TENSORFLOW: LARGE-SCALE MACHINE LEARNING ON HETEROGENEOUS DISTRIBUTED SYSTEMS

By Sanjay Surendranath Girija
WHAT IS TENSORFLOW?

• “TensorFlow is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms”
• Dataflow-like model for computation
• Supported on a variety of hardware platforms – mobile, pcs, specialized distributed machines with hundreds of gpus
• Python and C++ Front Ends
• Open Source (www.tensorflow.org)
HISTORY OF TENSORFLOW

DistBelief

• 2011
• First generation scalable distributed training and inference system
• Machine Learning system built for deep neural networks

TensorFlow

• 2015
• 2nd generation system for implementation and deployment of largescale machine learning models
• More flexible programming model
• Better performance
APPLICATIONS

Used for both research and production

- Google Search - RankBrain
- Google Photos
- Speech Recognition
- Google Translate
- Inception Image Classification
- Gmail
- Inbox – SmartReply
- DeepMind
PROGRAMMING MODEL

• Dataflow like model
• Directed Graph with a set of Nodes
• Each node has zero or more inputs and outputs
• Control Dependencies – To enforce happens-before relationships and orderings
• Support for control flow operations, loops, conditions
TENSORS

- n-dimensional array or list
- Only tensors may be passed between nodes in the computation graph.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Python type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT_FLOAT</td>
<td>tf.float32</td>
<td>32 bits floating point</td>
</tr>
<tr>
<td>DT_DOUBLE</td>
<td>tf.float64</td>
<td>64 bits floating point</td>
</tr>
<tr>
<td>DT_INT8</td>
<td>tf.int8</td>
<td>8 bits signed integer</td>
</tr>
<tr>
<td>DT_INT16</td>
<td>tf.int16</td>
<td>16 bits signed integer</td>
</tr>
<tr>
<td>DT_INT32</td>
<td>tf.int32</td>
<td>32 bits signed integer</td>
</tr>
<tr>
<td>DT_INT64</td>
<td>tf.int64</td>
<td>64 bits signed integer</td>
</tr>
<tr>
<td>DT_UINT8</td>
<td>tf.uint8</td>
<td>8 bits unsigned integer</td>
</tr>
<tr>
<td>DT_STRING</td>
<td>tf.string</td>
<td>Variable length byte arrays. Each element of a Tensor is a byte array</td>
</tr>
<tr>
<td>DT_BOOL</td>
<td>tf.bool</td>
<td>Boolean</td>
</tr>
<tr>
<td>DT_COMPLEX64</td>
<td>tf.complex64</td>
<td>Complex number made of two 32 bits floating points: real and imaginary parts</td>
</tr>
<tr>
<td>DT_QINT8</td>
<td>tf.qint8</td>
<td>8 bits signed integer used in quantized Ops</td>
</tr>
<tr>
<td>DT_QINT32</td>
<td>tf.qint32</td>
<td>32 bits signed integer used in quantized Ops</td>
</tr>
<tr>
<td>DT_QINT8</td>
<td>tf.quint8</td>
<td>8 bits unsigned integer used in quantized Ops</td>
</tr>
</tbody>
</table>
OPERATION

• Node in a TensorFlow Graph that performs computation on tensors
• Takes zero or more Tensor objects as input, and produces zero or more Tensor objects as output.
• Polymorphic - Same Operation can be used for int32, float)
• Kernel - Particular implementation of an Operation that can be run on a particular type of device
• Eg: tf.size(), tf.reshape(), tf.concat(concat_dim, values, name='concat'), tf.matmul(), tf.matrix_inverse(input, adjoint=None, name=None)
  tf.Graph.create_op()
  tf.nn.softmax(), tf.sigmoid()
  tf.train.Saver.save(), tf.train.Saver.restore(sess, save_path)
SESSIONS

- **Session**: Object that encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated.
- Provides an interface to communicate with Master and Worker processes
- **Master**
  - Provides instructions to worker processes
- **Worker**:
  - Arbitrates access to computational devices
  - Executing graph nodes on the worker nodes

- **tf.Session()**
- Creating session object and closing a session
  
  ```python
  sess = tf.Session()
  sess.run(...)  
  sess.close()
  ```
- Using the context manager
  ```python
  with tf.Session() as sess:
      sess.run(...)
  ```
- Session with arguments
  ```python
  tf.Session.__init__(target='', graph=None, config=None)
  ```
VARIABLES AND RUN

• **Variable** - Operation that returns a handle to a persistent mutable tensor that survives across executions of a graph

• **Run**:
  • Runs one "step" of TensorFlow computation, by running the necessary graph fragment to execute every Operation and evaluate every Tensor in fetches
  • Takes a set of output names that need to be computed, set of tensors to be fed into the graph in place of certain outputs of nodes

```python
# Create two variables.
weights = tf.Variable(tf.random_normal([784, 200], stddev=0.35), name="weights")
biases = tf.Variable(tf.zeros([200]), name="biases")

# Pin a variable to CPU.
with tf.device("/cpu:0"):
    v = tf.Variable(...)

# Pin a variable to GPU.
with tf.device("/gpu:0"):
    v = tf.Variable(...)

# Pin a variable to a particular parameter server task.
with tf.device("/job:ps/task:7"):
    v = tf.Variable(...)

tf.Session.run(fetched, feed_dict=None, options=None, run_metadata=None)
```
INSTALLATION AND ENVIRONMENT SETUP

- https://www.tensorflow.org/versions/r0.10/get_started/os_setup.html
EXAMPLE - MNIST

• Handwritten digit recognition using Neural Network
• Uses Multinomial Logistic Regression (Softmax)
• 28 by 28 pixel MNIST image
• Input to the graph – Flattened 2d tensor of floating point numbers of dimensionality 784 each (28 * 28)
• Output - One-hot 10-dimensional vector indicating which digit the corresponding MNIST image belongs to

\[ y = \text{tf.nn.softmax}(\text{tf.matmul}(x, w) + b) \]
SUMMARIES

• Operation which serializes and stores tensor as strings.
• Summaries can be added to an event file.
• SummaryWriter class provides a mechanism to create an event file in a given directory and add summaries and events to it.

```python
# Create a summary writer, add the 'graph' to the event file.
writer = tf.train.SummaryWriter(<some-directory>, sess.graph)
```

• Eg :  # Outputs a Summary protocol buffer with scalar values
  - tf.scalar_summary(tags, values, collections=None, name=None)
  
    # Outputs a Summary protocol buffer with images.
  - tf.image_summary(tag, tensor, max_images=3, collections=None, name=None)
TENSORBOARD

• Used to visualize TensorFlow graphs, plot quantitative metrics
• Operates by reading TensorFlow events files containing summary data generated when running TensorFlow
• Launching TensorBoard: tensorboard --logdir=path/to/log-directory
• Currently supports five visualizations: scalars, images, audio, histograms, graph
IMPLEMENTATIONS

LOCAL

- Client, master and worker run on a single machine (single operating system process)

DISTRIBUTED

- Client, master, and workers run in different processes on different machines.
NODE PLACEMENT

• Map the computation onto the set of available devices.

• Cost Model – Contains estimates of the sizes (in bytes) of the input and output tensors for each graph node, along with estimates of the computation time required.

• Uses greedy heuristic based on effect of node placement on Completion time – Execution time + Time for communication
  • Statically estimated based on heuristics associated with different operation types
    OR
  • Measured based on an actual set of placement decisions for earlier executions

• User can also control the placement of nodes by specifying device constraints
CROSS DEVICE COMMUNICATION

- Cross-device edge from x to y is replaced by
  - Edge from x to a **Send node** in x’s subgraph
  - Edge from **Receive node** to y in y’s subgraph
  - Edge from **Send node** to **Receive node**
- Ensures that data for a tensor is sent only once between source and destination device pair
- Allows the scheduling of nodes on different devices to be decentralized into workers
  - Send and Receive nodes impart the necessary synchronization between different workers and devices
FEED AND FETCH

• FEED
  
  • Tensors are patched directly into any operation in the graph

```python
input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)
output = tf.mul(input1, input2)

with tf.Session() as sess:
    print(sess.run([output], feed_dict={input1:[7.], input2:[2.]}))

# output:
# [array([ 14.], dtype=float32)]
```

• FETCH
  
  • Output of any operation can be fetched by passing tensors to retrieve as an argument to `run()`

```python
input1 = tf.constant([3.8])
input2 = tf.constant([2.8])
input3 = tf.constant([5.8])
intermed = tf.add(input2, input3)
mul = tf.mul(input1, intermed)

with tf.Session() as sess:
    result = sess.run([mul, intermed])
    print(result)

# output:
# [array([[  21.],
#       [   7.]], dtype=float32), array([  14.], dtype=float32)]
```
PARTIAL EXECUTION

• Tensorflow allows execution of subgraph of a graph
• Both Feed and Fetch operations help in partial execution of subgraphs
Fault Tolerance

- Failure detection:
  - Error in a communication between a Send and Receive node pair
  - Periodic health-checks from the master process to every worker process

- Upon failure detection - Entire graph execution is aborted and restarted from scratch

- Support consistent check-pointing and recovery of the state on a restart:
  - Each variable node is connected to a Save node. Periodically writes contents of variables to persistent storage
  - Each variable is also connected to a Restore node. Restore nodes are enabled in the first iteration after a restart

- Checkpoint Files: Binary files that roughly contain a map from variable names to tensor values.
FAULT TOLERANCE – SAVE & RESTORE

• Saving Variables

```python
# Create some variables.
v1 = tf.Variable(..., name="v1")
v2 = tf.Variable(..., name="v2")
...
# Add an op to initialize the variables.
init_op = tf.initialize_all_variables()

# Add ops to save and restore all the variables.
saver = tf.train.Saver()

# Later, launch the model, initialize the variables, do some work, save the
# variables to disk.
with tf.Session() as sess:
sess.run(init_op)
# Do some work with the model.
...
# Save the variables to disk.
save_path = saver.save(sess, "/tmp/model.ckpt")
print("Model saved in file: %s" % save_path)
```

• Restoring Variables

```python
# Create some variables.
v1 = tf.Variable(..., name="v1")
v2 = tf.Variable(..., name="v2")
...
# Add ops to save and restore all the variables.
saver = tf.train.Saver()

# Later, launch the model, use the saver to restore variables from disk, and
# do some work with the model.
with tf.Session() as sess:
    # Restore variables from disk.
saver.restore(sess, "/tmp/model.ckpt")
print("Model restored.")
# Do some work with the model
...```
The Optimizer base class provides methods to compute gradients for a loss and apply gradients to variables.

A collection of subclasses of Optimizer implement classic optimization algorithms.

Available optimizers:

```python
class tf.train.GradientDescentOptimizer
class tf.train.AdadeltaOptimizer
class tf.train.AdagradOptimizer
class tf.train.MomentumOptimizer
class tf.train.AdamOptimizer
class tf.train.FtrlOptimizer
class tf.train.RMSPropOptimizer```

CONTROL FLOW

• Operations and classes that control the execution of operations and add conditional dependencies to graphs

• Provides support for loops, conditions, cases, logical comparisons, debugging

• Eg: tf.while_loop(cond, body, loop_vars, parallel_iterations=10, back_prop=True, swap_memory=False, name=None)
  • tf.case(pred_fn_pairs, default, exclusive=False, name='case')
  • tf.logical_and(x, y, name=None)
  • tf.equal(x, y, name=None)
  • tf.is_finite(x, name=None)
  • tf.is_nan(x, name=None)
  • tf.Assert(condition, data, summarize=None, name=None)
PERFORMANCE TRACING

• Internal tool called EEG used to collect and visualize very fine-grained information about the exact ordering and performance characteristics of the execution of TensorFlow graphs.

Figure 12: EEG visualization of multi-threaded CPU operations (x-axis is time in μs).
CONCLUSION

• Versatile model for implementing machine learning algorithms
• Support for distributed implementation
• Provides graph visualization using TensorBoard
• Logging and check-pointing
• Open source, growing community of users
• Currently being used within and outside Google for research and production
REFERENCES

• Paper
  • TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, Google Research

• Official Documentation
  • https://www.tensorflow.org/

• Installation
  • https://www.tensorflow.org/versions/r0.10/get_started/os_setup.html

• MNIST Sample Code
  • https://github.com/tensorflow/tensorflow/blob/r0.10/tensorflow/examples/tutorials/mnist/mnist_softmax.py