TensorFlow: A system for large-scale machine Learning

Wen-Jen Hsieh, Eric Hsin, Kevin Chen

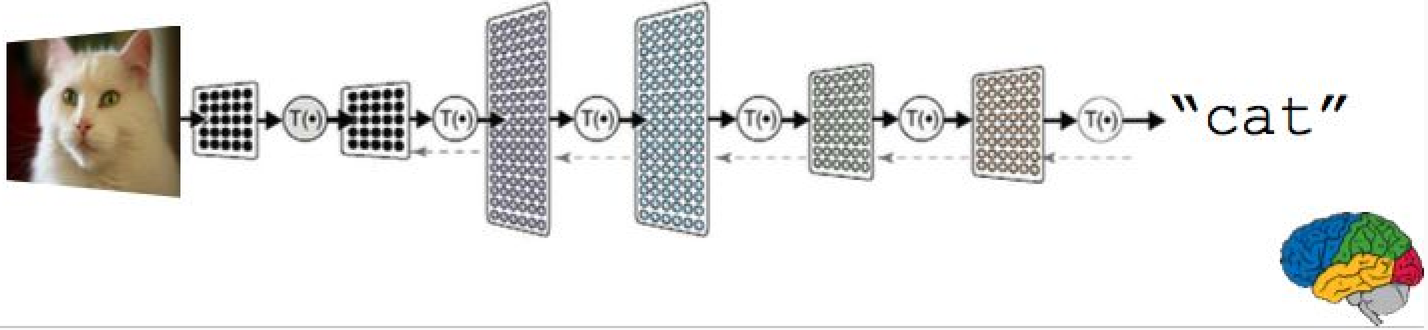
10/30/2017

Background

- A high-level overview of Deep Learning
- Deep Learning Frameworks
- Previous system: DistBelief
- Related work
- Design principles

What is Deep Learning

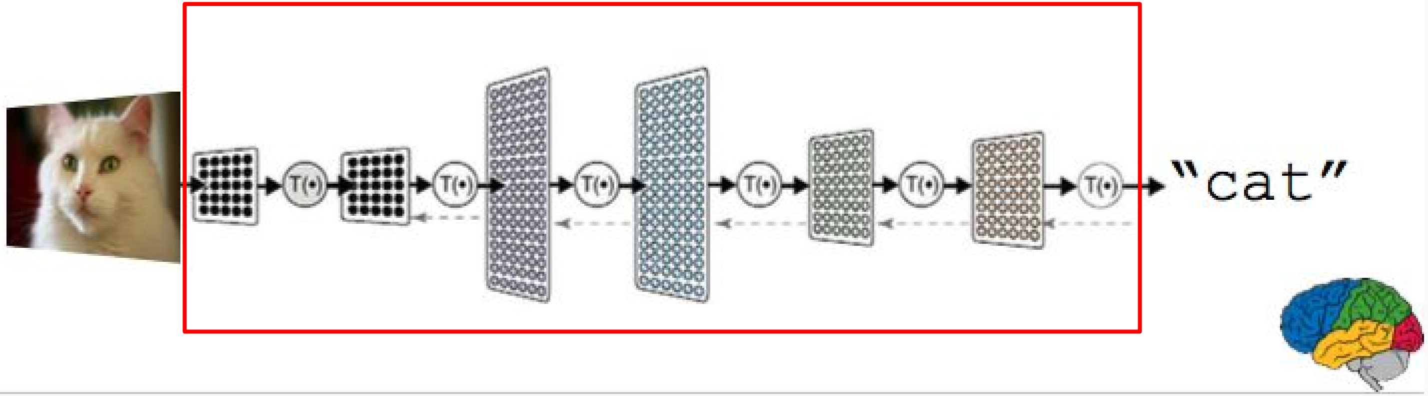
- A powerful class of machine learning model
- Modern reincarnation of artificial neural networks
- Collection of simple, trainable mathematical functions



Adapted from Jeff Dean "Large-Scale Deep Learning with Tensorflow"

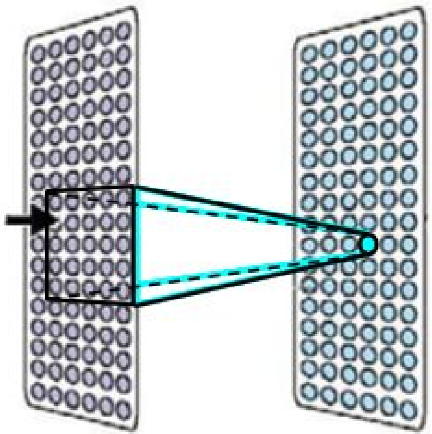
What is Deep Learning

- A powerful class of machine learning model
- Modern reincarnation of artificial neural networks
- Collection of simple, trainable mathematical functions



Adapted from Jeff Dean “Large-Scale Deep Learning with Tensorflow”

What is Deep Learning

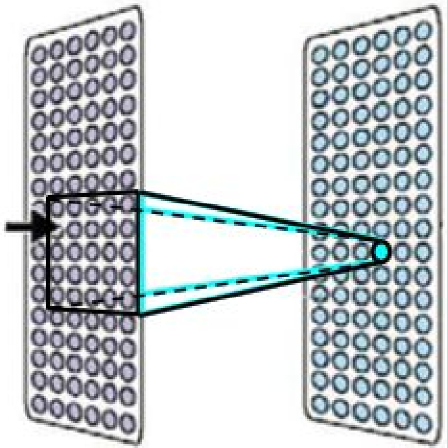


Commonalities with real brains:

- Each neuron is connected to a small subset of other neurons.
- Based on what it sees, it decides what it wants to say.
- Neurons learn to cooperate to accomplish the task.

Research at Google

What is Deep Learning



Each neuron implements a relatively simple mathematical function.

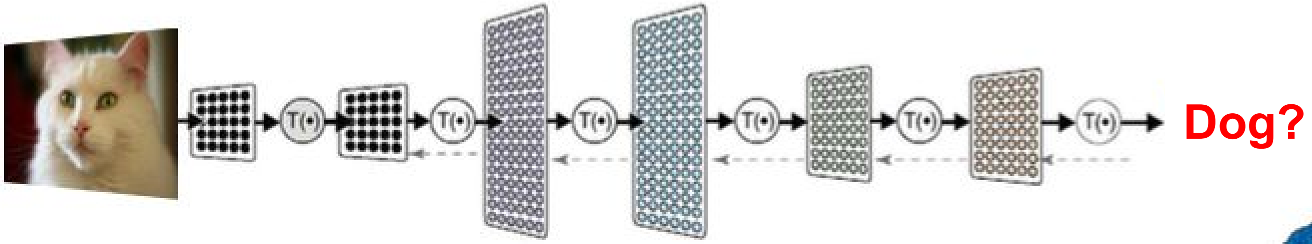
$$y = g(\vec{w} \cdot \vec{x} + b)$$

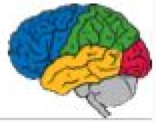
But the composition of 10^6 - 10^9 such functions is surprisingly powerful.

Research at Google

Adapted from Jeff Dean "Large-Scale Deep Learning with Tensorflow"

- A powerful class of machine learning model
- Modern reincarnation of artificial neural networks
- Collection of simple, trainable mathematical functions

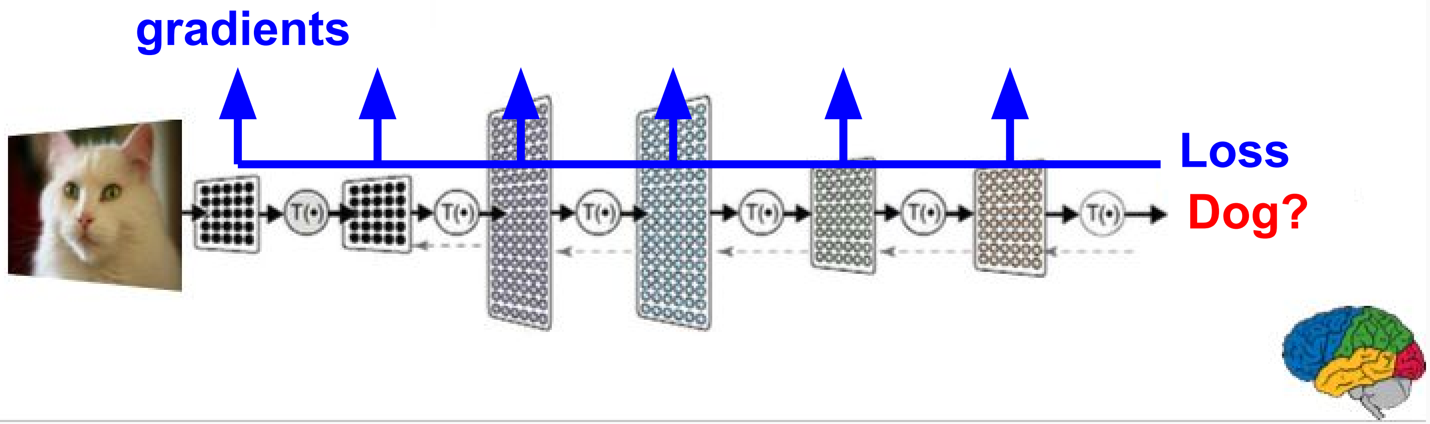




Adapted from Jeff Dean “Large-Scale Deep Learning with Tensorflow”

Back propagation

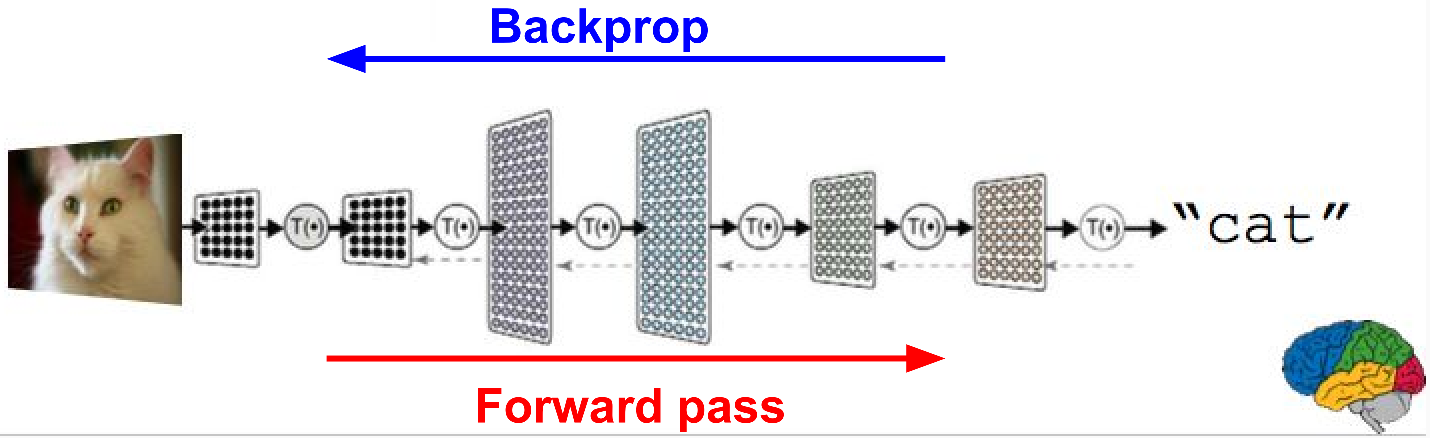
- A powerful class of machine learning model
- Modern reincarnation of artificial neural networks
- Collection of simple, trainable mathematical functions



Adapted from Jeff Dean "Large-Scale Deep Learning with Tensorflow"

What is Deep Learning

- A powerful class of machine learning model
- Modern reincarnation of artificial neural networks
- Collection of simple, trainable mathematical functions



Adapted from Jeff Dean "Large-Scale Deep Learning with Tensorflow"

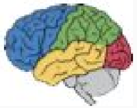
Large datasets

Important Property of Neural Networks

Results get better with

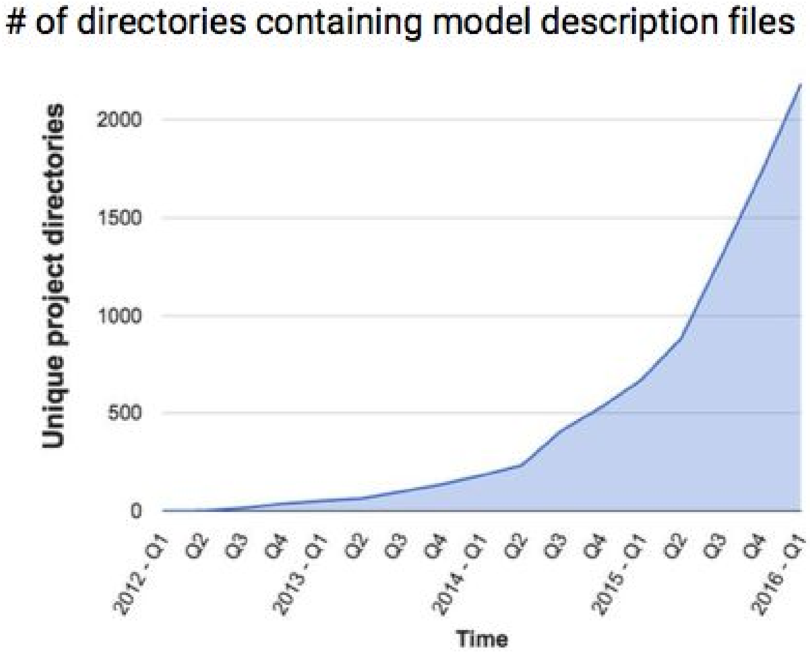
more data +
bigger models +
more computation

(Better algorithms, new insights and improved techniques always help, too!)



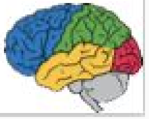
Adapted from Jeff Dean "Large-Scale Deep Learning with Tensorflow"

Growing Use of DL at Google



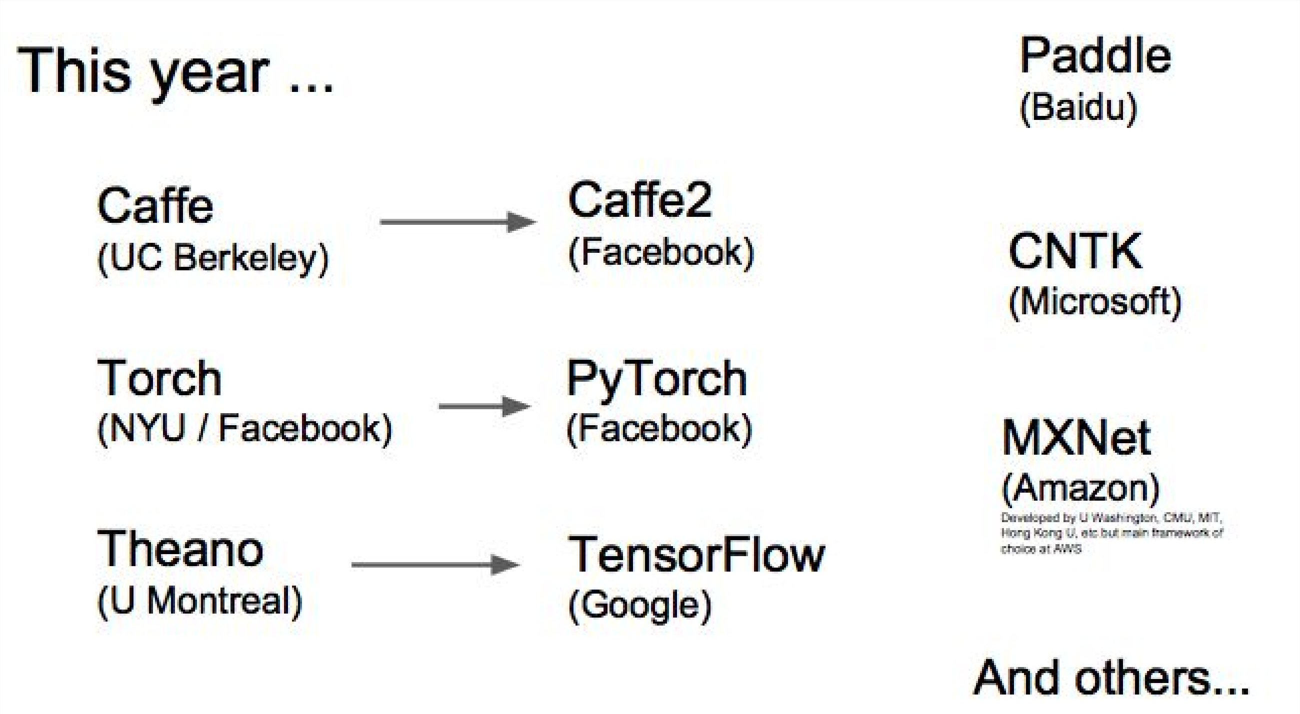
Across many products/areas:

Android
Apps
drug discovery
Gmail
Image understanding
Maps
Natural language understanding
Photos
Robotics research
Speech
Translation
YouTube
… many others …



Adapted from Jeff Dean “Large-Scale Deep Learning with Tensorflow”

DL frameworks



Adapted from Fei-Fei Li "CNN for Visual Recognition"

Comparison

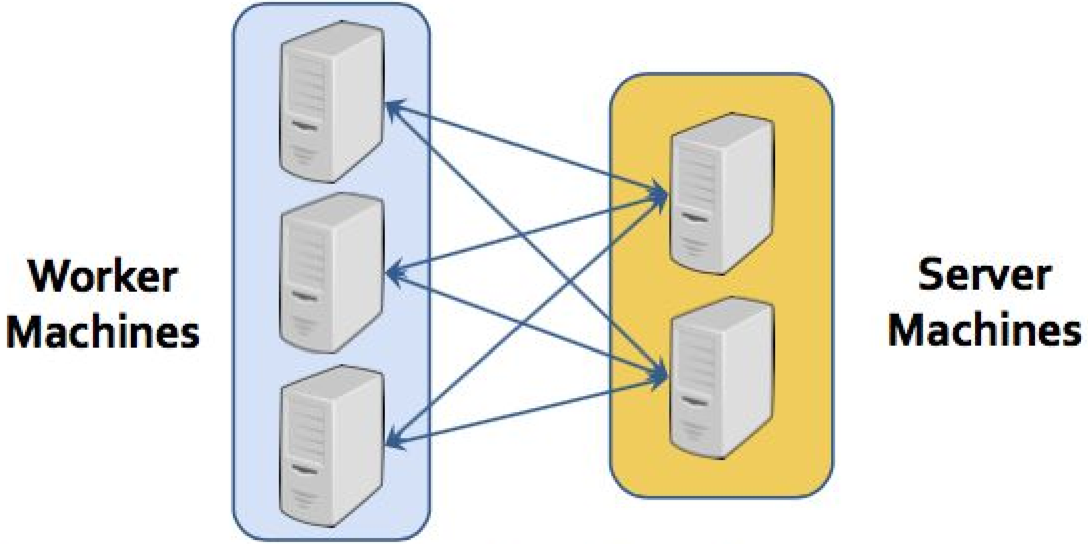
	Static Graph	Dynamic Graph	Device API	Pretrained Model	Ease of use for researchers
Tensorflow	V	V [1]	V	- (high-level API)	V
Theano	V			- (high-level API)	V
PyTorch		V	V	V	V
Caffe2	V		V (not supported in Caffe)	V	V
MXNet	V		V		

[1] Looks, Moshe, et al. "Deep learning with dynamic computation graphs." *arXiv preprint arXiv:1702.02181* (2017).

Previous system: DistBelief

- Parameter Server architecture
- DAG structure and knowledge of the layers’ semantics
- Most parameters only require weak consistency
  - Worker processes can compute updates independently
- Python-based interface
  - Simple requirements are fine

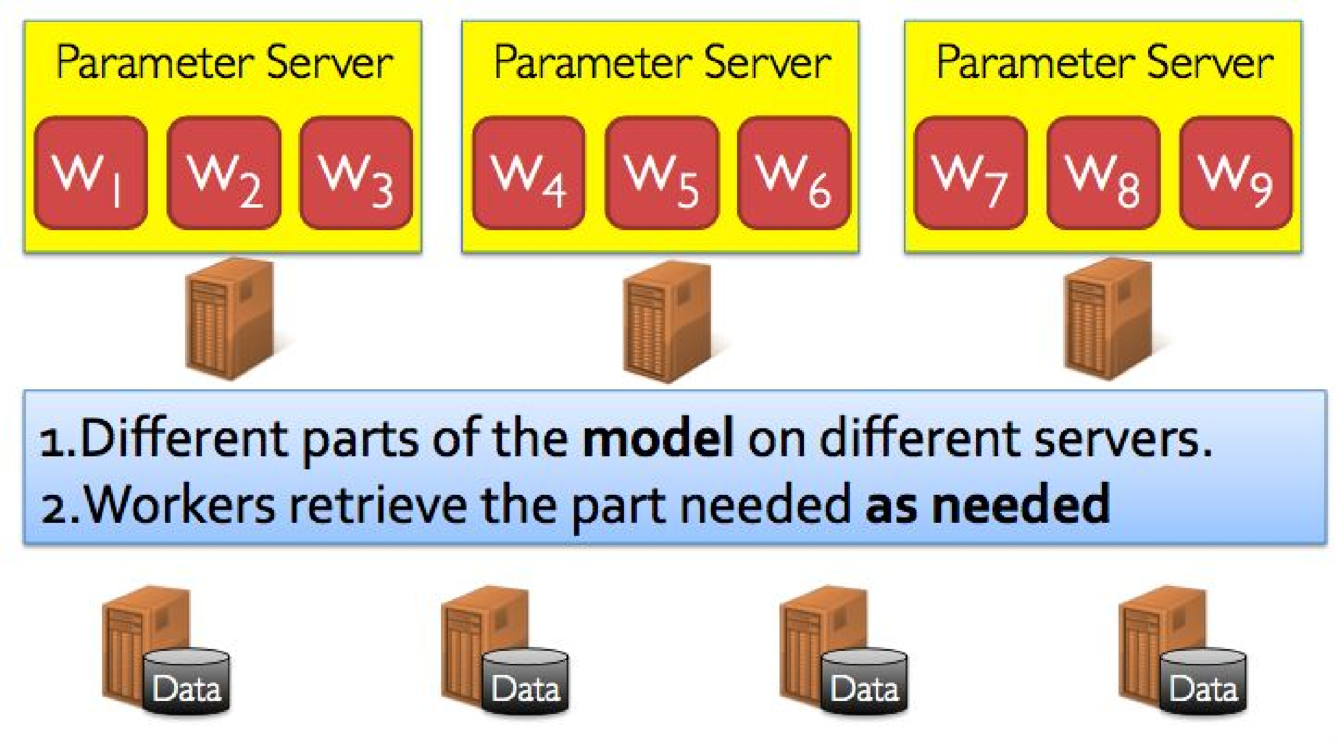
Parameter Sever (PS)



➢ Model parameters are stored on PS machines and accessed via key-value interface (distributed shared memory)

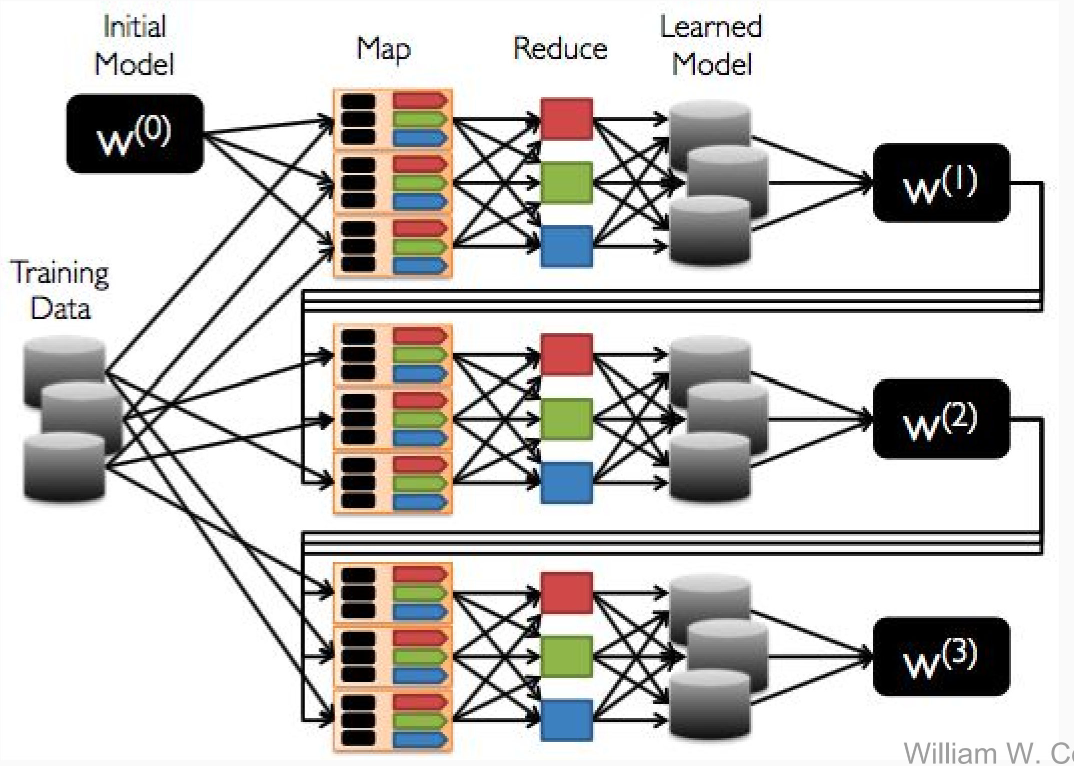
William W. Cohen “ML with Large Datasets”

Parameter Sever (PS)



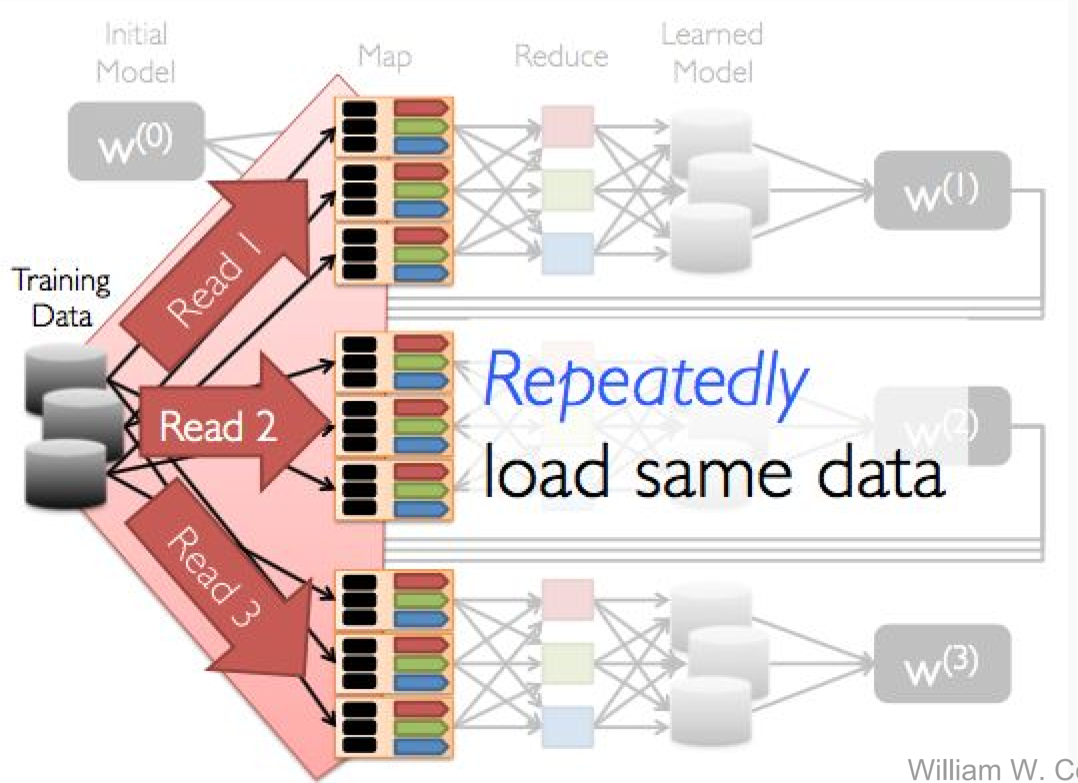
William W. Cohen “ML with Large Datasets”

PS vs Batch Processing Systems (MR)



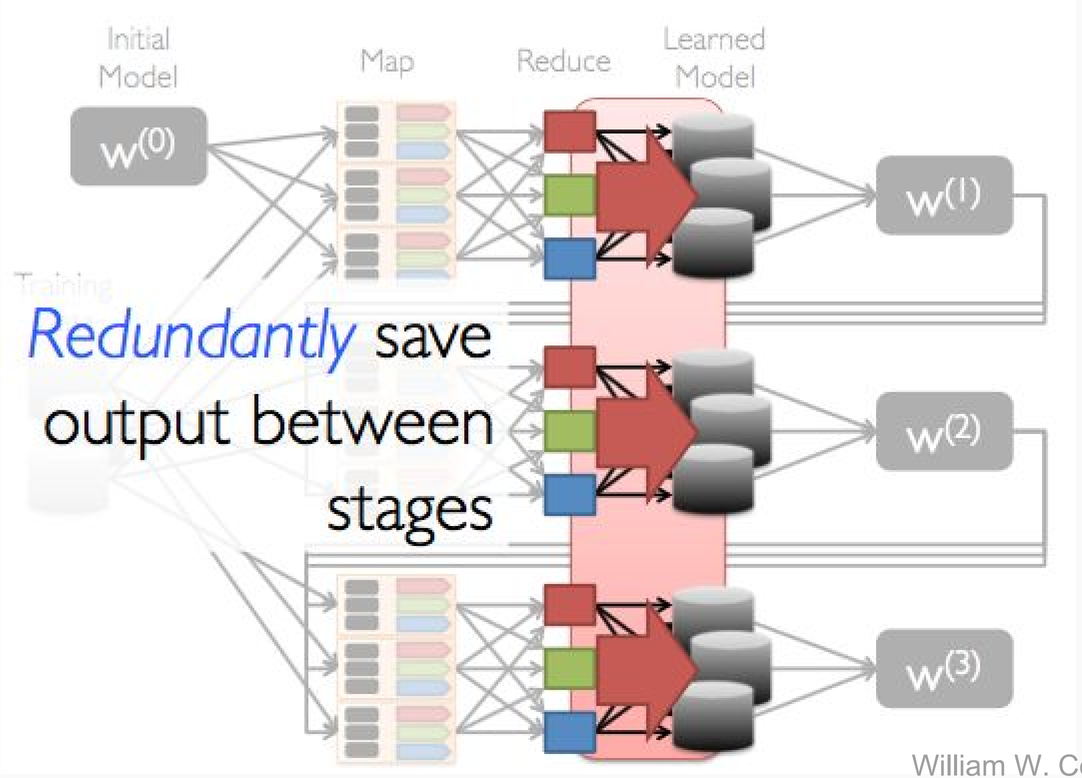
William W. Cohen "ML with Large Datasets"

PS vs Batch Dataflow Systems



William W. Cohen “ML with Large Datasets”

PS vs Batch Dataflow Systems



William W. Cohen "ML with Large Datasets"

Previous system: DistBelief

- Parameter Server architecture
- DAG structure and knowledge of the layers' semantics
- Most parameters only require weak consistency
    - Worker processes can compute updates independently
- Python-based interface
    - Simple requirements are fine

Problems of DistBelief

- Layers are C++ classes
  - Barrier for machine learning researchers
- Parameter Server
  - get() and put() interface for the PS is not ideal for all optimization methods
  - Sometimes more efficient to compute params on PS
- Workers follow a fixed execution pattern
  - RNN
  - Adversarial networks
  - Reinforcement learning
- Difficult to scale down to other environments

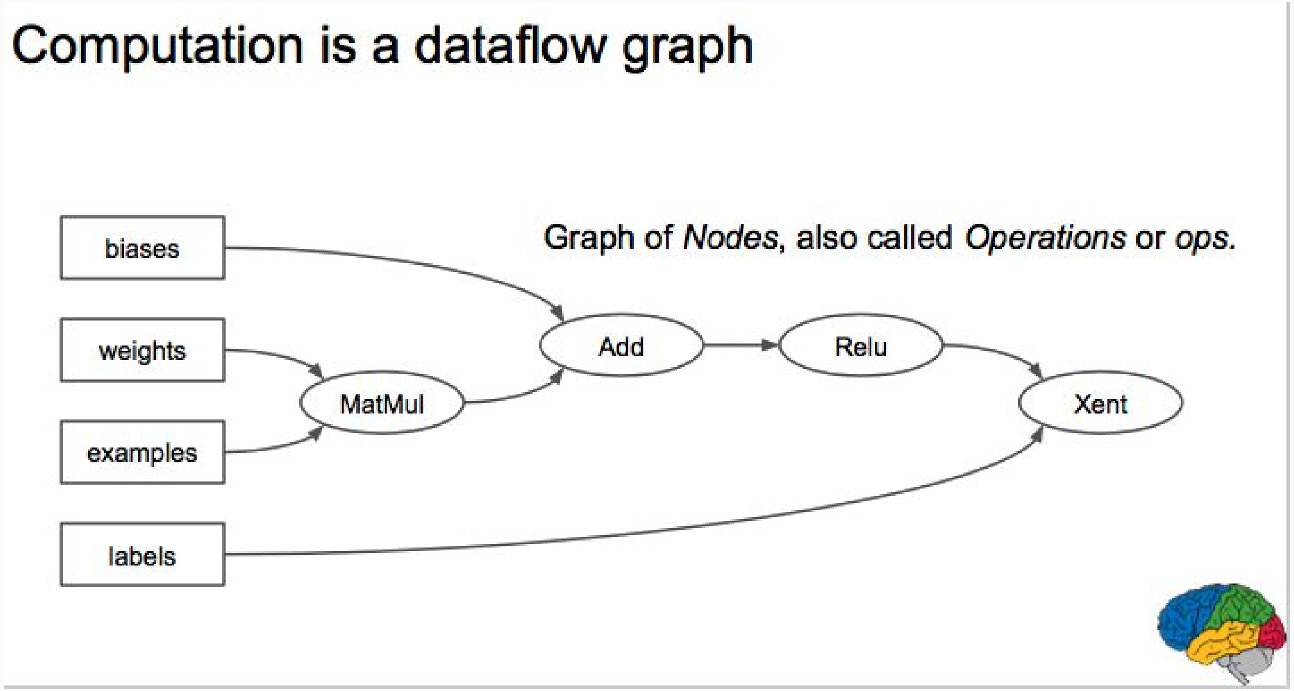
Design principles of Tensorflow

- More flexible than DistBelief, while retaining its ability

1. Individual mathematical operators in dataflow graphs
2. Deferred execution
3. Common abstraction for heterogeneous accelerators

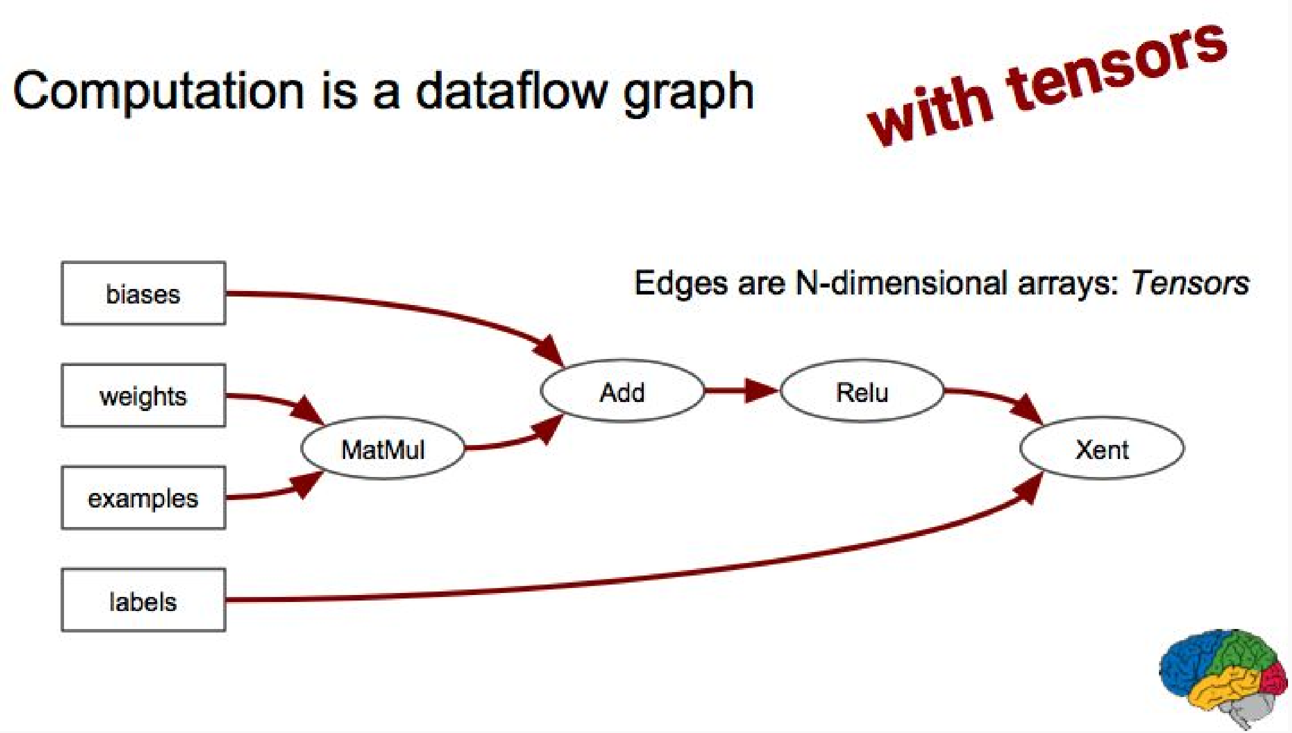
Dataflow Graphs

Simple



Adapted from Jeff Dean “Large-Scale Deep Learning with Tensorflow”

Dataflow Graphs



Adapted from Jeff Dean "Large-Scale Deep Learning with Tensorflow"

Deferred execution

Static graph

First **define** computational graph

```
# 1. Construct a graph representing the model.
x = tf.placeholder(tf.float32, [BATCH_SIZE, 784])   # Placeholder for input.
y = tf.placeholder(tf.float32, [BATCH_SIZE, 10])    # Placeholder for labels.

W_1 = tf.Variable(tf.random_uniform([784, 100]))    # 784x100 weight matrix.
b_1 = tf.Variable(tf.zeros([100]))                  # 100-element bias vector.
layer_1 = tf.nn.relu(tf.matmul(x, W_1) + b_2)       # Output of hidden layer.

W_2 = tf.Variable(tf.random_uniform([100, 10]))     # 100x10 weight matrix.
b_2 = tf.Variable(tf.zeros([10]))                   # 10-element bias vector.
layer_2 = tf.matmul(layer_1, W_2) + b_2             # Output of linear layer.

# 2. Add nodes that represent the optimization algorithm.
loss = tf.nn.softmax_cross_entropy_with_logits(layer_2, y)
train_op = tf.train.AdagradOptimizer(0.01).minimize(loss)
```

Run the graph many times

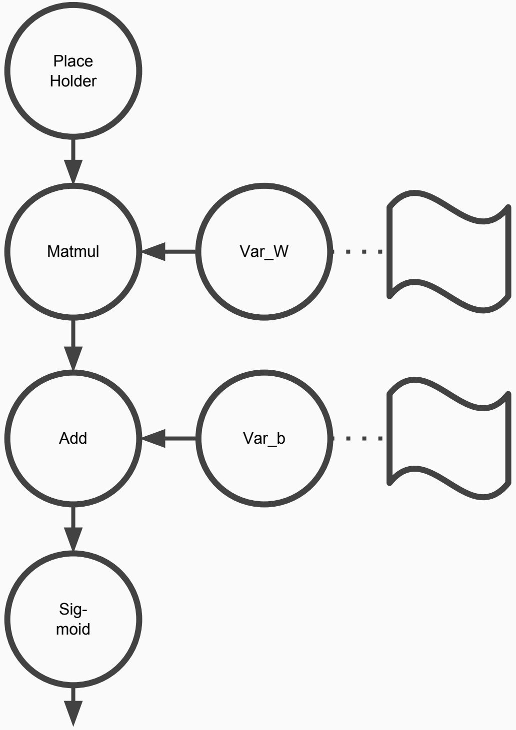
```
# 3. Execute the graph on batches of input data.
with tf.Session() as sess:                          # Connect to the TF runtime.
  sess.run(tf.initialize_all_variables())           # Randomly initialize weights.
  for step in range(NUM_STEPS):                     # Train iteratively for NUM_STEPS.
    x_data, y_data = ...                            # Load one batch of input data.
    sess.run(train_op, {x: x_data, y: y_data})      # Perform one training step.
```

Common abstraction for devices

- At a minimum, a device must implement methods for
  - Issuing a kernel for execution
  - Allocating memory for inputs and outputs
  - Transferring buffers to and from host memory
- Target CPU, GPU or TPU (Tensor Processing Unit) on same program
- Use tensors as a common interchange format

Execution Model

- Datagraph
  - Vertex: computation
  - Edges: dataflow in to / out from vertices
- Tensors
  - N-dimensional arrays of primitive types
  - Alternative sparse coding for sparse tensor
- Operations
  - Inputs: tensors; outputs: tensors
  - **Stateful Operation (w/ mutable table)**
    - **Variable**
    - **Queue (blocking)**



Execution Model (cont'd)

- Partial and Concurrent Execution
  - Multiple subgraphs can interact via stateful operations
  - Mutable tables: blocked buffer provides back-pressure; synchronize when necessary
  - Many ML algorithm allows weak consistency

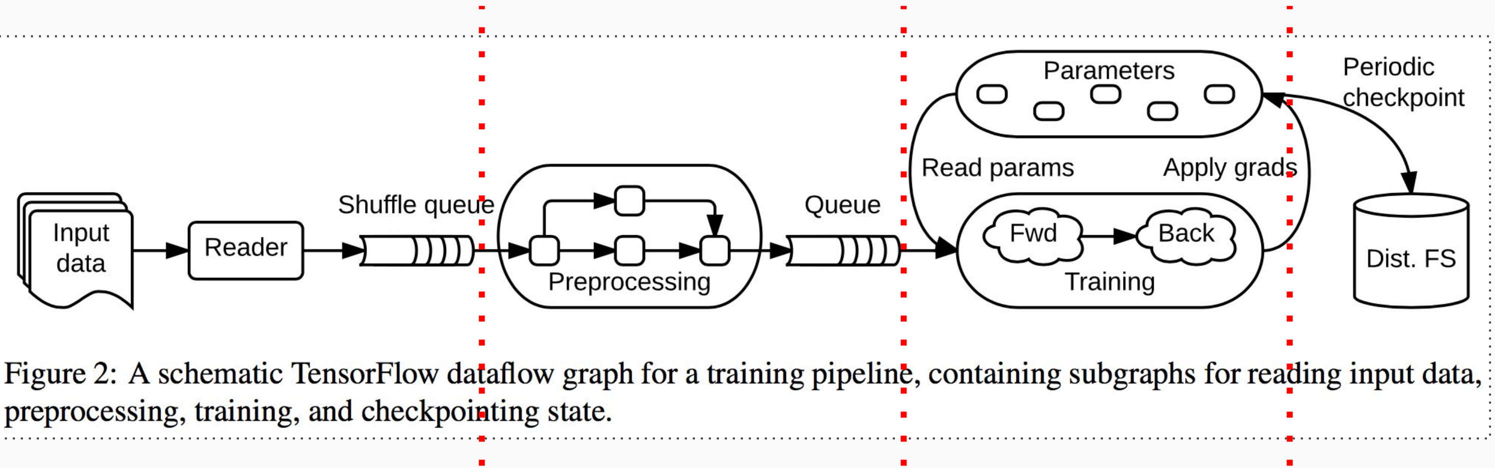


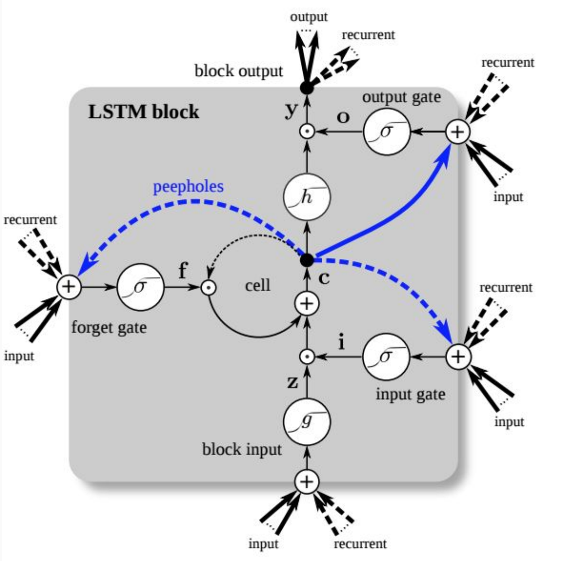
Figure 2: A schematic TensorFlow dataflow graph for a training pipeline, containing subgraphs for reading input data, preprocessing, training, and checkpointing state.

Execution Model (cont'd)

- Distributed Execution
  - Each operation of a task resides on a **Device**
  - Specified kernels to operation are implemented
  - Per-device subgraph, where a **Session** is responsible for its manipulation
  - Implicit / explicit constraints
    - **Colocation operation**
    - **Device preferences**
  - Devices communicate with **Send** and **Recv** operations

Execution Model (cont'd)

- Dynamic control flow
  - Cases such as RNN requires dynamic control
  - Conditional & Iterative programming
    - **Switch**, demultiplexer
    - **Merge**, multiplexer
    - **Dead** value for either of two cases and merges branches when Merge is met



A simple figure for LSTM (long short term memory cell)

Extensibility Case Studies

- Differentiation and Optimization
  - Given FP, update parameters via BFS (BP) automatically
  - Conditional differentiation by adding vertices
  - Easy to extend optimization algorithm such as Momentum, RMSProp, Adam, etc.

Extensibility Case Studies (cont’d)

- Very Large Model
  - Distributed representation
  - Example: **word embedding**

$$\boxed{X_{n,b}^{T} W_{n,d}} = M_{b,d}$$

Sparse, d<<n

  - Implemented **Sparse embedding** layer
    - Gather
    - Part
    - Stitch

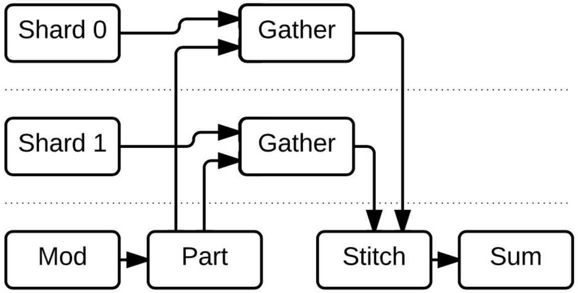


Figure 4: Schematic dataflow for an embedding layer (§4.2) with a two-way sharded embedding matrix.

Extensibility Case Studies (cont'd)

- Fault Tolerance
  - Less likely to require backup for individual operations
  - Plus, again, many ML algorithm doesn't require strong consistency
  - **Checkpoint states periodically**
    - Save (one per task) -> Restore -> Assign
    - Customization
    - Synchronization / Asynchronization

** Often used case: Transfer Learning, eg. VGG16, Resnet50 in CNN

Extensibility Case Studies (cont'd)

- Synchronous Replica Coordination
  - Async SGD allows updates with stale parameters
  - Sync SGD might be more efficient given current GPU utilization and corresponding scale
  - Blocking queue is used to synchronize
  - **Backup Workers** in replacement of stragglers

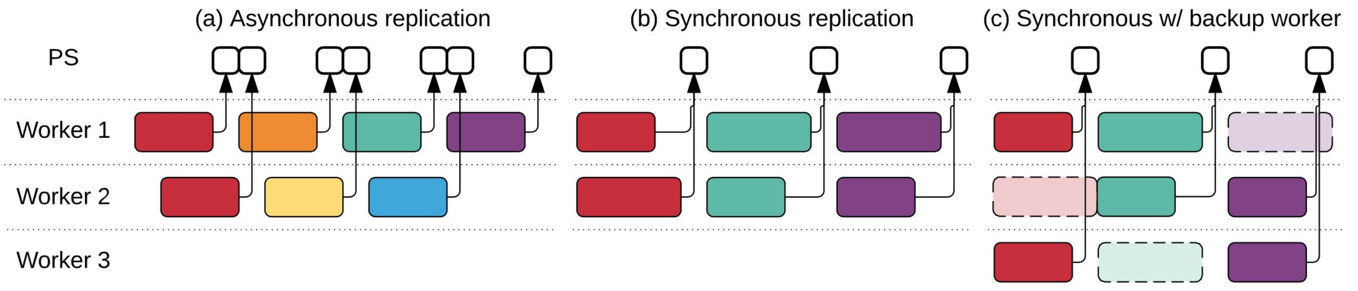


Figure 5: Three synchronization schemes for parallel SGD. Each color represents a different starting parameter value; a white square is a parameter update. In (c), a dashed rectangle represents a backup worker whose result is discarded.

Implementation

- C++
- OS: Works for Linux, Mac OSX, Windows, Android
- GPU: NVIDIA's KEPLER, MAXWELL ,PASCAL

Component

- Distributed master
  - Pruning and partitioning
  - Compiler level optimization
  - Cache and reuse
- Dataflow executor
  - Handles requests from the master
  - Dispatch kernels to local devices

Other optimization

- cuDNN
- Quantization
- gRPC over TCP
- RDMA over converged ethernet.
- Fused kernel: ReLU

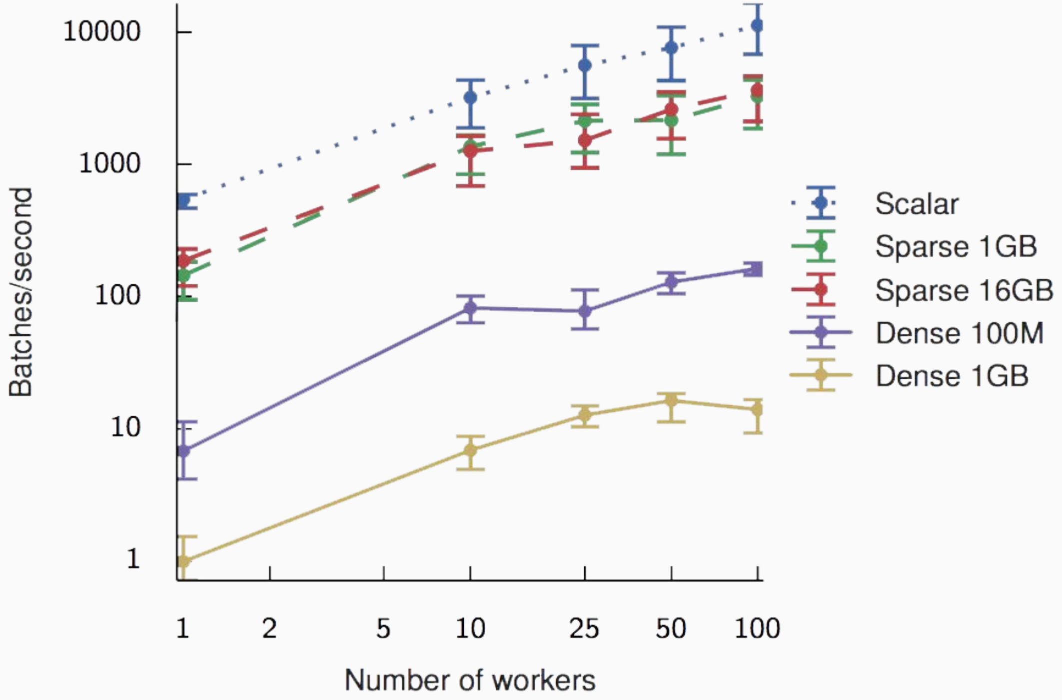
Evaluation - single machine performance

Library	Training step time (ms)			
	AlexNet	Overfeat	OxfordNet	GoogleNet
Caffe [38]	324	823	1068	1935
Neon [58]	87	**211**	**320**	**270**
Torch [17]	**81**	268	529	470
TensorFlow	**81**	279	540	445

Evaluation - single machine performance

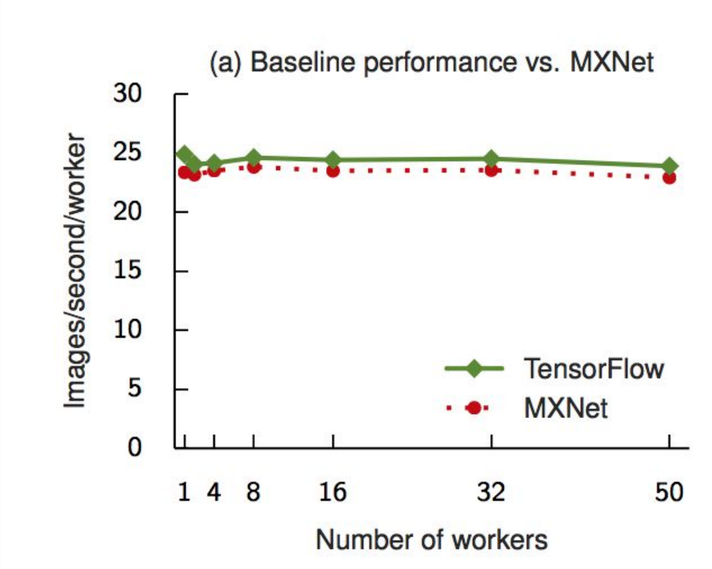
Library	Training step time (ms)			
	AlexNet	Overfeat	OxfordNet	GoogleNet
Caffe [38]	324	823	1068	1935
Neon [58]	87	**211**	**320**	**270**
Torch [17]	**81**	268	529	470
TensorFlow	**81**	279	540	445

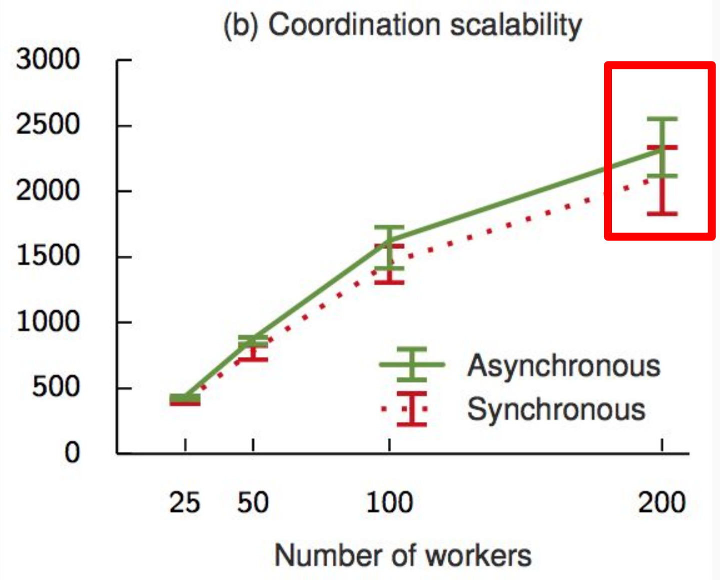
Evaluation - Replica



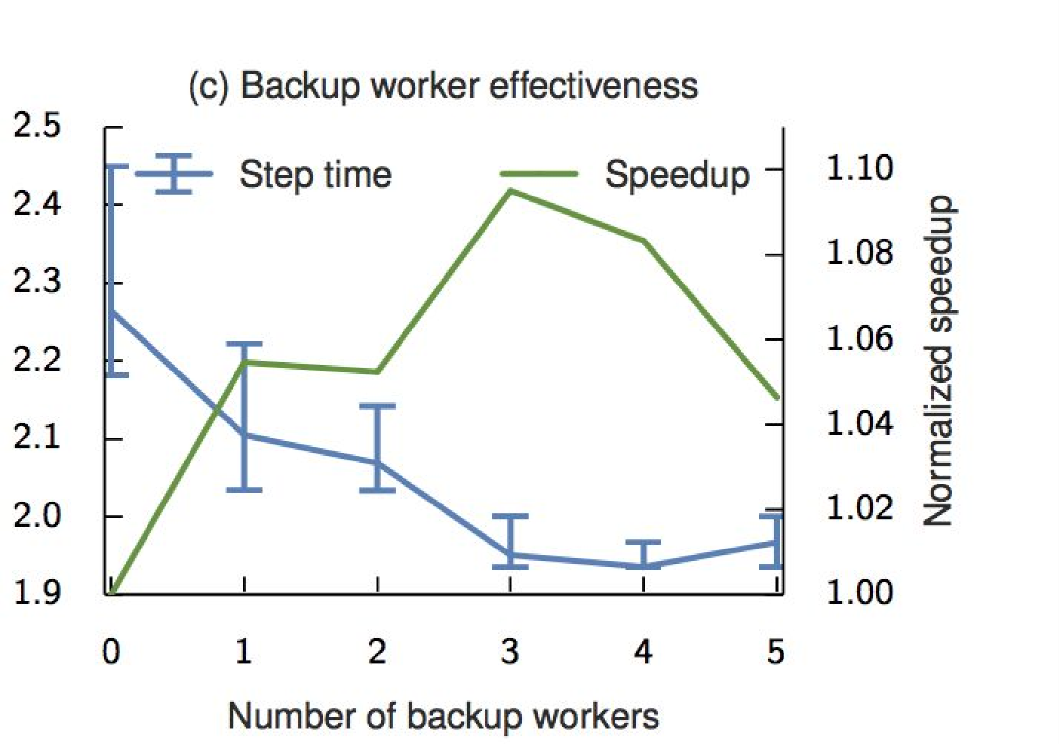
Evaluation - image classification

- MXNet v.s. Tensorflow
- Inception-v3 model

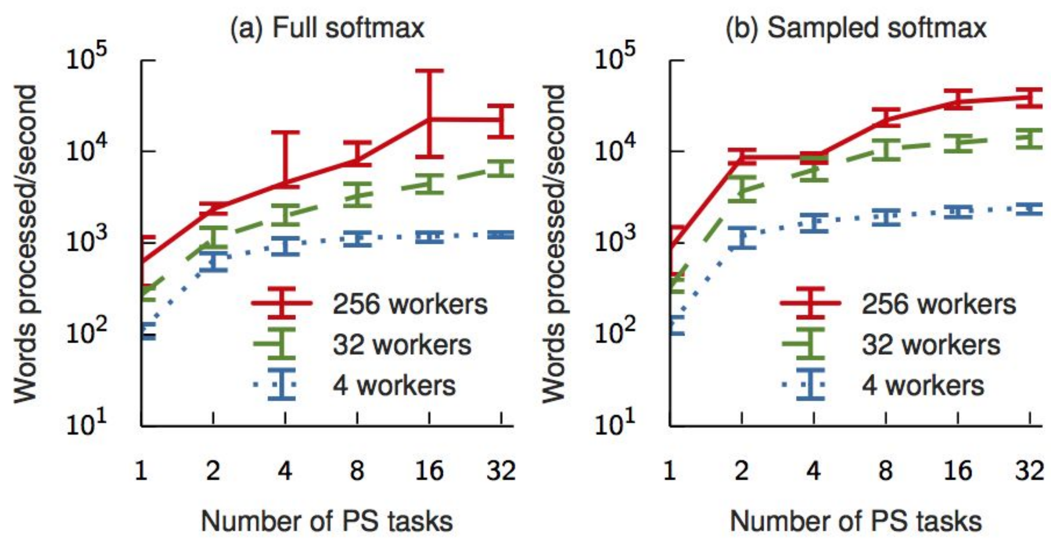




Evaluation - image classification



Evaluation - Language model



Conclusion

- Incorporate parameter server
- Scalable
- Heterogeneous system
- Future work