Graph compilers for AI training and inference

Karthee Sivalingam, Nina Mujkanovic

– CRAY EMEA Research Lab

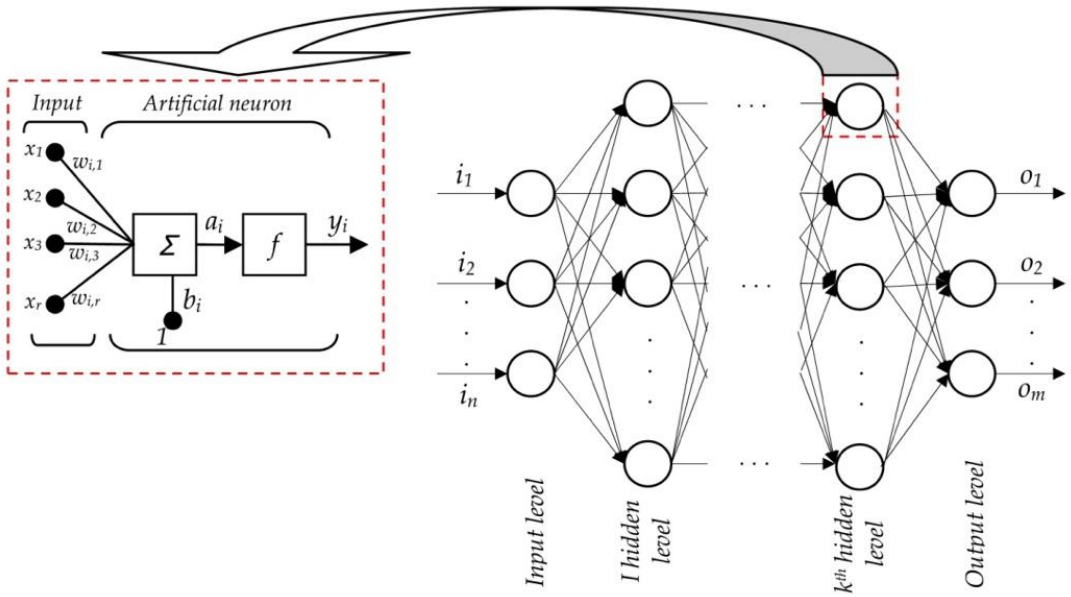

Figure 1 The structure of the feed-forward artificial Deep Neural Network (DNN) (image credit)

Introduction

Artificial Intelligence, the attempt to create an intelligent agent, gained popularity with the success of AlexNet for image recognition in 2012. The proposed combination of stacked neural networks (called Deep Neural network) and the use of GPUs to drastically reduce network training times compared to previous iterations, led to a boom in the use of such networks for image recognition and speech recognition.

The many frameworks since developed for Machine Learning (ML) – especially Deep Learning (DL) - have enjoyed varying levels of success. With dataflow at the heart of most of these computations, efforts have been made to improve performance through graph compilation techniques. In this blog post, we review the currently most popular ML frameworks, as well as the

supported graph compilers, with a view towards using lessons learned for the creation of an application optimiser component.

DL Frameworks

Deep Learning is a subset of Machine Learning centered on the use of Deep Neural Networks (DNN), or multiple layers of neural networks (as shown in Figure 1), which progressively extract higher level features from raw data. This makes them particularly useful for image recognition, speech recognition, natural language processing, and similar problems.

DNN models are usually represented as computational graphs, with nodes representing tensor operators, and edges the data dependencies between them. This computational graph is then used to further optimise for different hardware back-ends. Optimisations may include operator fusion, memory latency hiding, and mapping to hardware primitives.

DL Frameworks, which aid in the designing, training, and validating of DNNs, depend on optimized libraries for most tensor operations, which is not ideal, as the libraries do not always support the new operators created through kernel fusion optimisations. This is further complicated by the diversity in memory hierarchy and accelerator hardware, as well as the layout and dimensionality of input data. Autotuning based approaches are very costly as the search space of optimisation is very high and there is no accurate model for modern hardware.

Table 1 maps popular DL frameworks to their support of graph compilers for training (T) and inference (I). Training refers to the use of labelled or unlabeled data to learn model parameters, whereas Inference refers to the use of the trained model to perform predictions on previously unseen data. Training a model is computationally intensive and may require multiple days on multiple GPUs. Inference is usually performed on an edge device, such as a mobile or car, and should ideally complete within milliseconds. Despite differences regarding computational requirements and time to solution, Inference and Training share the same computation graph and graph compiler based optimisation techniques.

Popular Machine/Deep Learning frameworks include Google's TensorFlow, Facebook's PyTorch, Apache's MXNet (supported by cloud services including AWS and Microsoft Azure), Microsoft's Cognitive Toolkit, MILA's Theano (currently not in active development), and the INRIA supported Scikit-learn. TensorFlow, likely most popular of those listed, provides a low-level API for graph construction, but recommends using the high-level Keras API. Facebook's Caffe2 is another popular DL framework that has now been merged with PyTorch.

Graph Compilers

Graph compilers optimises the DNN graph and then generates an optimised code for a target hardware/backend, thus accelerating the training and deployment of DL models. We have reviewed a number of compilers, including XLA, TC, TVM, ONNC, GLOW, Intel nGraph, PlaidML, and TensorRT.

The XLA [1] (Accelerated Linear Algebra) compiler accelerates linear algebra computations in TensorFlow models and achieves a 1.15x speedup when enabled on standard benchmarks.

Framework / JIT Compilers	XLA	TC	TVM	ONNC	GLOW	Intel nGraph	Plaid ML	Tensor RT
TensorFlow	T	N	I	I	N	T I	T I +	I
PyTorch	N	T I *	I	I	T	T I	N	I
MXNet	N	N	I	I	N	T I	N	I
CNTK	N	N	I	I	N	N	N	I
Theano	N	N	N	N	N	N	N	N
Keras	N	N	I	I	N	N	T	N
SciKit	N	N	N	I	N	N	N	N

Table 1 Popular AI frameworks and support for graph compilers

T – Training I - Inference N – No support

* requires rewrite + added to Intel nGraph

Tensor Comprehensions (TC) [3] aims to improve performance of custom new operators that are not yet fully supported. It provides a mathematics-like language to represent operators, using

polyhedral JIT compilation and autotuning. TC supports Caffe2 and PyTorch and mainly focuses on optimisation across operators, and for data layout and size. TC has been evaluated on multiple popular kernels and achieves up to 4x speedup compared to Caffe2 + CUBLAS.

The TVM [2] compiler is based on Halide's compute/schedule separation concept. TVM introduces a new tensor expression language (DSL) to construct tensor operators, which is then converted to optimised kernels for different target hardware. TVM currently supports TensorFlow, MXNet, PyTorch, Keras, and CNTK on CPUs, GPUs, and FPGAs (embedded and server). It achieves portable performance of about 1.2x to 3.8x.

ONNX (Open Neural Network Exchange) is a standard open format for defining and representing deep learning models. This allows AI developers to port models across DL frameworks or use combinations that best suit their needs. This community project, created by Facebook and Microsoft, has gained support by a number of industry partners. ONNC[5] (Open Neural Network Compiler) is a retargetable compiler (built on top of LLVM) that supports compiling ONNX based models to any supported hardware like CPU, GPU, FPGA, DSP.

GLOW [4] optimises Neural Networks by lowering the graph to two intermediate representations. Glow works with PyTorch and supports multiple operators and targets. Glow can consume ONNX (open standard for serializing AI model) as an input and thus can support other frameworks. GLOW has been found to offer 2.7x speedup over TensorFlow (with XLA) and 1.3x over TVM (no autotuning) for ResNet50.

Intel nGraph is an end-to-end compiler for training and inference and supports TensorFlow, MXNet, ONNX, and PaddlePaddle. nGraph can deliver as much as a 45x increase in normalized inference throughput leveraging MKL-DNN on Intel Xeon Scalable processors.

PlaidML, a compiler for deep learning, is also available as a component of the Intel nGrpah compiler stack. PlaidML supports Keras, ONNX and accelerates by auto generating tiled code with performance comparable to CUDA on NVIDIA GPUs. This is useful mainly for supporting new kernels that are not supported by libraries.

NVIDIA's TensorRT compiler is built on top of CUDA and optimises inference by providing high throughput and low latency for deep learning inference applications. TensorRT supports ONNX, thus by extension supporting models trained by different frameworks. Providing optimisations based on reduced precision, these TensorRT models are found to provide 40x faster performance compared to CPU-only platforms during inference.

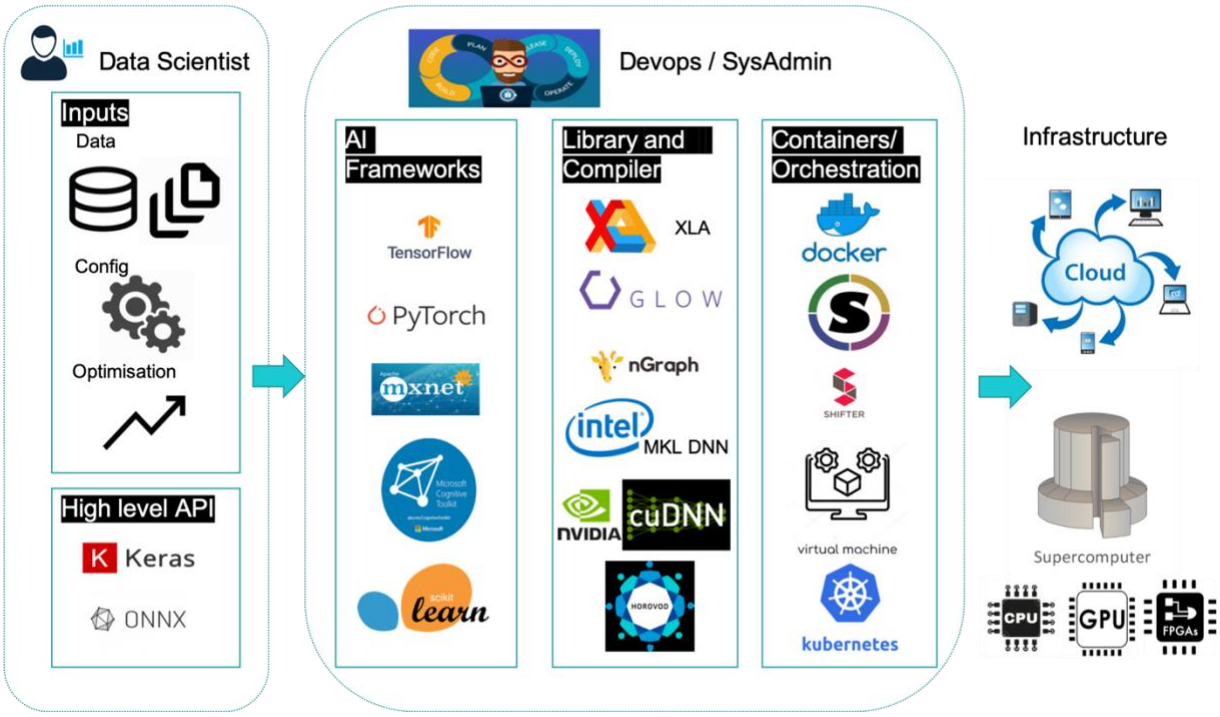

Figure 2 Production deployment workflow for an AI application

DL Benchmarks

Having good performance comparison based on standard benchmarks will help the community in selecting tools for specific deep learning models. There is an abundance of benchmarks a researcher can choose from and obtaining portable performance measurements on different frameworks and hardware is mostly an exercise in trial and error. Benchmarks[6] like MLPerf, DeepBench, and DAWNBench have been helpful in such efforts. MLPerf [7] can benchmark a broad range of DL applications, such as Image Classification, Object Detection, Translation, Recommendation, Sentiment Analysis, and Reinforcement Learning. MLPerf has separate suites of benchmarks for Training and Inference. Comparably, DAWNBench supports only two types of application classes - namely Image Classification and Question-Answering –, focusing instead on optimisation strategies and hyperparameter tuning. DeepBench focuses on underlying kernels and operations like dense matrix multiplication, convolutions, recurrent layers, and communication, instead of using the entire application.

DL Deployment

Figure 2 shows the workflow of deploying a DL application in production. The complexity of heterogenous infrastructure (HPC, Cloud) and hardware (CPU, GPU, FPGA) should be abstracted for the data scientist to enable seamless research. Container technologies like Docker and Singularity provide portable runtime for applications to run on heterogenous infrastructures. With multiple options for DL frameworks, Graph compilers, and libraries, the optimisation of an application and its deployment becomes even more important for both scientific throughput and savings in resource cost and energy.

In the SODALITE EU Project, we aim to solve these challenges by developing an Application Optimiser component that uses the performance model of an application based on DL benchmarks to optimise its runtime and deployment for heterogenous infrastructure and hardware. Based on the inputs from the data scientist about the configuration of the application and the optimisations to be enabled, the application optimiser component will select the optimised framework, compiler, and libraries, and build an optimised container. Also, the application parameters and hyperparameters will be chosen based on the performance model or auto tuned during run time. This optimised application in a container will then be deployed to cloud or HPC infrastructure.

References

[1] C. Leary and T. Wang, "Xla: Tensorflow, compiled," *TensorFlow Dev Summit*, 2017.

[2] T. Chen, T. Moreau, Z. Jiang, H. Shen, E. Q. Yan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Tvm: end-to-end opti- mization stack for deep learning," arXiv preprint arXiv:1802.04799, pp. 1–15, 2018.

[3] N. Vasilache, O. Zinenko, T. Theodoridis, P. Goyal, Z. DeVito, W. S. Moses, S. Verdoolaege, A. Adams, and A. Cohen, "Tensor comprehen- sions: Framework-agnostic high-performance machine learning abstrac- tions," arXiv preprint arXiv:1802.04730, 2018.

[4] N. Rotem, J. Fix, S. Abdulrasool, S. Deng, R. Dzhabarov, J. Hegeman, R. Levenstein, B. Maher, N. Satish, J. Olesen, J. Park, A. Rakhov, and M. Smelyanskiy, "Glow: Graph lowering compiler techniques for neural networks," CoRR, vol. abs/1805.00907, 2018. [Online]. Available: http://arxiv.org/abs/1805.00907

[5] W. F. Lin, D. Y. Tsai, L. Tang, C. T. Hsieh, C. Y. Chou, P. H. Chang, and L. Hsu, "ONNC: A compilation framework connecting ONNX to proprietary deep learning accelerators," in IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS 2019). IEEE, 2019.

[6] S. Verma, Q. Wu, B. Hanindhito, G. Jha, E. B. John, R. Radhakrishnan, and L. K. John, "Demystifying the mlperf benchmark suite," arXiv preprint arXiv:1908.09207, 2019.

[7] Mattson, P., Cheng, C., Coleman, C., Diamos, G., Micikevicius, P., Patterson, D., ... & Brooks, D. (2019). MLPerf Training Benchmark. arXiv preprint arXiv:1910.01500.