Recursion in Neural Programmer Interpreter

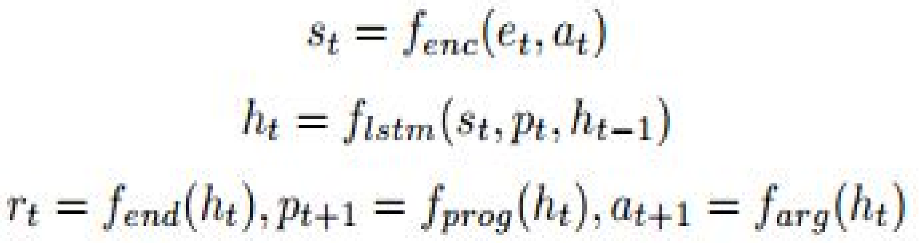
Ankit Kumar - 1301CS10
Arindam Banerjee - 1301CS12

NPA - Neural Programmer Architecture

- Aims to learn programs
- Traditional Seq2Seq models do not generalise well for even slightly larger models
- NPAs aims to generalise better than traditional Seq2Seq
- But even NPAs don’t generalise to very large inputs
- Reason – the network still doesn’t learn the actual program
- Recursion is a way to make simple nets that generalise completely

Neural Programmer Interpreters

Architecture



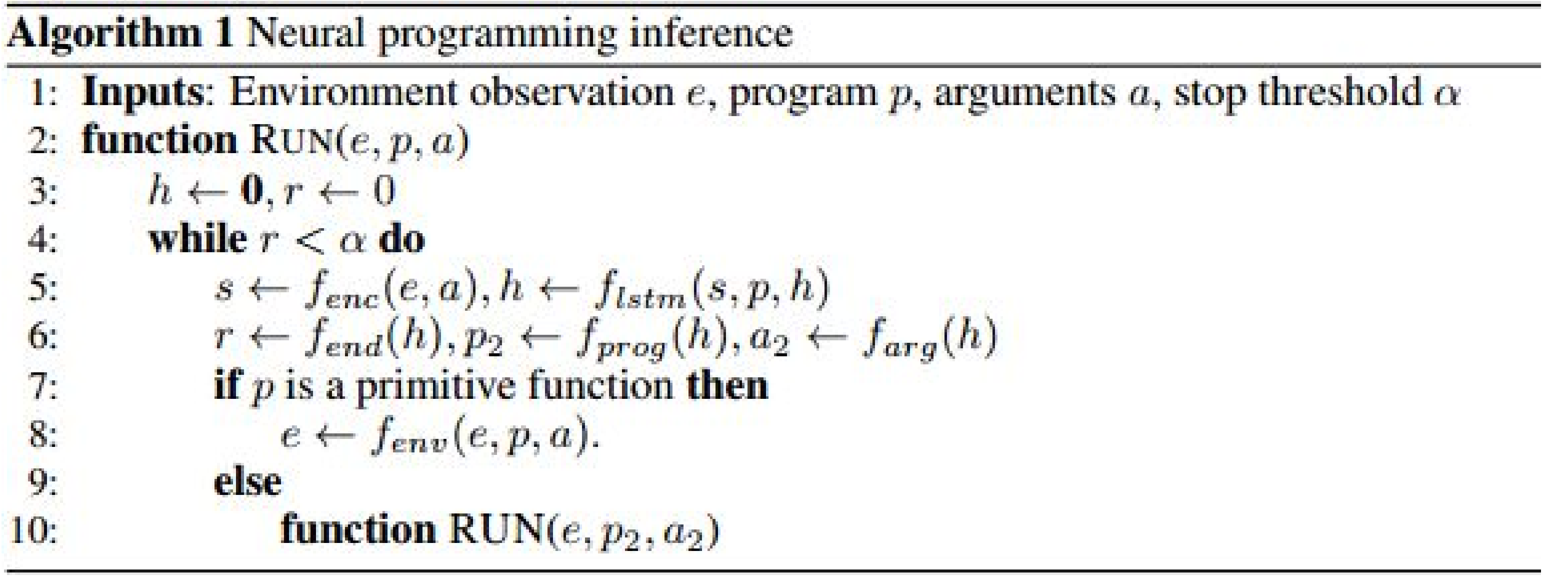
$$s_t = f_{enc}(e_t, a_t)$$

$$h_t = f_{lstm}(s_t, p_t, h_{t-1})$$

$$r_t = f_{end}(h_t), p_{t+1} = f_{prog}(h_t), a_{t+1} = f_{arg}(h_t)$$

1. It generally consists of different LSTMs to do different tasks
2. Central LSTM core has environment variables, a program to run, and it’s arguments as input
3. It predicts the next program & arguments to execute and a probability wether to return or call another function
4. Has a scratchpad to read and write from

Algorithm



Algorithm 1 Neural programming inference

```
Inputs: Environment observation $e$, program $p$, arguments $a$, stop threshold $\alpha$
function RUN($e, p, a$)
    $h \leftarrow \mathbf{0}, r \leftarrow 0$
    while $r < \alpha$ do
        $s \leftarrow f_{enc}(e, a), h \leftarrow f_{lstm}(s, p, h)$
        $r \leftarrow f_{end}(h), p_2 \leftarrow f_{prog}(h), a_2 \leftarrow f_{arg}(h)$
        if $p$ is a primitive function then
            $e \leftarrow f_{env}(e, p, a)$.
        else
            function RUN($e, p_2, a_2$)
```

NPI Recursion - Addition

- The input is a full program trace
- The last step is a tail recursive step
- Meaning the hidden state is cleared
- Means no concept of length of number
- Multiple simple ADD1

TRACE

- ADD
    - ADD1
        - ADDX
        - CARRY
    - LSHIFT
        - PTR1 LEFT
        - PTR2 LEFT
        - PTR0 LEFT
    - ADD (Recursion)

Progress till last presentation

- Went through various implementations of NPA, NPI suits the task best
- Went through the single implementation of NPI from net - too complex
- Understood the various moving parts of the concept
- Made a rough idea what needs to be implemented where
- Need to implement the architecture

Overall Architecture - 1

- 3 LSTMs total for ADD program
  - To generate next program
  - To generate the next program's arguments
  - To decide whether to call another function or to end the current stack
- Tail recursion - helps in case of the recursive call - not in Python
- Environment contains the inputs numbers and the generated output
- Used pointers to access various environment locations
- All the LSTMs trained separately, and used in the NPI core program
- Can add arbitrary length numbers with 100% accuracy

PLSTM - LSTM for next program

- Model trained to get series of program codes for execution
- Different sub-programs are given separate IDs to train
- Architecture composed of LSTM layer and dense layer of 3 neurons
- Generated possible program sequences for training and replicated data to make neural net memorize the system
- Achieved accuracy = 100%

RLSTM - LSTM for terminating program

- r values need to be checked for terminating loop
- Trained model to learn “r” value from sequence of program codes
- Architecture composed of LSTM layer with fully connected Dense Layer
- Tuning of hyperparameters (number of neurons) for optimum accuracy
- Achieved accuracy close to 100%

ALSTM - LSTM for next Argument

- Trained model to learn next argument values from present arguments
- Control shifting of pointers in addition
- Architecture composed of LSTM layer with fully connected Dense Layer
- Tuning of hyperparameters (number of neurons) for optimum accuracy
- Achieved accuracy close to 100%

Overall Architecture - 2

- Global variables are used as implicit environment
- 3 arrays to hold number1, number2 and output of addition
- 3 pointers to access memory locations of the numbers
- carryFlag - global variable to hold the previous carry
- Primitive functions are called by a separate “call” subroutine

Program Trace

- ADD -1
  - ADD1
  - LSHIFT
  - ADD
- ADD1 -2
  - ADDX
  - CARRY
- ADDX -4
  - (primitive)
  - Add numbers
  - Set Output variable
- CARRY -5
  - (primitive)
  - Find carry
  - Set environmental flag
- LSHIFT -3
  - PTR 1
  - PTR 2
  - PTR 3 (output)
- PTR (val) -6
  - (primitive)
  - Move “val” pointer left

Limitations in current implementation

- No common hidden state for ***r***, ***p***, and ***a***
- Base condition of recursion hardcoded - in ideal case, the architecture should automatically handle it
- True “Tail Recursion” not implemented due to python not supporting it.