TBD SUITE: BENCHMARKING AND PROFILING TOOLS FOR DNNS

Geoffrey X. Yu [1] **Hongyu Zhu** [1] **Anand Jayarajan** [2] **Bojian Zheng** [1] **Abhishek Tiwari** [1] **Gennady Pekhimenko** [1]

ABSTRACT

Understanding and debugging the training performance of deep neural networks (DNNs) is a difficult endeavor because it depends on many factors such as: (i) the compute utilization of the underlying hardware, (ii) memory capacity constraints, and (iii) the network bandwidth (for distributed training). To make matters worse, the lack of domain specific tools has made it difficult for non-experts to navigate this performance landscape. To help bridge this gap, we developed several DNN-focused performance analysis tools as a part of our work on the TBD Suite[a,b]—a memory profiler, network profiler, and a workflow for analyzing the computational performance characteristics associated with DNN training. In this demonstration, SysML'19 attendees will learn how to use these tools to diagnose performance problems and discover opportunities for performance improvements when training their models.

1 INTRODUCTION

As deep neural networks (DNNs) have become more widely used, there has been increasing interest in being able to perform DNN training efficiently both on a single accelerator (e.g., GPU or TPU) and in distributed configurations (both multiple machines and multiple accelerators per machine). However understanding and debugging DNN training performance is a difficult endeavor because it depends on many factors such as: (i) the compute utilization of the underlying hardware, (ii) memory capacity constraints, and (iii) the network bandwidth (for distributed training). To make matters worse, the lack of domain specific tools has made it difficult for non-experts to navigate this performance landscape.

To help bridge this gap we developed and open sourced[a] several DNN-focused performance analysis tools as a part of our work on the TBD Suite[b] (Zhu et al., 2018). We developed a memory profiler[c], network profiler (Jayarajan et al., 2019), and a workflow for analyzing the computational performance characteristics associated with DNN training. These tools and workflows are aimed at helping deep learning researchers and practitioners diagnose performance problems and discover opportunities for performance improvements in their DNNs. For example, we will show how our tools can be used to discover the training throughput differences between distinct models and how these insights ultimately lead to different optimization strategies. Overall, SysML'19 attendees will learn about our performance analysis workflow and will learn how they can use the TBD tools to uncover performance insights within their own models.

[1]Department of Computer Science, University of Toronto, Toronto, Ontario, Canada [2]Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada. Correspondence to: Geoffrey X. Yu <gxyu@cs.toronto.edu>, Gennady Pekhimenko <pekhimenko@cs.toronto.edu>.

[a]https://github.com/tbd-ai

[b]http://tbd-suite.ai

[c]http://tbd-suite.ai/tools-memory.html

2 TBD SUITE OVERVIEW

The TBD Suite is a collection of nine DNN benchmarks and a set of performance analysis tools for a number of DNN frameworks including TensorFlow (Abadi et al., 2016), MXNet (Chen et al., 2016), and PyTorch (Paszke et al., 2017). For this demonstration, we plan to show participants how they can use TBD's performance tools and TBD's MXNet memory and network profilers to perform training performance analysis on their DNNs.

3 DEMONSTRATION

In our demonstration we will run through our compute profiling workflow and will use the TBD memory and network profilers to analyze the performance of two DNNs in three distinct scenarios. We do not need any special equipment other than Wi-Fi access for our demo.

Live Action. We will run the TBD tools live on the scenarios described in Sections 3.1, 3.2, and 3.3. Demo participants will see step-by-step how the tools are used. Participants will also learn about our workflow in terms of how we gather and analyze performance data with our tools.

Interactivity. For each scenario, participants will have the opportunity to try using the tools for themselves using our laptops. Participants will be able to modify the DNN models themselves in each scenario and will be able to use the TBD profilers on these modified models to see how their training performance characteristics change.

3.1 Scenario 1: Initial Analysis with Nvprof

Nvprof is a common tool used to gather performance metrics and traces for applications that run on NVIDIA GPUs. With DNNs, however, naively running nvprof during training can lead to large output traces filled with many metrics—making the traces difficult to analyze. In this scenario we will show how to effectively use nvprof to gather performance insights from DNN training workloads.

We will show how to instrument the training code for ResNet-50 (He et al., 2016) on the ImageNet (Russakovsky et al., 2015) dataset using MXNet. We will run nvprof to demonstrate how to record runtime traces for a few training iterations and we will show how to view and interpret the outputted traces. Additionally, we will highlight and explain some of the key performance metrics for evaluating the performance of individual GPU kernels such as: (i) the floating point utilization, and (ii) the kernel's achieved occupancy.

3.2 Scenario 2: Performance Tuning with the Computation and Memory Profiler

The architecture of a DNN, as well as the hyperparameters chosen for training, can affect the training throughput. Figure 1 shows how the mini-batch size has implications on the training throughput and memory footprint (Zhu et al., 2018). In this scenario we will show how the TBD memory profiler combined with our computational profiling workflow can be used to tune a DNN model and its hyperparameters, such as the mini-batch size, to use the underlying hardware efficiently.

We will run the TBD memory profiler on ResNet-50 and a sequence to sequence (Seq2Seq) neural translation model (Sutskever et al., 2014), both using MXNet on the ImageNet and IWSLT'15 (Cettolo et al., 2015) datasets respectively. We will also walk through our computational analysis workflow to show how to gather performance metrics such as the GPU compute utilization. Using this information, we will demonstrate how these insights can be used to answer performance questions such as

- Does my DNN training process use my hardware accelerator efficiently?
- Which layers or data structures contribute the most to my DNN's memory footprint?
- What are the best opportunities to improve training performance?

3.3 Scenario 3: Distributed Training Analysis with the Network Profiler

Data parallelism (DP) (Dean et al., 2012) is the most commonly used partitioning approach for distributed DNN training. Prior work has shown that the effectiveness of DP is closely tied to the network bandwidth among the machines performing the training (Zhu et al., 2018). As a result, one challenge in applying DP is determining whether the DNN itself is well-suited for throughput speedups under a given (usually fixed) network bandwidth. In this scenario we will show how the TBD network profiler can be used to answer these types of questions.

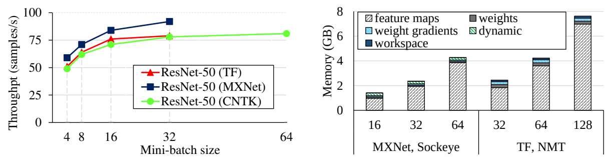

Figure 1. The training throughput and memory breakdown for the ResNet-50 and Seq2Seq models respectively.

We will run the TBD network profiler on a cluster of machines training ResNet-50 on ImageNet using the MXNet framework in a data parallel configuration. We will run the profiler several times while varying the cluster's network bandwidth. The profiler will generate performance traces that we will display—showing the communication overhead associated with DP for each layer for different network speeds (Jayarajan et al., 2019). We will demonstrate how participants can use these traces to determine (i) if there are layers that have a high communication overhead, (ii) whether DP leads to training throughput improvements, and if so, (iii) the minimum acceptable network bandwidth that results in training throughput improvements.

ACKNOWLEDGEMENTS

We thank members of the EcoSystem group for their feedback and the stimulating research environment they provide.

REFERENCES

Abadi, M. et al. TensorFlow: A system for large-scale machine learning. In *OSDI'16*, 2016.

Cettolo, M. et al. The IWSLT 2015 evaluation campaign. In *IWSLT'15*, 2015.

Chen, T. et al. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. In *NeurIPS Workshop on Machine Learning Systems*, 2016.

Dean, J. et al. Large scale distributed deep networks. In *NeurIPS'12*, 2012.

He, K. et al. Deep residual learning for image recognition. In *CVPR'16*, 2016.

Jayarajan, A. et al. Priority-based parameter propagation for distributed DNN training. In *SysML'19*, 2019.

Paszke, A. et al. Automatic differentiation in pytorch. In *NeurIPS Autodiff Workshop*, 2017.

Russakovsky, O. et al. ImageNet large scale visual recognition challenge. *Int. J. Comput. Vis. (IJCV)*, 2015.

Sutskever, I. et al. Sequence to sequence learning with neural networks. In *NeurIPS'14*, 2014.

Zhu, H. et al. Benchmarking and analyzing deep neural network training. In *IISWC'18*, 2018.