Efficient DNN Training at Scale: from Algorithms to Hardware

Gennady Pekhimenko, Assistant Professor

EcoSystem Group

https://github.com/tbd-ai/tbd-suite

tbd-suite.ai
Systems/Architecture Is a Servant for ML
Researchers proposed many **system-level optimizations** for DNN computation, however, their performance largely depends on the entire stack.

Given a full-stack configuration:

- How much better can we do to improve performance?
- How to identify future opportunities?
Machine Learning Benchmarking and Analysis

In collaboration with Project Fiddle (MSR)
Performance bottlenecks in DNN Training

Diverse benchmark suite with state-of-the-art models

Analysis & Optimizations

Tools

Key performance metrics
Performance bottlenecks in DNN Training

Analysis & Optimizations

Diverse benchmark suite with state-of-the-art models

Tools

Key performance metrics
# Training Benchmarks for DNNs (TBD), Jan. 2018

<table>
<thead>
<tr>
<th>Applications</th>
<th>Models</th>
<th>Dataset</th>
<th># of layers</th>
<th>Dominant layer</th>
<th>Maintainer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image Classification</strong></td>
<td>ResNet-50 $^{T,M,C}$ Inception-v3 $^{T,M,C}$</td>
<td>ImageNet</td>
<td>50 (152 max) 42</td>
<td>CONV</td>
<td>Hongyu Zhu</td>
</tr>
<tr>
<td><strong>Machine Translation</strong></td>
<td>Seq2Seq $^{T,M}$ Transformer $^{T,M}$</td>
<td>IWSLT15</td>
<td>5 12</td>
<td>LSTM Attention</td>
<td>Bojian Zheng, Andrew Pelegris</td>
</tr>
<tr>
<td><strong>Object Detection</strong></td>
<td>Faster RCNN $^{T,M}$ Mask RCNN $^P$</td>
<td>Pascal VOC</td>
<td>101</td>
<td>CONV</td>
<td>Hongyu Zhu, Zilun Zhang</td>
</tr>
<tr>
<td><strong>Speech Recognition</strong></td>
<td>Deep Speech 2 $^{P,M}$</td>
<td>LibriSpeech</td>
<td>7 (9 max)</td>
<td>RNN</td>
<td>Kuei-Fang Hsueh, Jiahuang Lin</td>
</tr>
<tr>
<td><strong>Recommendation System</strong></td>
<td>NCF $^P$</td>
<td>MovieLens</td>
<td>4</td>
<td>GMF, MLP</td>
<td>Izaak Niksan</td>
</tr>
<tr>
<td><strong>Adversarial Network</strong></td>
<td>WGAN $^T$</td>
<td>Downscaled ImageNet</td>
<td>14+14</td>
<td>CONV</td>
<td>Andrew Pelegris</td>
</tr>
<tr>
<td><strong>Reinforcement Learning</strong></td>
<td>A3C $^{T,M}$</td>
<td>Atari 2600</td>
<td>4</td>
<td>CONV</td>
<td>Mohamed Akrout</td>
</tr>
</tbody>
</table>

(Footnotes indicate available implementation: $T$ for , $M$ for , $C$ for , $P$ for )
# TBD Benchmark Suite, Aug. 2020 update

<table>
<thead>
<tr>
<th>Applications</th>
<th>Models</th>
<th>Dataset</th>
<th># of layers</th>
<th>Dominant layer</th>
<th>Maintainer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image Classification</strong></td>
<td>ResNet-50 (<em>{TM}) Inception-v3 (</em>{TM})</td>
<td>ImageNet</td>
<td>50 (152 max) 42</td>
<td>CONV</td>
<td>Xin Li</td>
</tr>
<tr>
<td><strong>Machine Translation</strong></td>
<td>Seq2Seq (<em>{TM}) Transformer (</em>{TM})</td>
<td>IWSLT16</td>
<td>5 12</td>
<td>LSTM Attention</td>
<td>Yu Bo Gao</td>
</tr>
<tr>
<td><strong>Object Detection</strong></td>
<td>Mask RCNN (<em>{TP}) EfficientDet (</em>{TP})</td>
<td>COCO</td>
<td>101</td>
<td>CONV</td>
<td>Yu Bo Gao</td>
</tr>
<tr>
<td><strong>Speech Recognition</strong></td>
<td>Deep Speech 2 (_P)</td>
<td>LibriSpeech</td>
<td>7 (9 max)</td>
<td>RNN</td>
<td>Cong Wei</td>
</tr>
<tr>
<td><strong>Language Modeling</strong></td>
<td>BERT (_P)</td>
<td>SQuAD</td>
<td>24</td>
<td>BERT block</td>
<td>Xin Li</td>
</tr>
<tr>
<td><strong>Reinforcement Learning</strong></td>
<td>MiniGo (_T)</td>
<td></td>
<td>38</td>
<td>CONV</td>
<td>Cong Wei</td>
</tr>
</tbody>
</table>

(Footnotes indicate available implementation: \(T\) for Tensorflow, \(M\) for mxnet, \(P\) for PyTorch)
The v0.7 (datacenter, edge, mobile) result highlights:

- 23 submitting organizations
- Over 1,200 peer-reviewed results - twice as many as the first round
- More than doubles the number of applications in the suite
- New dedicated set of MLPerf Mobile benchmarks
- Randomized third party audits for rules compliance

Read more in the press release.
Performance bottlenecks in DNN Training

Diverse benchmark suite with state-of-the-art models

Tools

Analysis & Optimizations

Key performance metrics
Performance Metrics

- **Throughput**
  *Number of data samples processed per second*

- **Compute Utilization**
  *GPU busy time over Elapsed time*

- **FP32/FP16/Tensor Core Utilization**
  *Average instructions executed per cycle over Maximum instructions per cycle*

- **Memory Breakdown**
  *Which data structures occupy how much memory*
Performance bottlenecks in DNN Training

Diverse benchmark suite with state-of-the-art models

Key performance metrics

Tools

Analysis & Optimizations
Memory Profiler (BERT)

Feature maps are still more important than weights for memory consumption
Performance bottlenecks in DNN Training

Analysis & Optimizations

Tools

Key performance metrics

Diverse benchmark suite with state-of-the-art models
Scaling Back-Propagation by Parallel Scan Algorithm

Shang Wang\textsuperscript{1,2}, Yifan Bai\textsuperscript{1}, Gennady Pekhimenko\textsuperscript{1,2}

\textsuperscript{1}Computer Science, University of Toronto

\textsuperscript{2}VECTOR INSTITUTE

MLSys 2020
Executive Summary

The back-propagation (BP) algorithm is popularly used in training deep learning (DL) models and implemented in many DL frameworks (e.g., PyTorch and TensorFlow).

Problem: BP imposes a strong sequential dependency along layers during the gradient computations.

Key idea: We propose scaling BP by Parallel Scan Algorithm (BPPSA):
- Reformulate BP into a scan operation.
- Scaled by a customized parallel algorithm.

Key Results: $\Theta(\log n)$ vs. $\Theta(n)$ steps on parallel systems.
Up to $10^8 \times$ backward pass speedup ($\rightarrow 2.17 \times$ overall speedup).
BP’s Strong Sequential Dependency

\[ \nabla_{\tilde{x}} l = \left( \frac{\partial f(\tilde{x})}{\partial \tilde{x}} \right)^T \nabla f(\tilde{x}) l \]

Strong Sequential Dependency along layers.
Rethinking BP from an Algorithm Perspective

- Problems with strong sequential dependency were studied in the past (80’), but in a much simpler context.

- We propose scaling Back-Propagation by Parallel Scan Algorithm (BPPSA):
  - Reformulate BP as a scan operation.
  - Scale BP by a customized Blelloch Scan algorithm.
  - Leverage sparsity in the Jacobians.
What is a Scan\textsuperscript{1} Operation?

Binary, associative operator: $+ \\
\text{Identity: } \emptyset$

Input sequence: 
\begin{tabular}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{tabular}

Exclusive scan: 
\begin{tabular}{cccccccc}
0 & 1 & 3 & 6 & 10 & 15 & 21 & 28 \\
\end{tabular}

Compute partial reductions at each step of the sequence.

\textsuperscript{1}Blelloch, Guy E. "Prefix sums and their applications". Technical Report (1990)
Linear Scan

Step: executing the operator once.

Number of Elements (n)

Worker (p): an instance of execution; e.g., a core in a multi-core CPU

On a single worker: perform scan linearly; takes n steps.

With more workers: Can we achieve sublinear steps?
Blelloch Scan: Up-sweep Phase

Compute partial sums via a reduction tree.
Bleloch Scan: ② Down-sweep Phase

Parallel

Down-sweep

A
B
B
A+B

Combine partial sums across branches.

Time

22
1
2
3
4
5
6
7
8
10
11
15
26
0
10
11
10
5
10
7
21
28
Bleloch Scan: Efficiency

Logarithmic steps along the critical path. $2\log n$
Reformulate BP as a Scan Operation

Binary, associative operator: $+A \diamond B = BA$  
Identity: $\mathbb{I}$

Input sequence: $[7, 2, 3, 4, 5, 6, 7, 8]$

Exclusive scan: $[0, 7, 6, 5, 10, 15, 21, 28]$

Key Insight: matrix multiplication in BP is also binary & associative!
Scale BP by Blelloch Scan

Logarithmic steps along the critical path!

Matrix multiplications are noncommutative.
End-to-end Training Speedup

Training curve of BPPSA v.s. the baseline when batch size $B=16$, sequence length $T=1000$:

Numerical differences do not effect convergence.

2.17$\times$ speedup on the overall training time.
Sensitivity Analysis: Model Length

Sequence length (T) reflects the model length n.

BPPSA scales with the model length (n);

until being bounded by the number of workers (p).
Horizontally Fused Training Array:
An Effective Hardware Utilization Squeezer For Training Novel Deep Learning Models

Shang Wang$^{4,1,2}$, Peiming Yang$^{*3,2}$, Yuxuan Zheng$^{*5}$, Xin Li$^{*2}$, Gennady Pekhimenko$^{1,2}$

https://github.com/UofT-EcoSystem/hfta

MLSys 2021
Does Training Utilize the Hardware Well?
Hardware Resource Usage @

Monitored over 2 months: **51K** jobs, **472K** GPU hours.

**GPU Hour Usage Breakdown**

- **Single-GPU Training** 50%
- **Dist. Training** 24%
- **Other** 26%

**Single-GPU training:**
- **Dominates** the GPU hour usage.
Repetitive single-GPU training:
• Dominates the GPU hour usage.
• Concurrent jobs; same program; different configs.
• For hyper-param. tuning or convergence stability testing.
Repetitive single-GPU training:
• Often have **low hardware utilization**.

Why?
Why Hardware Underutilization?

DL

硬件

?  

?
Performance Optimization is **Hard**.

More so for **system & architecture “novices”**.

Increasing the batch size? 😞

Does **not** work universally:

- Generalization gap.
- Batch size scaling limit.

---

Keskar et al. On large-batch training for deep learning: Generalization gap and sharp minima. ICLR, 2017
Accelerators Get More Powerful.

Unoptimized workload $\rightarrow$ Harder to utilize well.
Why Hardware Underutilization?

Performance optimization is hard.

Accelerators get more powerful.

How to improve hardware utilization?
Train >1 Models on 1 Accelerator Simultaneously?

**Special features** for sharing among **arbitrary processes**.
(e.g., MPS and MIG on NVIDIA GPUs)

Less effective for repetitive training jobs.
Other accelerators (e.g., TPUs) do **not** possess such features.

What to do instead? 😐
Key Ideas
Model Similarity

Launched repetitively (e.g., hyper-parameter tuning)

Learning Rate$_1 = 0.01$

Learning Rate$_2 = 0.03$
Model Similarity

Launched repetitively (e.g., hyper-parameter tuning)

Learning Rate_1 = 0.01

Learning Rate_2 = 0.03
Model Similarity

Launched repetitively (e.g., hyper-parameter tuning)

Learning Rate\(_1\) = 0.01 \[\text{Linear} \rightarrow \text{ReLU} \rightarrow \text{Linear}\]

Learning Rate\(_2\) = 0.03 \[\text{Linear} \rightarrow \text{ReLU} \rightarrow \text{Linear}\]

Same types of ops.
Model Similarity

Launched repetitively (e.g., hyper-parameter tuning)

Learning Rate\(_1\) = 0.01

Learning Rate\(_2\) = 0.03

Same types of ops with the same shapes.
Launched repetitively (e.g., hyper-parameter tuning)

Same types of ops with the same shapes.

What to do with it?
Inter-model Horizontal Operator Fusion

We propose:

Learning Rate\(_1\) = 0.01
\[
\begin{align*}
\text{X} & \rightarrow \text{Linear} & \rightarrow \text{ReLU} & \rightarrow \text{Linear}
\end{align*}
\]

Learning Rates = \{0.01, 0.03\}
\[
\begin{align*}
\text{X} & \rightarrow \text{Fused Linear} & \rightarrow \text{Fused ReLU} & \rightarrow \text{Fused Linear}
\end{align*}
\]

Learning Rate\(_2\) = 0.03
\[
\begin{align*}
\text{X} & \rightarrow \text{Linear} & \rightarrow \text{ReLU} & \rightarrow \text{Linear}
\end{align*}
\]

But, DL stack → training **single** models on **separate** accelerators.

Who to
Implementation Reuse

Horizontally fused ops $\rightarrow$ other existing mathematically equivalent ops.

What about other ops?
Horizontally Fused Training Array (HFTA)

Different ops $\rightarrow$ different rules $\rightarrow$ tools required.
Horizontally Fused Training Array (HFTA)

We choose **PyTorch** for its *popularity*, but the idea is *general*.
Horizontally Fused Training Array (HFTA)

Support all DL framework’s hardware backends.

HFTA

PyTorch/XLA

V100
A100
TPU v3
HFTA Components

Operators
Optimizers
Learning Rate Schedulers

HFTA

PyTorch

PyTorch/XLA

V100
A100
TPU v3
HFTA: Fused Operators

Conv1d, Conv2d, ConvTranspose2d
Linear
MaxPool2d, AdaptiveAvgPool2d
Dropout, Dropout2d,
BatchNorm1d, BatchNorm2d,
LayerNorm,
Embedding
ReLU, ReLU6, Tanh,
LeakyReLU,
MultiheadAttention,
TransformerEncoderLayer

What else can we fuse? 😕
HFTA: Fused Optimizers and LR Schedulers

Adadelta, Adam
StepLR

Learning Rate  Gradients

What else can we fuse?
No Impact on Convergence

Mathematically Equivalent Transformations

Training ResNet-18 on CIFAR-10 with Three Different Learning Rates

What about training throughputs?
Methodology: Environment

Accelerators:
- RTX6000
- V100
- A100
- TPU v3

Versions:
- PyTorch
- NVIDIA CUDA
- cuDNN

Initial Experimental results

Repeating Experiments
Methodology: Baselines

One model per accelerator.

The **common practice** in hyper-param. tuning frameworks.
Methodology: Baselines

Some accelerators (e.g. NVIDIA GPUs) support running >1 processes.

Kernels are **time-multiplexed**.
Methodology: Baselines

Some accelerators (e.g. NVIDIA GPUs) support running >1 processes.

Co-run >1 kernels if a single kernel underutilizes the GPU.
Methodology: Baselines

Some accelerators (e.g. NVIDIA GPUs) support running >1 processes.

Slice (only) A100 into (≤ 7) partitions.
Methodology: Workloads

Not intensively optimized → realistically reflect novel DL research workloads.
V100 Results

Keep sharing with more models until OOM.

- Serial
- Concurrent
- MPS

**PointNet Classification on V100**

<table>
<thead>
<tr>
<th>Throughput</th>
<th>Number of Training Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2.5</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Throughput vs Number of Training Models**

- hfta:fp32
- serial:fp32
- mps:fp32
- concurrent:fp32
Fixed memory budget, HFTA co-trains more models than MPS and concurrent.
V100 Results

Same # of models & same GPU, HFTA achieves higher throughput than all baselines.

PointNet Classification on V100

Mixed Precision?
How About Mixed Precision?

HFTA can **better exploit** tensor cores during AMP training than all baselines.

![Graph showing PointNet Classification on V100](image)

Different GPUs? 🤔
How About Fancier GPUs?

Since $\text{Mem}(\text{A100}) > \text{Mem}(\text{RTX6000})$, HFTA can fit more models on A100.
How About TPUs?

HFTA achieves $4.93 \times$ over Serial.

Why is HFTA effective?
Performance Analysis: Memory

Why?
Performance Analysis: Memory

Overhead: (reserved by DL framework stacks)

MPS duplicates the overhead.

Any other reason why HFTA is effective?
Performance Analysis: Compute

- sm_active on A100
- sm_occupancy on A100
- tensor_active on A100
**Performance Analysis: Compute**

- **sm_active**: Fraction of cycles when SMs have resident warps.
- **sm_occupancy**: Ratio of # resident warps over SM’s max. # warps.
- **tensor_active**: Fraction of cycles when tensor cores are active.

Proxy metrics for different aspects of GPU utilization.

---

**sm_active on A100**

**sm_occupancy on A100**

**tensor_active on A100**
While MPS & MIG does improve utilization, HFTA is more effective!
More Results in the Paper

PointNet Segmentation, DCGAN, ResNet-18, MobileNet\textsubscript{V3Large}, Transformer, BERT\textsubscript{Medium}

- On GPUs, HFTA achieves:
  - $2.42 \times$ to $11.50 \times$ over Serial.
  - $1.25 \times$ to $4.72 \times$ over MPS.
  - $1.33 \times$ to $4.88 \times$ over MIG.
- On TPUs, HFTA achieves $2.98 \times$ to $15.13 \times$ over Serial.

HFTA’s Integration with hyper-parameter tuning algorithms.
- Reduce total GPU hour cost by up to $5.10 \times$.

Performance sensitivity study on partially fused ResNet-18.
**ECHO**: Compiler-based GPU Memory Footprint Reduction for LSTM RNN Training

Bojian Zheng\(^1,2\), Nandita Vijaykumar\(^1,3\), Gennady Pekhimenko\(^1,2\)
Background: Feature Maps

- Stashed data by the forward pass to compute the backward gradients

Large Temporal Gap between Usage

- The cause of high memory footprint in Convolutional Neural Networks (CNNs). [1, 2]

---


Background: LSTM RNN

• Long-Short-Term-Memory Recurrent Neural Network (LSTM RNN)
• Heavily adopted in sequence analysis (e.g., machine translation (NMT) & speech recognition (DeepSpeech2)).
• Its training is inefficient on the GPUs, especially when compared with CNN. [1, 2]

Why LSTM RNN Training is Inefficient?

Training throughput **saturates** as batch size increases.

**ResNet (CNN)**

**NMT (LSTM RNN)**

Training throughput is limited by the **memory capacity**.

**Memory capacity** limits the NMT training throughput.
GPU Memory Profiling Results

Feature maps dominate the GPU memory footprint.
Selective Recomputation

- **Key Idea**: Trade runtime with memory.

- The recomputation path should only involve lightweight operators.
Accurate Footprint Estimation

For each recomputation to be efficient, need to estimate its effect on the global footprint.

Selective Recomputation causes:
- increased memory footprint &
- performance degradation!

Example: $Z = \tanh(X + Y)$
Accurate Footprint Estimation

For each recomputation to be efficient, need to estimate its effect on the global footprint.

Selective Recomputation causes:

\( (+) \) feature maps: \( T^2N \rightarrow 2TN \)

Global Footprint Analysis:

1. shapes and types
2. reuse Challenging!

Example: \( Z_i = \tanh(X + Y_i), i \in [1, T] \)
Non-Conservative Overhead Estimation

For each recomputation to be efficient, need to estimate its effect on the runtime overhead.

Layer-Specific Property:
\[
\frac{dE}{dX} = \frac{dE}{dY} W \quad \text{&} \quad \frac{dE}{dW} = \frac{dE^T}{dY} X
\]
(NO Dependency on \(Y\))

Example:  \(Y = XW^T\)

- **Compute-Heavy**
  - 50% of the NMT training time
- Excluded in prior works
**ECHO**: A Graph Compiler Pass

- Integrated in the MXNet NNVM\(^1\) module
- Fully Automatic & Transparent
  - Requires NO changes in the training source code.
- Addresses the 2 key challenges of Selective Recomputation:
  1. Accurate Footprint Estimation
     - *Bidirectional Dataflow Analysis*
  2. Non-Conservative Overhead Estimation
     - *Layer Specific Optimizations*

\(^1\) https://github.com/apache/incubator-mxnet/tree/master/src/nnvm
**ECHO**: Bidirectional Dataflow Analysis

- **Storage Reuse**
  Causes ALL correlated operators to forward propagate simultaneously.

\[
\text{sizeof} \left( \sum \text{FeatureMaps}_{\text{new}} \right) \leq \text{sizeof} \left( \sum \text{FeatureMaps}_{\text{old}} \right)
\]

\[T^2 N \not\leq 2TN \]

Example: \(Z_i = \tanh(X + Y_i), i \in [1, T]\)
Evaluation: Benchmarks

Sockeye\textsuperscript{[1]}


- State-of-the-Art Neural Machine Translation Toolkit under MXNet
- Datasets:
  - IWSLT’15 English-Vietnamese (Small)
  - WMT’16 English-German (Large)
- Key Metrics:
  - Training Throughput
  - GPU Memory Consumption
  - Training Time to Validation BLEU Score
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td>Baseline System without Selective Recomputation</td>
<td></td>
</tr>
<tr>
<td><strong>Echo</strong></td>
<td>Compiler-based Automatic and Transparent Optimizations</td>
<td></td>
</tr>
</tbody>
</table>
**Echo**’s Effect on Memory and Performance

Small Dataset, Single-GPU Experiment

<table>
<thead>
<tr>
<th>Reduction Ratio</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mirror</td>
<td>High</td>
</tr>
<tr>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

2× Training Batch Size

- **Baseline** $B = 128$
- **Mirror** $B = 128$
- **Echo** $B = 128$
- **Echo** $B = 256$

11 GB Memory Capacity

- Throughput (samples/s)
- Memory Consumption (GB)

**DOUBLE WIN!**
**ECHO**’s Effect on Training Convergence

Large Dataset, Multi-GPU Experiment, Same Number of Training Steps

**ECHO** achieves:
- Same Validation BLEU Score
- Faster Convergence
- Fewer Compute Devices
Gist: Efficient Data Encoding for Deep Neural Network Training

In collaboration with Project Fiddle (MSR)
Relu -> Pool

Relu Backward Propagation

\[ dX = f(Y, dY) \]
\[ dx = y > 0 \ ? \ dy : 0; \]

Binarize – 1 bit representation (Lossless)
Relu/Pool -> Conv

Sparsity analysis on VGG16 (10 epochs)

Sparse Storage Dense Compute (Lossless)
Compression Ratio

Up to 2X compression ratio
With minimal performance overhead
Gist Summary

• Systematic memory breakdown analysis for image classification
• Layer-specific **lossless** encodings
  • Binarization and sparse storage/dense compute
• Aggressive lossy encodings
  • With delayed precision reduction
• Footprint reduction measured on real systems:
  • Up to 2X reduction with only 4% performance overhead
  • Further optimizations – more than 4X reduction
New Generation of Debugging/Prediction Tools

- **Daydream**: Accurately Estimating the Efficacy of Performance Optimizations for DNN Training (USENIX ATC’20)

- **Skyline**: Interactive In-editor Performance Visualizations and Debugging for DNN Training (UIST’20)

- **Habitat**: Prediction-guided Hardware Selection for Deep Neural Network Training (USENIX ATC’21)
Interactive In-editor Performance Visualizations and Debugging for DNN Training

Geoffrey X. Yu, Tovi Grossman, Gennady Pekhimenko
Tired of **not knowing** why your model is **slow** and/or **uses up so much memory**?
Tired of not knowing why your model is slow and/or uses up so much memory?

Does anyone have any detailed tips, walkthrus, or tutorials on how to profile PyTorch code running on the GPU?

I'm trying to optimize efficientnet and want to see exactly where the time is spent.

I've tried the following:
- use a c4.4xlarge, in cpu mode, instead of Mac OS X, in qt
- use an aws g2, in cuda mode => twice as fast as Mac laptop
- use an aws p2, in cuda mode => another 50% fast as p1

Now at this point, I'm not sure which bits are slow.

If it was a C++ script, that didn't use cuda, I might use either the debugger, stop it, and store the stacktrace. do this eg 5-10 lines tend to me in man yof the stacktraces => this is the bottleneck. If it was slick or deep, well pre-instrumented them with nvprofi. In pytorch cuda, I suppose I should use an nvidia profiler?

It's not clear to me which bits of the program are taking the time at a higher level than nvidia profiler probably. Thoughts on ideas for profiling PyTorch?
Skyline: Interactive In-editor Performance Debugging

• Key performance metrics (throughput, memory usage)

• Iteration run time and memory footprint breakdowns

• Interactive visualizations linked to batch size predictions

• Live and proactive performance debugging during development
Skyline: Interactive In-editor Performance Debugging

- Key performance metrics (throughput, memory usage)
- Iteration run time and memory footprint breakdowns
- Interactive visualizations linked to batch size predictions
- Live and proactive performance debugging during development
Skyline: Interactive In-editor Performance Debugging

- Key performance metrics (throughput, memory usage)
- Iteration run time and memory footprint breakdowns
- Interactive visualizations linked to batch size predictions
- Live and proactive performance debugging during development
Skyline: Interactive In-editor Performance Debugging

- Key performance metrics (throughput, memory usage)
- Iteration run time and memory footprint breakdowns
- **Interactive** visualizations linked to batch size predictions
- Live and proactive performance debugging during development
Skyline: Interactive In-editor Performance Debugging

• Key performance metrics (throughput, memory usage)
• Iteration run time and memory footprint breakdowns
• Interactive visualizations linked to batch size predictions
• Live and proactive performance debugging during development
Interactive visualizations tied to the code!
My Students: **EcoSystem Research Group**

- Hongyu Zhu (PhD)
- Bojian Zheng (PhD)
- Alexandra Tsvetkova (PhD)
- James Gleeson (PhD, co-advised)
- Anand Jayarajan (PhD)
- Mustafa Quraish (PhD)
- Shang (Sam) Wang (MSc)
- Jiacheng Yang (MASc)
- Pavel Golikov (MSc)
- Yaoyao Ding (MASc)
- Daniel Snider (MSc)
- Kevin Song (MASc)
- Yu Bo Gao (BSc)
- Kimberly Hau (BASc)
- Qingyuan Qie (BSc)
- Chenhao Jiang (BSc)
- Murali Andoorveedu (BASc)