Adaptive Neural Compilation
Rudy Bunel, Alban Desmaison, Pushmeet Kohli, Philip H.S. Torr, M. Pawan Kumar
Torr Vision Group - OVAL Group, Department of Engineering Science, University of Oxford, UK
Microsoft Research

Problem
How to get the most efficient program for a given task?

Classic approach:
- Write program in high level language.
- Use a generic compiler to get efficient machine code.

Limits:
- Program enforces some behaviours not needed by the user.
- Cannot perform instance specific optimization.

Overview of our solution
- Runtime and Correctness measure programs’ quality.
- Input/Output pairs supervision.

Notations
The State of the program is given by:
- Instruction Register $IR$.
- Registers $R = \{r_1, r_2, \ldots, r_N\}$.
- Memory $M = \{m_1, m_2, \ldots, m_M\}$.

The Command, as one-hot encodings, contains:
- Which instruction $e$ to execute.
- Which registers $a$ and $b$ to use as arguments.
- Which register $o$ where to write the output.

It is generated by the Controller:
- Neural network architecture.
- Set of 4 independant small networks.
- Each is a single linear layer.

- $e = W_e \ast IR$.
- $a = W_a \ast IR$.
- $b = W_b \ast IR$.
- $o = W_o \ast IR$.

Model Execution
One step of the Machine execution:
- Get inputs:
  $arg_1 = \sum_{i=1}^N a_i r_i$ \hspace{1cm} (1)
  $arg_2 = \sum_{i=1}^N b_i r_i$ \hspace{1cm} (2).
- Apply each instruction $k$ to get $out_k$ from $arg_1$ and $arg_2$.
- Compute output:
  $out = \sum_{i=1}^N e' out_i$ \hspace{1cm} (3).
- Update State:
  $r'_i = (1-o_i)r_i + o_i out$ \hspace{1cm} (4)
- Memory updated according to WRITE:
  $M' = arg_1 \cdot arg_2 + (1-arg_1) \cdot M$ \hspace{1cm} (5)

Notations
- $\text{Arg1 dist}$ of the output is given by:
  $W(\text{Arg1 dist}) = \sum_{i,j \leq \text{arg}} (\text{Arg1 dist})_{ij} = c$ \hspace{1cm} (6)
- $W(\text{Arg2 dist}) = \sum_{i,j \leq \text{arg}} (\text{Arg2 dist})_{ij} = c$ \hspace{1cm} (7)
- $W(\text{Out dist}) = \sum_{i,j \leq \text{arg}} (\text{Out dist})_{ij} = c$ \hspace{1cm} (8)

Compilation Step
- Source
- Instruction
- Output

Optimization
- Add Softmax layers to get distributions from unconstrained weights.
- Use plain Adam optimizer.

- $\text{Correctness}$: Match the ground-truth.
- $\text{Halting}$: Should produce the STOP signal.
- $\text{Efficiency}$: STOP as soon as possible.
- $\text{Confidence}$: Penalize bad state when STOP.

Experimental Results
- Average runtime for initial, learnt and ideal algorithms.
- Fraction of training runs leading to good programs.

Tasks
- Access: get the element in an array at a given index.
  To exploit: Index is constant.
- Swap: swap elements from the list given indices.
  To exploit: Indices are constants.
- List: get the $k$-th element in a linked list.
  To exploit: List is contiguous in memory.
- Addition: add two natural numbers.
  To exploit: Given generic program is inefficient.
- Sort: sort the given array in place.
  To exploit: Only first elements are not sorted.
- Increment: add one to each element in an array.
  To exploit: All elements in the array are the same.

Example
- Read and write result.
- Count number of jumps.
- Loop through the list.
- Read and write result.
- Read problem settings.
- Offset compute.
- Read problem settings.

Future works
- REINFORCE-based optimization for simpler model.
- Optimization allowing bigger moves in the program space.
- Optimization that use the structure of the problem.

Supported by Microsoft Research
{rudy, alban}@robots.ox.ac.uk www.robots.ox.ac.uk/~rudy/publications/anc-nips16.html