Optimizing Computational Graphs Using Relaxed Graph Substitutions

Kaushik Bhowmick
u21cs022@coed.svnit.ac.in

Himanshu Thakur
u21cs046@coed.svnit.ac.in

Bhavya Hirani
u21cs100@coed.svnit.ac.in

September 30, 2024

Abstract

Most deep learning compilers use a greedy approach to optimizing computational graphs in that every successive graph substituion must result in a strict performance increase. This narrows the space of possible optimizations and motivates the notion of relaxed graph substitutions: substitutions that don't themselves result in a more optimal graph version, but facilitate further transformations down the line that ultimately yield a net improvement in performance. In order to determine the pipeline of transformations to be applied such that the net performance increase is guaranteed, we search the space of potential optimizations using a backtracking algorithm which, at a given iteration, determines the cost of the corresponding intermediate graph using programmer-defined heuristics. To test our optimization strategy, we plan to use the set of graph transformations available in TensorFlow's repository and we limited our performance heuristics to simply measuring execution time. Our work is based on the strategy presented in [2].

Chapter 1

Introduction

We use an example from a ResNet [1] module (figure 1.1) to illustrate our strategy. Each arrow represents a graph substitution that produces a graph *equivalent* to its input, where equivalence is defined as the following: two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are said to be equivalent if they produce the same outputs for an arbitary input. The first substitution pads a 1x1 convolutional kernel to form a 3x3 one, potentially reducing the performance by introducing more operations. However, doing so enables further node fusions which, as we can see, results in a computational graph giving better performance than what we started with.

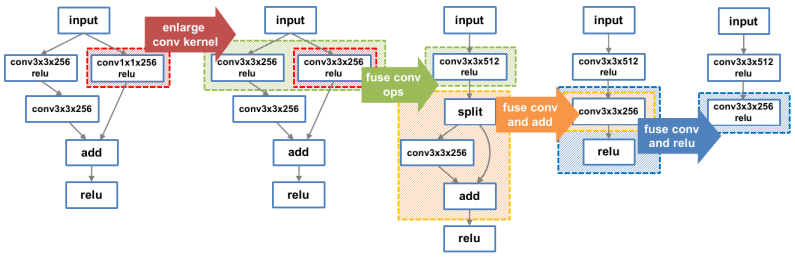

Figure 1.1: An example from [2] that demonstrates a relaxed graph substitution

Chapter 2

Theoretical Background and Literature Review

Chapter 3

Proposed Algorithm and Future Directions

3.1 Motivating our Approach Using an Example

3.2 The backtracking Algorithm

We take the following approach to search through the space of possible graph transformations:

1. Consider a list of possible graph transformations, T. At any given point in the algorithm, let $\mathcal{G}$ be the current intermediate graph version. We maintain a priority queue of intermediate graphs, ordered by their *cost*, defined by a heuristic (in our case, this is simply the execution time).
2. In each iteration:
   (a) We dequeue to get the next candidate intermediate representation $\mathcal{G}$.
   (b) For each optimization t_i in T, we apply t_i on $\mathcal{G}$ to obtain some equivalent graph $\mathcal{G}'$. If $cost(\mathcal{G}') < cost(\mathcal{G})$, we set $\mathcal{G} = \mathcal{G}'$. Else if $cost(\mathcal{G}') < \alpha \cdot cost(\mathcal{G})$, where α is a hyperparameter defined by the user (greater than one), then $\mathcal{G}'$ is considered to be a valid candidate for further search, and is added to the priority queue.
   (c) α helps define how much of a performance decrease we can tolerate. This value can't be too high so as to avoid blowing up the search space, and hence, the time it takes for our algorithm to find a beneficial sequence of graph transformations.
3. The above is repeated until there are still graphs to process in the queue.

Chapter 4

Conclusion

Bibliography

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[2] Zhihao Jia, James Thomas, Todd Warszawski, Mingyu Gao, Matei Zaharia, and Alex Aiken. Optimizing dnn computation with relaxed graph substitutions. *Proceedings of Machine Learning and Systems*, 1:27–39, 2019.

Shri. R. P. Gohil
Associate Professor, DoCSE, SVNIT