Distributed TensorFlow

Stony Brook University
CSE545, Fall 2017
Goals

- Understand TensorFlow as a workflow system.
- Know the key components of TensorFlow.
- Understand the key concepts of *distributed* TensorFlow.
- Do basic analysis in distributed TensorFlow.

Will not know but will be easier to pick up

- How deep learning works
- What is a CNN
- What is an RNN (or LSTM, GRU)
TensorFlow

A workflow system catered to numerical computation.

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

A workflow system catered to numerical computation.

Like Spark, but uses tensors instead of RDDs.

(i.stack.imgur.com)
TensorFlow

A workflow system catered to numerical computation.

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 $\neq$ Dimensions of a Matrix
TensorFlow

A workflow system catered to numerical computation.

Like Spark, but uses tensors instead of RDDs.

Example: Image definitions from assignment 2:

\[
\text{image}\{row\}[\text{column}][\text{rgbx}]
\]
TensorFlow

A workflow system catered to numerical computation.

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, what is valuable about TensorFlow?
TensorFlow

Efficient, high-level built-in linear algebra and machine learning operations (i.e. transformations).

enables complex models, like deep learning

Then, what is valuable about TensorFlow?
Efficient, high-level built-in linear algebra and machine learning operations.

```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 = [...] # 100-d vector, init to zeroes
# 784x100 matrix w/rnd vals
# Placeholder for input
# Relu(Wx+b)
# 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
```

Operations on tensors are often conceptualized as graphs:

A simpler example:

\[ d = b + c \]
\[ e = c + 2 \]
\[ a = d \times e \]
Ingredients of a TensorFlow

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

* technically, *operations* that work with tensors.

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

* technically, *operations* that work with tensors.

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

**operations** on abstract computation (e.g. matrix multiply, add) executed by device *kernels*
- `tf.Variable(initial_value, name)`
- `tf.constant(value, type, name)`
- `tf.placeholder(type, shape, name)`

**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**

### Category

<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

- **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**
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.

* technically, *operations* that work with tensors.
Demo

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

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

Matrix Solution:

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

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

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

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

Matrix Solution:

$\hat{\beta}_{ridge} = (X^T X + \lambda I)^{-1} X^T y$
Gradients

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

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

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

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

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

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

```
tf.gradients(cost, [params])
```
Gradients

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

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

Distributed TensorFlow

The layered TensorFlow architecture.

Distributed TensorFlow: Full Pipeline

Local Distribution

Multiple devices on single machine

Program 1

CPU:0  CPU:1

Program 2

GPU:0
Local Distribution

Multiple devices on single machine

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

Multiple devices on multiple machines

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

Transfer tensors between machines?
Cluster Distribution

"ps"

- task 0
  - TF Server
    - Master
    - Worker
  - CPU:0

"worker"

- task 0
  - TF Server
    - Master
    - Worker
  - CPU:1

- task 1
  - TF Server
    - Master
    - Worker
  - CPU:0

Machine A

Machine B

(Geron, 2017: HOML: p.324)
Cluster Distribution

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.
Summary

- TF is a workflow system, where records are always tensors
  - *operations* applied to tensors (as either Variables, constants, or placeholder)
- Optimized for numerical / linear algebra
  - automatically finds gradients
  - custom kernels for given devices
- “Easily” distributes
  - Within a single machine (local: many devices))
  - Across a cluster (many machines and devices)
  - Jobs broken up as parameter servers / workers makes coordination of data efficient