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.

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 pattern matching to identify matched sub-graphs...
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

![Figure 2: Equal expression in LSTM.](image)

![Figure 3: Optimization mapping pattern definition example.](image)

![Figure 4: Optimization mapping for cuDNN LSTM.](image)
same convergence curve while achieves 4.12x performance speedup, without application level code modification.

References