TensorFlow:
Large-Scale Machine Learning on Heterogeneous Distributed Systems

by Google Research

presented by Weichen Wang
2016.11.28
Outline

➤ Introduction
➤ The Programming Model
➤ The Implementation
  ➤ Single Device Execution
  ➤ Multi-Device & Distributed Execution
➤ Extensions & Optimizations
➤ Auxiliary Tools
➤ Status & Experience
**What is TensorFlow?**

A multi-dimensional array  
A directed graph

TensorFlow

A directed graph of operations that process multi-dimensional arrays.
TensorFlow

➤ An open source library for general machine learning
➤ Developed by Google
➤ First released Nov 2015
➤ Apache 2.0 licensed
➤ Particularly useful for Deep Learning
➤ Very popular!
The Motivation

➤ DistBelief, Google’s first scalable distributed training and inference system, is not flexible enough

➤ Better understanding of problem space leads to some dramatic simplifications

➤ Define a standard way of expressing machine learning ideas and computations

➤ easy to use, efficient in execution
The Programming Model

- A directed graph representing a dataflow computation of multiple operations.
- Each node represents the instantiation of an operation.
- Nodes can maintain persistent states and branching and looping controls like Naiad.
- Edges represent tensor data flow between nodes (from outputs to input).
- A tensor is a typed multidimensional array.
- Control dependencies: special edges with no data flows along.
Expressing High-Level Machine Learning Computations

# First, build the graph.
c = tf.add(a, b)
# Then run it.
with tf.Session() as s:
    print(s.run(c, {a=1, b=2}))
3
```python
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))  # 100-d vector, init to zeroes
W = tf.Variable(tf.random_uniform([784, 100], -1, 1))  # 784x100 matrix w/rnd vals
x = tf.placeholder(name="x")  # Placeholder for input
relu = tf.nn.relu(tf.matmul(W, x) + b)  # Relu(Wx+b)
C = [...]  # Cost computed as a function of Relu

s = tf.Session()
for step in xrange(0, 10):
    input = ...construct 100-D input array ...
    result = s.run(C, feed_dict={x: input})  # Create 100-d vector for input
    print step, result  # Fetch cost, feeding x=input
```

**Figure 1:** Example TensorFlow code fragment

**Figure 2:** Corresponding computation graph for Figure 1
Implementation: Operations & Kernels

- An **operation** is an abstract computation on tensors
  - e.g., “matrix multiply”, or “add”.
  - represented by a node in the graph.
  - can have attributes.

- A **kernel** is a particular implementation of an operation that can be run on a particular type of device (e.g., CPU or GPU).

- A TensorFlow binary defines the sets of operations and kernels available via a registration mechanism, and this set can be extended by linking in additional operation and/or kernel definitions/registrations.
## Built-in Operations

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element-wise math ops</td>
<td>Add, Sub, <strong>Mul</strong>, Div, Exp, Log, Greater, Less...</td>
</tr>
<tr>
<td>Matrix ops</td>
<td><strong>Concat</strong>, Slice, <strong>Split</strong>, Constant, Rank, <strong>Shape</strong>...</td>
</tr>
<tr>
<td>Matrix ops</td>
<td><strong>MatMul</strong>, MatrixInverse, MatrixDeterminant...</td>
</tr>
<tr>
<td>Stateful ops</td>
<td><strong>Variable</strong>, Assign, AssignAdd...</td>
</tr>
<tr>
<td>NN building blocks</td>
<td><strong>SoftMax</strong>, Sigmoid, <strong>ReLU</strong>, <strong>Convolution2D</strong>...</td>
</tr>
<tr>
<td>Checkpointing ops</td>
<td>Save, Restore</td>
</tr>
<tr>
<td>Queue &amp; synch ops</td>
<td>Enqueue, Dequeue, MutexAcquire...</td>
</tr>
<tr>
<td>Control flow ops</td>
<td>Merge, Switch, Enter, Leave...</td>
</tr>
</tbody>
</table>
Implementation: Sessions, Placeholders, Variables

- **Sessions** manage resources for graph execution.
  - It encapsulates the environment in which operation are executed and tensors are evaluated.
- **Placeholders** must be fed with data on execution.
- A **variable** is a modifiable tensor that lives in TensorFlow’s graph of interactive operations.
  - In-memory buffers containing tensors.
  - Holds and updates parameters to be trained.
  - Must be initialized before they have values!
IMPLEMENTATION: CLIENTS, WORKERS, DEVICES

➤ A **client** communicates with the **master** using **session** interface.

➤ The master manages one or more **worker** processes.

➤ Each worker is responsible for arbitrating one or more computational **devices** and for executing operations on those devices.

➤ A device name is composed of pieces that identify its type, its index, and an identification of the task of the worker.

➤ Example: /job:localhost/device:cpu:0
**Single machine vs. distributed system**

Figure 3: Single machine and distributed system structure
Node Placement & Cross-Device Communication

- Each node (i.e. operation) is placed onto one of the devices.
- Node placement is done in topological order with a greedy heuristic based on cost estimation (execution + communication).
- Once node placement is done, the graph is partitioned into a set of subgraphs, one per device.
- Cross device edges are removed and replaced by Send &Recv edge.
DISTRIBUTED EXECUTION & FAULT TOLERANCE

➤ Similar to cross-device execution.

➤ Send/Recv communication uses gRPC, Google’s remote procedure call framework.

➤ When a failure is detected, the entire graph execution is aborted and restarted from scratch.

➤ Support of checkpoint and recovery.

➤ Variable are periodically saved and can be restored at restart.
EXTENSIONS: GRADIENT COMPUTATION

TensorFlow has built-in support for automatic gradient computation.

If a tensor $C$ depends on some set of tensors $\{X_k\}$, then there is a built-in function that will return the tensors $\{dC/dX_k\}$.

Gradient tensors are computed by backtracking from $C$ to each $X_k$, and adding a corresponding “gradient function” node to the TensorFlow graph for each operation on the backward path.

Figure 5: Gradients computed for graph in Figure 2
EXTENSIONS: PARTIAL EXECUTION

- Allows execution of an arbitrary subgraph of the whole graph
- Allows injection of arbitrary data along any edge of the graph (Feed)
- Allows arbitrary data retrieval from any edge of the graph (Fetch)

Figure 6: Before and after graph transformation for partial execution
**Extensions: Device Constraints & Control Flows**

➤ Device constraint examples:
  ➤ “only place this node on a device of type GPU”
  ➤ “this node can only be placed in /job:worker/task:17”
  ➤ “Colocate this node with the node named variable13”

➤ Control Flow: support of cyclic dataflow graph.

➤ **Switch, Merge**: express if-conditions.

➤ **Enter, Leave, NextIteration**: express iterations.

➤ distributed coordination mechanism is needed.
**Extensions: Queues & Containers**

- TensorFlow has built-in support of normal FIFO queue and a shuffling queue.

- A **Container** is the mechanism within TensorFlow for managing longer-lived mutable state.
  - Useful for sharing states between disjoint companions from different Sessions.
Optimizations

➤ Common subexpression elimination to remote redundant calculation

➤ Controlling data communication and memory usage
  ➤ Topological ordering of nodes to identify critical path
  ➤ Prioritize computation/communication on critical path

➤ Asynchronous kernel to support non-blocking computation

➤ Reuse pre-existing highly-optimized numerical libraries

➤ Lossy compression of data, similar to the DistBelief system
TensorFlow toolkit hierarchy

High-level “out-of-box” API
Inspired by scikit-learn

Components useful when building custom NN models

Python API gives you full control

C++ API is quite low-level

TF runs on different hardware

Run TF at scale

tf.[contrib.]learn

tf.layers, tf.losses, tf.metrics

Tensorflow Python

Tensorflow C++

CPU | GPU | TPU | Android

http://scikit-learn.org/
Figure 10: TensorBoard graph visualization of a convolutional neural network model

Figure 11: TensorBoard graphical display of model summary statistics time series data
# Create a summary writer.
print("Writing Summaries to \%s" % MODEL_DIR)
train_summary_writer = tf.train.SummaryWriter(MODEL_DIR)

# ..then, as part of defining the model graph..
loss_summary = tf.scalar_summary("loss", loss)
train_summary_op = tf.merge_summary([loss_summary])

# A training step: run the training op, write the summary info
_train, loss_value, tsummary = sess.run(
    [train_op, loss, train_summary_op],
    feed_dict={images_placeholder: images_feed,
               labels_placeholder: labels_feed})
train_summary_writer.add_summary(tsummary, step)
Figure 12: EEG visualization of multi-threaded CPU operations (x-axis is time in µs).
Performance

- Not much data for apples-to-apples comparison, but general observations are TensorFlow is slower than other common deep-learning framework such as Theano or Torch.

<table>
<thead>
<tr>
<th>Library</th>
<th>Time (ms)</th>
<th>forward (ms)</th>
<th>backward (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nervana</td>
<td>590</td>
<td>180</td>
<td>410</td>
</tr>
<tr>
<td>CuDNN-R3 (Torch)</td>
<td>615</td>
<td>196</td>
<td>418</td>
</tr>
<tr>
<td>CuDNN-R2 (Torch)</td>
<td>1099</td>
<td>342</td>
<td>757</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>1840</td>
<td>545</td>
<td>1295</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Library</th>
<th>Time (ms)</th>
<th>forward (ms)</th>
<th>backward (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CuDNN-R2 (Torch)</td>
<td>564</td>
<td>174</td>
<td>390</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>590</td>
<td>54</td>
<td>536</td>
</tr>
</tbody>
</table>
**Experiences**

- Build tools to gain insight into the exact number of parameters in a given model.
- Start small and scale up.
- Always ensure that the objective (loss function) matches between machine learning systems when learning is turned off.
- Make a single machine implementation match before debugging a distributed implementation.
- Guard against numerical errors.
- Analyze pieces of a network and understand the magnitude of numerical error.
THANK YOU!

Questions?