Machine Learning Parallelism Could Be Adaptive, Composable and Automated

Ph.D. Thesis Oral

Hao Zhang
Machine Learning Computation

Training data: images w/ labels

Computational Worker w/ GPUs
Distributed Machine Learning

Large scale Training data

A Cluster of Workers with GPUs

Hao Zhang
Nowadays ML Models

Highly Structured

Increasingly Composable

John hit the ball
Distribute a Model

\[ \theta(t) = \theta(t-1) + \varepsilon \cdot \nabla \mathcal{L} \left( \theta(t-1), D^{(t)} \right) \]

Model

Data
Distribute a Model

Communication Architecture
- Parameter Server
- AllReduce/Collective
- P2P
- Partial Broadcasting

Consistency Models
- Synchronous
- Asynchronous
- Stale Synchronous

Partitioning/Placement
- Graph Partitioning
- Operator Partitioning
- Device Placement

Encoding/Decoding
- Gradient Quantization
- Deep Compression

Scheduling
- WFBP
- Pipeline Parallelism
- Byte Scheduler
**Usability Challenges: Implementations**

- **Horovod**: Sergeev et al. 2018
- **Bosen**: Wei et al. 2015
- **BytePS**: Peng et al. 2019
- **Orpheus**: Xie et al. 2019

- **Deep Compression**: Han et al. 2016
- **ColocRL**: Mirhoseini et al. 2017
- **FlexFlow**: Jia et al. 2019
- **Tofu**: Wang et al. 2019

- **tf.distribute.strategy**: Shazeer et al. 2018
- **Mesh-TensorFlow**: Zhang et al. 2017
- **Poseidon**: Kim et al. 2019
- **Parallax**: Next...

- **GeePS**: Cui et al. 2018
- **IterStore**: Cui et al. 2014
- **Litz**: Qiao et al. 2017
Usability Challenge: Distinct Interface

Operator Partitioning

Mesh-TensorFlow

# tf_images is a tf.Tensor with shape [388, 20, 20] and dtype tf.float32
# tf_labels is a tf.Tensor with shape [388] and dtype tf.int32
graph = tf.Graph()
echo = tf.GraphDef(), "my-model"
batch_dim = tf.Dimension("batch"), 188
row_dim = tf.Dimension("row"), 88
col_dim = tf.Dimension("col"), 88
hidden_dim = tf.Dimension("hidden"), 128
classes_dim = tf.Dimension("classes"), 188
images = tf.import graph_def(
  mesh, tf_images, shape=[batch_dim, row_dim, col_dim])
labels = tf.import graph_def(
  mesh, tf_labels, tensor="labels")

# With PyTorch, each rank can keep its own copy of the data
# Collectives can be used to average gradients
# This is possible because of the broadcast semantics

Graph Partitioning

ColocRL

# with tf.device("/job:local/task:0"):
#    first_batch = tf.alsoe, [8], [8])
#    mean = tf.reduce_mean(first_batch)
# with tf.device("/job:local/task:8"):
#    second_batch = tf.alsoe, [8], [8])
#    mean = tf.reduce_mean(second_batch)
#    mean = (mean1 + mean2) / 2

Write partitioning layout ("mesh")

Specify op-device mapping

Collective AllReduce

Horovod

# Add Horovod DistributedOptimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Make training operation
train_op = opt.minimize(loss)

Patch optimizers

Manually average grads

Collected AllReduce

PyTorch

--- Distributed samplers for TensorFlow ---

def train_step(x, labels):
  with tf.GradientTape() as tape:
    # Define the model
    y = model(x)
    loss = tf.reduce_mean(loss)

  # Compute the gradients
  grads = tape.gradient(loss, model.trainable_variables)

  # Apply the gradients
  optimizer.apply_gradients(zip(grads, model.trainable_variables))

  return loss, y

train_dataset = tf.data.Dataset.from_tensor_slices((x, labels))

for epoch in range(num_epochs):
  for x, y in train_dataset:
    # Train the model
    loss, y = train_step(x, y)
    # Save the model
    checkpoint.save()
Performance Challenge #1

Lack of sub-model optimization: the overall distribution strategy of a model is the composition of the fittest strategy of each model building block.

Strategy A
Strategy B
Strategy C
Strategy D
Strategy E

Figure from Carion et al. 2020
Existing Systems are Monolithic

- Collective AllReduce
  - Horovod
- Parameter Server
  - Bosen
  - BytePS
- SSP
  - Orpheus
  - Gradient Encoding
    - Deep Compression
- SFB (P2P)
  - IterStore
  - GeePS
  - Graph Partitioning
    - ColocRL
  - Operator Partitioning
    - FlexFlow
    - Tofu
    - Mesh-TF
Performance Challenge #2

Co-optimization: Different aspects of ML parallelization need to be considered together.

- Communication Architecture: PS?
- Consistency: staleness = 0?
- Gradient enc/dec: 1-bit?
- Replication/placement: GPU:0, GPU:1?
- Partitioning: [5, 1, 1]?

Strategy A:
- Communication Architecture: Collective AllReduce?
- Consistency: staleness = 5?
- Gradient enc/dec: no?
- Replication/placement: replicated over all devices?
- Partitioning: no?

Strategy B:
- Communication Architecture: PS?
- Consistency: staleness = 0?
- Gradient enc/dec: 1-bit?
- Replication/placement: replicated over all devices?
- Partitioning: no?
Summary of Challenges

**Usability #1**: Matching (model, resource) with the right distributed strategy is hard.

**Usability #2**: Distinct system implementations and interfaces -> added development overhead.

**Performance #1**: Increasingly more complex model structures -> opportunities sub-model strategy optimization.

**Performance #2**: Monolithic system design -> lower-than-expected performance and lack of cooptimization of multiple parallelization aspects.
Goal

- **Communication Architecture**
  - Parameter Server
  - P2P
  - Partial Broadcasting

- **Consistency Models**
  - Synchronous
  - Asynchronous
  - Stale Synchronous

- **Partitioning/Placement**
  - Graph Partitioning
  - Operator Placement
  - Device Placement

- **Encoding/Decoding**
  - Gradient Quantization
  - Compression

- **Scheduling**
  - WFBP
  - Pipeline Parallelism
  - Byte Scheduler

**Strategy auto-generation**

**Representation of ML parallelisms**

**Understanding and unifying various ML parallelisms**
The structured and composable nature of nowadays ML programs allows for building optimized parallelization strategies that are adaptive to model and cluster specifications, composable from different ML parallelisms, and auto-generated via learning-based methods.
Research Summary

Thesis Components

Understanding and optimize ML parallelization with adaptive parallelisms

Representation and Composability for ML parallelisms

Automating ML parallelization

Thesis-related Work

Communication

Consistency Model

Scheduling

Memory Management

Poseidon [ATC’17]

Impact of staleness [ICLR’19]

Poseidon [ATC’16]

GeePS [Eurosys’16]

Representations for dynamic batching

Derive

DyNet

Representations for distributed parallelism

Derive

AutoDist

Distributed Strategy Auto-optimization

AutoDist [Preprint]

AutoLoss [ICLR’19]

AutoSync [preprint]

Cavs [ATC’18]

Cavs

Open source systems

Petuum-proprietary

Lead

Co-author

Hao Zhang
Thesis Structure

Thesis Components

1. Understanding and optimize ML parallelization with adaptive parallelisms
   - Part 1

2. Representation and Composability for ML parallelisms
   - Part 2

3. Automating ML parallelization
   - Part 3

Thesis Contributions and Chapters

- Communication: Chapter 3
- Consistency Model: Chapter 3
- Scheduling: Chapter 4
- Memory Management: Chapter 5

- Representations for dynamic batching: Chapter 6
  Derive: DyNet

- Representations for distributed parallelism: Chapter 7
  Derive: AutoDist

- Distributed Strategy Auto-optimization: Chapter 8
  Derive: Cavs

Hao Zhang

Carnegie Mellon University
The Robotics Institute

16
The Rest of Today’s Talk

Thesis Components

1. Understanding and optimize ML parallelization with adaptive parallelisms
2. Representation and Composability for ML parallelisms
3. Automating ML parallelization

Thesis-related Work

- Communication: Poseidon [ATC'17]
  - Consistency Model: Impact of staleness [ICLR'19]
  - Scheduling: Poseidon [ATC'16]
  - Memory Management: GeePS [Eurosys'16]

- Representations for dynamic batching: Derive
  - DyNet: Cavs [ATC'18]

- Representations for distributed parallelism: Derive
  - AutoDist: AutoDist [Preprint]

- Distributed Strategy Auto-optimization: AutoLoss [ICLR'19]
  - AutoSync [preprint]
## The Rest of Today’s Talk

### Thesis Components

- Understanding and optimize ML parallelization with *adaptive parallelisms*
- Representation and Composability for ML parallelisms
- Automating ML parallelization

### Thesis-related Work

<table>
<thead>
<tr>
<th>Communication</th>
<th>Consistency Model</th>
<th>Scheduling</th>
<th>Memory Management</th>
</tr>
</thead>
</table>

- **Representations for dynamic batching**
  - Derive
  - DyNet

- **Representations for distributed parallelism**
  - Derive
  - AutoDist

- **Distributed Strategy Auto-optimization**
  - AutoLoss [ICLR’19]
  - AutoSync [preprint]
Parameter Server

PS collects, aggregates, and applies the gradients, and then broadcast the parameters back to workers.

\[ \theta(t+1) = \theta(t) + \varepsilon \sum_{p=1}^{P} \nabla_{L}(\theta(t), D_p^{(t)}) \]

In total P workers
Partition data
Replicated and happening locally on each worker
Problem: Server Bottleneck

Figure from Krizhevsky et al. 2012
P2P Broadcasting?

• Idea: Send lightweight SF updates \((u,v)\) instead of expensive full-matrix \(\Delta W\) updates – sufficient factor broadcasting

\[
\min_W \frac{1}{N} \sum_{i=1}^{N} f_i(Wa_i; b_i) + h(W)
\]

\[
\Delta W = uv^T \quad u = \frac{\partial f(Wa_i, b_i)}{\partial(Wa_i)} \quad v = a_i
\]

Full parameter matrix update \(\Delta W\) can be computed as outer product of two vectors \(uv^T\) (called sufficient factors)
Problem: Quadratic Overhead

<table>
<thead>
<tr>
<th></th>
<th>Size of one message</th>
<th>Number of messages</th>
<th>Network Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2P SV-Transfer</td>
<td>$O(J + K)$</td>
<td>$O(P^2)$</td>
<td>$O((J + K)P^2)$</td>
</tr>
<tr>
<td>Parameter Server</td>
<td>$O(JK)$</td>
<td>$O(P)$</td>
<td>$O(JKP)$</td>
</tr>
</tbody>
</table>

AlexNet

# neurons in layer fc6=4096

# neurons in layer fc7 =4096
Ring AllReduce

Figure from Sergeev et al. 2018
Problem: Sparse Gradients

Sparse Gradients in NLP Models

AllReduce

Figure from Kim et al. 2018
Adaptive Communication

Figure from Krizhevsky et al. 2012

11 x 11 x 3 = 363 floats
Adaptive Communication

- Send lightweight SF updates \((u, v)\) instead of expensive full-matrix

\[
\min_{W} \frac{1}{N} \sum_{i=1}^{N} f_i(Wa_i; b_i) + h(W)
\]

2x – 8x throughput scalability improvement on a variety of CNNs and clusters configurations with limited bandwidth (1 – 40Gbps).

Figure from Krizhevsky et al. 2012
Adaptive Communication

Sparse Gradients in NLP Models

Is all reduce always better?

One batch

Embedding size

Vocabulary Size

Figure from Kim et al. 2018
Adaptive Communication: Parallax

Sparse Gradients in NLP Models

Is all reduce always better?

2x - 4x throughput scalability improvement on many models with large embeddings.

Vocabulary Size

Embedding size

Figure from Kim et al. 2018
Part I: Main Messages

• Strategies that adapt to specific model properties or resource configurations yield improved performance.
• Adaptiveness might be exploited in many aspects of distributed ML.

Instantiations of adaptive parallelization strategies:

- Communication
- Consistency Model
- Scheduling
- Memory management
...
The Rest of Today’s Talk

Thesis Components

- Understanding and optimize ML parallelization with *adaptive parallelisms*
- Representation and Composability for ML parallelisms
- Automating ML parallelization

Thesis-related Work

- **Communication**
  - Poseidon [ATC’17]
- **Consistency Model**
  - Impact of staleness [ICLR’19]
- **Scheduling**
  - Poseidon [ATC’16]
- **Memory Management**
  - GeePS [Eurosys’16]

- **Representations for dynamic batching**
  - Derive
- **DyNet**
- **Cavs**
  - Cavs [ATC’18]
  - Cavs [ATC’18]

- **Representations for distributed parallelism**
  - Derive
- **AutoDist**
- **AutoLoss** [ICLR’19]

- **Distributed Strategy Auto-optimization**
  - Derive
- **AutoSync** [preprint]

- **Distributed Strategy Auto-optimization**
  - Derive
- **AutoSync** [preprint]
Goal

- Communication Architecture
  - Parameter Server
  - P2P
  - Partial Broadcasting
- Consistency Models
  - Synchronous
  - Asynchronous
  - Stale Synchronous
- Partitioning/Placement
  - Graph Partitioning
  - Operator
  - Device Placement
- Encoding/Decoding
  - Gradient Quantization
  - Compression
- Scheduling
  - WFBP
  - Pipeline Parallelism
  - Byte Scheduler

Strategy auto-generation

Representation of ML parallelisms

Understanding and unifying various ML parallelisms
An Optimization Perspective

Maximize Performance (Model & Configurations, Cluster Topology & Settings)

System throughput/convergence/cloud cost

Dataflow graphs
Strategy Representation: Overview

- **Communication Architecture**
  - Parameter Server
  - AllReduce/Collective
  - P2P
  - Partial Broadcasting

- **Consistency Models**
  - Synchronous
  - Asynchronous
  - Stale Synchronous

- **Partitioning/Placement**
  - Graph Partitioning
  - Operator Partitioning
  - Device Placement

- **Encoding/Decoding**
  - Gradient Quantization
  - Deep Compression

- **Scheduling**
  - WFBP
  - Pipeline Parallelism
  - Byte Scheduler
Strategy Representation: Overview

- Idea: dataflow graph rewriting

Dataflow graph rewriting w/ distributed semantics
Strategy Representation: Overview

Assembled from multiple base aspects

- Graph Partitioning
- Operator Placement
- Graph Replication
- Device Placement
- Parameter Server
- AllReduce
- SFB
- Decentralized
- Sync/Async/SSP
- Gradient Quantization
- Deep Compression
- WFBP
- Pipeline Parallelism
- Byte Scheduler

Partitioning & Replication | Placement | Synchronization | Scheduling
Semantic: Partitioning and Placement

Node Partitioning

Placement

Figure from Wang et al.

Shazeer et al. 2018
Jia, et al. 2019
Wang, et al. 2019
Harlap, etc.

Mirhoseini et al. 2017
Semantic: Replication

```
"graph": {
"replica": {
"10.1.0.1",
"10.1.0.2",
"10.1.0.3"
}
}
```
Semantic: Synchronization

```
'synchronizer': {
    'scheme': 'PS',
    'config': {
        'reduction_destinations': {
            '10.1.0.1:CPU',
            '10.1.0.2:CPU'
        },
        'local_replication': True,
        'staleness': '<3',
        'compressor': 'PowerSGD'
    }
}

'synchronizer': {
    'scheme': 'AllReduce',
    'config': {
        'algo': 'ring',
        'staleness': '0',
        'compressor': '1bit'
    }
}
```
Examples
Architecture

Strategy Builder A
Strategy Builder B
AutoDist
Coordination/Runtime

Coordination/Runtime

User Interface
StrategyBuilder
Representation

Compilation & Strategy
Partitioner
Device Setter
Synchronizer
Replicator
Encoder

“Kernels”

Coordination
Coordinator & Runner

Frameworks
TensorFlow / PyTorch
Contrast: Workflows

- Strategy Builder A
- Strategy Builder B
- Coordination/Runtime

AutoDist

Horovod

Bosen

BytePS

Poseidon
Usage: Strategy

Strategy: model- and resource-dependent expression instructing how to distribute the model.

Cluster spec $R$

Model graph $G$

graph:
  replica:
  - 10.1.0.1
  - 10.1.0.2
  - 10.1.0.3
nodes:
  Synchronizer:
    type: HierarchicalParameterServer
    w:
    - 10.1.0.1:CPU0
    - 10.1.0.2:CPU0
    b:
    - 10.1.0.1:CPU0
  Partitioner:
    mul: [0, 2, 0]
  DeviceSetter:
    mul:
    - 10.1.0.1:GPU0
    - 10.1.0.2:CPU0
    add:
    - 10.1.0.1:GPU2

Strategy $S$
Usage: Examples

```
import AutoDist
import AutoDist.strategy import mystrategy

filepath = os.path.join(os.path.dirname(__file__), 'resource_spec.yml')
autodist = AutoDist(resource_spec_file=filepath, strategy_builder=mystrategy)

with autodist.scope():
    train_dataset = build_dataset()
    train_iterator = tf.compat.v1.data.make_one_shot_iterator(train_dataset).get_next()
    model = build_model()
    loss_fn = tf.keras.losses.SparseCategoricalCrossentropy()
    optimizer = tf.keras.optimizers.SGD()

def train_step(inputs):
    x, y = inputs
    y_hat = model(x, training=True)
    loss = loss_fn(y, y_hat)
    all_vars = []
    for v in model.trainable_variables:
        all_vars.append(v)
    grads = tf.gradients(loss, all_vars)
    update = optimizer.apply_gradients(zip(grads, all_vars))
    return loss, update

fetches = train_step(train_iterator)
for _ in range(min(10, len(train_images)) // BATCH_SIZE * EPOCHS):
    loss, _ = sess.run(fetches)
    print(f"train_loss: {loss}"
```

Choose strategy and construct AutoDist object

Scope your code

Use AutoDist distributed session instead of native session

Hao Zhang
Usage: Strategy Builder

Strategy builder: a strategy generation function with expert knowledge.

```python
# Parallax strategy
# Heuristics:
# use PS for sparse variables,
# use AllReduce for dense variables
for grad in grads:
    if isinstance(grad, tf.Tensor):
        strategy[grad]['syncer'] = 'AllReduce'
    if isinstance(grad, tf.SparseTensor):
        strategy[grad]['syncer'] = 'PS'
```

```
# Poseidon strategy
# Heuristics:
# Reduce the grad of FC layers to vectors,
# and then broadcast vectors
for grad in grads:
    if np.linalg.matrix_rank(grad) == 1:
        strategy[grad]['syncer'] = 'AllReduce'
        strategy[grad]['compressor'] = 'SFB'
    else:
        strategy[grad]['syncer'] = 'PS'
        strategy[grad]['compressor'] = 'None'
```

Prebuilt strategy builders report *matched* or *slightly better* performance compared to their corresponded specialized systems (TF runtime).
The Rest of Today’s Talk

Thesis Components

1. Understanding and optimize ML parallelization with adaptive parallelisms
2. Representation and Composability for ML parallelisms
3. Automating ML parallelization

Thesis-related Work

- Communication
  - Poseidon [ATC'17]
- Consistency Model
  - Impact of staleness [ICLR'19]
- Scheduling
  - Poseidon [ATC'16]
- Memory Management
  - GeePS [Eurosys'16]

- Representations for dynamic batching
  - Derive
  - DyNet
- Model
  - Derive
  - AutoDist [Preprint]
- Memory Management
  - AutoDist [Preprint]
- Distributed Strategy
  - Auto-loss [ICLR'19]
  - AutoSync [preprint]
Usage: Strategy Builder

Strategy builder: a strategy generation function with expert knowledge.

Model graph $G$

Resource spec $R$

StrategyBuilder

- PSStrategyBuilder
- AllReduceBuilder
- PoseidonBuilder
- ParallaxBuilder

...
Problem: Design Auto StrategyBuilders

Strategy builder: a strategy generation function auto-optimizes over the strategy space.
An Optimization Perspective

Maximize *Performance* (Model & Configurations, Cluster Topology & Settings)

System throughput/convergence/cloud cost

Dataflow graphs

```
graph:
  replica:
  - 10.1.0.1
  - 10.1.0.2
  - 10.1.0.3
nodes:
  Synchronizer:
    type: HierarchicalParameterServer
    ws:
    - 10.1.0.1:CPU0
    - 10.1.0.2:CPU0
    bs:
    - 10.1.0.1:CPU0
  Partitioner:
    mlp: [0, 2, 0]
  DeviceSetter:
    mlp:
    - 10.1.0.1:GPU0
    - 10.1.0.1:GPU1
    add:
    - 10.1.0.1:GPU2
```
Maximize \( \text{Performance}(\text{Model & Configurations, Cluster Topology & Settings}) \)

\[
\max_s t(D, R, s) \quad \text{s.t. } s \in S_{\text{sync}}
\]
Solving the Problem

\[
\max_{s} t(D, R, s) \\
\text{s.t. } s \in S_{\text{sync}}
\]

- Non-continuity
- Measuring t is not cheap
- Transparent communication
- Domain-specific prior knowledge
Optimizer

Strategy proposals

Predefined features
Explicit modeling
Raw features
ML based simulator

Resource $R$
Model $D$

Knowledge guided sampler

High-scored proposals

Select $s^*$

If budget remains

Measure $t$
Communication Modeling: Features

Raw features:

\[
\begin{align*}
  m_i & \quad C_i \\
  \mathbb{I}_p & \quad \mathbb{I}_d(j, k) & \quad b_{j,k} & \quad T_{i,k}^{IP}
\end{align*}
\]

Predefined features:

\[
\begin{align*}
  T_{\text{service}}(v_i) & = \sum_{k=1}^{n_d} I_d(j, k) \cdot \frac{c_i(m_i)}{b_{j,k}} \cdot r_{j,k}^p + \sum_{k=1}^{n_d} I_d(j, k) \cdot r_{j,k}^h \cdot \phi + \gamma,
  \quad \text{GPU kernel latency}
  \\
  T_{\text{sync}}^{\text{CG}}(v_i) & = \frac{2(w_i - 1)c_i(m_i)}{w_i b_m} + \frac{(w_i - 1)c_i(m_i)}{w_i b_m} + \frac{c_i(m_i)}{b_m} + w_i \cdot \phi + \delta,
  \quad \text{network transfer}
\end{align*}
\]

For each variable

[ variable size, compression type, reduction structure, placement, bandwidth, #replicas ]

[ network transfer, network overhead, GPU memory kernel latency ]
Find strategies 1.1 – 1.6x faster than specialized systems, such as BytePS, Horovod, or handwritten strategy builders, on CNNs, transformers, RNNs, and recommendation models (TF runtime).

Budget: 200 trial runs = 200 * 50 = 10K warmup iters

Trained simulators exhibit certain transferability.
Transfer Trained Simulators

Case #1

Infer

Resource $R$
Smaller model $D$

Train

Resource $R$
Bigger model

Smaller cluster model $D$

Case #2

Infer

Bigger cluster model $D$

Train

Resource $R$
Bigger model

Smaller model $D$

Hao Zhang

54
Transfer Trained Simulators

**Target: VGG16, B**

- Improvement (x) vs. # Trials
- Horovod
- PS
- From (VGG16, A)
- AutoSync(-s)
- AutoSync

**Target: BERT-base, A**

- Improvement (x) vs. # Trials
- Horovod
- PS
- From (BERT-3L, A)
- AutoSync(-s)

**Target: BERT-base, B**

- Improvement (x) vs. # Trials
- Horovod
- PS
- AutoSync(-s)
- From (BERT-3L, A)
- From (BERT-base, A)
- From (BERT-3L, B)
The Rest of Today’s Talk

Research Roadmap

- Understanding Individual aspects and optimizations with adaptive parallelisms
- Unified Representation of aspects of ML parallelisms
- Auto-optimization for Distributed ML

Thesis-related Work

<table>
<thead>
<tr>
<th>Communication</th>
<th>Consistency Model</th>
<th>Scheduling</th>
<th>Memory Management</th>
<th>Resource Allocation</th>
</tr>
</thead>
</table>

- Representations for distributed parallelism
- Derive AutoDist [Preprint]
- Distributed Strategy Auto-optimization
- Representations for dynamic batching
- Derive DyNet
- Auto-optimize for Distributed ML

- AutoLoss [ICLR’19]
- AutoSync [preprint]
- Cavs [ATC’18]
- Cavs [ATC’18]
New Takeaways (after Thesis Proposal)

• Offer a rich set of de facto distributed strategies in one system
  • Performance matches corresponded specialized systems
• Unified and simpler interface for distributed ML
• Develop new distributed strategies using Python
• Offer preliminary strategy auto-optimization
  • Performance 1.1x – 1.6x faster than specialized systems
• It was open sourced: https://github.com/petuum/autodist (please try and start)
Summary: Distributed ML Training

Eager, prototyping, development

- **Python**
- **PyTorch**
- **JAX**

De facto distributed strategy (a.k.a. strategy builders)

Cluster/Resource scheduling, coordination, management, etc.

Declarative, massive training, production

- **Python**
- **TensorFlow**
- **TORCHSCRIPT**

Model/Program IR (active WIP)

Composable parallelism and strategy auto-optimization
Acknowledgement

Advisor

Committee members

Collaborators

Collaborators (6 more!)
Thank you
Architecture: “Kernels”

Kernels are implementations of atomic graph transformation operations.

Expression of a PS synchronizer (generated by strategy builder)

Implementations of transformations
Nowadays ML Frameworks

- **High-level Language APIs**: ML prototyping is made easier
- **Dataflow graph representation**: Composable and powerful representations for ML workloads, allows for various optimizations
- **Multi-layer architecture**: APIs $\rightarrow$ graph $\rightarrow$ runtime $\rightarrow$ kernel $\rightarrow$ device
Example: $y = Wx + b$
Distribution: Apply a Strategy to Transform the Graph

graph:
  replica:
  - 10.1.0.1
  - 10.1.0.2
  - 10.1.0.3

nodes:
  Synchronizer:
    type: HierarchicalParameterServer
  w:
    - 10.1.0.1:CPU0
    - 10.1.0.2:CPU0
  b:
    - 10.1.0.1:CPU0

Partitioner:
  mul: [0, 2, 0]

DeviceSetter:
  mul:
    - 10.1.0.1:GPU0
    - 10.1.0.1:GPU1
  add:
    - 10.1.0.1:GPU2
Distribution: Apply a Strategy to Transform the Graph

```yaml
graph:
  replica:
    - 10.1.0.1
    - 10.1.0.2
    - 10.1.0.3

nodes:
  Synchronizer:
    type: HierarchicalParameterServer
    w:
      - 10.1.0.1:CPU0
      - 10.1.0.2:CPU0
    b:
      - 10.1.0.1:CPU0
  Partitioner:
    mul: [0, 2, 0]
  DeviceSetter:
    mul:
      - 10.1.0.1:GPU0
      - 10.1.0.1:GPU1
    add:
      - 10.1.0.1:GPU2
```

Execute

Place onto corresponded devices
Other Work

• Scalable and Auto ML
  • AutoLoss [Zhang, et al. ICLR’19]

• Large-scale Applications
  • Large-scale topic models on twitter data [Zhang, KDD’15]
  • HD-CNN [ICCV’15]

• ML on/with structures
  • Structured GANs [Zhang et al. Neurips’18]
  • Symbolic Graph Reasoning [Neurips’18]
  • Semantic Manipulation [ECCV’18]
  • Visual paragraph generation [ICCV’17]
  • Learning concept taxonomies [ACL’16]