Distributed TensorFlow

Stony Brook University
CSE545, Spring 2019
Goals

- Understand TensorFlow as a data workflow system.
  - Know the key components of TensorFlow.
  - Understand the key concepts of distributed TensorFlow.
- Execute basic distributed tensorflow program.
- Establish a foundation to distribute deep learning models:
  - Convolutional Neural Networks
  - Recurrent Neural Network (or LSTM, GRU)
TensorFlow

A workflow system catered to numerical computation.

One view: Like Spark, but uses *tensors* instead of *RDDs*. 
TensorFlow

A workflow system catered to numerical computation.

One view: Like Spark, but uses tensors instead of RDDs.
TensorFlow

A workflow system catered to numerical computation.

One view: Like Spark, but uses tensors instead of RDDs.

A 2-d tensor is just a matrix.
1-d: vector
0-d: a constant / scalar

Note: Linguistic ambiguity:
Dimensions of a Tensor /=/ Dimensions of a Matrix

(i.stack.imgur.com)
TensorFlow

A workflow system catered to numerical computation.

One view: Like Spark, but uses tensors instead of RDDs.

Examples > 2-d:
Image definitions in terms of RGB per pixel
  Image[row][column][rgb]

Subject, Verb, Object representation of language:
  Counts[verb][subject][object]
TensorFlow

A workflow system catered to numerical computation.

One view: Like Spark, but uses tensors instead of RDDs.

Technically, less abstract than RDDs which could hold tensors as well as many other data structures (dictionaries/HashMaps, Trees, ...etc...).

Then, why TensorFlow?
TensorFlow

Efficient, high-level built-in linear algebra and machine learning optimization operations (i.e. transformations). Enables complex models, like deep learning.

Then, why TensorFlow?
TensorFlow

Efficient, high-level built-in **linear algebra** and **machine learning operations**.

enables complex models, like deep learning

(Bakshi, 2016, "What is Deep Learning? Getting Started With Deep Learning")
TensorFlow

Efficient, high-level built-in linear algebra and machine learning operations.

```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

```
Operations on tensors are often conceptualized as graphs:

```python
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))
W = tf.Variable(tf.random_uniform([784, 100], -1, 1))
x = tf.placeholder(name="x")
relu = tf.nn.relu(tf.matmul(W, x) + b)
C = [...]

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

TensorFlow

Operations on tensors are often conceptualized as graphs:

A simpler example:

c = tensorflow.matmul(a, b)
TensorFlow

Operations on tensors are often conceptualized as graphs:

example:

d = b + c
e = c + 2
a = d * e

Ingredients of a TensorFlow

**tensors**
- **variables** - persistent
  - mutable tensors
- **constants** - constant
- **placeholders** - from data

**operations**
- an abstract computation (e.g., matrix multiply, add)
  - executed by device **kernels**

**session**
- defines the environment in which operations **run**.
  - (like a Spark context)

**devices**
- the specific devices (cpus or gpus) on which to run the session.
Ingredients of a TensorFlow

- **tensors**
  - `variables` - persistent, mutable tensors
  - `constants` - constant
  - `placeholders` - from data

- **operations**
  - `tf.Variable(initial_value, name)`
  - `tf.constant(value, type, name)`
  - `tf.placeholder(type, shape, name)`

* technically, operations that work with tensors.

- **graph**

- **session**
  - defines the environment in which operations run.
  - (like a Spark context)

- **devices**
  - the specific devices (cpus or gpus) on which to run the session.
## Operations

**tensors**
- variables - persistent
- mutable tensors

**constants** - constant

**placeholders** - from data

### operations
an abstract computation (e.g. matrix multiply, add) executed by device kernels

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element-wise mathematical operations</td>
<td>Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...</td>
</tr>
<tr>
<td>Array operations</td>
<td>Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...</td>
</tr>
<tr>
<td>Matrix operations</td>
<td>MatMul, MatrixInverse, MatrixDeterminant, ...</td>
</tr>
<tr>
<td>Stateful operations</td>
<td>Variable, Assign, AssignAdd, ...</td>
</tr>
<tr>
<td>Neural-net building blocks</td>
<td>SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...</td>
</tr>
<tr>
<td>Checkpointing operations</td>
<td>Save, Restore</td>
</tr>
<tr>
<td>Queue and synchronization operations</td>
<td>Enqueue, Dequeue, MutexAcquire, MutexRelease, ...</td>
</tr>
<tr>
<td>Control flow operations</td>
<td>Merge, Switch, Enter, Leave, NextIteration</td>
</tr>
</tbody>
</table>
**Sessions**

- Place operations on devices
- Stores the values of variables (when not distributed)
- Carries out execution: `eval()` or `run()`

**session**
defines the environment in which operations `run`.
(like a Spark context)

**devices**
the specific devices (cpus or gpus) on which to run the session.

*operations*
an abstract computation
(e.g. matrix multiply, add)
executed by device *kernels*
Ingredients of a TensorFlow

**tensors**
- **variables** - persistent
- **mutable tensors**
- **constants** - constant
- **placeholders** - from data

**operations**
an abstract computation (e.g. matrix multiply, add) executed by device *kernels*

**graph**

**session**
defines the environment in which operations *run*. (like a Spark context)

**devices**
the specific devices (cpus or gpus) on which to run the session.

* technically, *operations* that work with tensors.
Distributed TensorFlow

Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors
Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices
Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

- Asynchronous Parameter Server
- Synchronous AllReduce (doesn’t work with Model Parallelism)
Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

- Asynchronous Parameter Server
- Synchronous AllReduce (doesn’t work with Model Parallelism)
Local Distribution

Multiple devices on single machine

Program 1
CPU:0
CPU:1

Program 2
GPU:0
Local Distribution

Multiple devices on single machine

with tf.device("/cpu:1")
    beta=tf.Variable(...)

with tf.device("/gpu:0")
    y_pred=tf.matmul(beta,X)

CPU:0   CPU:1   GPU:0
Cluster Distribution

Multiple devices on multiple machines

```
with tf.device("/cpu:1")
    beta = tf.Variable(...)
with tf.device("/gpu:0")
    y_pred = tf.matmul(beta, X)
```

Machine A
CPU: 0  CPU: 1

Machine B
GPU: 0
Distributed TensorFlow

Distributed:
- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:
- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:
- Asynchronous Parameter Server
- Synchronous AllReduce (doesn’t work with Model Parallelism)
Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

- Asynchronous Parameter Server
- Synchronous AllReduce (doesn’t work with Model Parallelism)
Cluster Distribution

Multiple devices on multiple machines

with tf.device("/cpu:1")
    beta=tf.Variable(...)

with tf.device("/gpu:0")
    y_pred=tf.matmul(beta,X)

Transfer Tensors
Cluster Distribution

Data Parallelism

```
... beta=tf.Variable(...) 
  pred=tf.matmul(beta,X)
... beta=tf.Variable(...) 
  pred=tf.matmul(beta,X)
... beta=tf.Variable(...) 
  pred=tf.matmul(beta,X)
... beta=tf.Variable(...) 
  pred=tf.matmul(beta,X)

CPU:0

CPU:1

GPU:0
```
Distributed TensorFlow

Distributed:

- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

Parallelisms:

- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

Model Updates:

- Asynchronous Parameter Server
- Synchronous AllReduce (doesn’t work with Model Parallelism)
Distributed TensorFlow

**Distributed:**
- Locally: Across processors (cpus, gpus, tpus)
- Across a Cluster: Multiple machine with multiple processors

**Parallelisms:**
- Data Parallelism: All nodes doing same thing on different subsets of data
- Graph/Model Parallelism: Different portions of model on different devices

**Model Updates:**
- Asynchronous Parameter Server
- Synchronous AllReduce (doesn’t work with Model Parallelism)
Asynchronous Parameter Server

Machine A

Machine B

(Geron, 2017: HOML: p.324)
Asynchronous Parameter Server

Parameter Server: Job is just to maintain values of variables being optimized.

Workers: do all the numerical “work” and send updates to the parameter server.

(Geron, 2017: HOML: p.324)
Synchronous AllReduce

Workers do computation, send parameter updates to other workers, and store parameter updates from other workers. Requires low latency communication.

(Geron, 2017: HOML: p.324)
Distributed TensorFlow: Full Pipeline

Parameters -- derived from gradients

TensorFlow has built-in ability to derive gradients given a cost function.

```
tf.gradients(cost, [params])
```
Parameters -- derived from gradients

Initial weight

Gradient

Global cost minimum $J_{\min}(w)$
Parameters -- derived from **gradients**.

**Linear Regression**: Trying to find “betas” that minimize:

$$\hat{\beta} = \arg\min_{\beta} \left\{ \sum_{i}^{N} (y_i - \hat{y}_i)^2 \right\}$$
Parameters -- derived from gradients.

Linear Regression: Trying to find “betas” that minimize:

\[
\hat{\beta} = \text{argmin}_\beta \left\{ \sum_{i} (y_i - \hat{y}_i)^2 \right\}
\]

\[
\hat{y}_i = X_i \beta
\]

Thus:

\[
\hat{\beta} = \text{argmin}_\beta \left\{ \sum_{i=0}^{N} (y_i - X_i \beta)^2 \right\}
\]
Parameters -- derived from gradients.

**Linear Regression:** Trying to find “betas” that minimize:

\[
\hat{\beta} = \arg\min_{\beta} \left\{ \sum_{i=0}^{N} (y_i - \hat{y}_i)^2 \right\}
\]

Thus:

\[
\hat{\beta} = \arg\min_{\beta} \left\{ \sum_{i}^{N} (y_i - X_i\beta)^2 \right\}
\]

In standard linear equation:

\[
y = mx + b \quad \text{let } x' = x + [1, 1, ..., 1]_N^T
\]

then, \( y = mx' \)

(if we add a column of 1s, \( mx + b \) is just matmul(m, x))
Parameters -- derived from **gradients**.

**Linear Regression:** Trying to find “betas” that minimize:

\[
\hat{\beta} = \arg\min_\beta \left\{ \sum_{i} (y_i - \hat{y}_i)^2 \right\}
\]

Thus:

\[
\hat{\beta} = \arg\min_\beta \left\{ \sum_{i=0}^{N} (y_i - X_i\beta)^2 \right\}
\]

Copyright 2014. Laerd Statistics.
Parameters -- derived from **gradients**.

**Linear Regression:** Trying to find “betas” that minimize:

\[ \hat{\beta} = \text{argmin}_\beta \left\{ \sum_{i=0}^{N} (y_i - \hat{y}_i)^2 \right\} \]

\[ \hat{y}_i = X_i \beta \quad \text{Thus:} \quad \hat{\beta} = \text{argmin}_\beta \left\{ \sum_{i=0}^{N} (y_i - X_i \beta)^2 \right\} \]

How to update? \[ \beta_{\text{new}} = \beta_{\text{prev}} - \alpha \times \text{grad} \]
Parameters -- derived from **gradients**.

**Linear Regression**: Trying to find “betas” that minimize:

\[
\hat{\beta} = \text{argmin}_{\beta} \left\{ \sum_{i=0}^{N} (y_i - \hat{y}_i)^2 \right\}
\]

\[
\hat{y}_i = X_i \beta
\]

Thus:

\[
\hat{\beta} = \text{argmin}_{\beta} \left\{ \sum_{i=0}^{N} (y_i - X_i \beta)^2 \right\}
\]

How to update? \[ \beta_{\text{new}} = \beta_{\text{prev}} - \alpha \times \text{grad} \]

(for gradient descent) \hspace{1cm} “learning rate”
Parameters -- derived from gradients.

Ridge Regression (L2 Penalized linear regression, $\lambda \| \beta \|_2^2$)

\[
\hat{\beta}_{ridge} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} (y_i - \sum_{j=1}^{m} x_{ij} \beta_j)^2 + \lambda \sum_{j=1}^{m} \beta_j^2 \right\}
\]

1. Matrix Solution:

\[
\hat{\beta}_{ridge} = (X^T X + \lambda I)^{-1} X^T y
\]
Demo

**Ridge Regression** (L2 Penalized linear regression, $\lambda \|\beta\|_2^2$)

\[
\hat{\beta}_{\text{ridge}} = \arg \min_{\beta} \left\{ \sum_{i=1}^{N} (y_i - \sum_{j=1}^{m} x_{ij} \beta_j)^2 + \lambda \sum_{j=1}^{m} \beta_j^2 \right\}
\]

2. Gradient descent solution
(Mirrors many parameter optimization problems.)

1. Matrix Solution:
\[
\hat{\beta}_{\text{ridge}} = (X^T X + \lambda I)^{-1} X^T y
\]
Gradients

Ridge Regression  (L2 Penalized linear regression, $\lambda \| \beta \|^2_2$)

\[ \hat{\beta}_{\text{ridge}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} (y_i - \sum_{j=1}^{m} x_{ij} \beta_j)^2 + \lambda \sum_{j=1}^{m} \beta_j^2 \right\} \]

Gradient descent needs to solve.
(Mirrors many parameter optimization problems.)

TensorFlow has built-in ability to derive gradients given a cost function.
Gradients

Ridge Regression  \( (\text{L2 Penalized linear regression}, \lambda \| \beta \|_2^2) \)

\[
\hat{\beta}_{\text{ridge}} = \arg \min_\beta \left\{ \sum_{i=1}^{N} (y_i - \sum_{j=1}^{m} x_{ij} \beta_j)^2 + \lambda \sum_{j=1}^{m} \beta_j^2 \right\}
\]

\textit{Gradient descent} needs to solve.  
(Mirrors many parameter optimization problems.)

TensorFlow has built-in ability to derive gradients given a cost function.

\[
tf.\text{gradients}(\text{cost}, [\text{params}])
\]
Gradients

TensorFlow has built-in ability to derive gradients given a cost function.

```
    tf.gradients(cost, [params])
```
Distributed TensorFlow: Full Pipeline