Big Data Analytics, The Class

Goal: Generalizations
A model or summarization of the data.

Data Frameworks
- Hadoop File System
- Spark
- Streaming
- MapReduce
- Tensorflow

Algorithms and Analyses
- Similarity Search
- Hypothesis Testing
- Graph Analysis
- Recommendation Systems
- Deep Learning
Limitations of Spark

Spark is fast for being so flexible

- Fast: RDDs in memory + Lazy evaluation: optimized chain of operations.
- Flexible: Many transformations -- can contain any custom code.
Limitations of Spark

Spark is fast for being so flexible

- Fast: RDDs in memory + Lazy evaluation: optimized chain of operations.
- Flexible: Many transformations -- can contain any custom code.

However:

- Hadoop MapReduce can still be better for extreme IO, data that will not fit in memory across cluster.
Limitations of Spark

Spark is fast for being so flexible

- Fast: RDDs in memory + Lazy evaluation: optimized chain of operations.
- Flexible: Many transformations -- can contain any custom code.

However:

- Hadoop MapReduce can still be better for extreme IO, data that will not fit in memory across cluster.
Limitations of Spark

Spark is fast for being so flexible

- Fast: RDDs in memory + Lazy evaluation: optimized chain of operations.
- Flexible: Many transformations -- can contain any custom code.

However:

- Hadoop MapReduce can still be better for extreme IO, data that will not fit in memory across cluster.
- Modern machine learning (esp. Deep learning), a common big data task, requires heavy numeric computation.

* this is the subjective approximation of the instructor as of February 2020. A lot of factors at play.
Limitations of Spark

Spark is fast for being so flexible

- Fast: RDDs in memory + Lazy evaluation: optimized chain of operations.
- Flexible: Many transformations -- can contain any custom code.

However:

- Hadoop MapReduce can still be better for extreme IO, data that will not fit in memory across cluster.
- Modern machine learning (esp. Deep learning), a common big data task, requires heavy numeric computation.

* This is the subjective approximation of the instructor as of February 2020. A lot of factors at play.
Learning Objectives

- 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)
What is TensorFlow?

A workflow system catered to numerical computation.

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

A workflow system catered to numerical computation.

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

![A multi-dimensional matrix](i.stack.imgur.com)
What is **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)
What is **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]`
What is **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?
What is 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?
What is TensorFlow?

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

enables complex models, like deep learning

What is 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
```

Operations on tensors are often conceptualized as graphs:

A simpler example:

\[ c = \text{tensorflow.mатмул}(a, b) \]
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

* technically, still operations

**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.
Ingredients of a TensorFlow

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

* technically, still **operations**

- `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.
# 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

<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>
Ingredients of a TensorFlow

- **Variables** - persistent
- **Mutable Tensors**
- **Constants** - constant
- **Placeholders** - from data

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

**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.
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.
Typical use-case: (Supervised Machine Learning)
Determine weights, $W$, of a function, $f$, such that $\varepsilon$ is minimized: $f(X|W) = Y + \varepsilon$
Typical use-case:

Determine weights, $\mathcal{W}$, of a function, $f$, such that $\epsilon$ is minimized: $f(X | \mathcal{W}) = Y + \epsilon$
Typical use-case:

Determine weights, $w$, of a function, $f$, such that $\epsilon$ is minimized: $f(x|w) = y + \epsilon$
Distributed TensorFlow

Typical use-case:

Determine weights, $\mathbf{W}$, of a function, $f$, such that $\varepsilon$ is minimized: $f(X \mid \mathbf{W}) = Y + \varepsilon$
Typical use-case:

Determine weights, $\mathbf{w}$, of a function, $f$, such that $|\varepsilon|$ is minimized:

$$f(X|\mathbf{W}) = \hat{Y}$$

$$Y = (X|\mathbf{W}) + \varepsilon$$

$$\varepsilon = \hat{Y} - Y$$
Distributed TensorFlow

Typical use-case:
Determine weights, $\mathcal{W}$, of a function, $f$, such that $\mathcal{E}$ is minimized:

$$f(X|\mathcal{W}) = \hat{Y}$$
$$Y = (X|\mathcal{W}) + \mathcal{E}$$
$$\mathcal{E} = \hat{Y} - Y$$

Typically, very complex!
Typical use-case:

Determine weights, $\mathcal{W}$, of a function, $f$, such that $\varepsilon$ is minimized: 

$$f(X|\mathcal{W}) = \mathcal{\hat{Y}}$$
$$Y = (X|\mathcal{W}) + \varepsilon$$
$$\varepsilon = \mathcal{\hat{Y}} - Y$$

$\mathcal{W}$ determined through gradient descent:

back propagating error across the network that defines $f$. 

$f$ given $\mathcal{w}_1, \mathcal{w}_2, ..., \mathcal{w}$

(typically, $p \geq m$)
Distributed TensorFlow

Typical use-case:

Determine weights, $\mathcal{W}$, of a function, $f$, such that $\varepsilon$ is minimized: $f(X|\mathcal{W})$. 

$\mathcal{W}$ determined through gradient descent:

back propagating error across the network that defines $f$.

$$f(X|\mathcal{W}) = \hat{Y}$$
$$Y = (X|\mathcal{W}) + \varepsilon$$
$$\varepsilon = \hat{Y} - Y$$

minimizes $\varepsilon$ on $N$ training examples