A Cross-Layer Review of Deep Learning Frameworks to Ease Their Optimization and Reuse

Hamid Tabani*, Roger Pujol*†, Jaume Abella* and Francisco J. Cazorla*
* Barcelona Supercomputing Center † Universitat Politècnica de Catalunya

***Abstract*—Machine learning and especially Deep Learning (DL) approaches are at the heart of many domains, from computer vision and speech processing to predicting trajectories in autonomous driving and data science. Those approaches mainly build upon Neural Networks (NNs), which are compute-intensive in nature. A plethora of frameworks, libraries and platforms have been deployed for the implementation of those NNs, but end users often lack guidance on what frameworks, platforms and libraries to use to obtain the best implementation for their particular needs. This paper analyzes the DL ecosystem providing a structured view of some of the main frameworks, platforms and libraries for DL implementation. We show how those DL applications build ultimately on some form of linear algebra operations such as matrix multiplication, vector addition, dot product and the like. This analysis allows understanding how optimizations of specific linear algebra functions for specific platforms can be effectively leveraged to maximize specific targets (e.g. performance or power-efficiency) at application level reusing components across frameworks and domains.**

***Index Terms*—Deep Learning, Frameworks, Linear Algebra**

I. Introduction

AI and Deep Learning (DL) approaches are at the heart of a variety of domains. Computer vision algorithms such as image classification and object detection are just among many algorithms that are built relying on DL since, in many cases, DL approaches provide much higher accuracy than any other alternative solution and, in some domains, they are the only existing solution. Given the widespread use of these approaches, the need for an accelerated AI development with a seamless path from prototyping to deployment is inevitable. In this context, DL frameworks offer access to building blocks for designing, training, validating, and testing DL models, through a high-level and simplified programming interface.

Widely used DL frameworks such as MXNet [1], PyTorch [2], TensorFlow [3], Keras [4], Caffe [5] and others provide implementations for CPUs by using libraries such as openBLAS [6], ATLAS [7] and Intel MKL [8], and implementations for GPUs with libraries such as cuBLAS [9] and cuDNN [10] to deliver high-performance multi-GPU accelerated training. Developers can get easy access to optimized DL framework containers from vendors such as NVIDIA. This eliminates the need to manage packages and dependencies or build DL frameworks from scratch.

While there are abundant frameworks, mostly building upon Neural Networks (NNs), as well as target platforms and platform-dependent libraries, each domain tends to focus on specific frameworks, platforms and libraries, thus missing opportunities for improved efficiency due to the lack of a global view of the problem and the ecosystem of components.

This paper analyzes the ecosystem of DL related components providing a structured view to help end users understand the landscape of choices available for their applications. Then, by relating those components and analyzing their implementations, we show how cross-domain reuse may enable increased efficiency for applications in different domains, and eases the process of determining those few low-level operations that need optimization for a given high-level goal.

II. Deep Learning Ecosystem Structured View

In this section, we provide a structured top-down view of the components for a DL based solution, spanning from the top-level NNs regarded as appropriate for the target application by end users, and reaching down to linear algebra operations that ultimately implement those applications. Note that our analysis aims at illustrating the different existing layers, but not being exhaustive in any of those since the number and type of frameworks, platforms and libraries is too large to be exhaustively covered in this short paper.

The framework. The choice of the DL framework to use is usually independent from the DL approach itself. Different DL frameworks provide different advantages such as flexibility, or simpler interfaces for instance. The users decide what framework fits better their needs for the development of their model. As an illustrative example, Keras [4] and TensorFlow [3] are both very well-known frameworks. Keras was developed to be more user-friendly with high-level APIs and a modular structure. However, sometimes the user needs to define something new such as a cost function, a metric, a layer, etc. Although Keras has been designed to be flexible, low-level libraries of TensorFlow provide further flexibility.

Most of the DL frameworks use similar libraries to perform the low-level operations, and they mainly differ at higher levels. For instance, unlike TensorFlow, the PyTorch framework uses a dynamically updated graph, which allows the user to make changes to the architecture in the process. Still, TensorFlow has a level of abstraction similar to that of MXNet, Theano, and PyTorch. In this level, mathematical operations such as Generalized Matrix-Matrix multiplication and NN primitives are implemented. Keras, instead, works at a higher abstraction level where the lower level primitives are used to implement NN abstractions such as various layers and models. In general, other useful APIs such as model saving and model training are implemented at this level.

The platform. Different frameworks offer implementations for different hardware platforms, for both training and inference. For instance, TensorFlow provides implementations for both CPU and GPU, and the user can choose the target hardware platform. The framework is then installed and run on the chosen platform accordingly. In the case of some specialized hardware platforms such as the NVIDIA Deep Learning Accelerator (NVDLA), additional libraries are required to launch and manage the operations on the platform. In the case of NVDLA, NVIDIA's TensorRT [11] is used to launch and manage the workloads on the NVDLA.

Low-level libraries. Once the hardware platform is selected, a platform-dependent optimized library is used, which includes the majority of the required low-level functions implemented and optimized for that specific target platform. For instance, *ATLAS* [7] and *openBLAS* [6] are two well-known libraries for CPUs, as well as *cuBLAS* [9] and *cuDNN* [10] for NVIDIA GPUs.

At this low level, different operations are performed by calling the corresponding functions of a library providing their required parameters. Such compute-intensive operations are linear algebra operations (mostly matrix operations) whose platform-dependent implementations are provided by the corresponding libraries. Normally, these implementations have been optimized with average performance in mind.

III. A Deeper View of the Low-level Libraries

We have analyzed some of the most popular DL libraries such as TensorFlow, Keras, MXNet, Caffe and PyTorch, as illustrative examples of this type of frameworks, considering both, CPU and GPU target platforms. The computations are translated into operations such as vector addition, scalar multiplication, dot products, linear combinations, and matrix operations with different dimensions. DL frameworks do not implement those operations directly but, instead, build upon low-level libraries providing efficient implementations (mostly targeting optimized average performance) for different CPU and GPU target platforms.

So far, DL approaches have been mostly used for applications where the main concern is performance and energy, with both objectives being achieved with almost identical optimizations. This relates to the fact that, performing the same computations in shorter time, as needed for average performance reasons, often leads to lower energy consumption due to the lower active time of the platform.

However, the increasing number of applications of DL approaches in different domains brings additional non-functional metrics to be optimized. For instance, autonomous navigation in cars, planes, satellites and drones has some degree of criticality since malfunctioning may cause fatalities or large economical losses. Moreover, the performance and power requirements of those systems may not need to be optimized but kept within specific budgets.

It is expected that those new goals together with new platforms needed in those domains, pose the need for new low-level libraries targeting, for instance, security or fault-tolerance for specific embedded platforms with limited power envelopes and specific deadlines. Thus, DL frameworks will have to further consider additional low-level libraries and the overall ecosystem is expected to grow.

Hence, it is particularly important to reuse efforts across domains and across platforms whenever possible. Low-level libraries build on linear algebra operations such as matrix multiplication, and hence, frameworks and applications above also end up inheriting the features of the linear algebra operations used in the lowest levels of the stack. Thus, the key to meet the requirements of applications ultimately lies on the ability to optimize specific matrix operations and other linear algebra operations for the different needs of applications, and let frameworks in-between act as interfaces to ease programmability while using highly optimized operations from the low-level libraries. It is, therefore, of paramount importance to devote efforts to the development and optimization of linear algebra operations for different goals such as average performance, worst-case execution time, energy consumption, security, fault-tolerance, testability, and maintainability among others.

IV. Conclusions

This paper presents a structured view of the different components composing the DL software ecosystem. DL frameworks are ultimately implemented with a series of linear algebra operations building upon some highly optimized low-level libraries. As a result, optimizing the low-level libraries can have a direct impact on the performance and testability, among other non-functional metrics, of the frameworks and applications at the highest level. This helps to focus on optimizing low-level libraries (or developing new ones) targeting performance, power, time-predictability, fault-tolerance or any other metric, and reusing libraries and frameworks across domains to maximize benefits.

Acknowledgment

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness under grant TIN2015-65316-P, the SuPerCom European Research Council (ERC) project under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 772773), and the HiPEAC Network of Excellence.

References

[1] T. Chen et al., "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.
[2] A. Paszke et al., "Automatic differentiation in pytorch," 2017.
[3] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: http://tensorflow.org/
[4] F. Chollet *et al.*, "Keras," https://keras.io, 2015.
[5] Y. Jia et al., "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014.
[6] "An optimized BLAS library (OpenBLAS)," http://www.openblas.net/.
[7] "Automatically Tuned Linear Algebra Software (ATLAS)," http://math-atlas.sourceforge.net/.
[8] "Intel, Math Kernel Library," https://software.intel.com/en-us/intel-mkl.
[9] "cuBLAS," http://docs.nvidia.com/cuda/cublas/.
[10] S. Chetlur et al., "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.
[11] "TensorRT: A platform for high-performance deep learning inference." [Online]. Available: https://developer.nvidia.com/tensorrt