End-to-End Benchmarking of Deep Learning Platforms*

Vincent Deuschle[1,2], Alexander Alexandrov[1,2], Tim Januschowski[1], and Volker Markl[2]

[1] Amazon Web Services, Inc., Berlin, Germany
deuscv@amazon.com
[2] Technische Universität Berlin, Berlin, Germany

Abstract. With their capability to recognise complex patterns in data, deep learning models are rapidly becoming the most prominent set of tools for a broad range of data science tasks from image classification to natural language processing. This trend is supplemented by the availability of deep learning software platforms and modern hardware environments. We propose a declarative benchmarking framework to evaluate the performance of different software and hardware systems. We further use our framework to analyse the performance of three different software frameworks on different hardware setups for a representative set of deep learning workloads and corresponding neural network architectures[3].

Keywords: Deep learning · Declarative Benchmarking · Cloud Computing

1 Introduction

Driven by the digitization of ever more aspects of life, the increase of available data as well as computing capacities has given rise to the application of complex predictive models across a wide range of sectors in research and industry. Most notably deep learning, used as an umbrella term for neural network based computational models, continues to capture academic as well as public attention. With their capacity to model complex structures and dependencies within data, neural networks are playing a rapidly increasing role as pattern recognition techniques with powerful means for regression and classification objectives.
The impact of these models and algorithms was fueled (and has driven) the development of a spectrum of publicly available software platforms that implement fundamental deep learning concepts such as auto-differentiation [17] and in particular backpropagation [22]. Build around accessible APIs, these platforms enable researchers to easily specify, train and deploy a broad range of neural network architectures. Simultaneously, advances in modern hardware have made

* The authors would like to thank Tim Januschowski and the ML Forecasting team of Amazon Web Services for supporting this work.
[3] Our framework is publicly available at https://github.com/vdeuschle/rysia.

efficient processing of large amounts of data, which are often required to train neural networks possible. Most notably, graphics processing units (GPUs) and GPU-related hardware units have proven to be efficient devices to perform the linear algebra operations that deep learning principles are build on.
This vast spectrum of soft- and hardware choices constitutes a challenge for researchers who are trying to settle on the optimal system combination for a given task, e.g. natural language processing or image classification. With this paper we introduce a novel benchmarking framework as an easy to use tool to compare the performance of various deep learning software platforms and hardware configurations. Utilizing our framework, we have conducted a number of performance experiments with representative deep learning workloads for popular software platforms on CPU-restricted and GPU-accelerated hardware. Specifically, we make the following contributions with this paper:

- We propose a declarative benchmarking framework for deep learning platforms and hardware environments that guarantees comparability and reproducability with a flexible interface to formulate and represent benchmarking workloads.
- We formulate three different workloads which represent popular subjects in the broader field of deep learning to benchmark feed forward networks [29], convolutional networks [25] and LSTM networks [23].
- We utilize our framework to conduct a number of training and inference experiments, executing the aforementioned workloads, to compare the performance of three different software platforms, namely Tensorflow [16], Apache MXNet [19] and Pytorch [21], in different hardware environments.

The key insights of our work are the following:

- The benefits of GPU-accelerated hardware during training depend on various factors, such as the neural network architecture and corresponding operators (e.g. matrix multiplication or convolution).
- MXNet outperforms both other platforms in most GPU-accelerated hardware environments during training. No platform outperforms any other under all circumstances.
- Pytorch outperforms both other platforms during inference in all tested hardware environments.

In the remainder of this paper we give brief introduction into the background of our work (Section 2), introduce our benchmarking framework (Section 3), describe our experiments and results (Section 4 and Section 5) and put our work in the context of other contemporary deep learning benchmarking efforts (Section 6).

2 Background

In this section we provide an overview over the background that is required to understand the context of our work. In Section 2.1 we introduce the three

software platforms that are subject of our analysis. In Section 2.2 we describe the hardware environments that we used for our work. Most importantly, we distinguish between *CPU-restricted* and *GPU-accelerated* setups.

2.1 Software Platforms

In this section we introduce the three software platforms that are subject of our analysis. We give a brief overview over each platform and specify which APIs and engines we are analyzing in particular. In Table 1 we provide a concrete listing of all software versions and APIs that are subject of our work.

Table 1: Platform versions, datatypes and operators that we use for our benchmark.

Version	**MXNet**	**Tensorflow**	**Pytorch**
CPU	1.4.1	1.12.2	1.1
GPU	1.4.1	1.12.2	1.1
Datatype	**MXNet**	**Tensorflow**	**Pytorch**
Feed Forward, Convolution	Variable	Variable	Variable
LSTM	LSTMCell	LSTMBlockCell, CudnnCompatibleLSTMCell	LSTMCell
Operator	**MXNet**	**Tensorflow**	**Pytorch**
Matrix-Multiplication	linalg_gemm2	matmul	@
Convolution (2D)	Convolution	conv2d	conv2d
Bias Addition	broadcast_add	+	+
Max Pooling (2D)	Pooling	max_pool	max_pool2d

Tensorflow [16] is an open source deep learning platform developed by Google Brain [9]. In it's *symbolic* runtime, which is subject of our analysis, Tensorflow builds, compiles and optimizes a computational graph, that represents a neural network architecture before execution. We have chosen Tensorflow's most low-level *Variable* API as subject of our analysis for feed forward and convolutional neural networks and two different LSTM cells for recurrent models.

Apache MXNet [19] is an open source deep learning platform that is currently part of the Apache Incubator project [2]. Symbolic as well as imperative programming is supported on CPU-restricted and GPU-accelerated hardware. For a purely symbolic execution, MXNet offers the "Symbol API" and "Module API" to the user, the latter one serving as a wrapper for the Symbol API, providing high level concepts such as in-build optimizer (e.g. gradient descent) to the user. Similar to Tensorflow, we choose the symbolic *Variable* API for feed forward and convolutional networks and prebuild LSTM cells for recurrent models.

PyTorch [21] is an open source deep learning platform developed by Facebook AI Research [8]. Imperative programming and eager execution are the central design principles with symbolic computation not supported[4]. Computational graphs are not precompiled and may be modified during runtime. To enable backpropagation and the computation of gradients, all operations that are performed during a forward pass through the computational graph are recorded when they occur during runtime. Subject of our analysis is PyTorch's *Tensor* API with prebuild LSTM cells being used for recurrent models.

2.2 Hardware Environments

In this section we describe the hardware setups that are subject of our analysis. We execute our experiments within the Amazon Web Services (AWS) cloud environment [1], utilizing various CPU-restricted and GPU-accelerated EC2 instances [6]. We provide a detailed overview over the specifics of all used hardware environments in Table 2. All experiments are executed in containerized virtual machines, using Docker [5].

Table 2: Hardware configurations of all EC2 instances that we use for training and inference experiments.

EC2 Instance	**Cores**	**CPU Type**	**Main Memory**	**GPU**
c4.2xlarge	8	2.9 GHz, Intel Xeon E5-2666 v3	15 GiB	n/a
c5.xlarge	4	3.0 GHz, Intel Xeon Platinum	8 GiB	n/a
c5.2xlarge	8	3.0 GHz, Intel Xeon Platinum	16 GiB	n/a
c5.4xlarge	16	3.0 GHz, Intel Xeon Platinum	32 GiB	n/a
c5.9xlarge	36	3.0 GHz, Intel Xeon Platinum	72 GiB	n/a
p2.xlarge	8	2.3 GHz, Intel Xeon E5-2686 v4	61 GiB	NVIDIA Tesla K80
p3.2xlarge	8	2.3 GHz, Intel Xeon E5-2686 v4	61 GiB	NVIDIA Tesla V100

CPU-restricted Runtime: Deep learning platforms utilize a variety of libraries for efficient linear algebra computation which enable multithreaded parallelism. Regarding the software platforms which are subject to this paper, MXNet and Pytorch both employ OpenBlas [14] by default, while Tensorflow uses the Eigen library [7]. In general, CPU-restricted computation constitutes a performance

[4] As of Version 1.0 Pytorch features a just-in-time compiler that will enable user to precompile static models before runtime without the need of symbolic operators.

limitation for deep learning workloads. The degree of parallelism is limited to the number of available CPU threads, which is restricted by the number of available CPU cores. Linear algebra operations however (and consequently deep learning), often benefit from a considerably higher degree of parallelism

GPU-accelerated Runtime: The software library most commonly utilized by deep learning platforms to allow GPU-accelerated computations is the NVIDIA CUDA Toolkit [13]. The three platforms that are subject to this paper all utilize CUDA for GPU-accelerated computations. Operators such as matrix multiplication are formalized as *kernels*, which are routines written within the CUDA framework, to be executed on a GPU. Most commonly, deep learning platforms provide CUDA implementations for all operators that may define a computational graph. The performance gain of GPU-accelerated computations depends on a variety of factors. For instance, a frequent data exchange between main memory and GPU may cause a runtime performance overhead.

3 The Rysia Benchmarking Framework

In this section we introduce the Rysia benchmnarking framework that we implemented for our experiments. We designed a declarative interface that enables users to specify and execute benchmarking workloads without any coding required. In Section 3.1 we describe the guiding design principles of our work. In Section 3.2 we describe the architecture of our framework and the corresponding workflow of any benchmarking experiments.

3.1 Design Principles

We propose a new benchmarking approach that aims to preserve the core principles of other approaches while ensuring a higher degree of comparability between software platforms. We follow three guiding principles that define the core concept of our benchmarking framework:

- **Comparability** We ensure *functional equivalence* between platforms for our experiments, meaning that equivalent operations (e.g. matrix multiplication, convolution) are used and the same data (e.g. same mini-batches in each iteration of stochastic gradient descent) is being processed at each step of the computation.
- **Reproducibility** We provide a way to formalize benchmarking workloads in a declarative fashion that enables users to easily store, rerun and modify experiment setups and workload specifications.
- **Flexibility** We implement an easy to use *domain specific language* to enable users to easily specify deep learning models of different architectures for various workloads. Our framework automatically compiles these specifications into platform specific implementations.

By ensuring functional equivalence between platforms, we make sure that for training workloads, each platform reaches the same accuracy after the same amount of training iterations and that the same mathematical operations (e.g. matrix multiplication, convolution) are being performed on the same batches of data in each iteration[5]. We measure the *runtime in seconds* that is required to perform a predefined amount of training iterations.
For inference workloads, we measure the throughput rate for a platform on a given model. This means that we count the number of forward passes that each platform accomplishes in a time window of one minute on a fix sized mini-batch of samples that are randomly selected from a dataset for each forward pass. Dividing the throughput rate with 60, we measure *mini-batches per second*.
While other benchmarking approaches rely on hardcoded scripts that correspond to a specific workload (e.g. image recognition), we offer a more flexible approach, that lets users easily specify new model architectures, data sets and hyperparameter in a unified, declarative fashion, without having to worry about platform specific implementations. This clear separation of benchmark specification and execution makes it easy to reproduce and compare a large variety of experiments. Our framework ensures that for each workload the concept of functional equivalence holds.
To ensure comparability between platforms it is crucial that identical workloads are executed under the same conditions. This especially includes the hardware environments which we utilize to run our experiments. We believe that the most reliable way to achieve this is a cloud-based runtime. We therefore natively integrate cloud functionality in our framework, enabling the user to choose between local execution and a variety of cloud-based hardware environments that will stay consistent over the course of any batch of experiments.

3.2 Architecture

In this section we describe the architecture of our benchmarking framework. Following the design principles described in the previous section, we have implemented a framework with which we aim to overcome some of the blind spots of other benchmarking approaches. In the remainder of this section, we will give a detailed introduction to the key components of our framework and how they relate to the three principles that we have formulated in the previous section.
As we stated before, comparability, reproducibility and flexibility are the guiding principles of our framework. We believe that the best way to materialize these concepts is to provide the user with an accessible way to formalize benchmarking workloads in a declarative fashion. To achieve that, we introduce the idea of *blueprint* configuration files, in which the user can specify all relevant parameters that define a benchmarking workload. Specifically, these blueprints include:

[5] Under ideal conditions, our approach should indeed result in exactly the same accuracy curves between platforms. In reality however, even ensuring that the same operations are performed on exactly the same data in each step, does not result in perfectly aligning accuracy rates.

- A **domain specific language** to formalize deep learning model architectures
- **Hyperparameter** Number of epochs, choice of optimizer, loss function, etc.
- **Metaparameter** Software platform to benchmark, hardware metrics to monitor and number of runs
- **Cloud Parameter** EC2 instance type, Docker image location
- **Data Paths** Path to datasets, result folder path or model parameter path

All blueprints are implemented as Python modules, which are dynamically imported at the beginning of our runtime. The most notable point of these declarative specifications is that they provide a generalized formalization for the user, whereas any platform specific implementation is automatized by our framework. The concept of a generalized domain specific language (DSL) to formalize deep learning models independently from the platform they will be executed on has been popularized first and foremost by the Keras API specification [10]. Users can specify deep learning models that consist of layers, which define the mathematical operation that is performed at the corresponding layer of the model (e.g. feed forward layer correspond to matrix multiplication). The key advantage of this approach is to separate the model architecture from any platform specific implementation, which are generated automatically.
For our benchmarking framework we follow a similar principle, with the focus shifted to build models for benchmarking workloads. Through our DSL, users may specify model architectures that consist of feed forward, convolutional and LSTM layers (with Max Pooling and Flattening as additional operators) since it is commonly assumed that these three operators make up the vast majority of computations in deep learning workloads. Within blueprint files, model architectures are specified as sequences of layers. During runtime, these sequences are converted to models which implement these layers for the platform that has been specified. For feed forward and convolutional layers, this means storing platform specific variables, which represent weight and bias matrices. LSTM layers hold a platform specific LSTM cell. All other layer types correspond to stateless operations. Implementing our own DSL (and not utilizing publicly available Keras implementations), enables us to maintain control over the operators and APIs that we select for each software platform. To ensure comparability across platforms, we have implemented our DSL with functionally equivalent low-level operators (e.g. matrix multiplication or convolution) for each platform.

4 Experiment Setup

In this section we describe the experiments that we conducted within the scope of this paper. Utilizing our framework described in Section 3, we analyze and address subjects regarding *training runtime* and *inference throughput* performance. For each subject we specify the questions that we will answer, as well as the corresponding experiment setup, which includes workloads, corresponding model architectures, datasets, and hardware specifications. In the remainder of

this chapter we provide a detailed description of all experiments, subjects and corresponding workloads. In Section 4.1 we describe the three workloads and corresponding model architectures that we use to analyze the three subjects listed above. In Table 3 we specify the runtime environment of our experiments. In the subsequent two sections, we provide further details for each subject such as individual hyperparameter and utilized hardware setups.

Table 3: Runtime environment that we use for our benchmark.

Version	**Docker Base Image**	**OS**
CPU	ubuntu:16.04	Ubuntu 16.04
GPU	nvidia/cuda:9.0-cudnn7-devel	Ubuntu 16.04
Version	**CUDA/CUDNN**	**Python**
CPU	n/a	3.6
GPU	9.0/7.3	3.6

4.1 Workloads and Deep Learning Model Architectures

In this section we specify the workloads and corresponding model architectures that we use for our experiments. Workloads and models will remain consistent across all subjects, meaning that the same three model architectures will be used to benchmark and analyze training and inference performance. We have chosen the following tasks as workloads for this paper, which are canonical and widely applied in the field of deep learning:

- Handwritten digits classification with feed forward networks (MNIST)
- Image classification with convolutional networks (CIFAR-10)
- Sentiment analysis with LSTM Networks (IMDB)

The MNIST dataset [12] consists of greyscale images of handwritten digits. Each image of size 28x28 pixels is being flattened to a one-dimensional vector with 784 features. For this relatively simple classification objective we have chosen a conventional feed forward neural network architecture of three layers with 128, 64 and 10 neurons per layer.
The CIFAR-10 dataset [3] consists of RGB images of ten different objects. Each image is of size 32x32x3 pixels and each label corresponds to an index for one of ten different objects. To solve this classification problem we chose a convolutional VGG16 neural network architecture [28] with 10 convolutional network layers, followed by three feed forward layers.
The IMDB dataset [26] consists of movie reviews in text form. Each review is treated as a sequence of words and each word is mapped to a word-vector of 50 features, as described by [27]. Each sequence is truncated at 250 words and each sequence that is smaller than 250 words is padded with an according amount of zero-vectors, thus creating samples of size 250x50 each. Since we are performing

sentiment analysis, each label marks a review as either "positive" or "negative". To solve this binary sequence classification task, we chose a LSTM network with one layer and 128 cells. The data representation follows [15].

4.2 Training Runtime Performance

For this subject, we compare the runtime performance and hardware utilization profiles of MXNet, Tensorflow and Pytorch for the three different training workloads described in the previous section. Within this context, we will address the following two questions:

- How do different software platforms perform with different training workloads?
- How do different hardware environments perform with different training workloads?

Table 4: Hyperparameter for training performance experiments.

	MNIST	**CIFAR-10**	**IMDB**
Epochs	200	100	100
Optimizer	SGD	Adam	Adam
Batch Size	128		
Learning Rate	0.01		
Loss Function	Cross-Entropy		

The hyperparameter configurations that we use for our training experiments are listed in Table 4. For each task we use a batch-size of 128 and a learning-rate of 0.01. The number of training epochs varies between tasks. We further chose the cross-entropy loss function for each workload. While the feed forward model for our MNIST job is trained with the vanilla gradient descent optimizer of each framework, we chose the more advanced Adam optimizer [24] for the other two training jobs.
In order to maximize comparability across platforms, we run all experiments on the same types of EC2 instances, thus ensuring consistent hardware configurations across each experiment setup. Table 2 shows the EC2 instances that have been used for each experiment. We compare instances with powerful CPU types (older C4.2xlarge and newer C5.2xlarge) and instances that provide GPU devices (older P2.xlarge and newer P3.2xlarge) in our training experiments.

4.3 Inference Throughput Performance

For this subject we compare the inference throughput performance of MXNet, Tensorflow and Pytorch for multiple different inference workloads. Specifically, we will address the following question:

- How well do different software platforms utilize different numbers of available CPU cores during inference?

We utilize again the three datasets and corresponding model architectures described in Section 4.1. We further take the pretrained models from Section 4.2 as initial model parameters. For each forward pass we chose mini-batches of 128 samples that are randomly selected from the whole dataset. We measure how many complete forward passes each platform is able to perform for each model within a time window of one minute. As opposed to our training experiments, where we were measuring runtime for a fixed amount of iterations, we measure the number of iterations for a fixed amount of time in our inference setup.
As with our training experiments, we run all throughput experiments on the same types of EC2 instances, in order to maximize comparability. A key difference for our inference subject is, that all experiments are executed on CPU-restricted hardware, since this setup is more commonly used for inference tasks. Table 2 shows the hardware specifications of each instance. We select C5 instances of different sizes with 4, 8, 16 and 36 CPU cores respectively.

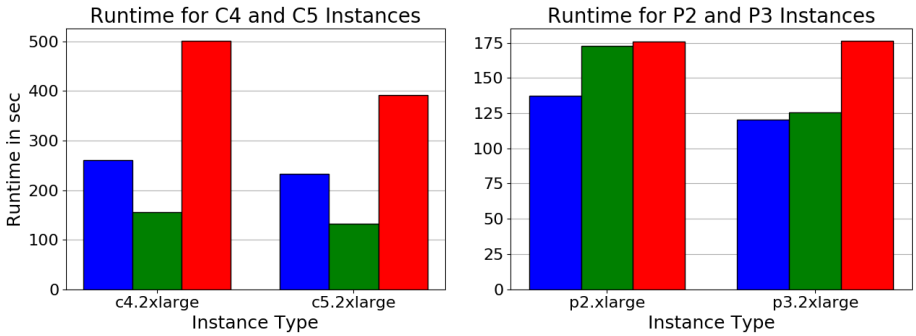
5 Experiment Results

In this chapter we analyze the results and findings of the experiments that we conducted in our work as described in Section 4. We show all relevant result metrics such as median runtime for training and median throughput for inference experiments.

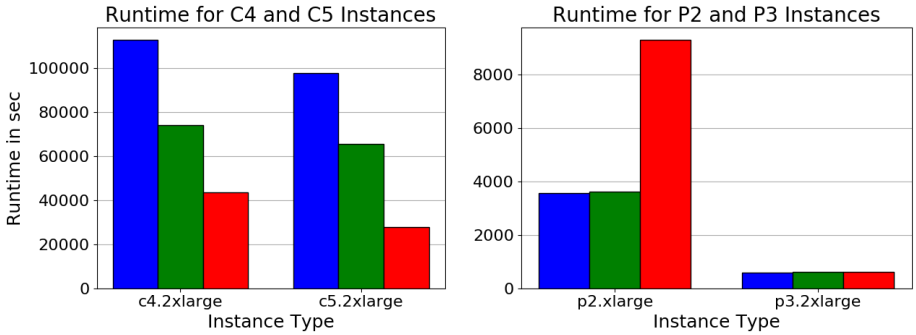
5.1 Training Performance

In this section we analyze the results and findings of our training performance experiments, described in Section 4.2. For each experiment we report the median runtime over seven runs.

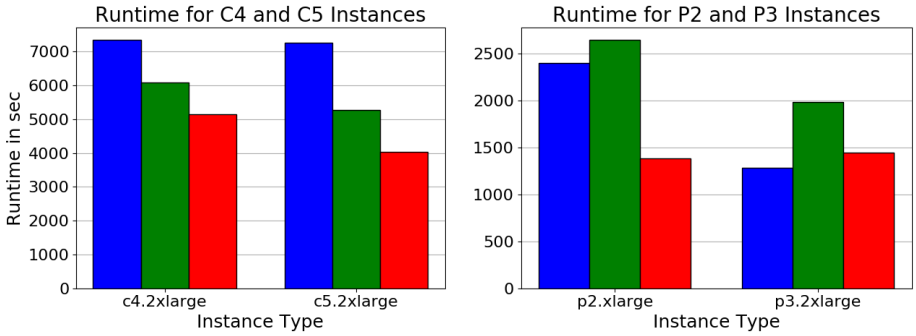
Handwritten Digits Classification with Feed Forward Networks : In Figure 1a we show the observed median runtime for each combination of platform and EC2 instance for the MNIST workload with feed forward networks. On CPU-restricted C4 and C5 instances, we measured the fastest runtime performance for Tensorflow, outperforming both other platforms with a median runtime of 155 seconds on C4 instances and 132 seconds on C5 instances. On the newer C5 instance generation we measured the most noticeable speedup for Pytorch and minor speedups for Tensorflow and MXNet.
On GPU-accelerated P2 and P3 instances, MXNet outperforms both Pytorch and Tensorflow by a small margin with a median runtime of 137 seconds on P2 instances and 120 seconds on P3 instances. On the newer P3 instance, we measured minor speedups across all platforms, compared to the older P2 instance.
Comparing the runtime on CPU-restricted and GPU-accelerated instances, we measured noticeable speedups for MXNet and Pytorch on the latter. For Tensorflow we did not measure any significant speedup, compared to its already fast



(a) Runtime comparison (training) for the MNIST workload with feed forward networks on different C (left) and P (right) instances.



(b) Runtime comparison (training) of the CIFAR-10 workload with convolutional networks on different C (left) and P (right) instances.



(c) Runtime comparison (training) for the IMDB workload with LSTM networks on different C (left) and P (right) instances.

MXNet Tensorflow Pytorch

Fig. 1: Results of our runtime experiments. Each bar shows the median runtime performance of one software platform over seven runs.

CPU-bound performance. In general, performance differences across platforms proved much smaller on GPU-accelerated than on CPU-restricted hardware. Averaging over all platforms and both types of CPU and GPU hardware environments respectively, we have measured a speedup of 1.84 on the latter. We conclude, that for relatively small feed forward neural networks, the speedup of GPU-accelerated hardware needs to be leveraged against the significant cost increase that comes with these types of hardware environments.

Image Classification with Convolutional Networks : In Figure 1b we show the observed median runtimes for each combination of platform and EC2 instance for the CIFAR-10 workload with a convolutional VGG16 network. On C4 and C5 instances, Pytorch outperforms both other platforms with a median runtime of 43474 seconds on C4 instances and 27837 seconds on C5 instances. Each platform shows a minor speedup on the newer C5 instance, compared to the older C4 instance.
On P2 instances we measured virtually identical runtime performance for Tensorflow and MXNet, both outperforming Pytorch with a slightly faster median runtime of 3556 seconds for MXNet. On P3 instances, we measured the fasted median runtime for MXNet with 600 seconds, albeit all three platforms perform almost identical.
Across all platforms, we have measured a tremendous performance increase, when training convolutional neural networks in GPU-accelerated hardware environments. Figure 1b shows that all frameworks perform multiple times faster on P2 instances compared to CPU-restricted hardware and again multiple times faster on P3 instances. Comparing the two newer generations of CPU and GPU instances (C5 and P3), MXNet runs roughly 163 times faster on the latter, Tensorflow 109 times faster and Pytorch 44 times. Averaging over all platforms and both types of CPU and GPU hardware environments respectively, we have measured a speedup of 23.05 on the latter.

Sentiment Analysis with LSTM Networks : In Figure 1c we show the observed median runtimes of seven runs for each combination of platform and EC2 instance for the IMDB workload with LSTM networks. On C4 and C5 instances, Pytorch outperforms both other platforms with median runtimes of 5138 and 4033 seconds respectively. Each platform shows a minor speedup on the newer C5 instance, compared to the older C4 generation.
On GPU-accelerated instances we measured significant speedups for all three platforms, compared to CPU-restricted hardware environments. On P2 instances, Pytorch outperforms both other platforms by a significant margin with a median runtime of 1386 seconds. On P3 instances, MXNet runs fastest with a median runtime of 1281 seconds. We further measured a significant performance increase on the newer P3 generation across all platforms, compared to the older P2 generation.
Overall, we observed that all three platforms are capable of utilizing GPU-accelerated hardware reasonably well, training LSTM networks. Averaging over

all platforms and both types of CPU and GPU hardware environments respectively, we have measured a speedup of 3.15 on the latter.

5.2 Inference Throughput Performance

In this section we analyze the results and findings of our inference performance experiments, described in Section 4.3. For each experiment we report the median throughput over seven runs.

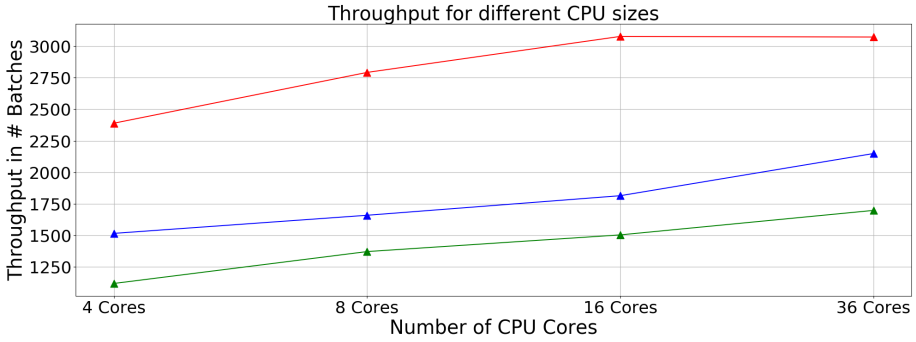
Handwritten Digits Classification with Feed Forward Networks : In Figure 2a we show the observed median throughput for each combination of platform and EC2 instance for the MNIST workload. For all CPU sizes we recorded the highest throughput rate for Pytorch and the lowest for Tensorflow. For Pytorch we observed a constant performance increase up until 16 cores with no further increase beyond that. For MXNet and Tensorflow we measured consistent performance increases up until the maximum of 36 CPU cores, albeit without ever coming close to the throughput rate of Pytorch. Across all platforms and instance types we recorded the highest throughput rate for Pytorch on c5.4xlarge (16 cores) instances with 3077.48 mini-batches per second.

Image Recognition with Convolutional Networks : In Figure 2b we show the observed median throughput for each combination of platform and EC2 instance for the CIFAR-10 workload. We recorded the highest throughput rate for Pytorch for all CPU sizes, with a continuous performance increase up to the maximum of 36 available cores. While Tensorflow showed a continuous raise in mini-batch throughput with more available cores too, the platform never reached the performance of Pytorch. We measured the lowest throughput rate for MXNet with no performance increase beyond 8 CPU cores. Across all platforms and instance types we recorded the highest throughput rate for Pytorch on c5.9xlarge (36 cores) instances with 12.55 mini-batches per second.

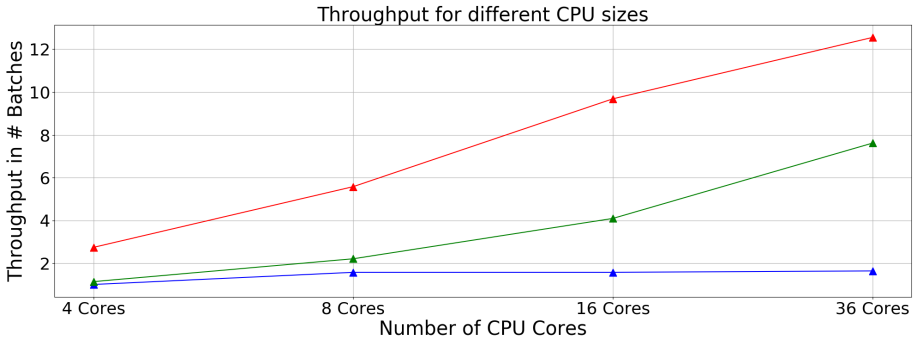
Sentiment Analysis with LSTM Networks : In Figure 2c we show the observed median throughput for each combination of platform and EC2 instance for the IMDB workload. We observed the highest throughput rate for Pytorch across all CPU sizes with a performance increase up to 16 available CPU cores and a drop in performance afterwards. For Tensorflow we recorded constant increases in performance up until 16 cores. For MXNet we recorded the lowest throughput rate and only observed an increase in performance between four and eight available cores with no further increase afterwards. Across all platforms and instance types we recorded the highest throughput rate for Pytorch on c5.4xlarge (16 cores) instances with 13.25 mini-batches per second.

6 Related Work

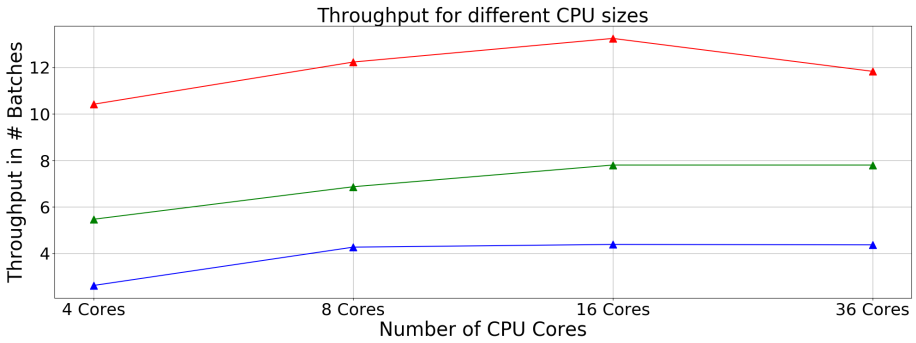
In this section we briefly introduce three other benchmarking approaches for deep learning platforms. While similar end-to-end benchmarking concepts have been



(a) Inference throughput comparison of the MNIST workload with feed forward networks on C5 instances of different sizes.



(b) Inference throughput comparison of the CIFAR-10 workload with convolutional networks on C5 instances of different sizes.



(c) Inference throughput comparison of the IMDB workload with LSTM networks on C5 instances of different sizes.

Fig. 2: Results of our inference experiments. Each plot shows the median runtime performance of one software platform over seven runs.

applied and implemented before, we distinguish our work by offering an end-to-end benchmark that lets users declaratively formulate deep learning workloads in blueprint files.

C. Bourrasset et al. [18] define a set of principle requirements for deep learning training and inference benchmarks in enterprise environments. For end-to-end benchmarks which are subject of our work, the authors identify several relevant metrics such as computation time and efficiency or energy and hardware resource consumption (amongst others). Principal requirements that are stated for benchmarking approaches are reproducibility, portability and comparability. These concepts align with the design principles that we have selected for our work.

MLPerf [11] is a benchmarking initiative that originated from the DAWNBench project [20] by the Stanford University and is now supported by a variety of academic and industrial actors. MLPerf aims to compare deep learning platforms on a workload level. The primary performance evaluation metric is defined as the wall clock time to train a deep learning model to a target quality. Participators can submit results for a predefined workload in a deep learning platform and a hardware setup of their own choosing. MLPerf strives for *mathematical equivalence*, which is asserted by explicitly predefining a model architecture, hyperparameter as well as initial model parameter that all submissions must follow.

I. Karmanov et al. [4] propose a "Rosetta Stone approach" for benchmarking deep learning platforms. The initiative was originated by data scientists at Microsoft, with contributions by various teams working on different deep learning platforms such as CNTK and Pytorch. The authors describe the project as an attempt to provide reference implementations for various deep learning workloads across a variety of different platforms. The authors predefine model architecture, dataset and hyperparameter for each given workload.

7 Conclusion

7.1 Summary

With this paper we have introduced the Rysia benchmarking framework as a novel end-to-end benchmark for deep learning platforms. Following the key principles of *comparability*, *reproducibility* and *flexibility* we have designed a framework that enables users to specify benchmarking workloads without the need of implementing any underlying functionality. To that purpose we introduced the concept of *blueprint files* in which users may specify any relevant parameters that define a benchmark, such as the model architecture, hyperparameter, datasets and cloud resource specifications. With this clear separation of workload specification and implementation, we guarantee reproducibility as well as

flexibility. We further guarantee comparability by ensuring that the platform-specific execution of blueprints are functionally equivalent across platforms. In each computational step our framework utilizes equivalent data structures and operators of each platform. Over the course of this paper we have shown that the aforementioned paradigms are reasonable guidelines for benchmarking approaches which are not fully realized by related work yet.
Utilizing our framework, we have conducted a broad range of representative benchmarking experiments for deep learning platforms and hardware environments. We have shown that the performance of individual software platforms and hardware environments during training depends on a given workload and its corresponding model architecture. For GPU-accelerated hardware we have measured by far the highest speedup when training convolutional networks on image data. For training workloads we have further measured significant performance differences across platforms, with no platform outperforming any other for each workload. We further noticed that symbolic setups with static computational graphs (MXnet and Tensorflow) do not necessarily guarantee better performance than imperative setups with dynamic graphs (Pytorch).
For inference workloads we observed the best performance for Pytorch in all tested hardware environments. The benefits of higher degrees of parallelism (provided by higher numbers of available CPU cores) largely depend on the workload, corresponding model architecture and software platform in question.

7.2 Future Work

In this paper we have shown the merits of separating a benchmark specification from its implementation and execution. By extending this concept, a far broader range of deep learning workloads could be covered and analyzed. Our domain specific language for formalizing model architectures could for instance be extended with further operators (e.g. attention layer, dropout layer, etc.) to cover a broader spectrum of neural network architectures. The crucial prerequisite for an extension of our own work would be the continued focus on functional equivalence during the transformation of a declarative formalization into a platform-specific implementation. We consider the Keras API specification best suited for expressing more complex model architectures.
We have chosen three different deep learning software platforms to compare and analyze. The landscape of available platforms is however far wider and an obvious extension to our own work would be the support and analysis of other platforms. The design principles that we followed throughout our work are universally applicable and not limited to any specific platform.
While we focused on single-node hardware environments in our work, distributed training is a highly relevant research topic within the deep learning community. A valuable extension of our work would be an implementation and analysis of training workloads on multi-node clusters or single machines with multiple GPUs. Our concept of declarative benchmark specifications could be extended accordingly, given that the analyzed software platforms offer distributed training.

Bibliography

[1] Amazon Web Services, https://aws.amazon.com/, accessed: 2019-05-28

[2] Apache Incubator, https://incubator.apache.org/, accessed: 2019-06-10

[3] CIFAR-10 dataset, https://www.cs.toronto.edu/~kriz/cifar.html, accessed: 2019-06-10

[4] Comparing Deep Learning Frameworks: A Rosetta Stone Approach, https://blogs.technet.microsoft.com/machinelearning/2018/03/14/comparing-deep-learning-frameworks-a-rosetta-stone-approach/, accessed: 2019-06-10

[5] Docker Platform, https://www.docker.com/, accessed: 2019-06-10

[6] EC2 Instances, https://aws.amazon.com/ec2/instance-types/, accessed: 2019-06-10

[7] Eigen Library, http://eigen.tuxfamily.org/index.php?title=Main_Page, accessed: 2019-06-10

[8] Facebook AI Research, https://research.fb.com/category/facebook-ai-research/, accessed: 2019-06-10

[9] Google Brain, https://ai.google/research/teams/brain, accessed: 2019-06-10

[10] Keras Framework, https://keras.io/, accessed: 2019-06-10

[11] MLPerf benchmark suite, https://mlperf.org/, accessed: 2019-06-10

[12] MNIST database of handwritten digits, http://yann.lecun.com/exdb/mnist/, accessed: 2019-06-10

[13] NVIDIA Cuda, https://developer.nvidia.com/cuda-toolkit, accessed: 2019-06-10

[14] OpenBlas Library, https://www.openblas.net/, accessed: 2019-06-10

[15] Perform sentiment analysis with LSTMs, using TensorFlow, https://www.oreilly.com/learning/perform-sentiment-analysis-with-lstms-using-tensorflow, accessed: 2019-06-10

[16] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: a system for large-scale machine learning. In: OSDI. vol. 16, pp. 265–283 (2016)

[17] Baydin, A.G., Pearlmutter, B.A., Radul, A.A., Siskind, J.M.: Automatic differentiation in machine learning: a survey. Journal of Machine Learning Research **18**(153), 1–43 (2018), http://jmlr.org/papers/v18/17-468.html

[18] Bourrasset, C., Boillod-Cerneux, F., Sauge, L., Deldossi, M., Wellenreiter, F., Bordawekar, R., Malaika, S., Broyelle, J.A., West, M., Belgodere, B.: Requirements for an Enterprise AI Benchmark. In: Performance Evaluation and Benchmarking for the Era of Artificial Intelligence. pp. 71–81 (2018)

[19] Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., Zhang, Z.: Mxnet: A flexible and efficient machine

learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274 (2015)
[20] Coleman, C., Narayanan, D., Kang, D., Zhao, T., Zhang, J., Nardi, L., Bailis, P., Olukotun, K., Ré, C., Zaharia, M.: DAWNBench: An End-to-End Deep Learning Benchmark and Competition. Training **100**(101), 102 (2017)
[21] Collobert, R., Kavukcuoglu, K., Farabet, C.: Torch7: A matlab-like environment for machine learning. In: BigLearn, NIPS workshop. No. EPFL-CONF-192376 (2011)
[22] Hecht-Nielsen, R.: Theory of the backpropagation neural network. In: Neural networks for perception, pp. 65–93. Elsevier (1992)
[23] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
[24] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
[25] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
[26] Maas, A.L., Daly, R.E., Pham, P.T., Huang, D., Ng, A.Y., Potts, C.: Learning Word Vectors for Sentiment Analysis. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies. pp. 142–150. Association for Computational Linguistics, Portland, Oregon, USA (June 2011), `http://www.aclweb.org/anthology/P11-1015`
[27] Pennington, J., Socher, R., Manning, C.D.: GloVe: Global Vectors for Word Representation. In: Empirical Methods in Natural Language Processing (EMNLP). pp. 1532–1543 (2014), `http://www.aclweb.org/anthology/D14-1162`
[28] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
[29] Svozil, D., Kvasnicka, V., Pospichal, J.: Introduction to multi-layer feed-forward neural networks. Chemometrics and intelligent laboratory systems **39**(1), 43–62 (1997)