Large-Scale Machine Learning

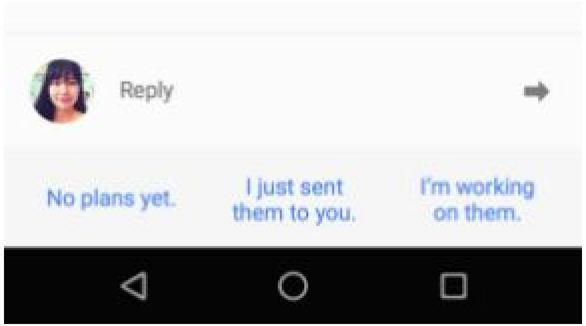
From DistBelief To TensorFlow

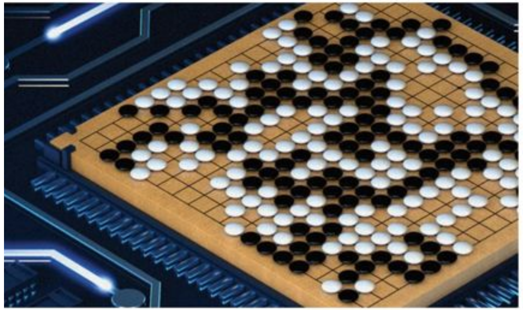
Overview of this Presentation

- Review two large-scale machine learning systems from Google
  - DistBelief
  - TensorFlow

Background: Massive Success of Deep Learning

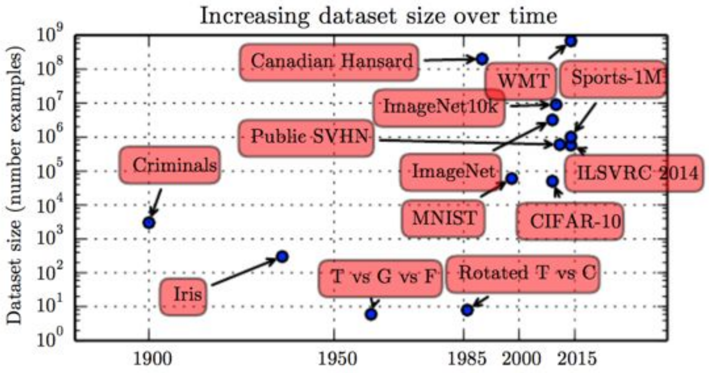
- Various real-world applications
    - Image recognition
    - Speech recognition
    - Natural language processing
    - Game play (e.g., Go)
    - …
- Beating existing machine-learning algorithms

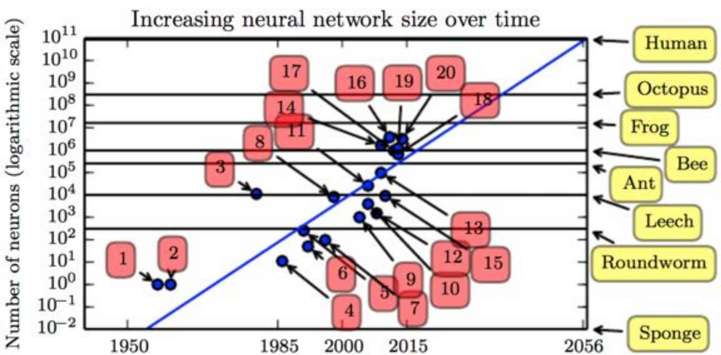




Challenges

- Require large amount of computational and storage resources
  - Increasing dataset size
  - Increasing number of neurons and connections
- Require rapid iteration on new learning algorithm development
  - Hot research topic that many people are actively working on
- Do not fit well with existing parallel/distributed programming models
  - E.g.) Mapreduce, Spark, graph execution engine, ...





What Google Has Been Building

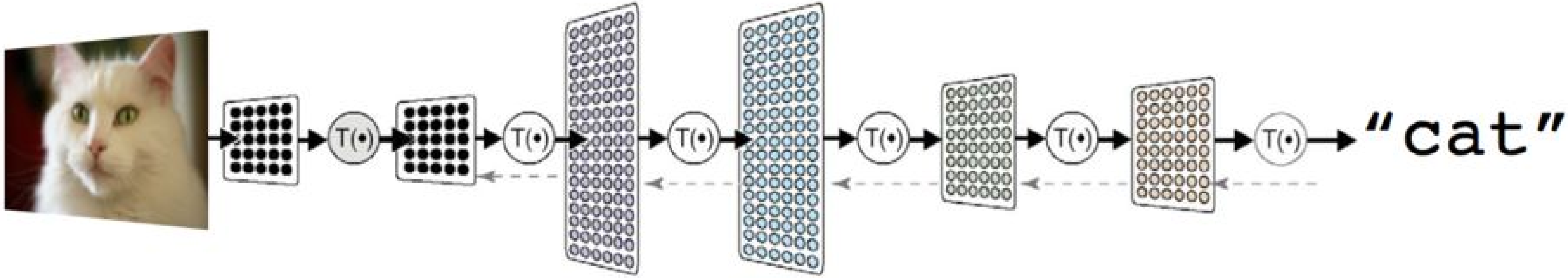
- DistBelief (2012)
  - Parallel and distributed execution of large-scale deep network

- TensorFlow (2015-)
  - More generalized data-flow execution engine
  - Still specialized for machine learning

DistBelief

Before Diving Into...

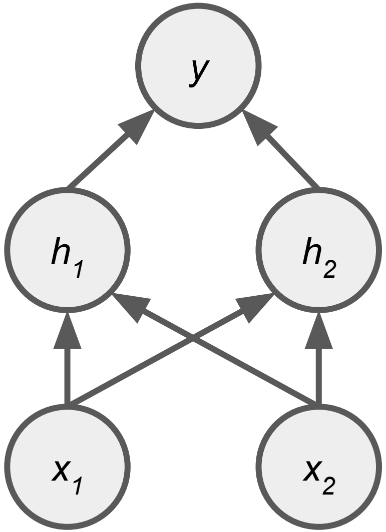
- What is **deep learning**?
  - The modern reincarnation of Artificial Neural Networks from the 1980s and 90s
  - A collection of simple trainable mathematical units, which collaborates to compute a complicated function
  - Compatible with supervised, unsupervised, and reinforcement learning



What is Neural Network?

- Approximate some function *f**
- Typically represented by composing many different functions
  - $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$
- Example: XOR

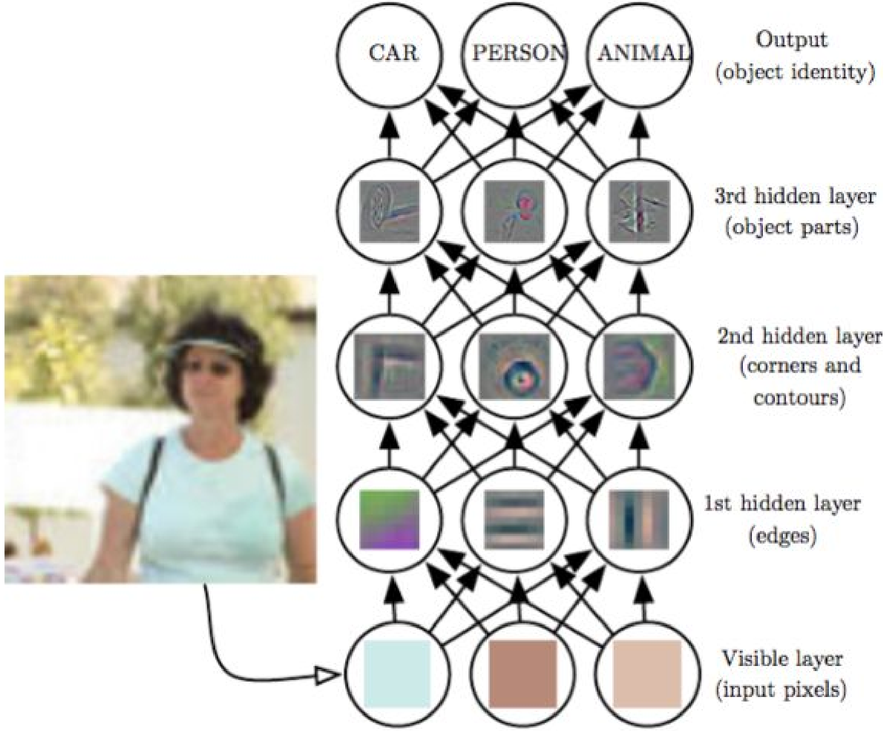
$$f(x) = \begin{bmatrix}1 & -2\end{bmatrix} max\{0, \begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} x + \begin{bmatrix}0\\ -1\end{bmatrix}\}$$

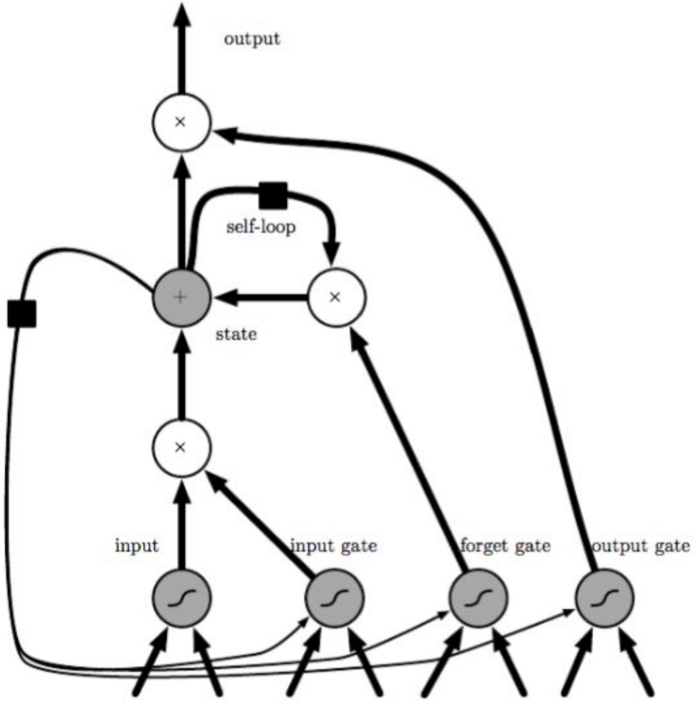


$$Y = \begin{bmatrix}0\\1\\1\\0\end{bmatrix}$$

$$X = \begin{bmatrix}0 & 0\\0 & 1\\1 & 0\\1 & 1\end{bmatrix}$$

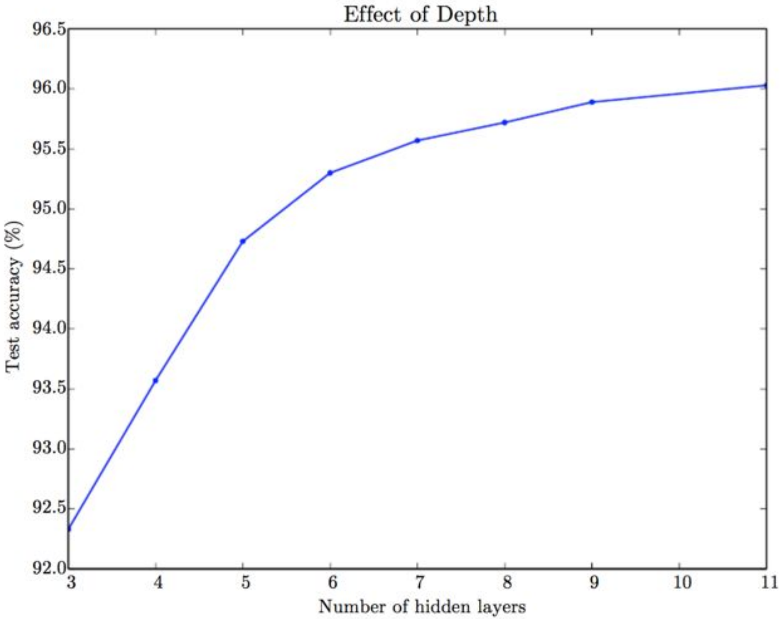
Example Deep Neural Networks





Accuracy Improvement with Deeper Networks

Example: Transcribe multi-digit numbers from photographs of addresses



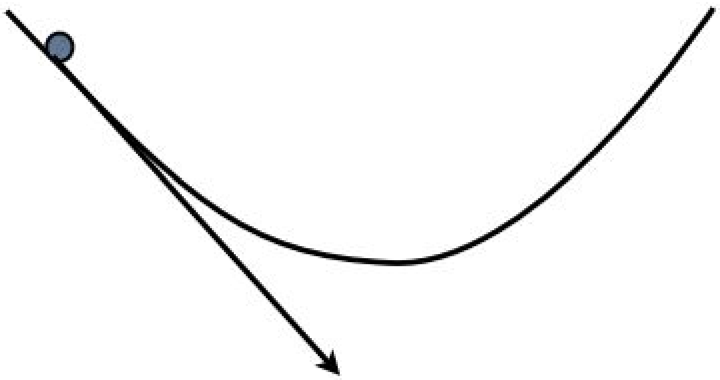
How Can We Train a Neural Network?

while not done:

    pick a random training case **`(x, y)`**
    run a neural network on input **x**
    modify connection weights to make prediction closer to **y**

How to Modify Connections?

- Follow the gradient of the error w.r.t. the connection (e.g., weight parameters)



Gradient points in direction of improvement

Compute Gradient with “Back-Propagation”

- Use the chain rule of calculus:

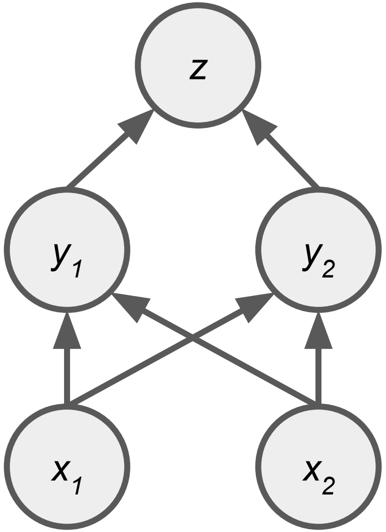
error rate

hidden node’s parameter

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}.$$

learning parameter

- Backtrack a network from output to input
- Memorize intermediate results
  to avoid recalculation



Compute Gradient with “Back-Propagation”

- Use the chain rule of calculus:

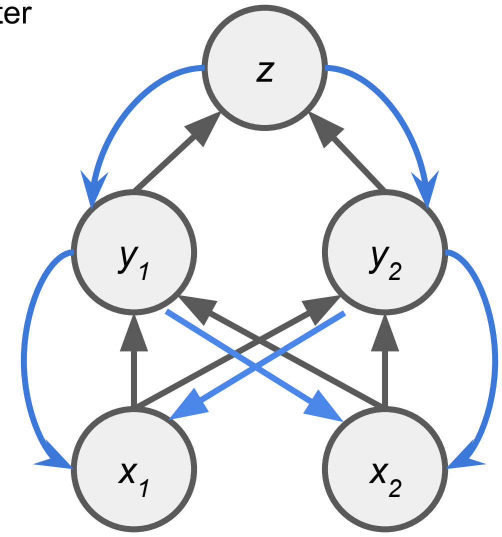
error rate

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}.$$

hidden node’s parameter

learning parameter

- Backtrack a network from output to input
- Memorize intermediate results
  to avoid recalculation



Stochastic Gradient Descent (SGD) Learning

- Estimate the gradient with **a small set of samples** (minibatch)

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while

Loss function

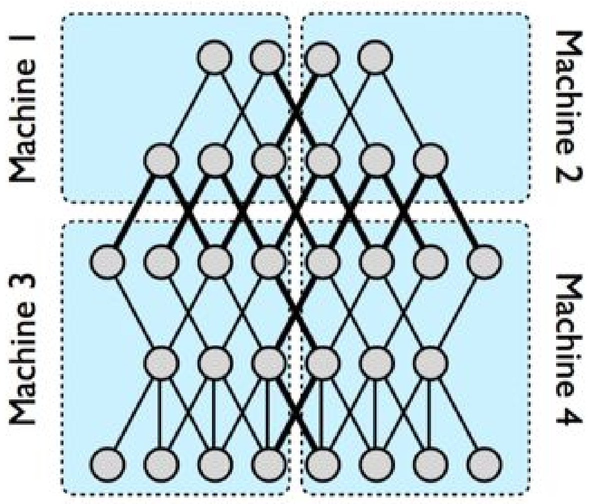
How Can We Train Large Neural Nets Quickly?

- Exploit many kinds of parallelism
  - Model parallelism
  - Data parallelism

- Note: bad fit with MapReduce
  - Mutation to learning parameters
  - Non-deterministic result
  - “Weak” correctness guarantee
    - OK if we can train a model accurately
  - ...

Model Parallelism

- Partition model across machines
  - The most densely connected areas are on the same partition
  - Up to 144 partitions with significant speedups

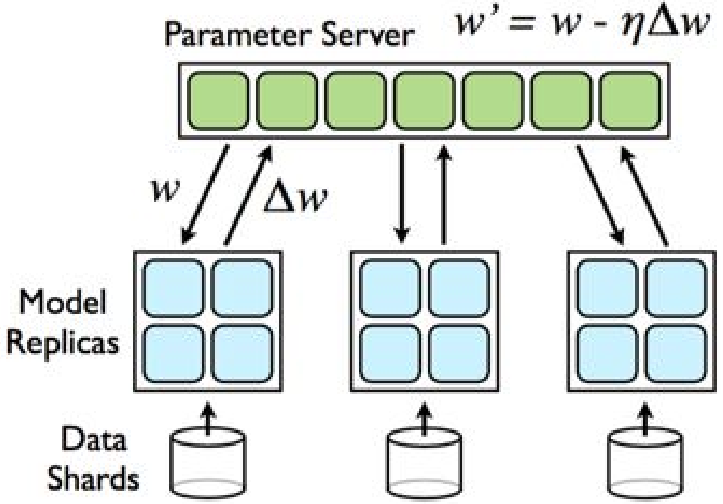


Data Parallelism

- Distribute training ***across*** multiple model instances
- Propose two algorithms
  - Downpour SGD (= variant of asynchronous SGD)
  - Sandblaster L-GFGS

Downpour SGD

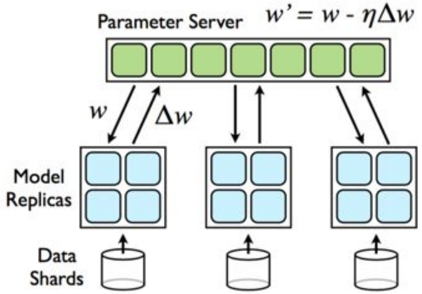
- Divide the training data into a number of subsets
- Run a copy of the model on each of these subsets
- Communicate updates through a centralized parameter server



Asynchronous Communication Between Servers

- Model replicas run independently of each other
- Parameter server shards also run independently of one another

- **Pros:** Can continue processing even when one machine is down
- **Cons:** Additional stochasticity in the optimization procedure
    - E.g.) A model replica computes its gradients based on out-of-date parameters



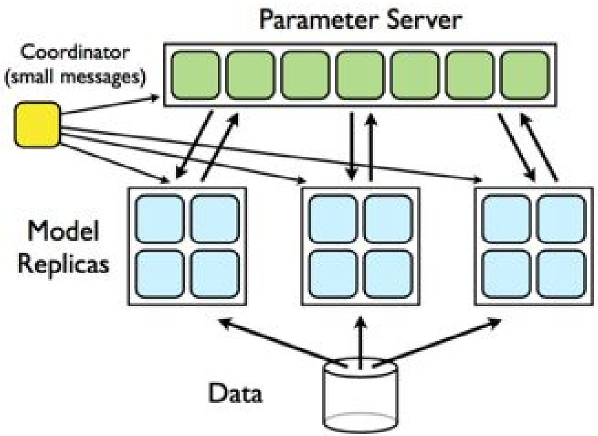
Optimizing Downpour SGD

- Fetch & pull parameters only every *N* steps
- “Warmstart” model training with only a single model replica before unleashing the other replicas
- Apply a non-fixed learning rate
    - “Adagrad” adaptive learning rate procedure

$$\eta_{i,K} = \gamma / \sqrt{\sum_{j=1}^{K} \Delta w_{i,j}{}^2}$$

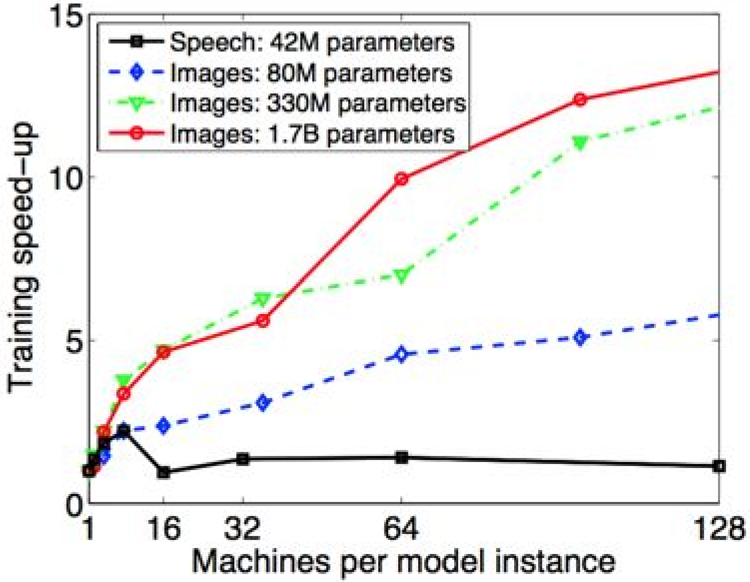
Sandblaster L-BFGS

- Avoid high-frequent, high-bandwidth communication
  - Coordinator issues “commands” to Parameter Servers
    - Parameter Servers execute commands and store results
  - Coordinator dynamically assigns tasks to Model Replicas
    - Multiple copies of work can be scheduled to address a slow bottleneck machine



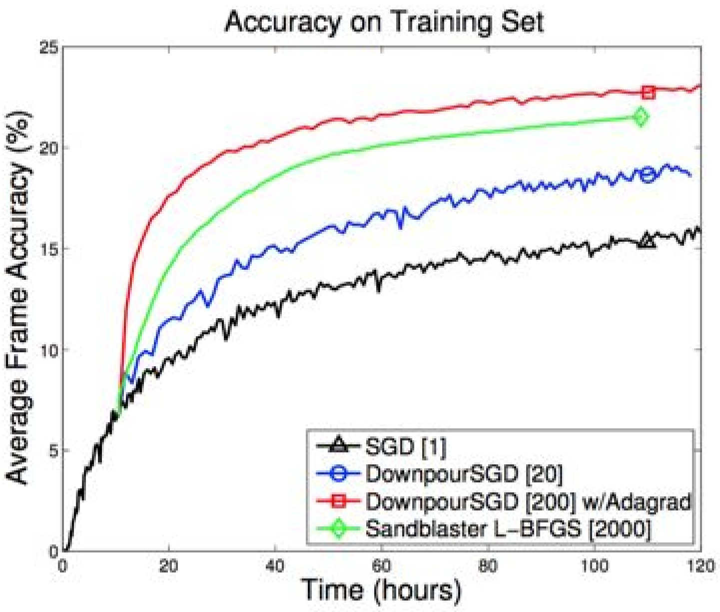
Evaluation: Model Parallelism

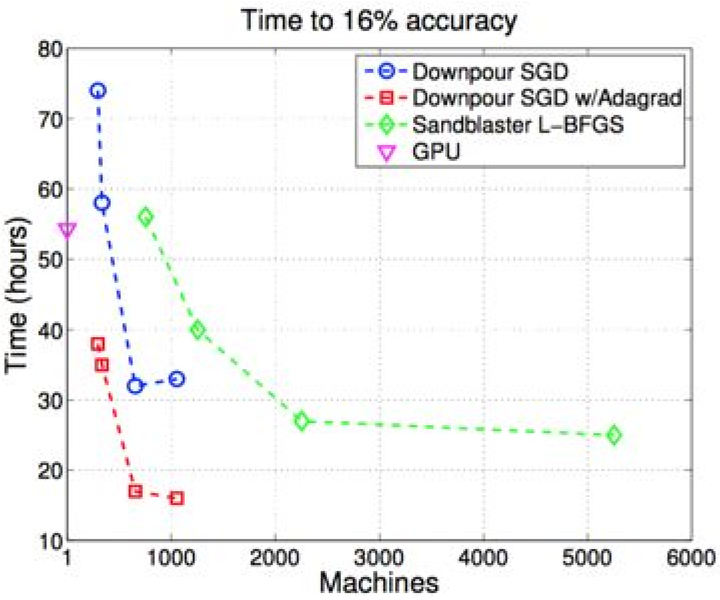
- Mean time to process a min-batch for a simple SGD
- The largest model benefits the most
  - More than 12x speedup using 81 machines



Evaluation: Data Parallelism

- Speech model in a variety of configurations





Summary of DistBelief

- Parallel and distributed execution of deep learning with SGC
  - Partition model across machines
  - Distribute training across multiple model instances

- Individual techniques are not surprising, but achieved very good results

TensorFlow

Lesson Learned From DistBelief

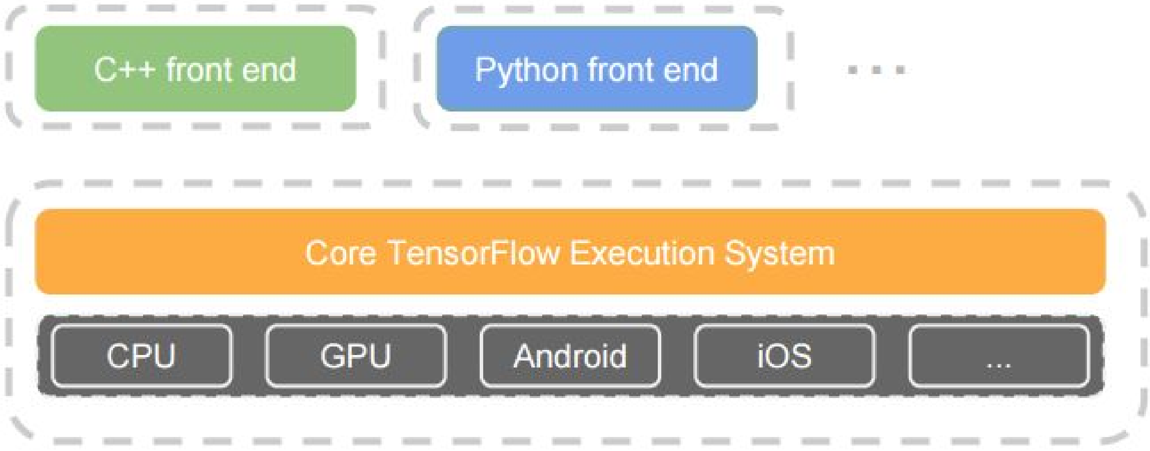
- Need a better abstraction layer
  - E.g.) Allow users to add new primitives or without changing DistBelief core
- Need one system for both large-scale training and small-scale deployment
  - E.g.) Experiment a new algorithm on a single machine first and then use the same code for large-scale deployment
- Need to support heterogeneous hardware
  - GPU, custom ASIC (e.g., TensorFlow Processing Unit)

TensorFlow™

- Second-generation system for the implementation and deployment of large-scale machine learning models
- Takes computations with a ***dataflow-like model*** and maps them onto a wide variety of different hardware platforms
    - Inference on mobile device platforms
    - Modest-sized training and inference on single machines with GPUs
    - Large-scale training on >100 specialized machines with >1000 GPUs

Expressing High-Level ML Computations

- Core in C++
  - Very low overhead
- Different front ends for specifying/driving the computation
  - Python and C++ today, easy to add more

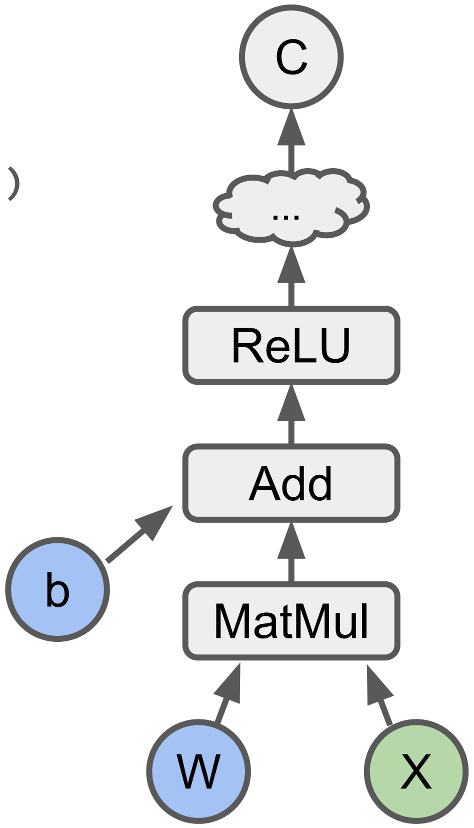


Example TensorFlow Program (in Python)

```
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
```

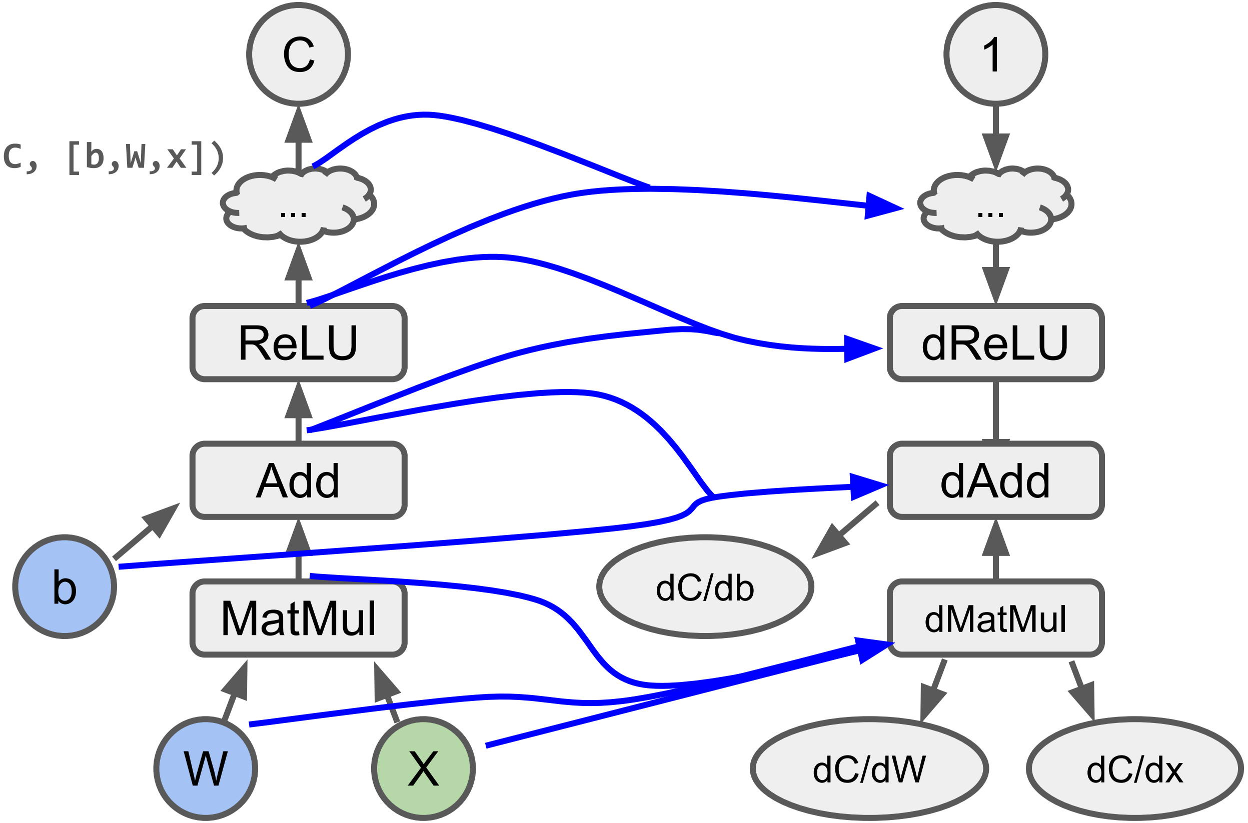
Representation of Computation Graph

```
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
```



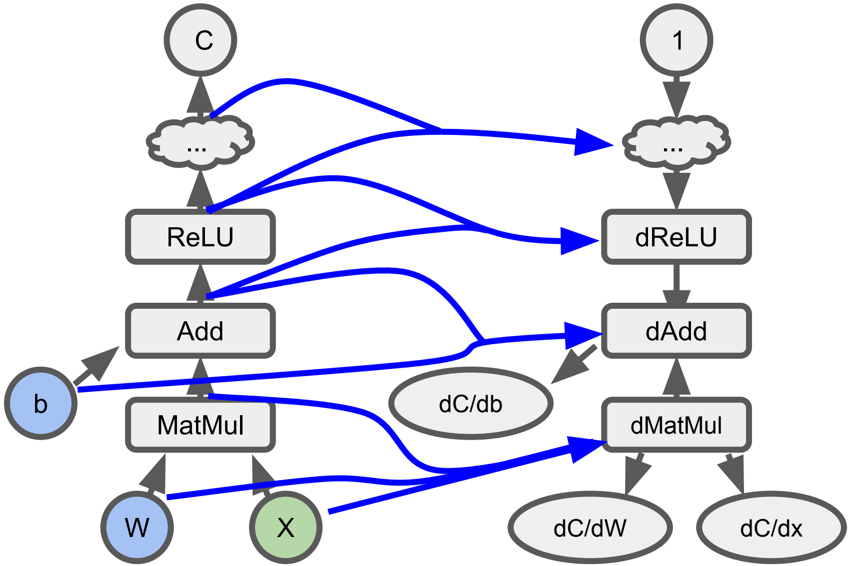
Graph for Computing Gradients

```
[db,dW,dx] = tf.gradients(C, [b,W,x])
```



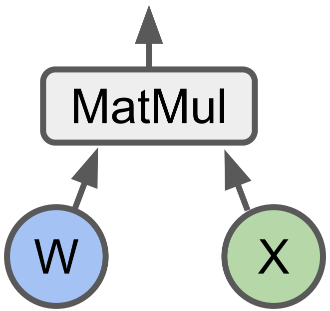
By Representing Gradient Computation as Graph..

- Allow Users to easily implement new algorithms for computing gradients
  - C.f.) Implementing the “momentum” algorithm in DistBelief required change to to the C++ parameter server and execution of arbitrary code in write operations



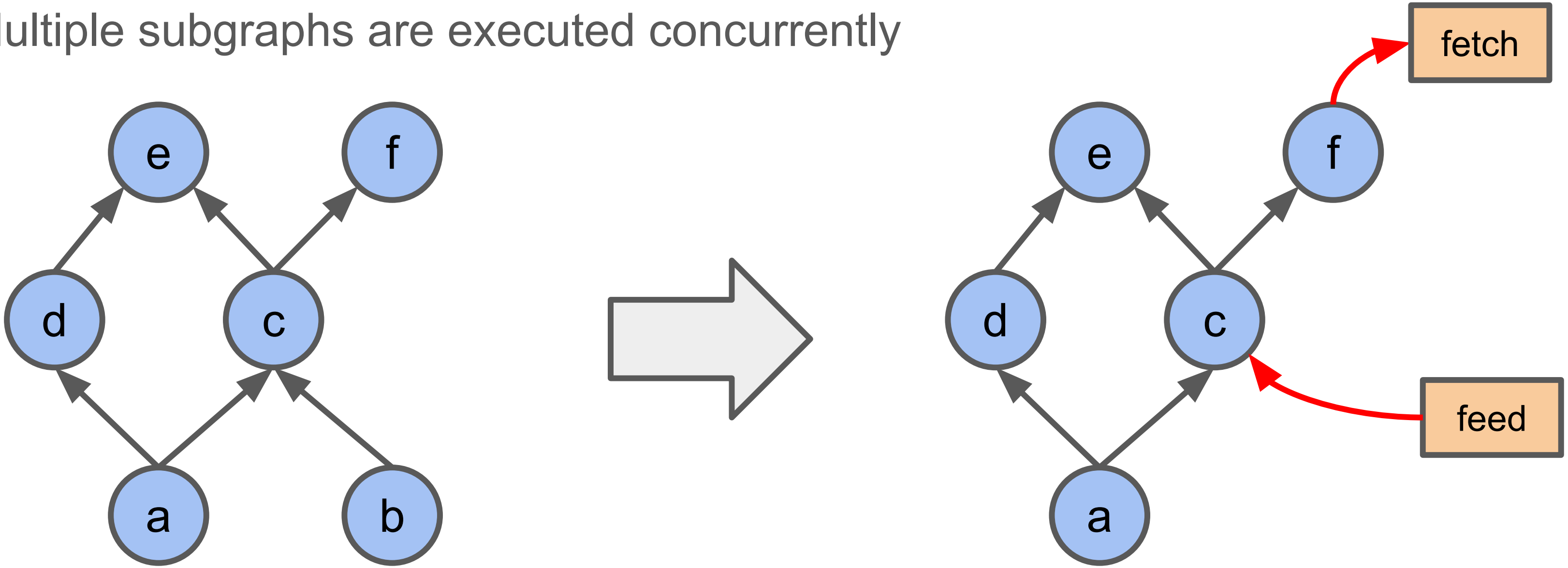
TensorFlow Execution Model

- **Vertex (= operation)**
  - An atomic unit of computation
  - May have ***mutable*** state that can be shared between different executions of the graph
  - E.g.) Add, MatMul, Variable, ReLU, Save, Merge, ...
- **Edge (= flow of tensors)**
  - Input from or output to a vertex
  - E.g.) Matrix multiplication takes two 2-D tensors and produces a 2-D tensor
  - E.g.) Mini-batch 2-D convolution takes two 4-D tensors and produces another 4-D tensor



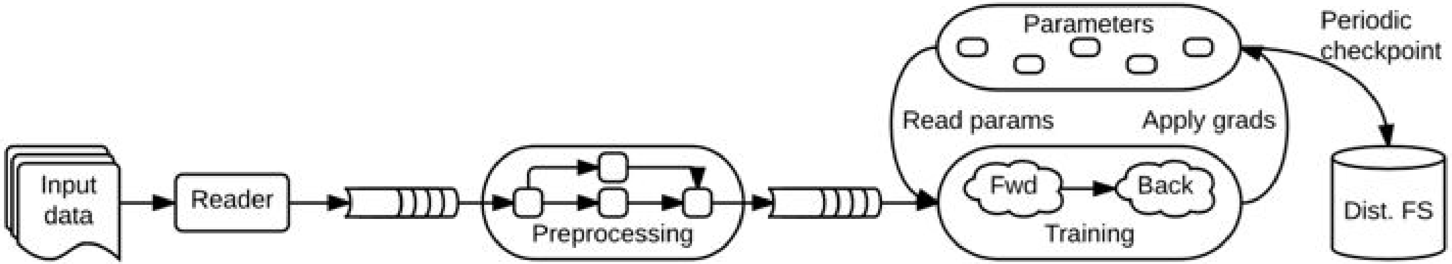
Parallel and Concurrent Execution

- Client specifies a ***subgraph*** that should be executed
  - Zero or more edges to ***feed*** input tensors
  - One or more edges to ***fetch*** output tensors
- Multiple subgraphs are executed concurrently



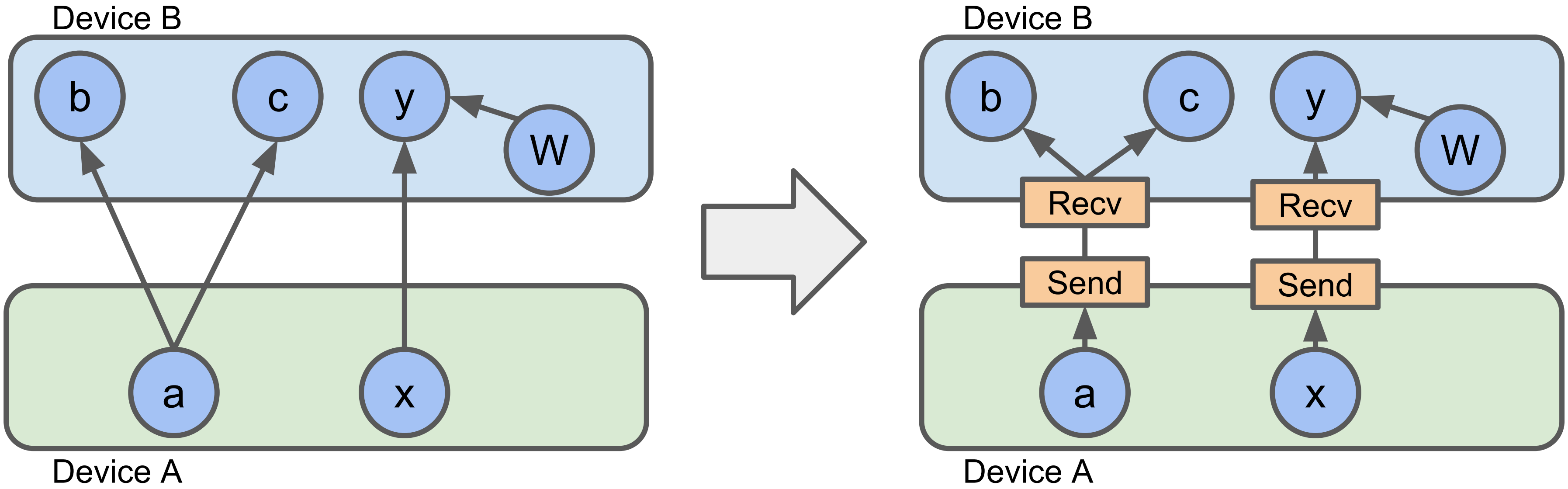
Concurrent Execution in a Training Application

- **I/O subgraph:** Read records (e.g., images) from a distributed file system
- **Preprocessing subgraph:** Transform individual input records
  - E.g.) Decode images and apply random distortions
- **Taining subgraph:** Update the model based on different input batches
  - Implement data-parallel training
  - Consist of many concurrent steps
- **Checkpointing subgraph:** Run periodically for fault tolerance



Distributed Execution

1. Place operations on *feasible* devices (e.g.., CPU, GPU)
    - E.g.) Only place this node on a device of type GPU
2. Partition operations into per-device subgraphs
    - Per-device subgraph contains all of the operations assigned to the device, with `Send` and `Recv`

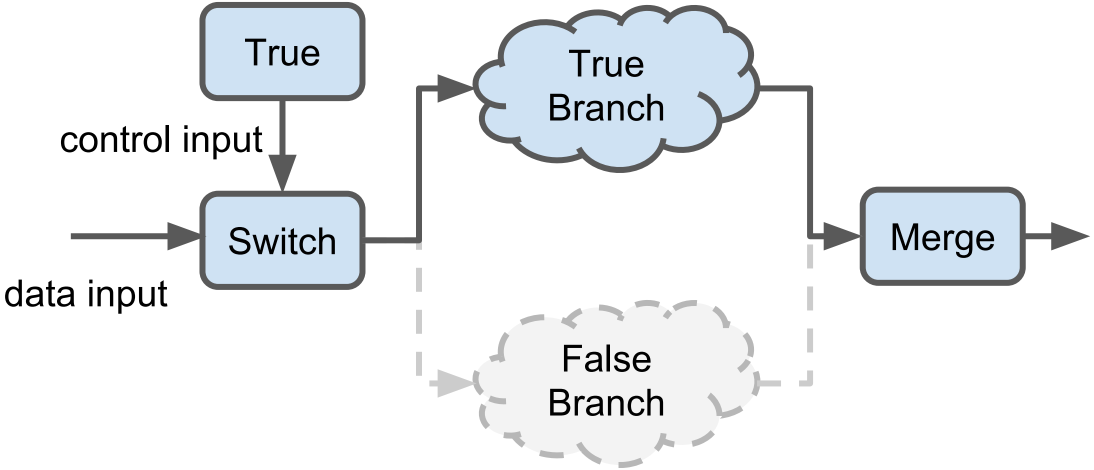


Dynamic Control Flow

- Most evaluation in TensorFlow is ***strict***
  - All inputs to an operation must be computed before the operation executes
- However, some advanced algorithms requires ***non-strict*** evaluation
  - E.g.) efficient training of a recurrent neural network

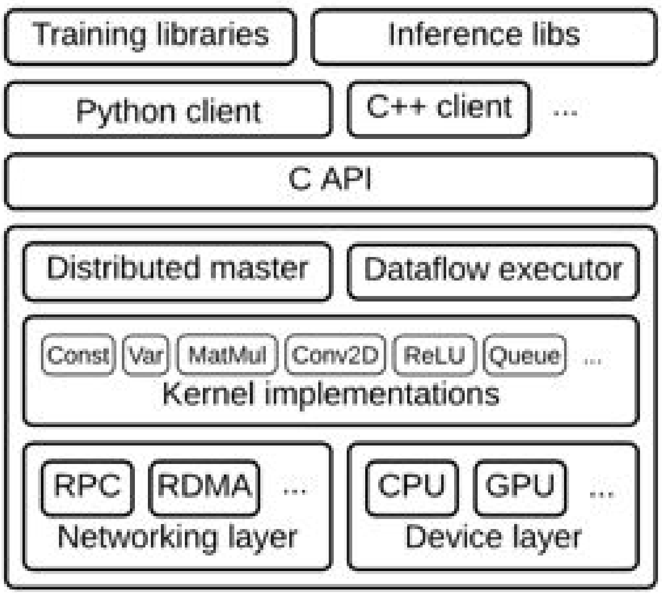
Dynamic Control Flow

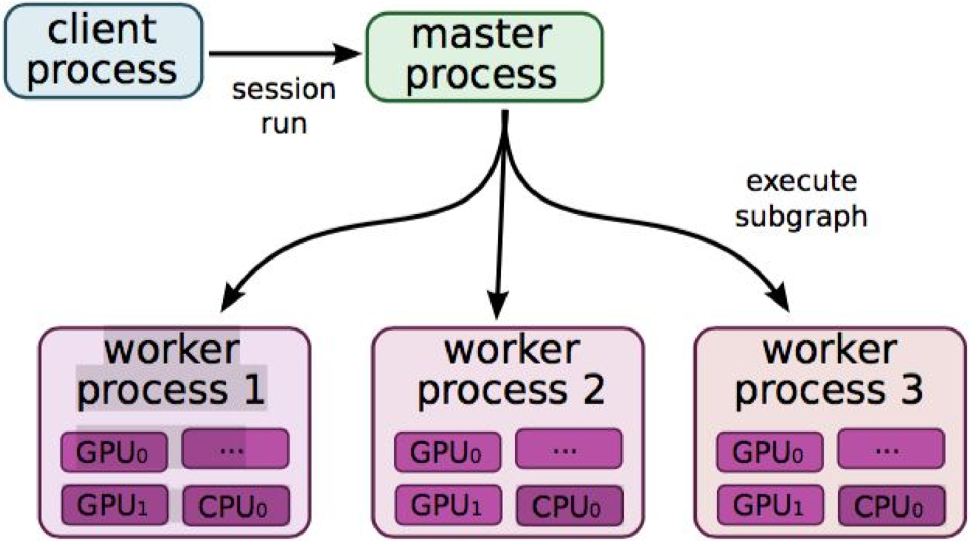
- Introduce `Switch` and `Merge`
- Can also express iteration



Implementation Sketch

- Master-worker architecture





Optimizations & Challenges

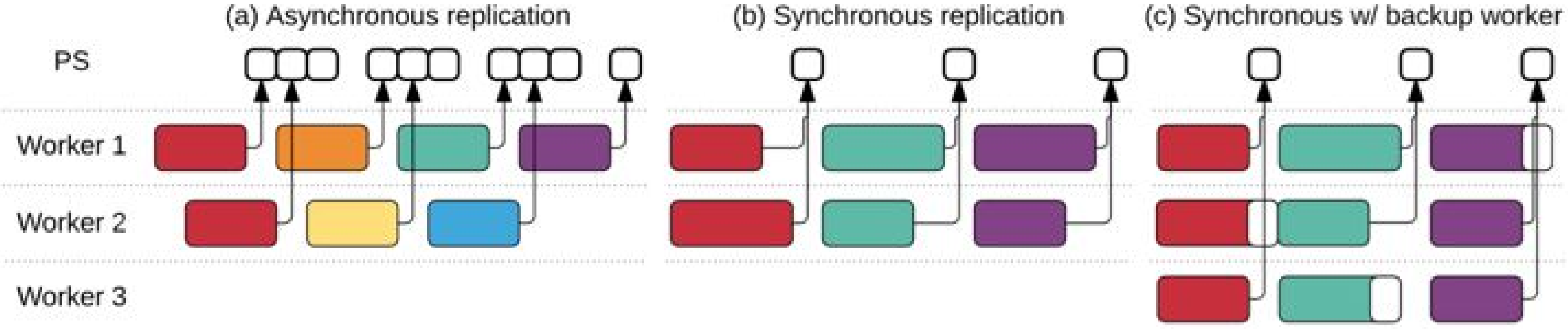
- Typical compiler optimizations
  - Common subexpression elimination
  - Fusion (e.g., replace multiple loops with a single one)
- Lossy compression
  - Convert a 32-bit float to a 16-bit float when sending data between devices
  - Not IEEE 16-bit floating point standard, but with 16 bits less precision in the significand
- Synchronous replica coordination
- Node placement and scheduling
- ....

$$1.2345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10}_{\text{base}}^{\overbrace{-4}^{\text{exponent}}}$$

Synchronous Replica Coordinations

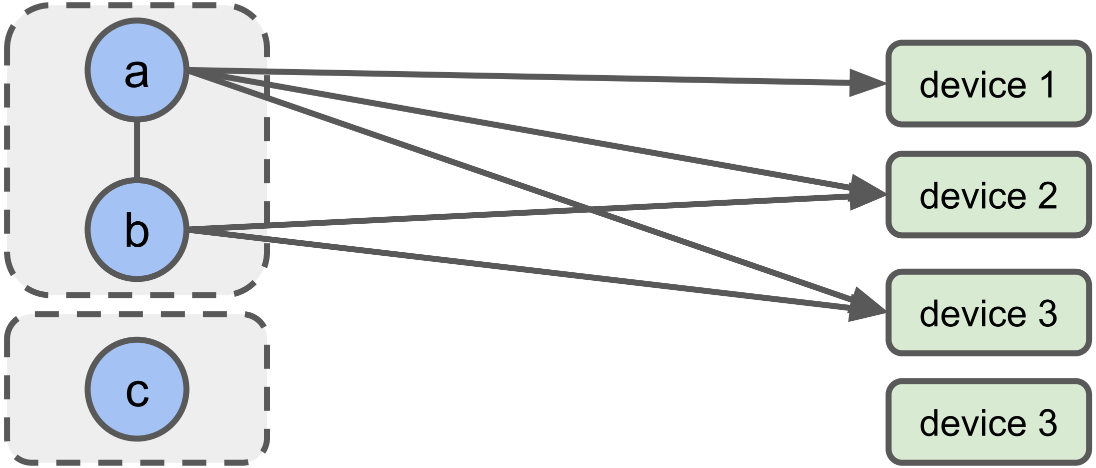
- Is ***synchronous*** training really a bad idea?
  - GPUs enable training with hundreds (not thousands) of machines

- Synchronous replication with ***backup workers***
  - Improve throughput by up to 15% by addressing stragglers



Basic Node Placement Algorithm

1. Compute the feasible set of devices for each node
2. Compute the graph components that must be placed together
    - Use union-find on the graph of colocation constraints
3. Compute the intersection of the feasible device sets for each component
4. Run the placement simulator



Greedy Heuristic Placement Simulator

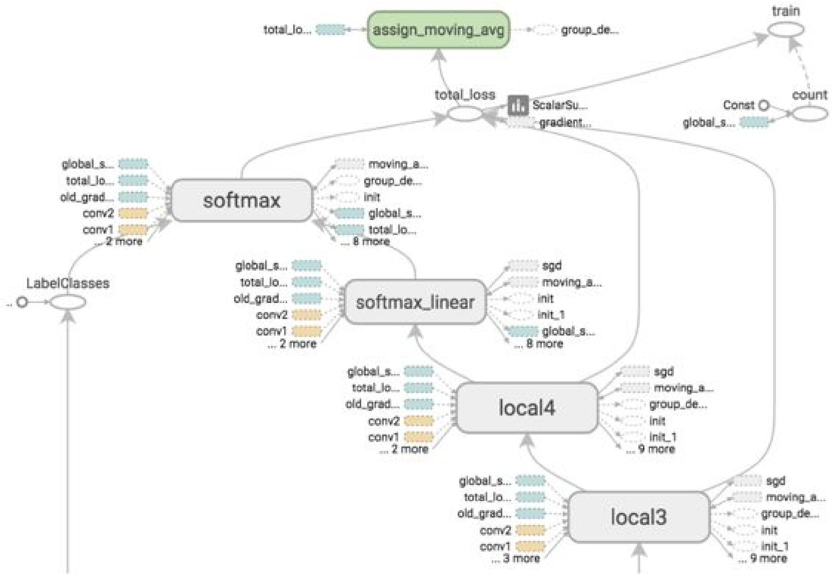
- Examines the completion time of a node on each possible device
  - Estimated (or measured) execution time of the operation
  - Communication cost for transmitting inputs to the node
- Selects the device where the node’s operation would finish soonest

Optimizations of Node Placement and Scheduling

- Scheduling of `RECV` nodes for reading remote values
  - Estimate when to start the `RECV` nodes by analyzing the critical paths
  - Perform as-soon-as-possible/as-late-as-possible (ASAP/ALAP) calculation
- Memory management for “back-propagation” gradient calculation
  - Manage GPU memory when iterating over long sequences in the input data
  - E.g.) keeping intermediate data v.s. recomputing
- …
- ...

Note on Engineering Efforts

- Great engineering accomplishment!
  - Complex mathematical operations
  - Heterogeneous environments
  - Stochastic behavior
  - Parallel and distributed execution
  - ...
- Various supporting tools developed
  - Graph visualization
  - Performance visualization



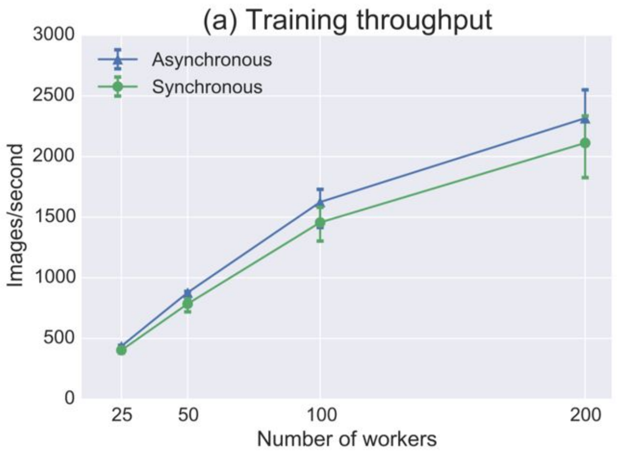
Evaluation: Single Machine Benchmarks

- Comparable performance
  - TensorFlow and Torch use the same version of the cuDNN library
  - The Neon library uses hand-optimized convolutional kernels

Library	Training step time (ms)			
	AlexNet	Overfeat	OxfordNet	GoogleNet
Caffe [36]	324	823	1068	1935
Neon [56]	87	**211**	**320**	**270**
Torch [17]	**81**	268	529	470
TensorFlow	**81**	279	540	445

Evaluation: Image Classification

- Investigate the scalability of training Google’s Inception v-3 model
  - 17 Parameter Server tasks and a varying number of worker tasks
  - Each Parameter Server task has 8 IvyBridge cores
  - Each worker task has 5 IvyBridge and one NVIDIA K40 GPU
- Throughput improves to 2,300 images per second (with 200 workers)



Summary

- Two large-scale machine learning systems
  - DistBelief
  - TensorFlow

- TensorFlow is available at https://www.tensorflow.org/
  - Used by various researchers and companies (e.g., UBER, Snapchat, ARM, Airbus)

References

- http://research.google.com/pubs/jeff.html
  - TensorFlow: A system for large-scale machine learning (2016)
  - TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems (2015)
  - Large Scale Distributed Deep Networks (2012)
  - …
- http://www.deeplearningbook.org/