Paper Presentation

Overview

One or a few slides introduce the high-level design and ideas of the work

Problem & Challenges

- What problem does this paper try to solve?
- What are the challenges we must address to solve the problem?

Methods & Contributions

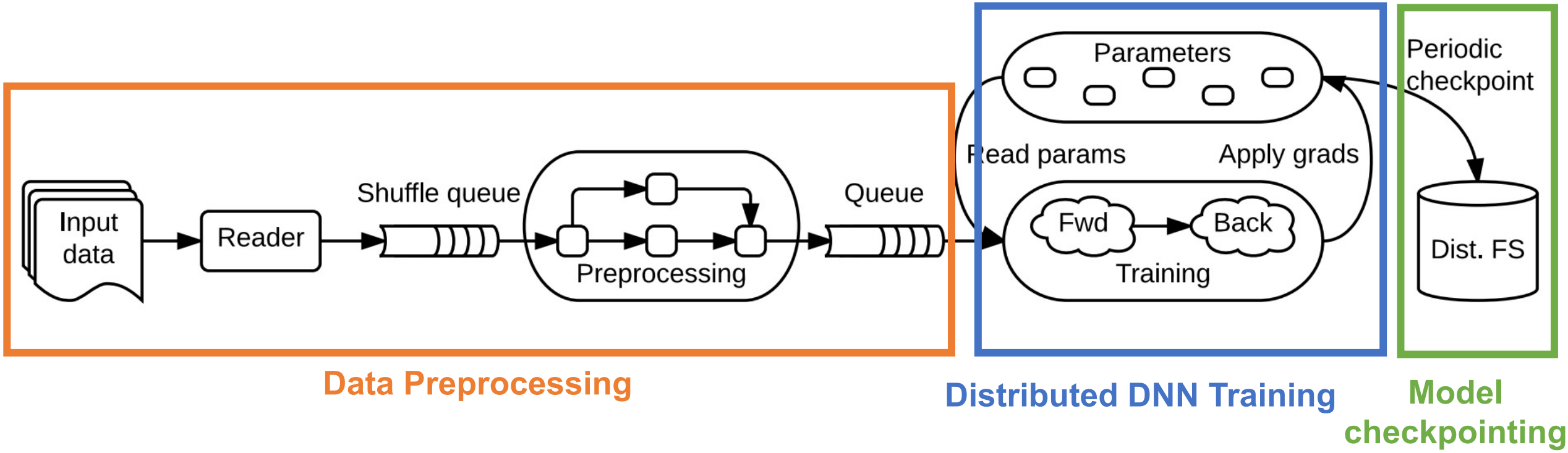
- What are the key techniques proposed in the paper?
- How do these techniques address the challenges?

Evaluation & Discussion

- How does this work compare against prior work?
- Your own thoughts such as its limitations and potential improvements

TensorFlow: A System for Large-Scale Machine Learning

Overview: TensorFlow is a System for Data-Parallel DNN Training



Problem: Large-Scale Machine Learning on Distributed Heterogeneous Devices

Challenges:

- New operators are continuously introduced, requiring an efficient interface for implementing new operators (especially by ML researchers)
- Support new training algorithms that don't follow the forward-backward-update patterns
  - Recurrent NNs (run forward/backward passes multiple times)
  - Adversarial DNNs (train two DNNs alternatively)
  - Reinforcement learning (loss function is computed by another system)
- How to support new optimization methods, such as variants of SGD?

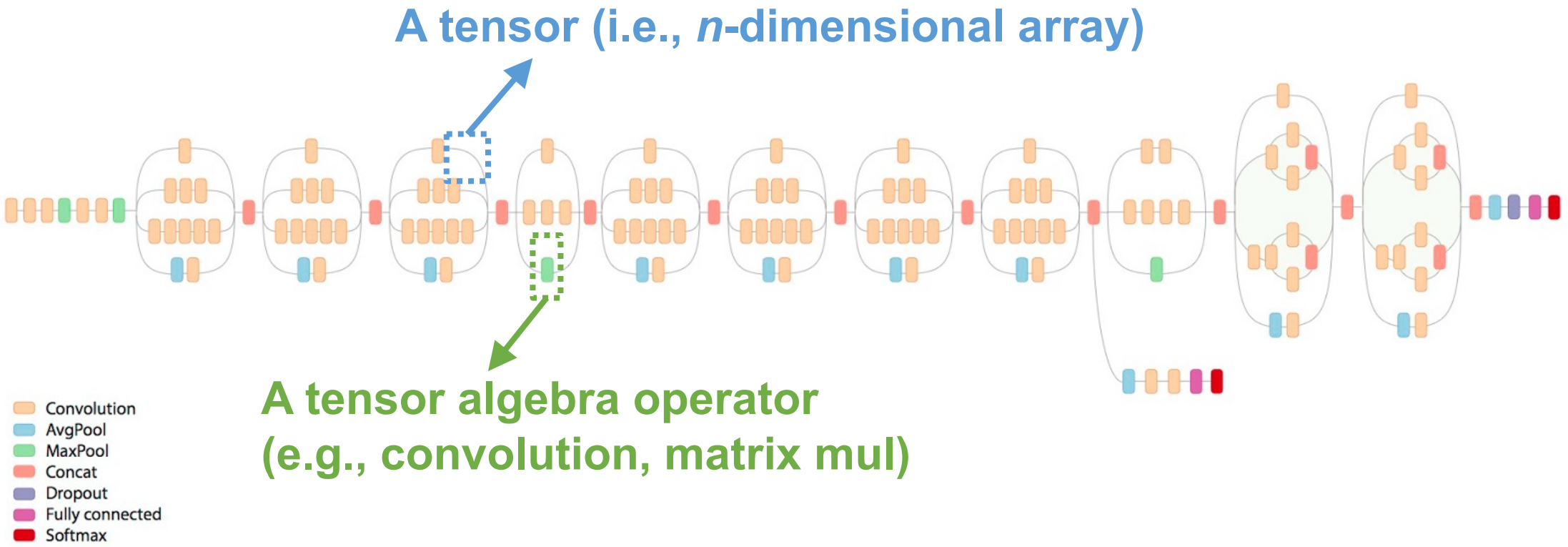
Methods: How TensorFlow Address these Challenges?

Techniques:

1. Dataflow graph to represent ML computation
2. Deferred execution to optimize performance
3. Dynamic control flow to support model dynamism
4. Synchronous training with backup workers for distributed training

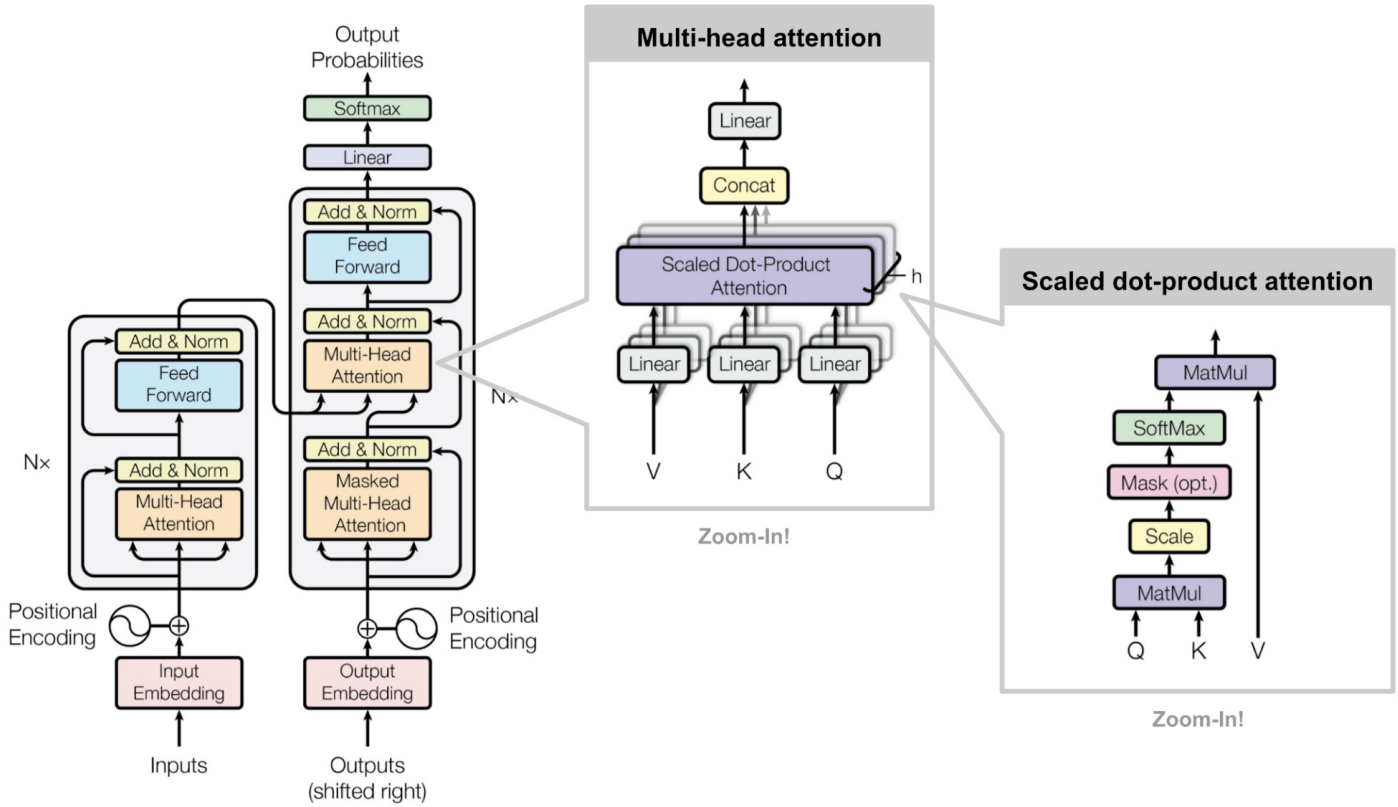
Technique #1: Dataflow Graphs with Primitive Operators

- Uses a directed acyclic graph (DAG) of primitive mathematical operators to represent the computation of an ML model



Technique #1: Dataflow Graphs with Primitive Operators

- Allow users to represent high-level and complex layers using compositions of primitive operators
- E.g., multi-head attention can be represented as additions, multiplications, and activations



Technique #1: Dataflow Graphs of Primitive Operators

- Allow users to represent high-level and complex layers using compositions of primitive operators
- E.g., multi-head attention can be represented as additions, multiplications, and activations

Benefits:

- Easy to support new ML layers, no need to implement new accelerator kernels for most new ML layers
- Easy to automatically differentiate ML models

Technique #2: Deferred Execution

Phase 1: Define an ML model as a dataflow 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)
```

Phase 2: Execute an optimized version of the graph

```
# 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.
```

Technique #2: Deferred Execution

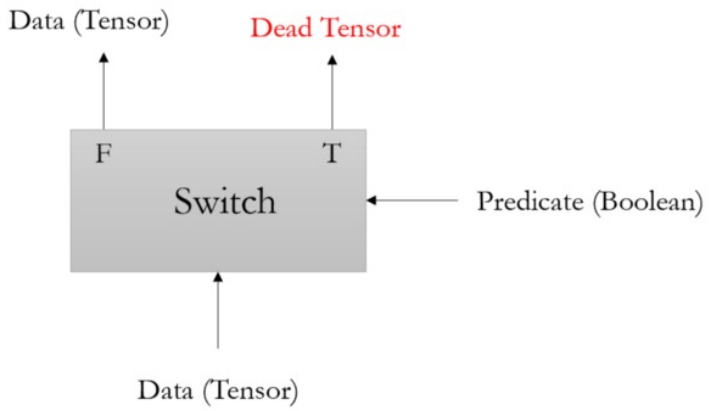
- Phase 1: define an ML model as a dataflow graph
- Phase 2: execute an optimized version of the graph

Benefits:

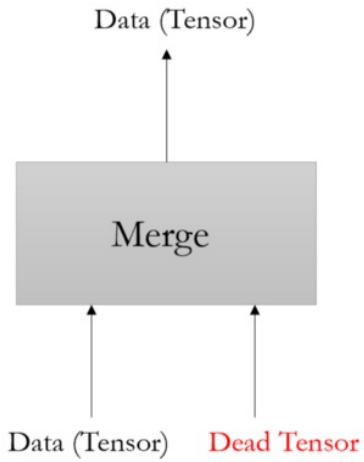
- Global optimizations: use global information to optimize the execution
  - which operators should run in parallel
  - how to order tasks on each device
- Device-level optimizations: launch a sequence of kernels on accelerators to hide kernel launch overheads by using the graph's dependency

Technique #3: Dynamic Control Flow

- **Key idea**: support algorithms with **dynamic** control flow in a **static** computation graph
- Switch: use a control input to select which output to produce a value
- Merge: forward at most one non-dead input to its output



Switch receives two arguments: Data and predicate (boolean), and has two outputs: data and dead Tensor



Merge receives two arguments: Data and dead Tensor, and has one outputs: data

*Figure from Hosein Fooladi

Technique #3: Dynamic Control Flow

Combine switch and merge to support dynamic control flow

E.g., if x < y return x + z; else return y^2

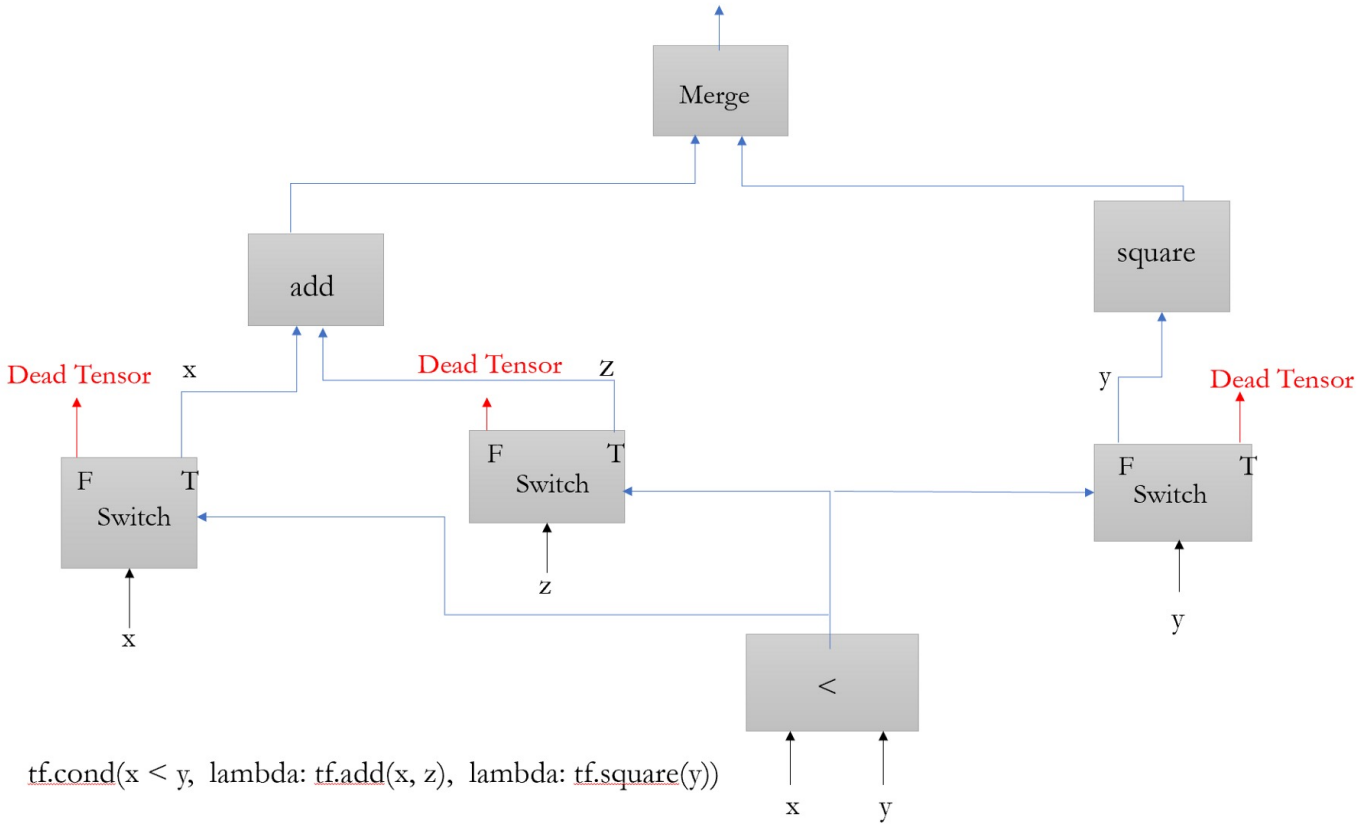


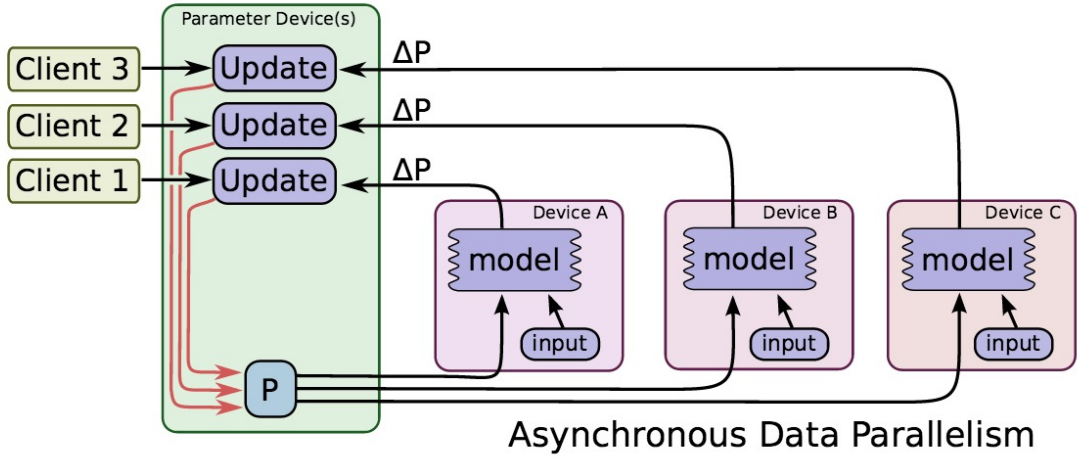
*Figure from Hosein Fooladi

Technique #3: Dynamic Control Flow

Benefits of supporting **dynamic** control flow in a **static** computation graph

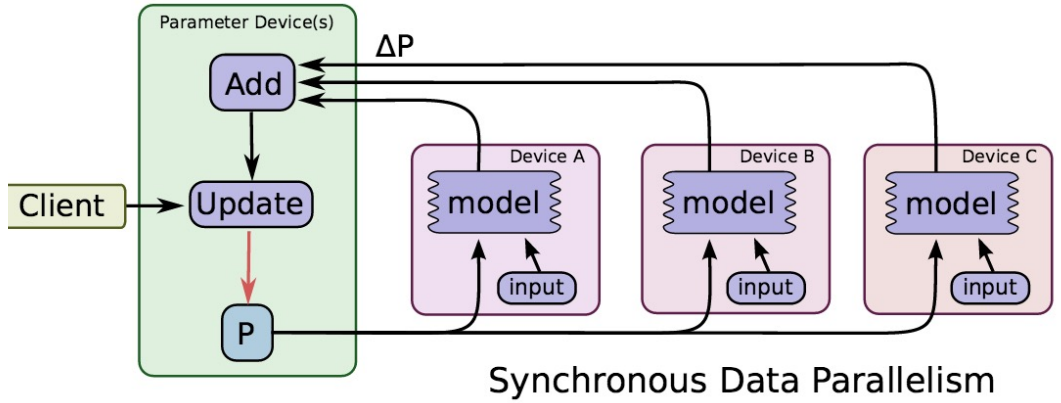
- Statically optimize a computation graph using global information
- Easy to identify which operators can be run in parallel
- Enable critical ***static*** optimizations such as constant folding, common subexpression elimination

Synchronous v.s. Asynchronous Data-Parallel Training



Asynchronous Data Parallelism

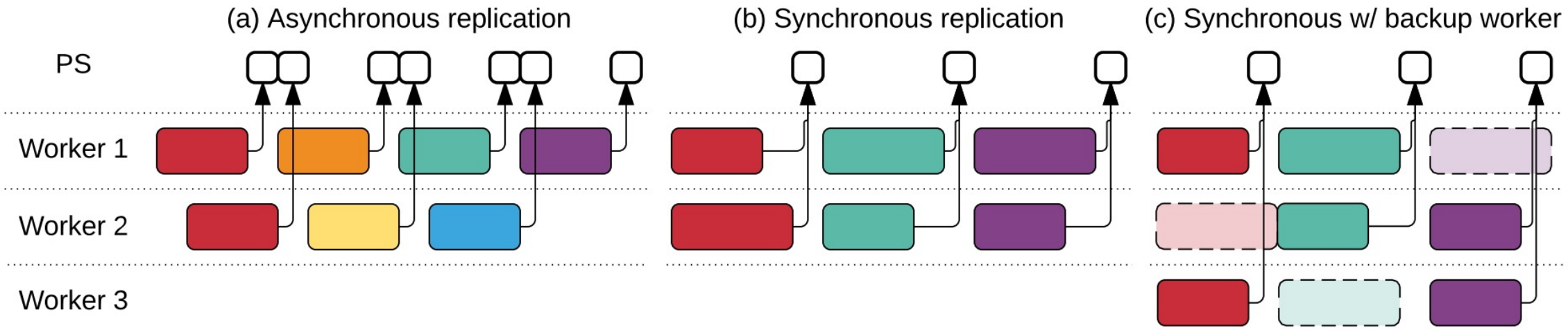
- Pro: non-blocking forward/backward passes
- Con: inconsistency between parameters and gradients on parameter servers (weight staleness)



Synchronous Data Parallelism

- Pro: statistically more efficient as there is no weight staleness
- Con: need to wait for outliers; low runtime performance

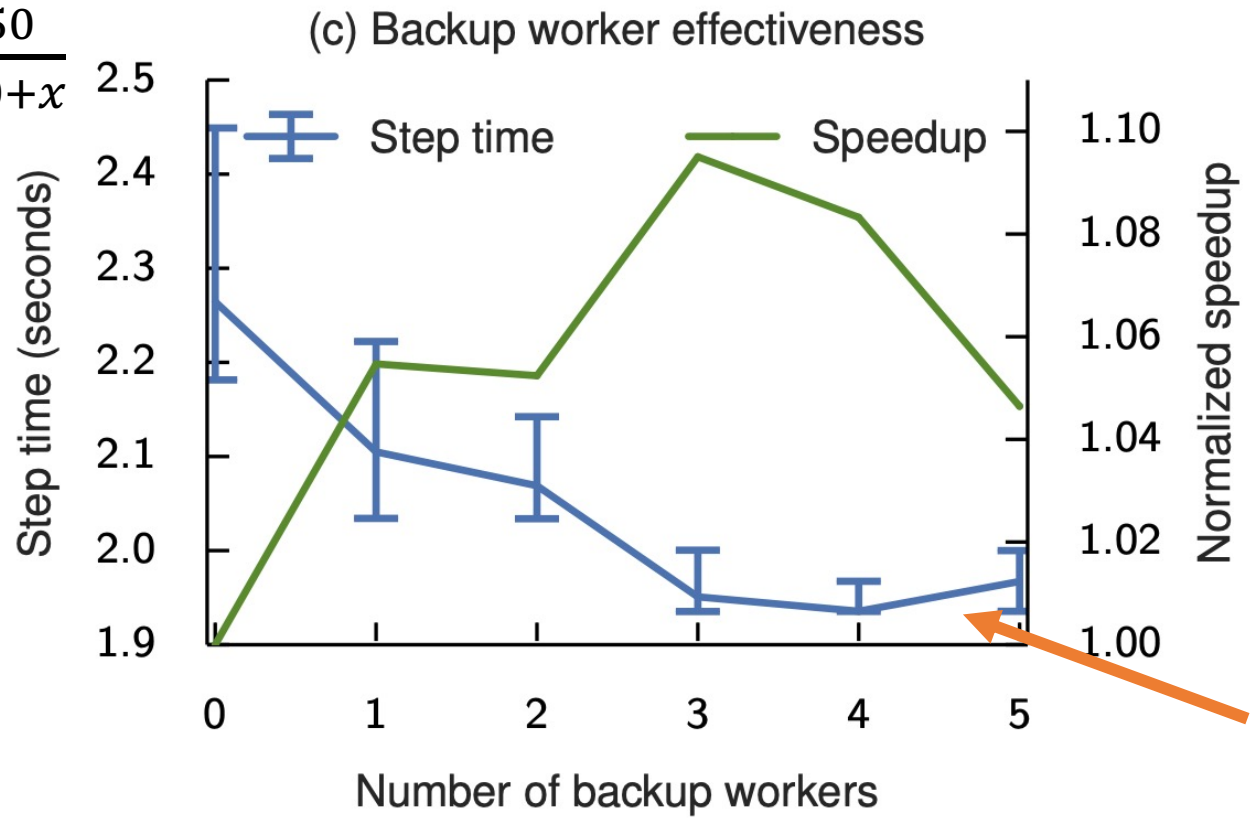
Technique #4: Synchronous Training with Backup Workers



- If we need gradients from m workers, launch n (n > m) workers and accept gradients from the first **m** of **n** workers.
- Leverage an ML property: SGD samples training data randomly at each step

Technique #4: Synchronous Training with Backup Workers

- 50 regular workers + X backup workers
- Step time: how long it takes to complete a step
- $speedup = \frac{t(50+x)}{t(50)} \times \frac{50}{50+x}$



Why adding more backup workers decrease performance

Discussion

- How does TensorFlow compare to other data analytics systems from Google, such as MapReduce?
- MapReduce assumes immutable intermediate results (i.e., tensors in a computation graph cannot be changed once computed). How can we train DNNs in MapReduce? Why is this inefficient?
- MapReduce saves ***all*** intermediate results to storage for fault tolerance. How to support fault tolerance in TensorFlow? Should we save parameters, gradients, activations?
- Others…

Systems for Deep Learning

PyTorch: An Imperative Style, High-Performance Deep Learning Library

What is Pytorch

Answered by the title of the paper:

- Deep learning library
- Adopt imperative programming model
- High-performance

There are many other deep learning frameworks such as Tensorflow, Caffe, MXNet…

What makes Pytorch special?

Questions to address

1. Why do we even need a deep learning system like Pytorch?

2. What makes Pytorch different from other frameworks?

Discussion:

What are the limitations of Pytorch?

Why deep learning systems

Steps to train a two-layers network on MNIST without deep learning libraries:

1. **Define the hypothesis:** $$h_\theta = \theta_2{}^T ReLU\left(\theta_1{}^T X\right)$$

2. **Define loss function:** $$l_{cross-entropy}\left(h_\theta(x), y\right) = -\log\left(\frac{exp\left(h_y(x)\right)}{\sum_{j=1}^{k} exp\left(h_j(x)\right)}\right)$$

3. **Solve the optimization problem(SGD)** $$min_\theta \ \frac{1}{n}\sum_{i=1}^{n} l\left(h_\theta(x^i), y^i\right)$$

$$\theta := \theta - \frac{\eta}{b}\nabla_\theta l\left(h_\theta(x^i), y^i\right)$$

Problems

Calculating the gradient by hand is complicated and painful.

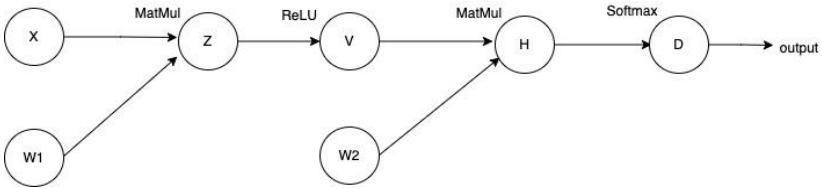
$$\theta := \theta - \frac{\eta}{b} \nabla_\theta l\left(h_\theta(x^i) , y^i \right)$$

Doable for a simple two-layers network, what about 10-layers or ResNet-50 involving convolutions , batch normalization, skip connections…

Deep learning is modular in nature. This code is ugly and not easy to maintain and extend.

Deep learning systems

Use dataflows to represent computations



Reverse AutoDiff to calculate the gradient in reverse topological order

Easy to modularize the code(build modules on top of ops)

Nodes represents tensors.

Also defines ops like MatMul, ScalarAdd, Reshape, Transpose… Each ops has a gradient method.

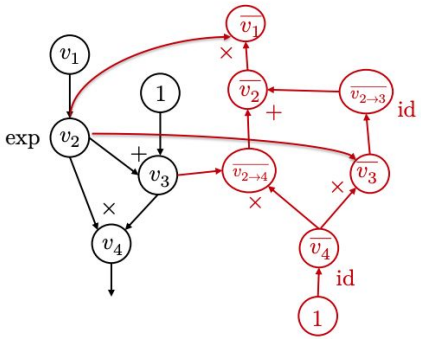
```
class ExpOp(Op):
    def __call__(self, a: Tensor) -> Tensor:
        pass

    def gradient(self, out_grad, node):
        return [exp(node.inputs[0]) * out_grad]
```

Note: the graph is simplified. Transpose ops are not shown.

Deep learning systems

Example of Reverse AD



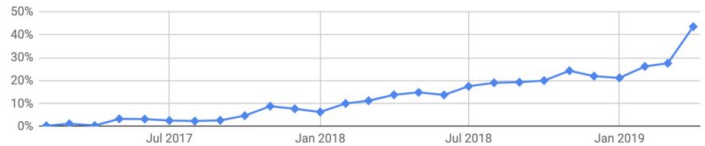
Need to compute the adjoint and partial adjoint while traversing the graph

Graph credit: slides of 10-714

Why Pytorch

There are already many great deep learning frameworks such as Tensorflow, Caffe, MXNet.. All of them adopts similar AD approach.

Tensorflow(Google) dominated in deep learning before Pytorch was released. Now, Pytorch supersedes Tensorflow..



Main contribution: Strike a good balance between ease-of-use and speed.

Dynamic execution: easy to use at cost of performance

TensorFlow 1.0

TensorFlow

PyTorch

Ease of Use

Pytorch is designed to be Pythonic

Simple and consistent interfaces

Not all interfaces of deep learning libraries are simple….

tf.layers.conv2d tf.nn.conv2d tf.keras.layers.conv2d

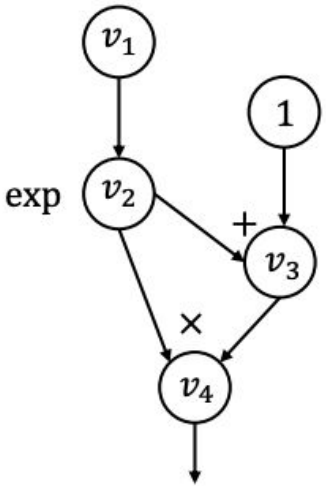
High level modular components(nn.Modules)

```
class LinearLayer(Module):
   def __init__(self, in_sz, out_sz):
      super().__init__()
      t1 = torch.randn(in_sz, out_sz)
      self.w = nn.Parameter(t1)
      t2 = torch.randn(out_sz)
      self.b = nn.Parameter(t2)

   def forward(self, activations):
      t = torch.mm(activations, self.w)
      return t + self.b
```

Imperative Programming Model

Execute the computational graph while constructing it




```
import torch

v1 = torch.Tensor([1,2,3])
v2 = torch.exp(v1)
v3 = v2 + 1
V4 = v2 * v3
```

Benefits of imperative model?

Tensorflow adopts **declarative programming model**:

constructing the graph first, then execute the graph by calling sess.run()

Imperative Programming Model

Easy to debug

print the tensor to get the values or shapes

```
import torch

v1 = torch.Tensor([1,2,3])
v2 = torch.exp(v1)
v3 = v2 + 1
print(v3)
V4 = v2 * v3
```

Print only works when executing the graph in static computational graphs, which makes debugging less flexible.

Imperative Programming Model

Mutable tensors

- Update the value of the tensor in place(e.g. update the weights during backpropagation)

- Has to create a **assign** node in static computational graph(Tensorflow). Programmers has to define the dependencies(tf.control_dependencies)

Imperative Programming Model

Control flow

- Able to use mix of Python control flow and Pytorch Ops to construct the graph

```
if  v2.numpy() > 1:
      v3 = v2 * 2
else:
      v3 = v2 * 0.5
```

- Not easy to achieve in Tensorflow (**tf.where**)

High-Performance

- Efficient C++ core
  - Python as a host language
  - Data structures and ops are implemented in C++ and CUDA
- Multiprocessing
  - Extend Python multiprocessing to torch.multiprocessing
- Caching tensor allocator
  - Incrementally builds up a cache of CUDA memory and reassigns it to later allocations without further use of CUDA APIs
  - Thread-caching malloc

Benchmarks

PyTorch is within 17% of that of of the fastest framework

Framework	*Throughput (higher is better)*					
	AlexNet	VGG-19	ResNet-50	MobileNet	GNMTv2	NCF
Chainer	778 ± 15	N/A	**219** ± 1	N/A	N/A	N/A
CNTK	845 ± 8	84 ± 3	210 ± 1	N/A	N/A	N/A
MXNet	**1554** ± 22	113 ± 1	218 ± 2	444 ± 2	N/A	N/A
PaddlePaddle	933 ± 123	112 ± 2	192 ± 4	**557** ± 24	N/A	N/A
TensorFlow	1422 ± 27	66 ± 2	200 ± 1	216 ± 15	$9631 \pm 1.3\%$	$4.8e6 \pm 2.9\%$
PyTorch	1547 ± 316	**119** ± 1	212 ± 2	463 ± 17	**15512** $\pm 4.8\%$	**5.4e6** $\pm 3.4\%$

Interesting observation: Pytorch has higher throughput than Tensorflow

2 Intel Xeon E5-2698 v4 CPUs and 1 NVIDIA Quadro GP100 GPU

Discussion

We have listed some advantages of Pytorch.

What are the disadvantages of imperative abstraction or what are the advantages of declarative abstraction?

Discussion

We have listed some advantages of Pytorch.

What are the disadvantages of imperative abstraction or what are the advantages of declarative abstraction?

Optimize the execution using global information

Scalability

Partition computations across devices