# Recursion in Neural Programmer Interpreter

Ankit Kumar(1301CS10) & Arindam Banerjee(1301CS12)

## 1 Abstract of the project

Empirically, neural networks that attempt to learn programs from data have exhibited poor generalizability. Moreover, it has traditionally been difficult to reason about the behavior of these models beyond a certain level of input complexity. In order to address these issues, we propose augmenting neural architectures with a key abstraction: recursion. As an application, we implement recursion in the Neural Programmer-Interpreter framework on simple task of grade-school addition.Training neural networks to synthesize robust programs from a small number of examples is a challenging task. We find that recursion makes it easier for the network learn the right program and generalize to unknown situations. Recursion enables provable guarantees on neural programsâ behavior without needing to exhaustively enumerate all possible inputs to the programs.

## 2 Introduction

### 2.1 Literature survey

There have been many previous known startegies to improve generalization through use of curriculum learning, where the model is trained on inputs of gradually increasing complexity. However, models that make use of this strategy eventually fail after a certain level of complexity (e.g. the single-digit multiplication task in Zaremba et al. (2016), the bubble sort task in Reed  de Freitas (2016), and the graph tasks in Graves et al. (2016)). In this version of curriculum learning, even though the inputs are gradually becoming more complex, the semantics of the program is succinct and does not change. Although the model is exposed to more and more data, it might learn spurious and overly complex representations of the program, as suggested in Zaremba et al. (2016). That is to say, the network does not learn the true program semantics. There has been previous work on general Neural Programming Architecture (NPA), similar to Neural Programmer-Interpreter (NPI) in Reed  de Freitas (2016).However the notion of recursion in the neural program architecture is unique and we expect to achieve 100 percent accuracy for our addition task.

### 2.2 Resources

Since our problem was grade school addition, we did not require any special dataset.We have trained three LSTM networks that control the flow of program and arguments for the program. The dataset for these LSTMs have been generated artificially by us to incorporate the algorithm mentioned in Section 3.

## 3 Core Architecture

```python
def RUN(p,a):
    if i[1] < 0: # Base Case Handling
        # handle base case
        return


    print("RUN:" + str(p) + " "+ str(a) + "\n")

    programs = getPrograms(p) # LSTM for programs
    probabilities = getProbabilities(programs) # LSTM for r values
    arguments = getArguments(programs) # LSTM for arguments


    r = 0
    for pid, aid, prid in zip(programs, arguments, probabilities):
        if isPrimitive(pid):
            call(pid, aid)
        else:
            RUN(pid, aid)

        if prid > 0.5:
            break
```

## 4 Work done

We have successfully implemented the recursion in the NPI propsed by Reed  de Freitas (2016).However instead of using a common hidden state, we have used three separate LSTM architectures and trained them separately.

- We have generated dataset artificially for training three LSTM networks namely PLSTM,RLSTM,ALSTM.
- PLSTM Model: This was the model trained to get series of program codes for execution in which different subprograms are given different ids for training purpose.We have generated possible program sequences for training and replicated data to make neural net memorize the system.Our PLSTM architecture consists of LSTM layer followed by a deep dense layer of four neurons.We have rigorously trained the LSTM network with abundant training data and by tuning the hyperparameters to achieve accuracy of 100 percent.
- RLSTM Model: In the algorithm mentioned in Section 3 values of 'r' determine the termination of the main loop.We needed to train model to learn ârâ value from sequence of program codes.The architecture of RLSTM was composed of LSTM layer with fully connected Dense Layer We have rigorously trained the LSTM network with abundant training data and by tuning the hyperparameters to achieve accuracy of 100 percen
- ALSTM ModeL : This was the model trained to learn next argument values from present argument values.It control shifting of pointers in addition.The architecture of RLSTM was composed of LSTM layer with fully connected Dense Layer.We have rigorously trained the LSTM network with abundant training data and by tuning the hyperparameters to achieve accuracy of 100 percent.
- Error Measures Since this is based on NPI, which are based on LSTM and there are very few training samples, the LSTMs memoized the sequences really well, giving us a 100 percent accuracy.
- We have achieved cent percent accuracy for the given task.
- Github Link: `https://github.com/SageEx/NPI-Recursion-Addition`

## 4.1 Future work

Due to constarint of time, we have done some modifications in the architecture.
The base case in NPI recursion is when the current arguments leads to the stack counter return to the previous stack item. The arguments and the programs are used to indicate that the current stack needs to be over. However in our case, the next program is independent of the arguments. Thus, we have coded the base case separately.
Another thing is that the language being python, we have not acchieved true tail recursion. We can implement tail recursion separtely or change the language.