Benchmarking and Analyzing Deep Neural Network Training

Hongyu Zhu
University of Toronto
Toronto, Canada
serailhydra@cs.toronto.edu

Mohamed Akrout
University of Toronto
Toronto, Canada
makrout@cs.toronto.edu

Bojian Zheng
University of Toronto
Toronto, Canada
bojian@cs.toronto.edu

Andrew Pelegris
University of Toronto
Toronto, Canada
andrew.pelegris@mail.utoronto.ca

Anand Jayarajan
University of British Columbia
Vancouver, Canada
anandj@cs.ubc.ca

Amar Phanishayee
Microsoft Research
Redmond, United States
amar@microsoft.com

Bianca Schroeder
University of Toronto
Toronto, Canada
bianca@cs.toronto.edu

Gennady Pekhimenko
University of Toronto
Toronto, Canada
pekhimenko@cs.toronto.edu

***Abstract*—The recent popularity of deep neural networks (DNNs) has generated considerable research interest in performing DNN-related computation efficiently. However, the primary focus is usually very narrow and limited to (i) inference – i.e. how to efficiently execute already trained models and (ii) image classification networks as the primary benchmark for evaluation.**

Our primary goal in this work is to break this myopic view by (i) proposing a new benchmark suite for DNN *training*, called `TBD`[1], which comprises a representative set of eight DNN models and covers six major machine learning applications: image classification, machine translation, speech recognition, object detection, adversarial networks, reinforcement learning, and (ii) performing an extensive performance analysis of these models on three major deep learning frameworks (TensorFlow, MXNet, CNTK) across different hardware configurations (single-GPU, multi-GPU, and multi-machine). We present a new toolchain for performance analysis for these models that combines the targeted usage of existing performance analysis tools, careful selection of performance metrics, and methodologies to analyze the results. We also build a new set of tools for memory profiling in three major frameworks. These tools can shed light on precisely how much memory is consumed by different data structures (weights, activations, gradients, workspace) in DNN training. Using our tools and methodologies, we make several important observations and recommendations on where future DNN training research and optimization should be focused.

I. Introduction

The availability of large datasets and powerful computing resources has enabled a new type of artificial neural network—the deep neural network (DNNs [16], [47])—to solve hard problems such as image classification, machine translation, and speech processing [13], [44], [46], [56], [82], [85]. While this recent success of DNN-based learning algorithms has naturally attracted a lot of attention, the primary focus of researchers, especially in the systems and computer architecture communities is usually on *inference*—i.e. how to efficiently execute already trained models, and *image classification* (which is used as the primary benchmark to evaluate DNN computation efficiency).

[1]**TBD** is short for **T**raining **B**enchmark for **D**NNs

While inference is inarguably an important problem, we observe that efficiently *training* new models is becoming equally important as machine learning is applied to an ever growing number of domains, e.g., speech recognition [13], [87], machine translation [15], [61], [79], the automobile industry [19], [49], and recommendation systems [31], [45]. Researchers currently lack comprehensive benchmarks and profiling tools for DNN training. In this paper, we present a new benchmark for DNN training, called `TBD`, that uses a representative set of DNN models covering a broad range of machine learning applications: image classification, machine translation, speech recognition, adversarial networks, and reinforcement learning. `TBD` also incorporates an analysis toolchain for performing detailed resource and performance profiling of these models, including the first publicly available tool for profiling memory usage on major DNN frameworks. Using `TBD` we perform a detailed performance analysis on how these different applications behave on three DNN training frameworks (TensorFlow [8], MXNet [22], CNTK [89]) across different hardware configurations (single-GPU, multi-GPU, and multi-machine), and gain some interesting insights.

`TBD`'s benchmark suite and analysis toolchain is driven by the motivation to address three main challenges:

1. Training differs significantly from inference. The algorithmic differences between training and inference lead to differences in requirements for the underlying systems and hardware architecture. First, the *backward pass* and *weight updates*, operations, which are unique to training, and need to stash a large number of intermediate results in GPU memory, might require tens of gigabytes of main memory [72]. In contrast, the memory footprint of inference is much smaller, on the order of tens of megabytes [42]. Second, training usually proceeds in waves of *mini-batches*, a set of inputs grouped and processed in parallel [39], [88]. Throughput is thus the primary performance metric of concern in training, while inference is latency sensitive, but computationally less taxing.

2. Workload diversity. Deep learning has achieved state-of-the-art results in a very broad range of application domains,

	Image Classification Only	Broader (include non-CNN workloads)
Training	[27] [32] [34] [48] [54] [55] [72] [78] [83]	[8] [20] [51] [58] [67] [69] [86]
Inference	[10] [11] [12] [23] [26] [34] [36] [38] [54] [59] [60] [66] [70] [74] [75] [76] [78] [90] [91]	[8] [35] [41] [43] [53] [67]

TABLE I: The table above shows a categorization of major computer architecture and systems conference papers (SOSP, OSDI, NSDI, MICRO, ISCA, HPCA, ASPLOS) since 2014. These papers are grouped by their focus along two dimensions: Algorithmic and Application Breadth. There are more papers which optimize inference over training (25 vs. 16, 4 papers for both training and inference). Similarly, more papers use image classification as the *only* application for evaluation (26 vs. 11).

yet most existing evaluations of DNN performance remain narrowly focused on just image classification as their benchmark application, and convolutional neural networks (CNNs) remain the most widely-used models for systems/architecture researchers (Table I). Consequently, many important non-CNN models have not received much attention. The computational characteristics of image classification models are very different from some of these networks. Furthermore, given the rapid pace of innovation across the realms of algorithms, systems, and hardware related to deep learning, such benchmarks risk being quickly obsoleted if not maintained.

3. Identifying bottlenecks. There are several plausible candidates for the critical bottleneck of DNN training. Typical convolutional neural networks (CNNs) are usually computationally intensive, making *computation* one of the primary bottlenecks in single GPU training. Efficiently using modern GPUs (or other hardware accelerators) requires training with large mini-batches, which could be limited due to modern GPU *main memory* capacity. Training DNNs in a distributed environment with multiple GPUs and machines, brings with it yet another group of potential bottlenecks, *network and interconnect bandwidths*. Even for a specific model, setting up implementation and hardware for pinpointing whether performance is bounded by computation, memory, or communication is not easy due to the limitations of existing profiling tools.

Our paper makes the following contributions.

- **`TBD`, a new benchmark suite.** We create a new benchmark suite for DNN training that currently covers *six* major application domains and *eight* different state-of-the-art models. We have open-sourced our benchmark suite and intend to continually expand it with new applications and models based on feedback and support from the community.
- **Tools to enable end-to-end performance analysis.** We develop a toolchain for end-to-end analysis of DNN training. As part of the toolchain, we also build new memory profiling tools for the major DNN frameworks we considered: TensorFlow [8], MXNet [22], and CNTK [89].
- **Findings and Recommendations.** Using our benchmark suite and analysis tools, we make several important observations and recommendations on where to focus future research and optimization of DNNs. These observations suggest several interesting research directions, including efficient RNN layer implementations and memory footprint reduction optimizations focused on feature maps for various models.

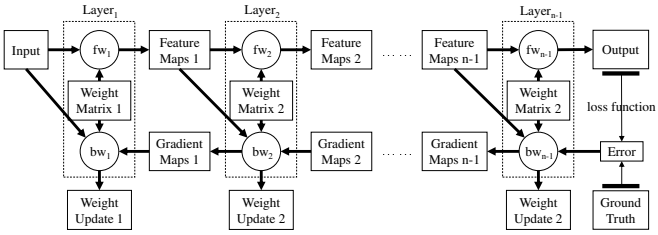


Fig. 1: Feed-forward and Back-propagation

II. Background

A. Deep Neural Network Training

A neural network can be seen as a function which takes data samples as inputs, and outputs certain properties of the input samples (Figure 1). Neural networks are composed of a series of neuron layers (e.g. fully-connected, convolutional, pooling, recurrent, etc). Each layer is associated with its own set of *weights*, and applies some mathematical transformation to its input and weights, producing intermediate results for its downsteam layers. The intermediate results generated by each layer are often called *feature maps* (or *activations*).

The goal of *training* is to find optimal weight values for each layer, so that the network as a whole can produce the desired outputs. Training a neural network is an iterative algorithm, where each iteration consists of a *forward* pass and a *backward* pass. Since the backward pass requires feature maps generated in the forward pass, they must be stashed in memory temporarily, mandating great GPU memory capacity [50].

Since modern training datasets are extremely large, it is impractical to use the entire set of the training data in each iteration. Instead, a training iteration randomly samples a *mini-batch* from the full dataset, and uses it as input [57].

Fully training a DNN is extremely time-consuming, but using modern GPUs can significantly reduce the training time (e.g. months to days). One way to further speed up neural network training is to parallelize the training procedure. A simple and effective approach is *data parallelism* [33], where the full dataset is partitioned into n subsets, one for each worker. Each worker trains a single network replica with its own subset of data respectively, periodically exchanging weight updates with all other workers.

B. DNN Frameworks and Low-level Libraries

DNN frameworks and low-level libraries are designed to simplify the life of ML programmers and to help them efficiently utilize existing complex hardware. A DNN framework (e.g., TensorFlow or MXNet) usually provides users with portable APIs to define computation logic, transforming the user program into an internal intermediate representation (e.g.,

dataflow graph representation [8], [17], [22]), which then becomes a basis for backend execution, including data transfers, memory allocations, and low-level CPU function calls or GPU kernel[2] invocations. Such low-level functions are usually provided by libraries such as cuDNN [25], cuBLAS [4], MKL [84], and Eigen [6], which provide efficient implementations of basic vector and multi-dimensional matrix operations (some operations are NN-specific such as convolutions or poolings) in C/C++ (CPU) or CUDA (GPU).

III. Methodology

A. Application and Model Selection

Based on a careful survey of existing literature and in-depth discussions with machine learning researchers and industry developers at several institutions (Google, Microsoft, and Nvidia), we identified an initial diverse set of interesting application domains, where deep learning has been emerging as the most promising solution: image classification, object detection, machine translation, speech recognition, generative adversarial nets, and deep reinforcement learning. Table II summarizes the models and datasets we chose for each application domain. When selecting the models, our emphasis has been on picking the most recent models capable of producing state-of-the-art results (rather than, for example, classical models of historical significance). The reasons are that these models are the most likely to serve as foundations for the development of future algorithms and also that they often use new types of layers not present in older models, with new resource profiles. Moreover, the design of models is often constrained by hardware limitations, which will have changed since the introduction of older models.

1) Image Classification: Image classification is the archetypal deep learning application, being the first domain where a deep neural network (AlexNet [56]) proved to be a watershed, beating all prior traditional methods. In our work, we use two very recent models, Inception-v3 [80] and Resnet [44], which are broadly similar to AlexNet, but improve accuracy through novel improvements that enable extremely deep networks.

2) Object Detection: Object detection applications, such as face detection, are another popular deep learning application and can be considered an extension of image classification, where an algorithm usually first breaks down an image into regions of interest and then applies image classification to each region. We choose to include Faster R-CNN [71], which achieves state-of-the-art results on the Pascal VOC datasets [37]. The convolution stack in a Faster R-CNN network is usually a standard image classification network, in our work a 101-layer ResNet.

3) Machine Translation: Unlike image processing, machine translation involves the analysis of sequential data and typically relies on RNNs using LSTM cells as its core model. We select *NMT* [85] and *Sockeye* [46], developed by *TensorFlow* and *Amazon Web Service* teams, respectively, as representative RNN-based models in this area. We also include an implementation of the recently introduced [82] *Transformer* model, which achieves a new state-of-the-art in translation quality using attention layers as an alternative to recurrent layers.

4) Speech Recognition: Deep Speech 2 [13] is an end-to-end speech recognition model from *Baidu Research*. It is able to accurately recognize both English and Mandarin Chinese, two very distant languages, with a unified model architecture and shows great potential for deployment in industry. The Deep Speech 2 model contains two convolutional layers, plus seven regular recurrent layers or Gated Recurrent Units (GRUs), different from the RNN models in machine translation included in our benchmark suite, which use LSTM layers.

5) Generative Adversarial Networks: A generative adversarial network (GAN) trains two networks, one generator network and one discriminator network. The generator is trained to generate data samples that mimic the real samples, and the discriminator is trained to distinguish whether a data sample is genuine or synthesized. While GANs are powerful generative models, training a GAN suffers from instability. The WGAN [14] is a milestone as it makes great progress towards stable training. Recently Gulrajani et al. [40] proposes an improvement based on the WGAN to enable stable training on a wide range of GAN architectures. We include this model into our benchmark suite as it is one of the leading DNN algorithms in the area of unsupervised learning.

6) Deep Reinforcement Learning: Deep neural networks are also responsible for recent advances in reinforcement learning, which has contributed to the creation of the first artificial agents to achieve human-level performance across challenging domains, such as the game of Go and various classical computer games. We include the *A3C* algorithm [62] in our benchmark suite, as it has become one of the most popular deep reinforcement learning techniques, surpassing the DQN training algorithms [63], and works in both single and distributed machine settings. A3C relies on asynchronously updated policy and value function networks trained in parallel over several processing threads.

B. Framework Selection

There are many open-source DNN frameworks, such as TensorFlow [8], Theano [17], MXNet [22], CNTK [89], Caffe [52], Chainer [81], Torch [30], Keras [28], PyTorch [68]. Since no single framework has emerged as the dominant leader in the field and since different framework-specific design choices and optimizations might lead to different results, we include several frameworks in our work. In particular, we

[2] A GPU kernel is a routine that is executed by an array of CUDA threads on GPU cores.

[a] We use the convolution stack of ResNet-101 to be the shared convolution stack between Region Proposal Network and the detection network.

[b] The official Deep Speech 2 model has 2 convolutional layers plus 7 RNN layers. Due to memory issue, we use the default MXNet configuration which has 5 RNN layers instead.

[c] Both the WGAN generator and discriminator are 4 residual block CNNs.

[d] We use the train+val set of Pascal VOC 2007 dataset.

[e] The entire LibriSpeech dataset consists of 3 subsets with 100 hours, 360 hours and 500 hours respectively. By default, the MXNet implementation uses the 100-hour subset as the training dataset.

Application	Model	Number of Layers	Dominant Layer	Implementations	Dataset
Image classification	ResNet-50 [56] Inception-v3 [80]	50 (152 max) 42	CONV	TensorFlow, MXNet, CNTK	ImageNet1K [73]
Machine translation	Seq2Seq [79] Transformer [82]	5 12	LSTM Attention	TensorFlow, MXNet TensorFlow	IWSLT15 [21] WMT-14 [18]
Object detection	Faster R-CNN [71]	101[a]	CONV	TensorFlow, MXNet	Pascal VOC 2007 [37]
Speech recognition	Deep Speech 2 [13]	9[b]	RNN	MXNet	LibriSpeech [64]
Adversarial learning	WGAN [40]	14+14[c]	CONV	TensorFlow	Downsampled ImageNet [29]
Deep reinforcement learning	A3C [62]	4	CONV	MXNet	Atari 2600

TABLE II: Suite Overview: models and datasets used, major layer types and counts, and frameworks with implementations.

Dataset	Number of Samples	Size	Special
ImageNet1K	1.2million	3x256x256 per image	N/A
IWSLT15	133k	20-30 words long per sentence	vocabulary size of 17188 (English to Vietnamese)
WMT-14	4.5million	up to 50 words (most sentences)	vocabulary size of 37000 (English to German)
Pascal VOC 2007	5011[d]	around 500x350	12608 annotated objects
LibriSpeech	280k	1000 hours[e]	N/A
Downsampled ImageNet	1.2million	3x64x64 per image	N/A
Atari 2600	N/A	4x84x84 per image	N/A

TABLE III: Training Datasets

choose TensorFlow [8], MXNet [22], and CNTK [89], as all three platforms have a large number of active users, are actively evolving, have many of the implementations for the models we are interested in[3], and support hardware acceleration using single and multiple GPUs.

C. Training Benchmark Models

To ensure that the results we obtain from our measurements are representative, we need to verify that the training process for each model results in classification accuracy comparable to state of the art results published in the literature. To achieve this, we train the benchmark models in our suite until they converge to some expected accuracy rate (based on results from the literature).

Figure 2 shows the classification accuracy observed over time for four representative models in our benchmark suite, *Inception-v3*, *ResNet-50*, *Seq2Seq*, and *A3C*, when trained on the single Quadro P4000 GPU hardware configuration described in Section IV. We observe that the training outcome of all models matches results in the literature. For the two image classification models (*Inception-v3* and *ResNet-50*), the Top-1 classification accuracy reaches 75–80% and the the Top-5[4] accuracy is above 90%, both in agreement with previously reported results for these models [44]. The accuracy of the machine translation models is measured using the BLEU score [65] metric, and we trained ours to achieve a BLEU score of around 20. For reinforcement learning, since the models are generally evaluated by Atari games, the accuracy of the A3C model is directly reflected by the score of the corresponding game. The A3C curve we show in this figure is from the Atari Pong game and matches previously reported results for that game (19–20) [62]. The training curve shape for different implementations of the same model on different frameworks can vary, but most of them usually converge to similar accuracy at the end of training.

[3]Note that implementing a model on a new framework from scratch is a highly complex task beyond the scope of our work. Hence in this paper we use the existing open-source implementations provided by either the framework developers on the official github repository, or third-party implementations when official versions are not available.

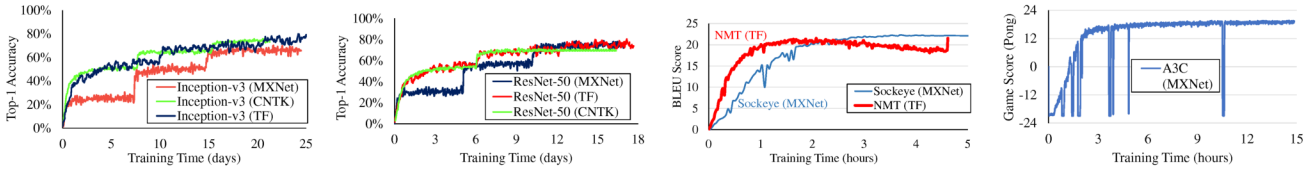
[4]In the Top-5 classification the classifier can select up to 5 top prediction choices, rather than just 1.

D. Performance Analysis Framework and Tools

In this section, we describe our analysis toolchain, which is designed to help us understand for each benchmark, where the training time goes, how well hardware resources are utilized and how to efficiently improve training performance.

1) Making implementations comparable across frameworks: Implementations of the same model on different frameworks might vary in a few aspects that can impact performance profiling results. Different implementations might have different hard-coded values for key hyper-parameters (e.g., learning rate, momentum, dropout rate, weight decay) in their code. To make sure that benchmarking identifies model-specific performance characteristics, rather than just implementation-specific details, we adapt implementations of the same model to make them comparable across frameworks. We also ensure that they define the same network, i.e., the same types and sizes of corresponding layers and layers are connected in the same way. Moreover, we make sure that the key properties of the training algorithm are the same across implementations. This is important for models, such as Faster R-CNN [71], where there are four different ways in which the training algorithm can share the internal weights.

2) Accurate and time-efficient profiling via sampling: The training of a deep neural network can take days or even weeks, making it impractical to profile the entire training process. Fortunately, as the training process is an iterative algorithm and almost all the iterations follow the same computation logic, we find that accurate results can be obtained via sampling only for a short training period (on the order of minutes) out of the full training run. In our experiments, we sample 50-1000



(a) Inception-v3 (b) ResNet-50 (c) Seq2Seq (d) A3C

Fig. 2: Model accuracy during training for selected models.

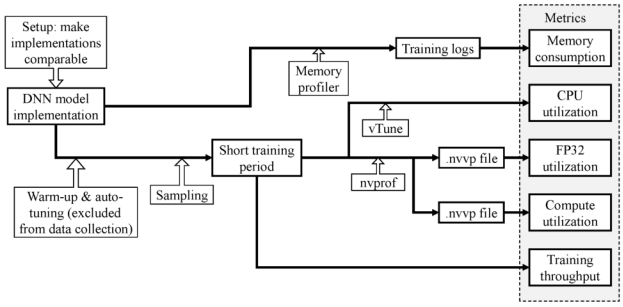


Fig. 3: Performance Analysis Toolchain

iterations (depending on the model) as the basis for the metrics of interest.

To obtain representative results, care must be taken when choosing the sample interval to ensure that the training process has reached stable state. Upon startup, a typical training procedure first goes through a warm-up phase (initializing the data flow graph, allocating memory and loading data) and then spends some time auto-tuning various parameters (e.g., system hyper-parameters, such as matrix multiplication algorithms, and workspace size). Only after that does the system enter the stable training phase for the rest of execution. While systems do not explicitly indicate when they enter the stable training phase, our experiments show that the warm-up and auto-tuning phase can be easily identified in measurements. We see that throughput stabilizes after several hundred iterations (a few thousand iterations in the case of Faster R-CNN). The sample time interval is then chosen after throughput has stabilized.

3) Relevant metrics: Below, we describe the metrics we collect as part of the profiling process.

• *Throughput:* Advances in deep neural networks have been tightly coupled to the availability of compute resources capable of efficiently processing large training data sets. As such, a key metric when evaluating training efficiency is the number of input data samples being processed per second, a metric we refer to as *throughput*. Throughput is particularly relevant to DNN training, which, unlike inference, is not latency sensitive.

For the speech recognition model, we slightly modify our definition of throughput. Due to the large variations in lengths among the audio data samples, we use the total duration of audio files processed per second instead of the number of files. The lengths of data samples also varies for machine translation models, but the throughput of these models is still stable, so we use the throughput determined by simple counting for them.

• *GPU Compute Utilization:* The GPU is the workhorse behind DNN training, as it is the unit responsible for executing the key operations involved in DNN training (broken down into basic operations, such as vector and matrix operations). Therefore, for optimal throughput, the GPU should be busy all the time. Low utilization indicates that throughput is limited by other resources, such as CPU or data communication, and further improvement can be achieved by overlapping CPU runtime or data communication with GPU execution.

We define GPU Compute Utilization as the fraction of time that the GPU is busy (i.e., at least one of its typically many cores is active):

$$\text{GPU utilization} = \frac{\text{GPU active time} \times 100}{\text{total elapsed time}}\% \quad (1)$$

• *FP32 utilization:* We also look at GPU utilization from a different angle, measuring how effectively the GPU's resources are being utilized *while the GPU is active*. More specifically, the training of DNNs is typically performed using single-precision floating point operations (FP32), so a key metric is how well the GPU's compute potential for doing floating point operations is utilized. We compare the number of FP32 instructions the GPU actually executes while it is active to the maximal number of FP32 instructions it can theoretically execute during this time, to determine what percentage of its floating point capacity is utilized. More precisely, if a GPU's theoretical peak capacity across all its cores is $FLOPS_{peak}$ single-precision floating point operations per second, we observe the actual number of floating point operations executed during a period of T seconds that the GPU is active, to compute *FP32 utilization* as follows:

$$\text{FP32 utilization} = \frac{\text{actual flop count during T} \times 100}{FLOPS_{peak} \times T}\% \quad (2)$$

The FP32 utilization gives us a way to calculate the theoretical upper bound of performance improvements one could achieve by a better implementation. For example, an FP32 utilization of 50% indicates that we can increase throughput by up to 2x if we manage to achieve FP32 utilization of 100%.

In addition to looking at the aggregate FP32 utilization across all cores, we also measure the per-core FP32 utilization for individual kernels, to identify the kernels with long duration, but low utilization. These kernels should be optimized with high priority.

• *CPU utilization:* While most training work is typically performed on the GPU, the CPU is also involved, for example, by executing framework frontends, launching GPU kernels,

and transferring data between the CPU and GPU. We report CPU utilization as the average utilization across all cores:

$$\text{CPU utilization} = \frac{\sum_c \text{total active time of core c} \times 100}{\text{CPU core count} \times \text{total elapsed time}}\% \quad (3)$$

• *Memory consumption:* In addition to compute cycles, the physical memory capacity has become a limiting factor in training big DNNs. In order to optimize memory usage during DNN training, it is important to understand where memory goes, i.e., what data structures occupy most of the memory. Unfortunately, there are no open-source tools currently available for existing frameworks that can provide this analysis. Hence we build our own memory profilers for three main frameworks (TensorFlow, MXNet, and CNTK). *We have open-sourced our memory profiling tool for MXNet, and will release tools for other frameworks in the future.* We expect them to be useful to others in developing and analyzing their models.

When building our memory profiler, we carefully inspect how the different DNN frameworks in our benchmark allocate their memory and identify the data structures that are the main consumers of memory. We observe that most data structures are allocated before the training iterations start for these three frameworks. Each of the data structures usually belongs to one of the three types: weights, weight gradients and feature maps (similarly to prior works [72]). These data structures are allocated statically. In addition, a framework might allocate some workspace as a temporary container for intermediate results in a kernel function, which gives us another type of data structure. The allocation of workspace can be either static, before the training iterations, or dynamic, during the training iterations. We observe that in MXNet, data structures other than workspace are allocated during the training iterations (usually for the momentum computation) as well. We assign these data structures to a new type called "dynamic". As memory can be allocated and released during the training, we measure the memory consumption by the maximal amount of memory ever allocated for each type.

IV. Evaluation

In this section, we use the methodology and frameworks described in the previous section for a detailed performance evaluation and analysis of the models in TBD benchmark suite.

A. Experimental Setup

We use Ubuntu 16.04 OS, TensorFlow v1.3, MXNet v0.11.0, CNTK v2.0, with CUDA 8 and cuDNN 6. All of our experiments are carried out on an 8-machine cluster, where each node is equipped with a Xeon E5-2680 28-core CPU and one to four NVidia Quadro P4000 GPUs (connected with 128 Gbps PCIe). Machines are connected with both Ethernet (1 Gbps) and high speed InfiniBand (100 Gbps) network cards.

As different GPU models provide a tradeoff between cost, performance, area and power, it is important to understand how different GPUs affect the key metrics in DNN training. We therefore also repeat a subset of our experiments using a second type of GPU, the NVidia TITAN Xp GPU. Table IV compares the technical specifications of the two GPUs in our work. We compare our metrics between TITAN Xp and P4000 in Section IV-C.

	TITAN Xp	Quadro P4000	Xeon E5-2680
Multiprocessors	30	14	
Core Count	3840	1792	28
Max Clock Rate (MHz)	1582	1480	2900
Memory Size (GB)	12	8	128
LLC Size (MB)	3	2	35
Memory Bus Type	GDDR5X	GDDR5	DDR4
Memory BW (GB/s)	547.6	243	76.8
Bus Interface	PCIe 3.0	PCIe 3.0	
Memory Speed (MHz)	5705	3802	2400

TABLE IV: Hardware specifications

B. Performance Analysis

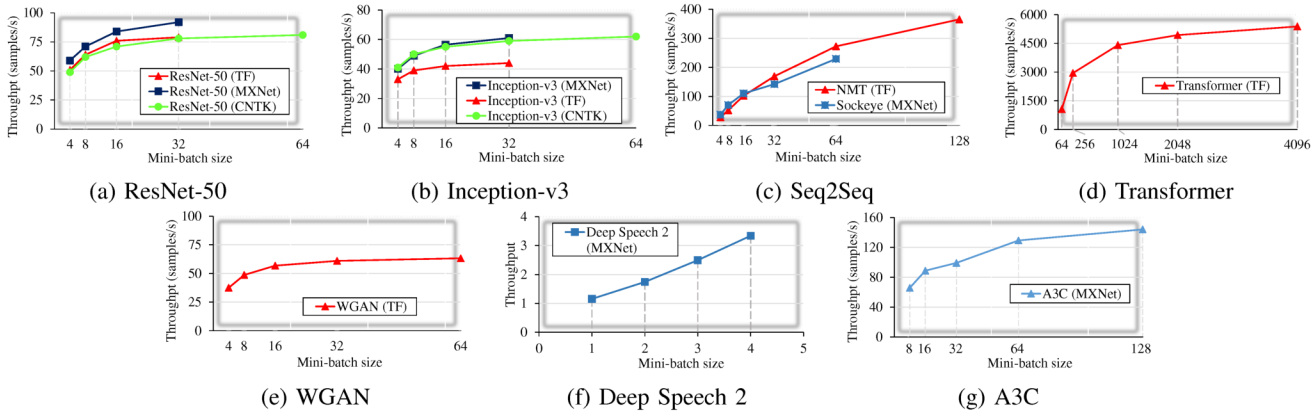
As previously explained, our analysis focuses on a set of key metrics: throughput, GPU and CPU compute utilization, FP32 utilization, as well as the memory consumption breakdown. We pay particular attention to how the above metrics vary across applications, models and frameworks.

Moreover, we use our setup to study the effects of a key hyper-parameter, the mini-batch size, on our metrics. It has been shown that to achieve high training throughput with the power of multiple GPUs using data parallelism, one must increase the mini-batch size, and additional work needs to be done on model parameters such as learning rate to preserve training accuracy [39], [88]. In the single-GPU case, it is often assumed that larger mini-batch sizes translate to higher GPU utilization, but the exact effects of varying mini-batch size are not well understood. In this work, we use our setup to quantify in detail how mini-batch size affects key performance metrics.

1) Throughput: Figure 4 shows the average training throughput for models from the TBD suite when varying the mini-batch size (the maximum mini-batch size is bounded by the GPU memory capacity). For Faster R-CNN, the number of images processed per iteration is fixed to be just one on a single GPU, hence we do not present a separate graph for Faster R-CNN. Both TensorFlow and MXNet implementations achieve a throughput of 2.3 images per second for this model. We make the following three observations from this figure.

Observation 1: Performance increases with mini-batch size for all models. As expected, the larger the mini-batch size, the higher the throughput for all models we study. We conclude that to achieve high training throughput on a single GPU, one should use reasonably large mini-batches, especially for non-convolutional models. We explain this behavior as we analyze the GPU and FP32 utilization metrics later in this section.

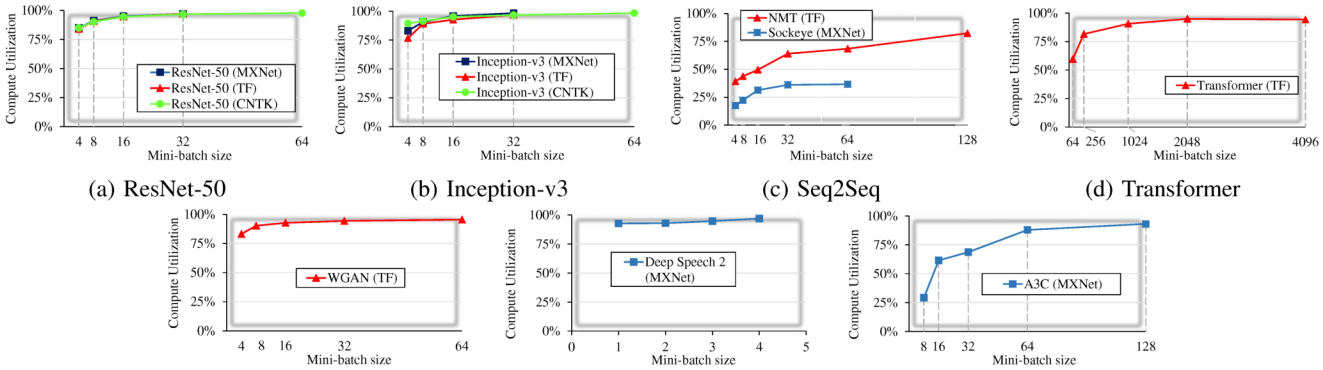
Observation 2: The performance of RNN-based models is not saturated within the GPU's memory constraints. The relative benefit of further increasing the mini-batch size differs vastly among different applications. For example, for the *NMT* model increasing mini-batch size from 64 to 128 increases training throughput by 25%, and the training throughput of Deep Speech 2 scales almost linearly. These two models' throughput (and hence performance) are essentially limited by



(a) ResNet-50 (b) Inception-v3 (c) Seq2Seq (d) Transformer

(e) WGAN (f) Deep Speech 2 (g) A3C

Fig. 4: DNN training throughput for different models on multiple mini-batch sizes.



(a) ResNet-50 (b) Inception-v3 (c) Seq2Seq (d) Transformer

(e) WGAN (f) Deep Speech 2 (g) A3C

Fig. 5: GPU compute utilization for different models on multiple mini-batch sizes.

the GPU memory capacity and we do not see any saturation point for them while increasing the mini-batch size. In contrast, other models also benefit from higher mini-batch sizes, but after a certain *saturation point* these benefits are limited. For example, in the *Inception-v3* model, going from batch size of 16 to 32 yields less than 10% in throughput improvement for implementations on all three frameworks.

Observation 3: Application diversity is important in comparing framework performance. We find that the results when comparing performance of models on different frameworks can greatly vary for different applications, and thus, it is crucial to use a diverse set of applications in framework comparisons. For example, we observe that for image classification, the *MXNet* implementations of both models (*ResNet-50* and *Inception-v3*) perform generally better than the corresponding *TensorFlow* implementations, but for machine translation, the *TensorFlow* implementation of *Seq2Seq* (*NMT*) performs significantly better than its *MXNet* counterpart, (*Sockeye*). TensorFlow also utilizes GPU memory better than MXNet for Seq2Seq models and can be trained with a maximum mini-batch size of 128, while MXNet can only be trained with a maximum of 64 (both limited by 8GB GPU memory). For the same memory budget, it allows TensorFlow to achieve higher throughput, 365 samples per second, vs. MXNet's 229 samples per second. We conclude that there is significant variance in how different frameworks perform on different models, making it vital to study a *diverse* set of applications and models as we propose in our benchmark pool.

2) GPU Compute Utilization: Figure 5 shows the GPU compute utilization, i.e. the amount of time GPU is busy running some kernels (as formally defined by formula 3) for different benchmarks as we vary mini-batch size. Again, for *Faster R-CNN*, only a batch of one is possible, and the TensorFlow implementation achieves a relatively high compute utilization of 89.4%, while the *MXNet* implementation achieves 90.3%. We make the following two observations from this figure.

Observation 4: Mini-batches should be big enough to keep the GPU busy. Similar to observation 1 about throughput, larger mini-batch sizes mean longer individual GPU kernel durations and better the GPU compute utilization, as the GPU spends more time doing computation rather than invoking and finishing small kernels. While larger batches also increase the data transfer overhead, our results show that this overhead is usually efficiently parallelized alongside computation.

Observation 5: GPU compute utilization is low for LSTM-based models. Non-RNN models and *Deep Speech 2*, which uses regular RNN cells (not LSTM) usually reach very high utilization (95% or higher) with large batches. Unfortunately, LSTM-based models (*NMT*, *Sockeye*) cannot drive up GPU utilization significantly, even at maximum batch sizes. Consequently, in general, these models do not utilize available GPU

hardware resources well, and further research should be done to optimize LSTM cells on GPUs. Moreover, it is important to note that the low compute utilization problem applies to the layer type, not the application – the *Transformer* model also used in machine translation does not suffer from low compute utilization as it uses *Attention* layers without recurrence.

3) GPU FP32 utilization: Figure 6 shows GPU FP32 utilization (formally defined by equation 2 in Section III) for different benchmarks as we change the mini-batch size (as far as memory capacity permits). For Faster R-CNN, the MXNet/TensforFlow implementations achieve an average utilization of 70.9%/58.9% correspondingly. We make three major observations from this figure.

Observation 6: Mini-batches should be large enough to exploit the FP32 computational power of GPU cores. As expected, we note that using larger batches also improves GPU FP32 utilization for all benchmarks we study. We conclude that both FP32 utilization (Observation 6) and GPU utilization (Observation 4) are key contributors to the increase in overall throughput with greater mini-batch size (Observation 1).

Observation 7: RNN-based models have low GPU FP32 utilization. Even with the maximum mini-batch size possible (on a single GPU), the GPU FP32 utilization of the two RNN-based models (*Seq2Seq* and *Deep Speech 2*, Figure 6c and Figure 6f, respectively) are much lower than for other non-RNN models. This clearly indicates the potential of designing more efficient RNN layer implementations used in TensforFlow and MXNet, and we believe further research should be done to understand the sources of these inefficiencies. Together with Observation 5 (low GPU utilization for LSTM-based models), this observation explains why in Observation 2, we do not observe throughput saturation for RNN-based models, even for very large mini-batches.

A major characteristic of the RNN-based models is the relatively small granularity of layer sizes and kernels. These layers are mostly fully connected layers, which usually contain more number of parameters and produce larger activations. On the contrast, the number of FLOPS required to compute one fully-connected layer is generally much less than CNN layers. This could be one of the reasons why saturating the training throughput of RNN models by simply increasing mini-batch size requires much larger GPU memory capacity.

Observation 8: There exist kernels with long duration, but low FP32 utilization even for highly optimized models. The previous observation might have raised the question of why even extremely optimized CNN models exhibit such low average FP32 utilization. In this observation, we provide an answer: Different kernels exhibit greatly varying FP32 utilization, and even optimized models have long-running kernels with low utilization. Table V and Table VI show the five most important kernels with the FP32 utilization *lower than average* (for *ResNet-50* model on TensorFlow and MXNet). We observe that cuDNN batch normalization kernels (having *bn* in their names) are the major source of inefficiency, with FP32 utilization more than 20% below the average. This is true for implementations on different frameworks. These kernels are top candidates for acceleration to achieve further progress in improving DNN training performance on GPUs.

Duration	Utilization	Kernel Name
8.36%	30.0%	magma_lds128_sgemm_kernel...
5.53%	42.3%	cudnn::detail::bn_bw_1C11_kernel_new...
4.65%	46.3%	cudnn::detail::bn_fw_tr_1C11_kernel_new...
3.12%	20.0%	Eigen::internal::EigenMetaKernel...
2.48%	40.0%	tensorflow::BiasNHWCKernel...

TABLE V: Longest 5 kernels with utilization level below the average (ResNet-50, mini-batch size 32, TensorFlow)

Duration	Utilization	Kernel Name
9.43%	40.7%	cudnn::detail::bn_bw_1C11_kernel_new...
7.96%	44.0%	cudnn::detail::bn_fw_tr_1C11_kernel_new...
5.14%	10.0%	cudnn::detail::activation_bw_4d_kernel...
3.52%	10.0%	cudnn::detail::activation_fw_4d_kernel...
2.85%	25.4%	_ZN5mxnet2op8mxnet_op20mxnet_generic_ kernel...

TABLE VI: Longest 5 kernels with FP32 utilization below the average (ResNet-50, mini-batch size 32, MXNet)

C. Hardware Sensitivity

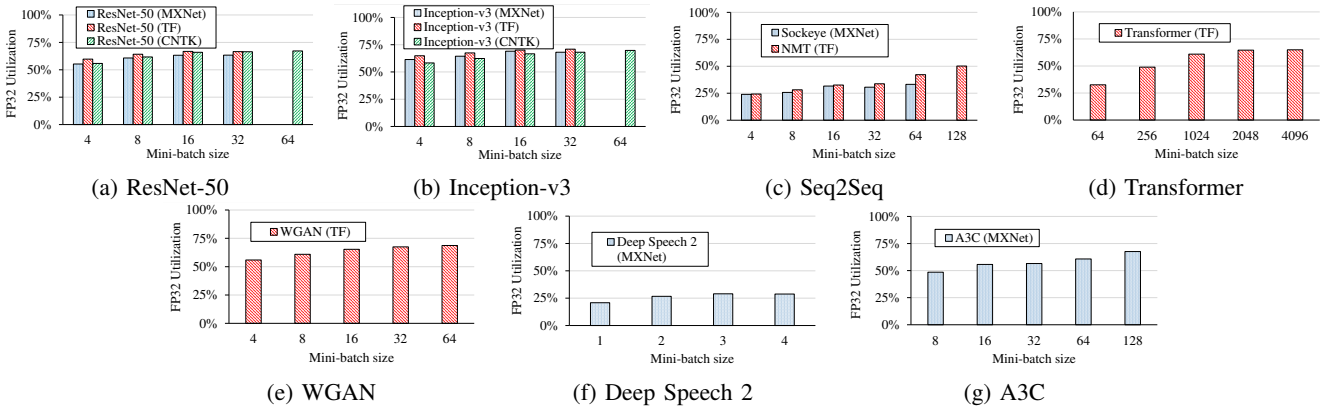
The results presented so far were based on experiments with the Quadro P4000 GPU. In this section, we are interested in seeing how the performance of DNN training depends on the hardware used. To this end, we apply our analysis workflow to our benchmark models on the more powerful TITAN Xp GPU. Figure 7 shows the comparison between these two generations of GPUs. For the throughput comparison, we normalized each model result to the throughput of the less powerful P4000 card. We make the following observation from this figure.

Observation 9: More advanced GPUs should be accompanied by better systems designs and more efficient libraries. The TITAN Xp usually helps improve training throughput (except for *Sockeye*), however the computation power of the TITAN Xp is not generally well-utilized. Both the GPU and FP32 utilization of the TITAN Xp appear to be worse than those of the P4000 for most of our models. Thus, we conclude that although the TITAN Xp is more computationally powerful (more multiprocessors, CUDA cores, and bandwidth, see Table IV), properly harnessing its resources requires a more careful design of existing GPU kernel functions, libraries (e.g., cuDNN), and algorithms.

D. Memory Profiling

As we have previously shown, DNN training throughput (and hence, performance) can be significantly bottlenecked by the available GPU memory. Figure 8 shows the result of our analysis, where the memory usage is separated into five categories: weights, gradient weights, feature maps, dynamic, and workspace. Where appropriate, we vary the mini-batch size (shown in parentheses). The Faster R-CNN model results are similar to image classification models, but only support one batch size (again, we do not plot them in a separate graph).

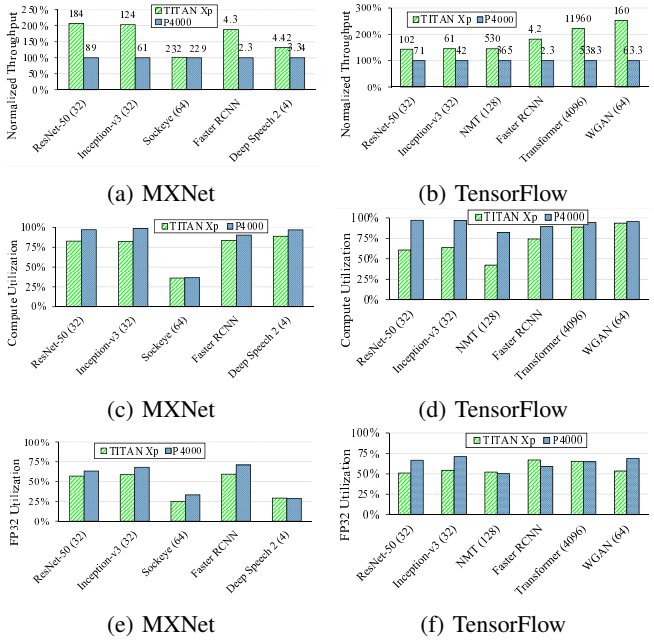
Observation 10: Feature maps are the dominant consumers of memory. It turns out that *feature maps* (intermediate layer outputs) are the dominant memory consumers, rather than weights, which are usually the primary focus of inference



(a) ResNet-50 (b) Inception-v3 (c) Seq2Seq (d) Transformer

(e) WGAN (f) Deep Speech 2 (g) A3C

Fig. 6: GPU FP32 utilization for different models on multiple mini-batch sizes.



(a) MXNet (b) TensorFlow

(c) MXNet (d) TensorFlow

(e) MXNet (f) TensorFlow

Fig. 7: Throughput, Compute Utilization, FP32 Utilization comparison between P4000 and TITAN Xp for different models.

memory optimization. The total amount of memory consumed by feature maps ranges from 62% in *Deep Speech 2* to 89% in *ResNet-50* and *Sockeye*. Hence, any optimization intended to reduce the memory footprint of training should first focus on feature maps. This is an interesting observation also because it expands on the results reported in the only prior work reporting on the memory consumption breakdown for DNN training by Rhu et al. [72]. The authors look at CNN training only and find that weights are responsible only for a very small portion of the total memory footprint. We extend this observation outside of CNNs, but also observe that there are models (e.g., Deep Speech 2) where weights are equally important.

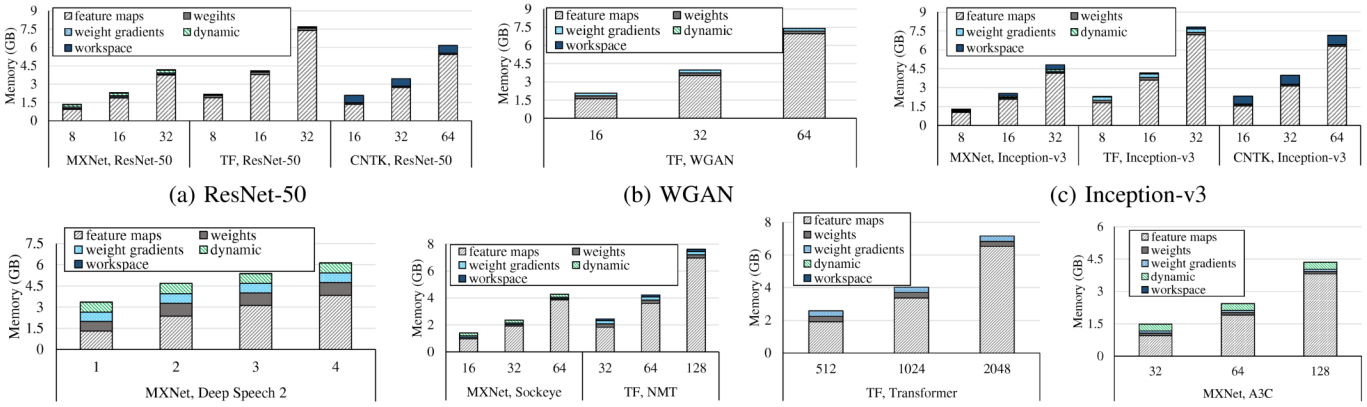
Observation 11: Simply exhausting GPU memory with large batch sizes may be inefficient. The memory consumption of feature maps scales almost linearly with mini-batch size. From observation 11, we know that reducing the mini-batch size can dramatically reduce the overall memory consumption needed for training. Based on observation 1, we also know that the side-effect of throughput loss from using smaller batches can be acceptable (for non-RNN models) if still above saturation. GPU memory saved can be given to workspace (perhaps for a faster implementation of matrix multiplication or convolution) and deeper models (e.g., ResNet-102 vs. ResNet-50).

E. Multi-GPU and Multi-Machine Training

Training large DNNs be accelerated by using multiple GPUs and/or multiple machines. This is usually achieved by using *data parallelism*, where mini-batches are split between individual GPUs and the results are then merged, for example, using the *parameter server* approach [58]. But in order to realize the computational potential of multiple GPUs, the communication channels between them need to have sufficient bandwidth to exchange weight updates. In our work, we analyze the performance scalability of DNN training using multiple GPUs and multiple machines. We use the *ResNet-50* model on MXNet and *Inception-v3* model on TensorFlow to perform this analysis. Figure 9 shows the throughput results from our experiments.

Observation 12: Network bandwidth must be large enough for good scalability. We observe that going from the one machine to the two machine configuration, the performance degrades significantly with bandwidth of only 1 Gbps. This is because DNN training requires constant synchronization between GPUs in distributed training. Hence faster networking is required to improve the situation. In contrast, DNN training on a single machine with multiple GPUs scales reasonably well, since PCIe 3.0 provides enough bandwidth. In summary, networking bandwidth is critical for performance of distributed training and different techniques (in both software and hardware) should be applied to either reduce the amount of data sent or increase the available bandwidth.

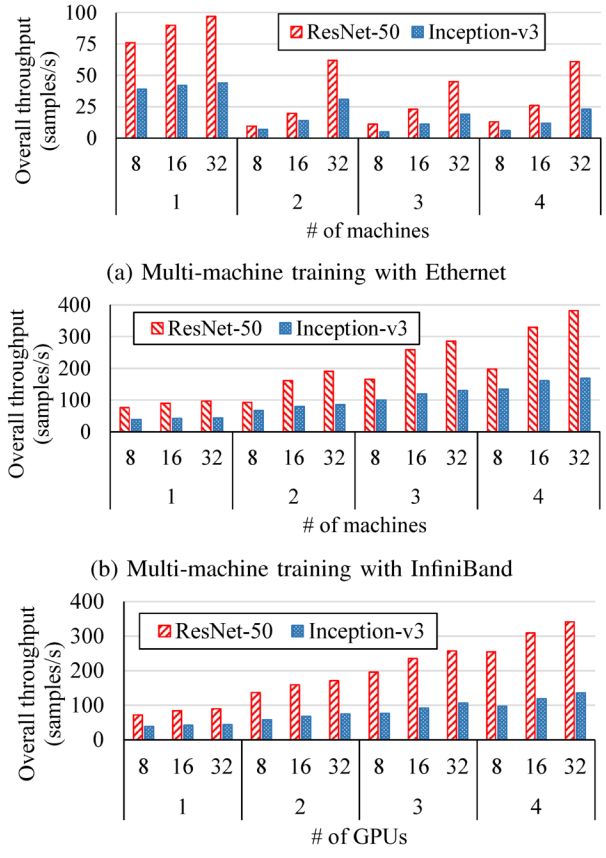
We also applied our toolchain to measure the GPU compute and FP32 utilization for multi-GPU and multi-machine training. Given sufficient bandwidth, these utilization levels almost resembles that of the single-GPU configuration. The GPU memory consumption per GPU remains the same if mini-batch size per GPU is the same. In this case, *to improve the*



(a) ResNet-50 (b) WGAN (c) Inception-v3

(d) Deep Speech 2 (e) Seq2Seq (f) Transformer (g) A3C

Fig. 8: GPU memory usage breakdown for different models on multiple mini-batch sizes.



(a) Multi-machine training with Ethernet

(b) Multi-machine training with InfiniBand

(c) Multi-GPU training on a single machine

Fig. 9: ResNet-50 on MXNet and Inception-v3 on TensorFlow with per-GPU mini-batch size of 8,16,32 on multiple GPUs/machines.

overall performance, one needs to focus again on addressing the single-GPU performance bottlenecks.

V. Related Work

There are only a handful of existing open-source DNN benchmark projects, each with a very different focus from ours. ConvNet [3], CNN-benchmarks [2] and Shaohuai et al. [77] focus exclusively on convolutional network models, mainly for image classification based on the ImageNet data, with the only exception being one LSTM network in [77]. In contrast the goal of our work is a benchmark suite that covers a wide range of models and applications, beyond just CNNs and image classification.

DeepBench [5] is an open-source project from Baidu Research, which is targeted at a lower level in the deep learning stack than our work: rather than working with implementations of deep learning models and frameworks, it instead benchmarks the performance of individual lower level operations (e.g. matrix multiplication) as implemented in libraries used by various frameworks and directly executed against the underlying hardware.

The Eyeriss project [1] presents evaluations of a few DNN processors [24], [41] on hardware metrics for inference rather than training, on several convolutional networks.

Among existing work, Fathom [9] is likely the closest to our own, as it also focuses on training and more than a single application (machine translation, speech recognition and reinforcement learning). However, their focus is on micro-architectural aspects of execution, breaking down training time into time spent on the various operation types (e.g. matrix multiplication). In contrast, our benchmark pool focuses on *system level* aspects of execution such as throughput, hardware utilization, and memory consumption profiling. Moreover, Fathom is based on only one framework (TensorFlow), does not consider distributed training and uses models that are somewhat outdated by now.

Recently there is a joint effort in the community that combines all main DNN benchmark works and aims to provide a new SPEC-like standard for machine learning (MLPerf [7]). Our benchmark suite has also contributed to the model selection of MLPerf. In the future, we plan to focus more on deeper understanding the performance issues under different environments and optimizations. Moreover, we are also planning to enrich our benchmark suite by introducing popular models from MLPerf.

VI. Conclusion

In this work, we proposed a new benchmark suite for DNN training, called TBD, that covers a wide range of machine applications from image classification and machine translation to reinforcement learning. TBD consists of eight state-of-the-art DNN models implemented on three major deep learning frameworks: TensorFlow, MXNet, and CNTK. We used these models to perform extensive performance analysis and profiling to shed light on the efficiency of DNN training for different hardware configurations (single-/multi-GPU and multi-machine). We developed a new tool chain for end-to-end analysis of DNN training that includes (i) piecewise profiling of specific parts of training using existing performance analysis tools, and (ii) merging and analyzing the results from these tools using the domain-specific knowledge of DNN training. Additionally, we built new memory profiling tools specifically for DNN training for all three major frameworks. These useful tools can precisely characterize where the memory consumption (one of the major bottlenecks in training DNNs) goes and how much memory is consumed by key data structures (weights, activations, gradients, workspace). By using our tools and methodologies, we made several important observations and recommendations on where the future research and optimization of DNN training should be focused. We hope that our TBD benchmark suite, tools, methodologies, and observations will be useful for ML developers and systems designers at large in optimizing their DNN training processes.

Ackownledgements

We thank the reviewers and our shepherd for their valuable suggestions. We thank the members of the EcoSystem group for their feedback and the stimulating research environment they provide. We acknowledge the support of our industrial partners: Microsoft, Huawei, and Nvidia. This research was partially supported by NSERC and CFI grants.

References

[1] Benchmarking dnn processors. http://eyeriss.mit.edu/benchmarking.html.

[2] cnn-benchmarks. https://github.com/jcjohnson/cnn-benchmarks.

[3] convnet-benchmarks. https://github.com/soumith/convnet-benchmarks.

[4] cublas. http://docs.nvidia.com/cuda/cublas/index.html.

[5] Deepbench. https://github.com/baidu-research/DeepBench.

[6] Eigen: A c++ linear algebra library. http://eigen.tuxfamily.org/index.php?title=Main_Page.

[7] Mlperf. https://mlperf.org/, 2018.

[8] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[9] Robert Adolf, Saketh Rama, Brandon Reagen, Gu-Yeon Wei, and David Brooks. Fathom: reference workloads for modern deep learning methods. In *Workload Characterization (IISWC), 2016 IEEE International Symposium on*, pages 1–10. IEEE, 2016.

[10] Jorge Albericio, Alberto Delmás, Patrick Judd, Sayeh Sharify, Gerard O'Leary, Roman Genov, and Andreas Moshovos. Bit-pragmatic deep neural network computing. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 382–394. ACM, 2017.

[11] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: ineffectual-neuron-free deep neural network computing. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 1–13. IEEE, 2016.

[12] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. Fused-layer cnn accelerators. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pages 1–12. IEEE, 2016.

[13] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pages 173–182, 2016.

[14] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[16] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[17] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.

[18] Ondrej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics.

[19] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[20] Mahdi Nazm Bojnordi and Engin Ipek. Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning. In *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*, pages 1–13. IEEE, 2016.

[21] Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, Roldano Cattoni, and Marcello Federico. The iwslt 2015 evaluation campaign. In *IWSLT 2015, International Workshop on Spoken Language Translation*, 2015.

[22] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[23] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 367–379. IEEE, 2016.

[24] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

[25] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[26] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 27–39. IEEE Press, 2016.

[27] Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *OSDI*, volume 14, pages 571–582, 2014.

[28] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[29] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

[30] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS workshop*, number EPFL-CONF-192376, 2011.

[31] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 191–198. ACM, 2016.

[32] Christopher De Sa, Matthew Feldman, Christopher Ré, and Kunle Olukotun. Understanding and optimizing asynchronous low-precision stochastic gradient descent. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 561–574. ACM, 2017.

[33] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[34] Caiwen Ding, Siyu Liao, Yanzhi Wang, Zhe Li, Ning Liu, Youwei Zhuo, Chao Wang, Xuehai Qian, Yu Bai, Geng Yuan, et al. C ir cnn: accelerating and compressing deep neural networks using block-circulant weight matrices. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 395–408. ACM, 2017.

[35] Zidong Du, Daniel D Ben-Dayan Rubin, Yunji Chen, Liqiang He, Tianshi Chen, Lei Zhang, Chengyong Wu, and Olivier Temam. Neuromorphic accelerators: A comparison between neuroscience and machine-learning approaches. In *Proceedings of the 48th International Symposium on Microarchitecture*, pages 494–507. ACM, 2015.

[36] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 92–104. ACM, 2015.

[37] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[38] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. Tetris: Scalable and efficient neural network acceleration with 3d memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 751–764. ACM, 2017.

[39] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[40] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5769–5779, 2017.

[41] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 243–254. IEEE Press, 2016.

[42] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[43] Johann Hauswald, Yiping Kang, Michael A Laurenzano, Quan Chen, Cheng Li, Trevor Mudge, Ronald G Dreslinski, Jason Mars, and Lingjia Tang. Djinn and tonic: Dnn as a service and its implications for future warehouse scale computers. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 27–40. ACM, 2015.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[45] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017.

[46] Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post. Sockeye: A toolkit for neural machine translation. *arXiv preprint arXiv:1712.05690*, 2017.

[47] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[48] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R Ganger, Phillip B Gibbons, and Onur Mutlu. Gaia: Geo-distributed machine learning approaching lan speeds. In *NSDI*, pages 629–647, 2017.

[49] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, et al. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716*, 2015.

[50] Animesh Jain, Amar Phanishayee, Jason Mars, Lingjia Tang, and Gennady Pekhimenko. Gist: Efficient data encoding for deep neural network training. In *Computer Architecture (ISCA), 2018 ACM/IEEE 45rd Annual International Symposium on*. IEEE, 2018.

[51] Yu Ji, YouHui Zhang, ShuangChen Li, Ping Chi, CiHang Jiang, Peng Qu, Yuan Xie, and WenGuang Chen. Neutrams: Neural network transformation and co-design under neuromorphic hardware constraints. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pages 1–13. IEEE, 2016.

[52] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[53] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12. ACM, 2017.

[54] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M Aamodt, and Andreas Moshovos. Stripes: Bit-serial deep neural network computing. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pages 1–12. IEEE, 2016.

[55] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 380–392. IEEE, 2016.

[56] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[57] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[58] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 1, page 3, 2014.

[59] Robert LiKamWa, Yunhui Hou, Julian Gao, Mia Polansky, and Lin Zhong. Redeye: analog convnet image sensor architecture for continuous mobile vision. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 255–266. IEEE Press, 2016.

[60] Wenyan Lu, Guihai Yan, Jiajun Li, Shijun Gong, Yinhe Han, and Xiaowei Li. Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pages 553–564. IEEE, 2017.

[61] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[62] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[63] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[64] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 5206–5210. IEEE, 2015.

[65] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[66] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. Scnn: An accelerator for compressed-

sparse convolutional neural networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 27–40. ACM, 2017.

[67] Jongse Park, Hardik Sharma, Divya Mahajan, Joon Kyung Kim, Preston Olds, and Hadi Esmaeilzadeh. Scale-out acceleration for machine learning. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 367–381. ACM, 2017.

[68] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[69] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18. ACM, 2017.

[70] Ao Ren, Zhe Li, Caiwen Ding, Qinru Qiu, Yanzhi Wang, Ji Li, Xuehai Qian, and Bo Yuan. Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 405–418. ACM, 2017.

[71] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[72] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W Keckler. vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pages 1–13. IEEE, 2016.

[73] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[74] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 14–26. IEEE Press, 2016.

[75] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. From high-level deep neural models to fpgas. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pages 1–12. IEEE, 2016.

[76] Yongming Shen, Michael Ferdman, and Peter Milder. Maximizing cnn accelerator efficiency through resource partitioning. *arXiv preprint arXiv:1607.00064*, 2016.

[77] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. Benchmarking state-of-the-art deep learning software tools. In *Cloud Computing and Big Data (CCBD), 2016 7th International Conference on*, pages 99–104. IEEE, 2016.

[78] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. Pipelayer: A pipelined reram-based accelerator for deep learning. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pages 541–552. IEEE, 2017.

[79] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[80] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[81] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*, volume 5, 2015.

[82] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010, 2017.

[83] Swagath Venkataramani, Ashish Ranjan, Subarno Banerjee, Dipankar Das, Sasikanth Avancha, Ashok Jagannathan, Ajaya Durg, Dheemanth Nagaraj, Bharat Kaul, Pradeep Dubey, et al. Scaledeep: A scalable compute architecture for learning and evaluating deep networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 13–26. ACM, 2017.

[84] Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, and Yajuan Wang. Intel math kernel library. In *High-Performance Computing on the Intel® Xeon Phi*, pages 167–188. Springer, 2014.

[85] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[86] Wencong Xiao, Jilong Xue, Youshan Miao, Zhen Li, Cheng Chen, Ming Wu, Wei Li, and Lidong Zhou. Tux2: Distributed graph computation for machine learning. In *NSDI*, pages 669–682, 2017.

[87] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. The microsoft 2016 conversational speech recognition system. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 5255–5259. IEEE, 2017.

[88] Yang You, Zhao Zhang, C Hsieh, James Demmel, and Kurt Keutzer. Imagenet training in minutes. *CoRR, abs/1709.05011*, 2017.

[89] Dong Yu, Adam Eversole, Mike Seltzer, Kaisheng Yao, Zhiheng Huang, Brian Guenter, Oleksii Kuchaiev, Yu Zhang, Frank Seide, Huaming Wang, et al. An introduction to computational networks and the computational network toolkit. *Microsoft Technical Report MSR-TR-2014–112*, 2014.

[90] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing dnn pruning to the underlying hardware parallelism. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 548–560. ACM, 2017.

[91] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. Cambricon-x: An accelerator for sparse neural networks. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pages 1–12. IEEE, 2016.