Optimization Mapping for Deep Learning

Wencong Xiao[†*], Cheng Chen[*], Youshan Miao[*], Jilong Xue[*], Ming Wu[*]
[†]SKLSDE Lab, Beihang University, [*]Microsoft Research

Abstract

The growing importance of deep learning has driven its deployments in extensive application scenarios and environments, and hence led to diversified optimizations through both customized hardware and novel model algorithms. However, this introduces burden for developers to incorporate these specific optimizations into existing models. We propose an *optimization mapping* framework that isolates the algorithm expressions from customized optimizations through providing a data flow graph based pattern representation and automatic pattern matching to map sub-graphs of deep learning computation to optimized implementations. The preliminary results on TensorFlow [6] show that optimization mapping can automatically identify a multi-layer Long Short Term Memory (LSTM) model from the data flow graph of applications and replace it with a cuDNN [9] based LSTM operator, resulting in 4.12x performance improvement.

1. Introduction

Deep learning frameworks [1, 3, 6, 7], which transform user script into data flow graph consisting of primitive operators and tensors, witness the proliferation of various models and applications in achieving great success for more and more scenarios, leading to the burst of deep learning specific optimizations.

We observe that the actual adoption of an optimization normally goes through a common process. Usually, novel deep learning models are invented and developed using existing primitive operators in a deep learning framework at the beginning. Once proved practical in the real industry scenario, their bottlenecks, often identified as sub-graphs, are investigated and the corresponding optimizations are applied.

These optimizations often have the feature of customization due to the following two factors. First, as the slowdown of CPU scaling [12, 16], recent trend advocates GPU, FPGA, and even ASIC-based hardware [8, 9, 11, 14, 15] to act as the accelerators for deep learning, achieving often orders of magnitude improvement on performance and power efficiency. Since customized hardware usually have their own design goals and limitations, they are often used to design customized operators for variant sub-graphs in different models. For example, ShiDianNao [11] focuses on accelerating Convolutional Neural Network (CNN) and cuDNN [9] has optimized Recurrent Neural Network (RNN) implementation. We envision more variant optimized components in deep learning computation to appear in near future.

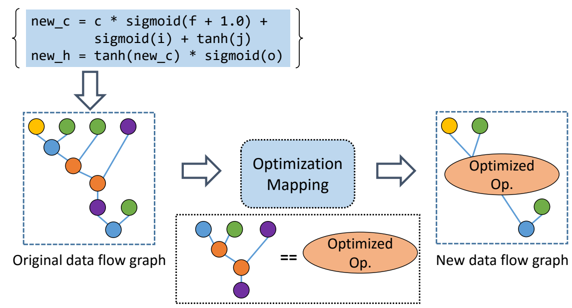


Figure 1: System overview of optimization mapping.

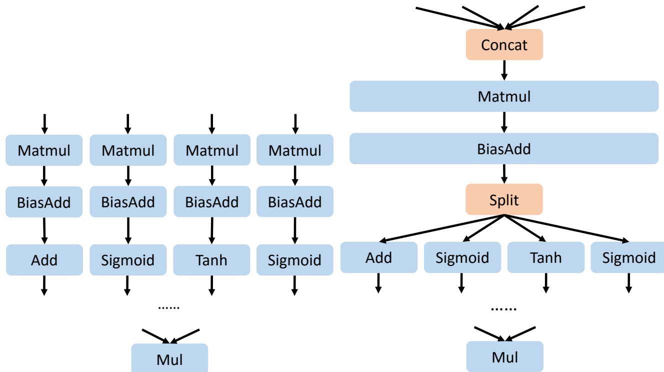
Secondly, optimizations may target for different application scenarios and environments, e.g., in cloud or mobile, which diversifies the optimization objectives considering different resource constraints and performance metrics. For instance, MobileNets [13] and ShuffleNet [18] adopt novel efficient structures for convolution network, keeping roughly the same accuracy while significantly reducing the model parameters for mobile devices.

Therefore, applying those new customized optimizations into legacy deep learning model code is not effort-free, but may require non-trivial code refactoring which significantly increases the maintenance overhead.

In this paper, as indicated in Figure 1, we propose an *optimization mapping* scheme to isolate the general deep learning algorithm expression from the customized optimizations specific to ad-hoc hardware or scenarios, through automatically applying optimized implementation over the data flow graph intermediate representation in a deep learning framework.

2. Optimization mapping

Suppose we already have an optimized operator along with its operation pattern in a unified representation. For an application data flow graph that is also in the unified intermediate representation, our optimization mapping framework can perform patten matching to identify matched sub-graphs



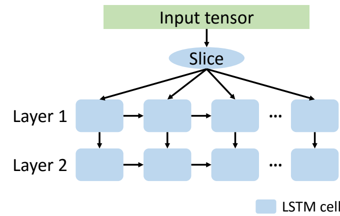
(a) Native LSTM. (b) Concat-split optimized LSTM.

Figure 2: Equal expression in LSTM.

and replace with the optimized operator, so as to apply such optimization.

For pattern matching, both the data flow graph and the pattern graph should be expanded into canonical form [2] to avoid the mismatch of equivalent linear algebraic expressions due to non-isomorphism graph structure (e.g. $(AB)^T = B^T A^T$). As for linear algebraic expression, polynomial expansion [4] can help to generate canonical form graph. However, some non-mathematic operators in deep learning data flow graph would also introduce isomerism. Figure 2a shows part of graph for the native LSTM. It contains four *Matmul–BiasAdd* pairs for the gate calculation in LSTM. Another equivalent expression is to use a *Concat* and a *Split* to merge these four *Matmul–BiasAdd* pairs together and get larger but less matrix operations, as shown in Figure 2b, which is more likely to outperform the previous one in performance for higher parallelism. Through the tensor dimension shape parameters in *Concat* and *Split* operators, we can expand it to the native version. We choose the native version as "canonical form" here.

In addition, we provide a powerful and flexible interface for users to define matching pattern, as illustrated in Figure 3b. It contains three fields. Field *cell* describes the operator attributes and topology information as a graph for pattern matching. Field *op_type* describes the optimized operator, including its type and parameters. The *cell_input* field describes the input of one *cell* to indicate their dependencies in this pattern. Optimization mapping not only supports fixed algorithm operator set as the pattern, but also dynamic recurrent operator set. The existence of *cell_input* field can indicate recurrent pattern. A concreted example is multi-layer LSTM, which contains multiple identical LSTM cells, as indicated in Figure 3a. A LSTM cell typically takes two inputs: one from the other cell, the other from the input or upper layer cell. Currently we assume the matching cells are organized in such 2D grids. Therefore, after finding out all matched sub-graphs, optimization mapping is able to figure out the steps per layer by counting the cells with external input. As indicated in *cell_input*

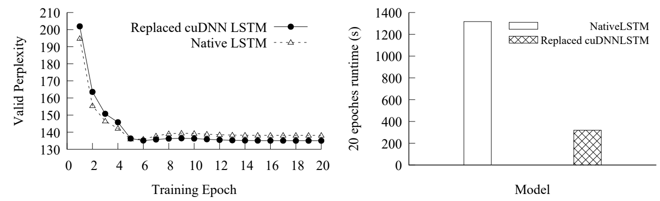


(a) Multi-layer LSTM structure.

```
cuDNN_LSTM_pattern{
    "cell": { … },
    "op_type": { … },
    "cell_input": [
        {external_input | other_cell},
        {other_cell}
    ]
}
```

(b) Sample of pattern for Figure 3a.

Figure 3: Optimization mapping pattern definition example.



(a) Convergence curve. (b) Performance.

Figure 4: Optimization mapping for cuDNN LSTM.

for two dimensions, the layer parameter can be calculated by: $\#matched\_cell$ / $\#per\_layer\_step$. Then optimization mapping is able to feed these input parameters during optimized operator replacement.

In optimization mapping, data-flow graph pattern matching, also known as NP-complete sub-graph isomorphism problem [10], is the most time consuming step. Fortunately, we can leverage some characteristics of deep learning algorithms to reduce the computation significantly. First, the data flow graph is heterogeneous, often consists of tens or hundreds of different operators types that reduce the search space. Second, the output of the operator may be used by many other operators, while the inputs of operator are much less. Such as matrix multiply and sigmoid, these mathematic operators or activation functions are either binary expression or unary expression. Searching multiple low degree nodes for matching is cheaper than searching less high-degree nodes. Especially the heterogeneous information can help to filter out a lot of nodes. Thus, matching from the output side back to input side in data flow graph can further reduce the search space.

3. Preliminary result

We have implemented the optimization mapping as an optimizer in the open-source version Tensorflow(r1.3). With such an implementation module, we define a pattern mapping from basic LSTM [17] to cuDNN LSTM and conduct an experiment on PTB dataset [5], using a GPU server equipped with dual 2.6GHz Intel Xeon E5-2650 processors, 128GB of memory, a NVIDIA GTX1080 GPU, to demonstrate the benefit.

Figure 4a and Figure 4b show the convergence curve and the runtime of native LSTM and the replaced cuDNN LSTM, with 2 layers and 20 steps for 20 epoches. The replaced cuDNN LSTM version comes up with roughly the same convergence curve while achieves 4.12x performance speedup, without application level code modification.

References

[1] Caffe2. https://caffe2.ai/.

[2] Canonical form. https://en.wikipedia.org/wiki/Canonical_form.

[3] CNTK. http://www.cntk.ai/.

[4] Polynomial expansion. https://en.wikipedia.org/wiki/Polynomial_expansion.

[5] Tensorflow RNN benchmark on PTB. https://github.com/tensorflow/models/tree/master/tutorials/rnn/ptb/.

[6] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*.

[7] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[8] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, et al. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE Computer Society, 2014.

[9] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[10] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[11] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam. Shidiannao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 92–104. ACM, 2015.

[12] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 365–376. IEEE, 2011.

[13] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[14] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. luc Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, and J. Ross. In-datacenter performance analysis of a tensor processing unit. 2017. URL https://arxiv.org/pdf/1704.04760.pdf.

[15] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen. Cambricon: An instruction set architecture for neural networks. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 393–405. IEEE Press, 2016.

[16] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobbs journal*, 30(3):202–210, 2005.

[17] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[18] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.