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.
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
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 $\alpha$ that determines the trade-off between search time and space explored. \textit{(See next slide)}
Search Algorithm

**Algorithm 1: A Backtracking Search Algorithm**

**Input:** An initial computation graph \( G_0 \), a cost model \( \text{Cost}(\cdot) \), a list of valid graph substitutions \( \{S_1, \ldots, S_m\} \), and a hyper parameter \( \alpha \)

**Output:** An optimised computation graph.

// \( Q \) is a priority queue of graphs sorted by \( \text{Cost}(\cdot) \)

\[
Q = \{G_0\}
\]

while \( Q \neq \emptyset \) do

\[
G = Q\text{.dequeue()}
\]

for \( i = 1 \) to \( m \) do

\[
G' = S_i(G)
\]

if \( \text{Cost}(G') < \text{Cost}(G_{opt}) \) then

\[
G_{opt} = G'
\]

end

if \( \text{Cost}(G') < \alpha \times \text{Cost}(G_{opt}) \) then

\[
Q\text{.enqueue}(G')
\]

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 \( \text{Cap}(v) \) 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

<table>
<thead>
<tr>
<th>DNN</th>
<th>Execution Time (ms)</th>
<th>Memory Accesses (GB)</th>
<th>Launched Kernels</th>
<th>FLOPs (GFLOPs)</th>
<th>Device Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TensorRT</td>
<td>MetaFlow</td>
<td>TensorRT</td>
<td>MetaFlow</td>
<td>TensorRT</td>
</tr>
<tr>
<td>Inception-v3</td>
<td>5.51</td>
<td>5.00</td>
<td>95.4</td>
<td>62.2</td>
<td>138</td>
</tr>
<tr>
<td>SqueezeNet</td>
<td>0.94</td>
<td>0.75</td>
<td>62.1</td>
<td>46.1</td>
<td>50</td>
</tr>
<tr>
<td>ResNet50</td>
<td>1.97</td>
<td>1.86</td>
<td>37.2</td>
<td>35.8</td>
<td>70</td>
</tr>
<tr>
<td>RNNTC</td>
<td>0.91</td>
<td>0.60</td>
<td>1.33</td>
<td>1.17</td>
<td>220</td>
</tr>
<tr>
<td>NMT</td>
<td>2.45</td>
<td>1.56</td>
<td>5.32</td>
<td>4.68</td>
<td>440</td>
</tr>
</tbody>
</table>

**Figure:** Comparison of different cost metrics
Evaluation

**Figure:** Evaluation of varying values of $\alpha$
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
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...
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?