Neural Programmer-Interpreters
Scott Reed and Nando de Freitas
Neural Programmer Interpreter (NPI) goals:

1. **Long-term prediction**: Model potentially long sequences of actions by exploiting compositional structure.

2. **Continual learning**: Learn new programs by composing previously-learned programs, rather than from scratch.

3. **Data efficiency**: Learn generalizable programs from a small number of example traces.

4. **Interpretability**: By looking at NPI’s generated commands, we can understand what it is doing at multiple levels of temporal abstraction.
Model
NPI training data

Traces:

\[ \xi_t^{inp} : \{ e_t, i_t, a_t \} \quad \xi_t^{out} : \{ i_{t+1}, a_{t+1}, r_t \} \]

- environment observation
- program indices
- program args
- return bit
i_t M_{\text{key}} M_{\text{prog}}
Demos
Adding numbers together - environment

Addition environment interface:
- Scratch pad with the two numbers to be added, a carry row and output row.
- LEFT, RIGHT programs that can move a pointer left or right, respectively.
- WRITE program that writes a specified value to the location of a specified pointer.
- 4 read/write pointers; one per row.
## Adding numbers together – learned programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Multi-digit addition</td>
<td>ADD1, LSHIFT</td>
</tr>
<tr>
<td>ADD1</td>
<td>Single-digit add</td>
<td>CARRY, ACT</td>
</tr>
<tr>
<td>CARRY</td>
<td>Mark a 1 in the carry row 1 step left.</td>
<td>ACT, LSHIFT, RSHIFT</td>
</tr>
<tr>
<td>LSHIFT</td>
<td>Shift specified pointer 1 step left.</td>
<td>ACT</td>
</tr>
<tr>
<td>RSHIFT</td>
<td>Shift specified pointer 1 step right.</td>
<td>ACT</td>
</tr>
<tr>
<td>ACT</td>
<td>Move pointer or write to the scratch pad</td>
<td>-</td>
</tr>
</tbody>
</table>
Adding numbers together
Bubble sort - environment

Sorting environment interface:
- Scratch pad with the array to be sorted.
- Read/write pointers.
- LEFT, RIGHT programs that can move a specified pointer left or right, respectively.
- SWAP program that swaps the values at two specified pointer locations.

Array:

$t=0$: 3 2 4 9 1
$t=1$: 3 2 4 9 1
$t=2$: 2 3 4 9 1
$t=3$: 2 3 4 9 1
# Bubble sort – learned programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUBBLESORT</td>
<td>Sort numbers in ascending order</td>
<td>BUBBLE, RESET</td>
</tr>
<tr>
<td>BUBBLE</td>
<td>Perform one sweep of bubble sort</td>
<td>BSTEP, ACT</td>
</tr>
<tr>
<td>RESET</td>
<td>Move pointers all back to the left</td>
<td>LSHIFT</td>
</tr>
<tr>
<td>BSTEP</td>
<td>Conditionally swap and advance pointers</td>
<td>COMPSWAP, RSHIFT</td>
</tr>
<tr>
<td>COMPSWAP</td>
<td>Conditionally swap two pointer values</td>
<td>ACT</td>
</tr>
<tr>
<td>LSHIFT</td>
<td>Shift specified pointer 1 step left.</td>
<td>ACT</td>
</tr>
<tr>
<td>RSHIFT</td>
<td>Shift specified pointer 1 step right.</td>
<td>ACT</td>
</tr>
<tr>
<td>ACT</td>
<td>Perform a swap or move a pointer.</td>
<td>-</td>
</tr>
</tbody>
</table>
Bubble sort
3D cars environment interface:
- Rendering of the car (pixels).
- Target angle and elevation coordinates.
- LEFT, RIGHT, UP, DOWN programs that can move the car 15 degrees at a time.
- The current car pose is NOT provided.
## 3D car models – learned programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOTO</td>
<td>Change 3D car pose to match target</td>
<td>HGOTO, VGOTO</td>
</tr>
<tr>
<td>HGOTO</td>
<td>Move horizontally to target angle</td>
<td>LGOTO, RGOTO</td>
</tr>
<tr>
<td>LGOTO</td>
<td>Move left to target</td>
<td>ACT</td>
</tr>
<tr>
<td>RGOTO</td>
<td>Move right to target</td>
<td>ACT</td>
</tr>
<tr>
<td>VGOTO</td>
<td>Move vertically to target elevation</td>
<td>UGOTO, DGOTO</td>
</tr>
<tr>
<td>UGOTO</td>
<td>Move up to target</td>
<td>ACT</td>
</tr>
<tr>
<td>DGOTO</td>
<td>Move down to target</td>
<td>ACT</td>
</tr>
<tr>
<td>ACT</td>
<td>Move 15 degrees up, down, left or right.</td>
<td>-</td>
</tr>
</tbody>
</table>
Canonicalizing the view of 3D car models

Car rendering  NPI inference  Generated commands

Output program

Previous NPI state

Environment observation

Input program

NPI Core

Next NPI state

GOTO 1 2

GOTO 1 2
General Artificial Intelligence
General Artificial Intelligence
Experiments
Data Efficiency – Sorting

Seq2Seq LSTM and NPI used the same number of layers and hidden units.

Trained on length 20 arrays of single-digit numbers.

NPI benefits from mining multiple subprogram examples per sorting instance, and additional parameters of the program memory.
Generalization – Sorting

For each length, we provided 64 example bubble sort traces, for a total of 1,216 examples.

Then, we evaluated whether the network can learn to sort arrays beyond length 20.
Generalization – Addition

Example problem: $90 + 160 = 250$, we could represent the sequence as:

$$90 \times 160 \times 250$$

and solve addition via sequence prediction, e.g. “Learning to Execute” paper.
Generalization – Addition problems

To make it easier, we can reverse and stack the inputs. (s2s-stack)

output: XXXX250
input 1: 090XXXX
input 2: 061XXXX

Even easier version: computation is entirely local. (s2s-easy)

output: 052
input 1: 090
input 2: 061
Generalization – Addition problems

Trained on problem sizes 1,...,20 digits.
Generalization – Addition problems

Addition generalization: NPI vs Seq2Seq

- NPI@32 per-sequence
- S2S-stack@32 per-character
- S2S-stack@512 per-character
- S2S-easy@32 per-sequence
- S2S-easy@64 per-sequence

Test sequence length
Generalization – Addition problems

Addition generalization: NPI vs Seq2Seq

Test sequence length

s2s-easy
Generalization – Addition problems

Addition generalization: NPI vs Seq2Seq

Test sequence length

NPI@32 per-sequence
S2S-stack@32 per-character
S2S-stack@512 per-character
S2S-easy@32 per-sequence
S2S-easy@64 per-sequence
Multi-task NPI – Core is shared across all programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Descriptions</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Perform multi-digit addition</td>
<td>ADD1, LSHIFT</td>
</tr>
<tr>
<td>ADDI</td>
<td>Perform single-digit addition</td>
<td>ACT, CARRY</td>
</tr>
<tr>
<td>CARRY</td>
<td>Mark a 1 in the carry row one unit left</td>
<td>ACT</td>
</tr>
<tr>
<td>LSHIFT</td>
<td>Shift a specified pointer one step left</td>
<td>ACT</td>
</tr>
<tr>
<td>RSHIFT</td>
<td>Shift a specified pointer one step right</td>
<td>ACT</td>
</tr>
<tr>
<td>ACT</td>
<td>Move a pointer or write to the scratch pad</td>
<td></td>
</tr>
<tr>
<td>BUBBLESORT</td>
<td>Perform bubble sort (ascending order)</td>
<td>BUBBLE, RESET</td>
</tr>
<tr>
<td>BUBBLE</td>
<td>Perform one sweep of pointers left to right</td>
<td>ACT, BSTEP</td>
</tr>
<tr>
<td>RESET</td>
<td>Move both pointers all the way left</td>
<td>LSHIFT</td>
</tr>
<tr>
<td>BSTEP</td>
<td>Conditionally swap and advance pointers</td>
<td>COMPSWAP, RSHIFT</td>
</tr>
<tr>
<td>COMPSWAP</td>
<td>Conditionally swap two elements</td>
<td>ACT</td>
</tr>
<tr>
<td>LSHIFT</td>
<td>Shift a specified pointer one step left</td>
<td>ACT</td>
</tr>
<tr>
<td>RSHIFT</td>
<td>Shift a specified pointer one step right</td>
<td>ACT</td>
</tr>
<tr>
<td>ACT</td>
<td>Swap two values at pointer locations or move a pointer</td>
<td></td>
</tr>
<tr>
<td>GOTO</td>
<td>Change 3D car pose to match the target</td>
<td>HGOTO, VGOTO</td>
</tr>
<tr>
<td>HGOTO</td>
<td>Move horizontally to the target angle</td>
<td>LGOTO, RGOTO</td>
</tr>
<tr>
<td>LGOTO</td>
<td>Move left to match the target angle</td>
<td>ACT</td>
</tr>
<tr>
<td>RGOTO</td>
<td>Move right to match the target angle</td>
<td>ACT</td>
</tr>
<tr>
<td>VGOTO</td>
<td>Move vertically to the target elevation</td>
<td>UGOTO, DGOTO</td>
</tr>
<tr>
<td>UGOTO</td>
<td>Move up to match the target elevation</td>
<td>ACT</td>
</tr>
<tr>
<td>DGOTO</td>
<td>Move down to match the target elevation</td>
<td>ACT</td>
</tr>
<tr>
<td>ACT</td>
<td>Move camera 15° up, down, left or right</td>
<td></td>
</tr>
<tr>
<td>RJMP</td>
<td>Move all pointers to the rightmost position</td>
<td>RSHIFT</td>
</tr>
<tr>
<td>MAX</td>
<td>Find maximum element of an array</td>
<td>BUBBLESORT, RJMP</td>
</tr>
</tbody>
</table>
Learning new programs with a fixed NPI core

Toy example: Maximum-finding in an array.

Simple (not optimal) way: Call BUBBLESORT and then take the right-most element of the array. Two new programs:

**RJMP**: Move all pointers to the rightmost position in the array by repeatedly calling RSHIFT program.

**MAX**: Call BUBBLESORT and then RJMP

Expand program memory by adding 2 slots. Randomly initialize, then learn by backpropagation with the NPI core and all other parameters fixed.
Learning new programs with a fixed NPI core

Protocol:
1. Randomly initialize new program vectors in memory
2. Freeze core and other program vectors
3. Backpropagate gradients to new program vectors
Quantitative Results

<table>
<thead>
<tr>
<th>Task</th>
<th>Single</th>
<th>Multi</th>
<th>+ Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>100.0</td>
<td>97.0</td>
<td>97.0</td>
</tr>
<tr>
<td>Sorting</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Canon. seen car</td>
<td>89.5</td>
<td>91.4</td>
<td>91.4</td>
</tr>
<tr>
<td>Canon. unseen</td>
<td>88.7</td>
<td>89.9</td>
<td>89.9</td>
</tr>
<tr>
<td>Maximum</td>
<td>-</td>
<td>-</td>
<td>100.0</td>
</tr>
</tbody>
</table>

- Per-sequence % accuracy.
- + Max indicates performance after addition of MAX program to memory.
- “unseen” uses a test set with disjoint car models from the training set.
Conclusions & Next Steps

• A single NPI can learn multiple programs in dissimilar environments with different affordances.
• NPI sorting and addition programs exhibit strong generalization compared to baseline Seq2Seq models.
• A trained NPI with a fixed core can continue to learn new programs without forgetting already learned programs.

• **Next steps**: reduce supervision, scale up #programs, integrate new perception modules and affordances.
Related work

Too much to cover in 20 minutes!

- Sigma-Pi Units (Rumelhart, 1986): activations of one network become the weights of a second network. Slowly changing network learns to control rapidly-changing network (Schmidhuber, 1992).
- Recent extensions of Seq2Seq: NTM, Pointer Networks, Memory Networks, Stack/Queue/Dequeue-augmented recurrent networks.
- Several other ICLR’16 papers on neural program induction. Main difference is that NPI explicitly incorporates compositional program structure.
- Recent models of prefrontal cognitive control (Donnarumma 2015).
- Learning to Execute (Zaremba 2014)
Thanks!
NPI single time step computation

Traces: \( \xi_t^{inp} : \{ e_t, i_t, a_t \} \quad \xi_t^{out} : \{ i_{t+1}, a_{t+1}, r_t \} \)

LSTM core input \( s_t = f_{enc}(e_t, a_t) \)
LSTM output \( h_t = f_{lstm}(s_t, p_t, h_{t-1}) \)
pred. return prob \( r_t = f_{end}(h_t) \)
next program key \( k_t = f_{prog}(h_t) \)
next program args \( a_{t+1} = f_{arg}(h_t) \)
NPI single time step computation

Traces: \( \xi_t^{inp} : \{e_t, i_t, a_t\} \quad \xi_t^{out} : \{i_{t+1}, a_{t+1}, r_t\} \)

\[
\begin{align*}
  i^* &= \arg \max_{i=1\ldots N} (M_i^{key})^T k_t, \\
  p_{t+1} &= M_{i^*}^{prog}
\end{align*}
\]
NPI single time step computation

Traces:

$\xi_t^{inp} : \{e_t, i_t, a_t\}$

$\xi_t^{out} : \{i_{t+1}, a_{t+1}, r_t\}$

Next environment observation; depends on selected program and arguments.

(Not controlled by NPI parameters)
NPI learning

Traces: \( \xi_t^{inp} : \{e_t, i_t, a_t\} \quad \xi_t^{out} : \{i_{t+1}, a_{t+1}, r_t\} \)

Objective:

\[
\theta^* = \arg \max_{\theta} \sum_{(\xi^{inp}, \xi^{out})} \log P(\xi^{out} | \xi^{inp}; \theta)
\]

\[
\log P(\xi^{out} | \xi^{inp}; \theta) = \sum_{t=1}^{T} \log P(\xi_t^{out} | \xi_1^{inp}, \ldots, \xi_t^{inp}; \theta)
\]