TensorFlow: A system for large-scale machine learning

Martín Abadi et. al, 2016

Presented by Harrison Brown for R244
Background

• Originally built by Google engineers as successor to proprietary system for distributed training called DistBelief
  • DistBelief paper published, code not released
  • DistBelief uses parameter server architecture
    • Stateless workers, stateful parameter servers

• Machine learning algorithms
  • DAG that terminates with a loss function, backpropagation, SGD

• TensorFlow used internally at Google before being released as open source

• Dataflow architecture
4 Extensions

• New layers
  • DistBelief uses C++, limits ability for researchers to experiment

• Refining training Algorithms
  • SGD can be optimized in several ways (Adam, AdaGrad, etc)
  • DistBelief requires modifications of parameter server implementation

• New training algorithms
  • Need system that works well for other ML algorithms besides feed-forward NNs (ex. Adversarial networks, reinforcement learning, expectation-maximization etc)

• Ease of prototyping on local machines, GPU acceleration
Figure 2: A schematic TensorFlow dataflow graph for a training pipeline, containing subgraphs for reading input data, preprocessing, training, and checkpointing state.

https://www.tensorflow.org/g/tensorboard/r1/graphs
Comparison

• Torch
  • Imperative model, control over execution and performance
  • Lack of dataflow graph hurts experimentation, training, and ease of deployment

• Caffe
  • Easy to create new models with existing layers, but difficult for research into new models or optimizers, not extensible
  • Focus on CNNs (at time of paper) difficult to use RNNs

• Theano
  • Computation graph, mathematical operations, control flow and loops. Flexible
  • Difficult to scale

• MXNet
  • Computation graph, runs and scales very efficiently
Technical Design

• High-level scripting interface, ease of use, research oriented

• Individual mathematical operators are nodes in dataflow
  • Easier to compose novel layers

• Two phases
  • Define program as symbolic graph
  • Execute optimized version on available devices

• Common abstraction for accelerators
  • Operations on Tensors

• Tasks (PS tasks and worker tasks)
Execution

- Single dataflow graph
  - Supports multiple concurrent executions on overlapping subgraphs
- Vertices (Operations) with mutable state
  - Permits in place updates
  - Takes in $m$ tensors as input, $n$ tensors as output
- Tensors
  - N-dimensional arrays with small number of primitive types
- Can support asynchronous and synchronized execution
  - Lock free SGD is most common
- Allows operations to be manually placed
- Automatic differentiation of control flow constructs
Implementation

• C++ implementation for performance, can run on standard architectures
• Master obtains subgraphs for each device
• Executor handles requests from the master
• Tooling support (graph visualization, profiler for traces, etc)
Evaluation examples

- Designed to be fast, not the fastest
- MxNet comparison on image classification
- Demonstrate the scalability
Impact

- One of the most popular systems for machine learning
  - Adopted very quickly
  - Used widely in industry and in research
- Built for machine learning, but general enough for other computations
- The original TensorFlow is high-quality software, built to be extensible
  - Over 60,000 commits and ~2.4 million lines of code today
- TensorFlow (arguably) killed Theano as it is nearly a complete replacement
Issues

• Static dataflow graphs places limitations on some algorithms such as deep reinforcement learning
  • The Ray project attempts to address some of these issues
• Fault tolerance doesn't account for strong consistency potentially needed by some algorithms
  • Note, the overhead required has a drastic change in performance
• Stated MxNet performance nearly identical in this paper, however that may not be the case
Questions?
Sources


