a kernel for each operation assigned to it
- The placement algorithm places operation on device subject to constraints in the graph
- Once operation is places on a device, TensorFlow partitions the operations into per-device subgraphs
- TensorFlow is optimized for executing large subgraphs repeatedly with low latency

Dynamic Control Flow

- TensorFlow should support advanced machine learning algorithms, e.g, *RNN*
- Core of RNN is a recurrent relation, where the output for sequence element i is a function of some state that accumulates across the sequence
- Dynamic control flow enables iteration over sequences that have variable lengths, without unrolling the computation to the length of the longest sequence
- It does so by adding conditional and iterative programming constructs in the dataflow graph itself
    - Execution of iterations can overlap and TensorFlow can partition conditional branches and loop bodies across multiple devices and processes for efficiency

Fault Tolerance

- **Fault tolerance:** long-running jobs are likely to experience failure or pre-emption without adding too much overhead since failure might be rare
- Client library allows to construct appropriate graph structure and use save and restore for user-level checkpointing for fault tolerance
- This is customizable so the user can implement it as necessary and apply different checkpoints for different subsets of the graph
- Thoughts?

(A)synchrony

- **Synchronous replica coordination:** originally designed for asynchronous training, but have been experimenting with synchronous methods.

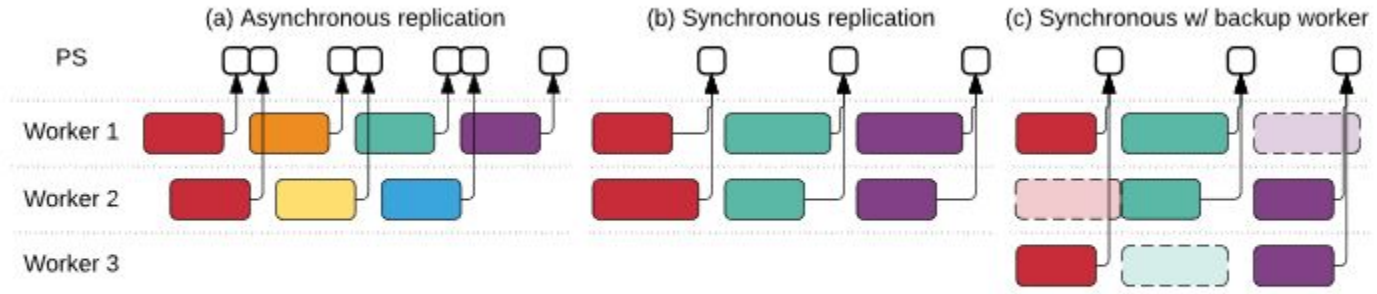
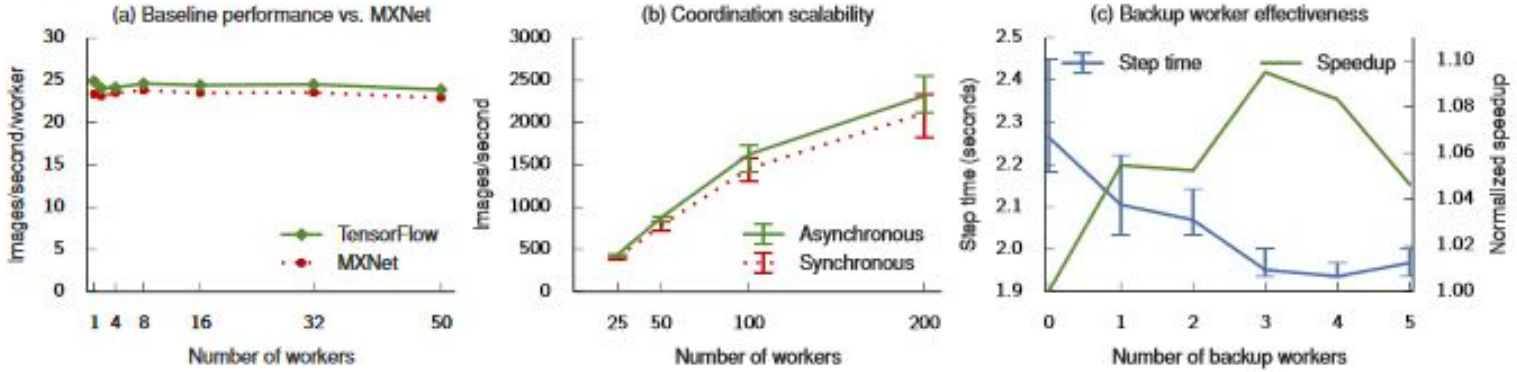


Figure 5: Three synchronization schemes for parallel SGD. Each color represents a different starting parameter value; a white square is a parameter update. In (c), a dashed rectangle represents a backup worker whose result is discarded.

Evaluation

- They evaluate TensorFlow on both image classification and language modeling
- They find that TensorFlow has less overhead, is scalable and flexible



When might it fail

- *Flexibility as a curse*: develop default policies that work well for all users (e.g., automatic optimization)
- Similarly, this heterogeneous resource utilization can add complexity, limiting usability
- TensorFlow tries to allocate all available GPU memory, and that can be undesirable

When might it fail

- Conflicts with already existing systems such as Theano (can compete for memory allocation)
- *“some users have begun to chafe at the limitations of a static dataflow graph, especially for algorithms like deep reinforcement learning. Therefore, we face the intriguing problem of providing a system that transparently and efficiently uses distributed resources, even when the structure of the computation unfolds dynamically”*

Future Work

- Large scale involvement and collaboration, as discussed above, will lead to fast improvement which has already been observed (e.g, multi-GPU support, graph visualization)
- Not clear what the benefits of TensorFlow are?
- Support for more complex settings
    - RNN support is lacking compared to Theano, easy gap to bridge
    - Support for reinforcement learning is weaker. Perhaps a system specifically for reinforcement learning

Future Work

– – –

- Some new research suggests that *synchronous* training might be faster, so more experimentation there. Called "*promising result*" in this paper
- *Automation*: for node scheduling instead of needing user specification
- Can have a system learn how to make a good device placement decision (using deep neural nets or reinforcement learning) [Mao et al, '16]
- Likewise automation of memory management and optimization technique
- Fault tolerance is left completely to the user, which might be an issue with new users, and automatic placement of checkpoints can be helpful

Future Work

- *Real-time support* (similar to robotics, self-driving cars), would be a great contribution
  - Can systems such as TensorFlow and parameter servers support these applications or do we need to design a new system?
- Fitting large ML models into *hand-held devices*, which are ubiquitous, is a challenge
  - There are interesting machine learning challenges to shrinking huge models. What systems techniques can help with this?
  - Can you run inference on a compressed neural network on your phone? How would that work?

Future Work

Thoughts from the group

(Johan)