Determinism in Deep Learning (S9911)

Duncan Riach, GTC 2019

RANDOMNESS

Pseudo-random number generation

Random mini-batching

Stochastic gradient descent

Data augmentation

Regularization / generalization

DETERMINISM

Elimination of truly random effects

Bit-exact reproducibility from run-to-run

Same model weights

Same inference results

Same graph generated

GOALS

Reasonably high performance

No changes to models

GUARANTEED FOR SAME

number of GPUs

GPU architecture

driver version

CUDA version

cuDNN version

framework version

distribution setup

ADVANTAGES

AUDITING

In safety-critical applications

EXPERIMENTATION

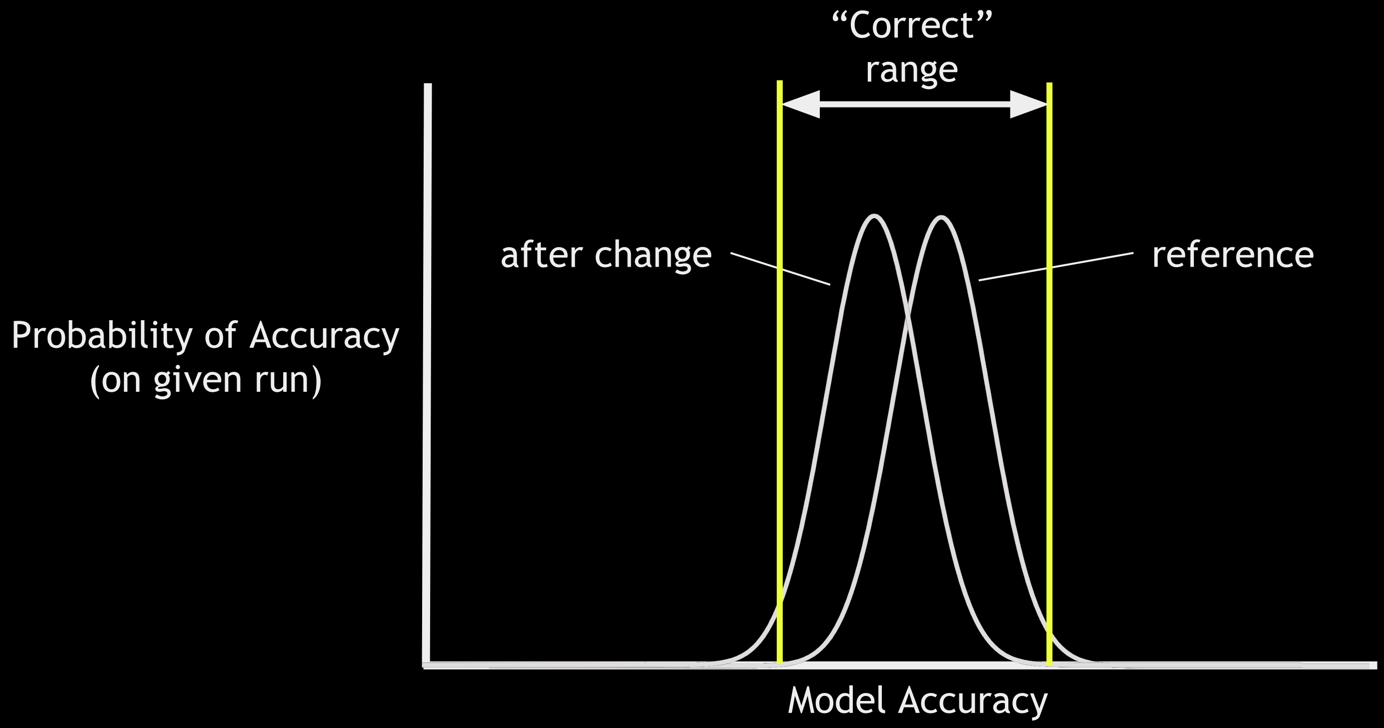
Hold all independent variables constant

DEBUGGING

Reproduce a failure in a long run

REGRESSION

Re-factor without introducing bugs


"Correct"
range
after change
reference
Probability of Accuracy
(on given run)
Model Accuracy

BELIEFS

"TensorFlow is inherently non-deterministic."

"GPUs are inherently non-deterministic."

"This problem can't be solved."

"Nobody cares about this."

"Non-determinism is required for high-performance."

"It's easy. Just set the seeds."

HYPOTHESES

random seeds
tf.reduce_sum / tf.reduce_mean
broadcast addition (for adding bias)
TensorFlow autotune
gate_gradients
TensorRT
asynchronous reductions
GEMM split between thread-blocks

eigen kernels
max-pooling
distributed gradient update
multi-threading in the data loader
image and video decoding
data augmentation
CPU compute
CUDA atomicAdd()

TWO-SIGMA BLOG POST

“A Workaround for Non-Determinism in TensorFlow”

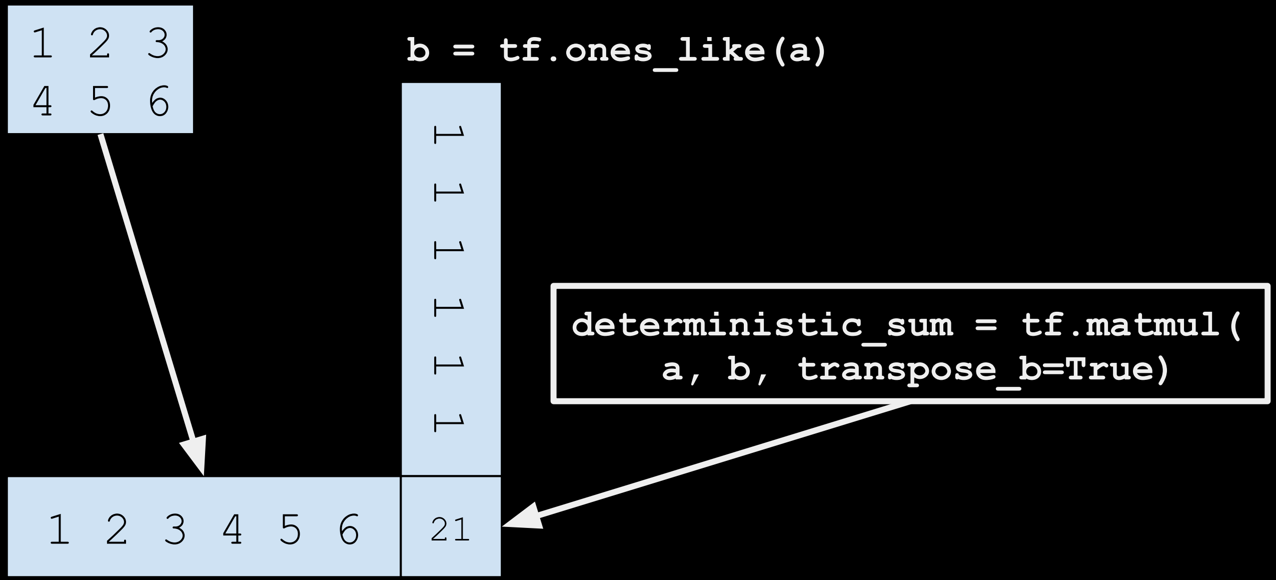
bit.ly/two-sigma-determinism

`tf.reduce_sum()`

add bias using `tf.add()`

WORK-AROUND PART 1

```
input = tf.constant([[1, 2, 3], [4, 5, 6]])
```




```
a = tf.reshape(input, [1, -1])
```

WORK-AROUND PART 2

$o = i * w + b$

layer inputs | layer outputs | batch

			w00	w01	w02	w03
			w10	w11	w12	w13
			b0	b1	b2	b3
i00	i01	1	o00	o01	o02	o03
i10	i11	1	o10	o11	o12	o13
i20	i22	1	o20	o21	o22	o23

```
deterministic_mm_with_bias = tf.matmul(concat_1(i), concat(w, b))
```

BUT NOW

tf.reduce_sum() is deterministic

tf.add() is deterministic

SOLVE A REAL PROBLEM

Project MagLev: at-scale machine-learning platform

2D object detection model for autonomous vehicles

Production scale:

- Millions of trainable variables
- Millions of training examples

HACKERNOON

How to Debug Any Problem

The ability to quickly and effectively find and resolve bugs in new and established systems is one of the most valuable engineering skills that you can develop. Since this skill enables the rapid...

8 min read | 2.3K claps

bit.ly/how-to-debug

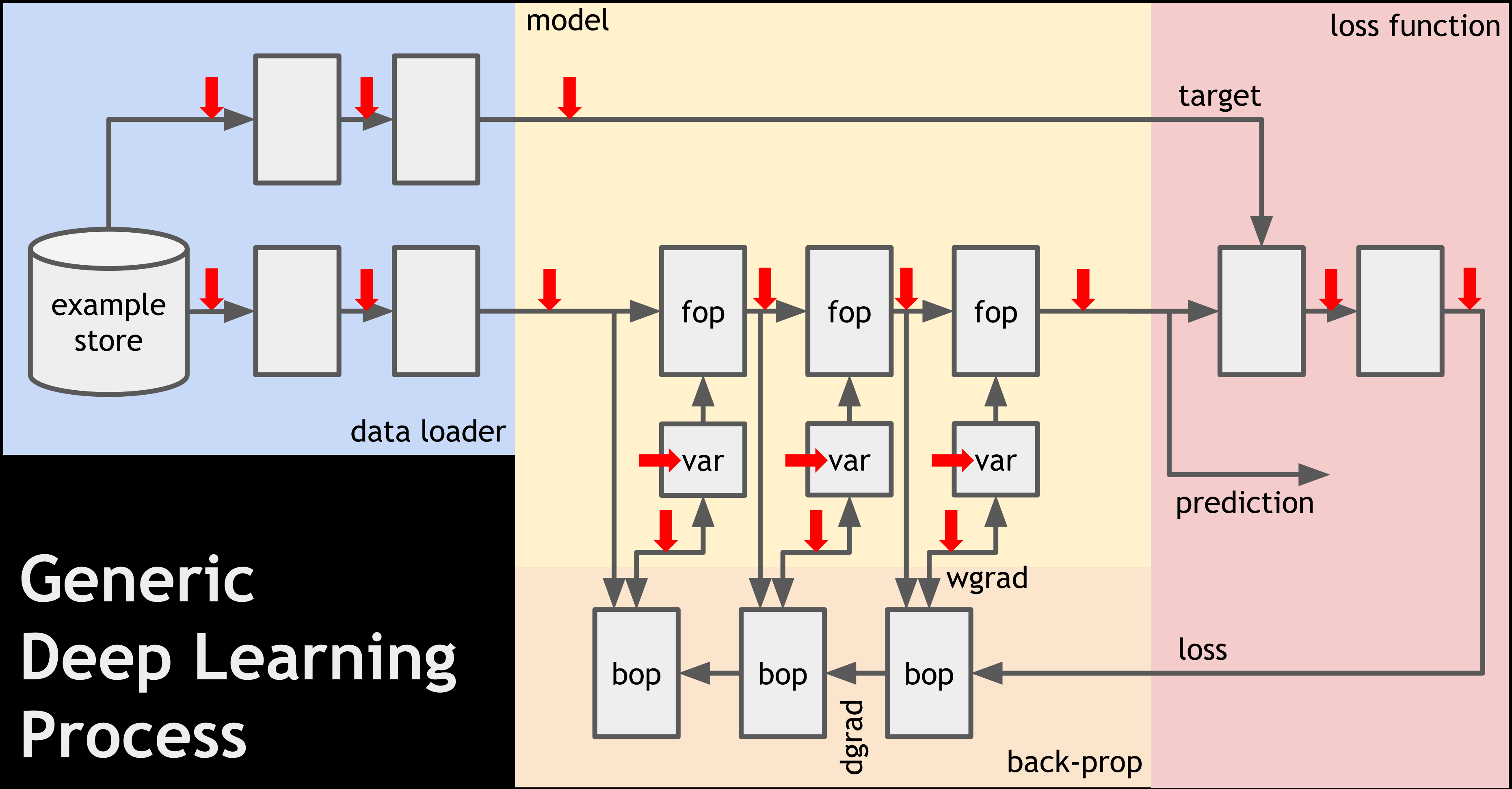
HOW TO DEBUG

Determine what is working

Determine precisely what is not working

Generate hypotheses

Test hypotheses using divide and conquer



Generic Deep Learning Process

DETERMINISM DEBUG TOOL

Insert probe ops at various places in graph

Train the model twice

Identifies location and step of non-determinism injection

DETERMINISM DEBUG TOOL

```
from tensorflow-determinism import probe

tensorflow_op_output = probe.monitor(
    tensorflow_op_output, "name_for_place_in_graph")
```

DETERMINISM DEBUG TOOL

Inserts back-propagatable monitor ops for:

- list, named-tuple, dict, or element
- element is int, float, string, or tf.Tensor (including zero-dimensional tensor)
- recursively, e.g. list-of-named-tuples-of-elements

DETERMINISM DEBUG TOOL

Some of the other types of monitors:

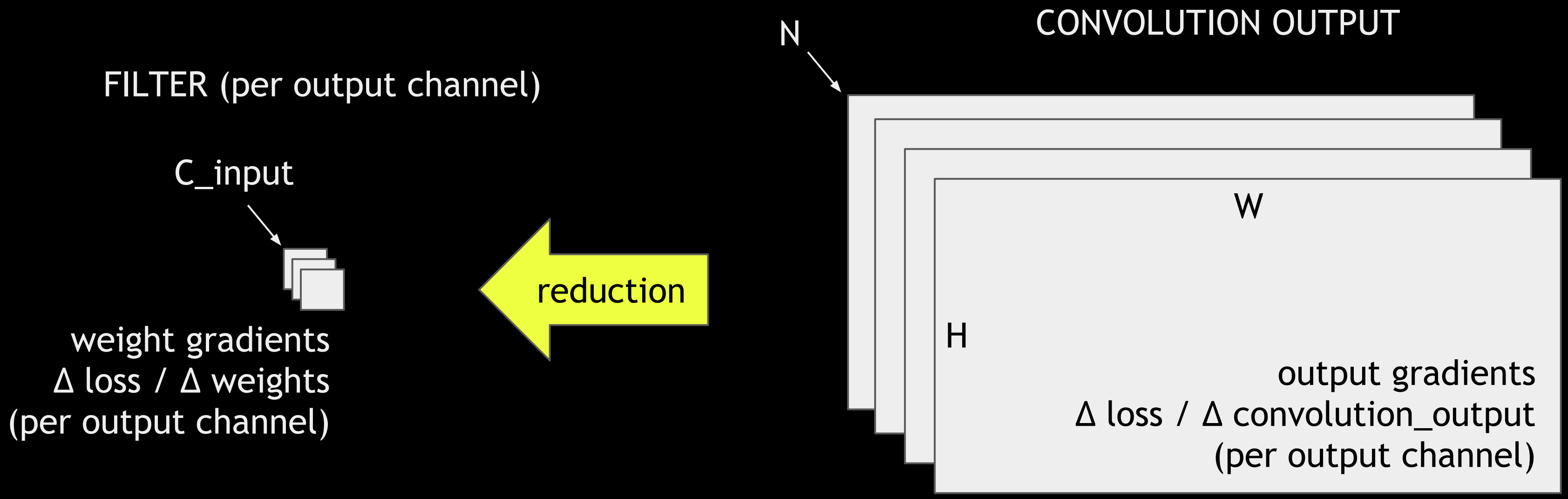
- `probe.monitor_keras()`
  For monitoring output of Keras layer
- `probe.monitor_gradients()`
  Place between `compute_gradients()` and `apply_gradients()`
- `probe.summarize_trainable_variables()`
  Use before training, after each step, or at the end of training

Also monitoring tools for tf.estimator and tf.keras, gradients and trainable variables

```
Run:                                       0 |                                      1
===========================================================================================
sum of weights (before training):
    0                3738.0405251979828 |                3738.0405251979828 (    MATCH)
load_examples_data_0:
    0 f67deaa7a7b36c2e3f44dd9451476993 | f67deaa7a7b36c2e3f44dd9451476993 (    MATCH)
    1 05dd626e553e4de9120777796b66ec80 | 05dd626e553e4de9120777796b66ec80 (    MATCH)
    2 c344c0ffde1f8d32f6ce15fd0b8d7c44 | c344c0ffde1f8d32f6ce15fd0b8d7c44 (    MATCH)
    3 cef41f355e431546da36a09cb920ce9d | cef41f355e431546da36a09cb920ce9d (    MATCH)
    4 a3252ed988249ca6808da11a008b0d2b | a3252ed988249ca6808da11a008b0d2b (    MATCH)
    5 efd549a1b751bb6ef004eca593a6754e | efd549a1b751bb6ef004eca593a6754e (    MATCH)
    6 0b323c26f87e83754f52c84be8e71c41 | 0b323c26f87e83754f52c84be8e71c41 (    MATCH)
                                     • • •
   99 59624187c5eab95623eda94eae06e2e3 | 59624187c5eab95623eda94eae06e2e3 (    MATCH)
activation_1:
    0 c9ec1a9495a6bdeada39824339217a6a | c9ec1a9495a6bdeada39824339217a6a (    MATCH)
    1 0e32644c2eba091a65fb1c9314df5bf4 | 0e32644c2eba091a65fb1c9314df5bf4 (    MATCH)
    2 ade8f097a13dadcb1b195e1d2c5dbb17 | 445cd6096eb2dc8d6eba40648f37f2b0 (MISMATCH)
    3 77a97d2da4e67984b43967ad57661c64 | 099448c86672be5133001f2c90bd8008 (MISMATCH)
                                     • • •
   98 7d8a669e30fed8b0ac8c58eb7e0fecb4 | 6bc154c6216ce33f080c8fbf8243152c (MISMATCH)
   99 c36fcdccc5a49b656aebfcd410e72f4c | 4f022b4a5a8301f19694bdad178342ca (MISMATCH)
Gradient for block_3b_conv_1/kernel:0:
    0 a25625ecc4b7ff956e91690aa2099426 | a25625ecc4b7ff956e91690aa2099426 (    MATCH)
    1 4481e9c648fddbdacc132ec4f8788685 | e9952981e265540d250ff208daa5b90e (MISMATCH)
    2 437a2069155ea7c3d6424246ef0c4e36 | 65637a8b82febfc8ddafe32fc0e1619a (MISMATCH)
    3 ce2caa7be36a50b6875a0d5ded29e3da | 0afb48a6df7d2dfd485fac5acf12af50 (MISMATCH)
```

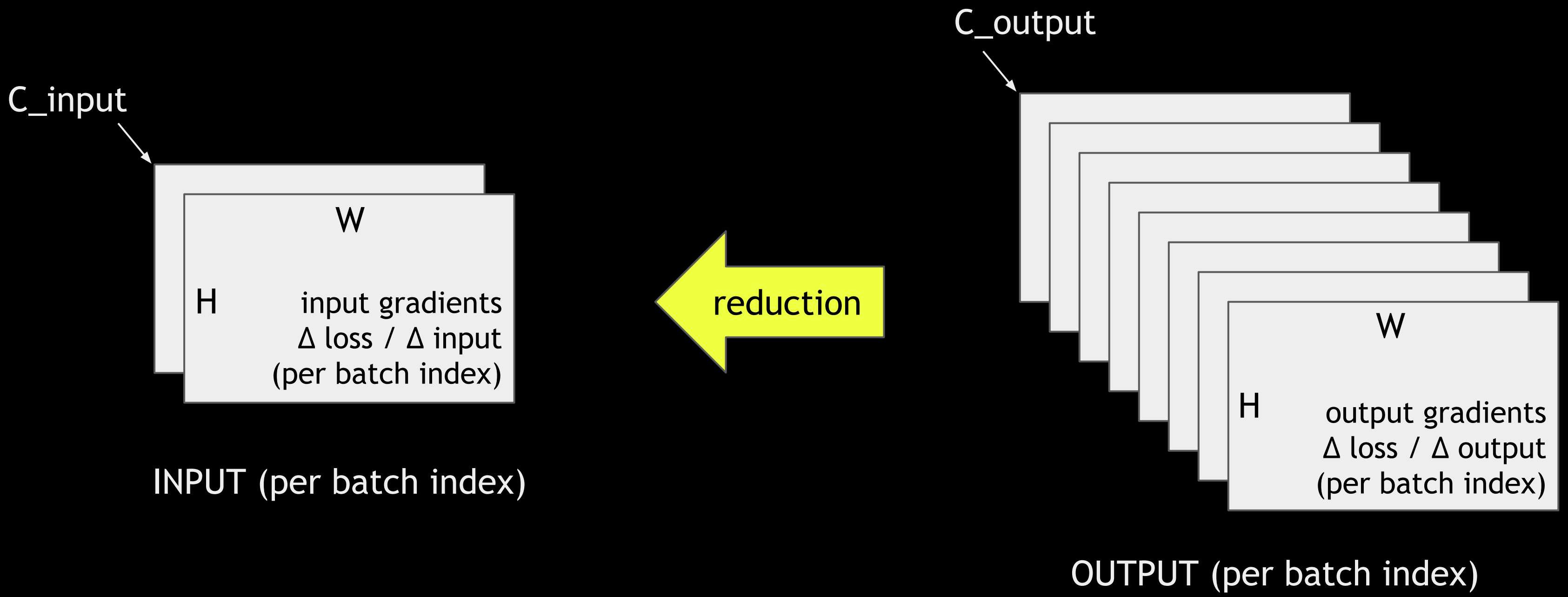
CONVOLUTION

Back-Prop to Weight Gradients



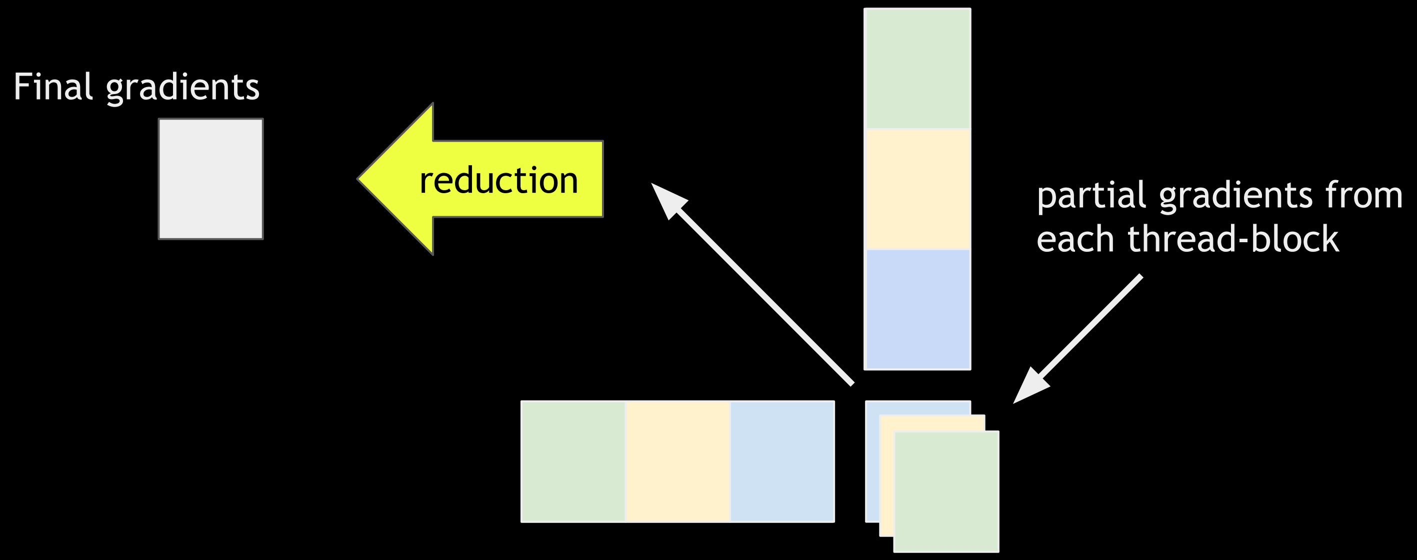
CONVOLUTION

Back-Prop to Data Gradients



CONVOLUTION

Matrix-Multiplication Hierarchical Reduction



CONVOLUTION

CUDA atomicAdd()

0x10000: 1.0

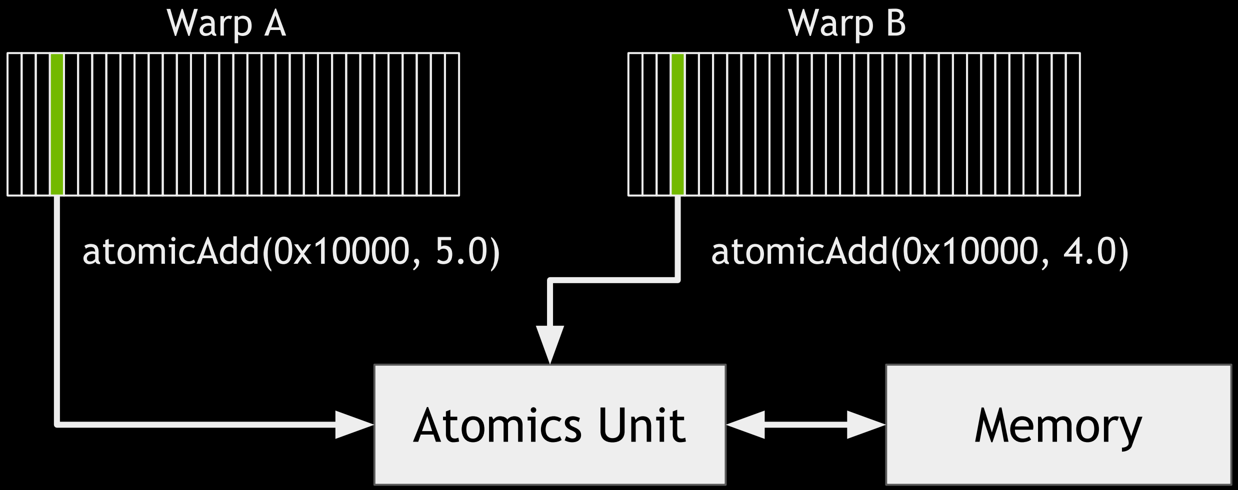
0x10000: 6.0

0x10000: 10.0

0x10000: 1.0

0x10000: 5.0

0x10000: 10.0



CONVOLUTION

atomicAdd() Advantages

Serializes operations without stalling parallel threads

Assures atomic read-modify-write of memory

- i.e. avoids race conditions

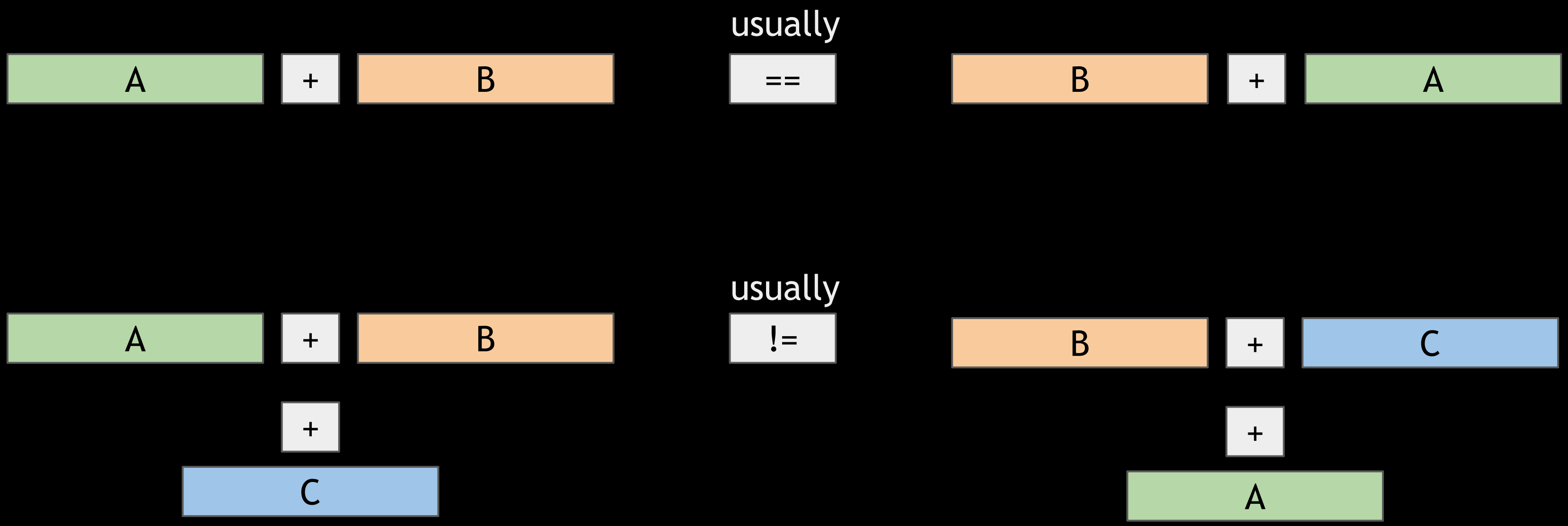
Very easy to program

No need to synchronize between thread-blocks

Very fast read-modify-write loop near memory/cache

CONVOLUTION

Floating-Point Rounding Errors



CONVOLUTION

Root Cause and Solution

CUDA `atomicAdd()`

TensorFlow cuDNN auto-tuning

`TF_CUDNN_DETERMINISTIC`
to disable auto-tuning and select deterministic cuDNN convolution algorithms

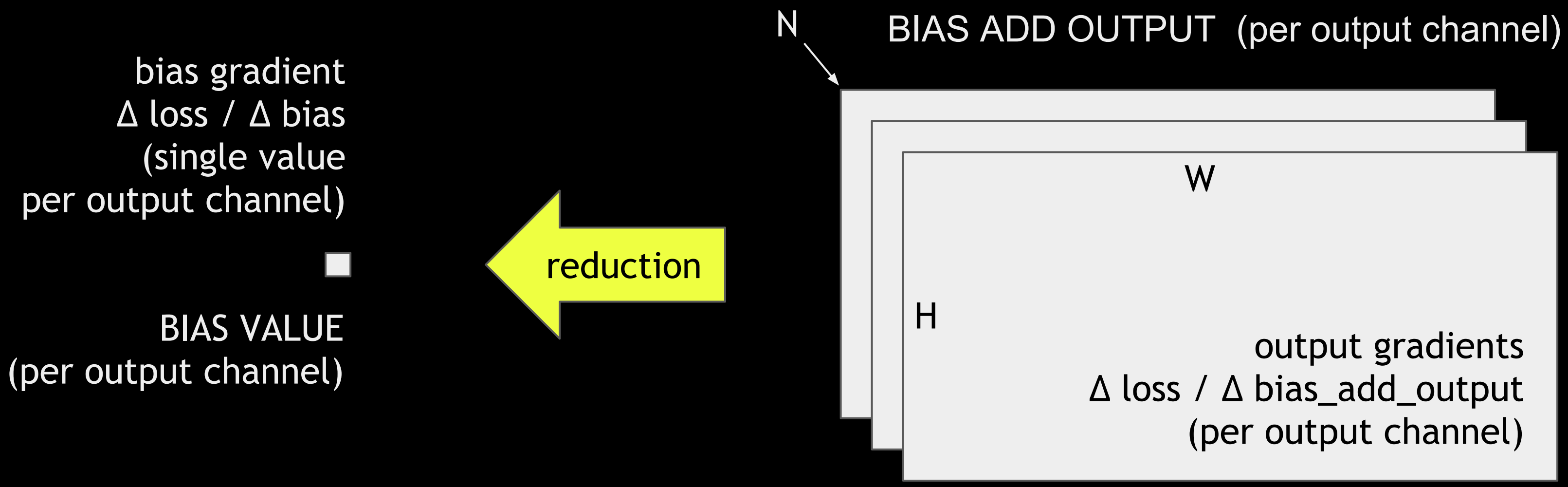
Added to TensorFlow master branch: bit.ly/tf-pr-24747

```
$ export TF_CUDNN_DETERMINISTIC=true
$ python tf_training_script.py
```

```
#!/usr/bin/python
import os
import tensorflow as tf
os.environ['TF_CUDNN_DETERMINISTIC'] = 'true'
# build a graph
```

BIAS ADDITION

Root Cause



`tensorflow.python.ops.nn.bias_add()` uses CUDA `atomicAdd()`

BIAS ADDITION

Temporary Solution

Dynamically patch `tensorflow.python.ops.nn.bias_add()`

Use deterministic ops including implicit broadcasting

```
if data_format == 'NCHW':
  value = tf.math.add(value, tf.reshape(bias, (1, tf.size(bias), 1, 1)))
elif data_format == 'NHWC' or data_format == None:
  value = tf.math.add(value, bias)
```

```
from tensorflow-determinism import patch
patch.bias_add()
```

RARER NON-DETERMINISM

`tf.nn.fused_batch_norm()` back-prop

- Approximately every 10 steps
- Temporary solution: run on CPU

`gate_gradients=tf.train.Optimizer.GATE_OP` (default)

- `optimizer.compute_gradients()` parameter
- Approximately every 100 steps
- `GATE_GRAPH` is guaranteed to be deterministic

RARESTNON-DETERMINISM

Every few thousand steps at random locations

Changed from Pascal to Volta card => non-determinism persisted

Added ability to dump and compare probed tensors between runs

Suspected memory allocation and ownership (time / location)

Ran on cluster => fully deterministic

Updated my driver => fully deterministic locally

Possible causes: off-by-one memory allocation, incorrect cache invalidation, race conditions, clock speed, interface trims

batch-norm and gate_gradients fixes **not required**

INTERIM STATUS

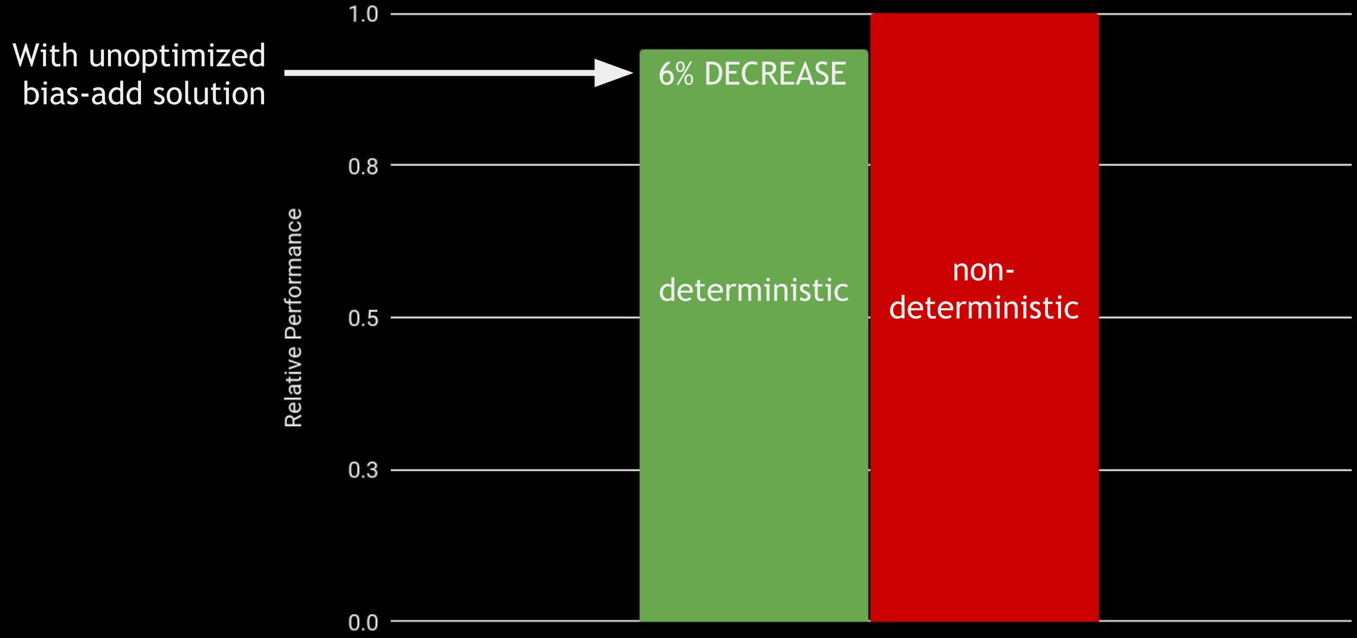
Autonomous-vehicle production model training fully deterministically and correctly on millions of examples

TensorFlow determinism debugging tool developed

Deterministic cuDNN convolution fixes upstreamed to TensorFlow master branch

SINGLE GPU PERFORMANCE

Proprietary AV Perception Model



MULTI-GPU WITH HOROVOD

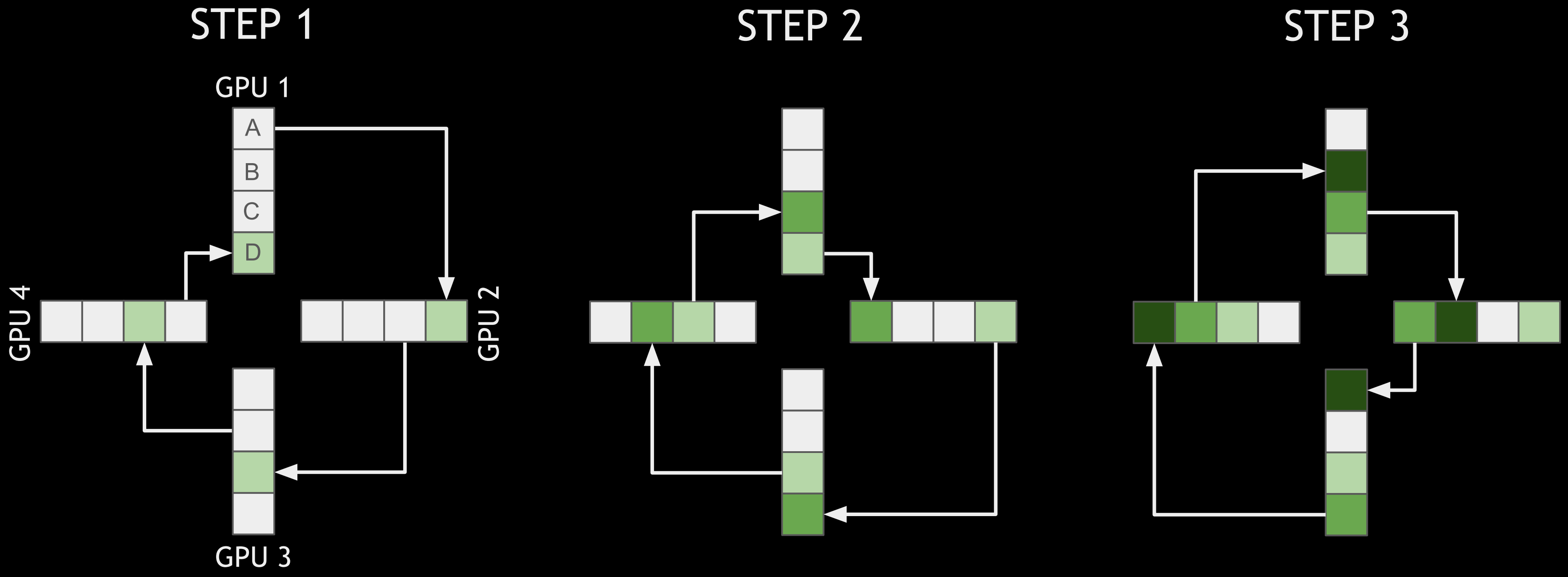
Based on single-GPU determinism recipe

Two GPUs: deterministic out-of-the-box

More than two GPUs non-deterministic

Horovod uses NCCL2 ring-allreduce

RING-ALLREDUCE



Patarasuk, Pitch & Yuan, Xin. (2007). Bandwidth Efficient All-reduce Operation on Tree Topologies. 1 - 8. 10.1109/IPDPS.2007.370405.

HOROVOD TENSOR FUSION

Batch-reduce partial gradient tensors as they become ready

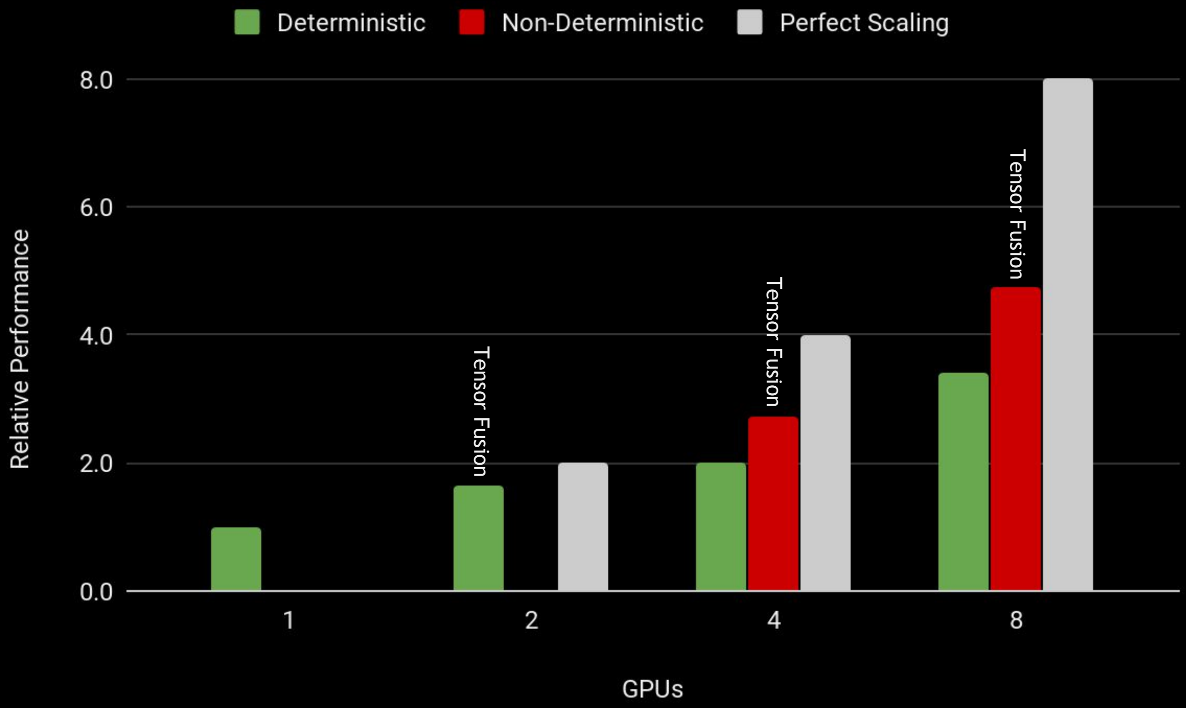
Order of reduction changes on each training step (apparently)

For now: disable Tensor Fusion

```
$ HOROVOD_FUSION_THRESHOLD=0 python train.py
```

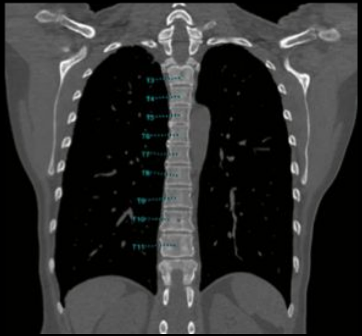
MULTI-GPU PERFORMANCE

Using Single-GPU Determinism Recipe

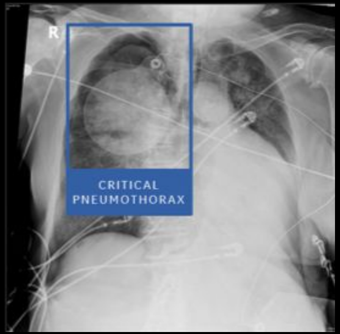


ANOTHER REAL PROBLEM

GE Healthcare



Segmentation and Labeling
CT : BoneVCAR



Alerts for Critical Conditions
X-Ray : GE Critical Care Suite

Optimal Scans
MR : GE **MR AIR**x

Edison

MAX-POOLING

1	2	1	1
1	3	1	1
1	1	1	1

3	3	1
3	3	1

MAX-POOLING

Root Cause & Solution

CUDA `atomicAdd()`

`TF_CUDNN_DETERMINISTIC`

Added to TensorFlow master branch: bit.ly/tf-pr-25269

```
$ export TF_CUDNN_DETERMINISTIC=true
$ python tf_training_script.py
```

```
#!/usr/bin/python
import os
import tensorflow as tf
os.environ['TF_CUDNN_DETERMINISTIC'] = 'true'
# build a graph
```

CPU NON-DETERMINISM

Noticed while I was debugging the distilled model

Much greater variance than GPU

Injection occuring at weight update step

Solution: Use single CPU thread

```
session_config.intra_op_parallelism_threads = 1  (default: 2)
session_config.inter_op_parallelism_threads = 1  (default: 5)
```

Only needed when running on CPU (vs GPU)

CPU

```
    SUM OF WEIGHTS          FINAL LOSS
====================    ==================

Training five times with no fixes
-13.4960977323353291 | 6.1724668502807614
 -9.3681446192786098 | 6.3305957317352295
 -9.1963089210912585 | 6.3364742755889889
-13.6303959703072906 | 6.1670220375061033
 -9.0079690776765347 | 6.3340478420257567

Training twice with all fixes
 -9.6487178248353302 | 6.1068549633026121
 -9.6487178248353302 | 6.1068549633026121

Training bigger config twice with all fixes
 -8.8775541735813022 | 4.1930521011352537 (66.96 s)
 -8.8775541735813022 | 4.1930521011352537 (66.70 s)
```

GPU

```
    SUM OF WEIGHTS          FINAL LOSS
====================    ==================

Training five times with no fixes
-13.5144761633127928 | 6.1083775520324703
-13.5144743174314499 | 6.1083775520324703
-13.5144757004454732 | 6.1083775520324703
-13.5144734960049391 | 6.1083775997161869
-13.5144746471196413 | 6.1083775997161869

Training twice with all fixes
-13.5144764725118876 | 6.1083775997161869
-13.5144764725118876 | 6.1083775997161869

Training bigger config twice with all fixes
  3.7987217940390110 | 3.9343416929244994 (2.43 s)
  3.7987217940390110 | 3.9343416929244994 (2.41 s)
```

COMPLETE RECIPE

1. Set `TF_CUDNN_DETERMINISTIC=true`
    - Disables TensorFlow cuDNN auto-tuning
    - Uses deterministic cuDNN convolution back-prop algorithms
    - Uses deterministic cuDNN max-pooling algorithm
2. Dynamically patch `tf.nn.bias_add()`
3. Set random seed for all random number generators
    - `random.seed(SEED)`, `np.random.seed(SEED)`, `tf.set_random_seed(SEED)`
4. `HOROVOD_FUSION_THRESHOLD=0` for more than 2 GPUs

TENSORFLOW & CUDA ATOMICS

Analysis of TF v1.12 , v1.13.1, and master branch (on 2019-03-03)

About 13 ops that use CUDA `atomicAdd()`

There are ten other CUDA atomic operations, e.g. `atomicCAS()`

‘atomic’ is present in 167 files in the TensorFlow repo

- Some of these may be related to CUDA atomics

CUDA atomics not always associated with non-determinism

There are faster, deterministic ways to reduce within thread-blocks

- i.e logarithmic tree reductions using inter-thread shuffling

INFERENCE

All forward propagation (of course)

- Probably no need to set `TF_CUDNN_DETERMINISTIC=true`
- Possible issues with “deconvolution”

Disable TensorFlow cuDNN autotuning

- Set `TF_CUDNN_USE_AUTOTUNE=false`

TensorRT

- ~500 CUDA kernels, all of them deterministic
- Timing-based auto-tuning running on target architecture can produce different graphs on each run
- We’re working on adding a mechanism to TensorRT to address this

PYTORCH

Set all the seeds

```
random.seed(SEED), np.random.seed(SEED),
os.environ['PYTHONHASHSEED']=str(SEED),
torch.manual_seed(SEED),
torch.cuda.manual_seed_all(SEED)
```

```
torch.backends.cudnn.deterministic=True
```

Covers convolution and max-pooling

I hear that some ops may still be non-deterministic

PLAN

Release current solution in NGC TensorFlow container

`TF_CUDNN_DETERMINISTIC` in TensorFlow v2.0 (end-of-year)

Make `bias_add` deterministic at CUDA kernel level

Open-source determinism debug tool

Add single deterministic switch for all of TensorFlow

Improve deterministic performance of Horovod

Deterministic simulated environments for reinforcement learning

CREDITS

Tero Karras
Tim Zaman
Hao Wu
Jose Alvarez Lopez
Ben Barsdell
Rakesh Ranjan
Simon Layton
John Montrym
Jorge Albericio Latorre
Nicolas Koumchatzky
Carl Case

Nathan Luehr
Conrado Silva Miranda
Jussi Rasanen
Dilip Sequeira
Mikko Ronkainen
Xiang Bo Kong
Sharan Chetlur
Luke Durant
Kevin Brown
Marc Edgar
Cindy Riach
Mostafa Hagog

Yifang Xu
William Zhang
Lauri Peltonen
Joey Conway
Matthijs De Smedt
Kevin Vincent
Bryan Catanzaro
Michael O’Connor
Stephen Warren
Bob Keating
Andrew Kerr

TAKEAWAYS

Neither TensorFlow nor GPUs are inherently non-deterministic

Root cause is asynchronous floating point operations

Use CUDA floating-point atomic operations with care

Deterministic kernels often already available

This was a hard problem to solve, but not impossible

It’s a very important topic. A lot of people care about it

New tools and methodology for debugging

Automated vigilance is warranted

CALL TO ACTION

watch: github.com/NVIDIA/tensorflow-determinism

follow: twitter.com/DuncanARiach

connect: www.linkedin.com/in/duncanriach

email: duncan@nvidia.com