Optimizing DNN Computation with Relaxed Graph Substitutions

Tim Lazarus

26 November, 2019

Graph Substitutions

We can optimise DNNs if we replace subgraphs with equivalent ones that improve overall performance

For a particular input $\mathcal{I}$, computation graph $\mathcal{G}$ will produce output $\mathcal{O}$, or written as $\mathcal{O} = \mathcal{G}(\mathcal{I})$

We then say that two graphs, $\mathcal{G}$ and $\mathcal{G}'$ are *equivalent* if they produce the same output for every input. $(\forall \mathcal{I} : \mathcal{G}(\mathcal{I}) = \mathcal{G}'(\mathcal{I}))$

Relaxed Graph Substitutions

This is a local form of optimisation and may not result in optimal results.

Previous work with graph substitutions employed a *greedy approach*.

As with most modern optimising compilers, sometimes further optimisations can be gained if we *decrease performance* in intermediate steps.

Example

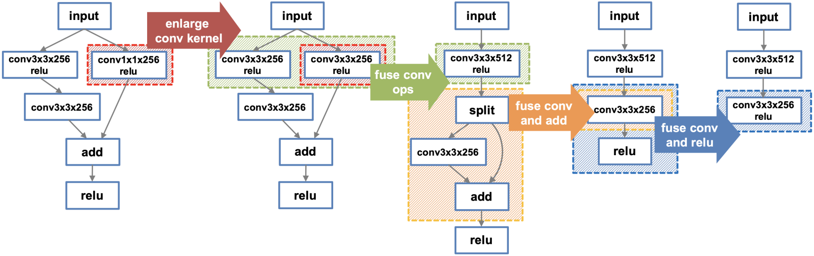


Figure: Example relaxed graph substitution optimisation

Defining substitutions

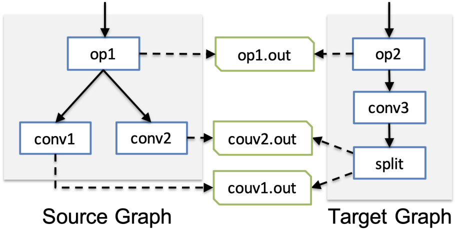
Essentially a mapping between a *source graph* and *target graph*.

Source graph defines constraints on a subgraph.

Target graph uses those constraints to create the substituted subgraph.

We need the substitution to be *valid*

Example




```
# Constraints on the source graph:
conv1.kernel == conv2.kernel
conv1.stride == conv2.stride
conv1.padding == conv2.padding
```

```
# Construct the target graph:
op2._ = op1._
conv3._ = conv1._
conv3.outChannels = conv1.outChannels + conv2.outChannels
conv3.weights = concat(conv1.weights, conv2.weights)
split.sizes = [conv1.outChannels, conv2.outChannels]
```

Figure: Example substitution definition

Cost Model

We need to estimate the cost of each substitution.

Cost model incorporates many metrics.

Can also accurately estimate dynamic execution too

Searching the Space

Use a priority queue to search most optimal graph first and backtrack if necessary.

The space can be huge if we consider all possible substitutions.

Use a parameter α that determines the trade-off between search time and space explored. *(See next slide)*

Search Algorithm

Algorithm 1: A Backtracking Search Algorithm

Input: An initial computation graph $\mathcal{G}_0$, a cost model $Cost(\cdot)$, a list of valid graph substitutions $\{S_1, ..., S_m\}$, and a hyper parameter α

Output: An optimised computation graph.

```
// Q is a priority queue of graphs sorted by Cost(·)
Q = {G_0}
while Q ≠ {} do
    G = Q.dequeue()
    for i = 1 to m do
        G' = S_i(G)
        if Cost(G') < Cost(G_opt) then
            G_opt = G'
        end
        if Cost(G') < α × Cost(G_opt) then
            Q.enqueue(G')
        end
    end
end
return G_opt
```

Graph Splitting

Split the graph into smaller subgraphs so the search is more manageable.

For each node v, we define the $Cap(v)$ as the number of substitutions that map to an in or out edge of v.

We can then minimise the number of substitutions that span across a split as the problem maps to a *minimum vertex cut* problem.

Can perform a local search around splits to find further potential optimisations.

Evaluation

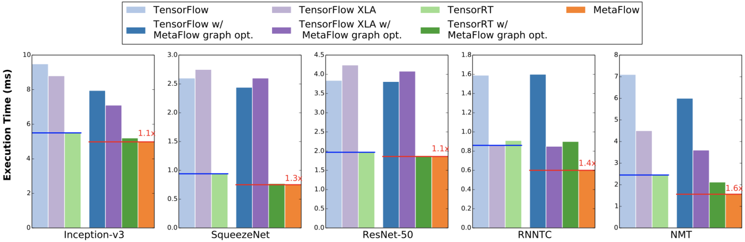


Figure: Compared with TensorFlow, TensorRT and TensorFlow XLA

Evaluation

DNN	Execution Time (ms)		Memory Accesses (GB)		Launched Kernels		FLOPs (GFLOPs)		Device Utilization	
	TensorRT	MetaFlow	TensorRT	MetaFlow	TensorRT	MetaFlow	TensorRT	MetaFlow	TensorRT	MetaFlow
Inception-v3	5.51	**5.00**	95.4	**62.2**	138	**115**	**5.68**	5.69	1.03	**1.14**
SqueezeNet	0.94	**0.75**	62.1	**46.1**	50	**40**	**0.64**	1.00	0.68	**1.35**
ResNet50	1.97	**1.86**	37.2	**35.8**	70	**67**	**0.52**	0.54	0.26	**0.29**
RNNTC	0.91	**0.60**	1.33	**1.17**	220	**83**	0.22	**0.20**	0.24	**0.33**
NMT	2.45	**1.56**	5.32	**4.68**	440	**135**	0.84	**0.78**	0.34	**0.50**

Figure: Comparison of different cost metrics

Evaluation

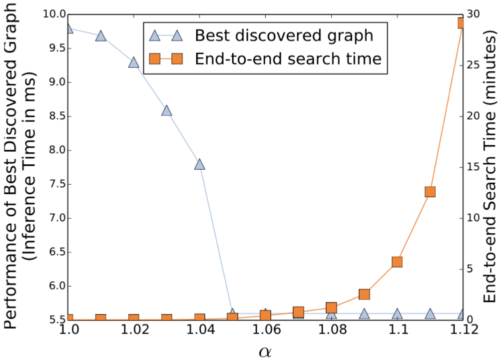


Figure: Evaluation of varying values of α

Criticism

Strengths

- Well defined problem
- System is open-source
- Good testing of system
- Can be used on top of other optimisations

Criticism

Strengths

- Well defined problem
- System is open-source
- Good testing of system
- Can be used on top of other optimisations

Weaknesses

- Paper lacked implementation detail
- Poor analysis of results

Extensions

Can be used with existing optimisations like TVM or FlexFlow (as we saw last week)

There's a new paper in town...

TASO

Extends this paper by automatically generating possible graph substitutions.

For a given set of operators, it enumerates all possible subgraphs up to a fixed size.

It then finds equivalent subgraphs through formal verification.

Questions?