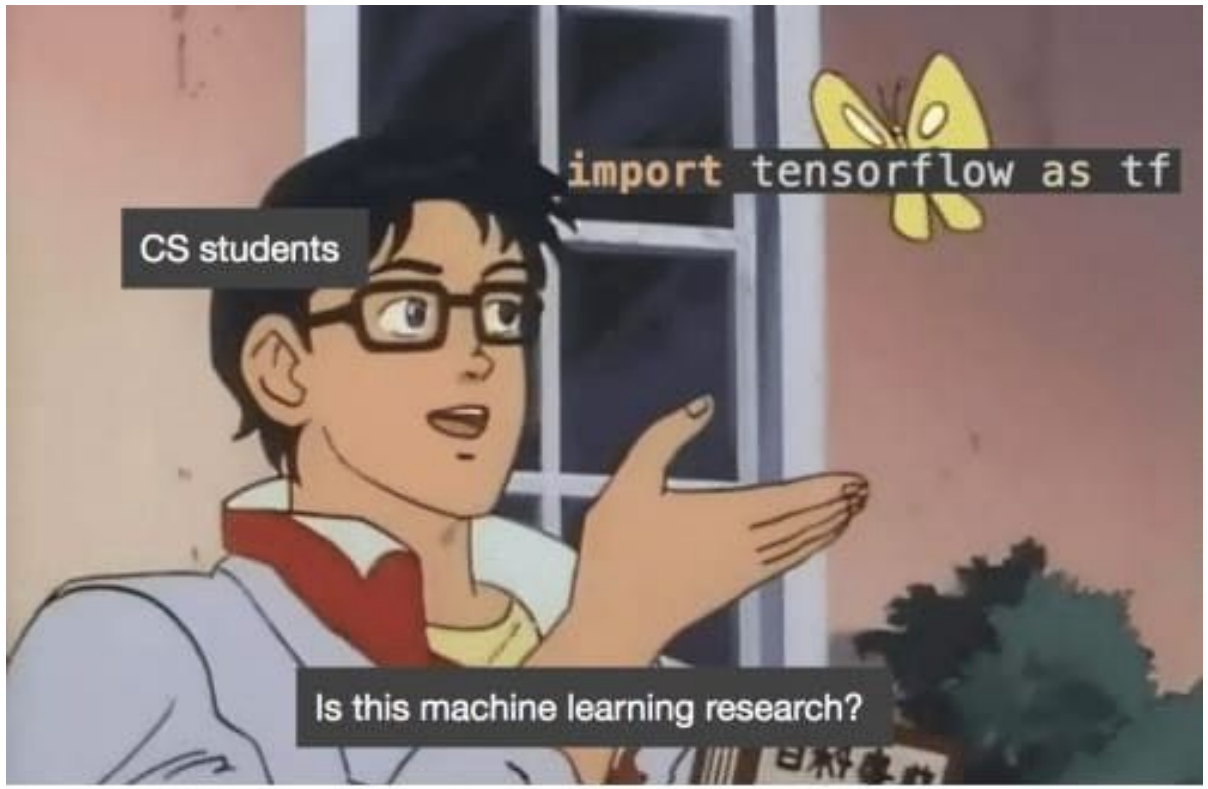
TensorFlow: A system for large-scale machine learning

Artem Abzaliev and Saisamrit Surbehera



Programmers Nowadays

How many of you have used Tensorflow or any other machine learning framework ?

Introduction of Frameworks

Machine learning has become successful for the following reasons:

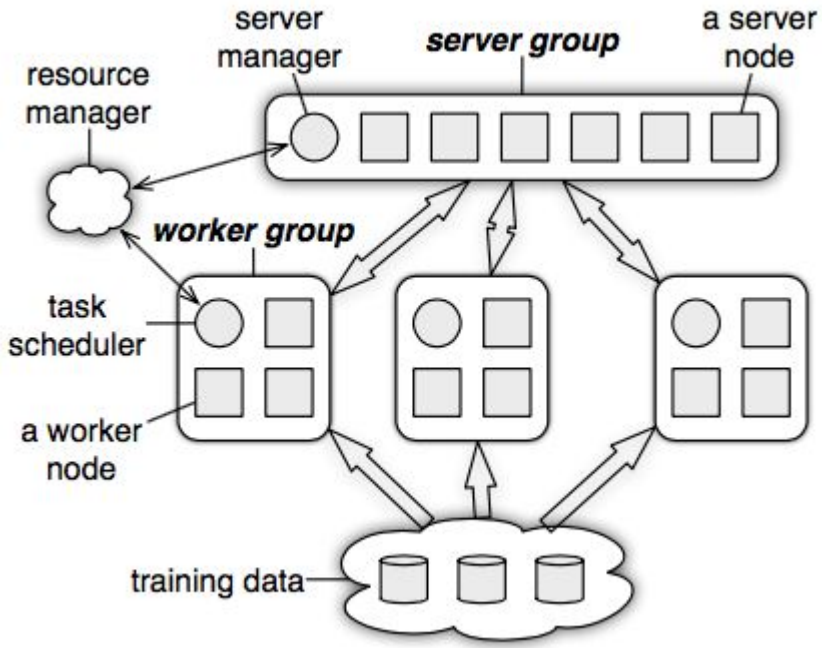
- the availability of large datasets for tackling problems in these fields
- the development of software platforms that enable the easy use of large amounts of computational resources for training such models on these large datasets
- more sophisticated machine learning models

Introduction of Frameworks

- Frameworks are interfaces, libraries, or tools, which are generally open-source that people with little to no knowledge of machine learning and AI can easily integrate.
- Scales machine learning code
- Computes gradients
- Standardized machine learning applications for sharing
- Large scale training and inference
- Machine Learning Zoo: Different Paradigms, Programming Languages, and abstractions
- Introduces interfaces for GPU to collaborate

Distbelief - Tf 0.0

- First Scalable system by Google.
- Parameter Service Architecture: a job comprises two disjoint sets of processes:
  - stateless worker processes
  - stateful parameter server
- Limitations
  - Distbelief graphs were implemented as C++ classes
  - Needs to modified for the new optimization methods i.e Adam
  - Building New advanced models:
    - Transformers, RI models and Adversarial models



https://medium.com/coinmonks/parameter-server-for-distributed-arning-fd79d99f84c3

Tensorflow introduction

- High-level programming models of dataflow systems + low-level efficiency of parameter servers

- Unified Dataflow graph to represent both the computation in an algorithm and the state on which the algorithm operates.

Design principles of Tensorflow

- Deferred Executions
  - 1st Step is to define the symbolic data-flow graph
  - 2nd step is to execute an optimized version of the program
- Dataflow of primitive operators
  - Tensorflow does math operations as nodes in the dataflow graph
- Abstraction for hetero heterogeneous accelrators
  - CPU, GPU, APIC (TPUs)
  - Each operator (e.g., matrix multiplication) can have multiple specialized implementations for different devices. As a result, the same program can easily target GPUs, TPUs, or mobile CPUs as required for training, serving, and offline inference

Design principles of Tensorflow

- TensorFlow as a set of tasks (named processes that can communicate over a network) that each export the same graph execution API and contain one or more devices.
- A subset of the tasks assumes the parameter server(PS) role and Worker tasks.
- PS task is capable of running arbitrary TensorFlow graphs, it is more flexible than a conventional parameter server.

TensorFlow Execution Model

TensorFlow uses a single dataflow graph to represent all computation and state in a machine learning algorithm, including the individual mathematical operations, the parameters and their update rules, and the input preprocessing.

The dataflow graph expresses the communication between sub-computations explicitly, thus making it easy to execute independent computations in parallel and to partition computations across multiple devices

How is TF different from batch dataflow system

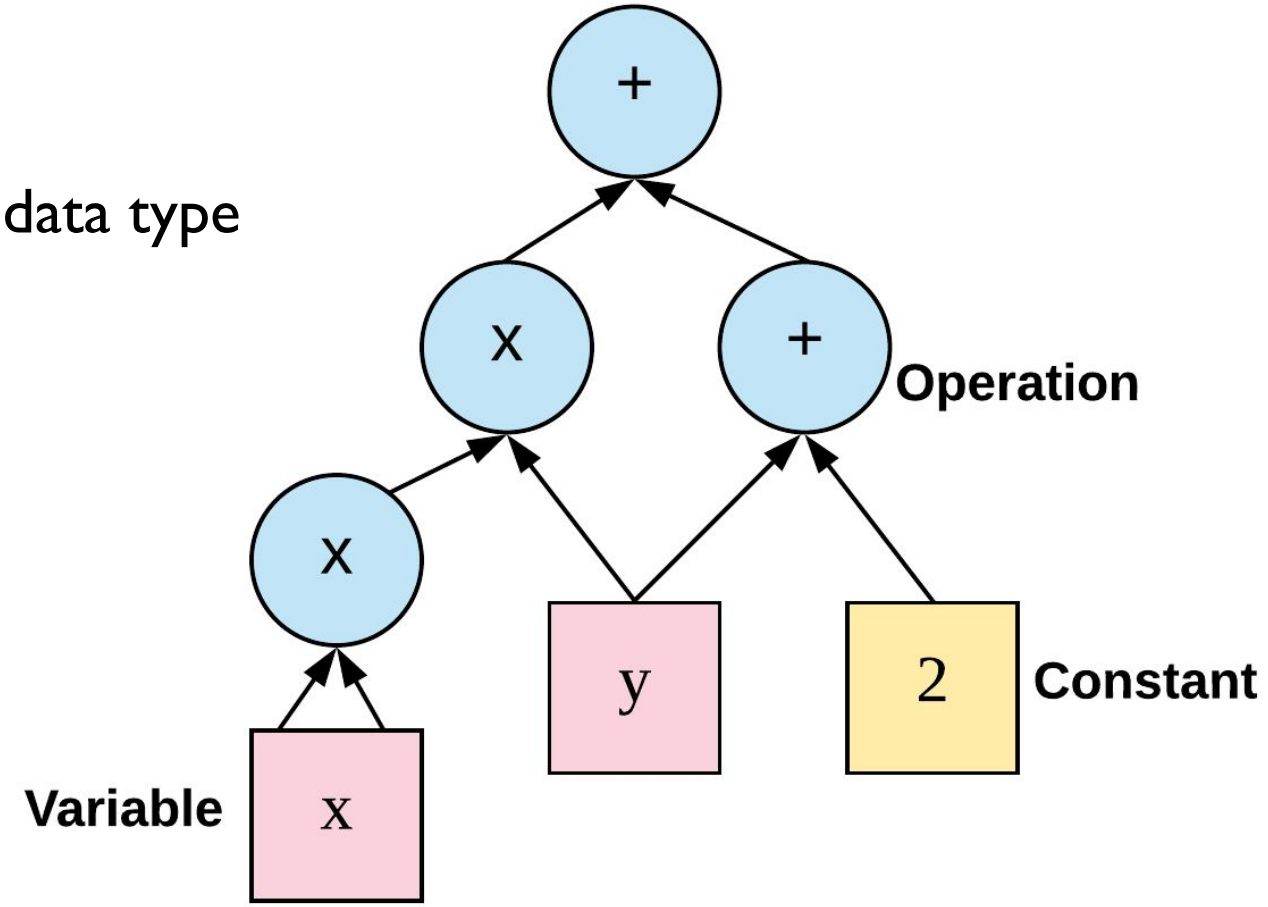
- The model supports multiple concurrent executions on overlapping subgraphs of the overall graph

- Individual vertices may have mutable state that can be shared between different executions of the graph.
  - This makes it easy to updates to very large parameters and propagate those updates to parallel training steps as quickly as possible
  - Arbitrary dataflow subgraphs on the machines that host the shared model parameters
  - different optimization algorithms, consistency schemes, and parallelization strategies

Tensorflow Example

$$f(x, y) = x^2y + y + 2$$

- Tensors (Inputs/Outputs)
  - N dimensional arrays with a primitive data type
  - Dense at the lowest level
- Operations
  - An operation can be polymorphic and variadic at compile-time
  - Const, MatMul, or Assign
- Stateful operations: Variables and Queues

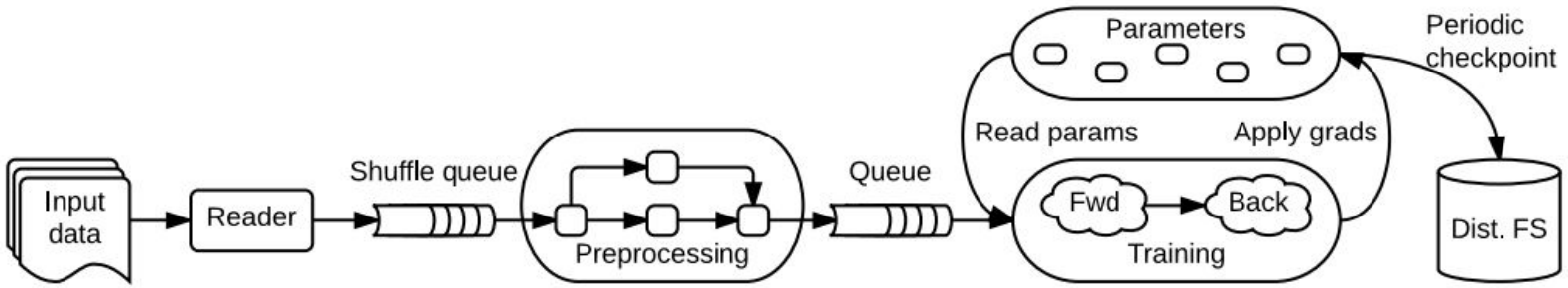


Partial and Concurrent Execution

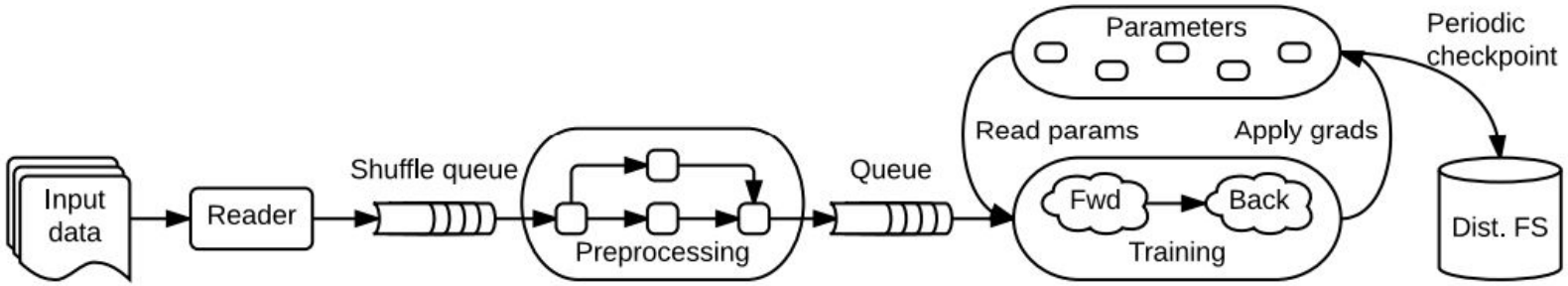
The API for executing a graph allows the client to specify declaratively the subgraph that should be executed.

Each invocation of the API is a **step**.

TensorFlow supports multiple concurrent steps on the same graph. Stateful operations allow steps to share data and synchronize when necessary.



Advantages : Partial and Concurrent Execution



- Subgraphs for each sub part of the dataflow pipeline
- Concurrent executions of a TensorFlow subgraph run asynchronously with respect to one another.

Distributed Execution

Each operation resides on a particular device, such as a CPU or GPU in a particular task. A device is responsible for executing a kernel for each operation assigned to it.

TensorFlow allows multiple kernels to be registered for a single operation, with specialized implementations for a particular device or data type.

The TensorFlow runtime places operations on devices, subject to implicit or explicit constraints in the graph

.

Dynamic control flow

- Will be covered by Artem in greater detail

- TensorFlow supports advanced machine learning algorithms that contain conditional (if statements) and iterative control (while) flow.
  - In LSTMs or other complex ML models, dynamic control flow enables iteration over sequences that have different lengths

Extensibility case studies

- Differentiation
  - Tensorflow extended the algorithm to differentiate conditional/iterative sub-computations by adding nodes to the graph that record the control flow decisions in the forward pass, and replaying those decisions in reverse during the backward pass.
- Optimization
  - We can implemented advanced optimization techniques like Adam with *Variable* operations and primitive mathematical operations without modifying the underlying system.

Training very large models

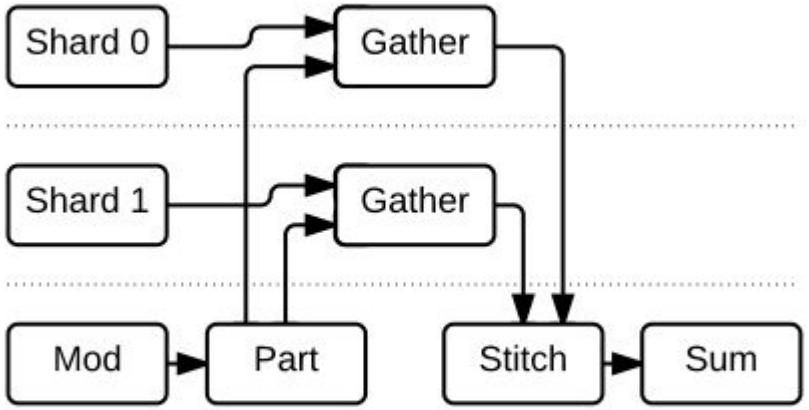


Figure 4: Schematic dataflow for an embedding layer (§4.2) with a two-way sharded embedding matrix.

Training very large models (sparse matrix):

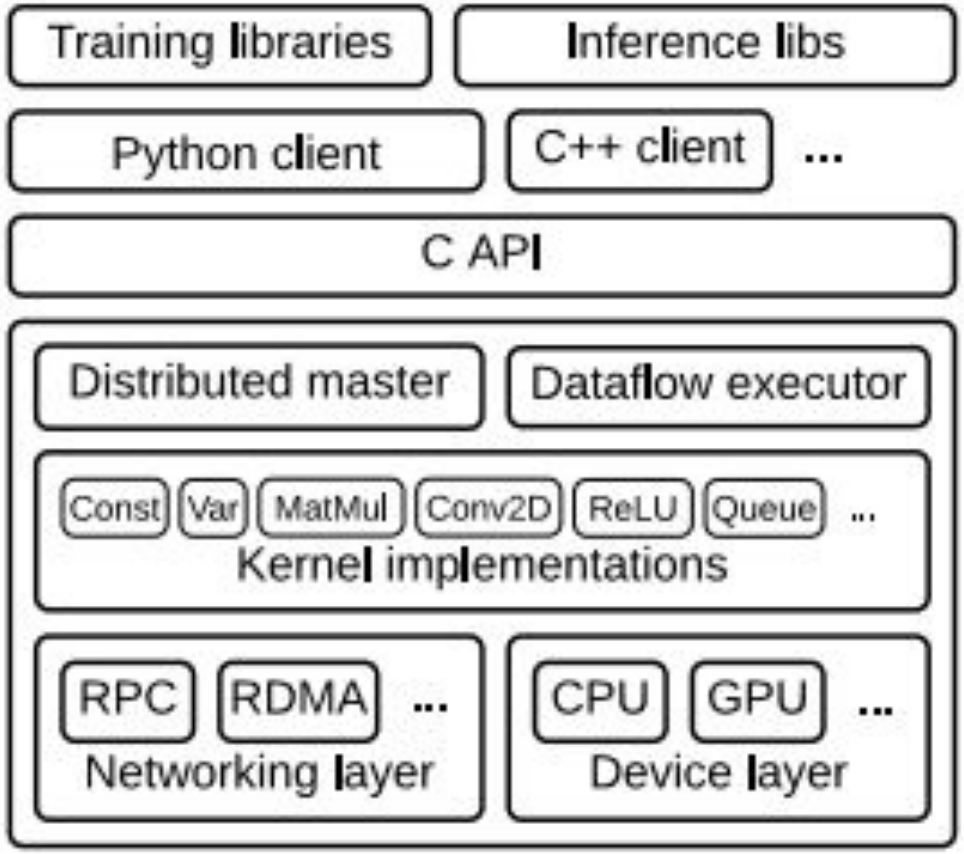
- **Gather**: extracts a sparse set of rows from a tensor
- **Part**: Divides the incoming different tensors that contain the indices to each shard
- **Stitch:** reassembles the partial results from each shard into a single result tensor

Fault Tolerance

Most of the Fault tolerance is pushed to the user. This can mainly done using **Save** and **Restore.**

- **Save** periodically saves a specific checkpoint
- **Restore** restores to the last checkpoint

Implementation



User level code is separated by the core tensorflow library through a C API

Evaluation

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

Table 1: Step times for training four convolutional models with different libraries, using one GPU. All results are for training with 32-bit floats. The fastest time for each model is shown in bold.

Evaluation

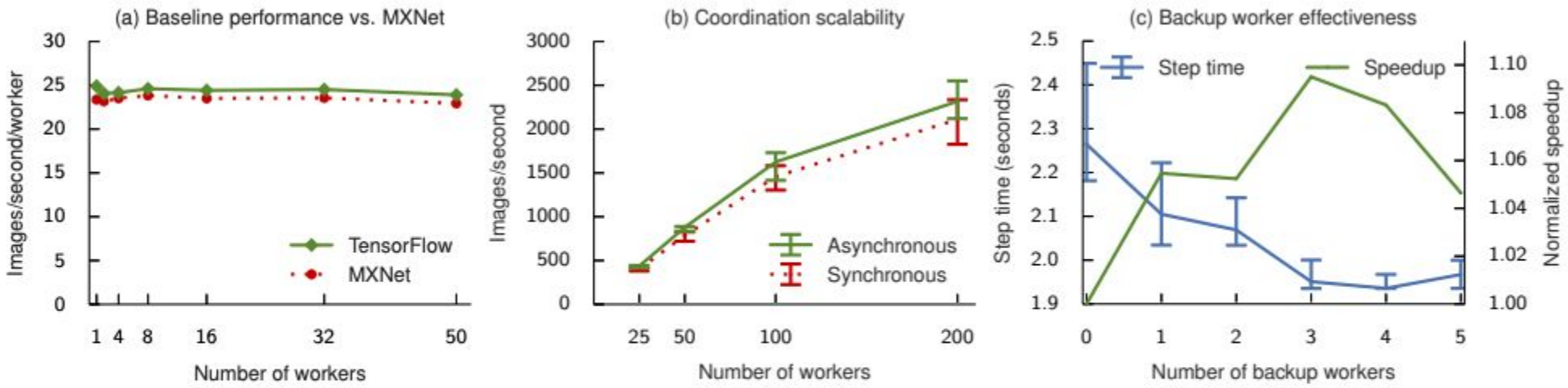
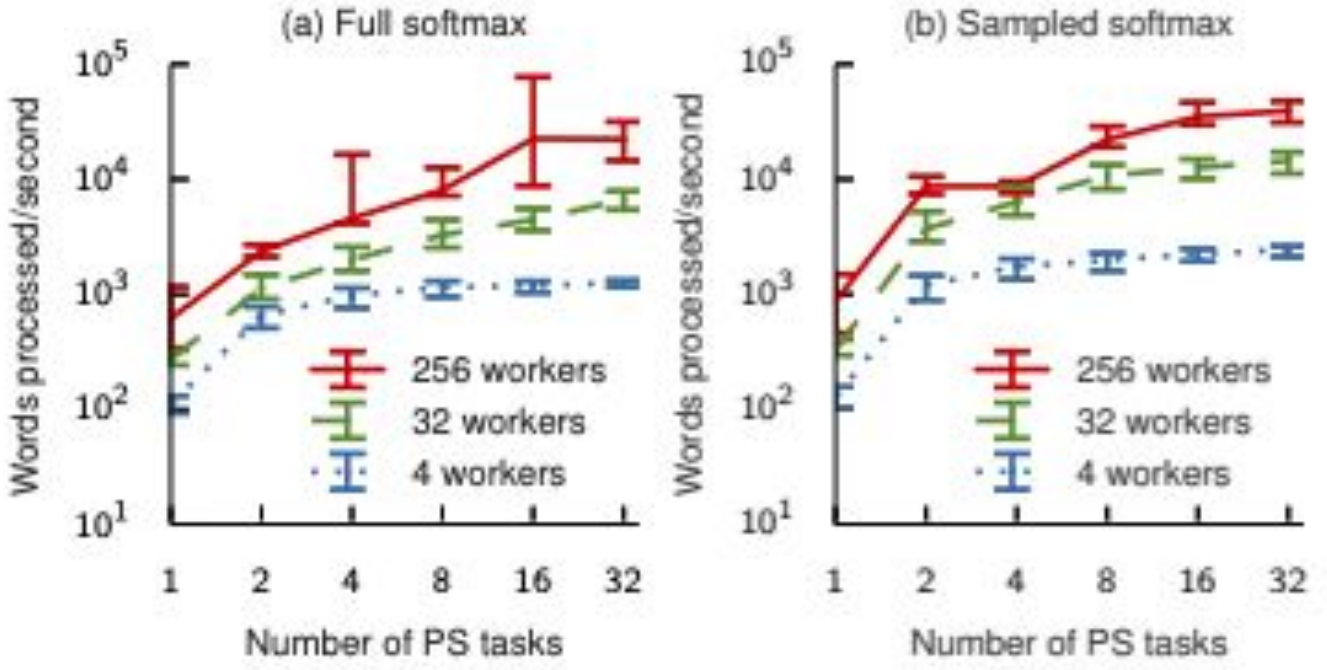


Figure 8: Results of the performance evaluation for Inception-v3 training (§6.3). (a) TensorFlow achieves slightly better throughput than MXNet for asynchronous training. (b) Asynchronous and synchronous training throughput increases with up to 200 workers. (c) Adding backup workers to a 50-worker training job can reduce the overall step time, and improve performance even when normalized for resource consumption.

Evaluation

Performance of training a RNN on One Billion Word Benchmark

Increasing the number of PS tasks leads to increased throughput for language model training, by parallelizing the softmax computation. Sampled softmax increases throughput by performing less computation.



Other frameworks

- Theano and Caffe expresses model as a dataflow. Theano is very similar to DistBelief where it is hard to add new layers or optimizers
- Torch/Pytorch offers a imperative programming model like Tensorflow
- Parameter Server Architecture is very similar to MXNet
- Tensorflow just like SparkNet and DryadLINQ has less costs in batches

Dynamic Control Flow in Large-Scale Machine Learning
(Yu et al., EuroSys'18)

Artem Abzaliev and Saisamrit Surbehera

"We analyzed more than 11.7 million (!) unique graphs for machine learning jobs at Google over the past year, and found that approximately 65% contain some kind of conditional computation, and approximately 5% contain one or more loops."

Adding `if-else` conditions and `while` loops in TensorFlow Graph

- ~~Large Scale Machine Learning~~ = TensorFlow
- ~~Control Flow~~ = computational graph
- ~~Dynamic~~ - `if-else` conditions and `while` loops

Recap: TensorFlow DataFlow Graph

- encodes control-flow decisions as operations, 'in-graph'
- has many advantages
- contrary to 'out-of-graph', where control-flow decisions are in the client process - (Py)Torch

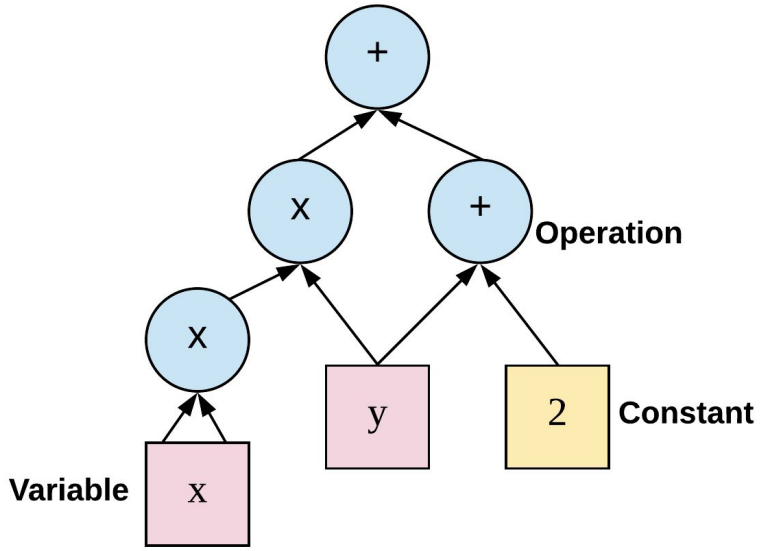


Image credit: https://easy-tensorflow.com/

Why do we need dynamic control flow? RNNs

- RNNs - apply same cell function to different tokens
- Without dynamic control flow - repeat the same computation N times in graph (static unrolling)
- With - apply the same computation in a while loop. Less memory and more parallelism!

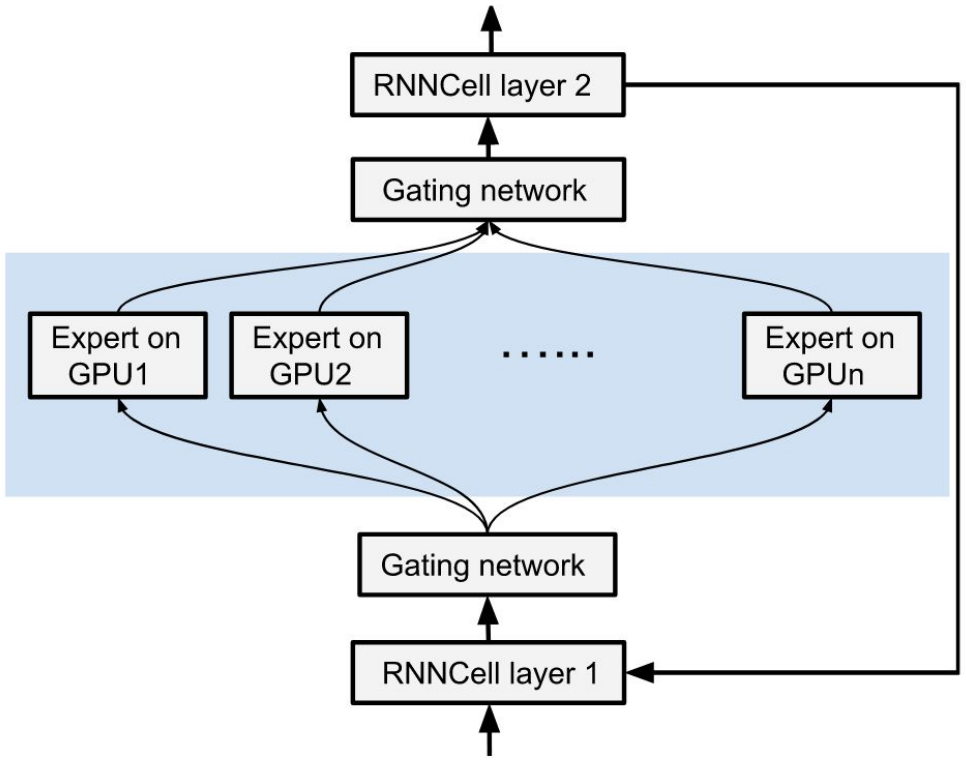


Figure 1: An example model architecture.

Why do we need dynamic control flow? More examples

- Mixture of Experts (MoE) - basically learnable if-elif-else
- Reinforcement learning
- custom requirements

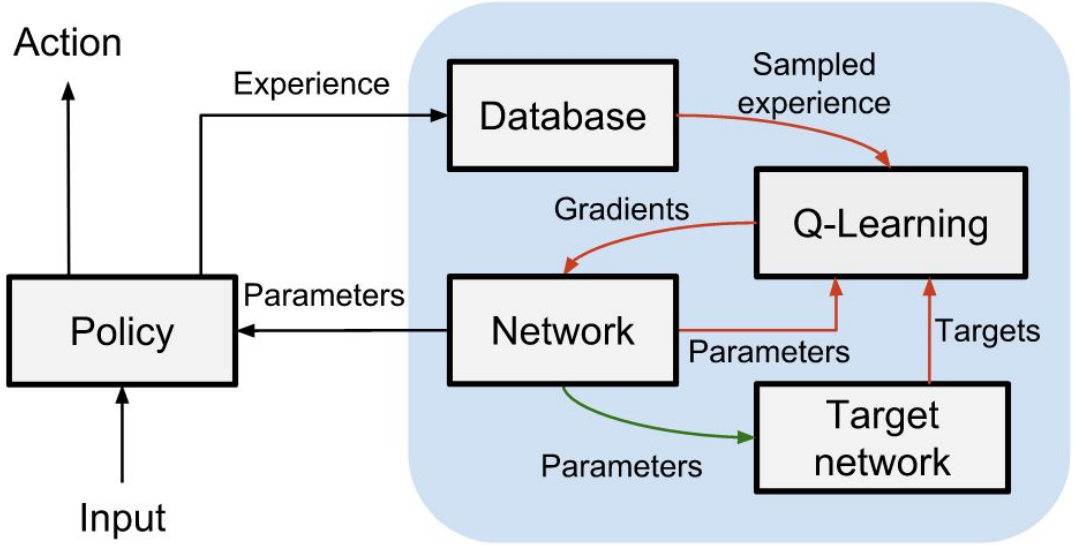


Figure 16: Dynamic control flow in Deep Q-Networks.

Why is it hard to implement dynamic control flow?

- most important: efficient automatic differentiation!
- should work for distributed infrastructure
- heterogeneous environment
- asynchronous

TensorFlow new execution model

- Authors redesigned the local executor of TF
- tensors are represented as `(value, is_dead, tag)`
- `is_dead` - whether the tensor is from untaken branch. Solves (almost) the problem of conditional!
- the tag defines a dynamic execution context—a frame (see the next slide)

The control-flow primitives

- every execution of operation takes a place within a frame
- frames are dynamically allocated execution contexts
- `cond(p, true_fn, false_fn)` uses only Switch and Merge
- while-loop is more complicated, see the next slide

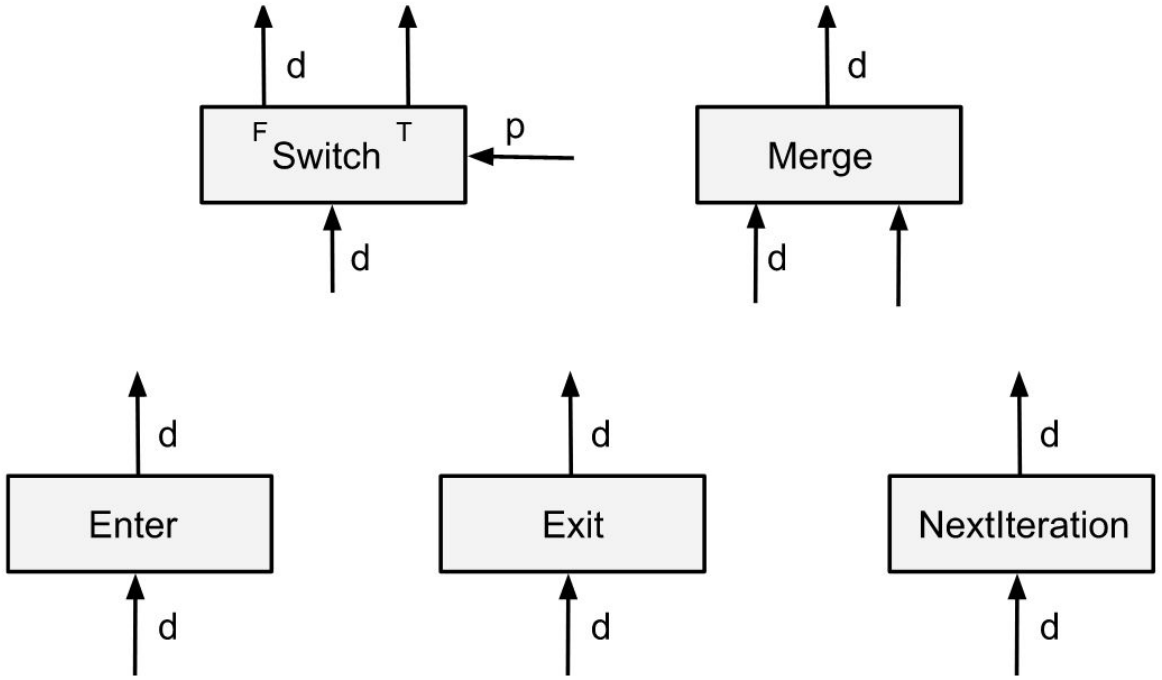


Figure 3: The control-flow primitives.

While-loop implementation

- G_pred - loop predicate and G_body - loop body
- output of `Merge` is used as
  - the input to G_pred to compute the loop termination condition `p`
  - as an input to `Switch`, which forwards the tensor either to `Exit` to terminate the current loop or to G_body

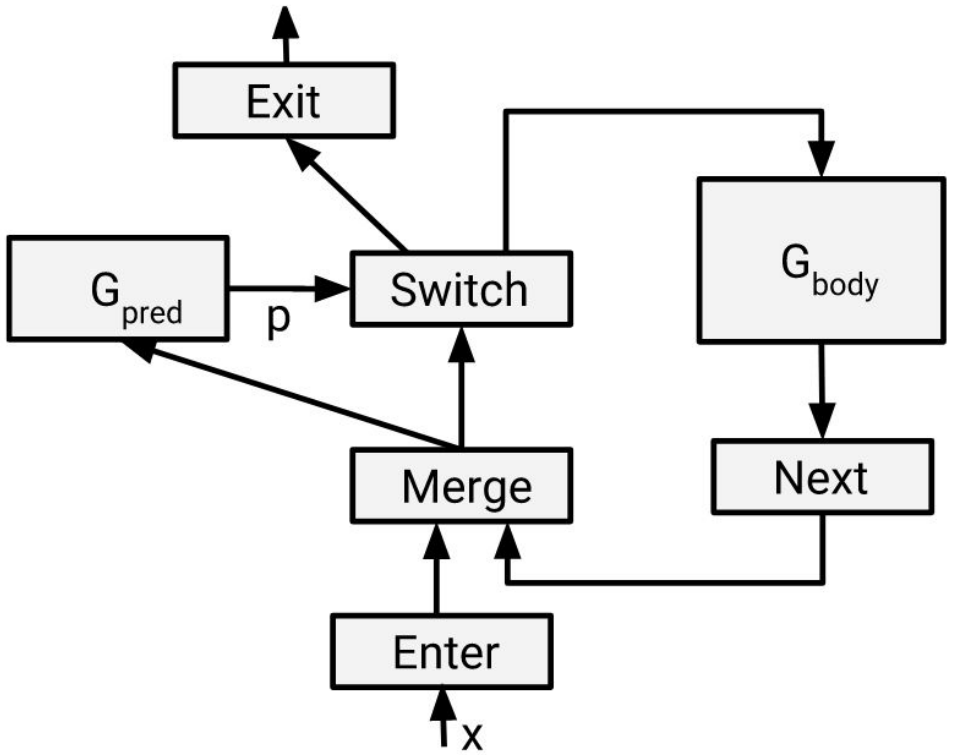


Figure 4: Dataflow graph for a while-loop.

Backprop in TF

```
# NOTE: `op` provides access to the inputs
# and outputs of the original operation.
def MatMulGrad(op, g_z):
  x, y = op.inputs
  g_x = tf.matmul(g_z, tf.transpose(y))
  g_y = tf.matmul(tf.transpose(x), g_z)
  return g_x, g_y
```

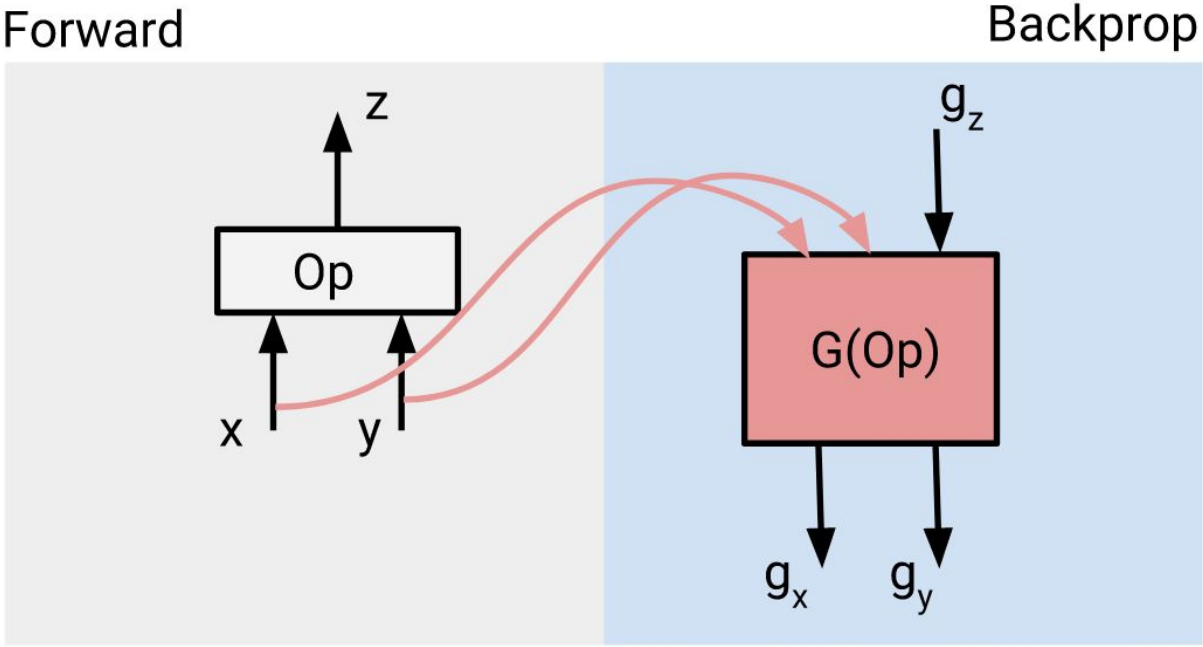
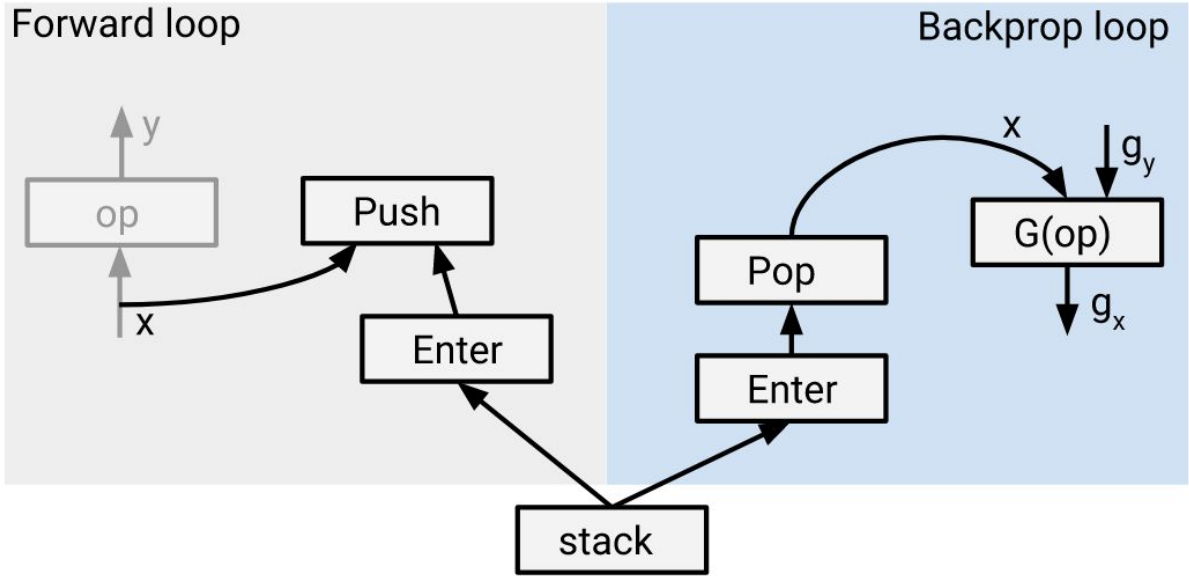


Figure 7: An operation and its gradient function.

Gradients for dynamic control flow

- Gradient of conditional: `tf.cond(pred, true_fn_grad(g_z), false_fn_grad(g_z))`
- Gradients for while-loop: saves values across the loops in a stack
  - the forward computation pushes onto the stacks
  - the gradient computation pops



Memory Management

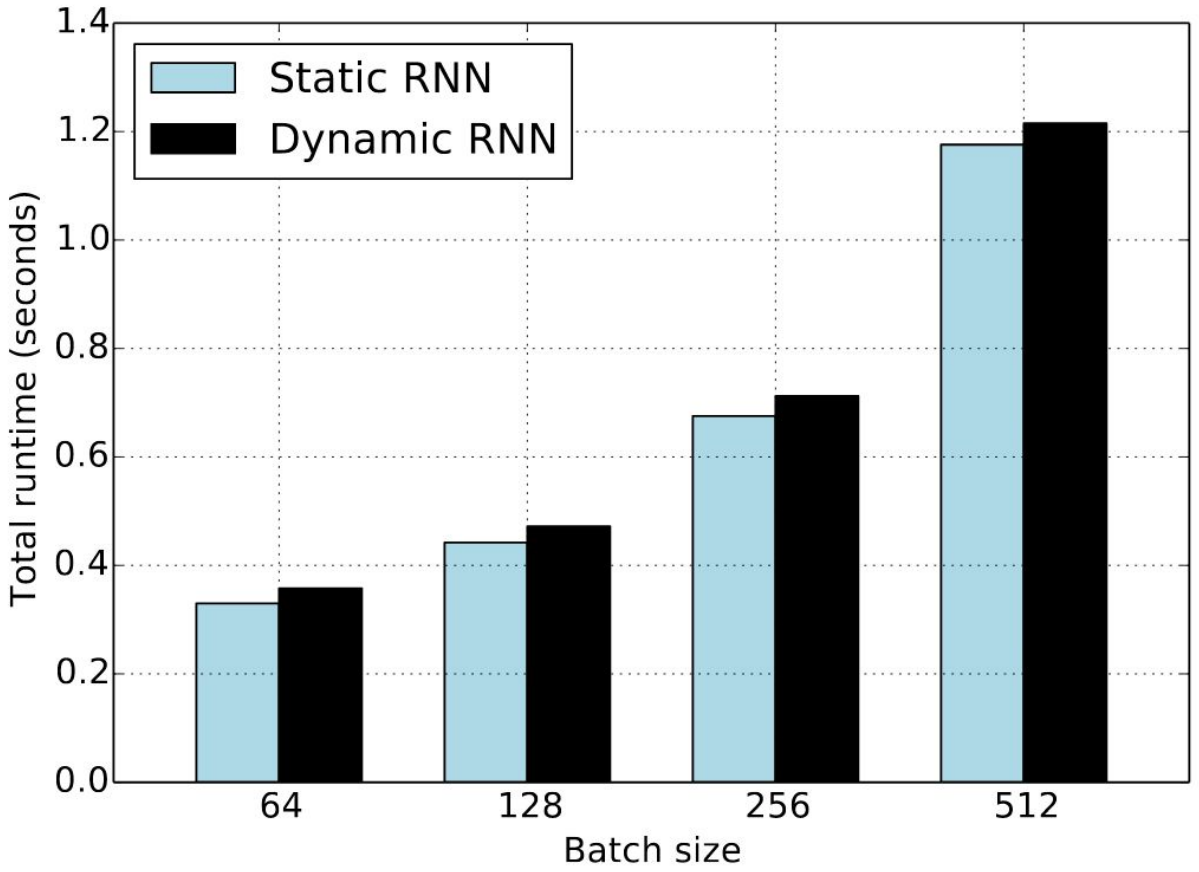
- tensors are moved from GPU to CPU memory when they are pushed onto stacks, and brought back gradually before they are needed in backpropagation
- Separate GPU streams for compute and I/O operations

	Training time per loop iteration (ms), by sequence length						
Swap	100	200	500	600	700	900	1000
Disabled	5.81	5.78	5.75	OOM	OOM	OOM	OOM
Enabled	5.76	5.76	5.73	5.72	5.77	5.74	5.74

Table 1: Training time per loop iteration for an LSTM model with increasing sequence lengths.

Evaluation results

- slowdown due to overhead of dynamic control flow
- but dynamic RNN allows for sequences of length 256, while static unrolling is out of memory at 128



Model Parallelism

- the speedup is sub-linear,due to the additional DMA overhead
- speedup of 5.5×at 8 GPUs

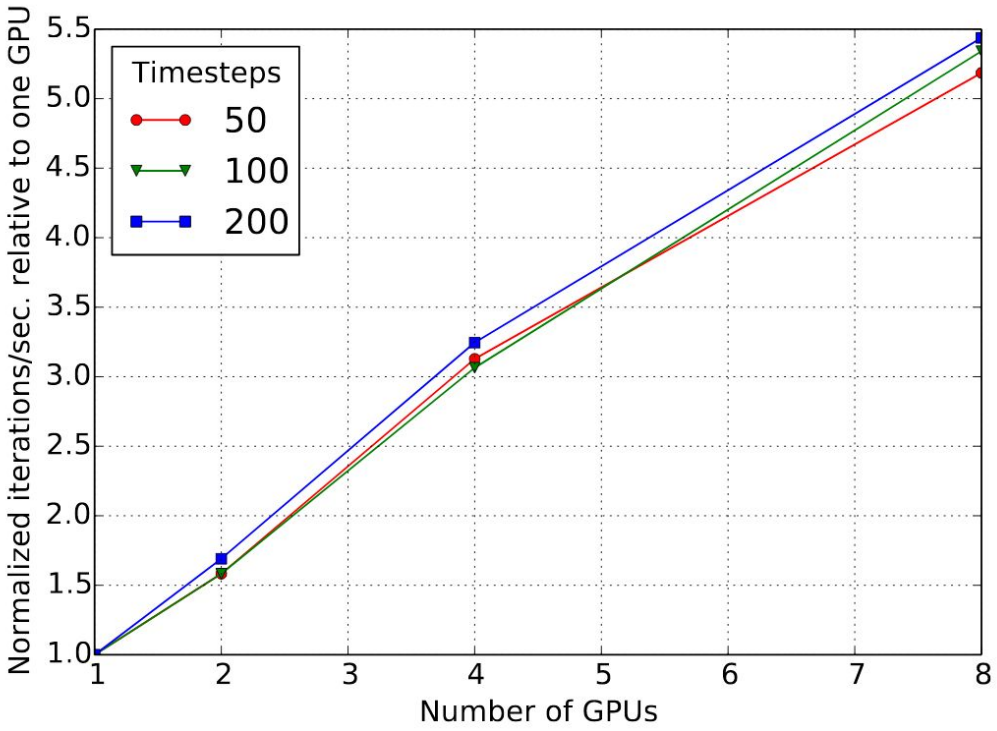


Figure 15: Parallel speedup for an 8-layer LSTM as we vary the number of GPUs from 1 to 8.

Summary

- dynamic control flow enables parallel and distributed execution in heterogeneous environment
- smart memory swapping enables better GPU memory utilization
- hard to compare with other frameworks

Advanced (only if we will have time)

Routine

The distributed master

- translates user requests into execution across a set of tasks.
- prunes and partitions the graph to obtain subgraphs for each participating device,
- caches these subgraphs and re-uses in subsequent steps.

The dataflow executor in each task handles requests from the master, and schedules the execution of the kernels that comprise a local subgraph. Runs the kernels in parallel

Implementation Details

- Many of the operation kernels are implemented using Eigen::Tensor, which uses C++ templates to generate efficient parallel code for multicore CPUs and GPUs
- Supports multiple languages. Ported established features to C++ top optimize implementation
- Users compose standard operations to build higher-level abstractions, such as neural network layers, optimization algorithms, and sharded embedding computations

How does distributed execution work

- Central coordinator partitions the graph into subgraphs, one per device
- When this partitioning would cut an edge between two devices, it automatically replaces the edge with a pair of communication operations, Send(t, k) and Recv(k), which share a **rendezvous** key
- without dynamic control flow, each operation in the graph executes exactly once => every Send/Recv pair receive a unique rendezvous key at graph-construction time
- **non-unique with dynamic control flow!**

What do we have to do to solve this problem?

- For conditional computation, we must provide a mechanism to inform any waiting Recv operations on untaken branches, to reclaim resources
- For iterative computation, we must provide a mechanism to allow a loop-participating partition to start its next iteration or terminate its computation

While-loop implementation (distributed)

- every participating device needs to receive a boolean at each iteration from the device that produces the loop predicate.

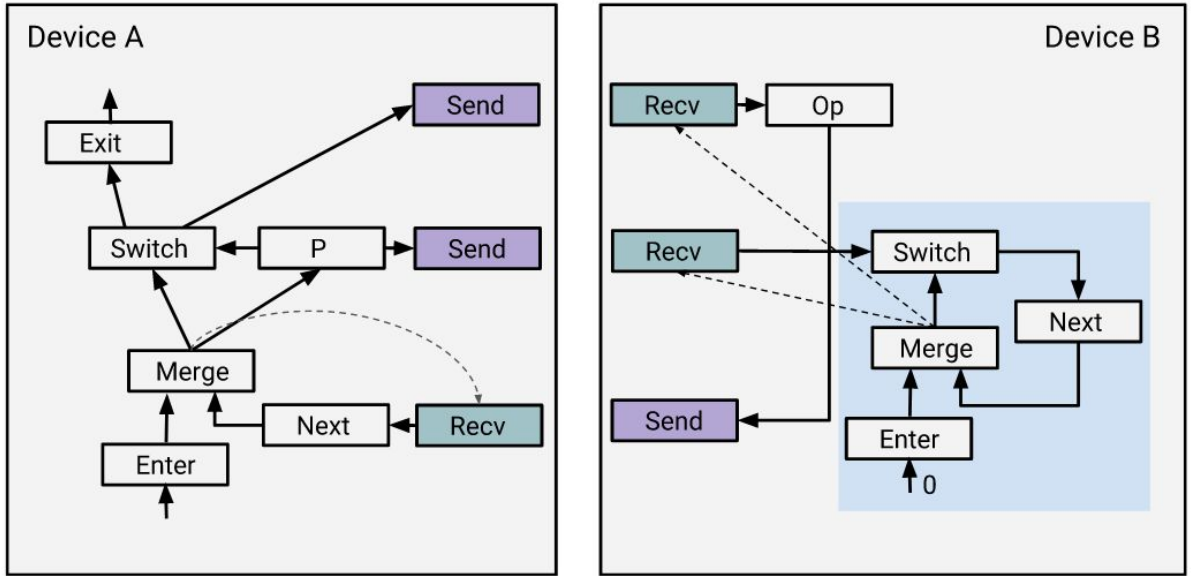


Figure 6: Distributed execution of a while-loop.