Scaling Distributed Machine Learning with the Parameter Server

Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, Bor-Yiing Su

OSDI'14: Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation

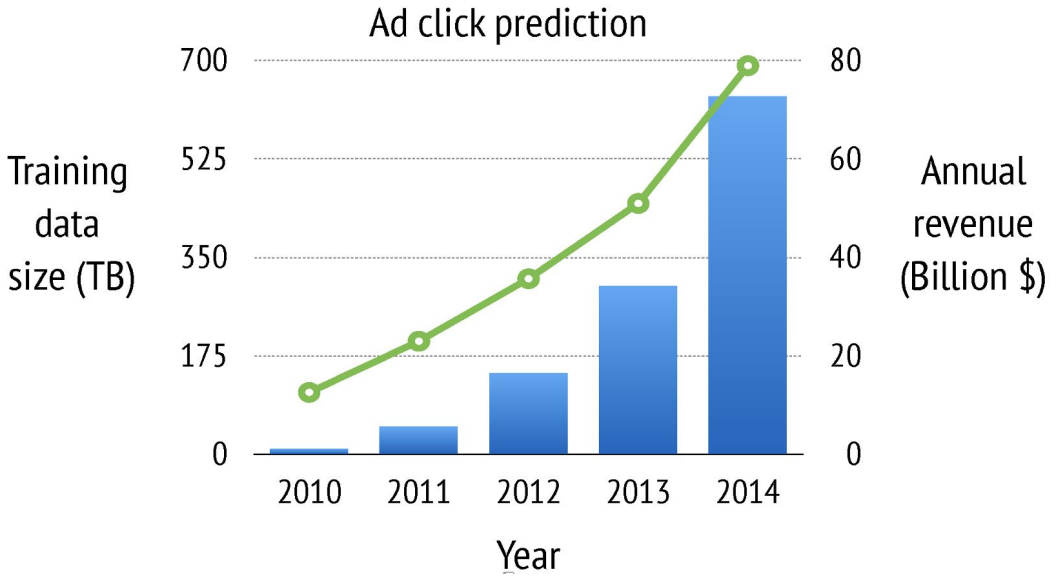
Presenters: Joe Peper, Chris Clarke, Roland Daynauth

Overview

- Background
  - Distributed Machine Learning
- Parameter Server Model
  - Goals
  - Architecture
    - Asynchronous tasks and Dependency
    - Flexible Consistency
    - Fault Tolerance
  - Evaluation

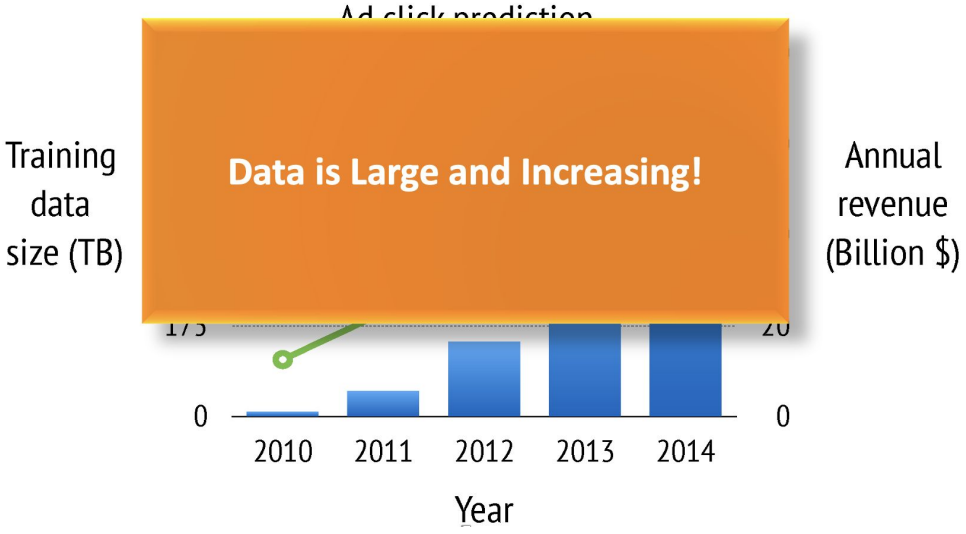
Distributed Machine Learning

- Many machine learning problems rely on large amounts of data for training and inference.



Distributed Machine Learning

- Such models consist of weights that will optimize for error in inference
- The number of weights/parameters run into orders billions to trillions thus making both learning and inference on a single machine impossible

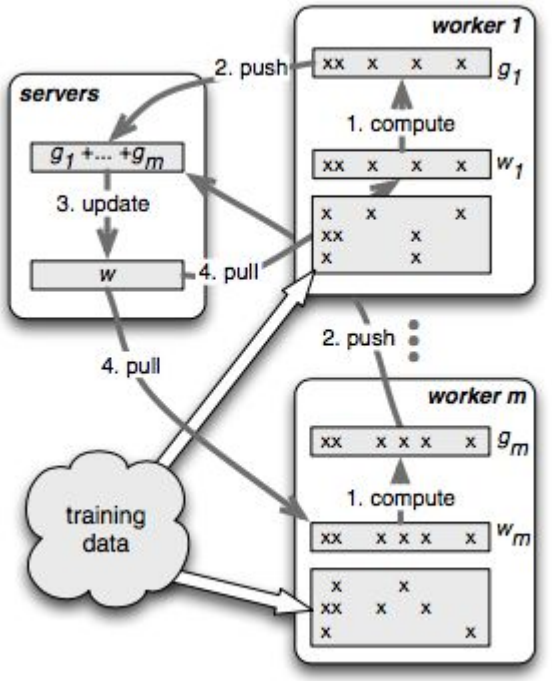


Challenges in Distributed Machine Learning

- Since these parameters need to be shared and updated across multiple nodes, these large numbers can be become bottleneck when it comes to sharing
  - Sharing is expensive in terms of bandwidth
  - Sequential ML jobs require **barriers** and hurt performance by blocking
  - At scale, fault tolerance is critical

Parameter Server: Goals

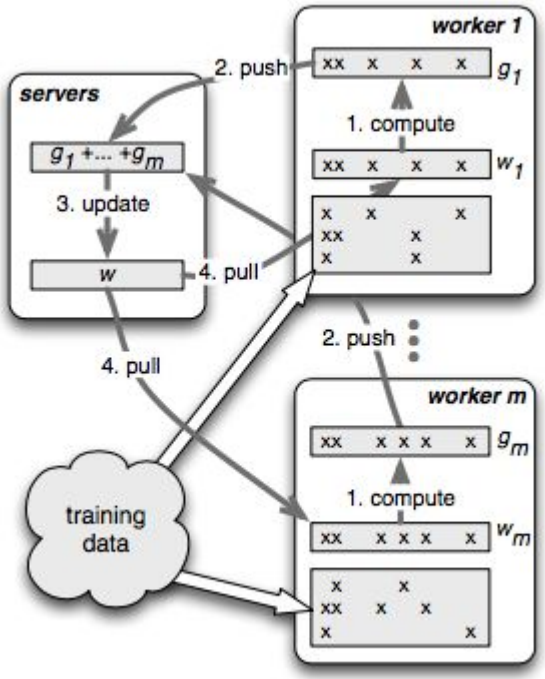
- To address the challenges mentioned, Parameter Servers aim for the following features:
  1. Efficient communication
  2. Flexible consistency models
  3. Elasticity for adding resources
  4. Efficient Fault tolerance
  5. Ease of use



Parameter Server: Architecture

- The Parameter Server appoints a single machine, the **parameter server**, the explicit responsibility of maintaining the current value of the parameters.
  - The most up-to-date gold-standard parameters are the ones stored in memory on the parameter server.
- The parameter server updates its parameters by using gradients that are computed by the other machines, known as **workers**, and pushed to the parameter server.
- Periodically, the parameter server **broadcasts its updated parameters** to all the other worker machines, so that they can use the updated parameters to compute gradients.

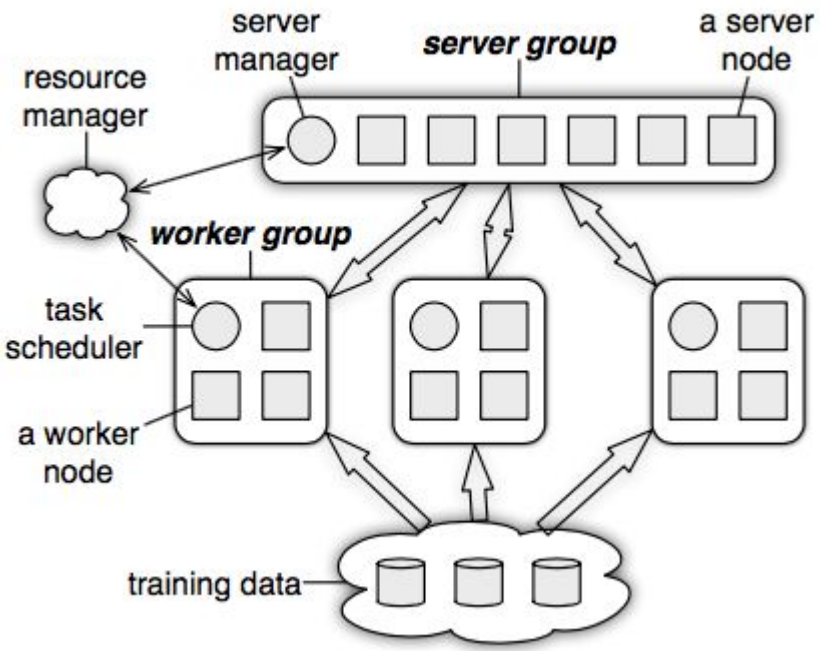
Parameter Server Model: Distributed Training Example



- Workers get the Assigned training data
- Workers **Pull** the Working set of Model
- Iterate until Stop:
  - Workers **Compute** Gradients
  - Workers **Push** Gradients
  - Servers **Aggregate** into current model
  - Workers **Pull** updated model

Parameter Server Model: Architecture

- Each server node in the server group is responsible for a partition of the keyspace/data.
- Servers can communicate with each other for migrating/replicating the data for scalability and availability.



Parameter Server: Dealing with parameters

- What are parameters of a ML model?
  - Usually an element of a vector, matrix, etc.
  - Need to do lots of linear algebra operations.
- Parameter Server approach models the Key-Value pairs as sparse Linear Algebra Objects.
- **Batch** several key-value pairs required to compute a vector/matrix instead of sending them one by one
- **Easy to Program!** – Lets us treat the parameters as key-values while endowing them with matrix semantics

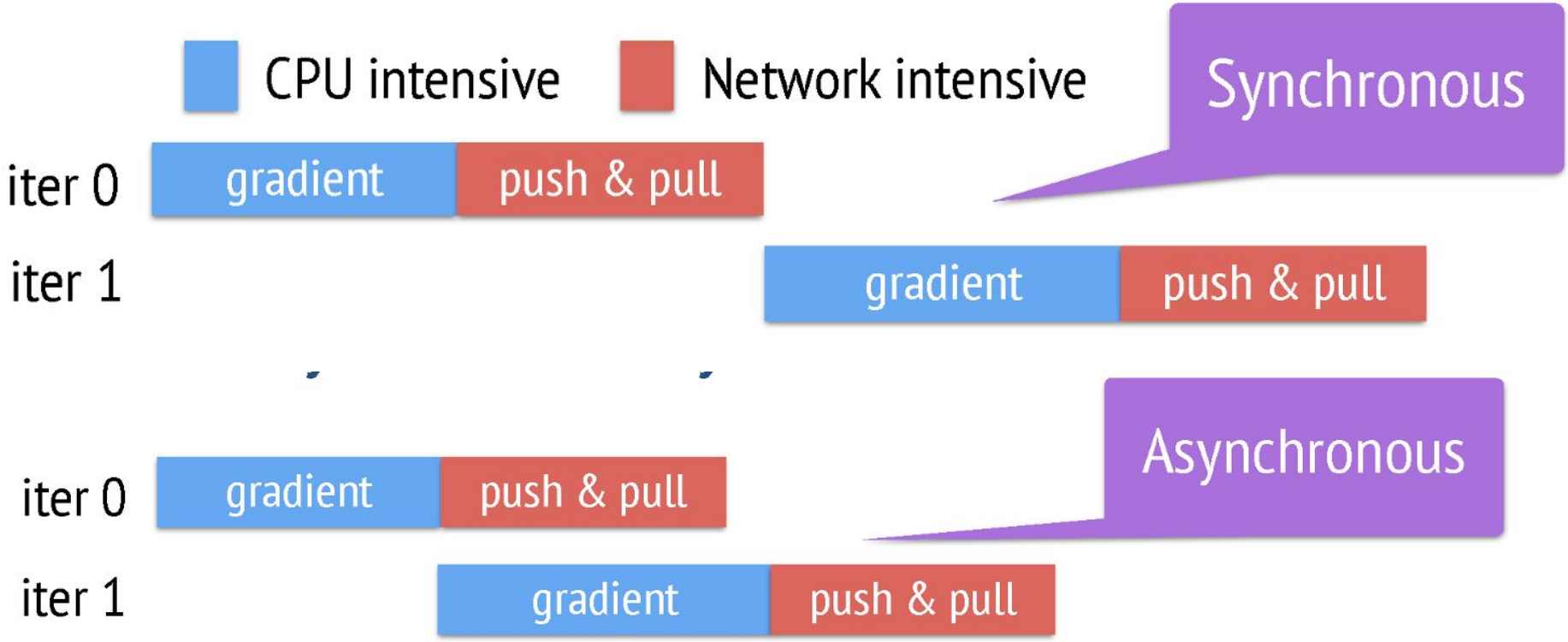
Parameter Server: Range based Push and Pull

- Weights are pulled from the server nodes and gradients are pushed to the server node.
- Parameter Server optimizes network bandwidth usage by using RANGE based PUSH and PULL.
- Example: Let **w** denote parameters of some model
  - w.push(**Range**, dest)
  - w.pull(**Range**, dest)
- These methods will send/receive all existing entries of **w** with **keys** in **Range**

Parameter Server: Asynchronous tasks and Dependency

- There is **MASSIVE** communication traffic due to frequent access of Shared Model .
- Global barriers between iterations leads to:
  - idle workers waiting for other computation to finish
  - High total finish time
- In Parameter Server tasks are generally asynchronous in nature and programs/applications can continue executing after issuing the task.
- A task can be marked as completed only when all subtasks that the given task depends on returns.

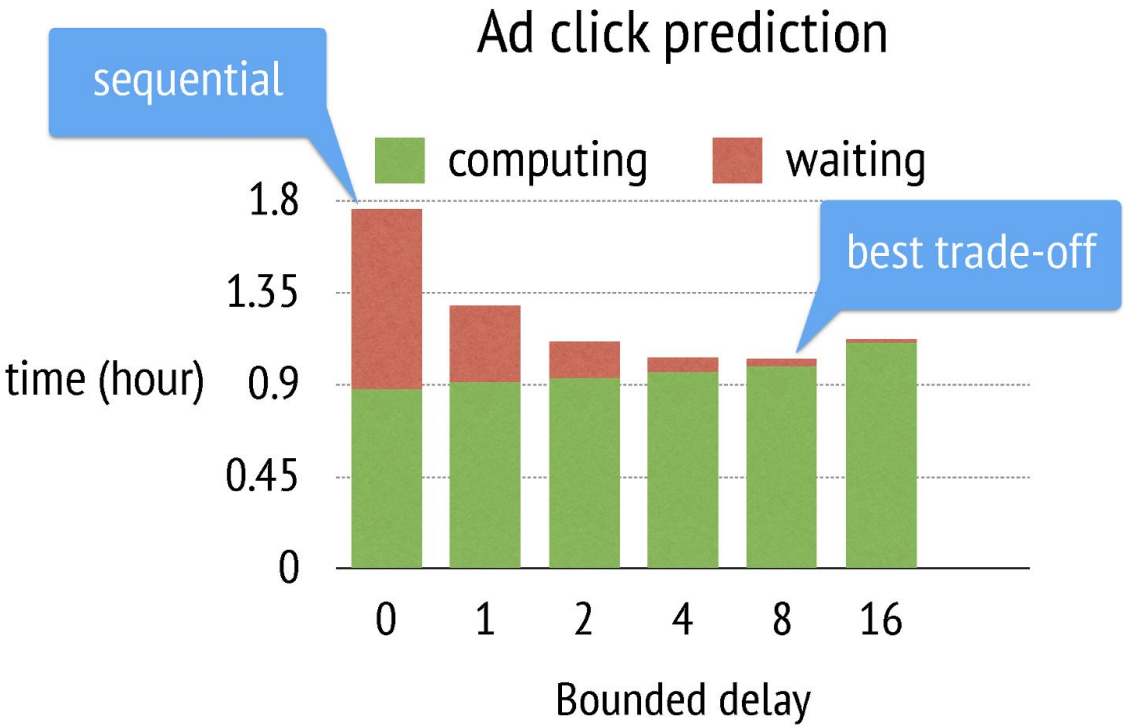
Parameter Server: Asynchronous tasks and Dependency



Parameter Server: Flexible Consistency

- By default tasks run in parallel and most often on remote nodes. Thus, where there is data dependency between various tasks, it may end up pulling old version of data.
- In machine learning, sometimes it is not too detrimental to pick up old weights or weights that are not too old, instead of the most recent weights.
- ParameterServer lets implementers select the consistency model that they are after.

Parameter Server: Flexible Consistency

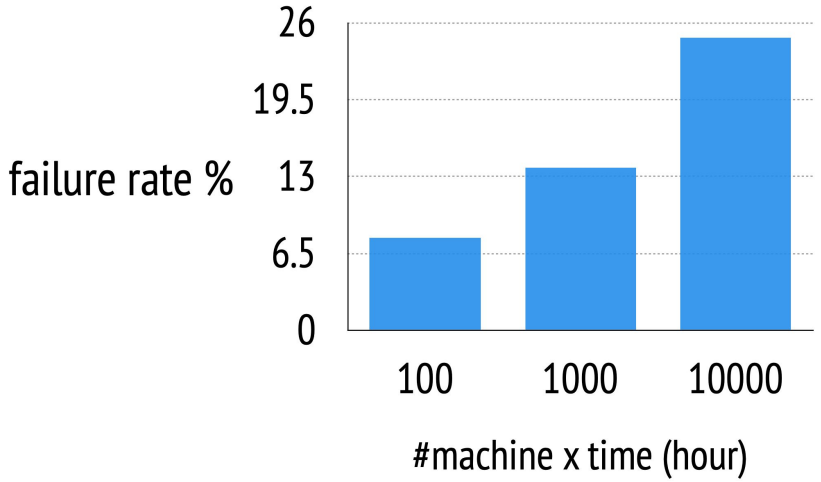


Parameter Server: User Defined Filters

- Selectively communicate (key, value) pairs
- Filters can be placed at either or both the Server machines and Worker machines.
- Allows for fine-grained consistency control within a task
- Example: Significantly modified filter:
  - Only pushes entries that have changed for more than an amount.
  - > 95% keys are filtered in the ad click prediction task

Parameter Server: Fault Tolerance

- They collected all job logs for a three month period from one cluster at a large internet company.
- Task failure is mostly due to being preempted or losing machines without necessary fault tolerance mechanisms.



Parameter Server: Fault Tolerance

- For fault tolerance and recovery, the following features were implemented:
  - Vector Clocks
    - establishes order of events in the system
  - Improved messaging
    - Compression and caching are used
  - Consistent Hashing
    - Allows for easy addition and removal of new nodes
  - Replication
    - Server nodes store a replica of (Key, value) pairs in k nodes counter clockwise to it.

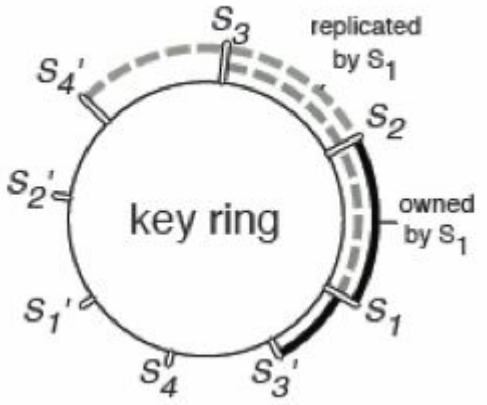
Parameter Server: Vector Clocks & Messaging

- Vector Clocks are attached for each (Key, value) pairs for several purposes:
  - Tracking Aggregation Status
  - Rejecting doubly sent data
  - Recovery from Failure
- As many (key, value) pairs get updated at the same time during one iteration, they can share the same clock stamps. This reduces the space requirements.
- Messages are sent in **Ranges** for efficient lookup and transfers.
- Messages are compressed using Google’s Snappy compression library.

Parameter Server: Consistent Hashing & Replication

- Consistent hashing is used for easy addition and removal of new nodes to the system. Every server node on the hashing ring is responsible for some keyspace.

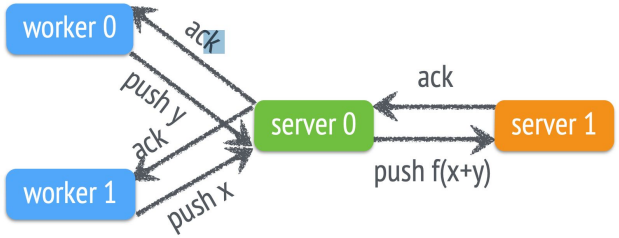
- Server nodes store a replica of (Key, value) pairs in k nodes counter clockwise to it.



Parameter Server: Fault tolerance

- Model is partitioned by consistent hashing
- Default replication: Chain replication (consistent, safe)

- Option: Aggregation reduces backup traffic (algo specific)



Parameter Server: Evaluation(Sparse Logistic Regression)

- One of the most popular large scale Risk Minimization algorithms.
- For example in the case of ads prediction, we want to predict the revenue an ad will generate.
- It can be done by running a logistic regression on the available data for ads which are ‘close to’ the ad we want to post.
- The experiment was run with:
  - 170 billion examples
  - 65 billion unique features
  - 636 TB of data in total
  - 1000 machines: 800 workers & 200 servers
  - Machines: 16 cores, 192 GB DRAM, and 10 Gb Ethernet links

	Method	Consistency
System-A	L-BFGS	Sequential
System-B	Block PG	Sequential
Parameter Server	Block PG	Bounded Delay + KKT

Parameter Server: Evaluation(Sparse Logistic Regression)

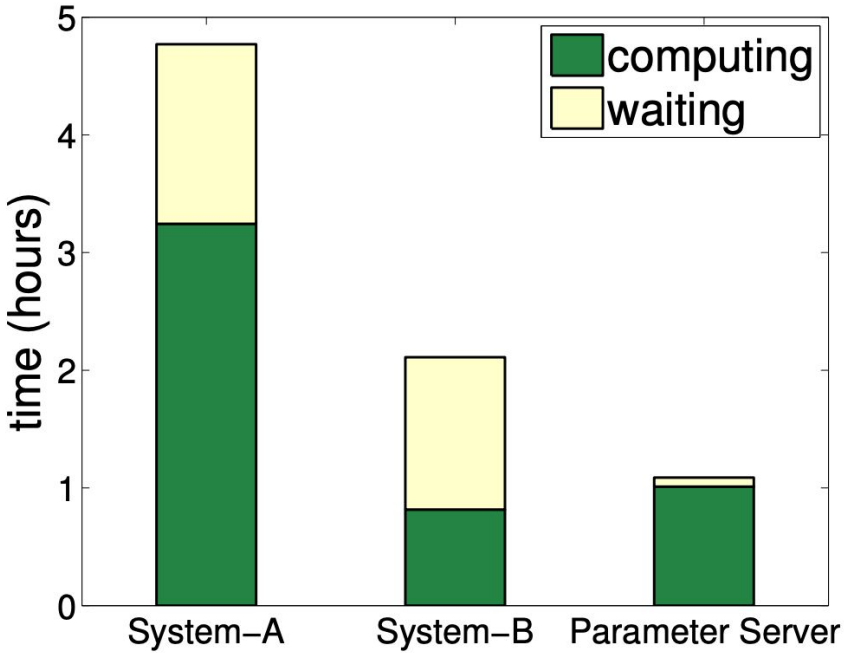


Figure 10: Time per worker spent on computation and waiting during sparse logistic regression.

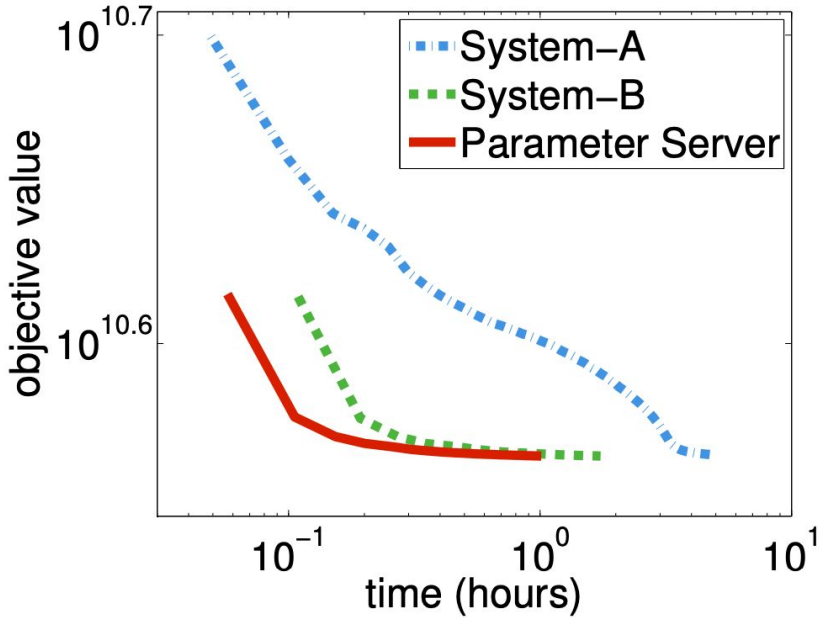
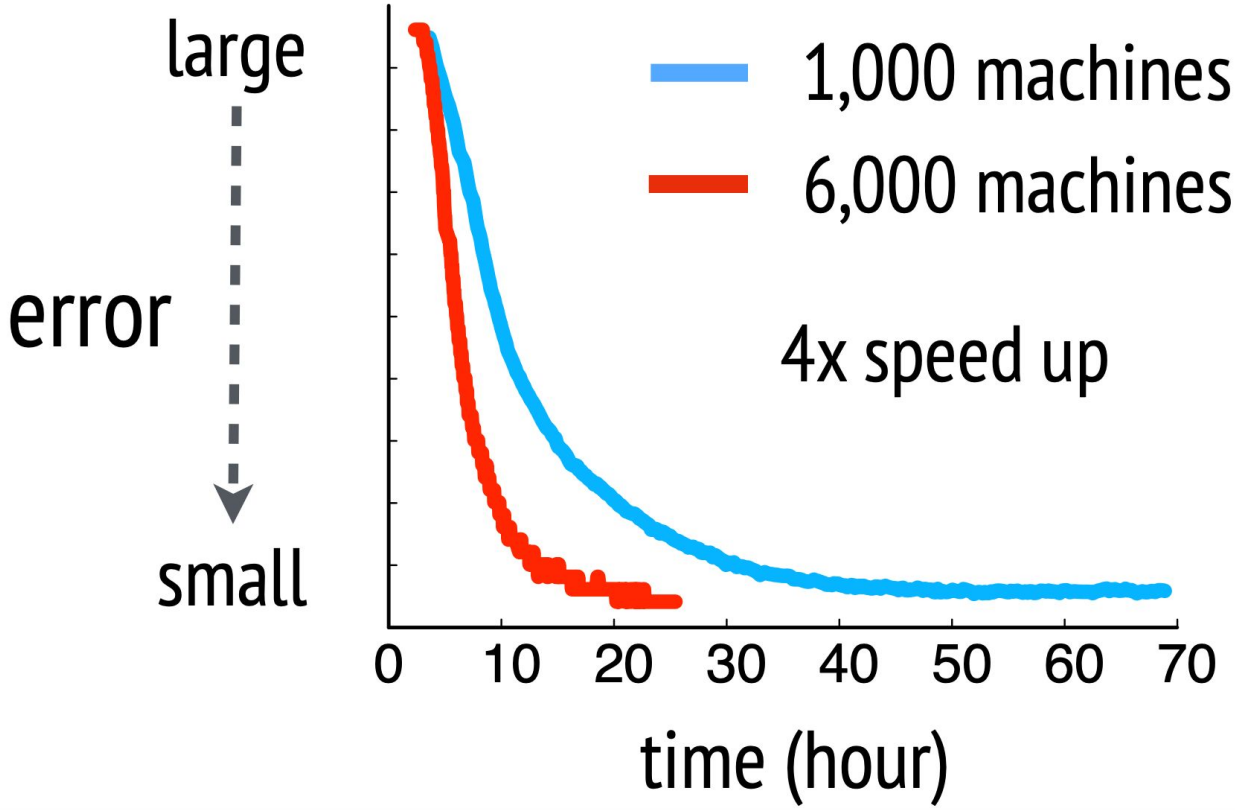


Figure 9: Convergence of sparse logistic regression. The goal is to minimize the objective rapidly.

Parameter Server: Evaluation (Latent Dirichlet Allocation)

- To demonstrate the versatility of the approach, they applied the same parameter server architecture to the problem of modeling user interests based upon which domains appear in the URLs they click on in search results.
  - **5B** users' click logs, Group users into **1,000** groups based on URLs they clicked.
  - **800** workers
  - **200** servers
  - **5000** workers
  - **1000** servers respectively

Parameter Server: Evaluation (Latent Dirichlet Allocation)



Conclusion

- Mu Li et al. introduce a parameter server framework that supports:
  - Asynchronous data communication between nodes
  - Flexible consistency models
  - Elastic scalability
  - Continuous fault tolerance.
- They demonstrate this scalability with experimental results on petabytes of real data with:
  - billions of examples and parameters
  - problems ranging from Sparse Logistic Regression to Latent Dirichlet Allocation and Distributed Sketching

Q&A

Project Adam: Building an Efficient and Scalable Deep Learning Training System

Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman, Microsoft Research

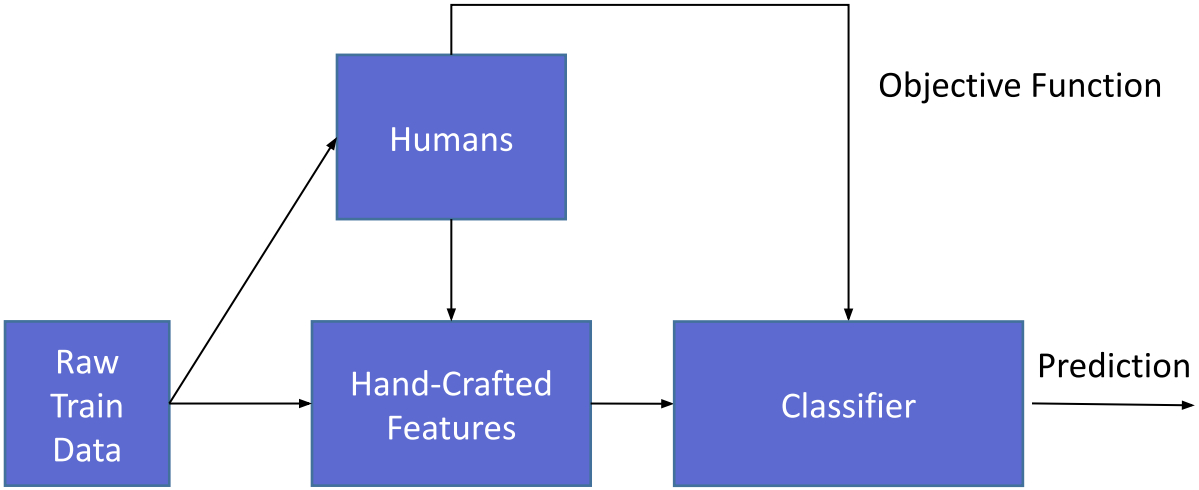
11th USENIX Symposium on Operating Systems Design and Implementation (ODSI '14)

Presenters: Joe Peper, Chris Clarke, Roland Daynauth

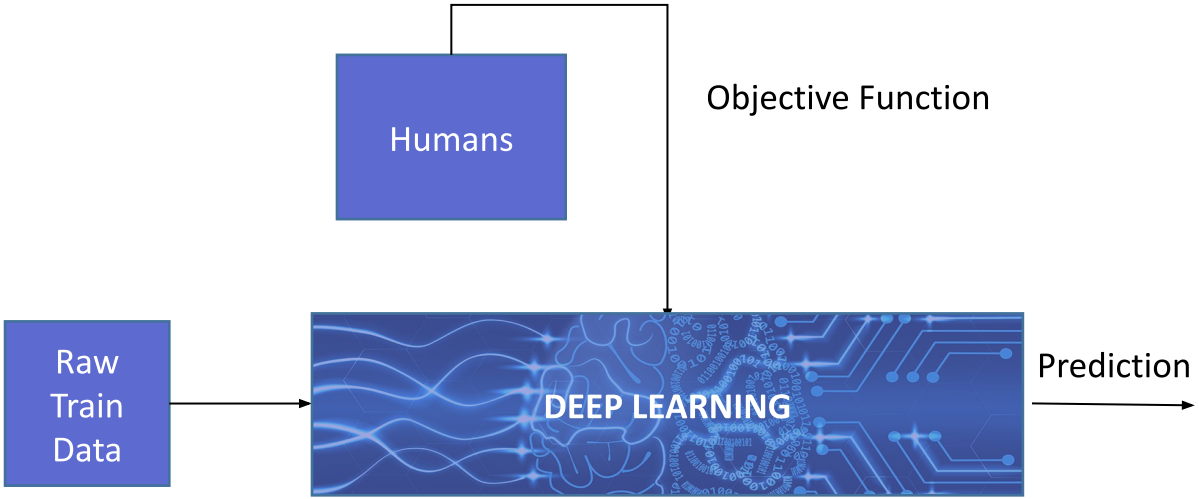
Overview

- Background on ML & DL
- Challenges with DL
- Project Adam
  - Objective + Summary
  - Key Optimizations
  - Evaluation + Results
- Analysis / Open Questions / Discussion

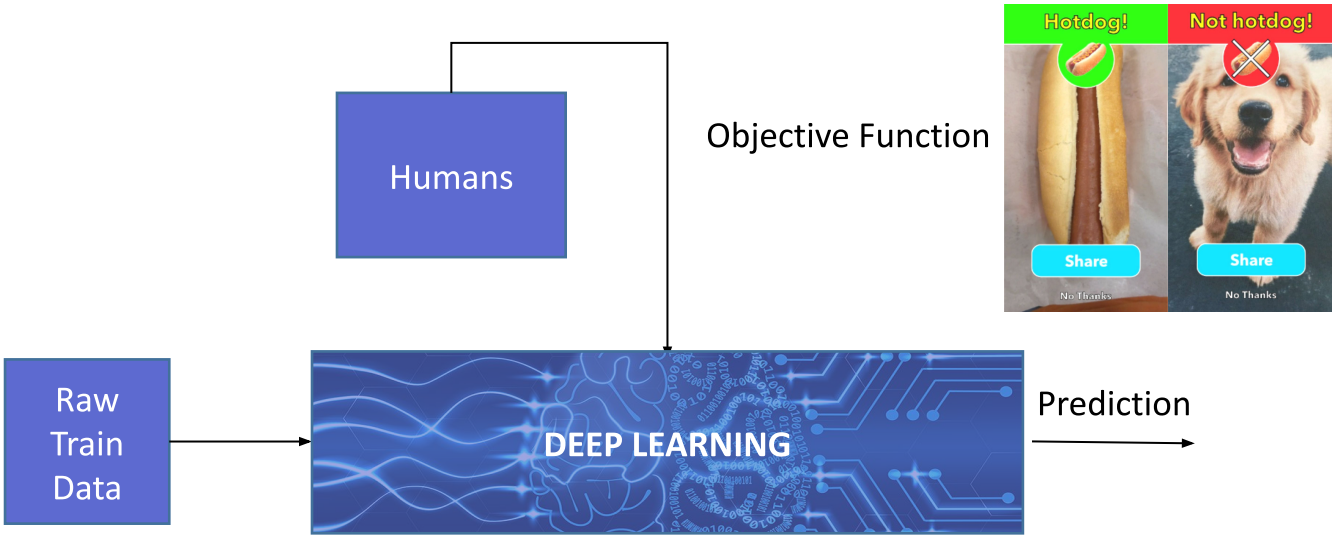
Classical ML relies heavily on human-in-loop feature engineering



Deep Learning eschews feature engineering for larger, data-hungry models



Deep Learning eschews feature engineering for larger, data-hungry models



DL models take in raw inputs and iteratively learn higher-level representations

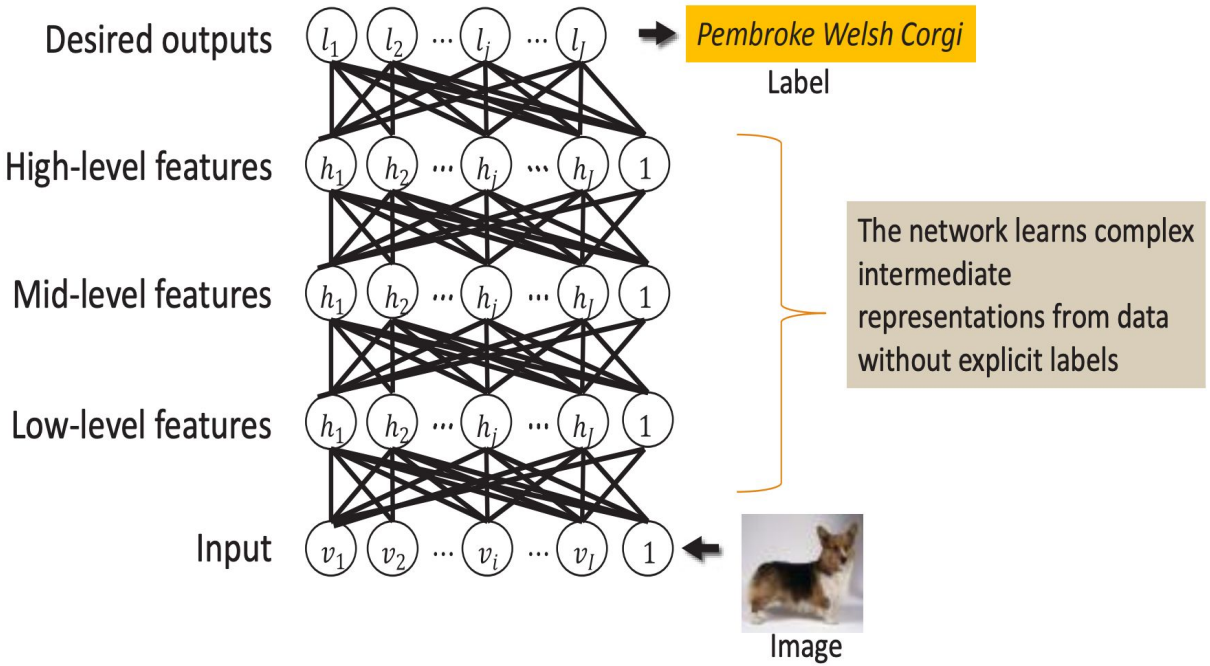
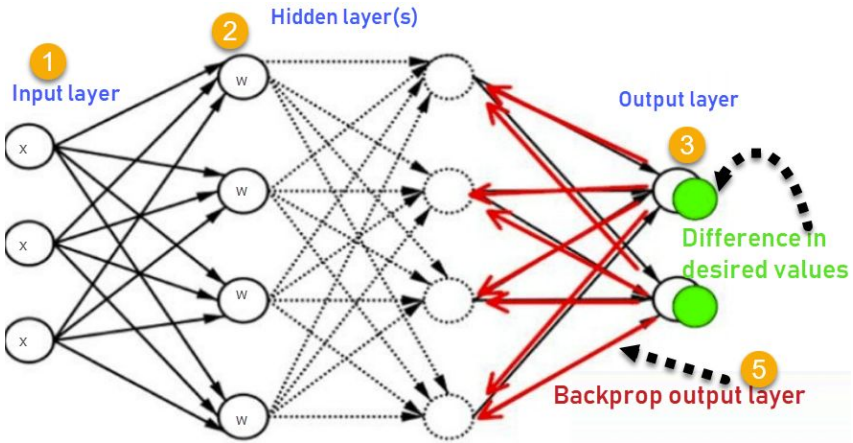


Figure 2. Deep networks learn complex representations.

Neural Network Training (Stochastic Gradient Descent)

- Run example(s) through network and make prediction
- Calculate the how erroneous the prediction was
- Calculate gradient of the error function with respect to each weight/parameter in the network
- Update the network weights to better predict on this example(s) in the future
    - Update is weighted by the learning rate to avoid severe updates / overcompensation



https://www.guru99.com/backpropogation-neural-network.html

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Accuracy scales with data and model size

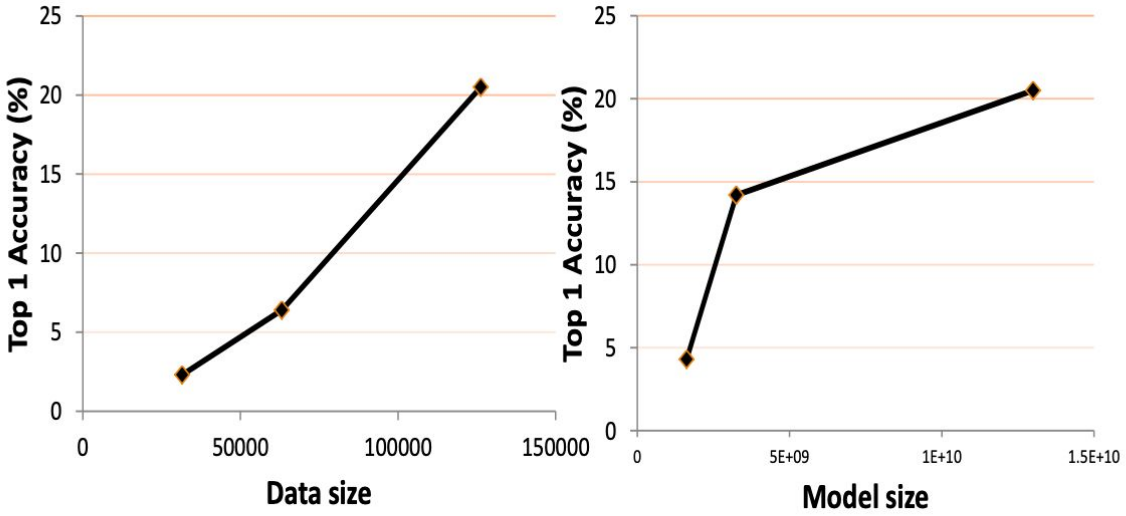


Figure 3. Accuracy improvement with larger models and more data.

DL compute requirements scales with model complexity and dataset size

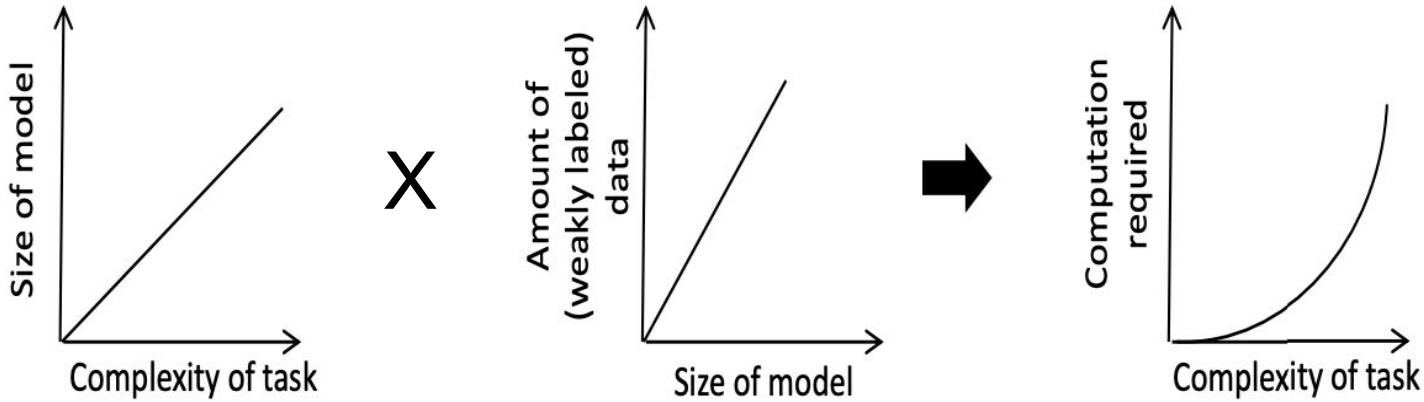


Figure 4. Deep Learning Computational Requirements.

Project Adam Highlights

- Focus on DL for computer vision / object identification
- Project Adam optimizes and balance DL computation and communication through a whole system co-design
  - Partition large models across machines to minimize memory bandwidth and cross machine communication
  - Restructure the computation across machines to reduce communication requirements.
- Achieves high performance and scalability by exploiting DL's ability to tolerate inconsistencies
  - Lock-less multithreaded model parameter updates without locks
  - Asynchronous batched parameter updates
  - Computing using potentially stale parameter values
- Its efficiency, scaling, and asynchrony actually contribute to improved model performance

Project Adam heavily leverages existing work and focuses on optimizing known painpoints and bottlenecks

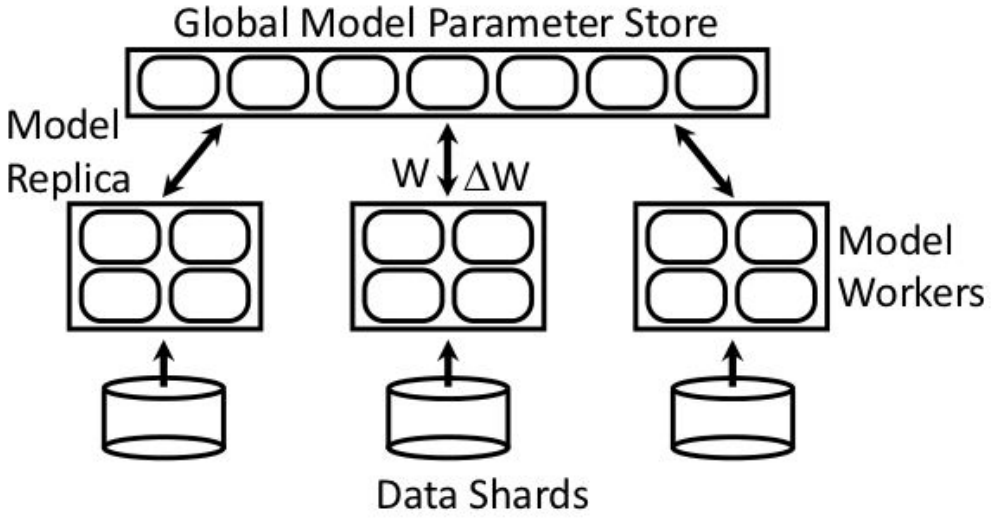
Related work on Distributed DL Training

- Multi-Spert [*Faerber et al.*]
- DistBelief [*Dean et al.]*
  - Poor scaling efficiency
  - Not viable cost-effective solution

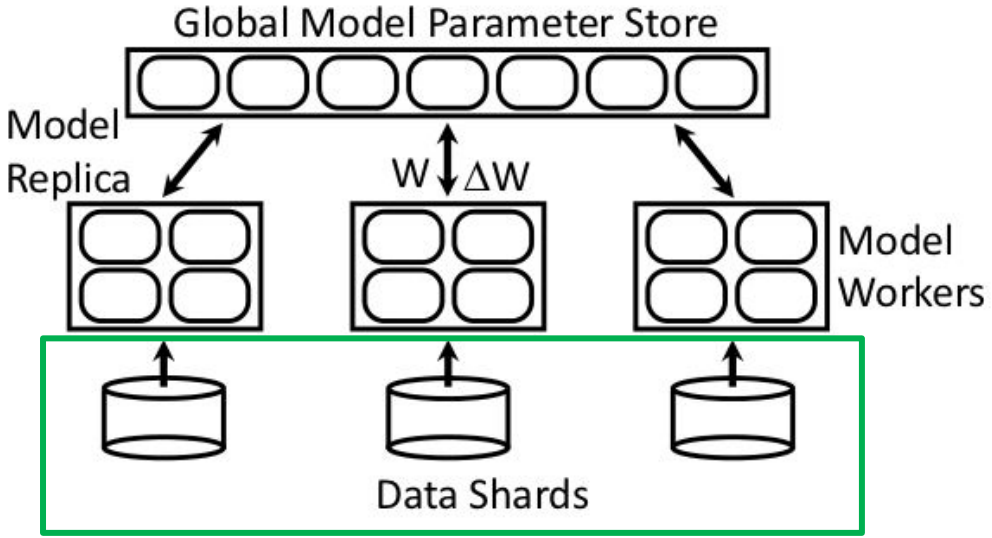
“Our high-level system architecture is also based on the Multi-Spert system and consists of data serving machines that provide training input to model training machines organized as multiple replicas that asynchronously update a shared model via a global parameter server. “

“While describing the design and implementation of Adam we focus on the computation and communication optimizations that improve system efficiency and scaling. These optimizations were motivated by our past experience building large-scale distributed systems... “

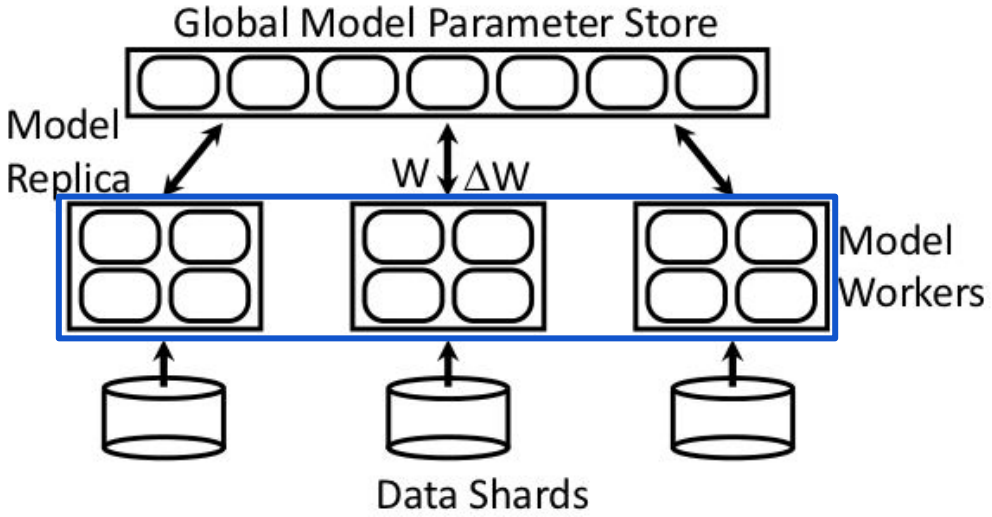
Adam architecture is comprised of 3 types of machines



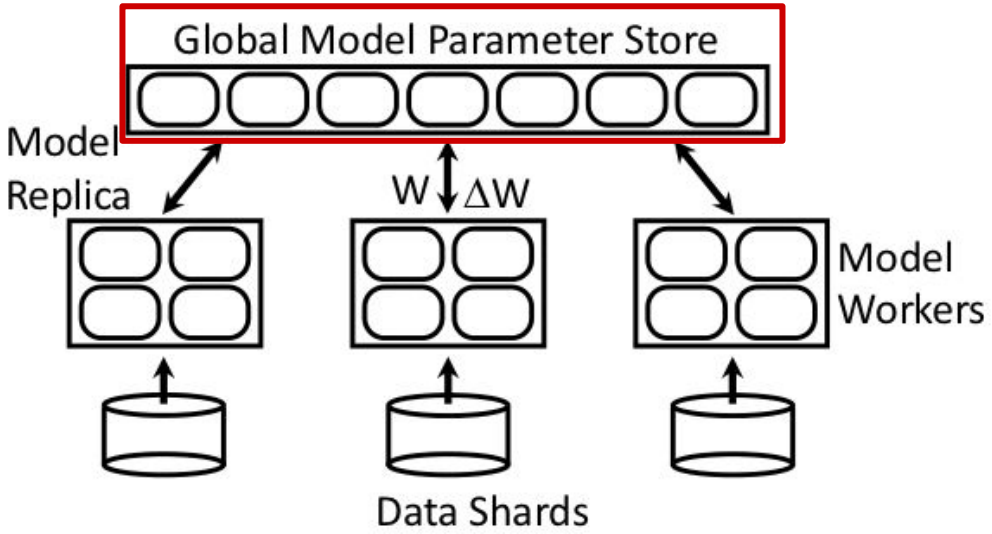
Data transformation machines serve and manipulate data for better generalization



Model workers machines do the primary training computation. (model replicas are comprised of multiple worker machines)



Parameter machines store the global parameters



Key Optimizations

Data serving machines enable fast dataflow and data computations

- Data requires manipulation to avoid overfitting
  - Rotation, scaling, inversion, etc…
- Data machines pre-cache images in memory
- Large quantities of data needed (10+ TBs)
- Model training machines fetch data via async calls (TODO)

Model worker machine optimizations

- Using lockless weight updates, they can take advantage of commutative and associative properties
- Multi-threaded training is enabled
- They find model performance is unimpaired by this laxing of constraints

Model worker machine optimizations

- Non-local data communication is minimized by sending data *pointers* of neurons whose values need to be computed
- Model partitioning enables model working set to live in L3 cache
- Avoid slow machine bottlenecks by ending training after 75% of model replicas have finished training
    - Apparently no impact on accuracy 🤔

Vertical partitioning reduces cross-machine communication during training

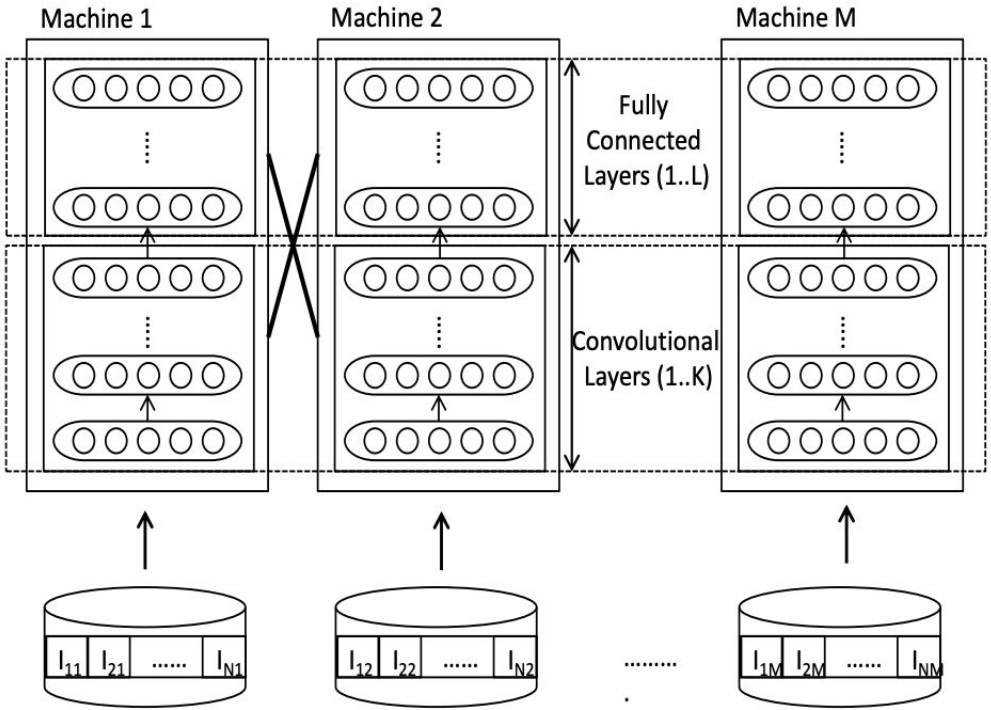
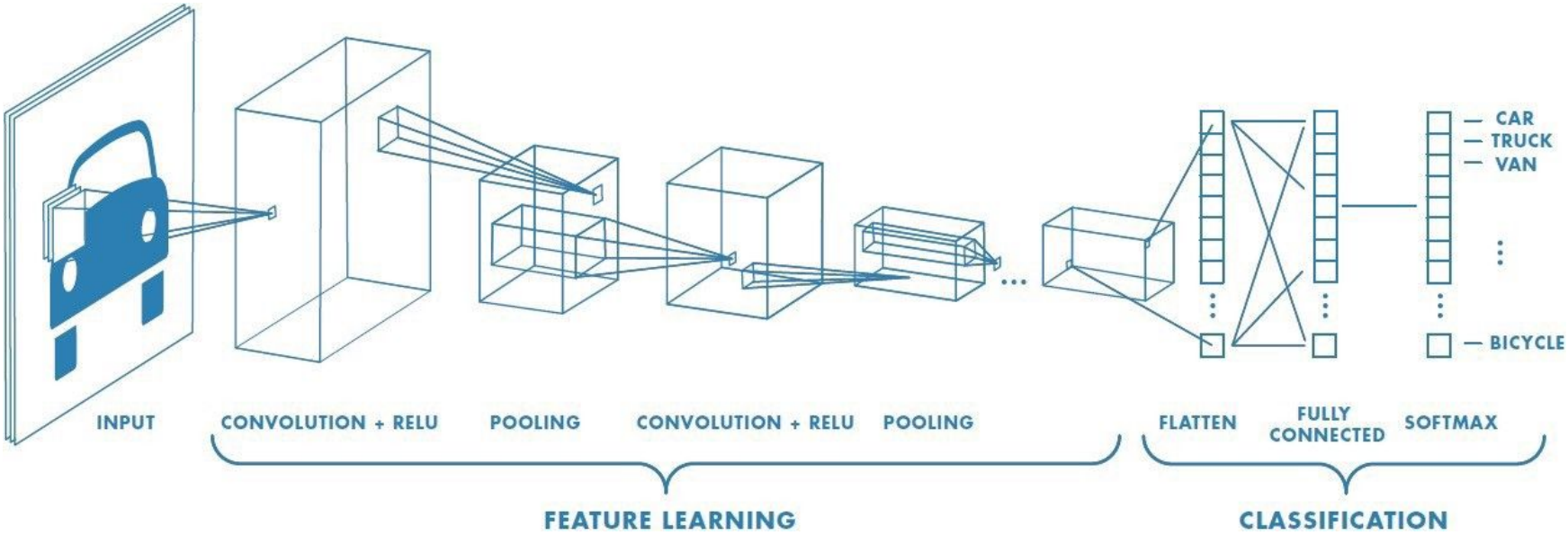


Figure 6. Model partitioning across training machines.

Vertical partitioning reduces cross-machine communication during convolutional layers



Global Parameter server optimizations

Two protocols for communicating parameter weight updates

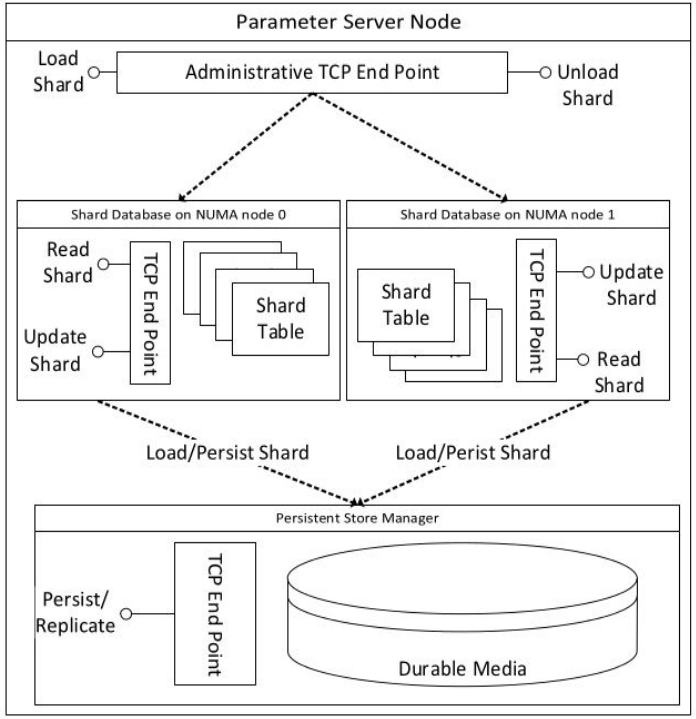
1. Convolution layers: locally compute and accumulate weight updates and send to parameter server. Good for sparse layers

2. FC layers: Send the activation and error gradient vectors to the parameter servers so that weight updates can be computed there
    - This reduces weight traffic volume from M*N to k*(M+N)
    - Also offloads compute to underutilized parameter machine, freeing up more for the training

Global Parameter server optimizations

- Lock-free parameter updates improve scalability
- Opportunistic batched updates improves temporal locality
- Model parameters divided into 1 MB shards
  - Improves spatial locality of update processing
- Shards hashed into storage buckets distributed equally among parameter servers
  - Improves load balancing
- Each machine has 2 independent 10Gb NICs
  - One for weight update processing
  - One for persistence
  - Also a 1Gb NIC for administrative traffic

Parameter server node



Global Parameter server optimizations

Delayed Persistence

- Parameter storage modelled as write back cache
- Potential data loss tolerable by Deep Neural Networks due to their inherent resilience to noise
  - The lock-free updates introduce ‘noise’ already too
  - updates can be recovered if needed by retraining the model
- Allows for compressed writes to durable storage as many updates can be folded into a single parameter update, due to the additive nature of updates, between rounds of flushes.

Fault Tolerant Operation

- The parameter servers are controlled by a set of parameter server (PS) controller machines that form a Paxos cluster
- Three copies of each parameter shard in the system
  - Each stored on different parameter servers

Evaluation + Results

- Object Identification Benchmark Overview
- System Configuration
- MNIST Benchmark Results
- ImageNet-22k Results + System Scaling Analysis

Evaluation + Results

- System Configuration
- MNIST Benchmark Results
- ImageNet-22k Results + System Scaling Analysis

System Configuration

120 identical HP Proliant machines

- Organized as three equally sized racks connected by IBM G8264 switches
- 16 total cores @1.8GHz, 98GB memory
- 2 x 10Gb, 1 x 1Gb NICs

90 Model Training Machines

20 Parameter Server Machines

10 Data/Image Server Machines

Some of each are reserved for fault tolerance.

MNIST

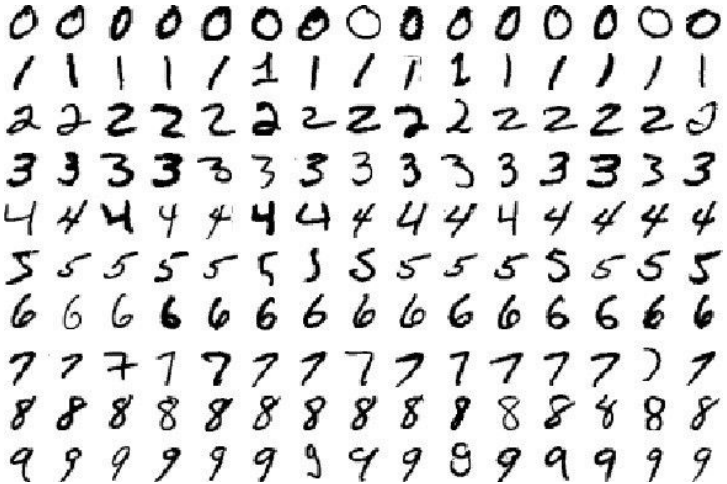
Task: Handwritten digit identification

Num classes: 10

Train set size: ~60k examples

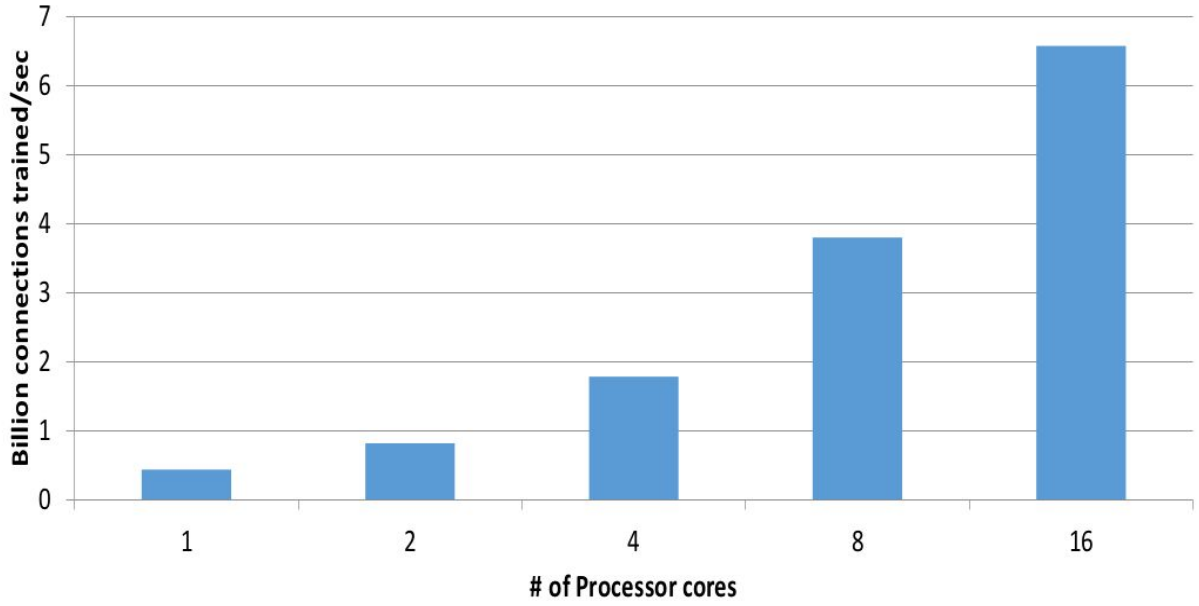
Experiment Setup:

- One training machine
- One parameter server

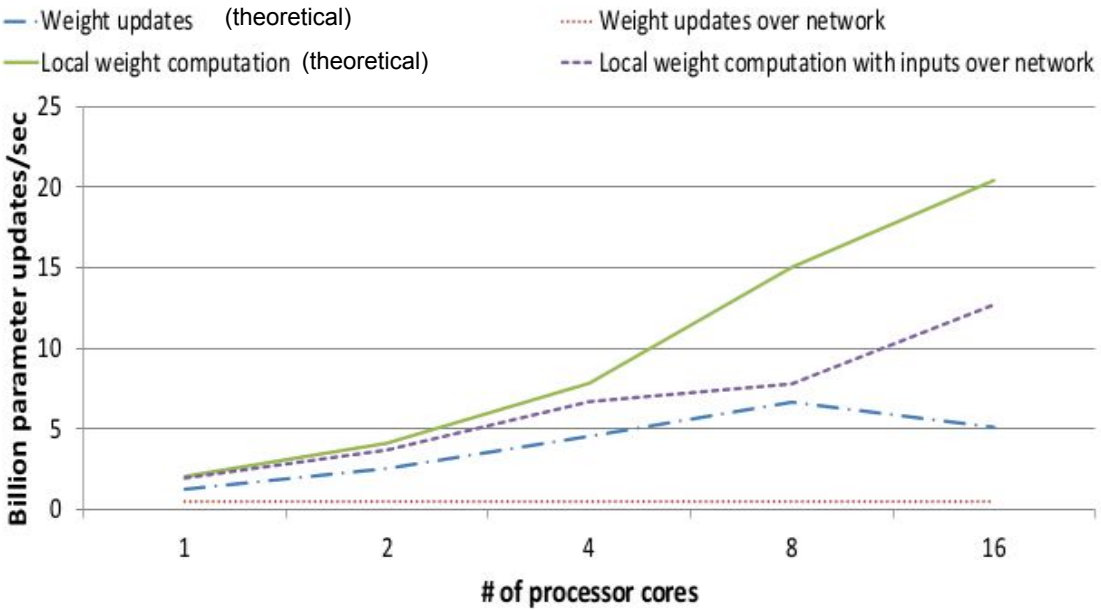


https://miro.medium.com/max/1168/1*2lSjt9YKJn9sxK7DSeGDyw.jpeg

Single machine (no PS) speed scales linearly+ with number of cores



Network throughput is bottleneck for weight updates, but is largely solved by sending only update inputs and doing the actual update compute on the Parameter Servers



Adam (async.) actually outperformed Adam (sync.) and the existing SoTA, contradicting conventional wisdom

Table 1. MNIST Top-1 Accuracy

Systems	MNIST Top-1 Accuracy
Goodfellow et al [12]	99.55%
Adam	**99.63%**
Adam (synchronous)	99.39%

ImageNet-22k

Task: Object classification

Num classes: 22,000

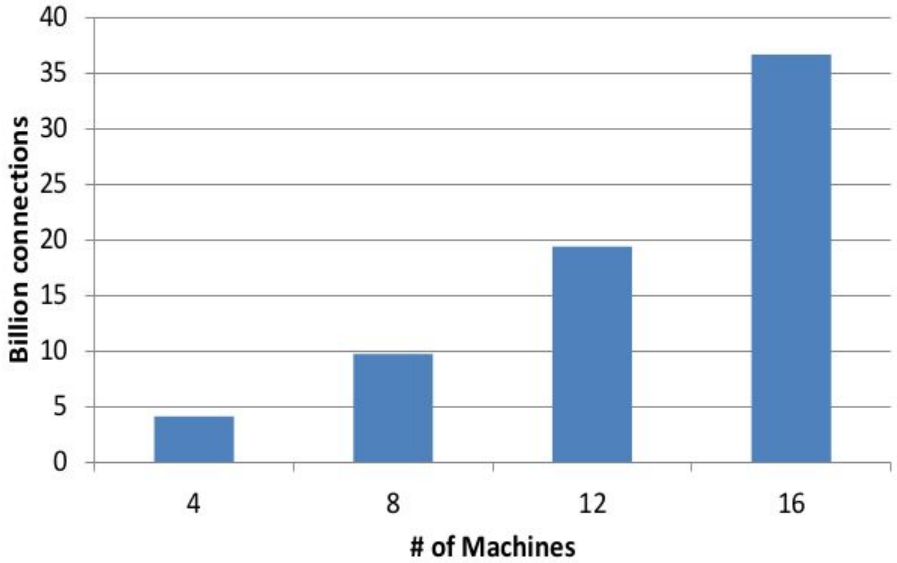
Train set size: 15 million images

Shiba Inu

Corgi

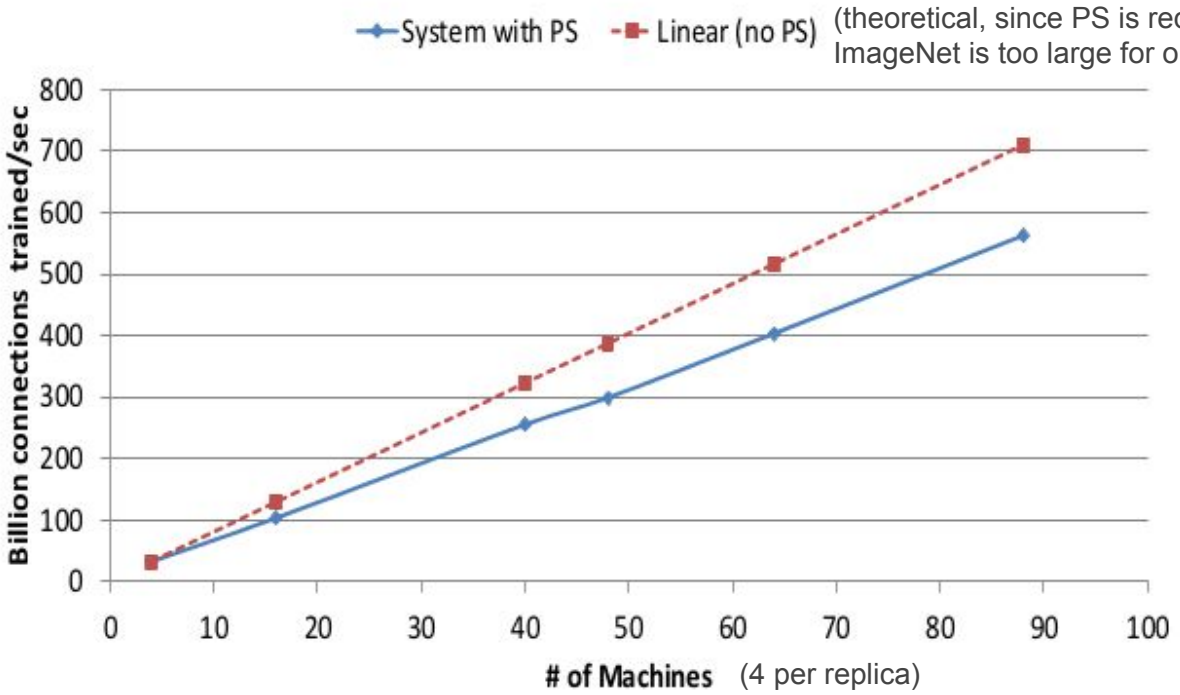
https://dogzone-tcwebsites.netdna-ssl.com/wp-content/uploads/2020/10/Shiba-Inu-vs-Corgi-2.jpg

The number of supported model parameters scales super-linearly with the number of machines the model is partitioned across



Note: Model replicas consist of 4 machines in all future references

The system exhibits good scaling even though the PS does add some overhead



(theoretical, since PS is required because ImageNet is too large for one machine)

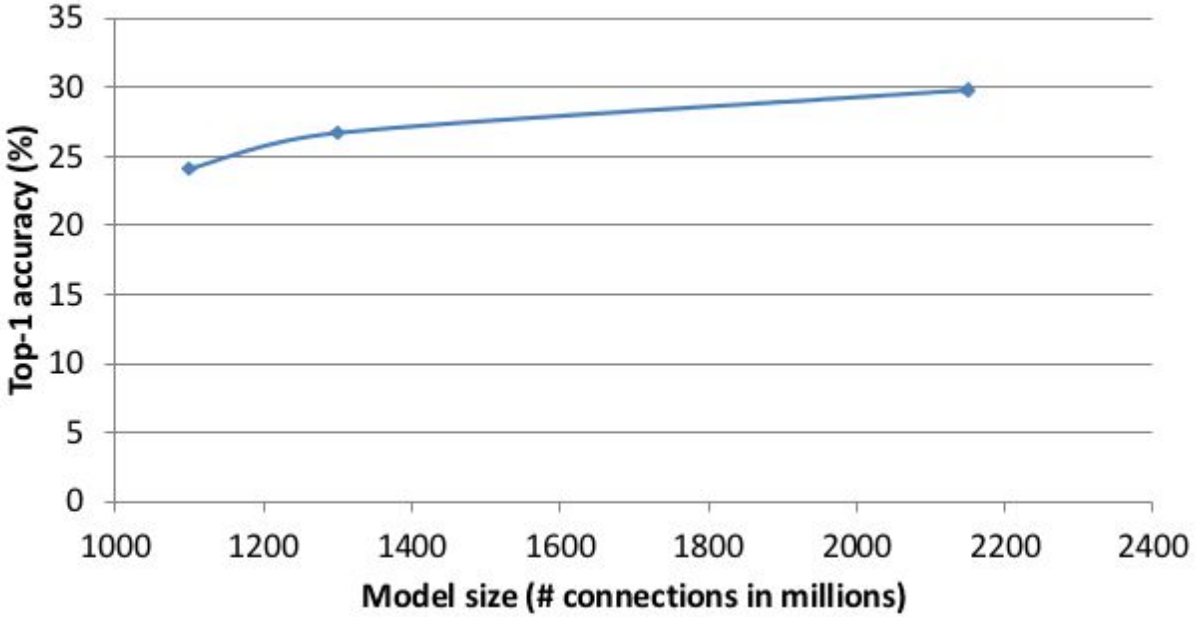
(4 per replica)

Adam beats state-of-the-art by a wide-margin*

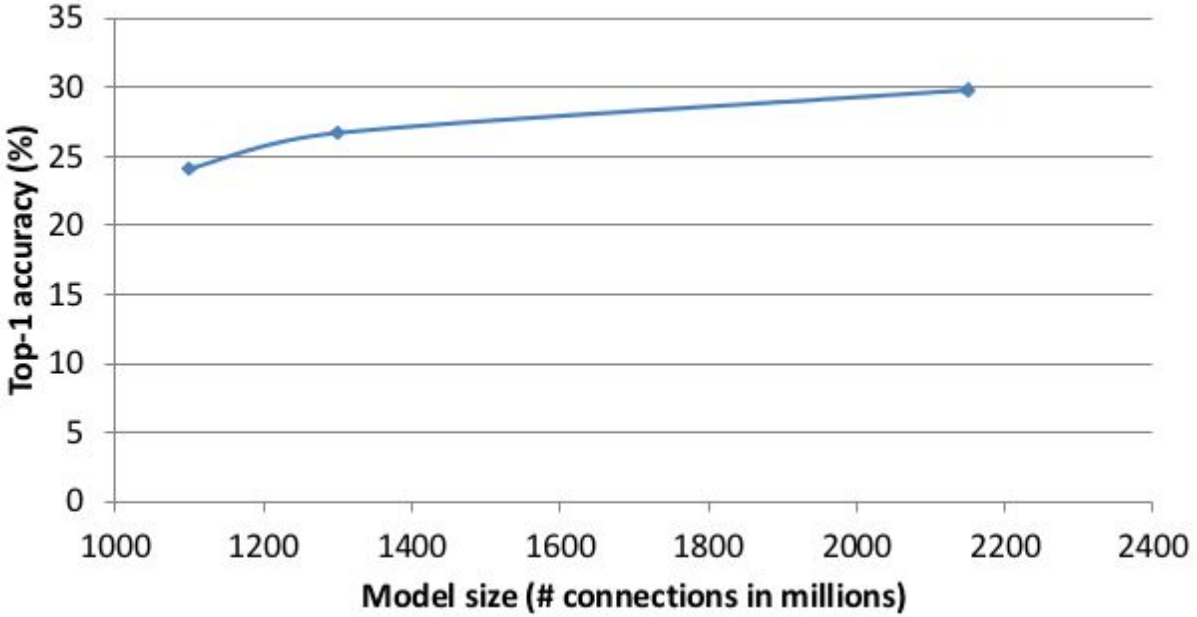
Systems	ImageNet 22K Top-1 Accuracy
Le et al. [18]	13.6%
Le et al. (with pre-training) [18]	15.8%
Adam	**29.8%**

* A cursory glance indicates *Le et al., "Building high-level features using large scale unsupervised learning",* used a purely unsupervised approach, so I'm not sure this is an apples-to-apples comparison

Improved accuracy with larger models validates the needs for continued work on scaling DL efficiently



Improved accuracy with larger models validates the needs for continued work on scaling DL efficiently



Final Thoughts

Strengths

- State-of-the-art accuracy on image classification datasets*
- Very optimized and scalable
- Performant even with ‘noisy’ asynchronous weight update approach
- Their asynchronous approach appears to perform better than synchronous

Weaknesses

- No evaluation on other DL tasks (e.g. natural language processing)
- Their ML evaluation seemed a bit weak, especially given their unusual findings

Questions?