• What is TensorFlow?
  • 3.e Martin Abadi et al., Large-Scale Machine Learning on Heterogeneous Distributed Systems (TensorFlow)

• What is TensorFlow Eager Execution?
  • 3.f Agrawal et al., TensorFlow Eager: A Multi-stage, Python-embedded DSL for Machine Learning

• What is Keras?
  • https://keras.io/
What is TensorFlow?
• TensorFlow is a **scalable distributed** training and interference system developed by Google
• Provides easy portability between vastly different hardware configurations
• Based on Google’s previous DistBelief system
• Uses a dataflow-like model for computations described by directed *graphs*
• Allows for automatic gradient computation
• **Graph** – dataflow computation allowing persistent state, looping, and branching

• **Node** – instantiation of an operation with inputs/outputs

• **Operation** – computation (+, -, *, /, etc.)

• **Kernel** – device implementation of an operation

• **Tensor** – arbitrary dimensionality arrays

• **Control Dependencies** – graph edges enforcing execution order

• **Variables** – mutable tensor
• **Client** uses Session interface to talk with **Master** and **Worker Process(es)** which have access to **device(s)**

```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
```

# 100-d vector, init to zeroes
# 784x100 matrix w/rnd vals
# Placeholder for input
# Relu(Wx+b)
# Cost computed as a function
# of Relu

# Create 100-d vector for input
# Fetch cost, feeding x=input
- Local – all components in one machine
- Single Device
- Multiple Devices
- Distributed – components spread across multiple networked machines
• Local implementation
• Single Device
  • Nodes executed in order based on dependencies (think Tomasulo algorithm)
• Multiple Devices
  • Nodes placed using greedy heuristics onto feasible devices
  • Inter-device communication consolidated
• Distributed implementation
  • Similar to multiple devices implementation with added fault tolerance for communication over TCP/IP or RDMA
• Synchronous Data Parallelism
  • One entity manages global gradient update

• Asynchronous Data Parallelism
  • Devices continually provide gradient updates without waiting for global update
• **Model Parallel Training**
  - Portion out different tasks of a model to different devices

• **Concurrent Steps**
  - Run multiple instances of a model on a single device for better utilization
Where is the local vs. distributed comparison?
• “A future version of this white paper will have a comprehensive performance evaluation section of both the single machine and distributed implementations.”

• We’ve all been in a similar situation

• However…
• TensorFlow is an abstraction from physical hardware and complicated algorithms to give users an easy ML experience
• Allows researchers to focus on their work and not on getting the models running
• We, as computer scientists, understand overhead whereas other researches may not!
What is TensorFlow Eager Execution?
• TensorFlow requires the entire computation graph to be defined before executing (permits compiler optimizations)
• Partial Execution allows clients to execute arbitrary subgraphs for testing purposes
• TensorFlow Eager is a Python-embedded domain-specific language that combines **imperative** and **staged computations** into a single package for easy-use **differentiable programming**

• Imperative – tell computer **how** to do something vs. Declarative – tell computer **what** you want

• Staged computations – fully construct a dataflow graph
• TensorFlow is declarative which translates to compiler optimizations and good performance; however, it is more difficult to prototype

• TensorFlow Eager is an extension of TensorFlow for easier developing

• Uses a decorator, function, which traces Python function calls and generates underlying dataflow graphs (C++) from imperative code for better performance
How do you quantify the “tradeoff between execution speed and development speed”? 
• The motivation for TensorFlow Eager is to provide developers with an easier-to-use language for ML.

• The tradeoff, as explained in the paper, is between execution speed and development speed.

• But how is development speed quantified? Are there better ways to describe this?
What is Keras?
“Keras is an API designed for human beings, not machines.”

- Keras Documentation
• Keras is a Python API that abstracts away the graph model from TensorFlow and presents an easier workflow for Neural Network development.

<table>
<thead>
<tr>
<th>Keras is to Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow is to C</td>
</tr>
<tr>
<td>CUDA is to Assembly</td>
</tr>
</tbody>
</table>
```python
import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
                                     tf.keras.layers.Dense(512, activation=tf.nn.relu),
                                     tf.keras.layers.Dropout(0.2),
                                     tf.keras.layers.Dense(10, activation=tf.nn.softmax)])

model.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```
<table>
<thead>
<tr>
<th>Sequential Model API</th>
<th>Functional Model API</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Off-the-shelf” model structures</td>
<td>Non-standard model structures</td>
</tr>
<tr>
<td><em>Single</em> input/output</td>
<td><em>Multiple</em> input(s)/output(s)</td>
</tr>
<tr>
<td>Linear topology</td>
<td>Non-linear topology</td>
</tr>
<tr>
<td>Separate layers</td>
<td>Shared layers</td>
</tr>
<tr>
<td>Dense layers</td>
<td>Variable layers</td>
</tr>
</tbody>
</table>

![Diagram of Keras Sequential vs. Functional Model APIs]

```plaintext
Keras Sequential vs. Functional Model APIs
```
Why should we care about the functional model API?
• Researchers should always be thinking about why the cool thing they created is important and why should people care!
• Keras provides an API with limitless neural network possibilities, but fails to explain why that’s important
• “Lean Neural Networks for Radar Waveform Design” – Anthony Baietto
Questions?